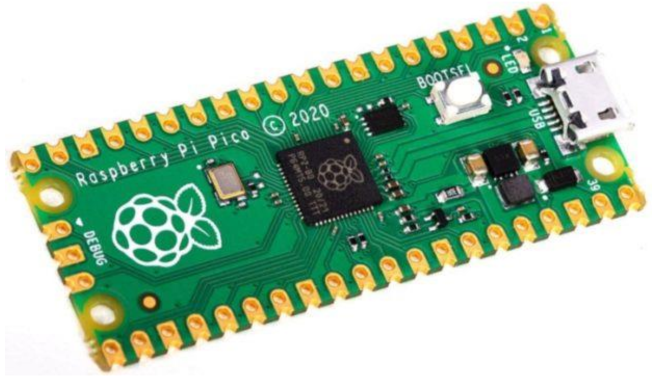
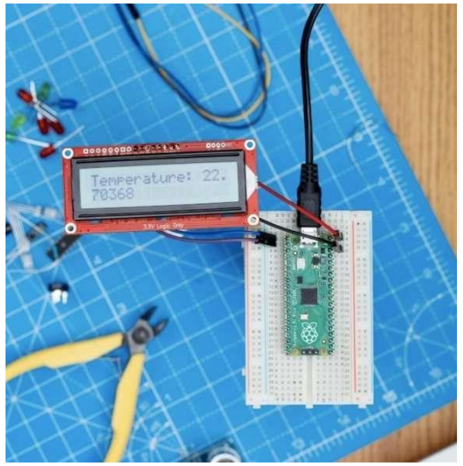
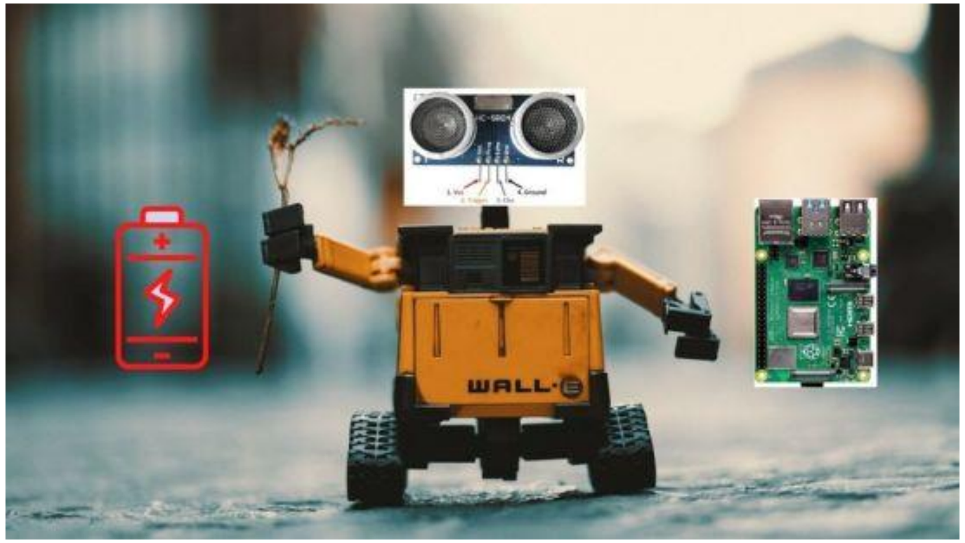
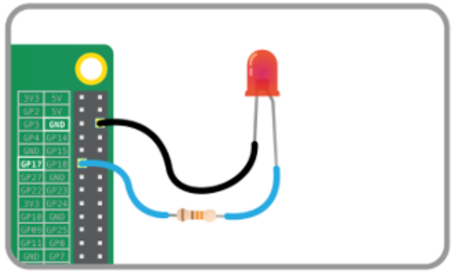
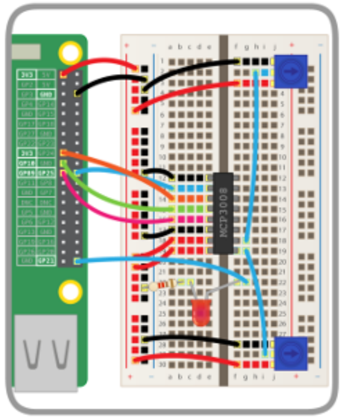



INTERNET OF THINGS (IOT) PROJECTS USING PYTHON

(CSE 4110)

(LECTURE – 8)

Why do we need Wi-Fi module?

Wi-Fi modules are used to provide internet connection to robotic and electronic projects. Wi-Fi modules allow developing IoT (Internet of Things) projects. By using Wi-Fİ modules you can send data over the internet to your robot or make it send data over the internet.

WiFi module, also known as serial to WIFI module, which belongs to the transmission layer of IoT. The function is to convert serial port or TTL level into embedded module which can conforming to WiFi wireless network communication standard, with built-in wireless network protocol IEEE802.

What does the Wi-Fi module have?

