6.5 LAN communication

1. The purpose of the experiment

In this lesson, we will learn how to set the WiFi module as the client, and the computer as the server to remotely control the lighting within the local area network.

2. Experiment preparation

Add the function of setting the WiFi module as a client in [2. WiFi module networking].

It can automatically connect to the server when it is turned on, and it can be set to transparent transmission mode.

3. Experimental principle

ESP8285 WiFi module acts as a client to receive information from the computer's TCP server, it will parse information, and realize different functions according to the data content.

It is equivalent to remote control of K210 development board in LAN.

4. Experiment procedure

4.1 IO4 and IO5 are the USB serial port pins of the K210 development board, using serial port 3, while the WiFi module pins are IO13 and IO14, using serial port 1.

```
void hardware_init(void)
{
    /* USB Serial port  */
    fpioa_set_function(PIN_UART_USB_RX, FUNC_UART_USB_RX);
    fpioa_set_function(PIN_UART_USB_TX, FUNC_UART_USB_TX);

    /* WIFI module Serial port */
    fpioa_set_function(PIN_UART_WIFI_RX, FUNC_UART_WIFI_RX);
    fpioa_set_function(PIN_UART_WIFI_TX, FUNC_UART_WIFI_TX);

    /* LED light */
    led_init(LED_ALL);
}
```

```
/*****************************HARDWARE-PIN*********************************/
// Hardware IO port, corresponding Schematic
#define PIN_UART_USB_RX       (4)
#define PIN_UART_USB_TX       (5)

#define PIN_UART_WIFI_RX      (13)
#define PIN_UART_WIFI_TX      (14)

/*****************************SOFTWARE-GPIO********************************/
// Software GPIO port, corresponding program
#define UART_USB_NUM          UART_DEVICE_3

#define UART_WIFI_NUM         UART_DEVICE_1

/*****************************FUNC-GPIO************************************/
// Function of GPIO port, bound to hardware IO port
#define FUNC_UART_USB_RX      (FUNC_UART1_RX + UART_USB_NUM * 2)
#define FUNC_UART_USB_TX      (FUNC_UART1_TX + UART_USB_NUM * 2)

#define FUNC_UART_WIFI_RX     (FUNC_UART1_RX + UART_WIFI_NUM * 2)
#define FUNC_UART_WIFI_TX     (FUNC_UART1_TX + UART_WIFI_NUM * 2)
```

4.2 Initialize the configuration of the serial port, the baud rate is set to 115200, the serial port data width is 8 bits, the stop bit is 1 bit, and parity is not used.

```
// Initialize USB serial port, Set the baud rate to 115200
uart_init(UART_USB_NUM);
uart_configure(UART_USB_NUM, 115200, UART_BITWIDTH_8BIT, UART_STOP_1, UART_PARITY_NONE);

/* Initialize the serial port of the WiFi module */
uart_init(UART_WIFI_NUM);
uart_configure(UART_WIFI_NUM, 115200, UART_BITWIDTH_8BIT, UART_STOP_1, UART_PARITY_NONE);
```

4.3 Send "hello yahboom!" when booting, prompting that booting is complete.

```
/* send hello yahboom! when booting */
char *hello = {"hello yahboom!\n"};
uart_send_data(UART_USB_NUM, hello, strlen(hello));
```

4.4 Custom variables are mainly used to record and save data.

```
/* Receive and send buffered data */
char recv = 0, send = 0;
/*Receive WiFi module data flag */
int rec_flag = 0;
/* Save the received data */
char recv_data[MAX_DATA] = {0};
/* recv_data */
uint16_t index = 0;
```

4.5 It is judged when the data is received. The default rec_flag is 0. If you receive a'$', it means to start receiving data, rec_flag=1, where MAX_DATA can be set by yourself, which is larger than the maximum value of the protocol data content.

