

淘宝店铺

优秀不够，你是否无可替代

知识从未如此性感。烂程序员关心的是代码,好程序员关心的是数据结构和它们之间的关系 --QQ群: 607064330 --本人

QQ:946029359 --淘宝 <https://shop411638453.taobao.com/>

随笔 - 696, 文章 - 0, 评论 - 311, 阅读 - 173万

导航

博客园

首页

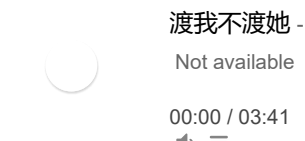
新随笔

联系

订阅 

管理

公告



1 渡我不渡她

2 小镇姑娘

3 PDD洪荒之力

 加入QQ群

昵称：杨奉武

园龄：5年8个月

粉丝：607

关注：1

搜索

我的标签

8266(88)
MQTT(50)
GPRS(33)
SDK(29)
Air202(28)
云服务器(21)
ESP8266(21)
Lua(18)
小程序(17)
STM32(16)
更多

随笔分类

Android(22)
Android 开发(8)
C# 开发(4)
CH395Q学习开发(8)
ESP32学习开发(8)
ESP8266 AT指令开发(基于STC89C52单片机)(3)
ESP8266 AT指令开发(基于STM32)(1)
ESP8266 AT指令开发基础入门篇备份(12)
ESP8266 LUA脚本语言开发(13)

8-网络芯片CH395Q学习开发-模块使用Socket0作为TCP服务器和电脑上位机TCP客户端局域网通信(单连接和多连接)

<p><iframe name="ifd" src="https://mnifdv.cn/resource/cnblogs/LearnCH395Q" frameborder="0" scrolling="auto" width="100%" height="1500"></iframe></p>

网络芯片CH395Q学习开发

开发板链接:[开发板链接](#)

模组原理图:[模组原理图](#)

资料源码下载链

接:<https://github.com/yangfengwu45/CH395Q.c>

■ [学习Android](#)

教程中搭配的Android，C#等教程如上，各个教程正在整理。

■ [1-硬件测试使用说明](#)

■ [2-学习资料说明,测试通信,获取硬件版本,程序移植说明](#)

■ [3-芯片初始化,网线连接检测实验](#)

■ [4-关于中断检测和DHCP实验](#)

■ [5-模块使用Socket0作为TCP客户端和电脑上位机TCP服务器局域网通信](#)

■ [6-模块使用Socket0-3作为4路TCP客户端和电脑上位机TCP服务器局域网通信](#)

■ [7-模块使用Socket0-5作为6路TCP客户端和电脑上位机TCP服务器局域网通信\(Socket缓存区配置\)](#)

■ [8-模块使用Socket0作为TCP服务器和电脑上位机TCP客户端局域网通信\(单连接和多连接\)](#)

■

■

■

ESP8266 LUA开发基础入门篇
备份(22)
ESP8266 SDK开发(32)
ESP8266 SDK开发基础入门篇
备份(30)
GPRS Air202 LUA开发(11)
HC32F460(华大) +
BC260Y(NB-IOT) 物联网开发
(5)
NB-IOT Air302 AT指令和LUA
脚本语言开发(25)
PLC(三菱PLC)基础入门篇(2)
STM32+Air724UG(4G模组)
物联网开发(43)
STM32+BC26/260Y物联网开
发(37)
STM32+ESP8266(ZLESP8266/
物联网开发(1)
STM32+ESP8266+AIR202/30:
远程升级方案(16)
STM32+ESP8266+AIR202/30:
终端管理方案(6)
STM32+ESP8266+Air302物
联网开发(58)
STM32+W5500+AIR202/302
基本控制方案(25)
STM32+W5500+AIR202/302
远程升级方案(6)
UCOSii操作系统(1)
W5500 学习开发(8)
编程语言C#(11)
编程语言Lua脚本语言基础入
门篇(6)
编程语言Python(1)
单片机(LPC1778)LPC1778(2)
单片机(MSP430)开发基础入门
篇(4)
单片机(STC89C51)单片机开发
板学习入门篇(3)
单片机(STM32)基础入门篇(3)
单片机(STM32)综合应用系列
(16)
电路模块使用说明(10)
感想(6)
软件安装使用: MQTT(8)
软件安装使用: OpenResty(6)
数据处理思想和程序架构(24)
数据库学习开发(12)
更多

最新评论

1. Re:C#委托+回调详解
好文，撒也不说了，直接收
藏！
--杨咩咩plus
2. Re:2-STM32 替换说明-
CKS32, HK32, MM32,
APM32, CH32, GD32,
BLM32, AT32(推荐), N32,
HC华大系列
有用，谢谢！
--你跟游戏过吧

阅读排行榜

1. ESP8266使用详解(AT,LUA,
SDK)(172075)
2. 1-安装MQTT服务器(Windo
ws),并连接测试(96481)
3. ESP8266刷AT固件与node
mcu固件(63745)

说明

这节演示一下模块使用Socket0作为TCP服务器和电脑上位机TCP客户端局域网通信.

关于单连接和多连接:

单连接:

模组使用其中一个Socket作为TCP通信,然后启用监听,该Socket就作为了TCP服务器.

但是只能一个客户端进行连接通信(所有版本都支持单连接)

多连接:(版本4及其以上版本支持)

模组使用其中一个Socket 作为TCP通信,然后启用监听,该监听只作为监
听客户端连接,并不做通信,只做客户端的连接和断开监听

其它Socket作为通信.

提醒:无论是SPI,USART,并口,程序操作步骤都是一样的!

- 4. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(62539)
- 5. 有人WIFI模块使用详解(38094)
- 6. (一)基于阿里云的MQTT远程控制(Android 连接MQTT服务器,ESP8266连接MQTT服务器实现远程通信控制----简单的连接通信)(35375)
- 7. 关于TCP和MQTT之间的转换(32225)
- 8. android 之TCP客户端编程(31279)
- 9. android服务端+eps8266+单片机+路由器之远程控制系統(31130)
- 10. C#中public与private与static(30936)

推荐排行榜

- 1. C#委托+回调详解(9)
- 2. 用ESP8266+android,制作自己的WIFI小车(ESP8266篇)(8)
- 3. 用ESP8266+android,制作自己的WIFI小车(Android 软件)(6)
- 4. ESP8266使用详解(AT,LUA,SDK)(6)
- 5. 关于TCP和MQTT之间的转换(5)

只是不同的接口发指令发给模块,然后用不同的接收接收数据而已.

测试本节代码(单连接)

由于单连接和多连接程序差异有点大,所以分开了.

- STM32F10xSPImultiple 2021
- STM32F10xSPISingle 2021

1.用户可以使用杜邦线根据自己的情况设置和连接引脚

```
2  #ifndef CH395SPI_H_
3  #define CH395SPI_H_
4
5  #include "CH395INC.H"
6
7  /*****配置GPIO (根据自己的修改)*****/
8  //时钟
9  #define CH395_CONFIG_SPI_CLK() ( RCC_APB1PeriphClockCmd( RCC_APB1Periph_SPI2,ENABLE) )
10 #define CH395_CONFIG_GPIO_CLK() ( RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA | RCC_APB2Peri
11 //设置使用的SPI
12 #define USE_SPI SPI2
13 //SPI_CS -- 连接模块scs引脚
14 #define CH395_CS_PORT GPIOB
15 #define CH395_CS_PIN GPIO_Pin_12
16 //SPI_CLK -- 连接模块SCK引脚
17 #define CH395_CLK_PORT GPIOB
18 #define CH395_CLK_PIN GPIO_Pin_13
19 //SPI_MISO -- 连接模块SDO引脚
20 #define CH395_MISO_PORT GPIOB
21 #define CH395_MISO_PIN GPIO_Pin_14
22 //SPI_MOSI -- 连接模块SDI引脚
23 #define CH395_MOSI_PORT GPIOB
24 #define CH395_MOSI_PIN GPIO_Pin_15
25 //RST -- 连接模块Rst引脚
26 #define CH395_RST_PORT GPIOA
27 #define CH395_RST_PIN GPIO_Pin_8
28 //TX -- 连接模块Tx引脚
29 #define CH395_TX_PORT GPIOA
30 #define CH395_TX_PIN GPIO_Pin_3
31 //INT -- 连接模块INT引脚 (检测到该引脚低电平信号之后再获取数据)
32 #define CH395_INT_PORT GPIOA
33 #define CH395_INT_PIN GPIO_Pin_0
34 /*****/
```

2,注意!