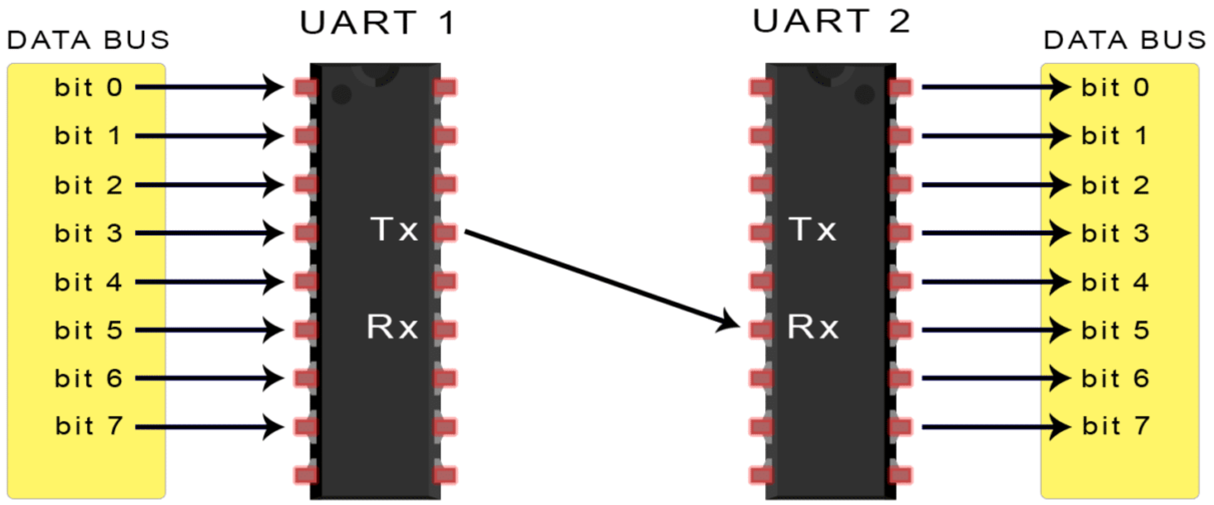
The Wi-Fi module generally contains two main parts: a Wi-Fi chip and an application host processor. The Wi-Fi subsystem includes an 802.11 radio physical layer (PHY), baseband, media access control (MAC), and perhaps a crypto engine for fast, secure Internet connection.

Important Terminology

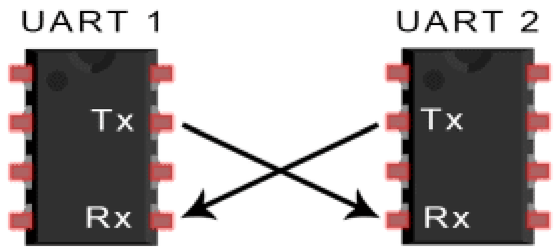
UART Communication

UART stands for Universal Asynchronous Receiver/Transmitter. It's not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone IC. A UART's main purpose is to transmit and receive serial data.

One of the best things about UART is that it only uses two wires to transmit data between devices.



In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a controlling device like a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART:



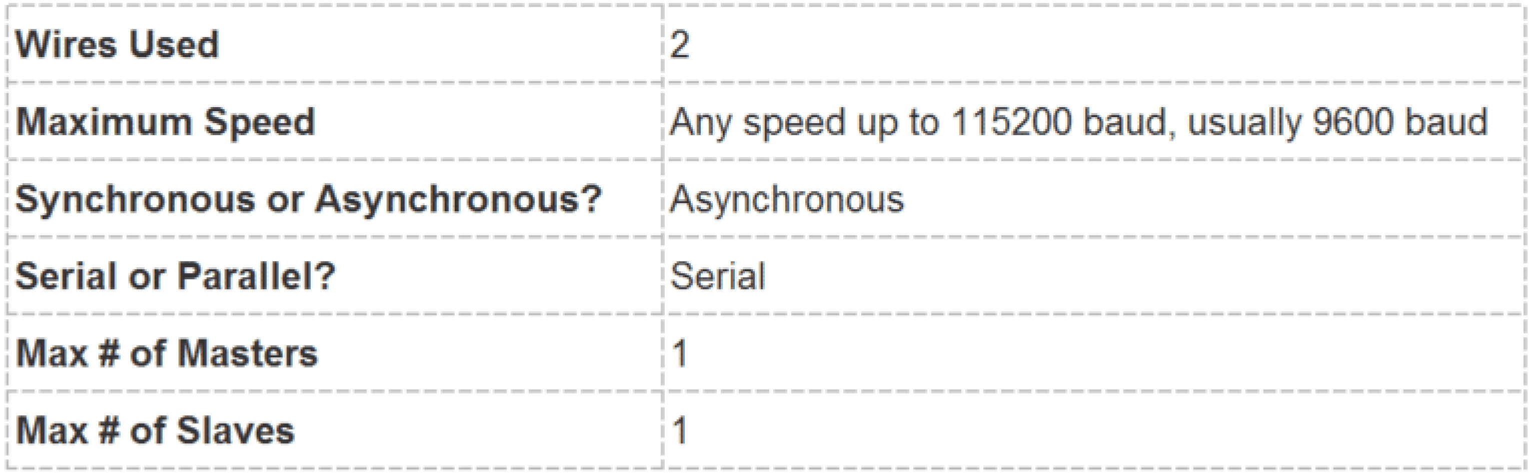
UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART. Instead of a clock signal, the transmitting UART adds start and stop bits to the data packet being transferred. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits.

