STM32 MCU Development

STM32 单片机开发

-- STM32 无线通信实时监控

目录

1. WiFi 模块实时上报温湿度

1.1. 编写 esp8266.h

在项目路径 Core\Inc 下编写 ESP8266 WiFi 模块相关接口函数的头文件，如下所示：

```
#ifndef INC_ESP8266_H_
#define INC_ESP8266_H_

#include <stdio.h>

#define wifi_huart              &huart2         /* WiFi 模块使用的串口 */
#define g_wifi_rxbuf            g_uart2_rxbuf   /* WiFi 模块的接收 buffer */
#define g_wifi_rxbytes          g_uart2_bytes   /* WiFi 模块接收的数据大小 */

/* 清除 WiFi 模块接收 buffer 里的数据内容宏，用宏不用函数是因为函数调用需要额外时间开销 */
#define clear_atcmd_buf()     do { memset(g_wifi_rxbuf, 0, sizeof(g_wifi_rxbuf)); \
                                   g_wifi_rxbytes=0; } while(0)

/* ESP8266 WiFi 模块发送 AT 命令函数。返回值为 0 表示成功，!0 表示失败 */
#define EXPECT_OK    "OK\r\n"
extern int send_atcmd(char *atcmd, char *expect_reply, unsigned int timeout);

/* ESP8266 WiFi 模块初始化函数。返回值为 0 表示成功，!0 表示失败  */
extern int esp8266_module_init(void);

/* ESP8266 WiFi 模块复位重启函数。返回值为 0 表示成功，!0 表示失败  */
extern int esp8266_module_reset(void);

/* ESP8266 WiFi 模块连接路由器函数。返回值为 0 表示成功，!0 表示失败  */
extern int esp8266_join_network(char *ssid, char *pwd);

/* ESP8266 获取自己的 IP 地址和网关 IP 地址。返回值为 0 表示成功，!0 表示失败  */
int esp8266_get_ipaddr(char *ipaddr, char *gateway, int ipaddr_size);

/* ESP8266 WiFi 模块做 ping 命令测试网络连通性。返回值为 0 表示成功，!0 表示失败  */
int esp8266_ping_test(char *host);

/* ESP8266 WiFi 模块建立 TCP socket 连接函数。返回值为 0 表示成功，!0 表示失败 */
extern int esp8266_sock_connect(char *servip, int port);

/* ESP8266 WiFi 模块断开 TCP socket 连接函数。返回值为 0 表示成功，!0 表示失败 */
extern int esp8266_sock_disconnect(void);

/* ESP8266 WiFi 通过 TCP Socket 发送数据函数。返回值为 0 表示失败，>0 表示成功发送字节数 */
extern int esp8266_sock_send(unsigned char *data, int bytes);

/* ESP8266 WiFi 通过 TCP Socket 接收数据函数。返回值为 0 无数据，>0 表示接收到数据字节数 */
extern int esp8266_sock_recv(unsigned char *buf, int size);

#endif /* INC_ESP8266_H_ */
```