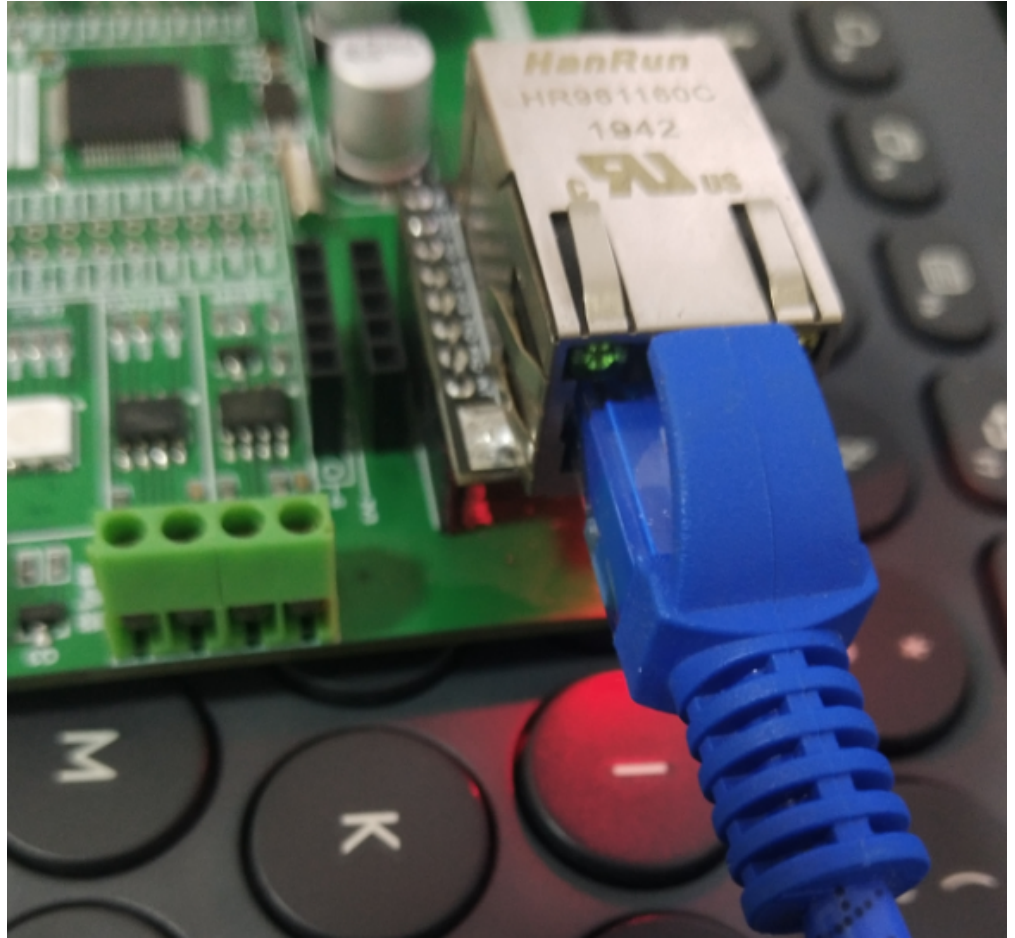
要想模块使用SPI通信,模块的TX引脚需要在模块重启之前设置为低电平.

上面的引脚分配把模块的TX引脚接到了单片机的PA3上,也就是串口2的RX上,如果用户使用了串口2,请注意!

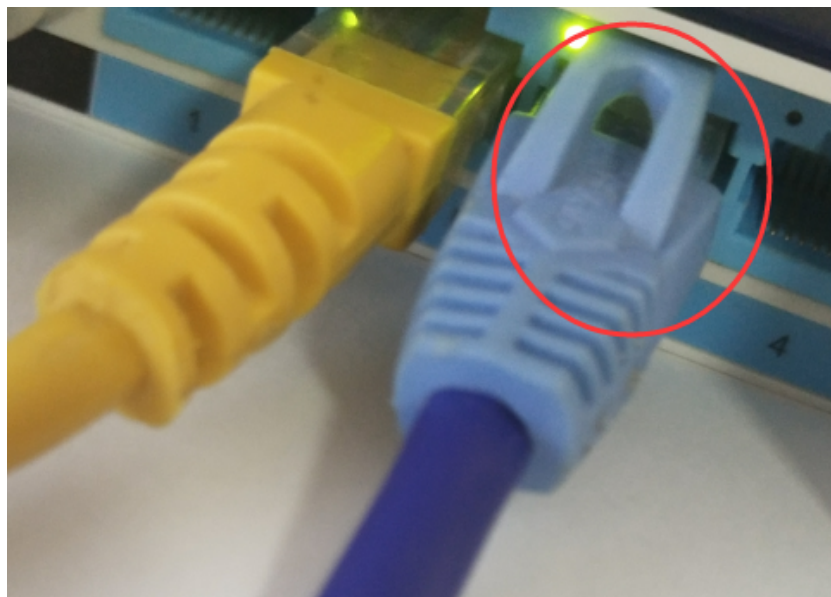
CH395 与单片机之间支持三种通讯接口: 8 位并行接口、SPI 同步串行接口、异步串口。在芯片上电复位时, CH395 将采样 SEL 和 TXD 引脚的状态, 根据这 2 个引脚状态的组合选择通讯接口, 参考下表 (表中 X 代表不关心此位, 0 代表低电平, 1 代表高电平或者悬空)。

SEL 引脚	TXD 引脚	选择通讯接口
1	1	异步串口
1	0	SPI 接口
0	1	8 位并口
0	0	错误接口

3.把模块用网线和路由器或者交换机(和上位机在同一个局域网下)



注意,连接路由器或者交换机的时候是连接其LAN口.





