REMOTE INTERFACING WITH 8051

Dhiraj Bennadi
Rajat Chaple

Final Project Report
ECEN 5613 Embedded System Design
December 11, 2021

APPENDICES

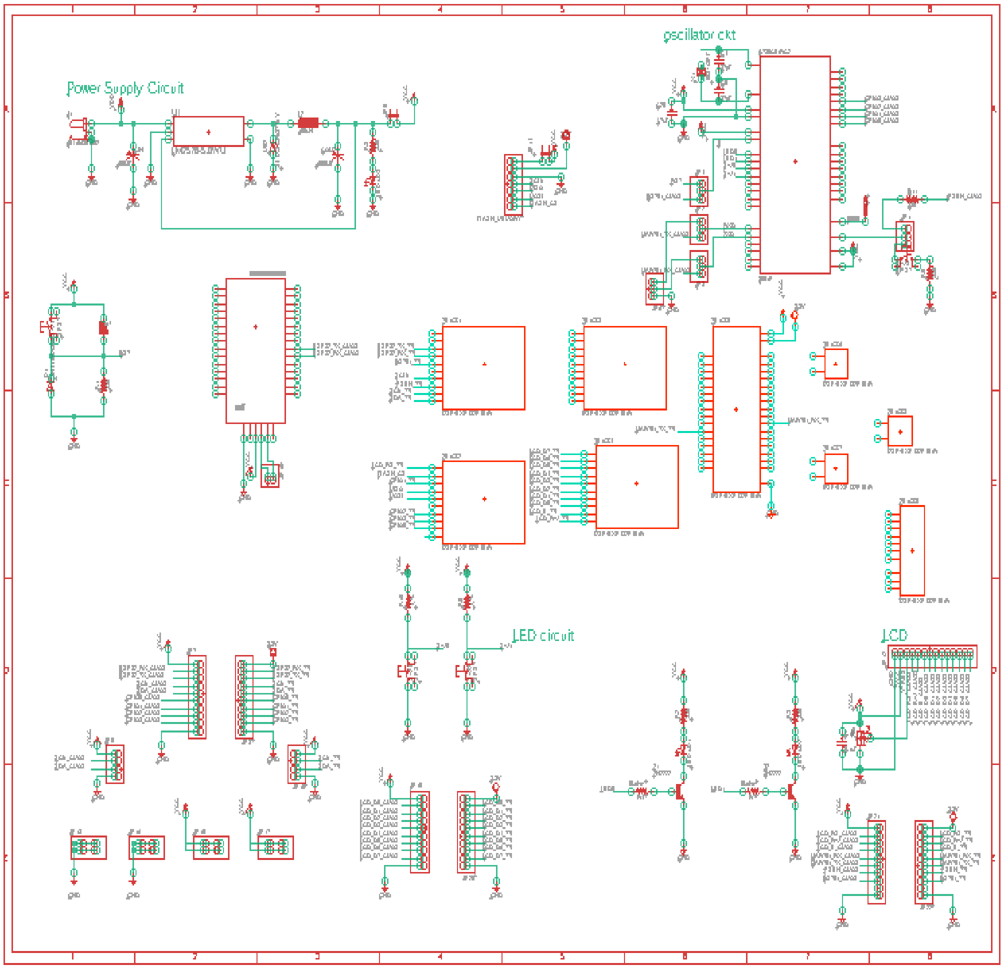
1.1 Appendix - Bill of Materials

BOM_esd_fall_final_project.xls

Major components are listed below (for detailed BOM, kindly refer embedded excel sheet)

Part Description	Source	Cost
AT89C51RD2	Digi-Key www.digikey.com	$20
ESP32 WROOM	Adafruit www.adafruit.com	$11.95
MSP432	TI www.ti.com	$19.90
Level shifters	Sparkfun www.sparkfun.com	$4.50
TOTAL		**$56.35**

1.2 Appendix – Schematics



1.3 Appendix - Firmware Source Code

Main.c

```
/*************************************************************************************
***
 * @file main.c :
 * @brief : application entry point
 *
 * @author :Rajat Chaple

*************************************************************************************
***/

/**************Includes************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "cbfifo.h"
#include "msp.h"
#include "uart.h"
#include "cmd_processor.h"
#include "msp.h"
#include "gpio.h"
#include "spi.h"
#include "esp32.h"
#include "timers.h"
#include "lcd.h"

/************Variables************/
extern cbfifo_t cbfifo_transmit_uart0;
extern cbfifo_t cbfifo_receive_uart0;
extern bool temp_ready_status;
extern bool generate_sine_wave_form;



/**===============================================================================
 * Application entry point
 ===============================================================================*/
void main(void)
    {

    bool is_hex_file_received;
    int8_t c;


    WDT_A->CTL = WDT_A_CTL_PW |             // Stop watchdog timer
            WDT_A_CTL_HOLD;

    //initiating system clock
    CS->KEY = CS_KEY_VAL;                   // Unlock CS module for register access
    CS->CTL0 = 0;                           // Reset tuning parameters
    CS->CTL0 = CS_CTL0_DCORSEL_3;           // Set DCO to 12MHz (nominal, center of
8-16MHz range)
    CS->CTL1 = CS_CTL1_SELA_2 |             // Select ACLK = REFO
            CS_CTL1_SELS_3 |                // SMCLK = DCO
            CS_CTL1_SELM_3;                 // MCLK = DCO
    CS->KEY = 0;                            // Lock CS module from unintended
accesses


    init_gpio();
    uart_init();
    init_lcd();


    // Enable global interrupt
    __enable_irq();


    putstr("\r\n-----------------------------------------------------------------------
-----------\r\n");
    putstr("Remote Interfacing with 8051...\r\n");

    lcd_print_str("8051 : Remote", "Interfacing");
    delay(4000000);

    lcd_print_str("Connecting to", "WIFI");
    esp_32_command(wifi_connect);
    lcd_print_str("WIFI Connected", "Starting server");
    esp_32_command(start_server);

    lcd_print_str("Server started","");
    putstr("Entering main while loop: Ready for receiving HEX file\r\n");
    while (1) {

        c = getchar();  //receive character from uart
        echo_uart2(c);  //send character to uart2 (esp32)
        c = getchar_uart2();
        echo(c);    //display character over uart0 console
        is_hex_file_received = receive_hex_file(c);
        if(is_hex_file_received == true)
        {
            send_hex_file_to_8051();
        }

    }


}
```

