```

#include <p24FV32KA302.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#include "test_LEDs.h"
#include "pic_frequency.h"
#include "limits.h"
#include <libpic30.h>

#define CHAR_NULL '\0'
#define CHAR_CR '\r'
#define CHAR_NEW_LINE '\n'

#define BUF_LEN 250

struct struct_buf
{
    char buf[BUF_LEN];
    unsigned char posWrite;
    unsigned char posRead;
};

struct struct_buf bufSendWiFi;
struct struct_buf bufRecvWiFi;

struct struct_buf bufSendSPI;
struct struct_buf bufRecvSPI;

extern unsigned long timer_ms;

void wifi( bool init );
void wifiComm( void );
void wifiCommandSet( char *command, bool addPrefix );
void wifiServer( void );
bool wifiResponseCheck( char* response );
bool wifiResponseEnd( void );
void wifiCommSend( void );
bool wifiCommSendChar( char data );
bool wifiCommRecv( void );
bool wifiCommRecvChar( char *data );
void wifiInit( void );

bool spiCommRecvChar( char *data );
bool spiCommRecv( void );
bool spiCommSendChar( char data );
void spiCommSend( void );
void spiComm( void );
bool spiResponseEnd( void );
void spiCommandSet( char *command );
void spiServer( void );
void spi( bool init );
void spiInit( void );

void wifiInit( )
{
    ANSBbits.ANSB2 = 0;

    U1MODEbits.UARTEN = 0b0;

#define UART_BAUD 9600

    U1BRG = (((FCY / 16) / UART_BAUD) - 1);

    U1MODEbits.USIDL = 0b0;
    U1MODEbits.IREN = 0b0;
    U1MODEbits.RTSMD = 0b0;
    U1MODEbits.UEN = 0b00;
    U1MODEbits.WAKE = 0b0;
    U1MODEbits.LPBACK = 0b0;
    U1MODEbits.ABAUD = 0b0;
    U1MODEbits.RXINV = 0b0;
    U1MODEbits.BRGH = 0b0;
    U1MODEbits.PDSEL = 0b00;
    U1MODEbits.STSEL = 0b0;

    U1STAbits.URXISEL = 0b00;
    U1STAbits.UTXINV = 0b0;
    U1STAbits.UTXBRK = 0b0;
    U1STAbits.UTXEN = 0b1;
    U1STAbits.URXISEL = 0b00;
    U1STAbits.ADDEN = 0b0;

    U1MODEbits.UARTEN = 0b1;
    U1STAbits.UTXEN = 0b1;



    ANSBbits.ANSB0 = 0;
    ANSBbits.ANSB1 = 0;

    U2MODEbits.UARTEN = 0b0;

    U2BRG = (((FCY / 16) / UART_BAUD) - 1);

    U2MODEbits.USIDL = 0b0;
    U2MODEbits.IREN = 0b0;
    U2MODEbits.RTSMD = 0b0;
    U2MODEbits.UEN = 0b00;
    U2MODEbits.WAKE = 0b0;
    U2MODEbits.LPBACK = 0b0;
    U2MODEbits.ABAUD = 0b0;
    U2MODEbits.RXINV = 0b0;
    U2MODEbits.BRGH = 0b0;
    U2MODEbits.PDSEL = 0b00;
    U2MODEbits.STSEL = 0b0;

    U2STAbits.URXISEL = 0b00;
    U2STAbits.UTXINV = 0b0;
    U2STAbits.UTXBRK = 0b0;
    U2STAbits.UTXEN = 0b1;
    U2STAbits.URXISEL = 0b00;
    U2STAbits.ADDEN = 0b0;

    U2MODEbits.UARTEN = 0b1;
    U2STAbits.UTXEN = 0b1;

    return;
}

void wifi( bool init )
{
    // here we handle all things wifi

    static int stage = 0;

    if( init == true )
    {
    bufSendWiFi.posRead = 0;
    bufSendWiFi.posWrite = 0;
    bufRecvWiFi.posRead = 0;
    bufRecvWiFi.posWrite = 0;
    stage = 0;
    wifiInit( );
    spi( true );
    }

    wifiComm( );
    spi( false );

    static bool runone = false;

    switch( stage )
    {
    case 0: // reset and init
    // set command
    wifiCommandSet( "RST", true );
    stage++;
    break;
    case 1:
    //wait for response
    if( wifiResponseCheck( "ready" ) == true )
