Experiment 8: ESP8266-IoT

In this experiment, student will learn how use ESP8266 modules in order to create Internet of Things (IoT) application. The ESP8266 is a low-cost Wi-Fi microchip, with a full TCP/IP stack and microcontroller capability, produced by Espressif Systems in Shanghai, China. The chip first came to the attention of Western makers in August 2014 with the ESP-01 module as shown in Figure 1, made by a third-party manufacturer Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. However, at first, there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, the chip, and the software on it, as well as to translate the Chinese documentation. The ESP8285 is an ESP8266 with 1 MBytes of built-in flash, allowing the building of single-chip devices capable of connecting to IEEE 802.11 b/g/n Wi-Fi.

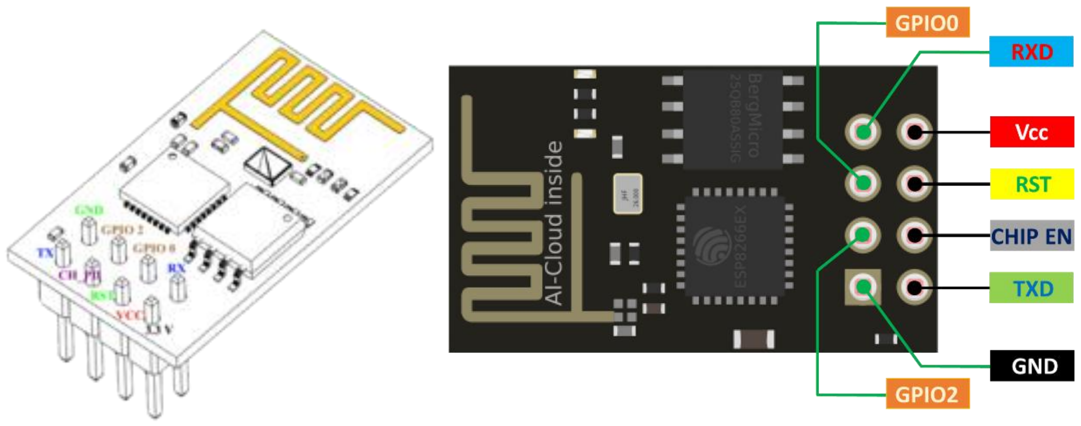


Figure 1. ESP-01 module.

1. AT Commands

AT commands are instructions used to control a modem. AT is the abbreviation of ATtention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands. ESP-01 module can be communicated by AT commands as shown in Table 1

Table 1. AT Commands for ESP-01 module

Function	***AT Command***	***Response***
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>: (CIPMUX=1): + IPD, <id>, <len>: <data>

ESP-01 is connected to STM32F103C8T6 via USART2 in KU Lab board as shown in Figure 2. There are three general pre-set Baud Rates for ESP-01 depended on which ESP-01 versions which are 9600, 19200 and 115200.

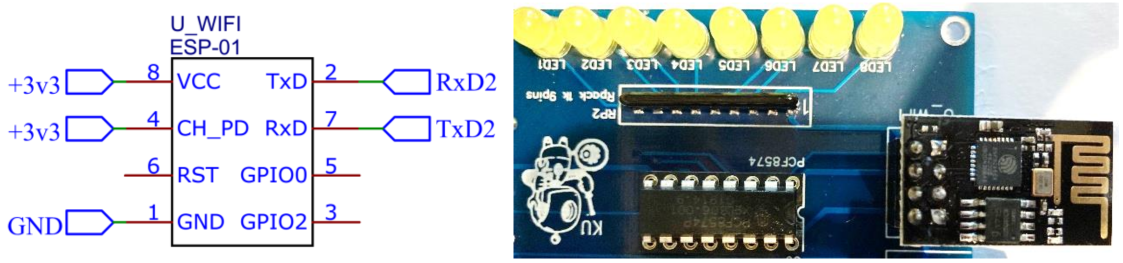


Figure 2. ESP-01 module connection

In this experiment, ESP-01 will be explored and used to create Internet-of-Things (IoT) application. In the first experiment (8.1), the extra hardware will be used. The USB-to-ESP-01 module, shown in Figure 3, is used in order for experimenting the AT command without using the KU Lab board.