1.2. 编写 esp8266.c

在项目路径 Core\Src 下编写 ESP8266 WiFi 模块相关接口函数的 C 文件，如下所示：

```
#include <stdlib.h>
#include "usart.h"
#include "esp8266.h"

/* WiFi 模块驱动调试宏,注释下面两个宏定义可以取消调试打印 */
//#define CONFIG_WIFI_DEBUG
#define CONFIG_WIFI_PRINT

#ifdef CONFIG_WIFI_DEBUG
#define wifi_dbg(format,args...) printf(format, ##args)
#else
#define wifi_dbg(format,args...) do{} while(0)
#endif

#ifdef CONFIG_WIFI_PRINT
#define wifi_print(format,args...) printf(format, ##args)
#else
#define wifi_print(format,args...) do{} while(0)
#endif


int send_atcmd(char *atcmd, char *expect_reply, unsigned int timeout)
{
        int                     rv = 1;
        unsigned int             i;
        char                    *expect;

        /* check function input arguments validation */
        if(  !atcmd || strlen(atcmd)<=0 )
        {
             wifi_print("ERROR: Invalid input arguments\r\n");
             return -1;
        }

        wifi_dbg("\r\nStart send AT command: %s", atcmd);
        clear_atcmd_buf();
        HAL_UART_Transmit(wifi_huart, (uint8_t *)atcmd, strlen(atcmd), 1000);

        expect = expect_reply ? expect_reply : "OK\r\n";

        /* Receive AT reply string by UART interrupt handler, stop by "OK/ERROR" or timeout */
        for(i=0; i<timeout; i++)
        {
             if( strstr(g_wifi_rxbuf, expect) )
             {
                  wifi_dbg("AT command Got expect reply '%s'\r\n", expect);
                  rv = 0;
                  goto CleanUp;
             }
```

```
            if( strstr(g_wifi_rxbuf, "ERROR\r\n") || strstr(g_wifi_rxbuf, "FAIL\r\n"))
            {
                rv = 2;
                goto CleanUp;
            }

            HAL_Delay(1);
        }


CleanUp:
        wifi_dbg("<<<< AT command reply:\r\n%s", g_wifi_rxbuf);
        return rv;
}

int atcmd_send_data(unsigned char *data, int bytes, unsigned int timeout)
{
        int                     rv = -1;
        unsigned int             i;

        /* check function input arguments validation */
        if(  !data || bytes <= 0 )
        {
            wifi_print("ERROR: Invalid input arguments\r\n");
            return -1;
        }

        wifi_dbg("\r\nStart AT command send [%d] bytes data\n", bytes);
        clear_atcmd_buf();
        HAL_UART_Transmit(wifi_huart, data, bytes, 1000);

        /* Receive AT reply string by UART interrupt handler, stop by "OK/ERROR" or timeout */
        for(i=0; i<timeout; i++)
        {
            if( strstr(g_wifi_rxbuf, "SEND OK\r\n") )
            {
                rv = 0;
                goto CleanUp;
            }

            if( strstr(g_wifi_rxbuf, "ERROR\r\n") )
            {
                rv = 1;
                goto CleanUp;
            }

            HAL_Delay(1);
        }


CleanUp:
        wifi_dbg("<<<< AT command reply:\r\n%s", g_wifi_rxbuf);
        return rv;
}
```

```
int esp8266_module_init(void)
{
	int	i;

	wifi_print("INFO: Reset ESP8266 module now...\r\n");
	send_atcmd("AT+RST\r\n", EXPECT_OK, 500);

	for(i=0; i<6; i++)
	{
		if( !send_atcmd("AT\r\n", EXPECT_OK, 500) )
		{
			wifi_print("INFO: Send AT to ESP8266 and got reply ok\r\n");
			break;
		}

		HAL_Delay(100);
	}

	if( i>= 6 )
	{
		wifi_print("ERROR: Can't receive AT replay after reset\r\n");
		return -2;
	}

	if( send_atcmd("AT+CWMODE=1\r\n", EXPECT_OK, 500) )
	{
		wifi_print("ERROR: Set ESP8266 work as Station mode failure\r\n");
		return -3;
	}

	if( send_atcmd("AT+CWDHCP=1,1\r\n", EXPECT_OK, 500) )
	{
		wifi_print("ERROR: Enable ESP8266 Station mode DHCP failure\r\n");
		return -4;
	}

#if 0
	if( send_atcmd("AT+GMR\r\n", EXPECT_OK, 500) )
	{
		wifi_print("ERROR: AT+GMR check ESP8266 reversion failure\r\n");
		return -5;
	}
#endif

	HAL_Delay(500);
	return 0;
}
```

```
int esp8266_join_network(char *ssid, char *pwd)
{
        char          atcmd[128] = {0x00};
        int           i;

        if( !ssid || !pwd )
        {
             wifi_print("ERROR: Invalid input arguments\r\n");
             return -1;
        }

        snprintf(atcmd, sizeof(atcmd), "AT+CWJAP=\"%s\",\"%s\"\r\n", ssid, pwd);
        if( send_atcmd(atcmd, "CONNECTED", 10000) )
        {
             wifi_print("ERROR: ESP8266 connect to '%s' failure\r\n", ssid);
             return -2;
        }
        wifi_print("INFO: ESP8266 connect to '%s' ok\r\n", ssid);

        /* check got IP address or not by netmask (255.)*/
        for(i=0; i<10; i++)
        {
             if( !send_atcmd("AT+CIPSTA_CUR?\r\n", "255.", 1000) )
             {
                  wifi_print("INFO: ESP8266 got IP address ok\r\n");
                  return 0;
             }

             HAL_Delay(300);
        }

        wifi_print("ERROR: ESP8266 assigned IP address failure\r\n");
        return -3;
}

/*