Uart.c

```
/*************************************************************************************
***
 * @file uart.c :
 * @brief : This file contains UART initialization and functions for UART handling
 *
 * @author : Dhiraj Bennadi
 * @date : Oct 22 2021
 *

*************************************************************************************
***/

/*---Includes---*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "msp.h"
#include "uart.h"
#include "cbfifo.h"

/*---Variables---*/


extern cbfifo_t cbfifo_transmit_uart0;
extern cbfifo_t cbfifo_receive_uart0;

extern cbfifo_t cbfifo_transmit_uart2;
extern cbfifo_t cbfifo_receive_uart2;

extern cbfifo_t cbfifo_transmit_uart3;
extern cbfifo_t cbfifo_receive_uart3;

/*----------------------------------------------------------------------------
 * Initializing UART0
 * (refer uart.h for additional details)
 * @Resource : This function is written with the help of example code provided
 * in assignment document
 ----------------------------------------------------------------------------*/
void uart_init()
{
    //----------------------------Configure UART pins-----------------------------
    P1->SEL0 |= BIT2 | BIT3;                    // set 2-UART pin as secondary function

    // Configure UART
    EUSCI_A0->CTLW0 |= EUSCI_A_CTLW0_SWRST; // Put eUSCI in reset
    EUSCI_A0->CTLW0 = EUSCI_A_CTLW0_SWRST | // Remain eUSCI in reset
            EUSCI_B_CTLW0_SSEL__SMCLK;      // Configure eUSCI clock source for SMCLK
    // Baud Rate calculation
    // 12000000/(16*9600) = 78.125
    // Fractional portion = 0.125
    // User's Guide Table 21-4: UCBRSx = 0x10
    // UCBRFx = int ( (78.125-78)*16) = 2
    EUSCI_A0->BRW = 78;                     // 12000000/16/9600
    EUSCI_A0->MCTLW = (2 << EUSCI_A_MCTLW_BRF_OFS) |
            EUSCI_A_MCTLW_OS16;

    EUSCI_A0->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Initialize eUSCI
    EUSCI_A0->IFG &= ~EUSCI_A_IFG_RXIFG;    // Clear eUSCI RX interrupt flag
    EUSCI_A0->IE |= EUSCI_A_IE_RXIE;        // Enable USCI_A0 RX interrupt

    // Enable eUSCIA0 interrupt in NVIC module
    NVIC->ISER[0] |= 1 << ((EUSCIA0_IRQn) & 31);



    //-------------------configuring UART2---------------------------
    // Configure UART pins
    P3->SEL0 |= 0x0C;           // Configure P3.2(RXD) and P3.3(TXD) as UART port
    P3->SEL1 &= ~0x0C;

    // Configure UART
    EUSCI_A2->CTLW0 |= EUSCI_A_CTLW0_SWRST; // Put eUSCI in reset
    EUSCI_A2->CTLW0 = EUSCI_A_CTLW0_SWRST | // Remain eUSCI in reset
            EUSCI_B_CTLW0_SSEL__SMCLK;      // Configure eUSCI clock source for SMCLK
    // Baud Rate calculation
    // 12000000/(16*9600) = 78.125
    // Fractional portion = 0.125
    // User's Guide Table 21-4: UCBRSx = 0x10
    // UCBRFx = int ( (78.125-78)*16) = 2
    EUSCI_A2->BRW = 78;//6;//(115200);                     // 12000000/16/9600
    EUSCI_A2->MCTLW = (2 << EUSCI_A_MCTLW_BRF_OFS) |    //8 (115200)
            EUSCI_A_MCTLW_OS16;

    EUSCI_A2->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Initialize eUSCI
    EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;    // Clear eUSCI RX interrupt flag
    EUSCI_A2->IE |= EUSCI_A_IE_RXIE;        // Enable USCI_A0 RX interrupt

    // Enable eUSCIA0 interrupt in NVIC module
    NVIC->ISER[0] |= 1 << ((EUSCIA2_IRQn) & 31);


    //configuring UART3
    // Configure UART pins
    P9->SEL0 |= BIT6 | BIT7;           // Configure P3.2(RXD) and P3.3(TXD) as UART
port
    //P9->SEL1 &= ~0x0C;
    //P9->SEL0 &= ~(BIT6 | BIT7);

    // Configure UART
    EUSCI_A3->CTLW0 |= EUSCI_A_CTLW0_SWRST; // Put eUSCI in reset
    EUSCI_A3->CTLW0 = EUSCI_A_CTLW0_SWRST | // Remain eUSCI in reset
            EUSCI_B_CTLW0_SSEL__SMCLK;      // Configure eUSCI clock source for SMCLK
    // Baud Rate calculation
    // 12000000/(16*9600) = 78.125
    // Fractional portion = 0.125
    // User's Guide Table 21-4: UCBRSx = 0x10
    // UCBRFx = int ( (78.125-78)*16) = 2
    EUSCI_A3->BRW = 78;                     // 12000000/16/9600
    EUSCI_A3->MCTLW = (2 << EUSCI_A_MCTLW_BRF_OFS) |
            EUSCI_A_MCTLW_OS16;

    EUSCI_A3->CTLW0 &= ~EUSCI_A_CTLW0_SWRST; // Initialize eUSCI
    EUSCI_A3->IFG &= ~EUSCI_A_IFG_RXIFG;    // Clear eUSCI RX interrupt flag
    EUSCI_A3->IE |= EUSCI_A_IE_RXIE;        // Enable USCI_A0 RX interrupt

    // Enable eUSCIA0 interrupt in NVIC module
    NVIC->ISER[0] |= 1 << ((EUSCIA3_IRQn) & 31);

}

/*----------------------------------------------------------------------------
 * This function echoes back characters to serial output
 * (refer uart.h for additional details)
 ----------------------------------------------------------------------------*/
void echo(char c)
{
    char str[1] = {c};
    //str[1] = 0;

       switch((int8_t)c)
        {

              //no new character available
        case -1:
               break;

               //user presses backspace
        case '\b':
                 putstr("\b \b");
               break;

               //user presses enter
        case '\r':
               putstr("\r\n");
               break;

        case 0xFF:
            break;

               //echoing received chars
        default:
               putstr(str);
               break;
        }
}

/*----------------------------------------------------------------------------
 * This function echoes back characters to serial output
 * (refer uart.h for additional details)
 ----------------------------------------------------------------------------*/
void echo_uart2(char c)
{
    char str[1] = {c};
```

```c
    //str[1] = 0;

     switch((int8_t)c)
      {

         //no new character available
      case -1:
          break;

          //user presses backspace
      case '\b':
            putstr("\b \b");
          break;

          //user presses enter
      case '\r':
          putstr_uart2("\r\n");
          break;

      case 0xFF:
          break;

          //echoing received chars
      default:
          putstr_uart2(str);
          break;
      }
}

/*-----------------------------------------------------------------------------
 * This function forms a line of command as characters are entered over serial
 * (refer uart.h for additional details)
 ----------------------------------------------------------------------------*/
char* get_line_of_command_raw(int8_t c, uint32_t* num_of_chars_entered_per_command)
{
       static char input_raw[1024] = {0};
       static char* p = input_raw;

       char* ret = NULL;

       switch(c)
               {
                    //no new character available
               case -1:
                      break;

                      //user presses backspace
               case '\b':
                      if(p != input_raw)
                      {
                            *p-- = '\0';
                            (*num_of_chars_entered_per_command)--;
                      }
                      break;
```

```
                        //user presses enter
                case '\r':
                        *p++ = '\0';
                        *num_of_chars_entered_per_command = 0;
                        ret = input_raw;
                        p = input_raw;
                        break;

                        //append characters to command line string
                default:
                        *p++ = c;
                        (*num_of_chars_entered_per_command)++;
                        break;
                }

        return ret;

}

/*----------------------------------------------------------------------------
 * This function forms a line of command as characters are entered over serial
 * (refer uart.h for additional details)
 -----------------------------------------------------------------------------*/
void get_esp32_response(char* readline)
{
    char* p = readline;

    int received_char = -1;

    while(1)
    {
        received_char = getchar_uart2();
        if(received_char != -1)
        {
            //echo(received_char);

            if(/*received_char == '\r' || */received_char == '\n')
                break;
            else
                *p++ = (char)received_char;
        }

    }
}



/*********************************************************************
//Sends string oveer UART
*********************************************************************/
int putstr(char *buf)
{
    uint32_t masking_state;
    int num_of_enqueued_elements = 0;
    int remaining_elements = strlen(buf);
    static char* character_buffer;
    character_buffer = buf;

    //as buf might be larger than available space in cbfifo
    while(remaining_elements)
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        num_of_enqueued_elements = cbfifo_enqueue(&cbfifo_transmit_uart0,
character_buffer, remaining_elements);
        __set_PRIMASK(masking_state);

        //updating remaining data
        remaining_elements = remaining_elements - num_of_enqueued_elements;
        character_buffer += num_of_enqueued_elements;

        if((remaining_elements == 0) ||
                (cbfifo_length(&cbfifo_transmit_uart0) ==
cbfifo_capacity(&cbfifo_transmit_uart0)))
        {
            //enabling transmit interrupt
            EUSCI_A0->IE |= EUSCI_A_IE_TXIE;
        }
    }

    return 0;
}


/**********************************************************************