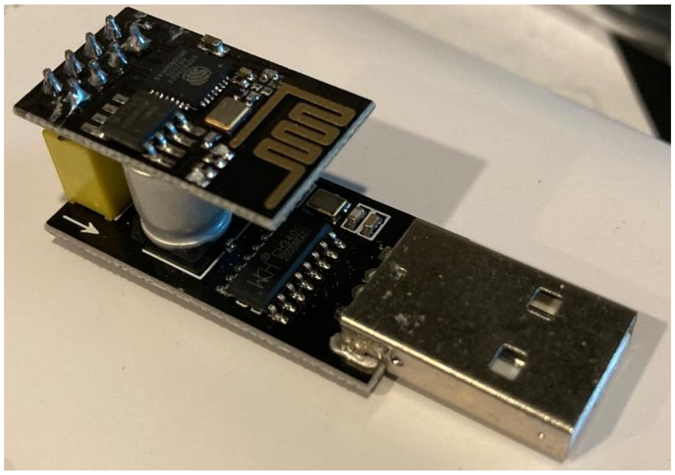
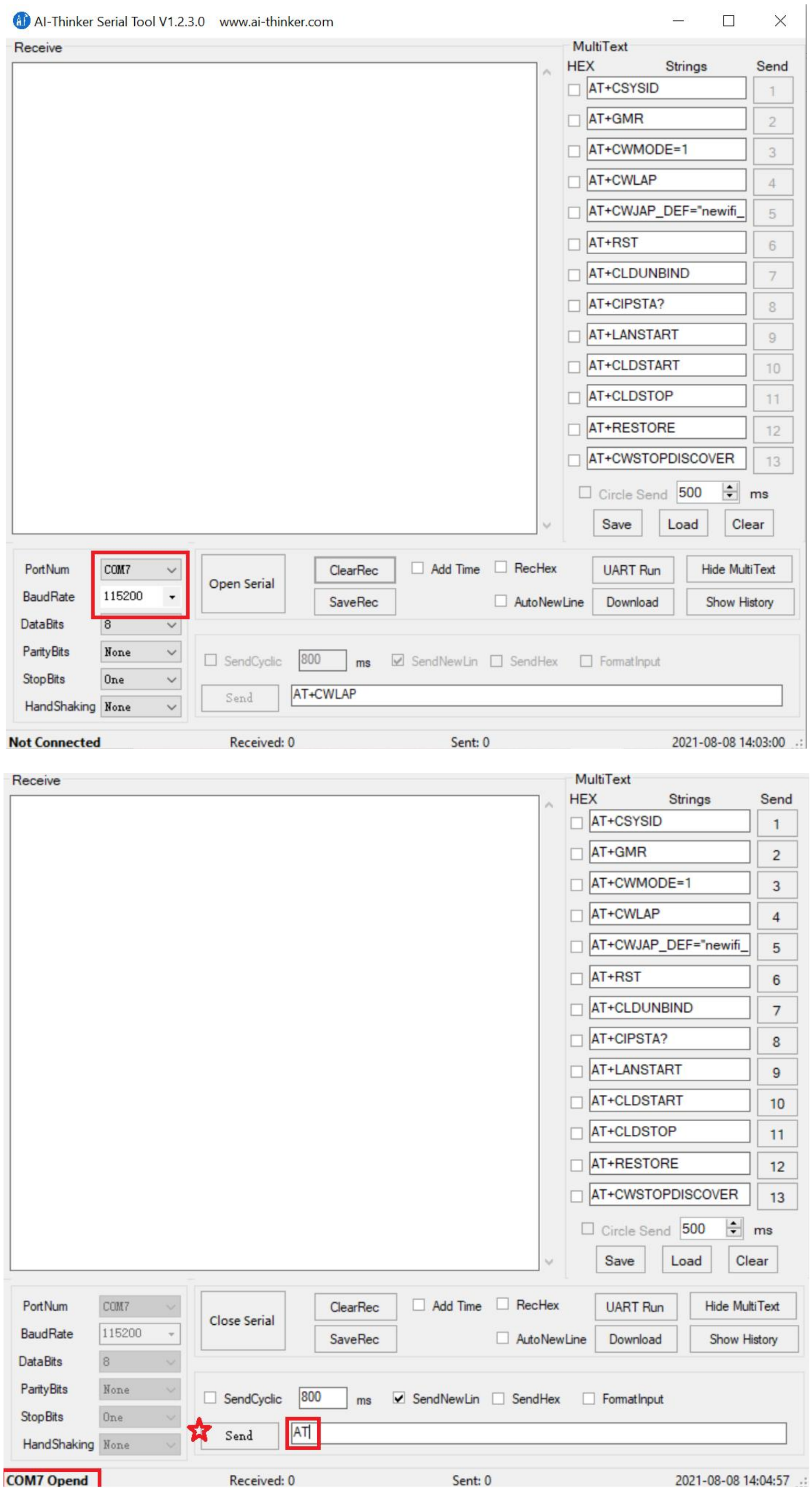


Figure 3. USB-to-ESP-01 module

Experiment 8.1

1. First, remove ESP-01 from KU lab board and plug it into the USB-to-ESP-01 module as show in Figure 2. Then, connect this module to notebook or desktop USB port. Using device manager to identify Serial Port (COM port) number.
2. Open AiThinker Serial Tool, make sure that the COM port number is correct and click Open Serial button to connect to ESP-01. When COM port is opened, then, change the command to AT and click on Send button to get response OK from ESP-01 in the Receive screen.



	Strings	Send
☐	AT+CSYSID	1
☐	AT+GMR	2
☐	AT+CWMODE=1	3
☐	AT+CWLAP	4
☐	AT+CWJAP_DEF="newifi_	5
☐	AT+RST	6
☐	AT+CLDUNBIND	7
☐	AT+CIPSTA?	8
☐	AT+LANSTART	9
☐	AT+CLDSTART	10
☐	AT+CLDSTOP	11
☐	AT+RESTORE	12
☐	AT+CWSTOPDISCOVER	13

Circle Send 500 ms

PortNum COM7
BaudRate 115200
DataBits 8
ParityBits None
StopBits One
HandShaking None

SendCyclic 800 ms ☑ SendNewLin

AT+CWLAP

Not Connected Received: 0 Sent: 0 2021-08-08 14:03:00

AT

COM7 Opend Received: 0 Sent: 0 2021-08-08 14:04:57

Figure 4. The AiThinker Serial Tool Windows and configuration

3. Complete the following table in the Response Column:

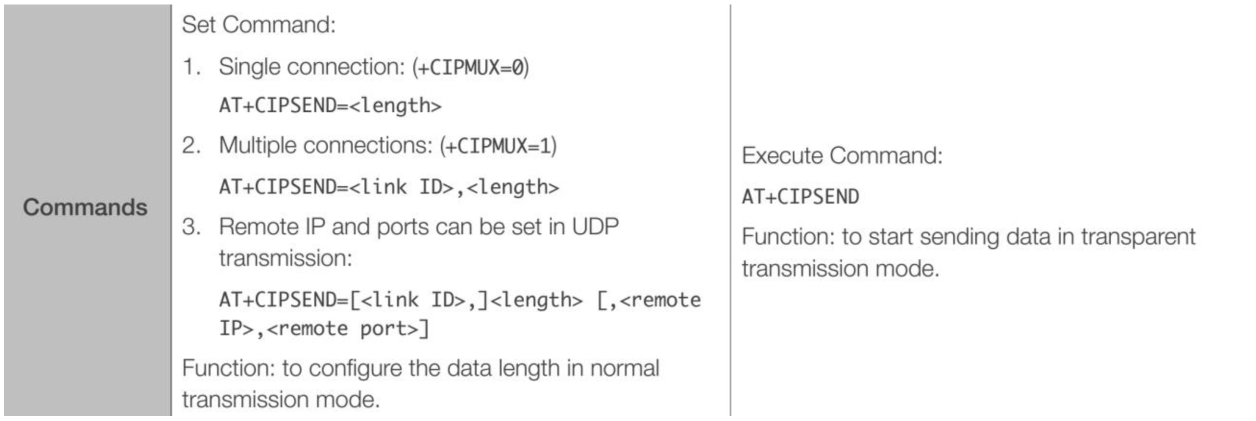
AT Command	Response
AT+CSYSID	
AT+GMR	
AT+CWMODE=1	
AT+CWLAP	
AT+CWJAP="Tenda_48B2E0","1234567890"	
AT+CIPSTA?	

