**Remote Interfacing with 8051**

Dhiraj Bennadi

Rajat Chaple

Final Project Report

ECEN 5613 Embedded System Design

December 11, 2021

## APPENDICES

## Appendix - Bill of Materials



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** |

## Appendix – SchematicsDiagram, schematic Description automatically generated

## 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};

//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;

//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\_ \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 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

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 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\_ \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @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;

}

**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\_ \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @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

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @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\_ \*/

## 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>