We set to 10, which means that it can hold up to 10 characters.

```
#define MAX_DATA        10
```

```
/* Receive information from WIFI module*/
if(uart_receive_data(UART_WIFI_NUM, &recv, 1))
{
    /* Send the received data to the USB serial port for display*/
    uart_send_data(UART_USB_NUM, &recv, 1);

    /* Determine whether it meets the data requirements */
    switch(rec_flag)
    {
    case 0:
        /* Start with '$' sign as data */
        if(recv == '$')
        {
            rec_flag = 1;
            index = 0;
            for (int i = 0; i < MAX_DATA; i++)
                recv_data[i] = 0;
        }
        break;
```

4.6 Starting receiving data. First, we need to determine whether it is the'#' terminator, if it is, it ends, and call parse_data to parse the data.
If the maximum data is exceeded and the'#' terminator is not received, it means an exception, make rec_flag to 0.
Finally, it will save the data to recv_data.

```
case 1:
    if (recv == '#')
    {
        /* End with '#' sign as data */
        rec_flag = 0;
        parse_data(recv_data);
    }
    else if (index >= MAX_DATA)
    {
        /* If the data exceeds the maximum and the terminator '#' is not received, i
        rec_flag = 0;
        index = 0;
    }
    else
    {
        /* Save the data to recv_data*/
        recv_data[index++] = recv;
    }
    break;
default:
    break;
```

4.7 Analyzing the data, compare whether the data is consistent with the setting, and execute the corresponding content if it is consistent.

```
void parse_data(char *data)
{
    // uart_send_data_dma(UART_USB_NUM, DMAC_CHANNEL0, data, sizeof(data));
    /* Analyze and compare the sent data */
    if (0 == memcmp(data, "led0_0", 6))
    {
        led0_state(LED_OFF);
    }
    else if (0 == memcmp(data, "led0_1", 6))
    {
        led0_state(LED_ON);
    }
    else if (0 == memcmp(data, "led1_0", 6))
    {
        led1_state(LED_OFF);
    }
    else if (0 == memcmp(data, "led1_1", 6))
    {
        led1_state(LED_ON);
    }
}
```

4.8 Transmit the received serial data to the WiFi module.

```
/* Receive the information from the serial port and send it to the WiFi modul
if(uart_receive_data(UART_USB_NUM, &send, 1))
{
    uart_send_data(UART_WIFI_NUM, &send, 1);
}
```

4.9 Compile and debug, burn and run

Copy the wifi_module folder to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

cmake .. -DPROJ=wifi_module -G "MinGW Makefiles"