Important Terminology

Baud Rate

When the receiving UART detects a start bit, it starts to read the incoming bits at a specific frequency known as the baud rate. Baud rate is a measure of the speed of data transfer, expressed in bits per second (bps). Both UARTs must operate at about the same baud rate. The baud rate between the transmitting and receiving UARTs can only differ by about 10% before the timing of bits gets too far off.

Both UARTs must also must be configured to transmit and receive the same data packet structure.

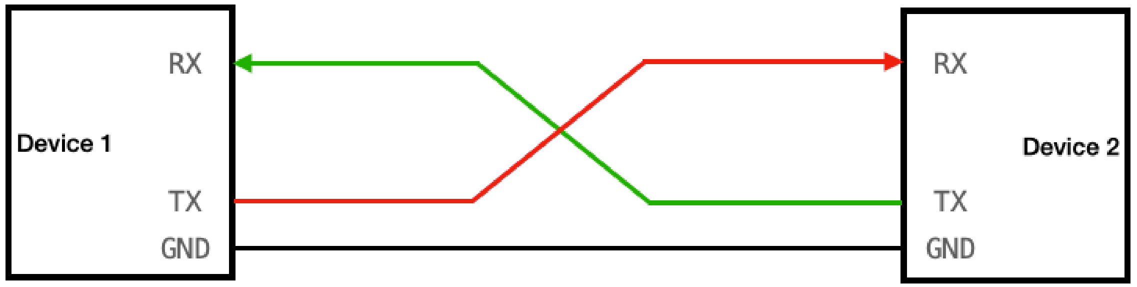


Wires Used	2
Maximum Speed	Any speed up to 115200 baud, usually 9600 baud
Synchronous or Asynchronous?	Asynchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	1

Default Baud Rates
Esp01 : 115200
Arduino: 9600

Baud Rate

UART



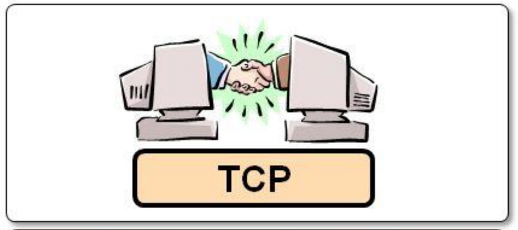
Important Terminology

TCP/IP Protocol

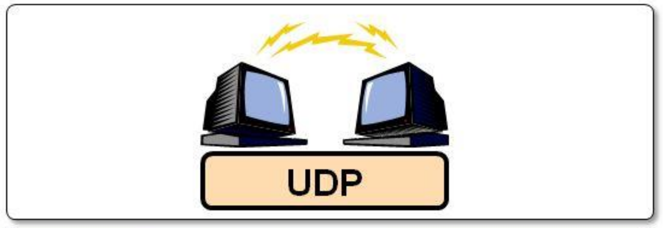
Transmission Control Protocol (TCP) widely known as Internet Protocol (IP) is a set of communications protocols used over the internet for delivery of services or packets within or across the network. It is commonly known as internet protocol suite.

UDP Protocol

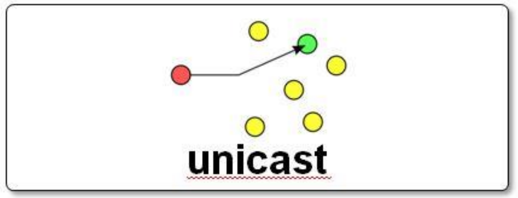
User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an unreliable and connectionless protocol. So, there is no need to establish a connection prior to data transfer. The UDP helps to establish low-latency and loss-tolerating connections establish over the network. The UDP enables process to process communication.

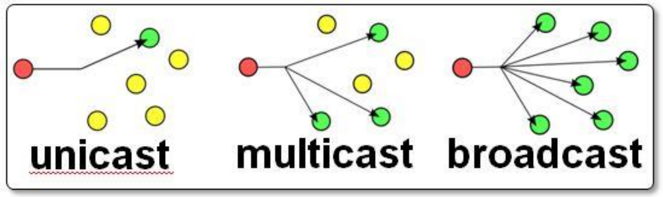


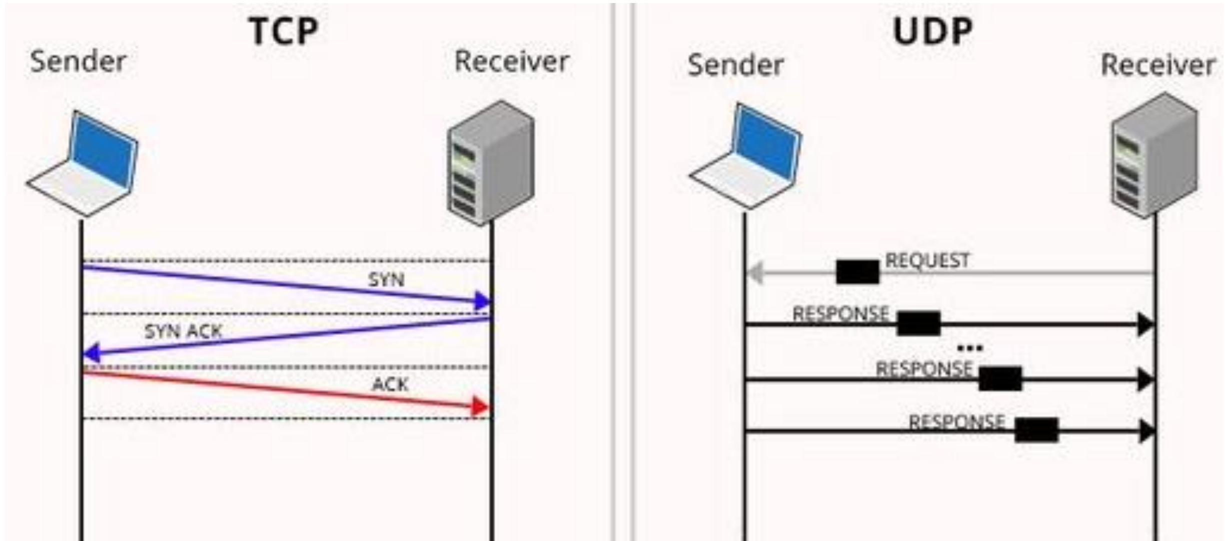
- Slower but reliable transfers
- Typical applications:
  - Email
  - Web browsing