//Sends string oveer UART
**********************************************************************/
int putstr_uart2(char *buf)
{
    uint32_t masking_state;
    int num_of_enqueued_elements = 0;
    int remaining_elements = strlen(buf);
    static char* character_buffer;
    character_buffer = buf;

    //as buf might be larger than available space in cbfifo
    while(remaining_elements)
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        num_of_enqueued_elements = cbfifo_enqueue(&cbfifo_transmit_uart2,
character_buffer, remaining_elements);
        __set_PRIMASK(masking_state);

        //updating remaining data
        remaining_elements = remaining_elements - num_of_enqueued_elements;
        character_buffer += num_of_enqueued_elements;
        if((remaining_elements == 0) ||
                (cbfifo_length(&cbfifo_transmit_uart2) ==
cbfifo_capacity(&cbfifo_transmit_uart2)))
        {
            //enabling transmit interrupt
            EUSCI_A2->IE |= EUSCI_A_IE_TXIE;
        }
    }

    return 0;
}

/*********************************************************************
//Sends string over UART3
*********************************************************************/
int putstr_uart3(char *buf)
{
    uint32_t masking_state;
    int num_of_enqueued_elements = 0;
    int remaining_elements = strlen(buf);
    static char* character_buffer;
    character_buffer = buf;

    //as buf might be larger than available space in cbfifo
    while(remaining_elements)
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        num_of_enqueued_elements = cbfifo_enqueue(&cbfifo_transmit_uart3,
character_buffer, remaining_elements);
        __set_PRIMASK(masking_state);

        //updating remaining data
        remaining_elements = remaining_elements - num_of_enqueued_elements;
        character_buffer += num_of_enqueued_elements;

        if((remaining_elements == 0) ||
                (cbfifo_length(&cbfifo_transmit_uart3) ==
cbfifo_capacity(&cbfifo_transmit_uart3)))
        {
            //enabling transmit interrupt
            EUSCI_A3->IE |= EUSCI_A_IE_TXIE;
        }
    }

    return 0;
}
/*********************************************************************
// receive a character from user input
*********************************************************************/
int getchar()
{
    int c = 0;
    uint32_t masking_state;
```

```c
    //returns -1 to indicate that no new char has received
    if(cbfifo_length(&cbfifo_receive_uart0) < 1)
    {
        return -1;
    }
    else
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        cbfifo_dequeue(&cbfifo_receive_uart0, &c, sizeof(c));
        __set_PRIMASK(masking_state);
        return c;
    }
}


/********************************************************************
// receive a character from user input
*********************************************************************/
int getchar_uart2()
{
    int c = 0;
    uint32_t masking_state;
    //returns -1 to indicate that no new char has received
    if(cbfifo_length(&cbfifo_receive_uart2) < 1)
    {
        return -1;
    }
    else
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        cbfifo_dequeue(&cbfifo_receive_uart2, &c, sizeof(c));
        __set_PRIMASK(masking_state);
        return c;
    }
}

/********************************************************************
// receive a character from user input
*********************************************************************/
int getchar_uart3()
{
    int c = 0;
    uint32_t masking_state;
    //returns -1 to indicate that no new char has received
    if(cbfifo_length(&cbfifo_receive_uart3) < 1)
    {
        return -1;
    }
    else
    {
        //Entering critical section
        masking_state = __get_PRIMASK();
        __disable_irq();
        cbfifo_dequeue(&cbfifo_receive_uart3, &c, sizeof(c));
        __set_PRIMASK(masking_state);
        return c;
    }
}
/*----------------------------------------------------------------------------
 * This function echoes back characters to serial output
 * (refer uart.h for additional details)
 -----------------------------------------------------------------------------*/
void echo_uart3(char c)
{
    char str[1] = {c};
    //str[1] = 0;

     switch((int8_t)c)
      {

         //no new character available
      case -1:
          break;

          //user presses backspace
      case '\b':
            putstr("\b \b");
          break;

          //user presses enter
      case '\r':
          putstr_uart3("\r\n");
          break;

      case 0xFF:
          break;

          //echoing received chars
      default:
          putstr_uart3(str);
          break;
      }
}
/*********************************************************************
// receive a number from user input
// Returns length of printed string with null char
**********************************************************************/
int accept_number_from_user()
{
    char received_char = '`';
    char number_in_char[25] = "\0";
    bool valid_number_status = false;
    int number = 0;
    int i = 0;
    while (valid_number_status == false)
    {
        received_char = (char)getchar();

        if(received_char == 0xFF)
                continue;

        echo(received_char);

        i++;
        // if(received_char is in range of allowable values) //sanity check for
alphabetical chars
        if (!(received_char >= '0' && received_char <= '9') && (received_char !=
'\r'))
        {
            valid_number_status = false;
            memset(number_in_char, 0, strlen(number_in_char));
            //putstr("\r\n\t\t Error: Re-enter the number: ");
            i = 0;

        }
        else if (received_char == '\r')
        {
            strcat(number_in_char, '\0');
            number = atoi(number_in_char);
            valid_number_status = true;
            //return number;
        }
        strncat(number_in_char, &received_char, 1);

        if (i >= 24)
        {
            putstr("\r\n\t\t Error: Reading a number failed, Reached maximum length
supported");
            putstr("\r\n\t\t Error: Re-enter the number: ");
            valid_number_status = false;
            memset(number_in_char, 0, sizeof(number_in_char));
            i = 0;
        }
    }
    return number;
}
```

```
/************************************************************************************
*****
 * @file uart.h :
 * @brief : This file contains defines, includes, and function prototypes for uart.c
 *
 * @author : Dhiraj Bennadi
 * @date : Nov 6 2021
 *

*************************************************************************************
*****/

#ifndef UART_H_
#define UART_H_

#include <stdint.h>

#define putchar(c) putstr(&c)

#define send_to_esp32(s) putstr_uart2(s)
#define receive_from_esp32() getchar_uart2()

#define send_to_8051(s) putstr_uart3(s)
#define receive_from_8051() getchar_uart3()
/** -----------------------------------------------------------------------------
 *  @brief      Initializes UART  to work at specified
 *                               BAUD_RATE
 *                               DATA_SIZE
 *                               PARITY
 *                               STOP_BITS
 *                         in uart.c
 *
 * @param buf :  none
 *
 * @return none
 --------------------------------------------------------------------------------**/
void uart_init(void);
/** -----------------------------------------------------------------------------
 *  @brief       echoes back characters to serial out
 *
 * @param    :      c -> character c to be echoed
 * @param    :      num_of_chars -> number of characters to keep track of characters
entered per
 *                         command line
 * @return none
 --------------------------------------------------------------------------------**/
void echo(char c);
void echo_uart2(char c);

/** -----------------------------------------------------------------------------
 *  @brief       receives raw line of command over uart until enter key is pressed
 *
  * @param    :      num_of_chars -> number of characters to keep track of characters
entered per
 *                         command line
 * @return command_line when enter key is pressed
 ---------------------------------------------------------------------------------**/
char* get_line_of_command_raw(int8_t c, uint32_t*);
void get_esp32_response(char *);
/********************************************************************
// Send string over UART
*********************************************************************/
int putstr(char *buf);
int putstr_uart2(char *buf);
int putstr_uart3(char *buf);

/********************************************************************
// receive character over UART
*********************************************************************/
int getchar();
int getchar_uart2();
int getchar_uart3();

/********************************************************************
// convert number to string
*********************************************************************/
char* num_to_str(float num);

/********************************************************************
// accepts a number from the user
*********************************************************************/
int accept_number_from_user(void);

#endif /* UART_H_ */
```