make

```
Generating .bin file ...
[100%] Built target wifi_module
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **wifi_module.bin** file will be generated in the build folder.

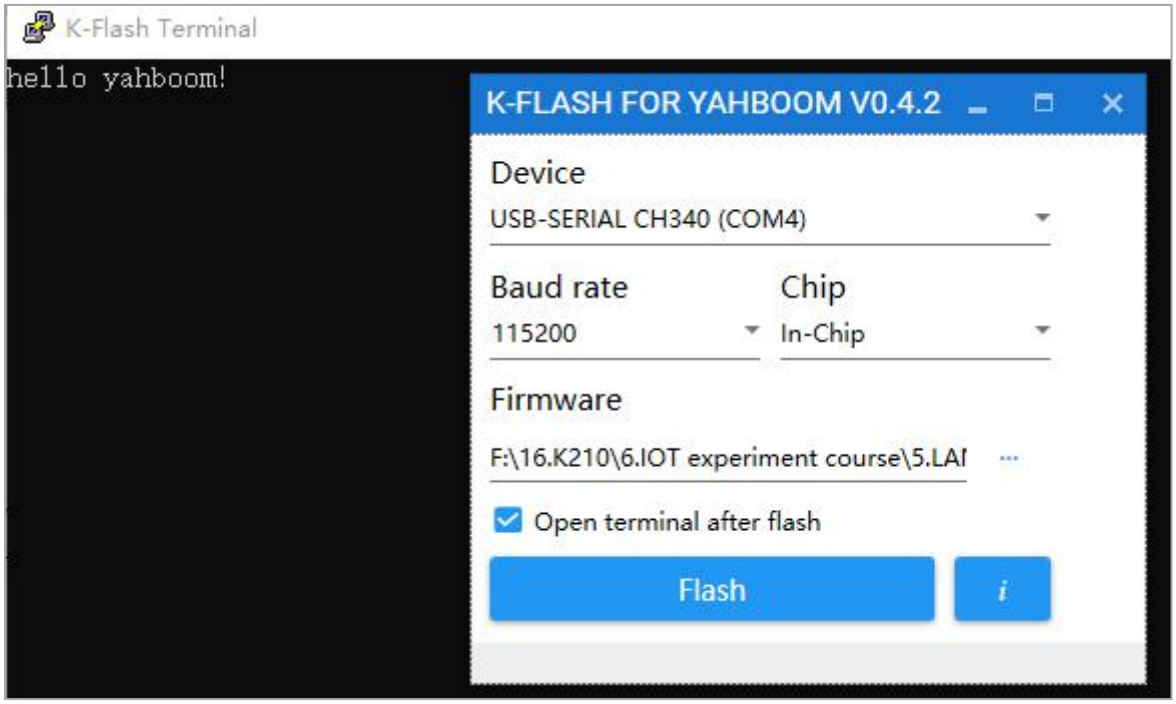
We need to use the type-C data cable to connect the computer and the K210 development board.

Open kflash, select the corresponding device, and then burn the **wifi_module.bin** file to the K210 development board.

5. Experimental phenomenon

5.1 After the firmware is burned, a terminal interface will pop up. If the terminal interface does not