- Fast but non-guaranteed transfers ("best effort")
- Typical applications:
  - VoIP
  - Music streaming







Important Terminology

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.

AT: This type of command is used to test the startup function of WiFi module. The response would be ok, against this command if everything is ok.

AT+GMR : This type of AT command is used to check the version of AT command and we used SDK version of AT command in this type of WIFI module.

AT+CIPSERVER=0: This configures the ESP8266 as server and sets the mode as 0 which means delete server (need to follow by restart)

AT+RST: This type of command is used for reset the WiFi module when it is in working condition. The response would be ok, when reset the module.

AT+RESTORE: This type of command is used to restore factory settings means, when this command is entered then all the parameters are reset automatically to default one's.

AT+CWMODE? : This type of command is used to query the WiFi mode of ESP8266.

AT+CWMODE=1 : This sets the WiFi mode of ESP8266 in this case in station mode.

AT+CWJAP="SSID","PASSWORD"\r\n', timeout=TIME_ms : This connects the ESP8266 with an AP whose SSID and password are given, The timeout here is the reconnection time.

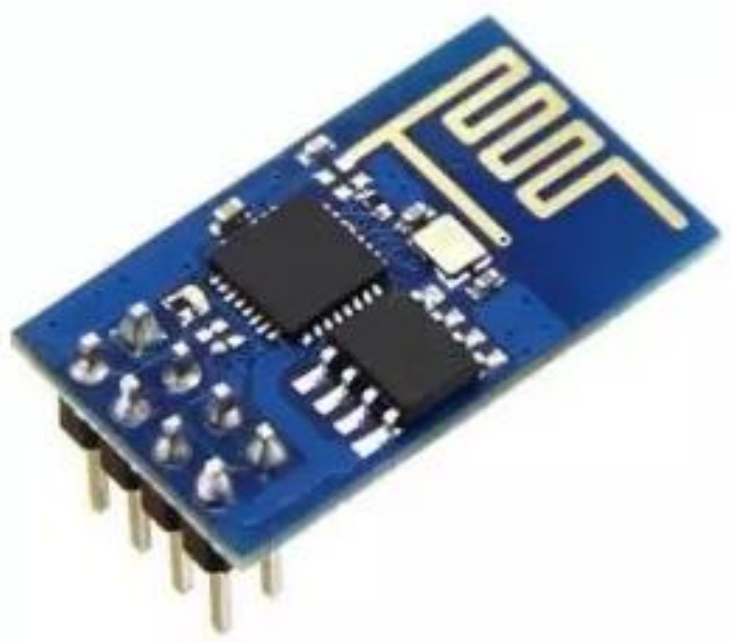
Best Wi-Fi module?

The ESP-01 is a popular, inexpensive microcontroller board with built-in Wi-Fi. It is a breakout board that makes use of the, now, widely used ESP8266 microcontroller chip.