Esp32.c

```
/************************************************************************************
***
 * @file esp32.c :
 * @brief : This file contains wifi configuration
 *
 * @date : Nov 29 2021
 * @author: Rajat Chaple

*************************************************************************************
***/

/*---Includes---*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "msp.h"
#include "uart.h"
#include "cbfifo.h"
#include "gpio.h"
#include "esp32.h"
#include "timers.h"
#include "lcd.h"

//states for wifi connections
typedef enum wifi_state_e{
STATE_AT_CHECK,
STATE_WIFI_MODE_3,
STATE_SET_SSID_PASSWORD,
STATE_WIFI_MODE_1,
STATE_CONNECT_TO_WIFI,
NO_COMMAND = 'z'
}wifi_state_t;

//states for starting wifi
typedef enum tcp_server_state_e{
STATE_MULTI_CONNECTIONS,
STATE_SET_STATION_IP,
STATE_DELETE_EXISTING_SERVER,
STATE_START_SERVER,
STATE_GET_SERVER_IP
}tcp_server_state_t;


//Connecting to Wi-Fi network command set
esp32_commands_t wifi_connect[] =
{
 {STATE_AT_CHECK,            "AT",                                          "SUCCESS:
AT OK successful",    "ERROR: AT OK Failed"},
 {STATE_SET_STATION_IP,      "AT+CIPSTA=\"192.168.243.100\"",               "SUCCESS:
station IP set",      "ERROR: station ip not set"},
 {STATE_WIFI_MODE_3,         "AT+CWMODE=3",                                 "SUCCESS:
WIFI MODE SET to 3",  "ERROR: WIFI MODE NOT SET"},
 {STATE_SET_SSID_PASSWORD,    "AT+CWSAP=\"dhirajm512\",\"1234512345\",1,2",  "SUCCESS:
ssid password set",   "ERROR: ssid password not set"},
 {STATE_WIFI_MODE_1,          "AT+CWMODE=1",                                 "SUCCESS:
WIFI MODE SET to 1",  "ERROR: WIFI MODE NOT SET"},
 {STATE_CONNECT_TO_WIFI,      "AT+CWJAP=\"dhirajm512\",\"1234512345\"" ,     "SUCCESS:
WIFI connected",      "ERROR: wifi connection failed"},
 {NULL,                       NULL,                                           NULL,
NULL}
};




//Starting a server command set
esp32_commands_t start_server[] =
{
 {STATE_MULTI_CONNECTIONS,       "AT+CIPMUX=1",                    "SUCCESS:
multiconnection set success",   "ERROR: multiconnection set failed"},
 {STATE_DELETE_EXISTING_SERVER,  "AT+CIPSERVER=0",                 "SUCCESS:
existing server deleted",       "ERROR: existing server not deleted"},
 {STATE_START_SERVER,            "AT+CIPSERVER=1,80",              "SUCCESS:
started server",                "ERROR: server start failed"},
 {STATE_GET_SERVER_IP,           "AT+CIFSR",                       "SUCCESS:
retrieved server IP",           "ERROR: server IP not retrieved"},
 {NULL,                           NULL,                             NULL,
NULL}
};


/*---Variables---*/

char endline = 0x0D;
#define SEND_ENDLINE send_to_esp32(&endline)

bool is_wifi_connected = false;
bool is_server_started = false;

/** ------------------------------------------------------------------------------
 *  @brief     This function executes commands from esp32_commands

 * @param buf :  none
 *
 * @return none
 --------------------------------------------------------------------------------**/
void esp_32_command(esp32_commands_t* command_set)
{
    char readline[100] = "";
    char c = 0x0D;  //<CR>
    char a = 0x0A;  //<LF>
    command_set->current_state = 0;

    while(1)
    {
        send_to_esp32(command_set[command_set->current_state].command); //sending
command from a command table
```

```
        send_to_esp32(&c);
        send_to_esp32(&a);

        while(1)
        {
            memset(readline,0,sizeof(readline));

            get_esp32_response(readline);
            delay(50000);
            putstr(readline);
            putstr("\r\n");

            if(strstr(readline,"OK") != NULL)
                {   putstr(command_set[command_set-
>current_state].ok_response_msg_str);putstr("\r\n");
                    command_set->current_state++;
                    break;
                }
            else if(strstr(readline,"ERROR") != NULL)
                {
                    putstr(command_set[command_set-
>current_state].error_response_msg_str);putstr("\r\n");
                    break;
                }
            else if(strstr(readline,"CONNECT") != NULL) //displays over LCD that
client is connected
            {
                lcd_print_str("Client", "Connected");
            }
        }

        if(command_set[command_set->current_state].command == NULL)
           break;

        delay(50000);
    }
}
```