pop up, we can open the serial port assistant to display the debugging content.

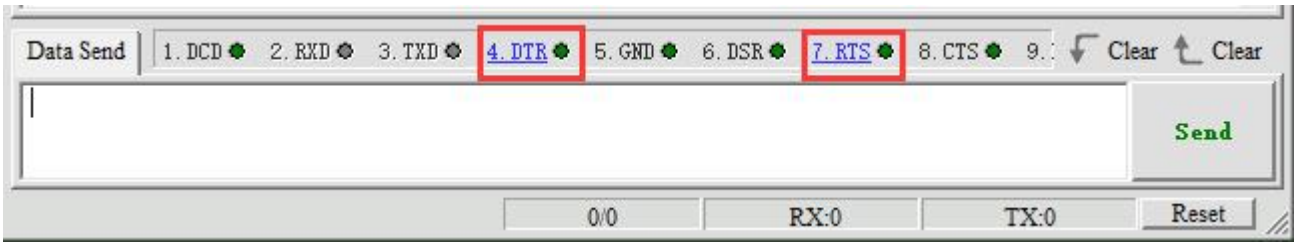


5.2 **Close terminal interface.** Open the serial port assistant of the computer, select the corresponding serial port number of the corresponding K210 development board, set the baud rate to 115200, and then click to open the serial port assistant.

! Note:

We also need to set the DTR and RTS of the serial port assistant.

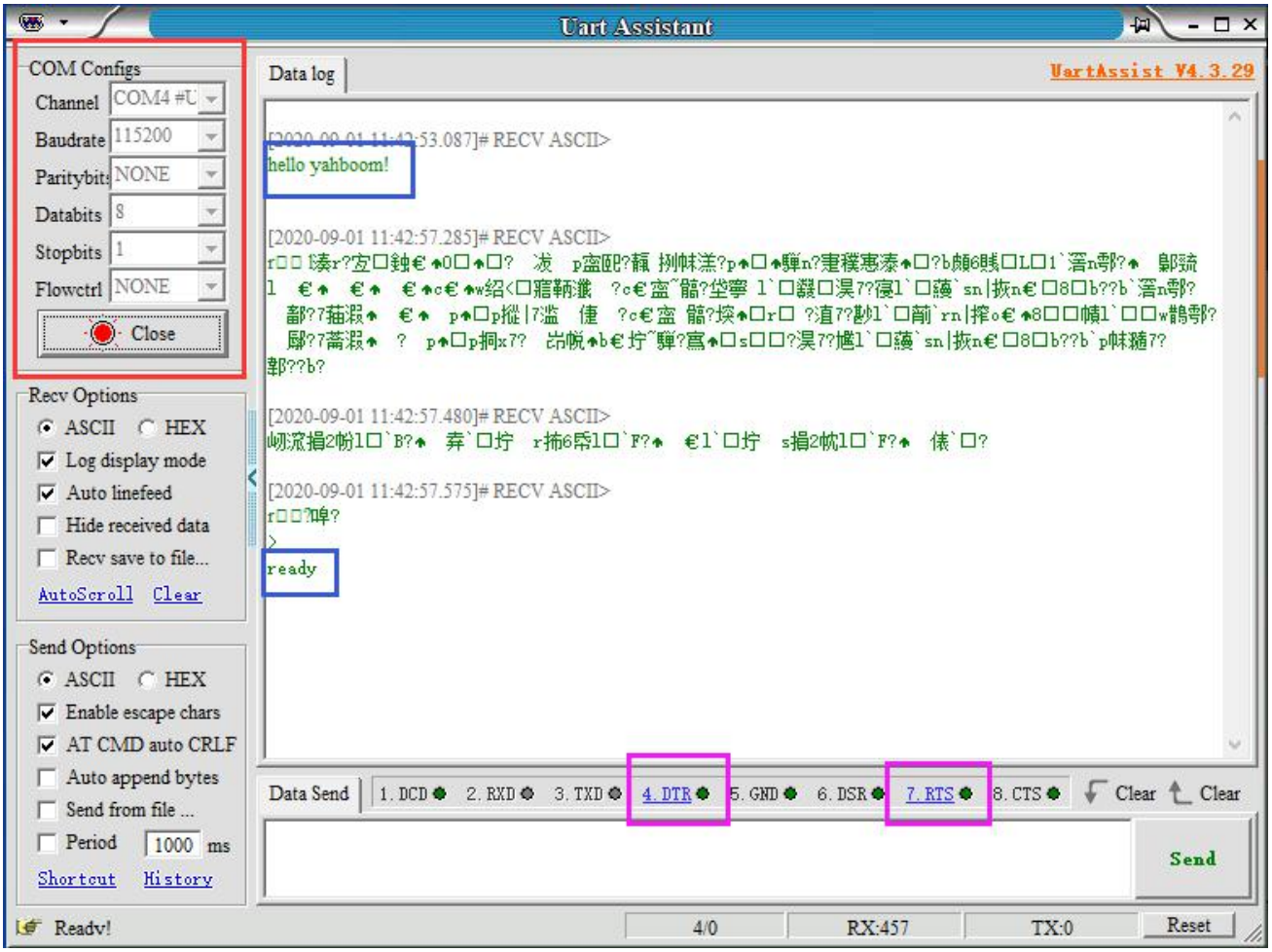
At the bottom of the serial port assistant, we can see that 4.DTR and 7.RTS are red by default. Click 4.DTR and 7.RTS to set both to green, and then press the reset button of the K210 development board.