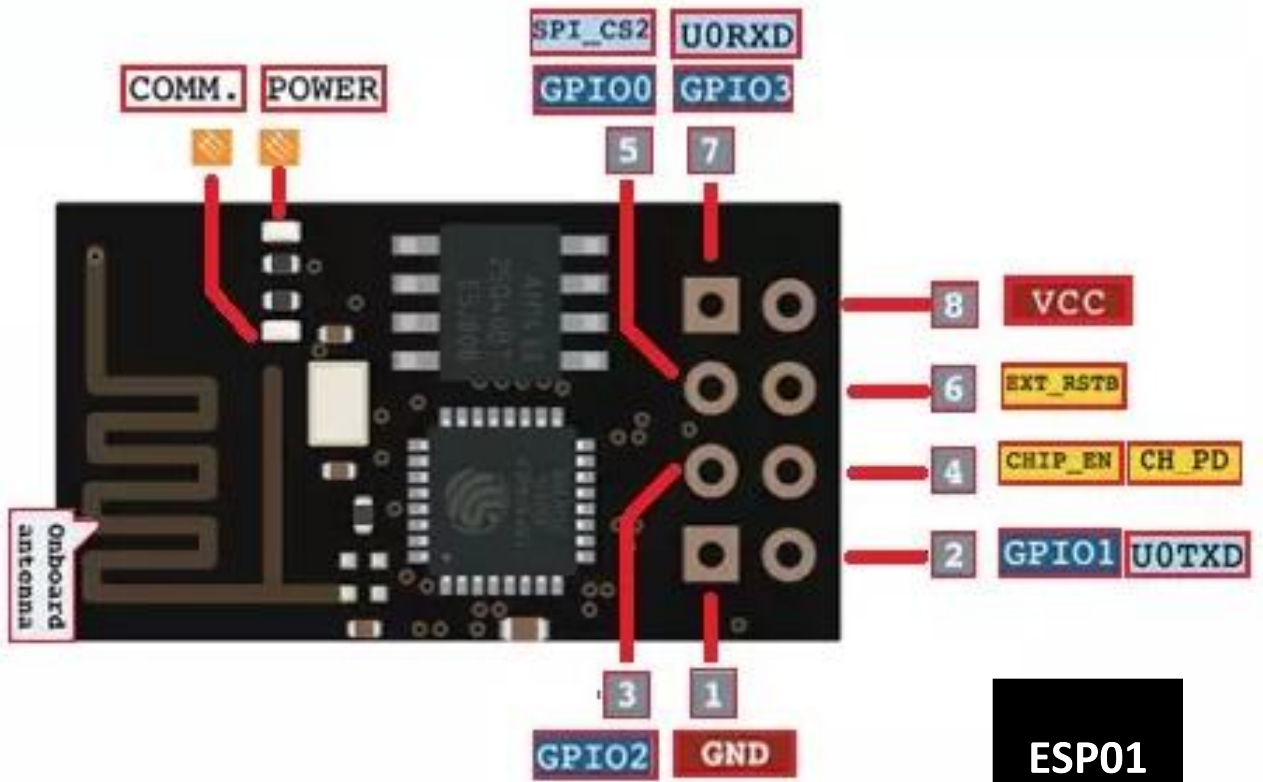
Raspberry Pi Pico RP2040 board features a dual-core Arm Cortex-M0+ processor with 264KB internal RAM & up to 16MB of off-chip Flash. The RP2040 is much faster compared to Arduino but it has a limitation of wireless network connectivity. This is the reason why Raspberry Pi Pico alone can’t be used for wireless & IoT-related applications. Hence a low-cost ESP8266 WiFi module could be a good choice to add wireless connectivity to Raspberry Pi Pico.

The set of AT commands is preloaded into the program memory of ESP8266 and does not require additional programming. We will be sending these specific commands through the serial port (UART) of Raspberry Pi Pico. The coding is done in MicroPython & the device is programmed using the Thonny IDE. The Web Server displays the on-chip temperature sensor reading on a web page running on the web browser.

ESP - 01Wi-Fi module



ESP-01 Module

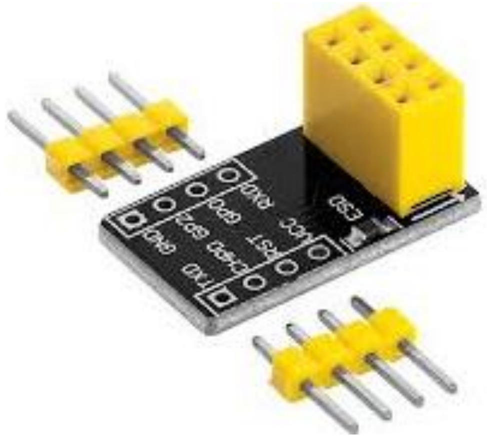


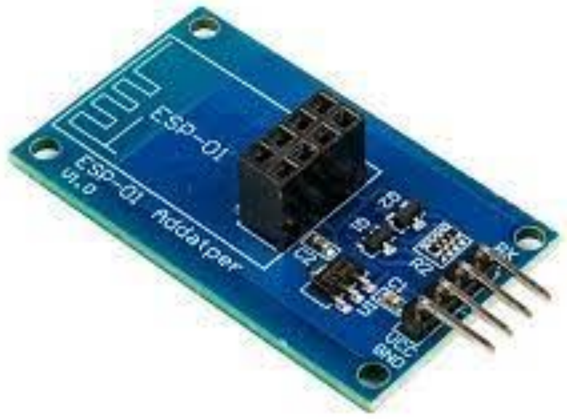
ESP01 Pinout

ESP-01 is an inexpensive, small-sized WiFi module, which consists of TCP/IP stack along with a built-in microcontroller. So, we can directly program this small chip and can bring WiFi capability in our Embedded projects.

1. GND - Ground
2. GPIO2 - General Purpose Input/Output
3. GPIO0 - General Purpose Input/Output (Used for boot mode too)
4. UXRXD - Receiver (for serial communication)
5. U0TXD - Transmitter (for serial communication)
6. CH_PD - Chip powerdown
7. EXT_RSTB - Reset
8. VCC - 3.3v input voltage