Esp32.h

```
/************************************************************************************
*****
 * @file esp32.h :
 * @brief : This file contains defines, includes, and function prototypes for esp32.c
 *
 * @date : Oct 22 2021
 * @author: Rajat Chaple

*************************************************************************************
*****/

#ifndef ESP32_H_
#define ESP32_H_

#include <stdint.h>

typedef struct esp32_commands_s{
    int current_state;
    char* command;
    char* ok_response_msg_str;
    char* error_response_msg_str;
}esp32_commands_t;


extern esp32_commands_t wifi_connect[];
extern esp32_commands_t start_server[];

/** ------------------------------------------------------------------------------
 *  @brief    This function executes commands from esp32_commands

 * @param buf :  none
 *
 * @return none
 ---------------------------------------------------------------------------------**/
void esp_32_command(esp32_commands_t*);


#endif /* ESP32_H_ */
```

Cmd_processor.c

```
/************************************************************************************
***
 * @file cmd_processor.c :
 * @brief : This file contains functions to perform lexical analysis to extract
commands
 *                   over serial input and run respective handlers
 *
 * @author: Rajat Chaple

*************************************************************************************
***/

/*----------------Includes---------------*/
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>
#include "uart.h"
#include "cmd_processor.h"
#include "gpio.h"
#include "msp.h"
#include "spi.h"
#include "stdint.h"
#include "timers.h"
#include <string.h>
#include "lcd.h"


/*----------------Defines and variables----------*/
typedef enum command_hex_file_dump_e{
STATE_WAITING_FOR_START_FRAME,
STATE_HEX_FILE_DATA,
STATE_END_OF_FRAME
}command_hex_file_dump_t;

typedef enum programming_state_e{
    ENTER_BOOTLOADER,
    TRANSMIT_HEX_FILE,
    RESET_8051,
}programming_state_t;

command_hex_file_dump_t hex_file_dump_to_msp32_state = STATE_WAITING_FOR_START_FRAME;

char hex_file[50000];   //stores 50KB of hex file
static uint16_t hex_file_dump_index;


/*--------------------------------------------------------------------------------
* Receive hex file over UART2 usign ESP32
* SOF for each frame: $$$
* EOF: ###
--------------------------------------------------------------------------------*/
bool receive_hex_file(int c)
{
```

```
static uint8_t SOF_count = 0;   //$$$
static uint8_t EOF_count = 0;   //###
bool hex_file_received_status = false;

if(c == -1)
    return false;

switch(hex_file_dump_to_msp32_state)
{
case STATE_WAITING_FOR_START_FRAME:

    if(c == '$')
    {
        SOF_count++;
        if(SOF_count == 3)  //$$$
        {
            hex_file_dump_to_msp32_state = STATE_HEX_FILE_DATA;
            SOF_count = 0;
            lcd_print_str("Receiving", "hex file");
        }
    }
    else
    {
        SOF_count = 0;
    }
    break;

case STATE_HEX_FILE_DATA:
    if(c == '#')
    {
        EOF_count++;
        hex_file[hex_file_dump_index++] = '\0';
        hex_file_dump_to_msp32_state = STATE_END_OF_FRAME;

        break;
    }
    else if(c == '$')
    {
        SOF_count++;
        hex_file_dump_to_msp32_state = STATE_WAITING_FOR_START_FRAME;
        break;
    }
    else if(c == '+')
    {
        hex_file_dump_to_msp32_state = STATE_WAITING_FOR_START_FRAME;
        break;
    }
    else if(c == '\r' || c== '\n')
    {
        break;
    }
    hex_file[hex_file_dump_index++] = c;
    break;

case STATE_END_OF_FRAME:
        if(c == '#')
        {
            EOF_count++;
            if(EOF_count == 2)
            {
                EOF_count = 0;
                hex_file_dump_index = 0;
                hex_file_received_status = true;
                putstr("\r\n\r\nReceived hex file: \r\n");
                lcd_print_str("Hex file", "Received");
                putstr(hex_file);
            }
        }
        else
        {
            EOF_count = 0;
        }
        break;
    }

    return hex_file_received_status;
}


/*--------------------------------------------------------------------------------
* Sending hex file to 8051
--------------------------------------------------------------------------------*/
void send_hex_file_to_8051()
{
    bool is_8051_programmed = false;
    programming_state_t programming_state = ENTER_BOOTLOADER;
    char* token;
    while(1)
    {
        switch(programming_state)
        {
        case ENTER_BOOTLOADER:
            putstr("\r\nEntering Bootloader...\r\n");
            lcd_print_str("8051: Entering", "Bootloader");
            enter_8051_into_bootloader();
            delay(100000);
            programming_state = TRANSMIT_HEX_FILE;
            break;

        case TRANSMIT_HEX_FILE:
            lcd_print_str("transferring", "HEX file");
            putstr("\r\nSending Hex file...\r\n");
            send_to_8051("U\0");
            delay(100000);
            /* get the first token */
            token = strtok(hex_file, ":");

            /* walk through other tokens */
            while( token != NULL ) {
               send_to_8051(":\0");
               send_to_8051(token);
               delay(200000);
               token = strtok(NULL, ":");
            }

             programming_state = RESET_8051;
            break;

        case RESET_8051:
            lcd_print_str("8051: Exiting", "Bootloader");
            putstr("\r\nResetting 8051...\r\n");
            reset_8051();
            is_8051_programmed = true;
            break;

        default:
            break;
        }

        if(is_8051_programmed == true)
        {
            putstr("\r\nDone programming\r\n");
            break;
        }
    }

    putstr("\r\n8051 programmed successfully\r\n");
    lcd_print_str("8051: Firmware", "flashed");

}
```