5.3 Press the **Reset button** on the K210 development board, and the serial debugging assistant will print “hello yahboom!”.

Press the reset button of the WiFi module, you can see a series of garbled codes, as long as you see the “ready” character, it means that the WiFi module is normal.

Since the router has been connected in the previous lesson, we don’t have to repeat the connection this time.



5.4 Open the network debugging assistant Netassist, set the parameters of the network debugging assistant in the upper left corner.

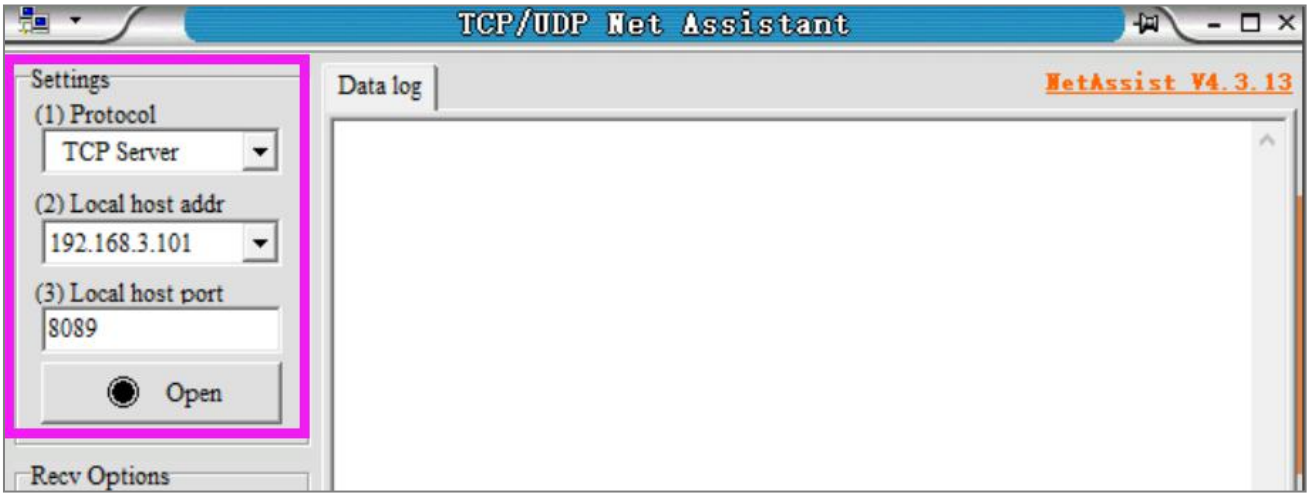