+CIPSTA_CUR:ip:"192.168.2.100"
+CIPSTA_CUR:gateway:"192.168.2.1"
*/
static int util_parser_ipaddr(char *buf, char *key, char *ipaddr, int size)
{
        char       *start;
        char       *end;
        int         len;

        if( !buf || !key || !ipaddr )
        {
             return -1;
        }

        /* find the key string */
        start = strstr(buf, key);
        if( !start )
        {
             return -2;
        }
```

```
        start+=strlen(key) + 1;  /* Skip " */
        end = strchr(start, '"');  /* find last " */
        if( !end )
        {
             return -3;
        }

        len = end - start;
        len = len>size ? size : len;

        memset(ipaddr, 0, size);
        strncpy(ipaddr, start, len);

        return 0;
}

int esp8266_get_ipaddr(char *ipaddr, char *gateway, int ipaddr_size)
{
        if( !ipaddr || !gateway || ipaddr_size<7 )
        {
             wifi_print("ERROR: Invalid input arguments\r\n");
             return -1;
        }

        if( send_atcmd("AT+CIPSTA_CUR?\r\n", "255.", 1000) )
        {
             wifi_print("ERROR: ESP8266 AT+CIPSTA_CUR? command failure\r\n");
             return -2;
        }

        if( util_parser_ipaddr(g_wifi_rxbuf, "ip:", ipaddr, ipaddr_size) )
        {
             wifi_print("ERROR: ESP8266 AT+CIPSTA_CUR? parser IP address failure\r\n");
             return -3;
        }

        if( util_parser_ipaddr(g_wifi_rxbuf, "gateway:", gateway, ipaddr_size) )
        {
             wifi_print("ERROR: ESP8266 AT+CIPSTA_CUR? parser gateway failure\r\n");
             return -4;
        }

        wifi_print("INFO: ESP8266 got IP address[%s] gateway[%s] ok\r\n", ipaddr, gateway);
        return 0;
}

int esp8266_ping_test(char *host)
{
        char          atcmd[128] = {0x00};

        if( !host )
        {
             wifi_print("ERROR: Invalid input arguments\r\n");
             return -1;
        }
```

```
        snprintf(atcmd, sizeof(atcmd), "AT+PING=\"%s\"\r\n", host);
        if( send_atcmd(atcmd, EXPECT_OK, 3000) )
        {
            wifi_print("ERROR: ESP8266 ping test [%s] failure\r\n", host);
            return -2;
        }

        wifi_print("INFO: ESP8266 ping test [%s] ok\r\n", host);
        return 0;
}

int esp8266_sock_connect(char *servip, int port)
{
        char          atcmd[128] = {0x00};

        if( !servip || port<=0 )
        {
            wifi_print("ERROR: Invalid input arguments\r\n");
            return -1;
        }

        send_atcmd("AT+CIPMUX=0\r\n", EXPECT_OK, 1500);

        snprintf(atcmd, sizeof(atcmd), "AT+CIPSTART=\"TCP\",\"%s\",%d\r\n", servip, port);
        if( send_atcmd(atcmd, "CONNECT\r\n", 1000) )
        {
            wifi_print("ERROR: ESP8266 socket connect to [%s:%d] failure\r\n", servip, port);
            return -2;
        }

        wifi_print("INFO: ESP8266 socket connect to [%s:%d] ok\r\n", servip, port);
        return 0;
}

int esp8266_sock_disconnect(void)
{
        send_atcmd("AT+CIPCLOSE\r\n", EXPECT_OK, 1500);
        return 0;
}

int esp8266_sock_send(unsigned char *data, int bytes)
{
        char          atcmd[128] = {0x00};

        if( !data || bytes<= 0)
        {
            wifi_print("ERROR: Invalid input arguments\r\n");
            return -1;
        }

        snprintf(atcmd, sizeof(atcmd), "AT+CIPSEND=%d\r\n", bytes);
        if( send_atcmd(atcmd, ">", 500) )
        {
            wifi_print("ERROR: AT+CIPSEND command failure\r\n");
            return 0;
        }
```

```
        if( atcmd_send_data((unsigned char *)data, bytes, 1000) )
        {
             wifi_print("ERROR: AT+CIPSEND send data failure\r\n");
             return 0;
        }

        return bytes;
}


int esp8266_sock_recv(unsigned char *buf, int size)
{
        char               *data = NULL;
        char               *ptr = NULL;

        int                 len;
        int                 rv;
        int                 bytes;

        if( !buf || size<= 0)
        {
             wifi_print("ERROR: Invalid input arguments\r\n");
             return -1;
        }

        if( g_wifi_rxbytes<= 0 )
        {
             return 0;
        }

        /* No data arrive or not integrated */
        if( !(ptr=strstr(g_wifi_rxbuf, "+IPD,")) ||  !(data=strchr( g_wifi_rxbuf, ':' )) )
        {
             return 0;
        }

        data ++;
        bytes = atoi(ptr+strlen("+IPD,"));

        len = g_wifi_rxbytes - (data-g_uart2_rxbuf);

        if( len < bytes )
        {
             wifi_dbg("+IPD data not receive over, receive again later ...\r\n");
             return 0;
        }

        memset(buf, 0, size);
        rv = bytes>size ? size : bytes;
        memcpy(buf, data, rv);

        clear_atcmd_buf();

        return rv;
}
```