WAN端口：连接网线

LAN端口：连接电脑（任选一个端口就行）

4.程序里面默认监听的端口号为 8080

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
22  #include "CH395SPI.H"
23  #include "CH395INC.H"
24  #include "CH395CMD.H"
25
26
27  /*存储网络接收的数据*/
28  #define recv_buff_len 1500
29  unsigned char recv_buff[recv_buff_len];
30
31  char ch395_version=0;//获取版本号
32
33  unsigned char buf[20];
34  int ch395_status=0;//获取中断事件
35
36  /* socket 相关定义*/
37  UINT16 SocketServerPort = 8080; /*本地监听的 Socket 端口 */
38  char SocketServerStatus = 0;//SocketServer状态 0:未启动监听; 1:启动监听
39
40
41  /**
```

5.下载程序到单片机,查看串口打印的日志

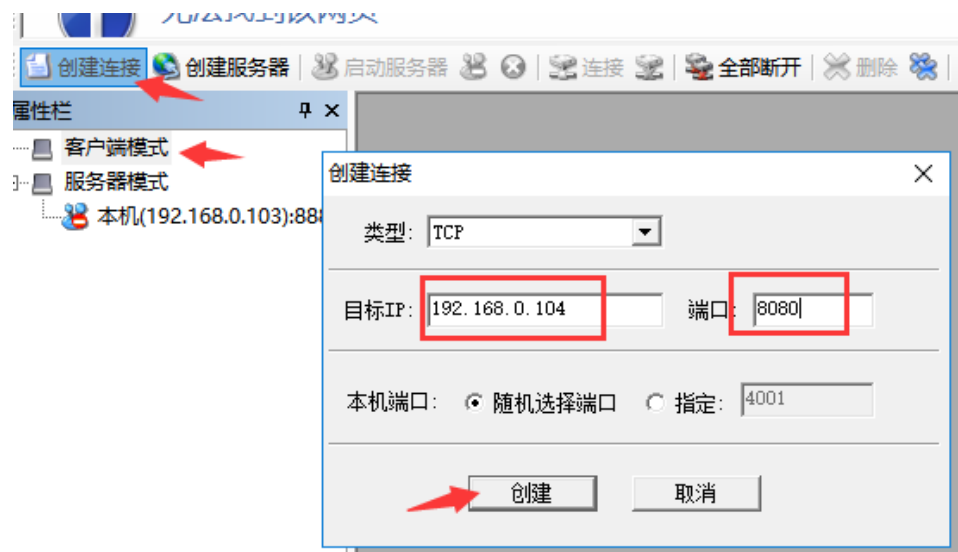
下面打印了模块的IP地址.

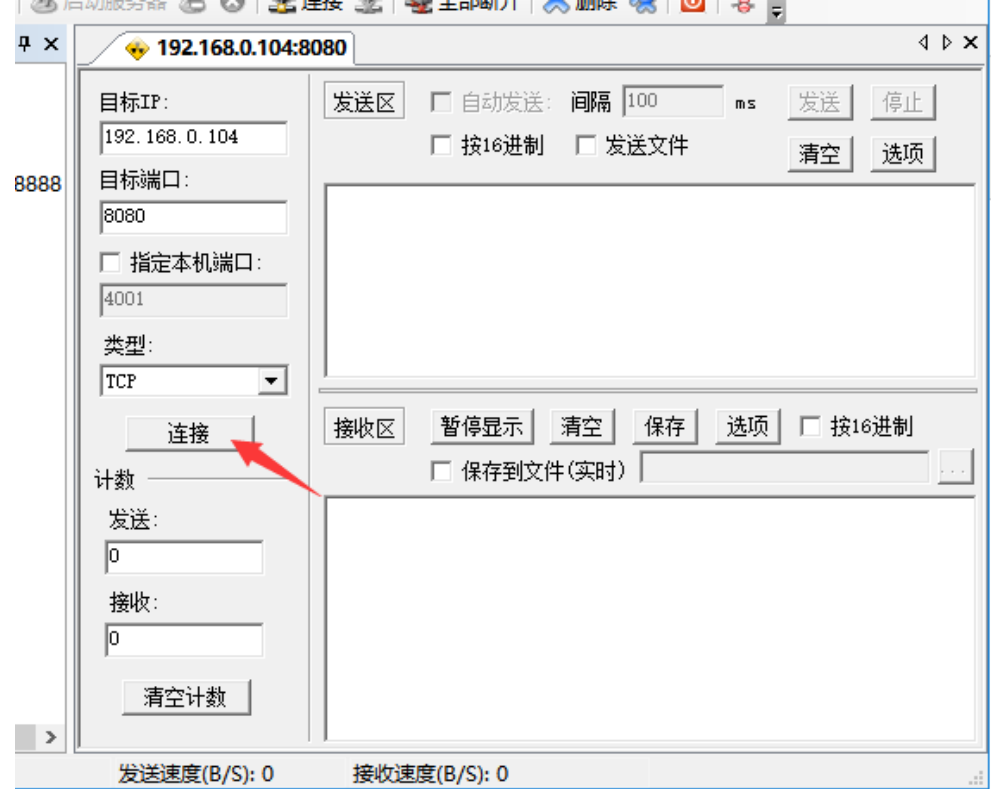
我的模块当前TCP服务器的IP地址为: 192.168.0.104 端口号为: 8080

```
ATK XCOM V2.0
CH395CMDGetVer =46
start
CH395TCPListen
PHY_CONNECTED
IP:192.168.0.104
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
```

6.打开电脑端TCP调试助手,并配置连接

名称	修改日期
COMNET.exe	2021/6
config.ini	2021/6
Log_202103.txt	2021/6
TCPUDPDebug102_Setup.exe	2020/7

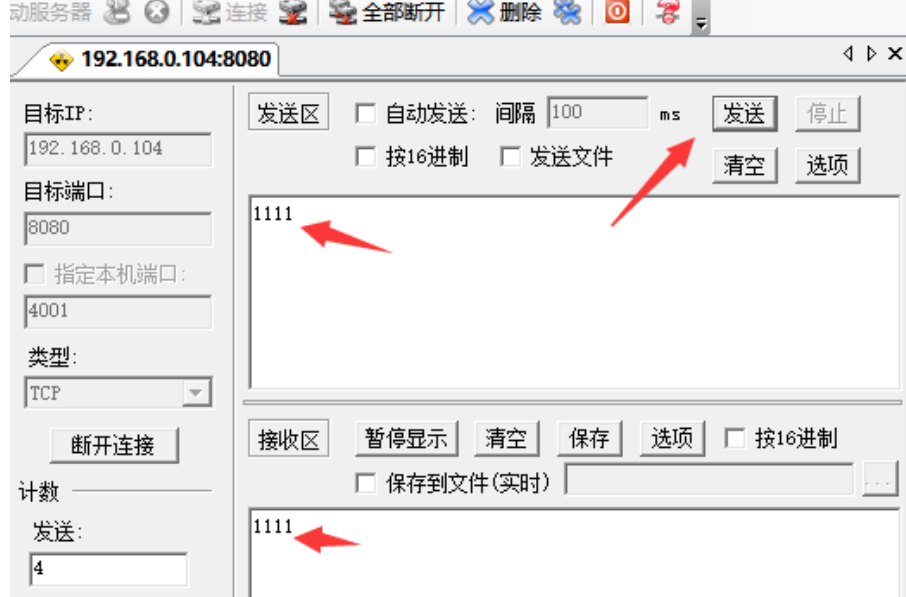




```
ATK XCOM V2.0
CH395CMDGetVer =46
start
CH395TCPListen
PHY_CONNECTED
IP:192.168.0.104
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
socket0 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 47028
```

7.客户端发送数据给服务器

注:服务器默认把接收的数据返回给客户端



```
ATK XCOM V2.0
CH395CMDGetVer =46
start
CH395TCPListen
PHY_CONNECTED
IP:192.168.0.104
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
socket0 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 47028
socket0 receive len = 4
1111
```

程序说明

1.模块连接路由器通信需要启用DHCP,并打印模块分得的地址信息


```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

193
194
195 //INT引脚产生低电平中断以后进去判断
196 if(Query395Interrupt())
197 {
198     /*获取中断事件*/
199     if(ch395_version>=0x44)
200     {
201         ch395_status = CH395CMDGetGlobIntStatus_ALL();
202     }
203     else
204     {
205         ch395_status = CH395CMDGetGlobIntStatus();
206     }
207
208     /* 处理PHY改变中断*/
209     if(ch395_status & GINT_STAT_PHY_CHANGE)
210     {
211         if(CH395CMDGetPHYStatus() == PHY_DISCONN)//网线断开
212         {
213             printf("\r\nPHY_DISCONN\r\n");
214         }
215         else//网线连接
216         {
217             printf("\r\nPHY_CONNECTED\r\n");
218             CH395DHCPEnable(1);//启动DHCP
219         }
220     }
221
222     /* 处理DHCP/PPPOE中断 */
223     if(ch395_status & GINT_STAT_DHCP)
224     {
225         if(CH395GetDHCPStatus() == 0)//DHCP OK
226         {
227             CH395GetIPInf(buf);//获取IP, 子网掩码和网关地址
228             printf("IP:%d.%d.%d.%d\r\n",buf[0],buf[1],buf[2],buf[3]);
229             printf("GWIP:%d.%d.%d.%d\r\n",buf[4],buf[5],buf[6],buf[7]);
230             printf("Mask:%d.%d.%d.%d\r\n",buf[8],buf[9],buf[10],buf[11]);
231             printf("DNS1:%d.%d.%d.%d\r\n",buf[12],buf[13],buf[14],buf[15]);
232             printf("DNS2:%d.%d.%d.%d\r\n",buf[16],buf[17],buf[18],buf[19]);
233         }
234     }
235 }
```

2.初始化配置和启动TCP监听(使用的Socket 0)

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

169 while(CH395CMDInitCH395() != 0)
170 {
171     printf("\r\nCH395CMDInitCH395 ERR\r\n");
172     delay_ms(100);
173 }
174
175 printf("\r\nstart\r\n");
176 while(1)
177 {
178     IWDG_Feed();//喂狗
179
180     /*检测到没有启动服务器,则执行启动服务器*/
181     if(SocketServerStatus == 0)
182     {
183         if(ch395_socket_tcp_server_init(0,SocketServerPort) == 0)
184         {
185             if(CH395TCPListen(0) == 0) //Socket 0 启动TCP监听
186             {
187                 printf("\r\nCH395TCPListen\r\n");
188                 SocketServerStatus = 1;//服务器状态设置为启动监听
189             }
190         }
191     }
192 }
193 }
```