ESP 01 Adapter

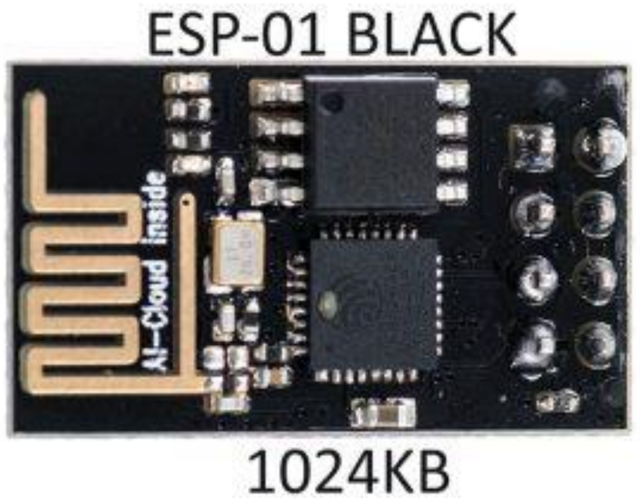




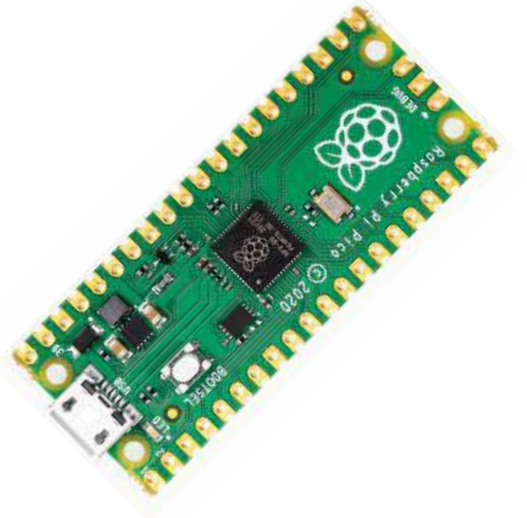
ESP-01 breadboard adapter.