    {
        stage++;
    }
    break;
    case 2:
    wifiCommandSet( "RFPOWER=82", true );
    stage++;
    break;
    case 3:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 4:
    wifiCommandSet( "CWMODE_CUR=1", true );
    stage++;
    break;
    case 5:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 6:
    wifiCommandSet( "CWLAP", true );
    //wifiCommandSet( "ATE0", false );
    stage++;
    break;
    case 7:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 8:
    wifiCommandSet( "CWJAP_CUR=\"mWiFi\",\"mahadaga\"", true );
    wifiCommandSet( "CWJAP_CUR=\"AW2\",\"*****\"", true );
    stage++;
    break;
    case 9:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 10:
    wifiCommandSet( "CIFSR", true );
    stage++;
    break;
    case 11:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 12:
    wifiCommandSet( "CIPMUX=1", true );
    stage++;
    break;
    case 13:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 14:
    wifiCommandSet( "CIPSERVER=0", true );
    stage++;
    break;
    case 15:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 16:
    wifiCommandSet( "CIPSERVER=1", true );
    stage++;
    break;
    case 17:
    if( wifiResponseCheck( "OK" ) == true )
    {
        stage++;
    }
    break;
    case 18:
    if( runone == false )
    {
        //      spiCommandSet( "\r\nConnected!*" );
        for( int inx = 0; inx < 40; inx++ )
        {
        if( LED1READ == 0 )
        {
            LED1SET = 1;
            LED2SET = 1;
        }
        else
        {
            LED1SET = 0;
            LED2SET = 0;
        }
        __delay_ms( 25 );
        runone = true;
        }
        LED1SET = 0;
        LED2SET = 0;
    }
    wifiServer( );
    break;
    }

    if( wifiResponseEnd( ) == true )
    {
    bufRecvWiFi.posRead = 0;
    bufRecvWiFi.posWrite = 0;
    }

}

void wifiServer( void )
{

    int inPort = 0;
    int inDataLen = 0;

    // check the buffers and see if we have a command
    if( wifiResponseEnd( ) == true )
    {
    // check if data came in
    if( strncmp( "+IPD,", bufRecvWiFi.buf, 5 ) == 0 )
    {
        // get the port
#define ESP_BUF_IPD_START_PORT 5
#define ESP_BUF_IPD_START_LEN 7
        char tmpbuf[3];

        tmpbuf[0] = bufRecvWiFi.buf[ESP_BUF_IPD_START_PORT];
        tmpbuf[1] = CHAR_NULL;

        inPort = atoi( tmpbuf );

        tmpbuf[0] = bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN];
        switch( bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN + 1] )
        {
        case ':':
        tmpbuf[1] = CHAR_NULL;
        bufRecvWiFi.posRead = 9;
        break;
        default:
        tmpbuf[1] = bufRecvWiFi.buf[ESP_BUF_IPD_START_LEN + 1];
        tmpbuf[2] = CHAR_NULL;
        bufRecvWiFi.posRead = 10;
        break;
        }

        inDataLen = atoi( tmpbuf );

        // we make things here static in case a command spans multiple data packets from the esp8266
        static bool inCommand = false;
#define ESP_BUF_IPD_COMMAND_PROCESS_SIZE 200
        static char bufRecvWiFiCommand[ ESP_BUF_IPD_COMMAND_PROCESS_SIZE ];