2. ESP8266 WiFi 收发测试

2.1. 修改 main.c 文件

修改 main.c 文件，添加 ESP8266 网络 socket 收发的测试程序。该程序上电后将自动连接相应的路由器，然后再 socket 连接服务器并每隔 3s 发送一个 Hello 字符串，同时接收并打印来自服务器端的数据。

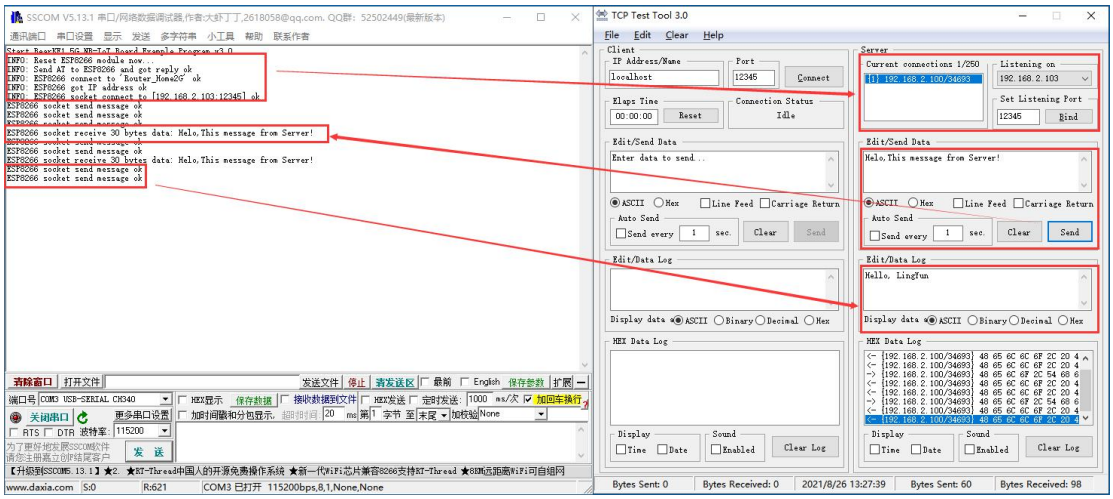
```
... ...

在文件开始处添加头文件 esp8266.h ：
/* USER CODE BEGIN Includes */
#include <string.h>
#include "dht11.h"
#include "sht30.h"
#include "core_json.h"
#include "oled.h"
#include "esp8266.h"
/* USER CODE END Includes */

... ...

int main(void)
{
   /* USER CODE BEGIN 1 */
   uint32_t           last_time = 0;  /* 每隔 3s 上报一次,上一次上报的时间 */
   unsigned char      buf[256];  /* WiFi 模块 socket 接收的 buffer */
   int                rv;
   /* USER CODE END 1 */
  ... ...

  /* USER CODE BEGIN WHILE */
  sysled_hearbeat();
  beep_start(2, 300);
  printf("Start BearKE1 5G NB-IoT Board Example Program v3.0\r\n");

  OLED_Init();
  OLED_ShowBanner(TIME_1S*2);

  esp8266_module_init();
  esp8266_join_network("Router_Home2G", "password");
  esp8266_sock_connect("192.168.2.103", 12345);

  while (1)
  {
     if( (rv=esp8266_sock_recv(buf, sizeof(buf))) > 0 )
     {
        printf("ESP8266 socket receive %d bytes data: %s\n", rv, buf);
     }

     if( time_after(HAL_GetTick(), last_time+3000) )
     {
        rv = esp8266_sock_send("Hello, LingYun", strlen("Hello, LingYun"));
        if( !rv )
```

```
            printf("ESP8266 socket send message failure, rv=%d\n", rv);
        else
            printf("ESP8266 socket send message ok\n");



       last_time = HAL_GetTick(); /*update last report time */
     }
   /* USER CODE END WHILE */

   /* USER CODE BEGIN 3 */  }
 ... ...
}

... ...
```