ESP-01 USB-to-serial programming adapter

ESP 01 Available in market

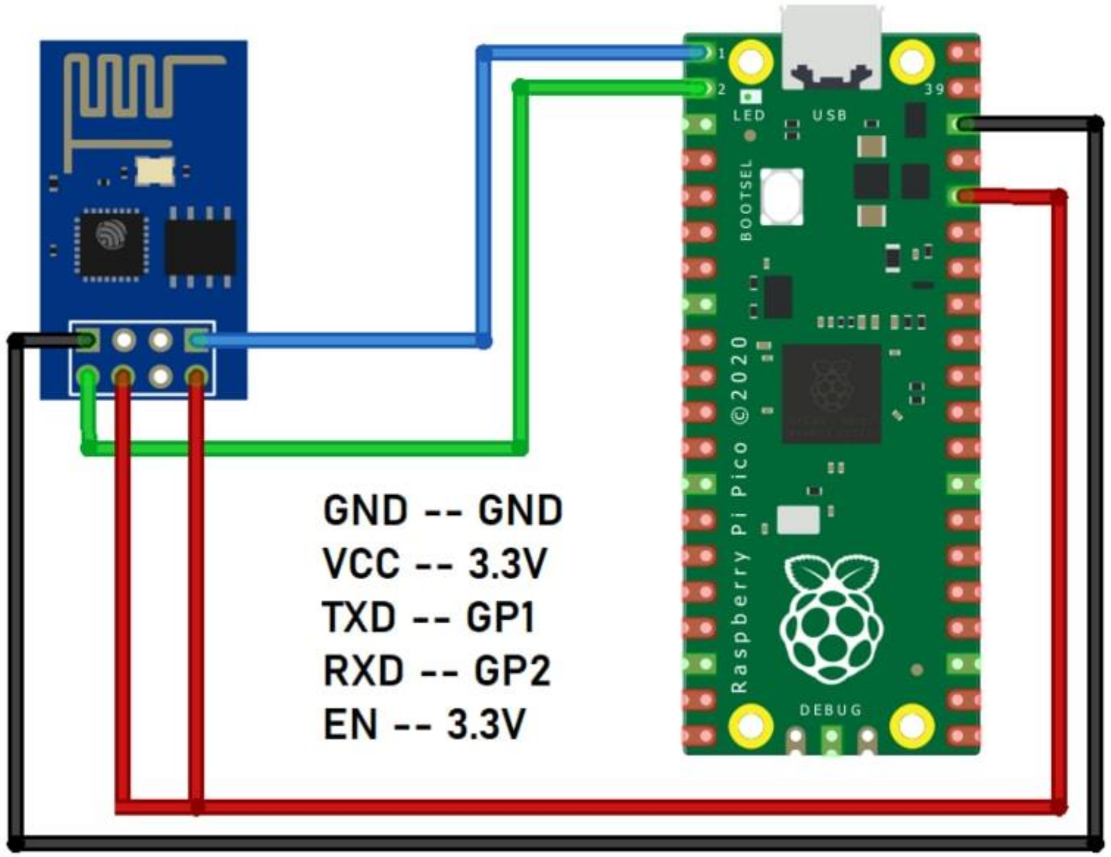


Runs only on 3.3 V

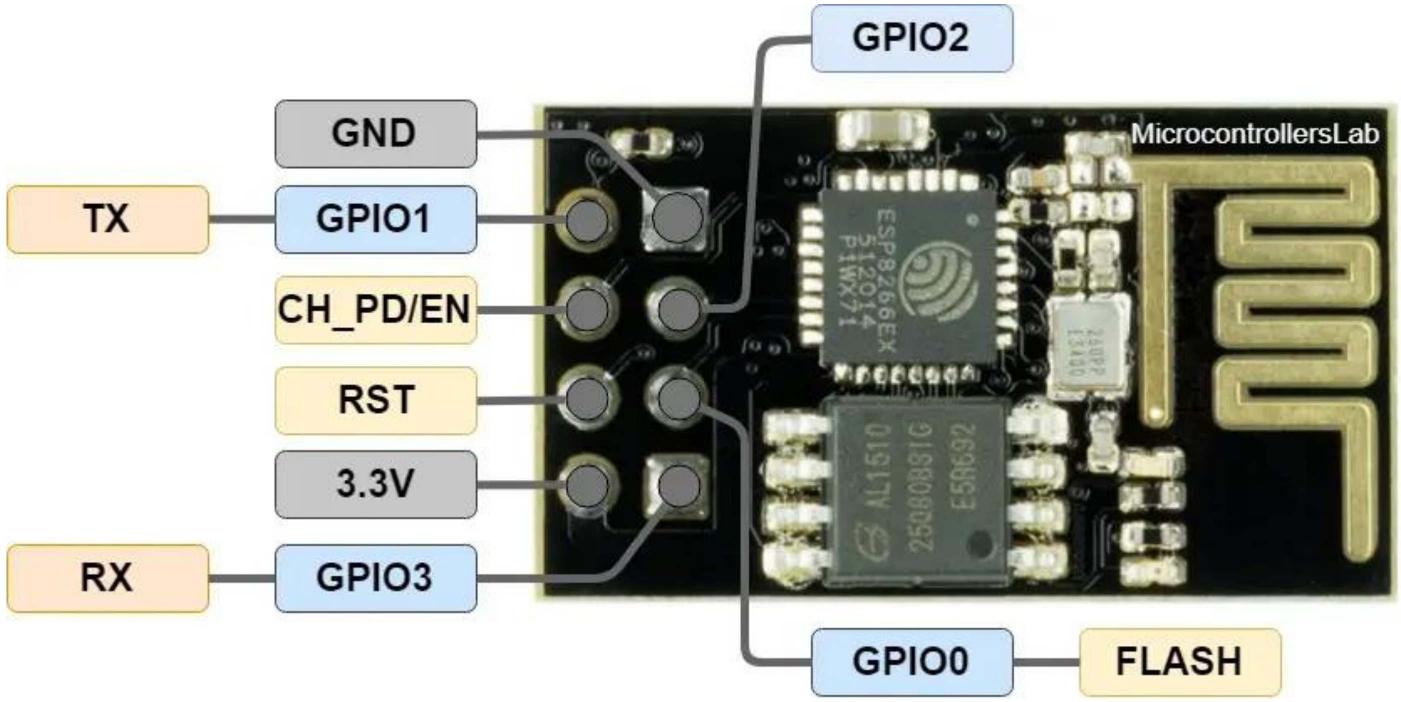


Best Choice

Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

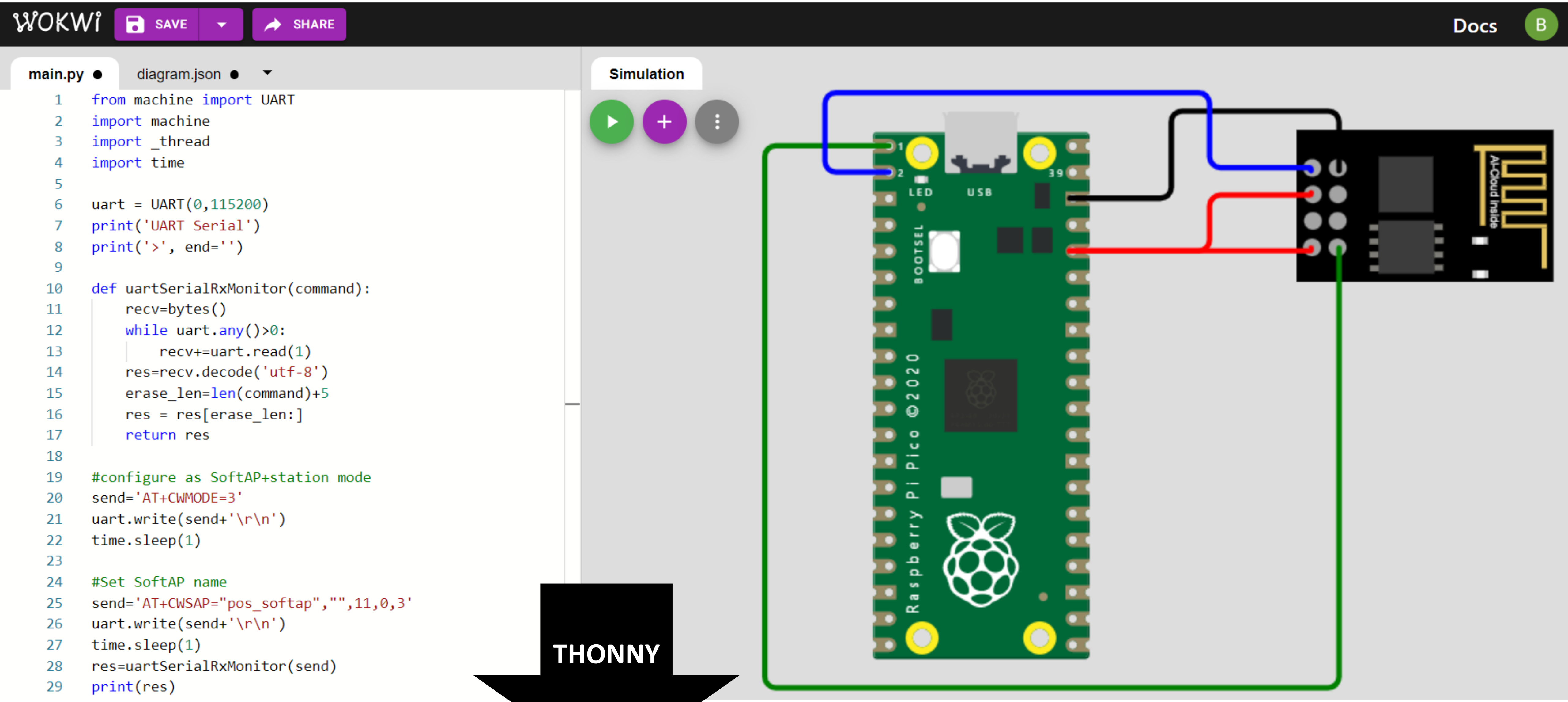


The Raspberry Pi Pico has two inbuilt UART. In this design, we will use UART-0. Similarly, the default baud rate of the