        static int bufRecvWiFiCommandPos = 0;

        bufRecvWiFiCommandPos = 0;


        while( bufRecvWiFi.buf[bufRecvWiFi.posRead ] != CHAR_NULL )
        {
        if( inCommand == false )
        {
            if( bufRecvWiFi.buf[bufRecvWiFi.posRead] == '!' )
            {
            inCommand = true;

            //          if( LED1READ == 0 )
            //          {
            //              LED1SET = 1;
            //          }
            //          else
            //          {
            //              LED1SET = 0;
            //          }
            }
        }

        if( inCommand == true )
        {
            bufRecvWiFiCommand[bufRecvWiFiCommandPos] = bufRecvWiFi.buf[bufRecvWiFi.posRead ];
            bufRecvWiFiCommandPos++;
            if( bufRecvWiFiCommandPos >= ESP_BUF_IPD_COMMAND_PROCESS_SIZE )
            {
            bufRecvWiFiCommandPos = (ESP_BUF_IPD_COMMAND_PROCESS_SIZE - 1);
            }

            if( bufRecvWiFi.buf[bufRecvWiFi.posRead ] == '*' )
            {
            if( LED2READ == 0 )
            {
                LED2SET = 1;
            }
            else
            {
                LED2SET = 0;
            }

            bufRecvWiFiCommand[bufRecvWiFiCommandPos] = CHAR_NULL;
            inCommand = false;

            spiCommandSet( bufRecvWiFiCommand );
            bufRecvWiFiCommandPos = 0;
            }
        }
        bufRecvWiFi.posRead++;

        }
    }
    bufRecvWiFi.posRead = 0;
    bufRecvWiFi.posWrite = 0;


    }
}

void wifiCommandAdd( char *addCommand )
{
    //AT+CIPSENDBUF=<link ID>,<length>

    const char *commandDataPrefix = "CIPSEND=";

    int inx;
    char temp[BUF_LEN];
    int inxtemp;

    inx = 0;
    inxtemp = 0;

    while( commandDataPrefix[inx] != CHAR_NULL )
    {
    temp[inxtemp] = commandDataPrefix[inx];
    inxtemp++;
    if( inxtemp >= BUF_LEN )
    {
        inxtemp = (BUF_LEN - 1);
    }
    inx++;
    }

    temp[inxtemp] = '0';
    inxtemp++;
    temp[inxtemp] = ',';
    inxtemp++;


    // need the length of addCommand
    int dataLen = 0;

    while( addCommand[dataLen] != '*' )
    {
```

```
    dataLen++;
    }
    dataLen++; // we need to add one since this is a count, not index

    char dataLenChar[5];

    itoa( dataLenChar, dataLen, 10 );

    inx = 0;
    while( dataLenChar[inx] != CHAR_NULL )
    {
    temp[inxtemp] = dataLenChar[inx];
    inxtemp++;
    inx++;
    }

    temp[inxtemp] = CHAR_NULL;
    wifiCommandSet( temp, true );

    bufSendWiFi.buf[ bufSendWiFi.posWrite] = CHAR_NULL;
    bufSendWiFi.posWrite++;
    if( bufSendWiFi.posWrite >= BUF_LEN )
    {
    bufSendWiFi.posWrite = (BUF_LEN - 1);
    }

    inx = 0;
    while( addCommand[inx] != '*' )
    {
    bufSendWiFi.buf[ bufSendWiFi.posWrite] = addCommand[inx];
    bufSendWiFi.posWrite++;
    if( bufSendWiFi.posWrite >= BUF_LEN )
    {
        bufSendWiFi.posWrite = (BUF_LEN - 1);
    }
    inx++;
    }

    bufSendWiFi.buf[ bufSendWiFi.posWrite] = '*';
    bufSendWiFi.posWrite++;
    if( bufSendWiFi.posWrite >= BUF_LEN )