2.2. 运行测试

PC 上首先 TCP Test Tools 程序，开启 socket 服务器。然后编译并烧录运行单片机程序，使用串口调试助手打开 PC 上相应的串口。

我们可以从串口到 ESP8266 WiFi 模块将自动配置并连接无线路由器，之后开始每隔 3 秒发送一个字符串给 PC 端的 socket 服务器；我们通过 PC 端的 socket 服务端给 WiFi 模块发送数据时单片机端也能接收到这个消息。



该程序有些缺陷：

- 如果路由器在 ESP8266 连接之前没有开启，则系统不会再连了；
- 如果 PC 端服务器在 ESP8266 socket 连接之前没有开启，则 socket 也连接不上；
- 如果路由器或 PC 上服务器程序重启之后，单片机也将连接不上；

接下来我们将继续修改完善该程序。

2.3. 程序 bug 修复

之所以出现上述 bug，这是因为初始化 ESP8266 WiFi 模块、连接路由器、连接 socket 只是在 while(1)循环前只做了一次，这样出现异常之后就不能修复了。这个 bug 告诫我们，在编程的时候不应该只考虑程序正确运行该怎么运行，更多的应该是考虑如果程序异常该如何处理？

一个很容易的解决方法就是，在程序中设个标志用来表示 ESP8266 的 WiFi 连接的状态，如果没有连上 WiFi 模块，则在 while(1)循环里一直尝试连接；如果没有连上则不用再连接了。这样当程序发送失败时，就可以重连。