cmd_processor.h

```
/************************************************************************************
*******
 * @file cmd_processor.h :
 * @brief : This file contains includes, defines and function prototypes for
cmd_processor.c
 *
 * @date : Nov 29, 2021
 * @author: Rajat Chaple

*************************************************************************************
*******/

#ifndef CMD_PROCESSOR_H_
#define CMD_PROCESSOR_H_

/** ------------------------------------------------------------------------------
 * @brief This function receives hex file over uart from esp32
 *
 * @param a character over uart
 *
 * @return None
 ---------------------------------------------------------------------------------**/
bool receive_hex_file(int c);

/** ------------------------------------------------------------------------------
 * @brief This function sends hex file to 8051 ocer UART3
 *
 * @return None
 ---------------------------------------------------------------------------------**/
void send_hex_file_to_8051(void);

#endif /* CMD_PROCESSOR_H_ */
```

```c
/****************************************************************************
 * gpio.c
 * This file initializes gpios
 *
 *  Created on: Nov 3, 2021
 *  @author: Dhiraj Bennadi
 ****************************************************************************/

#include "gpio.h"
#include "msp.h"
#include "uart.h"
#include "timers.h"

/*----------------------------------------------------------------------------
 * Initializes pins to Inputs and outputs as defined in gpio_init.h
 * (refer gpio.h for more details)
 ----------------------------------------------------------------------------*/
void init_gpio()
{
       // Set P2.0 to output direction
        RED_LED_PORT->DIR |= RED_LED_PIN;
        // Set P2.1 to output direction
        GREEN_LED_PORT->DIR |= GREEN_LED_PIN;
        // Set P2.2 to output direction
        BLUE_LED_PORT->DIR |= BLUE_LED_PIN;

        // Set P2.4 to output direction
        SPI_CS_PORT->DIR |= SPI_CS_PIN;
        SPI_CS_PORT->OUT |= SPI_CS_PIN;

        /*Set Output of Port 6 Pin 5*/
        AT8051_CTRL_PORT->OUT |= AT8051_PSEN_PIN;
        /*Set Output of Port 6 Pin 4*/
        AT8051_CTRL_PORT->OUT &= ~AT8051_RESET_PIN;

        /*Set Data Direction of Port 6 Pin 4*/
        AT8051_CTRL_PORT->DIR |= AT8051_RESET_PIN;

        /*Set Data Direction of Port 6 Pin 5*/
        AT8051_CTRL_PORT->DIR |= AT8051_PSEN_PIN;


#ifdef TEST_ON_OSCILLOSCOPE
        P1->DIR |= BIT5;
#endif


}

/*----------------------------------------------------------------------------
 * Sets Led color to passed parameter
 * (refer gpio.h for more details)
 ----------------------------------------------------------------------------*/
void set_led(led_color_t color)
{
    if(color & 0b001)   //Checking if Blue LED needs to be turned or
    {
        BLUE_LED_PORT->OUT |= BLUE_LED_PIN;   //Turns Blue LED ON
    }
    else
    {
        BLUE_LED_PORT->OUT &= ~BLUE_LED_PIN;  //Turns Blue LED off
    }

    if(color & 0b010)   //Checking if Green LED needs to be turned or
    {
        GREEN_LED_PORT->OUT |= GREEN_LED_PIN; //Turns Green LED on
    }
    else
    {
        GREEN_LED_PORT->OUT &= ~GREEN_LED_PIN; //Turns Green LED off
    }

    if(color & 0b100)   //Checking if Red LED needs to be turned or
    {
        RED_LED_PORT->OUT |= RED_LED_PIN;   //Turns Red LED on
    }
    else
    {
        RED_LED_PORT->OUT &= ~RED_LED_PIN;   //Turns Red LED off
    }

}


/*----------------------------------------------------------------------------
 * Enter 8051 into bootloader mode
 * Sequence: RESET HIGH -> PSEN LOW -> RESET LOW -> PSEN HIGH
 * (refer gpio.h for more details)
 ----------------------------------------------------------------------------*/
void enter_8051_into_bootloader()
{
    /*Reset*/
    delay(150000);
    AT8051_CTRL_PORT->OUT |= AT8051_RESET_PIN;
    delay(150000);
    /*PSEN*/
    AT8051_CTRL_PORT->OUT &= ~AT8051_PSEN_PIN;
    delay(150000);

    AT8051_CTRL_PORT->OUT &= ~AT8051_RESET_PIN;
    delay(150000);

    AT8051_CTRL_PORT->OUT |= AT8051_PSEN_PIN;
    delay(150000);


}

/*----------------------------------------------------------------------------
```

```
 * Enter 8051 into application mode
 * (refer gpio.h for more details)
 ----------------------------------------------------------------------------*/
void reset_8051()
{
    delay(100000);
    /*Reset*/
    AT8051_CTRL_PORT->OUT |= AT8051_RESET_PIN;
    delay(100000);

    AT8051_CTRL_PORT->OUT &= ~AT8051_RESET_PIN;
    delay(100000);

}
```