(1) Protocol choose TCP server.

(2) Remote host address: Enter the address of the WiFi module.

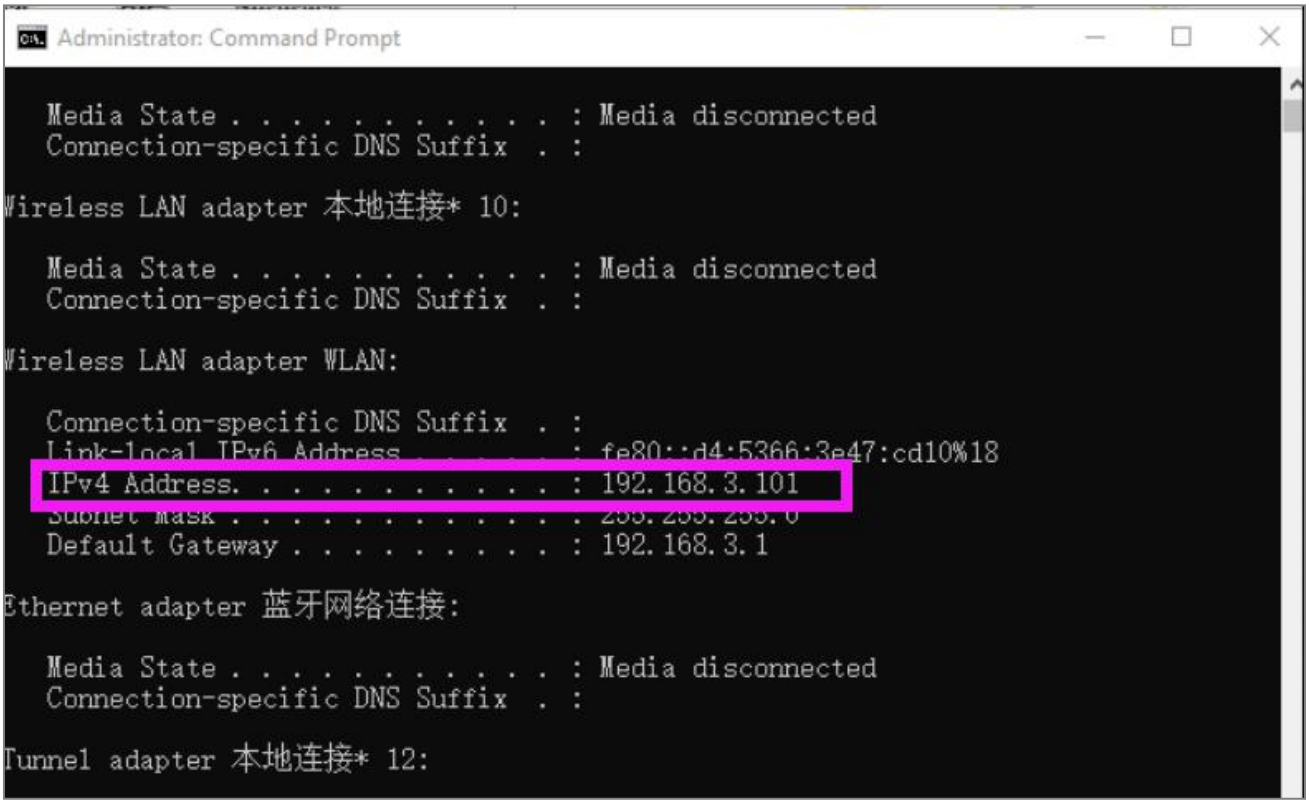
If you forget it, you can enter the AT+CIFSR command to view it.

(3) Remote host port: 8086, which corresponds to the port in the previous step.

(4) Finally, click Connect.

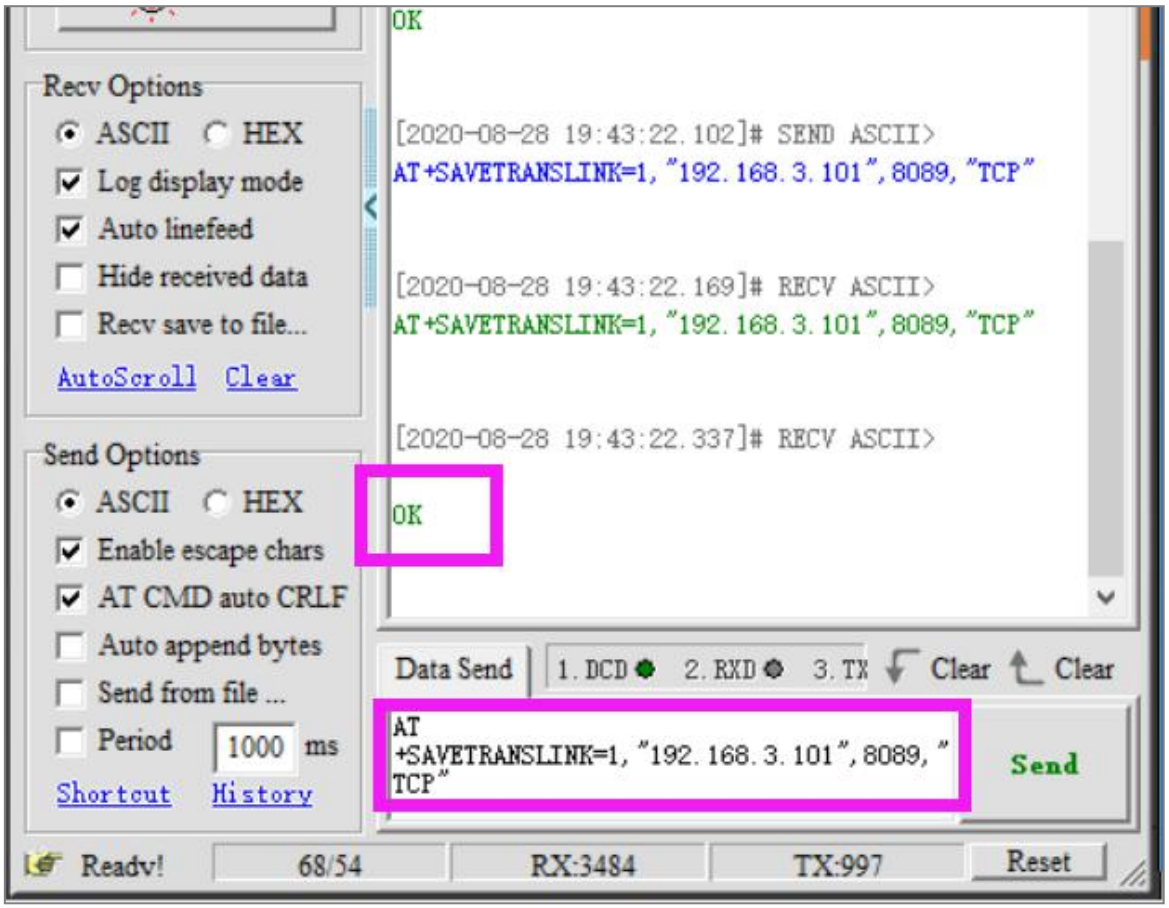


5.5 View your computer IP address.
Press Win+R, input cmd, Then input command ipconfig.



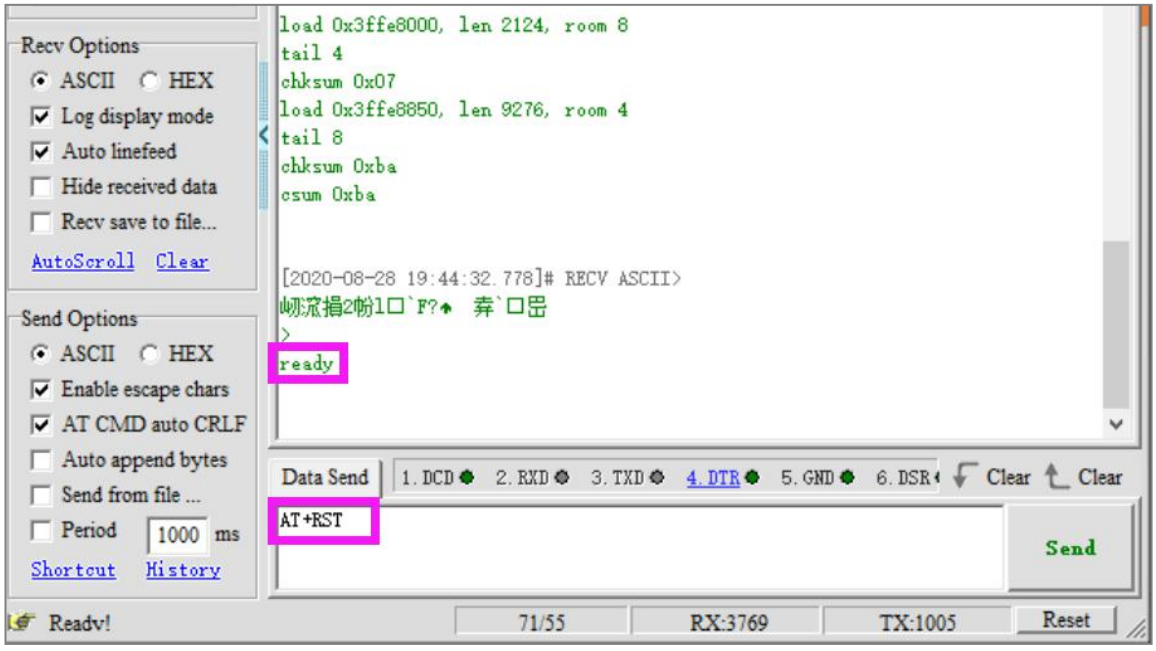
5.6 After the server is turned on, when there is no client connection, the display client is displayed as 0, and a value is automatically increased by 1 for each connection.

