

```

/*
 * tcpclient.c
 *
 * Notes:
 * 1) W5500 Power issue @ High SPI Clock >25MHz
 * 2) Supply the MCU over Extra 5V USB
 * 3) Some how W5500 is stay in Reset State so add extra delay time in POWER ON RESET
 */

#include <stdio.h>
#include "W5500/w5500.h"
#include "main.h"
#include "socket.h"
#include "tcpclient.h"
#include "wizInterface.h"

#define SERVER_IP1    192 //server ip
#define SERVER_IP2    168
#define SERVER_IP3    1
#define SERVER_IP4    130
#define SERVER_PORT   8079 //server port

extern UART_HandleTypeDef huart3;
uint8_t noNetworkConf =0;

extern void myprintf(const char *fmt, ...);

/* HTTP : g_buffer used in sharing purpose*/
static uint8_t recv_buf[2048];
uint8_t gDATABUF[DATA_BUF_SIZE];
uint16_t wizNetPort =6000;

//TCP client task
int8_t ConfigWizTcpClient() {

    char ack_msg [500];
    memset (ack_msg,0,sizeof (ack_msg));

    if(!WIZ_ChipInit())
    {
        sprintf(ack_msg,"WIZ_ChipInit failed.\n");
        HAL_UART_Transmit(&huart3, (uint8_t *)ack_msg, strlen(ack_msg), 500);
        memset (ack_msg,0,sizeof (ack_msg));
        return false;
    }

    if(!WIZ_NetworkInit())
    {
        sprintf(ack_msg,"WIZ_NetworkInit failed.\n");
        HAL_UART_Transmit(&huart3, (uint8_t *)ack_msg, strlen(ack_msg), 500);
        memset (ack_msg,0,sizeof (ack_msg));
        return false;
    }

    return true;
}

```

```

}

int8_t SnPrevState = -1, SnCrntState;

#if 1
int tcp_client(uint8_t sn, uint8_t *dest_ip, uint16_t dest_port)
{
    int32_t sentLength;
    int8_t ret;
    uint16_t recv_size = 0;
    uint16_t send_size = 0;

    SnCrntState = getSn_SR(sn);
    if(SnCrntState != SnPrevState){
        myprintf(">>> :: SnCrntState - SnPrevState : 0x%02X - 0x%02X :: <<<\r\n", SnCrntState,
SnPrevState);
        SnPrevState = SnCrntState ;
    }

    switch (SnCrntState)
    {
        case SOCK_CLOSED:
        {
            /* open socket */
            myprintf("TCP client start\r\n");
            if ((ret = socket(sn, Sn_MR_TCP, wizNetPort, 0x00)) != sn) {

                myprintf("%d socket %d open fail\r\n", sn, ret);

                if(ret == -3)
                {
                    WIZ_GetNetworkInfo();
                }
                return ret;
            }
            myprintf("socket %d open success\r\n", sn);
            break;
        }

        case SOCK_INIT:
        {
            /* connect server */
            myprintf("try to connect %d.%d.%d.%d:%d...\r\n", dest_ip[0], dest_ip[1], dest_ip[2],
dest_ip[3], dest_port);
            if ((ret = connect(sn, dest_ip, dest_port)) != SOCK_OK) {

                myprintf("ERROR: 0x%02X\r\n", ret);
                return ret;
            }
            break;
        }

        case SOCK_ESTABLISHED:
        {

            /* socket has been established */

```

```

        if(getSn_IR(sn) & Sn_IR_CON) {
            myprintf("%d:Connected to - %d.%d.%d.%d : %d\r\n",sn, dest_ip[0],
dest_ip[1], dest_ip[2], dest_ip[3], dest_port);
            setSn_IR(sn,Sn_IR_CON);
        }

        //get the size of receive data in receive buffer
        if ((recv_size = getSn_RX_RSR(sn)) > 0)
        {

            if (recv_size > DATA_BUF_SIZE) {

                recv_size = DATA_BUF_SIZE;
            }

            //recv data and store into received buffer
            ret = recv(sn, recv_buf, DATA_BUF_SIZE);
            if (ret <= 0) {

                myprintf("%d:recv fail\r\n", sn);
                return ret;
            } else {

                // The actual received size
                recv_size = (uint16_t)ret;
                myprintf("%d:recv size:%d\r\n", sn, recv_size);
                recv_buf[recv_size] = '\0';
                myprintf("%d:recv data:[%s]\r\n", sn, recv_buf);
            }
        }
//
    }

    //send data
    sentLength = send(sn, gDATABUF, strlen((char*)gDATABUF));
    if (sentLength <= 0) {
        myprintf("%d:send fail\r\n", sn);
        //close the socket
        close(sn);
        //return ret;
    } else {

        // The actual sent size
        send_size = (uint16_t)sentLength;
        myprintf("%d:send size:%d\r\n", sn, send_size);
    }
}
break;
}

case SOCK_CLOSE_WAIT:
{
    /* closing the socket */
    if ((ret = disconnect(sn)) != SOCK_OK) {

        myprintf("%d:disconnect fail\r\n", sn);
        return ret;
    }
}

```

```
        myprintf("%d: socket is closed\r\n", sn);
        break;
    }

    default:
    {
        break;
    }
}

return 1;
}
#endif
```