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

31 char ch395_version=0;//获取版本号
32
33 unsigned char buf[20];
34 int ch395_status=0;//获取中断事件
35
36 /* socket 相关定义*/
37 UINT16 SocketServerPort = 8080; /*本地监听的 Socket 端口 */
38 char SocketServerStatus = 0;//SocketServer状态 0:未启动监听; 1:启动监听
39
40
41 /**
42 * @brief 初始化socket
43 * @param sockindex Socket索引(0,1,2,3,4,5,6,7)
44 * @param None
45 * @param None
46 * @param surprot 本地端口号
47 * @retval 0:初始化成功; others:初始化失败
48 * @warning None
49 * @example
50 */
51 char ch395_socket_tcp_server_init(UINT8 sockindex,UINT16 surprot)
52 {
53     CH395SetSocketProtType(sockindex,PROTO_TYPE_TCP); /* 协议类型 */
54     CH395SetSocketSourPort(sockindex,surprot); /* 本地端口号 */
55     if(CH395OpenSocket(sockindex) !=0) /* 打开Socket */
56     {
57         return 1;
58     }
59     return 0;
60 }
61
```

3.在中断检测事件里面处理Socket相关事件

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

235
236
237 /* 处理不可达中断，读取不可达信息 */
238 if(ch395_status & GINT_STAT_UNREACH){
239     CH395CMDGetUnreachIPPT(buf);
240 }
241
242 /* 处理IP冲突中断，建议重新修改CH395的 IP，并初始化CH395*/
243 if(ch395_status & GINT_STAT_IP_CONFLI){
244 }
245
246 /* 处理 SOCK0 中断 */
247 if(ch395_status & GINT_STAT_SOCK0){
248     ch395_socket_tcp_client_interrupt(0);
249 }
250 /* 处理 SOCK1 中断 */
251 if(ch395_status & GINT_STAT_SOCK1){
252 }
253
254 /* 处理 SOCK2 中断 */
255 if(ch395_status & GINT_STAT_SOCK2){
256 }
257
258 /* 处理 SOCK3 中断 */
259 if(ch395_status & GINT_STAT_SOCK3){
260 }
261
262
263
```

```

timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
64  /**
65  * @brief  socket处理函数(把此函数放到全局socket中断里面)
66  * @param  sockindex  Socket索引(0,1,2,3,4,5,6,7)
67  * @param  None
68  * @param  None
69  * @param  None
70  * @retval None
71  * @warning None
72  * @example
73  */
74  void ch395_socket_tcp_client_interrupt(UINT8 sockindex)
75  {
76      UINT8 sock_int_socket;
77      UINT16 len;
78
79      /* 获取socket 的中断状态 */
80      sock_int_socket = CH395GetSocketInt(sockindex);
81
82      /* 发送缓冲区空闲,可以继续写入要发送的数据 */
83      if(sock_int_socket & SINT_STAT_SENBUF_FREE)
84      {
85      }
86
87      /* 发送完成中断 */
88      if(sock_int_socket & SINT_STAT_SEND_OK)
89      {
90      }
91
92      /* 接收数据中断 */
93      if(sock_int_socket & SINT_STAT_RECV)
94      {
95          len = CH395GetRecvLength(sockindex);/* 获取当前缓冲区内数据长度 */
96          printf("\r\nsocket%d receive len = %d\r\n",sockindex,len);
97          if(len == 0)return;
98          if(len > recv_buff_len)len = recv_buff_len;
99          CH395GetRecvData(sockindex,len,recv_buff);/* 读取数据 */
100
101          /*把接收的数据发送给TCP客户端*/
102          CH395SendData(sockindex,recv_buff,len);

```

```

100
101  /*把接收的数据发送给TCP客户端*/
102  CH395SendData(sockindex,recv_buff,len);
103
104  /*使用串口打印接收的数据*/
105  PutData(&rb_t_usart1_send,recv_buff,len);
106  USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
107  }
108
109  /* 连接中断, 仅在TCP模式下有效 */
110  if(sock_int_socket & SINT_STAT_CONNECT)
111  {
112      printf("socket%d SINT_STAT_CONNECT\r\n",sockindex);
113
114      CH395CMDGetRemoteIPP(sockindex,buf);/*获取连接的tcp客户端的信息
115
116      printf("IP address = %d.%d.%d.%d\n", (UINT16)buf[0], (UINT16)buf[1], (UINT16)buf[2], (UINT16)buf[3]);
117      printf("Port = %d\n", ((buf[5]<<8) + buf[4]));
118  }
119
120  /* 断开中断, 仅在TCP模式下有效 */
121  if(sock_int_socket & SINT_STAT_DISCONNECT)
122  {
123      printf("socket%d SINT_STAT_DISCONNECT \r\n",sockindex);
124
125      SocketServerStatus = 0;//服务器状态设置为未启动监听
126  }
127
128  /* 超时中断, 仅在TCP模式下有效 ,TCP CLIENT无法顺利连接服务器端会进入此中断*/
129  if(sock_int_socket & SINT_STAT_TIM_OUT)
130  { /*此时可以把Socket源端口号进行自加处理, 以新的端口去连接服务器*/
131      printf("socket%d SINT_STAT_TIM_OUT\n",sockindex);
132      SocketServerStatus = 0;//服务器状态设置为未启动监听
133  }
134  }
135

```

4.注意事项

在单连接模式下客户端Socket连接,然后断开后需要重新配置并打开Socket监听.

所以在断开和超时事件里面清零监听状态,以让程序重新配置并打开Socket监听

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
107     }
108
109     /* 连接中断, 仅在TCP模式下有效*/
110     if(sock_int_socket & SINT_STAT_CONNECT)
111     {
112         printf("socket%d SINT_STAT_CONNECT\r\n", sockindex);
113
114         CH395CMDGetRemoteIPP(sockindex, buf); //获取连接的TCP客户端的信息
115
116         printf("IP address = %d.%d.%d.%d\n", (UINT16)buf[0], (UINT16)buf[1], (UINT16)buf[2], (UINT16)buf[3]);
117         printf("Port = %d\n", ((buf[4]<<8) + buf[5]));
118     }
119
120     /* 断开中断, 仅在TCP模式下有效 */
121     if(sock_int_socket & SINT_STAT_DISCONNECT)
122     {
123         printf("socket%d SINT_STAT_DISCONNECT \r\n", sockindex);
124
125         SocketServerStatus = 0; //服务器状态设置为未启动监听 ←
126     }
127
128     /* 超时中断, 仅在TCP模式下有效, TCP CLIENT无法顺利连接服务器端会进入此中断*/
129     if(sock_int_socket & SINT_STAT_TIM_OUT)
130     {
131         /*此时可以把Socket源端口号进行自加处理, 以新的端口去连接服务器*/
132         printf("socket%d SINT_STAT_TIM_OUT\n", sockindex);
133         SocketServerStatus = 0; //服务器状态设置为未启动监听 ←
134     }
135 }
```

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  ti
170     printf("\r\nCH395CMDInitCH395 ERR\r\n");
171     delay_ms(100);
172 }
173
174 printf("\r\nstart\r\n");
175 while(1)
176 {
177     IWDG_Feed(); //喂狗
178
179     /*检测到没有启动服务器,则执行启动服务器*/
180     if(SocketServerStatus == 0) ←
181     {
182         if(ch395_socket_tcp_server_init(0, SocketServerPort) == 0)
183         {
184             if(CH395TCPListen(0) == 0) //Socket 0 启动TCP监听
185             {
186                 printf("\r\nCH395TCPListen\r\n");
187                 SocketServerStatus = 1; //服务器状态设置为启动监听
188             }
189         }
190     }
191 }
192 }
```