    {
    bufSendWiFi.posWrite = (BUF_LEN - 1);
    }

    return;

}

void wifiCommandSet( char *command, bool addPrefix )
{
    int inx;

    const char *commandPrefix = "AT+";
    const char *commandPostfix = "\r\n";

    if( addPrefix == true )
    {
    inx = 0;
    while( commandPrefix [inx] != CHAR_NULL )
    {
        bufSendWiFi.buf[ bufSendWiFi.posWrite] = commandPrefix[inx];
        bufSendWiFi.posWrite++;
        if( bufSendWiFi.posWrite >= BUF_LEN )
        {
        bufSendWiFi.posWrite = (BUF_LEN - 1);
        }
        inx++;
    }
    }

    inx = 0;
    while( command[inx] != CHAR_NULL )
    {
    bufSendWiFi.buf[bufSendWiFi.posWrite] = command[inx];
    bufSendWiFi.posWrite++;
    if( bufSendWiFi.posWrite >= BUF_LEN )
    {
        bufSendWiFi.posWrite = (BUF_LEN - 1);
    }
    inx++;
    }

    inx = 0;
    while( commandPostfix [inx] != CHAR_NULL )
    {
    bufSendWiFi.buf[ bufSendWiFi.posWrite] = commandPostfix[inx];
    bufSendWiFi.posWrite++;
    if( bufSendWiFi.posWrite >= BUF_LEN )
    {
        bufSendWiFi.posWrite = (BUF_LEN - 1);
    }
    inx++;
    }

    return;

}

bool wifiResponseCheck( char* response )
{
    bool match = false;

    if( wifiResponseEnd( ) == true )
    {
    match = true;
    int inx = 0;

    while( (response[inx] != CHAR_NULL) && (match == true) )
    {
        if( response[inx] != bufRecvWiFi.buf[inx] )
        {
        match = false;
        }
        inx++;
    }
    }

    if( match == true )
    {
    if( LED1READ == 0 )
    {
        LED1SET = 1;
    }
    else
    {
        LED1SET = 0;
    }
    }

    return match;
}

bool wifiResponseEnd( void )
{
    bool responseEnd = false;
    if( bufRecvWiFi.posWrite >= 2 )
    {
```

```c
    if( bufRecvWiFi.buf[bufRecvWiFi.posWrite - 2] == CHAR_CR )
    {
        if( bufRecvWiFi.buf[bufRecvWiFi.posWrite - 1] == CHAR_NEW_LINE )
        {
        responseEnd = true;
        }


    }
    }

    return responseEnd;
}

void wifiComm( )
{
    wifiCommSend( );
    wifiCommRecv( );

    return;
}

void wifiCommSend( void )
{
    if( bufSendWiFi.posRead != bufSendWiFi.posWrite )
    {
    if( U1STAbits.UTXBF == 0 )
    {
        if( wifiCommSendChar( bufSendWiFi.buf[bufSendWiFi.posRead] ) == true )
        {
        bufSendWiFi.posRead++;
        if( bufSendWiFi.posRead >= BUF_LEN )
        {
            bufSendWiFi.posRead = 0;
        }
        }
    }
    }
    else
    {
    bufSendWiFi.posRead = 0;
    bufSendWiFi.posWrite = 0;
    }
    return;
}

bool wifiCommSendChar( char data )
{
    bool dataSent = false;
    static bool pause = false;
    static unsigned long pauseTimer;
    static bool pauseTimerOverflowWait;
#define PAUSE_TIME_MS 100

    if( pause == false )
    {
    if( data == CHAR_NULL )
    {
        pause = true;
        if( (ULONG_MAX - timer_ms) > PAUSE_TIME_MS )
        {
        pauseTimer = timer_ms + PAUSE_TIME_MS;
        }
        else
        {
        pauseTimer = PAUSE_TIME_MS - (ULONG_MAX - timer_ms);
        pauseTimerOverflowWait = true;
        }

        dataSent = true;
    }
    else