Lab instructor / TA signature

The ESP8266 ESP-01 module has three operation modes:

1. Access Point (AP)
2. Station (STA)
3. Both

In AP the Wi-Fi module acts as a Wi-Fi network, or access point (hence the name), allowing other devices to connect to it. This does not mean that you will be able to check your Facebook from your device while the ESP-01 module is operating in the AP mode. It simply establishes a two-way communication between the ESP8266 and the device that is connected to it via Wi-Fi. In STA mode, the ESP-01 can connect to an AP such as the Wi-Fi network from your house. This allows any device connected to that network to communicate with the module. The third mode of operation permits the module to act as both an AP and a STA. ESP-01 mode can be changed using AT+CWMODE=x Command when 1, 2 and 3 are STA, AP and Both respectively. We need to enable multiple connections by configuring the ESP-01 module via AT+CIPMUX=1 command. To start ESP-01 as server service, command AT+CIPSERVER=1,80 is used. In this mode, the ESP-01 will require for IP address from the access point router. The second number indicates the port that the client uses to connect to a server. Port 80 is chosen because this is the default port for HTTP protocol. Server service can be used after the ESP-01 has IP address while works in STA mode. In experiment 8.2, ESP-01 will be used as the client and will send data to server using AT+CIPSEND command as shown in Figure 5.



Commands	Set Command:
	1. Single connection: (+CIPMUX=0)
	`AT+CIPSEND=<length>`
	2. Multiple connections: (+CIPMUX=1)
	`AT+CIPSEND=<link ID>,<length>`
	3. Remote IP and ports can be set in UDP transmission:
	`AT+CIPSEND=[<link ID>,]<length> [,<remote IP>,<remote port>]`
	Function: to configure the data length in normal transmission mode.
		Execute Command:
		`AT+CIPSEND`
		Function: to start sending data in transparent transmission mode.

Figure 5. AT+CIPSEND command

Experiment 8.2

1. First, new project according to experiment 0 and assign project name as Lab8.
2. Write down the **USART12_setup( )**, **USART2_NVIC_Config( )**, **USART1_sendC( )**, **USART2_sendC( )**, **USART1_putS( )** and **USART2_putS( )** function to the **main.c** file in the project before main( ) function as below:

```c
void USART12_setup()
{
  USART_InitTypeDef usart_init_struct;
/* Baud rate 115200, 8-bit data, One stop bit No parity, Do both Rx and Tx, No HW flow control*/
  usart_init_struct.USART_BaudRate = 115200;
  usart_init_struct.USART_WordLength = USART_WordLength_8b;
  usart_init_struct.USART_StopBits = USART_StopBits_1;
  usart_init_struct.USART_Parity = USART_Parity_No ;
  usart_init_struct.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
  usart_init_struct.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
/* Configure USART1 & 2 */
  USART_Init(USART1, &usart_init_struct);
  USART_Init(USART2, &usart_init_struct);
/* Configure Interrupt */
/* Disable Tx and Enable Rx */
  USART_ITConfig(USART2,USART_IT_RXNE,ENABLE);
  USART_Cmd(USART1, ENABLE);
  USART_Cmd(USART2, ENABLE);
}
```

```c
void USART2_NVIC_Config(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;
// Priority grouping
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_3);
// Configure interrupt priority
  NVIC_InitStructure.NVIC_IRQChannel = USART2_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

```c
void USART1_sendC(unsigned char c)
{
  while(USART_GetFlagStatus(USART1,USART_FLAG_TXE)==RESET);
  USART_SendData(USART1,(unsigned char) c);
}
```

```
void USART2_sendC(unsigned char c)
{
  while(USART_GetFlagStatus(USART2,USART_FLAG_TXE)==RESET);
  USART_SendData(USART2,(unsigned char) c);
}

void USART1_putS(unsigned char *s)
{
  while (*s) {
            USART1_sendC(*s);
            s++;
  }
}

void USART2_putS(unsigned char *s)
{
  while (*s) {
            USART2_sendC(*s);
            s++;
  }
}
```

3. Write down the **GPIO_configuration( )**, **clear_rx_buf( )** , **waitforOK( )** and **USART2_IRQHandler( )** function to the **main.c** file in the project before main( ) function as below:

```
void GPIO_Configuration()
{
// Define variable type GPIO
  GPIO_InitTypeDef GPIO_InitStruct;
// Enable clock for GPIOC and GPIO A and GPIO B and AFIO and USART1 and USART2
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC | RCC_APB2Periph_GPIOA , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO | RCC_APB2Periph_USART1 , ENABLE);
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB , ENABLE);
//  Configure PC13 as output push pull 50Hz
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStruct);
  GPIO_WriteBit(GPIOC,GPIO_Pin_13,0);
//  Configure PA0 and PA1 as Analog input
  GPIO_InitStruct.GPIO_Pin= GPIO_Pin_0 | GPIO_Pin_1;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
//  Configure PA9 Tx for USART1 and PA2 Tx for USART2
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_2;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
  GPIO_Init(GPIOA, &GPIO_InitStruct);
//  Configure PA10 Rx for USART1 and PA3 Rx for USART2
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_10 | GPIO_Pin_3;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOA, &GPIO_InitStruct);