其它注意的就是如果使用Socket4或5或6或7作为TCP服务器监听

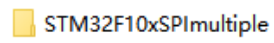

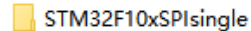
记得重新配置下Socket内存分配(参考上一节)

测试本节代码(多连接,仅4,及其以上版本才支持)

1.提示

CH395 共有 8 个独立的 Socket 通道，TCP SERVER 多连接模式下，首先需要先创建一个监听连接，然后按照与监听连接相同端口号创建数据连接，且数据连接至少创建一个，具体数据连接创建个数根据 TCP SERVER 实际支持的客户端连接个数而定，假定需要支持 N 个客户端，则需要创建的数据连接个数为 N。比如创建一个 TCPSERVER，支持 4 个 TCP CLIENT，需要创建 1 个监听连接，4 个数据连接，

2,打开这节的程序

3,这节的程序配置Socket 0 作为了监听,然后其它Socket作为数据连接

用户根据自己的习惯设置服务器监听的端口号

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
26 芯片共有48块缓存区,每个缓存区512字节
27 芯片共有8个Socket,默认把48块缓存区分给了Socket0,Socket1,Socket2,Socket3
28 这四个Socket,每个 Socket 使用8块缓存区作为接收,4块缓存区作为发送,
29 即Socket0,Socket1,Socket2,Socket3的接收区各为512*8 = 4KB
30 即Socket0,Socket1,Socket2,Socket3的发送区各为512*4 = 2KB
31 如果要使用Socket4,Socket5,Socket6,Socket7需要重新分配缓存区
32 */
33
34
35 /*存储网络接收的数据*/
36 #define recv_buff_len 1500
37 unsigned char recv_buff[recv_buff_len];
38
39 char ch395_version=0;//获取版本号
40
41 unsigned char buf[20];
42 int ch395_status=0;//获取中断事件
43
44 /* socket 相关定义*/
45 UINT16 SocketServerPort = 8080; /* Socket 目的端口 */
46 char SocketServerStatus = 0;//SocketServer状态 0:未启动监听; 1:启动监听
47
48
49 /*配置 Socket 发送和接收缓存区*/
50 /*芯片有0-47个块,每个块512字节大小*/
51 void socket_buffer_config(void )
--
```

4.把程序下载到单片机

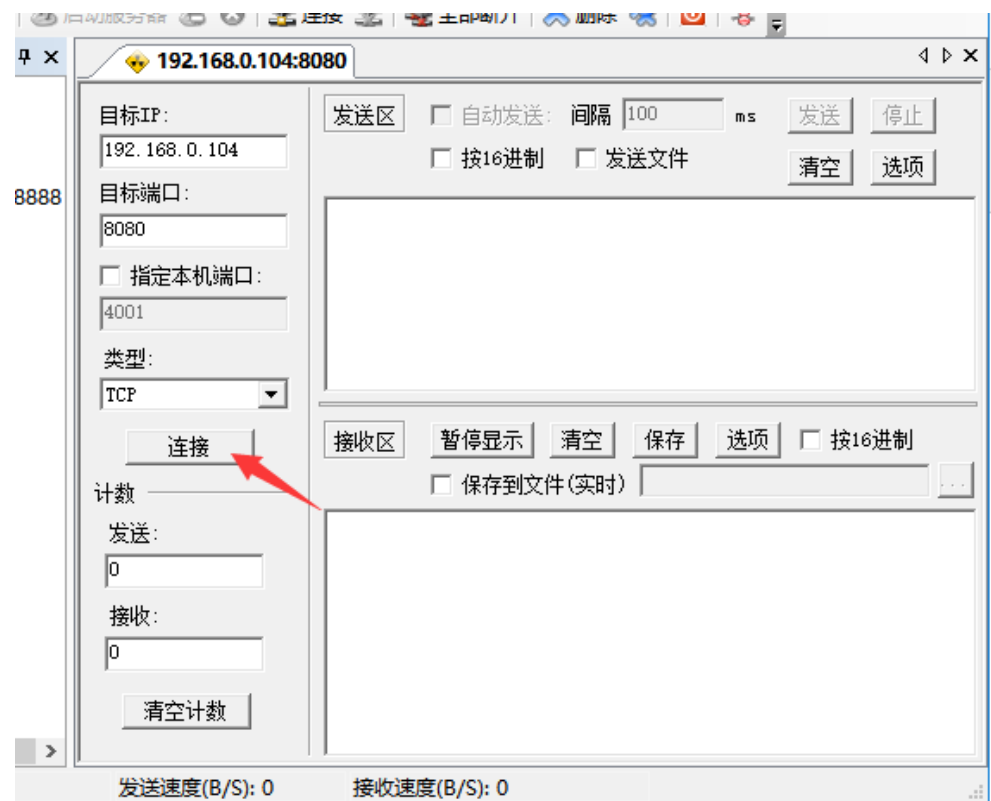
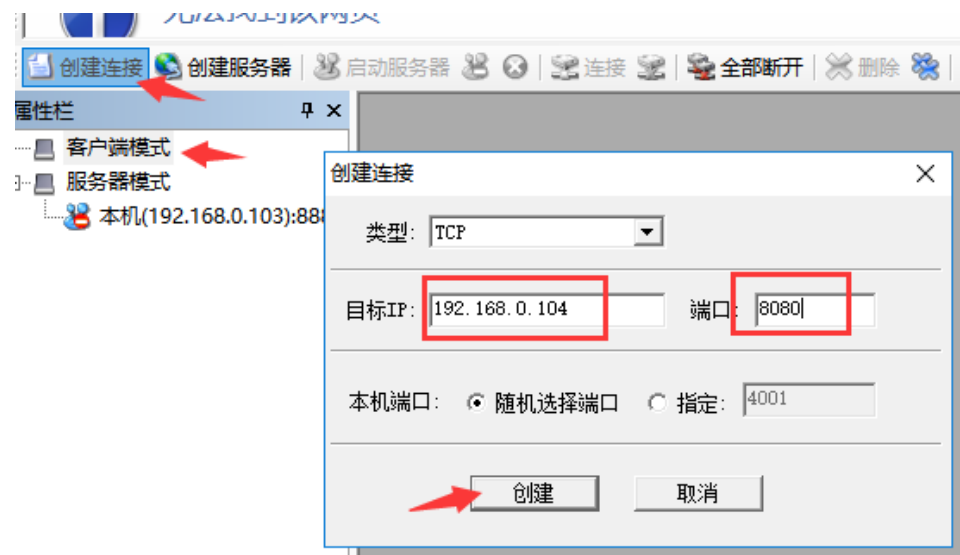
正常运行会打印模块分配的IP地址信息

此时模块作为TCP服务器的IP地址为:192.168.0.104 端口号为:8080

```
AT&#x26; XCOM V2.0
CH395CMDGetVer =46
start
CH395TCPListen
PHY_CONNECTED
IP:192.168.0.104
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
```

5.打开电脑端TCP调试助手,并配置连接

名称	修改日期
COMNET.exe	2021/6
config.ini	2021/6
Log_202103.txt	2021/6
TCPUDPDebug102_Setup.exe	2020/7