5.7 Send the following command to set the WiFi module as a client to connect to the server.
AT+SAVETRANSLINK=1, "Server IP", remote port, "TCP"
For example, AT+SAVETRANSLINK=1, "192.168.3.101", 8089, "TCP"

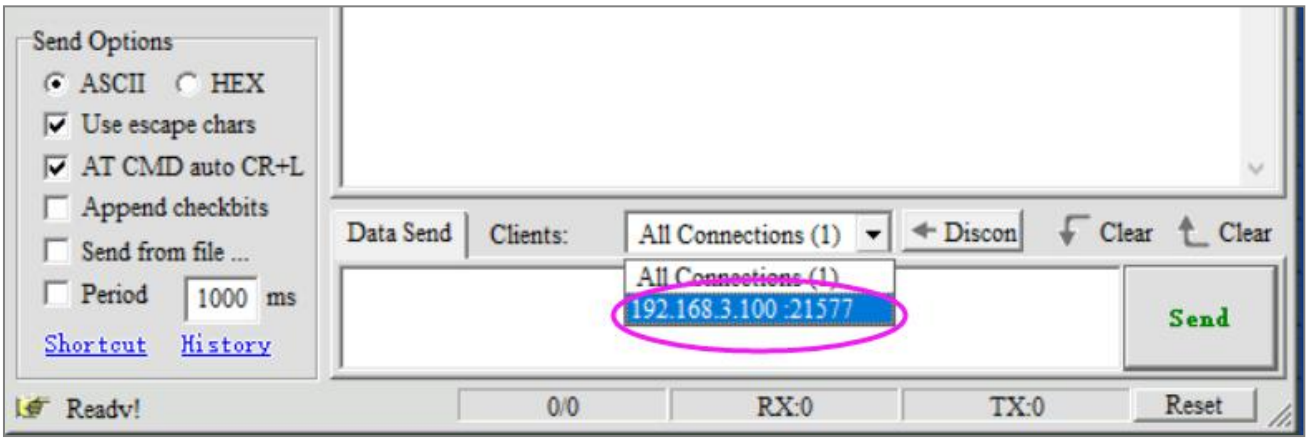


If OK is received, it means that the data has been written in.

We need to restart the WiFi module. Enter the AT+RST command to restart the WiFi module, and if ready is received, it means that the connection has been successful.



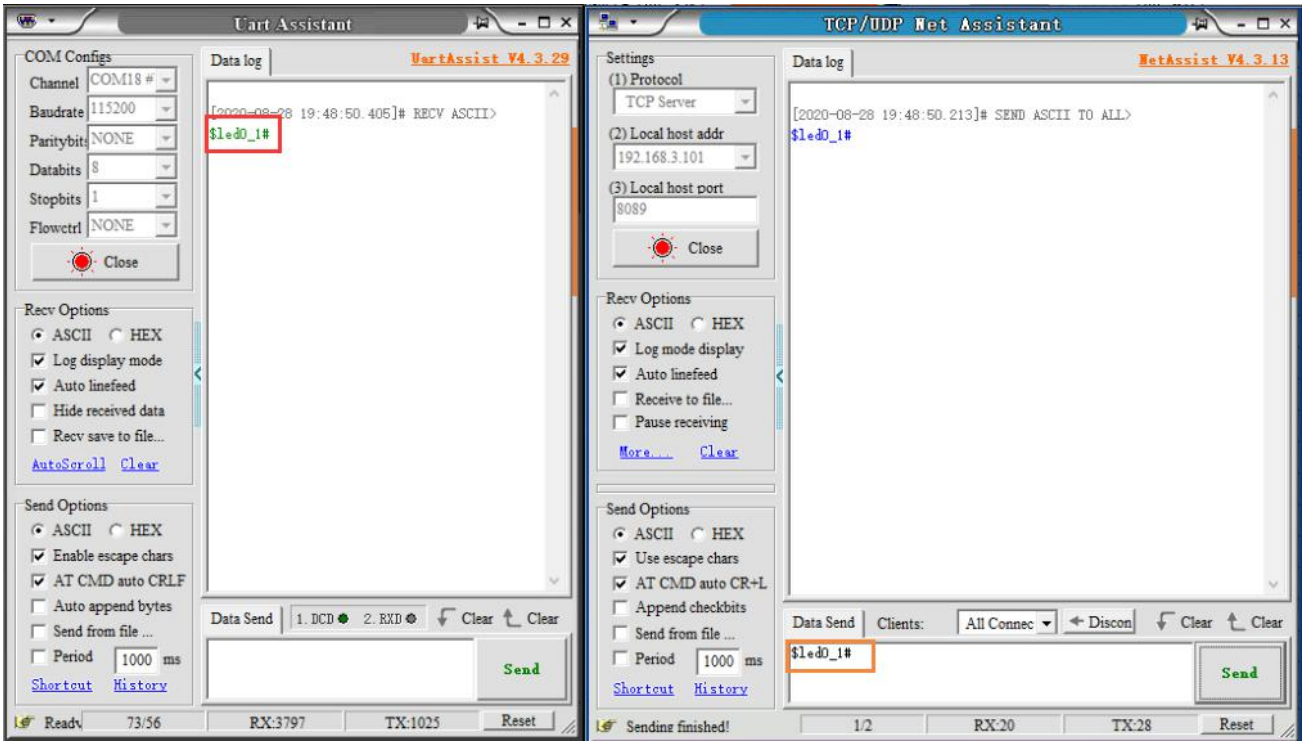