// Configure PB1 and PB15 as output push pull 50MHz
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_15 | GPIO_Pin_1;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOB, &GPIO_InitStruct);

// Configure PB3, PB4, PB8 and P9 as input floating
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3 | GPIO_Pin_4 | GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOB, &GPIO_InitStruct);
}
```

```
/* Public declaration */
#define RX_BUF_SIZE 512
#define RX_BUF_MASK RX_BUF_SIZE-1
volatile char rx_buf[RX_BUF_SIZE]="";
volatile u16 rx_pos=0;

void USART2_IRQHandler(void)
{
  unsigned char rx_b;
  if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET)
  {
        rx_b = USART_ReceiveData(USART2);
        rx_buf[rx_pos++]=rx_b;
        rx_pos &= RX_BUF_MASK;
  }
  rx_buf[rx_pos+1]='\0';
}
```

```
void clear_rx_buf(void)
{
        for(;rx_pos!=0;rx_pos--) rx_buf[rx_pos]='\0';
}
```

```
void waitforOK(void)
{
unsigned char ch1 , ch2;
unsigned char check=0;
while(check==0) // wait unit get OK from ESP8266 usart2
{
  if (rx_pos >= 4)

```

```
    {
        ch1 = rx_buf[rx_pos-4];
        if (ch1 =='O')
        {
             while(1)
             {
                  ch2 = rx_buf[rx_pos-3];
                  if (ch2 =='K')
                  {
                       check = 1;
                  }
             if (check == 1) break;
             }
        }
        if(check == 1) break;
    }
  }
  }
```

4. Add **DelayMC( )** function from Experiment 3 and edit the **main( )** function in **main.c** as follow:

```
/* ---- ESP 01 Predefined AT Protocol ----- */
const char AT_test[] =          "AT\r\n";
const char echoOff[] =          "ATE0\r\n";
const char restore[] =          "AT+RESTORE\r\n";  //Restore factory settings
const char uart_baud [] =       "AT+UART=115200,8,1,0,0\r\n" ;//Baudrate
const char list_AP [] =         "AT+CWLAP\r\n";
const char station_mode [] =    "AT+CWMODE=1\r\n"; //station mode
const char conect_to_AP [] =    "AT+CWJAP=\"Tenda_48B2E0\",\"1234567890\"\r\n";
const char disconectAP [] =      "AT+CWQAP\r\n";
const char conect_AP_status [] = "AT+CWJAP?\r\n";
const char firmware_version [] = "AT+GMR\r\n";
const char NoMultConn[] =        "AT+CIPMUX=0\r\n";
const char rst [] =              "AT+RESTORE\r\n";
const char get_IP [] =           "AT+CIFSR\r\n";
const char send_TSP [] =         "AT+CIPSEND=";
int main(void)
{
   unsigned char * msg;
   unsigned int ADCRead_value = 0 , i = 0 ;
   static const char digits [] = "0123456789ABCDEF";
   unsigned char adc_msg[] = "xxxx\r\n\n\0";
   GPIO_Configuration();
   USART1_setup();
   USART2_NVIC_Config();
   USART2_setup();
   USART1_putS("< Configuration Finish > \n\r");
   USART1_putS("Start Connecting to ESP01\r\n\0");
   USART1_putS("Waiting  for Echo \n\r");

   USART2_putS(AT_test);
   DelayMC(20);
```

```
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(firmware_version);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(station_mode);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(list_AP);
    DelayMC(200);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(conect_to_AP);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(conect_AP_status);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(get_IP);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

    USART2_putS(NoMultConn);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();