```
ATK
COM XCOM V2.0

CH395CMDGetVer =46

start

CH395TCPListen

PHV_CONNECTED
IP:192.168.0.104
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
socket1 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 48997
```

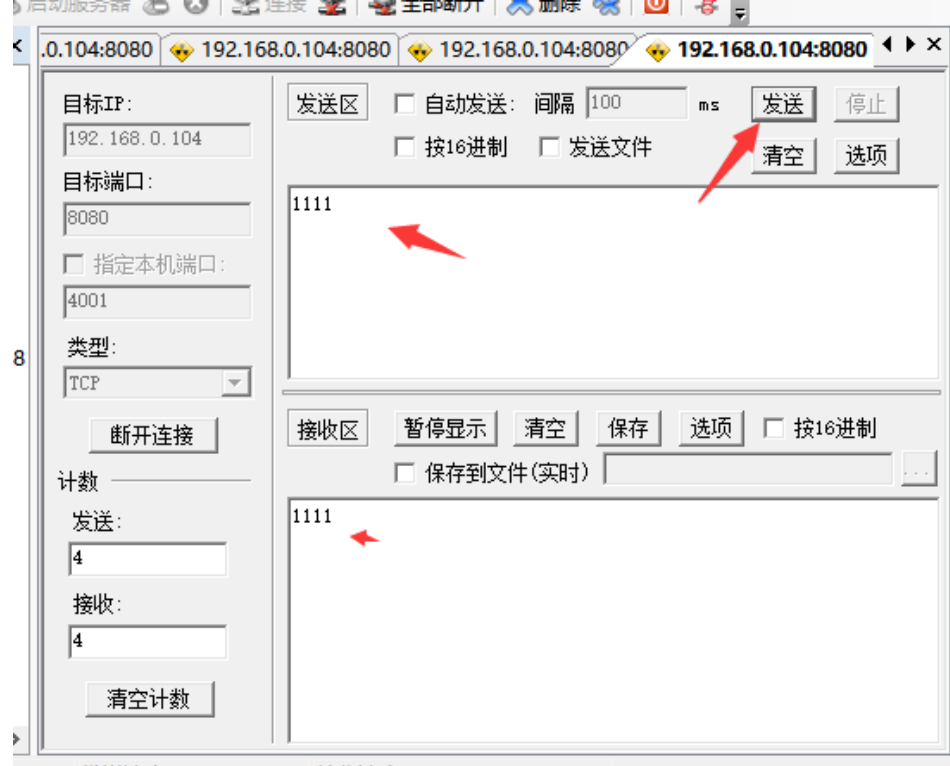
6.按照同样的步骤还可以创建其它的6个客户端连接服务器



```
GWIP:192.168.0.1
Mask:255.255.255.0
DNS1:192.168.1.1
DNS2:192.168.0.1
socket1 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 48997
socket2 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49040
socket3 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49041
socket4 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49046
socket5 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49047
socket6 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49050
```

7,使用客户端发送数据给服务器

注:服务器默认把接收的数据返回给客户端



```
DNS2:192.168.0.1
socket1 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 48997
socket2 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49040
socket3 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49041
socket4 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49046
socket5 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49047
socket6 SINT_STAT_CONNECT
IP address = 192.168.0.103Port = 49050

socket6 receive len = 4
1111
```

A red arrow points to the '1111' data received by socket6.

程序说明

1.版本大于等于0x44才允许运行; 执行多链接函数,配置
Socket 缓存区域分配

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C
218
219     IWDG_Init(IWDG_Prescaler_256,156*10);
220
221     /*获取芯片版本*/
222     while((ch395_version = CH395CMDGetVer()) < 0x44)
223     {
224         printf("CH395CMDVer ERR, no support!\r\n");
225         delay_ms(100);
226     }
227     printf("CH395CMDGetVer =%2x\r\n",ch395_version);
228
229     /*测试命令，按位取反返回说明测试通过*/
230     while(CH395CMDCheckExist(0x55) != 0xaa)
231     {
232         printf("\r\nCH395CMDCheck ERR\r\n");
233         delay_ms(100);
234     }
235
236     /*TCP SERVER支持多连接时，需初始化此启动参数*/
237     CH395SetStartPara(FUN_PARA_FLAG_TCP_SERVER);
238     delay_ms(100);
239
240     /*配置 Socket 发送和接收缓存区大小
241     socket_buffer_config();
242     delay_ms(100);
243
244     /*初始化模块:成功返回 0 */
245     while(CH395CMDInitCH395() != 0)
246     {
247         printf("\r\nCH395CMDInitCH395 ERR\r\n");
248         delay_ms(100);
249     }
250
251     printf("\r\nstart\r\n");
252     while(1)
253     {
```

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
47
48
49     /*配置 Socket 发送和接收缓存区*/
50     /*芯片有0-47个块,每个块512字节大小*/
51     void socket_buffer_config(void)
52     {
53         /*Socket 0*/
54         CH395SetSocketRecvBuf(0,0,4);//第0,1,2,3块缓存区作为Socket的接收缓存区(512*4=2KB)
55         CH395SetSocketSendBuf(0,4,2);//第4,5块缓存区作为Socket的发送缓存区(512*2=1KB)
56         /*Socket 1*/
57         CH395SetSocketRecvBuf(1,6,4);//第6,7,8,9块缓存区作为Socket的接收缓存区(512*4=2KB)
58         CH395SetSocketSendBuf(1,10,2);//第10,11块缓存区作为Socket的发送缓存区(512*2=1KB)
59         /*Socket 2*/
60         CH395SetSocketRecvBuf(2,12,4);//第12,13,14,15块缓存区作为Socket的接收缓存区(512*4=2KB)
61         CH395SetSocketSendBuf(2,16,2);//第16,17块缓存区作为Socket的发送缓存区(512*2=1KB)
62         /*Socket 3*/
63         CH395SetSocketRecvBuf(3,18,4);//第18,19,20,21块缓存区作为Socket的接收缓存区(512*4=2KB)
64         CH395SetSocketSendBuf(3,22,2);//第22,23块缓存区作为Socket的发送缓存区(512*2=1KB)
65
66         /*硬件版本大于4的才有Socket4-7*/
67         // if(ch395_version>=0x44)
68         {
69             /*Socket 4*/
70             CH395SetSocketRecvBuf(4,24,4);//第24,25,26,27块缓存区作为Socket的接收缓存区(512*4=2KB)
71             CH395SetSocketSendBuf(4,28,2);//第28,29块缓存区作为Socket的发送缓存区(512*2=1KB)
72             /*Socket 5*/
73             CH395SetSocketRecvBuf(5,30,4);//第30,31,32,33块缓存区作为Socket的接收缓存区(512*4=2KB)
74             CH395SetSocketSendBuf(5,34,2);//第34,35块缓存区作为Socket的发送缓存区(512*2=1KB)
75
76             /*Socket 6*/
77             CH395SetSocketRecvBuf(6,36,4);//第36,37,38,39块缓存区作为Socket的接收缓存区(512*4=2KB)
78             CH395SetSocketSendBuf(6,40,2);//第40,41块缓存区作为Socket的发送缓存区(512*2=1KB)
79
80             /*Socket 7*/
81             CH395SetSocketRecvBuf(7,42,4);//第42,43,44,45块缓存区作为Socket的接收缓存区(512*4=2KB)
82             CH395SetSocketSendBuf(7,46,2);//第46,47块缓存区作为Socket的发送缓存区(512*2=1KB)
83         }
84     }
85
```

2.启用DHCP,打印模块获取的IP地址信息

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

269
270 //INT引脚产生低电平中断以后进去判断
271 if(Query395Interrupt())
272 {
273     /*获取中断事件*/
274     // if(ch395_version>=0x44)
275     // {
276     //     ch395_status = CH395CMDGetGlobIntStatus_ALL();
277     // }
278     // else
279     // {
280     //     ch395_status = CH395CMDGetGlobIntStatus();
281     // }
282
283     /* 处理PHY改变中断*/
284     if(ch395_status & GINT_STAT_PHY_CHANGE)
285     {
286         if(CH395CMDGetPHYStatus() == PHY_DISCONN)//网线断开
287         {
288             printf("\r\nPHY_DISCONN\r\n");
289         }
290         else//网线连接
291         {
292             printf("\r\nPHY_CONNECTED\r\n");
293             CH395DHCPEnable(1);//启动DHCP
294         }
295     }
296
297     /* 处理DHCP/PPPOE中断 */
298     if(ch395_status & GINT_STAT_DHCP)
299     {
300         if(CH395GetDHCPStatus() == 0)//DHCP OK
301         {
302             CH395GetIPInf(buf);//获取IP, 子网掩码和网关地址
303             printf("IP:%d.%d.%d.%d\r\n",buf[0],buf[1],buf[2],buf[3]);
304             printf("GWIP:%d.%d.%d.%d\r\n",buf[4],buf[5],buf[6],buf[7]);
305             printf("Mask:%d.%d.%d.%d\r\n",buf[8],buf[9],buf[10],buf[11]);
306             printf("DNS1:%d.%d.%d.%d\r\n",buf[12],buf[13],buf[14],buf[15]);
307             printf("DNS2:%d.%d.%d.%d\r\n",buf[16],buf[17],buf[18],buf[19]);
308         }
309     }
310 }
```