修改 main.c 的测试代码如下：

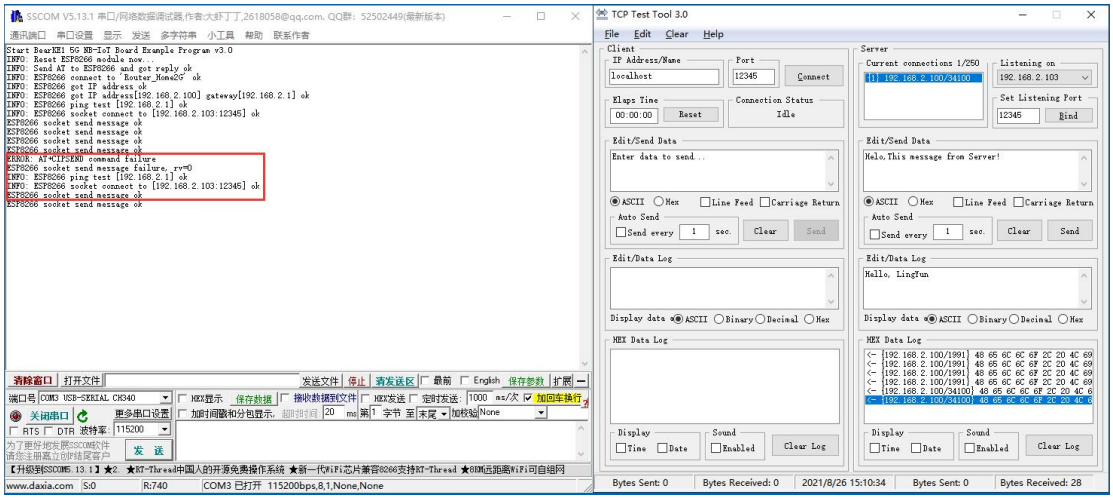
```
... ...

在这里添加两个宏定义，这是 WiFi 连接状态的相应标志位：
/* USER CODE BEGIN PV */
#define FLAG_WIFI_CONNECTED     (1<<0)   /* WiFi 连接路由器标志位 */
#define FLAG_SOCK_CONNECTED     (1<<1)   /* Socket 连接服务器标志位 */
/* USER CODE END PV */
... ...

int main(void)
{
  /* USER CODE BEGIN 1 */
  uint32_t           last_time = 0; /* 每隔 3s 上报一次,上一次上报的时间 */
  unsigned char      buf[256];  /* WiFi 模块 socket 接收的 buffer */
  int                rv;
  char               ipaddr[16];
  char               gateway[16];
  unsigned char      wifi_flag = 0;
    /* USER CODE END 1 */
  ... ...

  /* USER CODE BEGIN WHILE */
  sysled_hearbeat();
  beep_start(2, 300);
  printf("Start BearKE1 5G NB-IoT Board Example Program v4.0\r\n");

  OLED_Init();
  OLED_ShowBanner(TIME_1S*2);

  esp8266_module_init();

  while (1)
  {
     /* WiFi 没有连接上无线路由器的话，开始连接无线路由器并 ping 测试 */
     if( ! (wifi_flag&FLAG_WIFI_CONNECTED) )
     {
           if( esp8266_join_network("Router_Home2G", "password") )
           {
                esp8266_module_init();
                HAL_Delay(2000);
                continue;
```

```
            }



            if( esp8266_get_ipaddr(ipaddr, gateway, sizeof(ipaddr) ) )
            {
                HAL_Delay(1000);
                continue;
            }

            if( esp8266_ping_test(gateway) )
            {
                HAL_Delay(1000);
                continue;
            }
            wifi_flag |= FLAG_WIFI_CONNECTED; /* set wifi connected flag */
        }

        /* 网络 socket 没有连上 socket 服务器的话就开始连接服务器 */
        if( ! (wifi_flag&FLAG_SOCK_CONNECTED) )
        {
            if( esp8266_sock_connect("192.168.2.103", 12345) )
            {
                HAL_Delay(1000);
                continue;
            }
            wifi_flag |= FLAG_SOCK_CONNECTED; /* set socket connected flag */
        }

        /* 接收并打印 socket 服务器发过来的数据 */
        if( (rv=esp8266_sock_recv(buf, sizeof(buf))) > 0 )
        {
            printf("ESP8266 socket receive %d bytes data: %s\n", rv, buf);
        }

        /* 定时发送数据到 socket 服务器 */
        if( time_after(HAL_GetTick(), last_time+3000) )
        {
            rv = esp8266_sock_send("Hello, LingYun", strlen("Hello, LingYun"));
            if( rv > 0 )
            {
                printf("ESP8266 socket send message ok\n");
            }
            else
            {
                printf("ESP8266 socket send message failure, rv=%d\n", rv);
                wifi_flag &= ~FLAG_SOCK_CONNECTED; /* clear socket connected flag */

                if( esp8266_ping_test(gateway) )
                {
                    wifi_flag &= ~FLAG_WIFI_CONNECTED; /* clear wifi connected flag */
                }
            }

            last_time = HAL_GetTick(); /*update last report time */
        }
        /* USER CODE END WHILE */
    }
```

```
 ... ...
}
```

2.4. 修复后运行测试

代码 bug 修复后，重新编译并烧录运行单片机程序，使用串口调试助手打开 PC 上相应的串口。这时候无论 PC 端 socket 服务器程序先启动，还是 STM32 单片机先启动，程序都能够正常运行。程序运行期间 socket 断开，程序也会自动重连并正常收发数据。