//---- Change Code for Experiment 8.3 After This Line ----//

    while(1){

    }
}
```

5. Compile and program into the board, connect board to ProcessingGrapher program and connect via Serial communication. Reset the STM32F103C8T6 and explain the result that show on ProcessingGrapher screen.

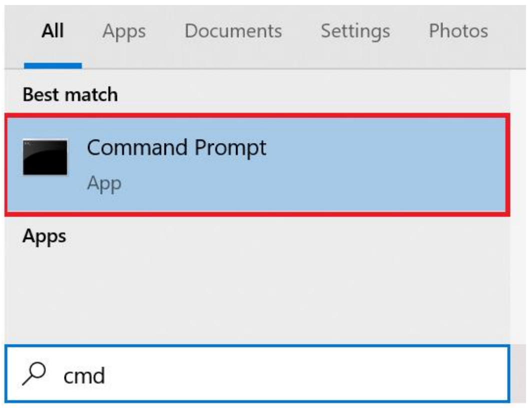
____
____
____
____
____
____
____
____

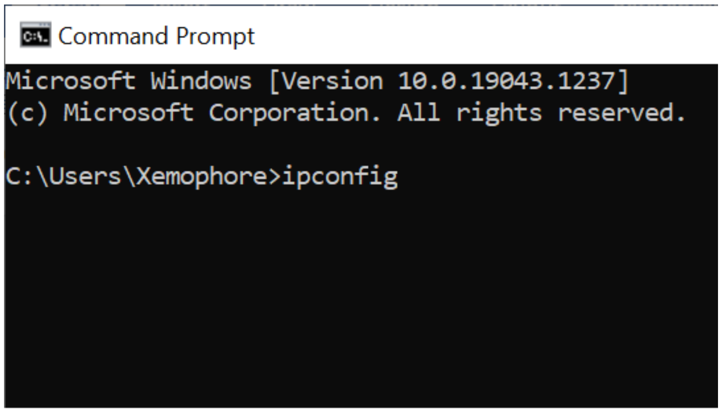
Your Board IP address ______________________________

Lab instructor / TA signature

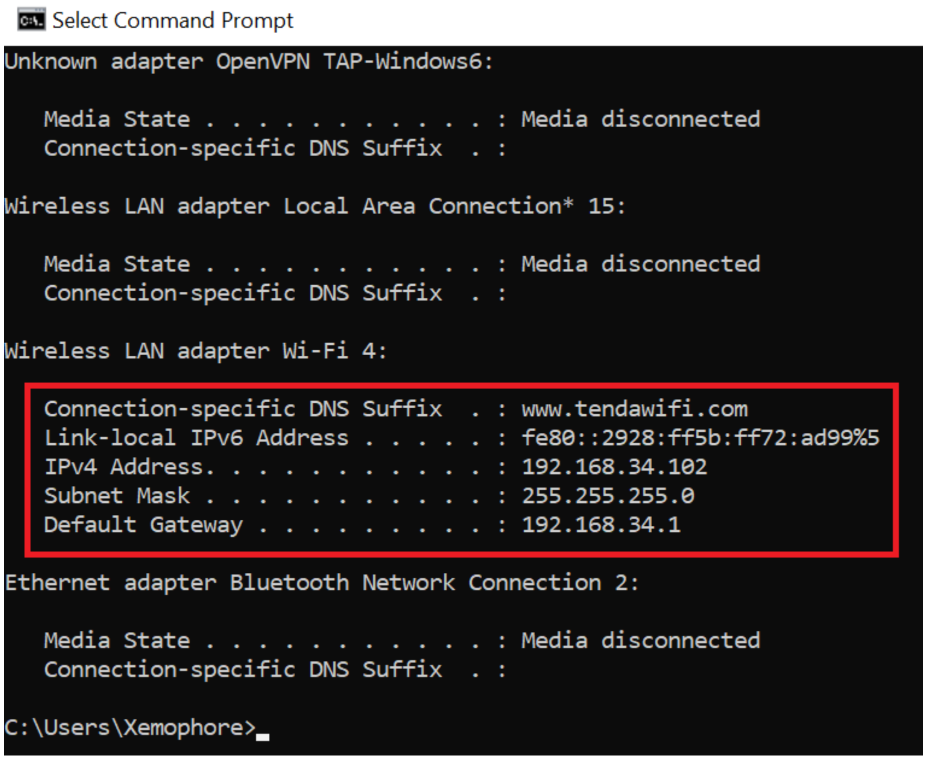
2. ESP-01 Client

From the previous experiment, ESP-01 can connect to Access point (Router or Hub) and get the IP address. This experiment will control the ESP-01 to connect to Server via server IP address through WIFI with TCP protocol. The port that will be used in this experiment is 80 in order to simulate the www default port. Analog to digital module in STM32F103C8T6 will be used to read the voltage of variable resister VR0. Before start this experiment, the IP address of the Server must be found. Your computer will be used as the Server in this experiment. The IP can be found by using terminal. By typing "cmd" in search, then, select Command Prompt. The new Command Prompt windows will pop-up, then, type "ipconfig" and see what is your computer IP address. The steps to find computer IP address is shown in Figure 6





```
Microsoft Windows [Version 10.0.19043.1237]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Xemophore>ipconfig
```

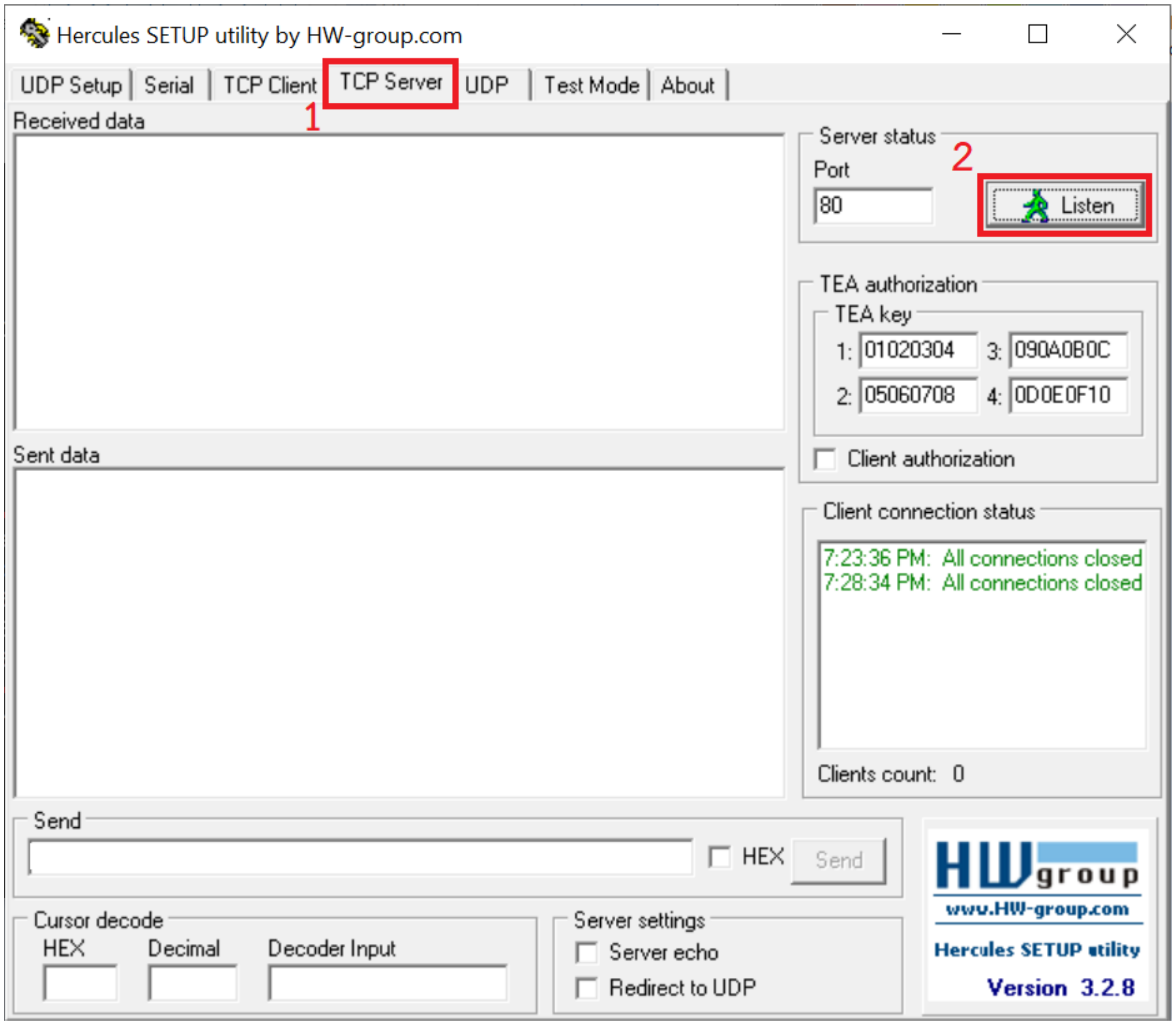