3,初始化配置和启动监听

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h

251 printf("\r\nstart\r\n");
252 while(1)
253 {
254     IWDG_Feed();//喂狗
255
256     /*配置模块启动Socket监听 (多链接只执行一次就可以)*/
257     if(SocketServerStatus == 0)
258     {
259         if(ch395_socket_tcp_server_init() == 0)
260         {
261             if(CH395TCPListen(0) == 0) //Socket 0 启动TCP监听
262             {
263                 printf("\r\nCH395TCPListen\r\n");
264                 SocketServerStatus = 1;
265             }
266         }
267     }
268 }
```

```

timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
92  * @param None
93  * @retval 0:初始化成功; others:初始化失败
94  * @warning None
95  * @example
96  **/
97  char ch395_socket_tcp_server_init(void)
98  {
99      /*让Socket作为监听连接*/
100      CH395SetSocketProtType(0,PROTO_TYPE_TCP); /* 协议类型 */
101      CH395SetSocketSourPort(0,SocketServerPort); /* 本地端口号 */
102      if(CH395OpenSocket(0) !=0) /* 打开Socket */
103      {
104          return 1;
105      }
106
107      /*其它Socket作为数据通信*/
108      /*想要几路Socket客户端连接通信,就需要配置几个Socket,所以模块最多支持7个Socket TCP客户端通信*/
109      CH395SetSocketProtType(1,PROTO_TYPE_TCP); /* 协议类型 */
110      CH395SetSocketSourPort(1,SocketServerPort);/* 本地端口号 */
111
112      CH395SetSocketProtType(2,PROTO_TYPE_TCP); /* 协议类型 */
113      CH395SetSocketSourPort(2,SocketServerPort);/* 本地端口号 */
114
115      CH395SetSocketProtType(3,PROTO_TYPE_TCP); /* 协议类型 */
116      CH395SetSocketSourPort(3,SocketServerPort);/* 本地端口号 */
117
118      CH395SetSocketProtType(4,PROTO_TYPE_TCP); /* 协议类型 */
119      CH395SetSocketSourPort(4,SocketServerPort);/* 本地端口号 */
120
121      CH395SetSocketProtType(5,PROTO_TYPE_TCP); /* 协议类型 */
122      CH395SetSocketSourPort(5,SocketServerPort);/* 本地端口号 */
123
124      CH395SetSocketProtType(6,PROTO_TYPE_TCP); /* 协议类型 */
125      CH395SetSocketSourPort(6,SocketServerPort);/* 本地端口号 */
126
127      CH395SetSocketProtType(7,PROTO_TYPE_TCP); /* 协议类型 */
128      CH395SetSocketSourPort(7,SocketServerPort);/* 本地端口号 */
129
130      return 0;
131  }
132
133

```

4.在不同的Socket中断事件里面执行中断执行函数

```

timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
317      /* 处理IP冲突中断, 建议重新修改CH395的 IP, 并初始化CH395*/
318      if(ch395_status & GINT_STAT_IP_CONFLI){
319
320      }
321
322      /* 处理 SOCK0 中断 */
323      if(ch395_status & GINT_STAT_SOCK0){
324          ch395_socket_tcp_client_interrupt(0);
325      }
326      /* 处理 SOCK1 中断 */
327      if(ch395_status & GINT_STAT_SOCK1){
328          ch395_socket_tcp_client_interrupt(1);
329      }
330      /* 处理 SOCK2 中断 */
331      if(ch395_status & GINT_STAT_SOCK2){
332          ch395_socket_tcp_client_interrupt(2);
333      }
334      /* 处理 SOCK3 中断 */
335      if(ch395_status & GINT_STAT_SOCK3){
336          ch395_socket_tcp_client_interrupt(3);
337      }
338
339      //      if(ch395_version>=0x44)
340      {
341          /* 处理 SOCK4 中断 */
342          if(ch395_status & GINT_STAT_SOCK4){
343              ch395_socket_tcp_client_interrupt(4);
344          }
345          /* 处理 SOCK5 中断 */
346          if(ch395_status & GINT_STAT_SOCK5){
347              ch395_socket_tcp_client_interrupt(5);
348          }
349          /* 处理 SOCK6 中断 */
350          if(ch395_status & GINT_STAT_SOCK6){
351              ch395_socket_tcp_client_interrupt(6);
352          }
353          /* 处理 SOCK7 中断 */
354          if(ch395_status & GINT_STAT_SOCK7){
355              ch395_socket_tcp_client_interrupt(7);
356          }
357      }
358

```

```

timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
134
135 /**
136  * @brief  socket处理函数 (把此函数放到全局socket中断里面)
137  * @param  sockindex  Socket索引 (0,1,2,3,4,5,6,7)
138  * @param  None
139  * @param  None
140  * @param  None
141  * @retval None
142  * @warning None
143  * @example
144  */
145 void ch395_socket_tcp_client_interrupt(UINT8 sockindex)
146 {
147     UINT8 sock_int_socket;
148     UINT16 len;
149
150     /* 获取socket 的中断状态 */
151     sock_int_socket = CH395GetSocketInt(sockindex);
152
153     /* 发送缓冲区空闲, 可以继续写入要发送的数据 */
154     if(sock_int_socket & SINT_STAT_SENBUF_FREE)
155     {
156     }
157
158     /* 发送完成中断 */
159     if(sock_int_socket & SINT_STAT_SEND_OK)
160     {
161     }
162
163     /* 接收数据中断 */
164     if(sock_int_socket & SINT_STAT_RECV)
165     {
166         len = CH395GetRecvLength(sockindex); /* 获取当前缓冲区内数据长度 */
167         printf("\r\nsocket%d receive len = %d\r\n", sockindex, len);
168         if(len == 0) return;
169         if(len > recv_buff_len) len = recv_buff_len;
170         CH395GetRecvData(sockindex, len, recv_buff); /* 读取数据 */
171
172         /*把接收的数据发送给服务器*/
173         CH395SendData(sockindex, recv_buff, len);
174
175         /*使用串口打印接收的数据*/
176         PutData(&rb_t_usart1_send, recv_buff, len);
177         USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
178     }
179
180     /* 连接中断, 仅在tcp模式下有效 */
181     if(sock_int_socket & SINT_STAT_CONNECT)
182     {
183         printf("socket%d SINT_STAT_CONNECT\r\n", sockindex);
184
185         CH395CMDGetRemoteIPP(sockindex, buf); /*获取客户端信息
186
187         printf("IP address = %d.%d.%d.%d\n", (UINT16)buf[0], (UINT16)buf[1], (UINT16)buf[2], (UINT16)buf[3]);
188         printf("Port = %d\n", ((buf[5]<<8) + buf[4]));
189     }
190
191     /* 断开中断, 仅在tcp模式下有效 */
192     if(sock_int_socket & SINT_STAT_DISCONNECT)
193     {
194         printf("socket%d SINT_STAT_DISCONNECT \r\n", sockindex);
195     }
196
197     /* 超时中断, 仅在tcp模式下有效 */
198     if(sock_int_socket & SINT_STAT_TIM_OUT)
199     {
200         printf("socket%d SINT_STAT_TIM_OUT\n", sockindex);
201     }
202 }
203

```

5.提示

Socket0-7都是使用的void

ch395_socket_tcp_client_interrupt(UINT8 sockindex)

但是因为Socket0作为监听,所以连接和断开都会进入的Socket0

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  ti
314      CH395CMDGetUnreachIPPT(buf);
315      }
316
317      /* 处理IP冲突中断, 建议重新修改CH395的 IP, 并初始化CH395 */
318      if(ch395_status & GINT_STAT_IP_CONFLI){
319
320      }
321      /* 处理 SOCK0 中断 */
322      if(ch395_status & GINT_STAT_SOCK0){
323          ch395_socket_tcp_client_interrupt(0);
324      }
325      /* 处理 SOCK1 中断 */
326      if(ch395_status & GINT_STAT_SOCK1){
327          ch395_socket_tcp_client_interrupt(1);
328      }
329      /* 处理 SOCK2 中断 */
330      if(ch395_status & GINT_STAT_SOCK2){
```