Gpio.h

```c
/***********************************************************************************
 * gpio.h
 * This file contains declarations, typedefs and function prototypes for gpio.c
 *
 *  Created on: Nov 1, 2021
 *  @author: Dhiraj Bennadi
 ***********************************************************************************/

#ifndef GPIO_H_
#define GPIO_H_
/*--------------- Defines---------------*/
#define RED_LED_PORT    P2
#define RED_LED_PIN     BIT0

#define GREEN_LED_PORT    P2
#define GREEN_LED_PIN     BIT1

#define BLUE_LED_PORT    P2
#define BLUE_LED_PIN     BIT2

#define SWITCH1_PORT    P1  //right switch
#define SWITCH1_PIN     BIT1

#define SWITCH2_PORT    P1
#define SWITCH2_PIN     BIT4

#define SPI_CS_PORT     P6
#define SPI_CS_PIN      BIT7

#define AT8051_CTRL_PORT (P6)
#define AT8051_RESET_PIN  (BIT4)
#define AT8051_PSEN_PIN  (BIT5)

//#define TEST_ON_OSCILLOSCOPE

/*--------------- Datatypes-------------*/
typedef enum led_color{
    OFF,    //000
    BLUE,   //001
    GREEN,  //010
    CYAN,   //011 -> GREEN+BLUE
    RED,    //100
    MAGENTA,//101 -> RED+BLUE
    YELLOW, //110
    WHITE   //111 -> WHITE
}led_color_t;
/*----------function prototypes--------*/
void init_gpio(void);
void set_led(led_color_t color);
void Port_Mapping(void);
void enter_8051_into_bootloader(void);
void reset_8051();

#endif /* GPIO_H_ */
```

```c
/************************************************************************************
*****
 * @file irq.c :
 * @brief : This file contains interrupt handlers for peripherals
 *
 * @author : Rajat Chaple (rajat.chaple@colorado.edu)
 * @date : Nov 15, 2021
 *
*************************************************************************************
*****/

#include "msp.h"
#include "cbfifo.h"
#include "gpio.h"
#include "stdbool.h"
#include "uart.h"

extern cbfifo_t cbfifo_transmit_uart0;
extern cbfifo_t cbfifo_receive_uart0;

extern cbfifo_t cbfifo_transmit_uart2;
extern cbfifo_t cbfifo_receive_uart2;

extern cbfifo_t cbfifo_transmit_uart3;
extern cbfifo_t cbfifo_receive_uart3;

extern uint16_t spi_tx_data;
extern uint16_t spi_rx_data;

extern uint8_t i2c_tx_data;
extern uint8_t i2c_rx_data;


/*--------------- Datatypes-------------*/
typedef enum led_status_e{
    GREEN_LED,
    BLUE_LED,  //010
    NO_CHANGE
}led_status_t;



led_status_t led_status = BLUE_LED;//as init is always green
led_status_t prev_led_status = GREEN_LED;

bool toggle_state_by_switch = true;
bool debounce_period_elapsed = false;

volatile long temp;
volatile float IntDegF;

bool temp_ready_status = false;


/** -------------------------------------------------------------------------------
```

```
 * @brief UART0 IRQ handler for tranmit and receive
 * @param none
 * @return none
 --------------------------------------------------------------------------------**/
void EUSCIA0_IRQHandler(void)
{

    int RXData;
    int TXData;
    uint32_t masking_state;

    //data is received over UART
        if (EUSCI_A0->IFG & EUSCI_A_IFG_RXIFG)
        {
            EUSCI_A0->IFG &= ~EUSCI_A_IFG_RXIFG;// Clear interrupt
            RXData = EUSCI_A0->RXBUF;           // Clear buffer
            if(cbfifo_length(&cbfifo_receive_uart0) !=
cbfifo_capacity(&cbfifo_receive_uart0)) //if fifo not full
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                cbfifo_enqueue(&cbfifo_receive_uart0, &RXData, sizeof(RXData));
//adding element to the queue
                __set_PRIMASK(masking_state);
            }
            else
            {
                //character discarded when queue is full
            }
        }



        //Transmit interrupt received
        if((EUSCI_A0->IFG & EUSCI_A_IFG_TXIFG) &
                ( EUSCI_A0->IE & EUSCI_A_IE_TXIE))
        {
            if(cbfifo_length(&cbfifo_transmit_uart0) != 0)
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                 if(cbfifo_dequeue(&cbfifo_transmit_uart0, &TXData, 1) == 1)
//dequeue elemnt to be sent
                     EUSCI_A0->TXBUF = TXData;
                __set_PRIMASK(masking_state);
            }
            else
            {
                EUSCI_A0->IE &= ~EUSCI_A_IE_TXIE;    //disabling transmit interrupt
            }
        }
}
```

```
/** ------------------------------------------------------------------------------
 * @brief UART2 IRQ handler for tranmit and receive
 * @param none
 * @return none
 ---------------------------------------------------------------------------------**/
void EUSCIA2_IRQHandler(void)
{

    int RXData;
    int TXData;
    uint32_t masking_state;

    //data is received over UART
        if (EUSCI_A2->IFG & EUSCI_A_IFG_RXIFG)
        {
            EUSCI_A2->IFG &= ~EUSCI_A_IFG_RXIFG;// Clear interrupt
            RXData = EUSCI_A2->RXBUF;           // Clear buffer
            //echo(RXData);
            if(cbfifo_length(&cbfifo_receive_uart2) !=
cbfifo_capacity(&cbfifo_receive_uart2)) //if fifo not full
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                cbfifo_enqueue(&cbfifo_receive_uart2, &RXData, sizeof(RXData));
//adding element to the queue
                __set_PRIMASK(masking_state);
            }
            else
            {
                //character discarded when queue is full
            }
        }



        //Transmit interrupt received
        if((EUSCI_A2->IFG & EUSCI_A_IFG_TXIFG) &
                ( EUSCI_A2->IE & EUSCI_A_IE_TXIE))
        {
            if(cbfifo_length(&cbfifo_transmit_uart2) != 0)
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                 if(cbfifo_dequeue(&cbfifo_transmit_uart2, &TXData, 1) == 1)
//dequeue element to be sent
                     EUSCI_A2->TXBUF = TXData;
                __set_PRIMASK(masking_state);
            }
            else
            {
                EUSCI_A2->IE &= ~EUSCI_A_IE_TXIE;    //disabling transmit interrupt
            }
        }
}

/** ------------------------------------------------------------------------------
 * @brief UART3 IRQ handler for tranmit and receive
 * @param none
 * @return none
 ---------------------------------------------------------------------------------**/
void EUSCIA3_IRQHandler(void)
{

    int RXData;
    int TXData;
    uint32_t masking_state;

    //data is received over UART
        if (EUSCI_A3->IFG & EUSCI_A_IFG_RXIFG)
        {
            EUSCI_A3->IFG &= ~EUSCI_A_IFG_RXIFG;// Clear interrupt
            RXData = EUSCI_A3->RXBUF;           // Clear buffer
            //echo(RXData);
            if(cbfifo_length(&cbfifo_receive_uart3) !=
cbfifo_capacity(&cbfifo_receive_uart3)) //if fifo not full
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                cbfifo_enqueue(&cbfifo_receive_uart3, &RXData, sizeof(RXData));
//adding element to the queue
                __set_PRIMASK(masking_state);
            }
            else
            {
                //character discarded when queue is full
            }
        }



        //Transmit interrupt received
        if((EUSCI_A3->IFG & EUSCI_A_IFG_TXIFG) &
                ( EUSCI_A3->IE & EUSCI_A_IE_TXIE))
        {
            if(cbfifo_length(&cbfifo_transmit_uart3) != 0)
            {
                //entering critical section
                masking_state = __get_PRIMASK();
                __disable_irq();
                 if(cbfifo_dequeue(&cbfifo_transmit_uart3, &TXData, 1) == 1)
//dequeue elemnt to be sent
                     EUSCI_A3->TXBUF = TXData;
                __set_PRIMASK(masking_state);
            }
            else
            {
                EUSCI_A3->IE &= ~EUSCI_A_IE_TXIE;    //disabling transmit interrupt
            }
        }
}



/*---------------------------------------------------------------------------
@brief: SPI IRQ handler
@resource: example mentioned in assignment document
 ---------------------------------------------------------------------------*/
void EUSCIB3_IRQHandler(void)
{
    if (EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG)
    {
        EUSCI_B3->TXBUF = spi_tx_data;           // Transmit characters
        EUSCI_B3->IE &= ~EUSCI_B__TXIE;     //disabling tx interrupt

        // Wait till a character is received
        //while (!(EUSCI_B0->IFG & EUSCI_B_IFG_RXIFG));

        // USCI_B0 TX buffer ready?
        while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
    }

    if(EUSCI_B3->IFG & EUSCI_B_IFG_RXIFG)
    {
        spi_rx_data = EUSCI_B3->RXBUF;

        // Clear the receive interrupt flag
        EUSCI_B3->IFG &= ~EUSCI_B_IFG_RXIFG;
    }
}
```