    {
        if( U1STAbits.UTXBF == 0 )
        {
        U1TXREG = data;
        dataSent = true;
        //      U2TXREG = data;
        }
    }
    }
    else
    {
    if( pauseTimerOverflowWait == true )
    {
        if( timer_ms < pauseTimer )
        {
        pauseTimerOverflowWait = false;
        }
    }

    if( pauseTimerOverflowWait == false )
    {
        if( timer_ms >= pauseTimer )
        {
        pause = false;
        }
    }
    }

    return dataSent;
}

bool wifiCommRecv( void )
{
    bool dataReceived = false;
    char data;

    if( wifiCommRecvChar( &data ) == true )
    {
    bufRecvWiFi.buf[bufRecvWiFi.posWrite] = data;
    dataReceived = true;
    bufRecvWiFi.posWrite++;
    if( bufRecvWiFi.posWrite >= BUF_LEN )
    {
        bufRecvWiFi.posWrite = 0;
    }
    bufRecvWiFi.buf[bufRecvWiFi.posWrite] = CHAR_NULL;
    }

    return dataReceived;
}

bool wifiCommRecvChar( char *data )
{
    bool dataReceived = false;

    if( U1STAbits.URXDA == 1 )
    {
    *data = U1RXREG;
    dataReceived = true;
    U2TXREG = *data;
    }

    return dataReceived;
}

void spi( bool init )
```

```
{
    if( init == true )
    {
    bufSendSPI.posRead = 0;
    bufSendSPI.posWrite = 0;
    bufRecvSPI.posRead = 0;
    bufRecvSPI.posWrite = 0;
    spiInit( );
    U2TXREG = '\r';
    __delay_ms( 100 );
    U2TXREG = '\n';

    }

    static bool spiInReset = false;
    if( PORTBbits.RB12 == 1 )
    {
    if( spiInReset == false )
    {
        spiInReset = true;
        SPI2STATbits.SPIEN = 0; //disable SPI
    }
    }
    else
    {
    if( spiInReset == true )
    {

        spiInReset = false;
        bufRecvSPI.posRead = 0;
        bufRecvSPI.posWrite = 0;
        //      spiCommandSet( "!Set;Watts;55*" );
        SPI2STATbits.SPIEN = 1; //enable SPI
    }

    spiComm( );
    spiServer( );

    // here we need to check if we pass on a command

    if( spiResponseEnd( ) == true )
    {
        bufRecvSPI.posRead = 0;
        bufRecvSPI.posWrite = 0;
    }
    }
}

void spiServer( )
{

    // check the buffers and see if we have a command
    if( spiResponseEnd( ) == true )
    {
    //  LED1SET = 1;

    if( bufRecvSPI.buf[0] == '!' )
    {
        // LED1SET = 1;
        wifiCommandAdd( bufRecvSPI.buf );
    }
    }
}

void spiCommandSet( char *command )
{
    int inx;

    inx = 0;
    while( command[inx] != '*' )
    {
    //LED1SET = 1;
    bufSendSPI.buf[bufSendSPI.posWrite] = command[inx];
    bufSendSPI.posWrite++;
    if( bufSendSPI.posWrite >= BUF_LEN )
    {
        bufSendSPI.posWrite = (BUF_LEN - 1);
    }
    inx++;
    }
    bufSendSPI.buf[bufSendSPI.posWrite] = command[inx];
    bufSendSPI.posWrite++;
    if( bufSendSPI.posWrite >= BUF_LEN )
    {
    bufSendSPI.posWrite = (BUF_LEN - 1);
    }

    return;
}

bool spiResponseEnd( void )
{
    bool responseEnd = false;

    if( bufRecvSPI.buf[ bufRecvSPI.posWrite - 1 ] == '*' )
    {
    //  if( LED1READ == 0 )
    //  {
    //      LED1SET = 1;
    //  }
    //  else
    //  {
    //      LED1SET = 0;
    //  }

    responseEnd = true;
    }