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
176      PutData(&rb_t_usart1_send,recv_buff,len);
177      USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
178  }
179
180  /* 连接中断, 仅在TCP模式下有效 */
181  if(sock_int_socket & SINT_STAT_CONNECT)
182  {
183      printf("socket%d SINT_STAT_CONNECT\r\n",sockindex);
184
185      CH395CMDGetRemoteIPP(sockindex,buf); //获取客户端信息
186
187      printf("IP address = %d.%d.%d.%d\n", (UINT16)buf[0], (UINT16)buf[1], (UINT16)buf[2], (UINT16)buf[3]);
188      printf("Port = %d\n", ((buf[5]<<8) + buf[4]));
189  }
190
191  /* 断开中断, 仅在TCP模式下有效 */
192  if(sock_int_socket & SINT_STAT_DISCONNECT)
193  {
194      printf("socket%d SINT_STAT_DISCONNECT \r\n",sockindex);
195  }
196
197  /* 超时中断, 仅在TCP模式下有效 */
198  if(sock_int_socket & SINT_STAT_TIM_OUT)
199  {
200      printf("socket%d SINT_STAT_TIM_OUT\n",sockindex);
201  }
202 }
```

Socket1-7作为数据通信,数据通信会进去他们的中断

timer.c usart.c main.c CH395CMD.H CH395INC.H CH395CMD.C timer.h

```
320 }
321 /* 处理 SOCK0 中断 */
322 if(ch395_status & GINT_STAT_SOCK0){
323     ch395_socket_tcp_client_interrupt(0);
324 }
325 /* 处理 SOCK1 中断 */
326 if(ch395_status & GINT_STAT_SOCK1){
327     ch395_socket_tcp_client_interrupt(1);
328 }
329 /* 处理 SOCK2 中断 */
330 if(ch395_status & GINT_STAT_SOCK2){
331     ch395_socket_tcp_client_interrupt(2);
332 }
333 /* 处理 SOCK3 中断 */
334 if(ch395_status & GINT_STAT_SOCK3){
335     ch395_socket_tcp_client_interrupt(3);
336 }
337
338
339 //
340 if(ch395_version>=0x44)
341 {
342     /* 处理 SOCK4 中断 */
343     if(ch395_status & GINT_STAT_SOCK4){
344         ch395_socket_tcp_client_interrupt(4);
345     }
346     /* 处理 SOCK5 中断 */
347     if(ch395_status & GINT_STAT_SOCK5){
348         ch395_socket_tcp_client_interrupt(5);
349     }
350     /* 处理 SOCK6 中断 */
351     if(ch395_status & GINT_STAT_SOCK6){
352         ch395_socket_tcp_client_interrupt(6);
353     }
354     /* 处理 SOCK7 中断 */
355     if(ch395_status & GINT_STAT_SOCK7){
356         ch395_socket_tcp_client_interrupt(7);
357     }
358 }
359 }
```

```
timer.c  usart.c  main.c  CH395CMD.H  CH395INC.H  CH395CMD.C  timer.h
143  * @example
144  **/
145  void ch395_socket_tcp_client_interrupt(UINT8 sockindex)
146  {
147      UINT8 sock_int_socket;
148      UINT16 len;
149
150      /* 获取socket 的中断状态 */
151      sock_int_socket = CH395GetSocketInt(sockindex);
152
153      /* 发送缓冲区空闲,可以继续写入要发送的数据 */
154      if(sock_int_socket & SINT_STAT_SENBUF_FREE)
155      {
156      }
157
158      /* 发送完成中断 */
159      if(sock_int_socket & SINT_STAT_SEND_OK)
160      {
161      }
162
163      /* 接收数据中断 */
164      if(sock_int_socket & SINT_STAT_RECV)
165      {
166          len = CH395GetRecvLength(sockindex);/* 获取当前缓冲区内数据长度 */
167          printf("\r\nsocket%d receive len = %d\r\n",sockindex,len);
168          if(len == 0)return;
169          if(len > recv_buff_len)len = recv_buff_len;
170          CH395GetRecvData(sockindex,len,recv_buff);/* 读取数据 */
171
172          /*把接收的数据发送给服务器*/
173          CH395SendData(sockindex,recv_buff,len);
174
175          /*使用串口打印接收的数据*/
176          PutData(&rb_t_usart1_send,recv_buff,len);
177          USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
178      }
179
180      /* 连接中断,仅在TCP模式下有效*/
181      if(sock_int_socket & SINT_STAT_CONNECT)
```

分类: CH395Q学习开发

好文要顶

关注我

收藏该文



杨奉武

关注 - 1

粉丝 - 607

0

0

« 上一篇: 7-网络芯片CH395Q学习开发-模块使用Socket0-5作为6路TCP客户端和电脑上位机TCP服务器局域网通信(Socket缓存区配置)

posted on 2021-06-12 16:53 杨奉武 阅读(0) 评论(0) 编辑 收藏 举报

刷新评论 刷新页面 返回顶部

发表评论

编辑 预览

B

支持 Markdown

自动补全

[Ctrl+Enter快捷键提交]

- 【推荐】百度智能云618年中大促，限时抢购，新老用户同享超值折扣
- 【推荐】大型组态、工控、仿真、CAD\GIS 50万行VC++源码免费下载!
- 【推荐】阿里云爆品销量榜单出炉，精选爆款产品低至0.55折
- 【推荐】限时秒杀！国云大数据魔镜，企业级云分析平台
- 【推荐】华为应用软件专题日 | 生态市场企业特惠GO

园子动态：

- 致园友们的一封检讨书：都是我们的错
- 数据库实例 CPU 100% 引发全站故障
- 发起一个开源项目：博客引擎 fluss

最新新闻：

- 欧洲杯正式开幕 vivo成首个开闭幕式冠名合作伙伴
 - 美媒：苹果向美国政府提供大量用户隐私数据信息
 - 马斯克：Model S Plaid比保时捷更快，但比沃尔沃更安全
 - 毛利不高 但制造业让国家工业更安全！王传福：比亚迪24天成全球最大口罩厂
 - 在线教育员工：不用公司开人，我也要离职了
- » 更多新闻...

历史上的今天：

- 2021-06-12 6-网络芯片CH395Q学习开发-模块使用Socket0-3作为4路TCP客户端和电脑上位...
- 2021-06-12 5-网络芯片CH395Q学习开发-模块使用Socket0作为TCP客户端和电脑上位机TCP...

Powered by:

博客园

Copyright © 2021 杨奉武

Powered by .NET 5.0 on Kubernetes



单片机,物联网,上位机,...

扫一扫二维码，入群聊。