3. STM32 无线通信实时监控

3.1. WiFi 模块网络上报温湿度值

此前我们已经做好了使用 SHT30 采样温湿度值，并通过串口定时上报的代码。现在我们只需要在这个基础上，做些小的修改即可实现 ESP8266 无线上报。

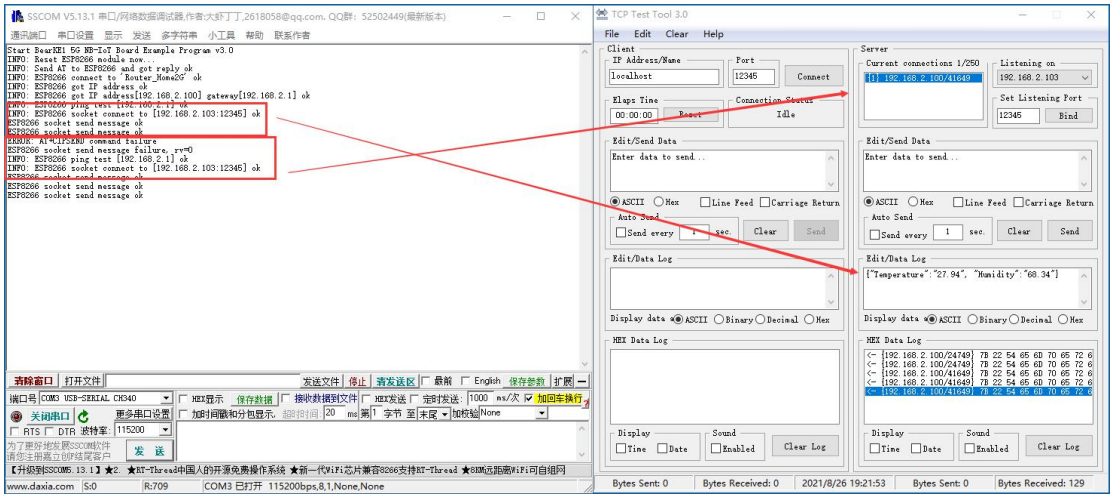
修改 while(1)循环里的定时上报代码，这里直接调用以前定义的上报函数 `report_tempRH_json()` 。

```
... ...

  while (1)
  {
      ... ...

      /* 接收并打印 socket 服务器发过来的数据 */
      if( (rv=esp8266_sock_recv(buf, sizeof(buf))) > 0 )
      {
            printf("ESP8266 socket receive %d bytes data: %s\n", rv, buf);
      }

      /* 定时发送数据到 socket 服务器 */
      if( time_after(HAL_GetTick(), last_time+3000) )
      {
            rv = report_tempRH_json();
            if( 0 == rv )
            {
                 printf("ESP8266 socket send message ok\n");
            }
            else
            {
                 printf("ESP8266 socket send message failure, rv=%d\n", rv);

                 wifi_flag &= ~FLAG_SOCK_CONNECTED; /* clear socket connected flag */

                 if( esp8266_ping_test(gateway) )
                 {
                      wifi_flag &= ~FLAG_WIFI_CONNECTED; /* clear wifi connected flag */
                 }
            }

            last_time = HAL_GetTick(); /*update last report time */
      }
  }

 ... ...
```

修改 report_tempRH_json() 函数，将之前的串口发送改成 ESP8266 WiFi 的 socket 发送函数。

```
... ...

  int report_tempRH_json(void)
  {
      char         buf[128];
      float        temperature, humidity=0.0;
      uint32_t     temp, humd;
      int           rv;

      if ( SHT30_SampleData(&temperature, &humidity) < 0 )
      {
          printf("ERROR: SHT30 Sample data failure\n");
          return -1;
      }

      memset(buf, 0, sizeof(buf));
      snprintf(buf, sizeof(buf), "{\"Temperature\":\"%.2f\", \"Humidity\":\"%.2f\"}", temperature,
humidity);

      temp = (int)(temperature*100);
      humd = (int)(humidity*100);
      OLED_ShowTempHumdity(temp, humd, TIME_1S*2);

      rv = esp8266_sock_send((uint8_t *)buf, strlen(buf));

      return rv>0 ? 0 : -2;
  }

  /* USER CODE END WHILE */
  ... ...
```

重新编译并烧录运行单片机程序，使用串口调试助手打开 PC 上相应的串,运行 Windows 上的 TCP Test Tools 程序并开启 TCP 服务。这时候可以看到单片机每隔 3 秒钟以 JSON 协议格式上报一次当前采样温湿度值，如果我们把 socket 断开，程序也会自动重连并正常上报。



3.2. WiFi 模块远程控制 Led 灯实现

修改 while(1)循环里的 ESP8266 WiFi 接收数据的代码，添加 Led 控制命令解析函数控制 Led 灯亮灭。

```
... ...

  while (1)
  {
      ... ...

      /* 接收并打印 socket 服务器发过来的数据 */
      if( (rv=esp8266_sock_recv(buf, sizeof(buf))) > 0 )
      {
            parser_led_json((char *)buf, rv);
            printf("ESP8266 socket receive %d bytes data: %s\n", rv, buf);
      }

      /* 定时发送数据到 socket 服务器 */
      ... ...

  }
... ...
```

重新编译并烧录运行单片机程序，使用串口调试助手打开 PC 上相应的串,运行 Windows 上的 TCP Test Tools 程序并开启 TCP 服务，这时候可以看到单片机每隔 3 秒钟以 JSON 协议格式上报一次当前采样温湿度值。

在 TCP Test Tools 程序上给单片机客户端发 JSON 格式命令 {"RedLed":"off","GreenLed":"off","BlueLed":"off"} 就可以控制相应灯的亮灭。