5.8 At this time, server will display a connected device. As shown below.

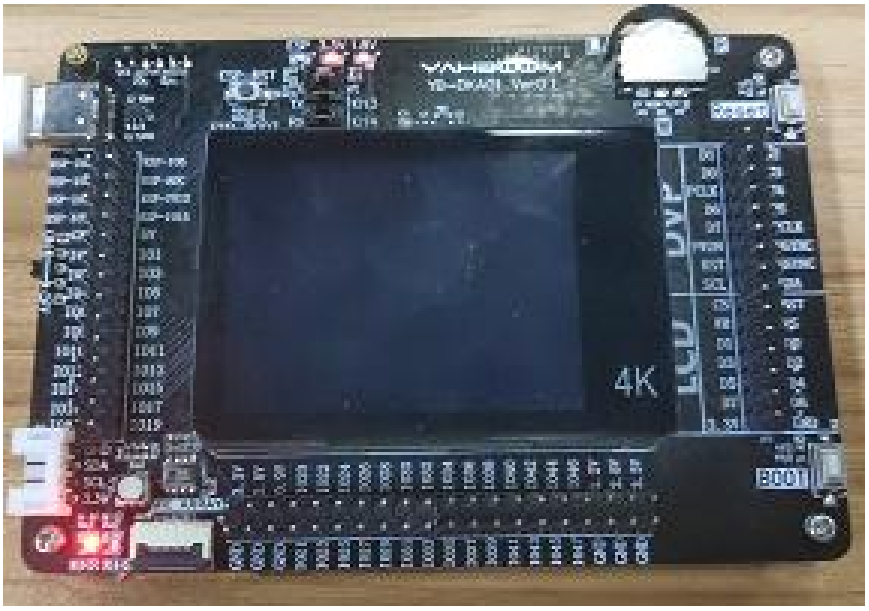


5.9 At this time, data is sent from the server to the client, and the client can display the received data.

Send data from the client to the server, and the server can also receive and display the data. This is how the transparent transmission mode works.

We send the protocol '$led0_1#' to light up LED0 from the server, and we can see that the red light of LED0 lights up.





Functions corresponding to the protocol:

Command	Functions
$led0_1#	LED0 is on
$led0_0#	LED0 is off
$led1_1#	LED1 is on
$led1_0#	LED0 is off