```
Unknown adapter OpenVPN TAP-Windows6:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 15:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi 4:

   Connection-specific DNS Suffix  . : www.tendawifi.com
   Link-local IPv6 Address . . . . . : fe80::2928:ff5b:ff72:ad99%5
   IPv4 Address. . . . . . . . . . . : 192.168.34.102
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.34.1

Ethernet adapter Bluetooth Network Connection 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Users\Xemophore>_
```

Figure 6. IP address of computer Server.

Your Server IP address is ________________________________

In order to create Server, Hercules is used. By starting the program Hercules and select TCP Server tap and click on Listen button, then, the Server will start as shown in Figure 7. ESP-01 send command is explained as Figure 8.



```
7:23:36 PM:  All connections closed
7:28:34 PM:  All connections closed
```

Figure 7. Hercules program for Server.

Experiment 8.3

1. Use the same project as experiment 8.2. Add the ADC1_Init( ) and **readADC1 ( )** function from **Experiment 3** into the **main.c** .
2. Find computer IP address and use it as Server IP address in Code
3. Open Hercules Program and start the TCP Server.
4. Add function **waitforPromp( )** to main.c. Modify the **main( )** function from experiment 8.2 after Line 87 `//---- Change Code for Experiment 8.3 After This Line ----//` by replace xxx.xxx.xxx.xxx with Your Server IP address (for example `192.168.34.102`):

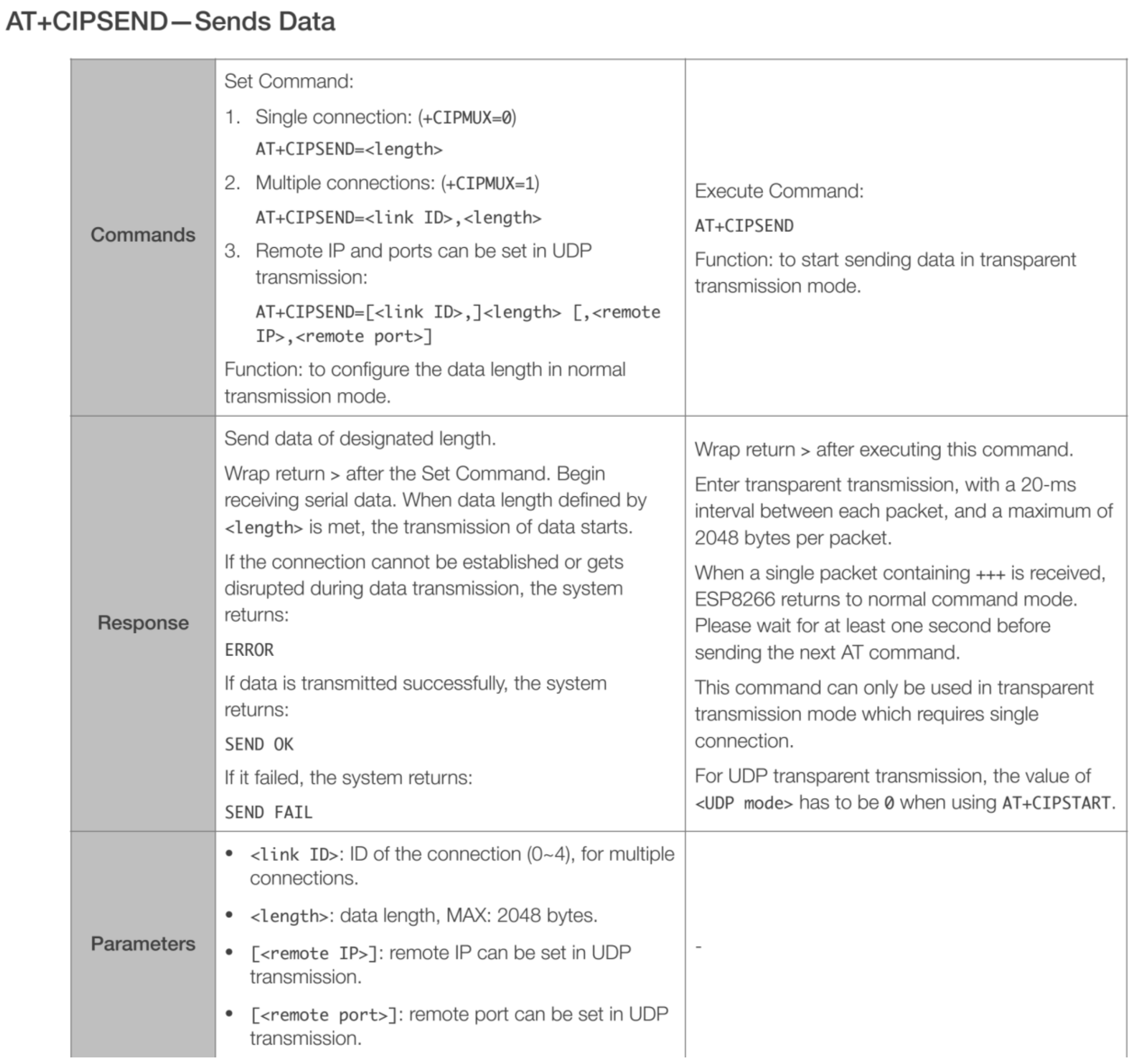
```c
void waitforPromp(void)
{
  unsigned char ch1;
  DelayMC(80);
  while(1) // wait unit get > from ESP8266 usart2
  {
        ch1 = rx_buf[rx_pos-2];
        if(ch1 =='>')
        {
            break;
        }
  }
}
```