Lcd.c

```
/************************************************************************************
*****
 * @file lcd.c :
 * @brief : This file contains lcd routines
 *
 * @author : Dhiraj Bennadi
 * @date : Dec 6, 2021
 * @source: msp432 example code
*************************************************************************************
*****/


#include "lcd.h"
#include "timers.h"

static int cursorPointer = 0x80;

void init_lcd(void) {
    P4->DIR = 0xFF;     /* make P4 pins output for data and controls */
    delayMs(30);                /* initialization sequence */
    LCD_nibble_write(0x30, 0);
    delayMs(10);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x30, 0);
    delayMs(1);
    LCD_nibble_write(0x20, 0);  /* use 4-bit data mode */
    delayMs(1);

    LCD_command(0x28);      /* set 4-bit data, 2-line, 5x7 font */
    LCD_command(0x06);      /* move cursor right after each char */
    LCD_command(0x01);      /* clear screen, move cursor to home */
    LCD_command(0x0F);      /* turn on display, cursor blinking */

    LCD_command(1);
    LCD_command(0x80);
}

/* With 4-bit mode, each command or data is sent twice with upper
 * nibble first then lower nibble.
 */
void LCD_nibble_write(unsigned char data, unsigned char control) {
    data &= 0xF0;           /* clear lower nibble for control */
    control &= 0x0F;        /* clear upper nibble for data */
    P4->OUT = data | control;       /* RS = 0, R/W = 0 */
    P4->OUT = data | control | EN;  /* pulse E */
    delayMs(0);
    P4->OUT = data;                 /* clear E */
    P4->OUT = 0;
}

void LCD_command(unsigned char command) {
    LCD_nibble_write(command & 0xF0, 0);    /* upper nibble first */
    LCD_nibble_write(command << 4, 0);      /* then lower nibble */
    if (command < 4)
        delayMs(4);         /* commands 1 and 2 need up to 1.64ms */
    else
        delayMs(1);         /* all others 40 us */
}

void LCD_data(unsigned char data) {

    LCD_nibble_write(data & 0xF0, RS);    /* upper nibble first */
    LCD_nibble_write(data << 4, RS);      /* then lower nibble  */
    updateCursorPointer();

    delayMs(1);
}

/* delay milliseconds when system clock is at 3 MHz */
void delayMs(int n) {
    int i, j;

    for (j = 0; j < n; j++)
        for (i = 750; i > 0; i--);      /* Delay */
}


void lcd_print_str(char *str1, char *str2)
{
    LCD_command(1);
    LCD_command(0x80);
    cursorPointer = 0x80;
    while(*str1 != '\0')
    {
        LCD_data(*str1++);
    }

    LCD_command(0xC0);
    cursorPointer = 0xC0;
    while(*str2 != '\0')
    {
        LCD_data(*str2++);
    }

    delay(100000);
}

void updateCursorPointer(void)
{
    int update = 0;

    if(cursorPointer == 0x8F)
    {
        cursorPointer = 0xC0;
        LCD_command(cursorPointer);
        update = 1;
    }
```

```c
    if(cursorPointer == 0xCF)
    {
        cursorPointer = 0x80;
        LCD_command(cursorPointer);
        //LCD_command(1);
        update = 1;
    }
    if(update == 0)
    {
        cursorPointer++;
    }


}

/*************************************************************************************
*****
 * @file lcd.h :
 * @brief : This file contains header files for
 *
 * @author : Dhiraj Bennadi
 * @date : Dec 6, 2021
 * @source: msp432 example code
**************************************************************************************
*****/

#ifndef LCD_H_
#define LCD_H_


#include "msp432.h"

#define RS 1     /* P4.0 mask */
#define RW 2     /* P4.1 mask */
#define EN 4     /* P4.2 mask */

void delayMs(int n);
void LCD_nibble_write(unsigned char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(unsigned char data);
void init_lcd(void);
void lcd_print_str(char *str1, char *str2);
void updateCursorPointer(void);


#endif /* LCD_H_ */
```

```c
/*******************************************************************************
***
 * @file spi.c :
 * @brief : This file contains UART initialization and functions for SPI handling
 *
 * @author : Dhiraj Bennadi
 * @date : Nov 24 2021
 *

*******************************************************************************
***/

/*---Includes---*/
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include "msp.h"
#include "uart.h"
#include "cbfifo.h"
#include "gpio.h"

/*---Defines---*/

uint16_t spi_tx_data = 0;
uint16_t spi_rx_data = 0;
/*----------------------------------------------------------------------------
 * Initializing SPI
 * (refer spi.h for additional details)
 * @Resource : This function is written with the help of example code provided
 * in assignment document
 ----------------------------------------------------------------------------*/
void init_spi()
{
    //P1.6 MOSI
    //P1.7 MISO
    //P1.5 CLK

    // Configure SPI
    P10->SEL0 |= BIT0 | BIT1 | BIT2 | BIT3;  // set 4-SPI pin as second function

    EUSCI_B3->CTLW0 |= EUSCI_B_CTLW0_SWRST; // Put state machine in reset

    EUSCI_B3->CTLW0 = EUSCI_B_CTLW0_SWRST | // Remain in reset state
            EUSCI_B_CTLW0_MST |             // SPI master
            EUSCI_B_CTLW0_SYNC |            // Synchronous mode
            EUSCI_B_CTLW0_CKPL |            // Clock polarity high
            EUSCI_B_CTLW0_MSB |             // MSB first
            EUSCI_B_CTLW0_MODE_2 |          // 4-pin mode
            EUSCI_B_CTLW0_STEM |            // STE mode select
            EUSCI_B_CTLW0_SSEL__ACLK;       // ACLK

    EUSCI_B3->BRW = 0x00;                   // /2,fBitClock = fBRCLK/(UCBRx+1).
    EUSCI_B3->CTLW0 &= ~EUSCI_B_CTLW0_SWRST;// **Initialize USCI state machine**
    // Enable eUSCI_B0 interrupt in NVIC module
    NVIC->ISER[0] = 1 << ((EUSCIB3_IRQn) & 31);

}

/*----------------------------------------------------------------------------
 * transmit over SPI
 * (refer spi.h for additional details)
 * @Resource : This function is written with the help of example code provided
 * in assignment document
 ----------------------------------------------------------------------------*/
void spi_tx(uint16_t data)
{
//    SPI_CS_PORT->OUT &= ~SPI_CS_PIN;
    spi_tx_data = data;
    EUSCI_B3->IFG |= EUSCI_B_IFG_TXIFG;// Clear TXIFG flag
    EUSCI_B3->IE |= EUSCI_B_IE_TXIE;    // Enable TX interrupt

}
```

```c
/************************************************************************************
*****
 * @file spi.h :
 * @brief : This file contains defines, includes, and function prototypes for spi.c
 *
 * @author : Dhiraj Bennadi
 * @date : Nov 24 2021
 *

*************************************************************************************
*****/

#ifndef SPI_H_
#define SPI_H_

#include <stdint.h>

/** ------------------------------------------------------------------------------
 *  @brief      Initializes SPI  to work at specified

 * @param buf :  none
 *
 * @return none
 ---------------------------------------------------------------------------------**/
void init_spi(void);

/*----------------------------------------------------------------------------
 * transmit over SPI
 * (refer spi.h for additional details)
 * @Resource : This function is written with the help of example code provided
 * in assignment document
 ----------------------------------------------------------------------------*/
void spi_tx(uint16_t data);

#endif /* SPI_H_ */
```

```
/*******************************************************************************
 * timers.c
 *  initializes timer routines
 *  Created on: Oct 1, 2021
 * @author: Dhiraj Bennadi
 *******************************************************************************/

#include "msp.h"



/*----------------------------------------------------------------------------
 * Simple loop delay
 ----------------------------------------------------------------------------*/
void delay(uint32_t count)
{
    uint32_t i;
    for(i = 0; i< count; i++);

}
/*
 * timers.h
 *
 *  Created on: Oct 1, 2021
 *      Author: Dhiraj Bennadi
 */
#ifndef TIMERS_H_
#define TIMERS_H_


/*----------------------------------------------------------------------------
 * This functionsets delay
 ----------------------------------------------------------------------------*/
void delay(uint32_t);

#endif /* TIMERS_H_ */
```

1.4 Appendix - Data Sheets and Application Notes

1. Power supply design: https://www.ti.com/lit/gpn/LM2576

2. MSP432 Reference manual: https://schaumont.dyn.wpi.edu/ece4703b21/_downloads/8bf98313124444641502c686bb90dbaa/msp432p401r-trm.pdf

3. MSP432 interfacing with 16x2 LCD: http://www.sparkfun.com/datasheets/LCD/GDM1602K.pdf

4. MS432 Datasheet: https://www.ti.com/lit/ds/slas826e/slas826e.pdf

5. AT89C51RC2 bootloader: http://ww1.microchip.com/downloads/en/devicedoc/doc4180.pdf