    return responseEnd;
}

void spiComm( void )
{
    spiCommSend( );
    spiCommRecv( );

    return;
}

void spiCommSend( void )
{
    if( bufSendSPI.posRead != bufSendSPI.posWrite )
    {
    //LED1SET = 1;
    //  if( U2STAbits.UTXBF == 0 )
    //  {
    if( spiCommSendChar( bufSendSPI.buf[bufSendSPI.posRead] ) == true )
    {
        bufSendSPI.posRead++;
        if( bufSendSPI.posRead >= BUF_LEN )
        {
        bufSendSPI.posRead = 0;
        }
    }
    //  }
    }
    else
    {
```

```
    bufSendSPI.posRead = 0;
    bufSendSPI.posWrite = 0;
    }
    return;
}

//bool spiCommSendCharUART( char data )
//{
//    bool dataSent = false;
//
//    if( U2STAbits.UTXBF == 0 )
//    {
//  U2TXREG = data;
//  dataSent = true;
//    }
//
//    return dataSent;
//}

bool spiCommSendChar( char data )
{
    bool sendGood = false;


    if( SPI2STATbits.SPITBF == 0b0 ) //if in enhance mode use SPI1STATbits.SR1MPT
    {
    SPI2BUF = data;
    sendGood = true;
    //  U2TXREG = data;
    }

    return sendGood;
}

bool spiCommRecv( void )
{
    bool dataReceived = false;
    char data;

    if( spiCommRecvChar( &data ) == true )
    {
    bufRecvSPI.buf[bufRecvSPI.posWrite] = data;
    dataReceived = true;
    bufRecvSPI.posWrite++;
    if( bufRecvSPI.posWrite >= BUF_LEN )
    {
        bufRecvSPI.posWrite = 0;
    }
    bufRecvSPI.buf[bufRecvSPI.posWrite] = CHAR_NULL;
    }

    return dataReceived;
}

//bool spiCommRecvCharUART( char *data )
//{
//    bool dataReceived = false;
//
//    if( U2STAbits.URXDA == 1 )
//    {
//  *data = U2RXREG;
//  dataReceived = true;
//    }
//
//    return dataReceived;
//}

bool spiCommRecvChar( char *data )
{
    bool recvGood = false;

    if( SPI2STATbits.SPIRBF == 0b1 )
    {
    *data = SPI2BUF;
    recvGood = true;
    if( *data != '@' )
    {
        U2TXREG = *data;
    }
    }

    return recvGood;
}

void spiInit( void )
{
    // make sure analog is turned off - it messes with the pins
    ANSA = 0;
    ANSB = 0;
    ANSBbits.ANSB12 = 0;

    TRISBbits.TRISB5 = 1;
    TRISBbits.TRISB6 = 0;

    TRISBbits.TRISB12 = 1;
    TRISAbits.TRISA7 = 1;

    //SPI2 Initialize as Slave
    SPI2CON1bits.MSTEN = 0;

    SPI2CON1bits.DISSCK = 0b1; // SPI clock disabled
    SPI2CON1bits.DISSDO = 0b0; // SDO used
    SPI2CON1bits.MODE16 = 0b0; // 8 bit mode
    SPI2CON1bits.SMP = 0b0; // sample phase mode middle
    SPI2CON1bits.CKE = 0b1; // serial data changes on active to idle clock state
    SPI2CON1bits.SSEN = 0b1; // yes a slave
    SPI2CON1bits.CKP = 0b1; // clock idle is high
    SPI2CON1bits.SPRE = 0b000; // secondary prescale 8:1, not used - no clock is run

    SPI2CON2bits.FRMEN = 0b0; // frame mode, unused
    SPI2CON2bits.SPIFSD = 0b0; // frame mode, unused
    SPI2CON2bits.SPIFPOL = 0b0; // frame mode, unused
    SPI2CON2bits.SPIFE = 0b0; // frame mode, unused

    SPI2CON2bits.SPIBEN = 0b0; // 1=enhanced buffer mode

    SPI2STATbits.SPIROV = 0; //clear flag for overflow data

    SPI2BUF = SPI2BUF; //clear the buffer
    SPI2STATbits.SPIEN = 1; //enable SPI

    return;
}
```