```c
//---- Change Code for Experiment 8.3 After This Line ----//
  msg = "AT+CIPSTART=\"TCP\",\"xxx.xxx.xxx.xxx\",80\r\n\0";
  USART2_putS(msg);
  DelayMC(20);
  waitforOK();
  USART1_putS(rx_buf);
  clear_rx_buf();
  USART1_putS("Send Information \r\n\0");
  msg = "AT+CIPSEND=5\r\n\0";
  USART2_putS(msg);
  DelayMC(20);
  waitforPromp();
  USART1_putS(rx_buf);
  clear_rx_buf();
  msg = "Hello\n\0";
  USART2_putS(msg);
  DelayMC(20);
  waitforOK();
  USART1_putS(rx_buf);
  clear_rx_buf();

  ADC1_Init();

  while(1){

     ADCRead_value = readADC1(0);
     //adc_msg[] = "xxxx\n\r\n\0";
     adc_msg[0]= digits[(int)(ADCRead_value /1000)%10];
     adc_msg[1]= digits[(int)(ADCRead_value /100)%10];
     adc_msg[2]= digits[(int)(ADCRead_value /10)%10];
     adc_msg[3]= digits[(int)(ADCRead_value /1)%10];

     USART1_putS("-------ADC read-------\r\n");
     USART1_putS(adc_msg);
     USART1_putS("----------------------\r\n");

     msg = "AT+CIPSEND=6\r\n\0";
     USART2_putS(msg);
```

```
    DelayMC(20);
    waitforPromp();
    USART1_putS(rx_buf);
    clear_rx_buf();
    USART2_putS(adc_msg);
    DelayMC(20);
    waitforOK();
    USART1_putS(rx_buf);
    clear_rx_buf();
    DelayMC(700);
     }
}
```

5. Compile and program into the board. Connect board to ProcessingGrapher program and connect via Serial communication. Reset the STM32F103C8T6 and explain the result that show on ProcessingGrapher screen and on Hercules program. Rotate variable resistor VR0 on KU Lab board explain the result.

Lab instructor / TA signature



AT+CIPSEND—Sends Data

Commands	Set Command:
	1. Single connection: (`+CIPMUX=0`)
	`AT+CIPSEND=<length>`
	2. Multiple connections: (`+CIPMUX=1`)
	`AT+CIPSEND=<link ID>,<length>`
	3. Remote IP and ports can be set in UDP transmission:
	`AT+CIPSEND=[<link ID>,]<length> [,<remote IP>,<remote port>]`
	Function: to configure the data length in normal transmission mode.
		Execute Command:
		`AT+CIPSEND`
		Function: to start sending data in transparent transmission mode.

Response	Send data of designated length.
	Wrap return > after the Set Command. Begin receiving serial data. When data length defined by `<length>` is met, the transmission of data starts.
	If the connection cannot be established or gets disrupted during data transmission, the system returns:
	`ERROR`
	If data is transmitted successfully, the system returns:
	`SEND OK`
	If it failed, the system returns:
	`SEND FAIL`
		Wrap return > after executing this command.
		Enter transparent transmission, with a 20-ms interval between each packet, and a maximum of 2048 bytes per packet.
		When a single packet containing +++ is received, ESP8266 returns to normal command mode. Please wait for at least one second before sending the next AT command.
		This command can only be used in transparent transmission mode which requires single connection.
		For UDP transparent transmission, the value of `<UDP mode>` has to be `0` when using `AT+CIPSTART`.

Parameters	• `<link ID>`: ID of the connection (0~4), for multiple connections. • `<length>`: data length, MAX: 2048 bytes. • `[<remote IP>]`: remote IP can be set in UDP transmission. • `[<remote port>]`: remote port can be set in UDP transmission.	-

Figure 8. AT+CIPSEND command.

3. ESP-01 Receive Data

In this experiment, ESP-01 will be connected to the Server as experiment 8.3 and also receive data from Server. If the Server send data to ESP-01, then, ESP-01 will response +IPD (Receives Network Data) string out of IC via serial communication. In this experiment, Server will send command via WIFI to ESP-01 that connected to STM32 for on and off the LED on KU Lab board.

+IPD—Receives Network Data

Command	Single connection: `(+CIPMUX=0)+IPD,<len>[,<remote IP>,<remote port>]:<data>`	multiple connections: `(+CIPMUX=1)+IPD,<link ID>,<len>[,<remote IP>,<remote port>]:<data>`
Parameters	The command is valid in normal command mode. When the module receives network data, it will send the data through the serial port using the `+IPD` command. • `[<remote IP>]`: remote IP, enabled by command `AT+CIPDINFO=1`. • `[<remote port>]`: remote port, enabled by command `AT+CIPDINFO=1`. • `<link ID>`: ID number of connection. • `<len>`: data length. • `<data>`: data received.	

Experiment 8.4

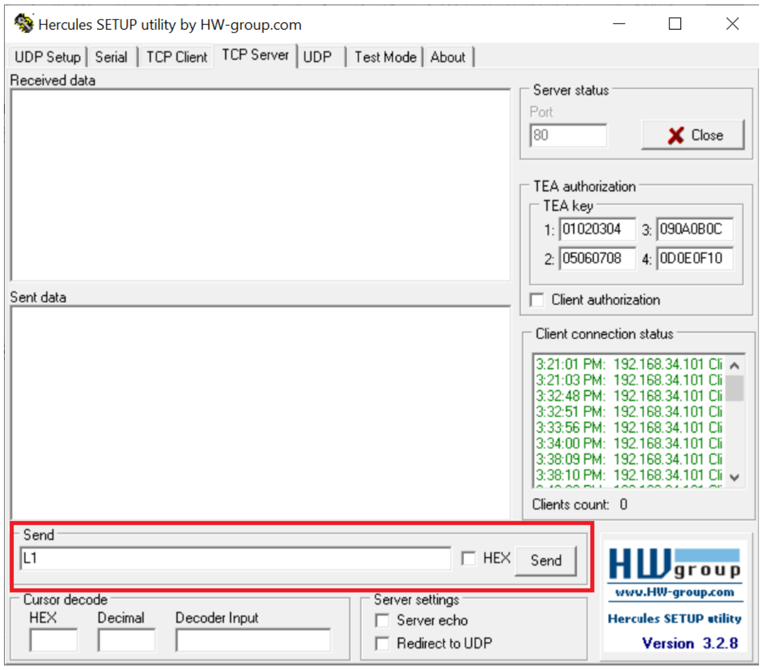
1. Use the same project as experiment 8.3. Add **waitforIPD( )** function into **main.c** as follow:

```c
void waitforIPD()
{
  unsigned char ch1 , ch2 , ch3;
  unsigned char check=0;
  unsigned int i;
  while(check==0) // wait unit get +IPD from ESP8266 usart2
  {
      if (rx_pos >= 6)
      {
        i = 0;
        while(i < (rx_pos+2) )
        {
            ch1 = rx_buf[rx_pos-i];
            if (ch1 =='D')
            {
                ch2 = rx_buf[rx_pos-i-1];
                if (ch2 =='P')
                {
                    ch3 = rx_buf[rx_pos-i-2];
                    if (ch3 =='I')
                    {
                        check =1;
                        break;
                    }
                }
            }
            i++;
            if(check==1) break;
        }
      }
  }
DelayMC(50);
}
```

2. Modify the **main( )** function from experiment 8.2 after Line 87 `//---- Change Code for Experiment 8.3 After This Line ----//` by replace xxx.xxx.xxx.xxx with Your Server IP address (for example `192.168.34.102`):

```
//---- Change Code for Experiment 8.3 After This Line ----//
  msg = "AT+CIPSTART=\"TCP\",\"xxx.xxx.xxx.xxx\",80\r\n\0";
  USART2_putS(msg);
  DelayMC(20);
  waitforOK();
  USART1_putS(rx_buf);
  clear_rx_buf();
  while(1)
  {
    //--------------- Receive ----------------//
    USART1_putS("Wait for Command \r\n\0");
    clear_rx_buf();
    DelayMC(20);
    waitforIPD2();
    USART1_putS("Got Command \r\n\0");
    USART1_putS("--------------------\r\n\0");
    USART1_putS(rx_buf);
    USART1_putS("\n\r\0");
    USART1_putS("--------------------\r\n\0");
    USART1_putS("Executing Command \r\n\0");
    USART1_putS("--------------------\r\n\0");
    i=0;
    while(1)
    {
        //USART1_sendC(rx_buf[it]);
        if (rx_buf[i] != ':')
           {   i++; }
        else
           {  break; }
    }
    if(rx_buf[i+1]=='L')
    {
        //USART1_sendC(rx_buf[it+1]);
        if(rx_buf[i+2]=='1')
            GPIO_WriteBit(GPIOB,GPIO_Pin_1,1);
        if(rx_buf[i+2]=='2')
            GPIO_WriteBit(GPIOB,GPIO_Pin_15,1);
        if(rx_buf[i+2]=='3')
            GPIO_WriteBit(GPIOB,GPIO_Pin_1,0);
        if(rx_buf[i+2]=='4')
            GPIO_WriteBit(GPIOB,GPIO_Pin_15,0);
    }
    USART1_putS("\n\r\0");
    clear_rx_buf();
  }
}
```

3. Find your computer IP address. Open Hercules program and set to TCP Server mode.
4. Compile and program into the board. Connect board to ProcessingGrapher program and connect via Serial communication. Reset the STM32F103C8T6 and explain the result that show on ProcessingGrapher screen and on Hercules program. Wait until KU Lab board show message “Wait for Command” on ProcessingGrapher terminal, the, send L1 from Hercules via WIFI to KU Lab board. See the result from LED on the board, then, change command from L1 to L2, to L3 and to L4, write down the results.



Port: 80

TEA key 1: 01020304 2: 05060708 3: 090A0B0C 4: 0D0E0F10

```
3:21:01 PM:  192.168.34.101 Cli
3:21:03 PM:  192.168.34.101 Cli
3:32:48 PM:  192.168.34.101 Cli
3:32:51 PM:  192.168.34.101 Cli
3:33:56 PM:  192.168.34.101 Cli
3:34:00 PM:  192.168.34.101 Cli
3:38:09 PM:  192.168.34.101 Cli
3:38:10 PM:  192.168.34.101 Cli
```

Clients count: 0

Send: L1

Lab instructor / TA signature

Homework

1. Modified waitforIPD( ) function to waitforIPD(timeout) which timeout is the time that this function has to wait for information from TCP server in msec unit. Then, combine experiment 8.3 and 8.4 together to create a new program that can send voltage from VR0 and VR1 and receive command from TCP server to on and off LEDs (GPIOB pin 1 and pin 15) via WIFI communication simultaneously.

Lab instructor / TA signature