ESP8266 is 115200. We need to configure the Raspberry Pi Pico with the same baud rate in order to maintain synchronization with the ESP8266.



The ESP8266 Module is not capable of 5-3V logic shifting and will require an external Logic Level Converter. Please do not power it directly from your 5V dev board.

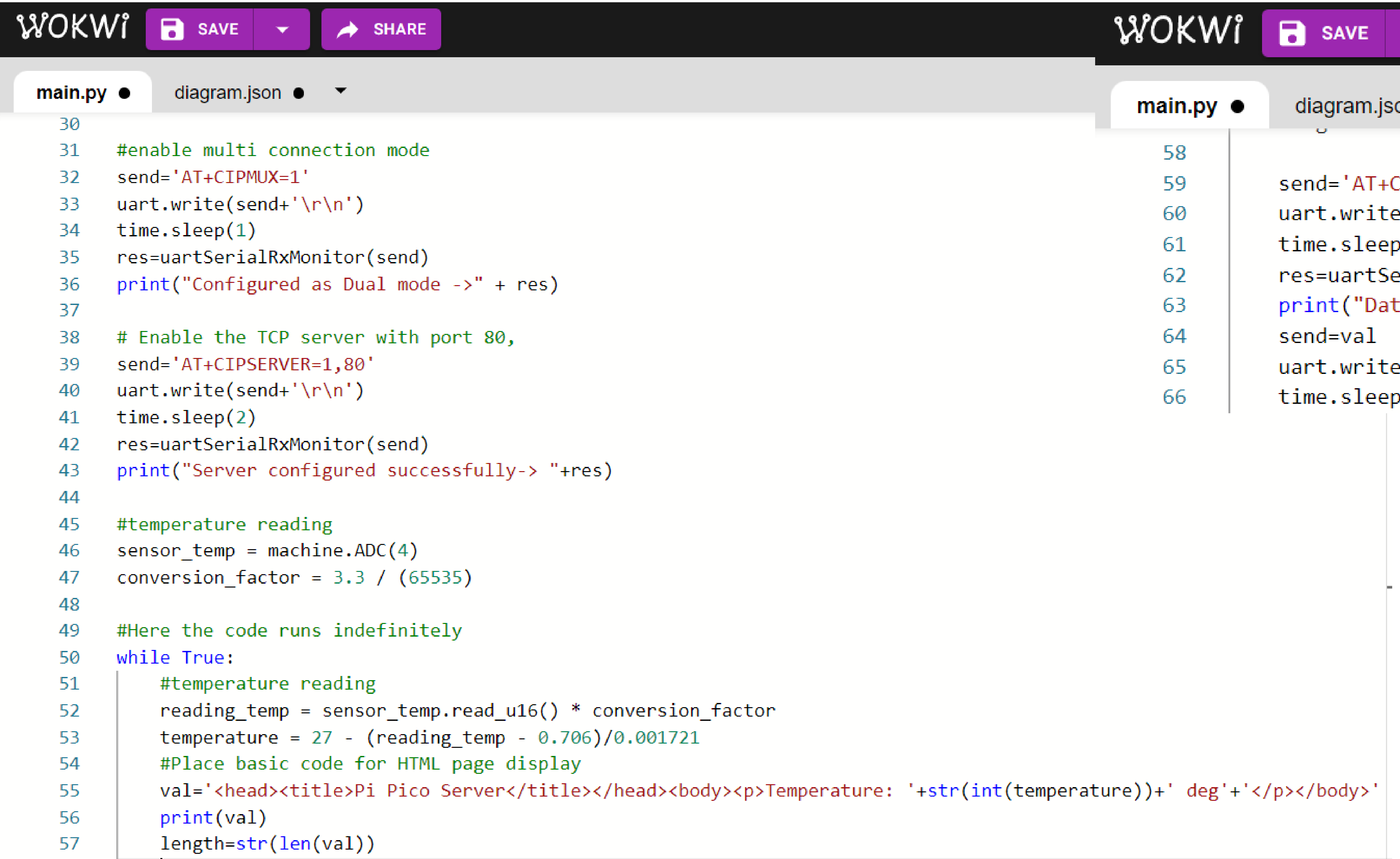
Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico



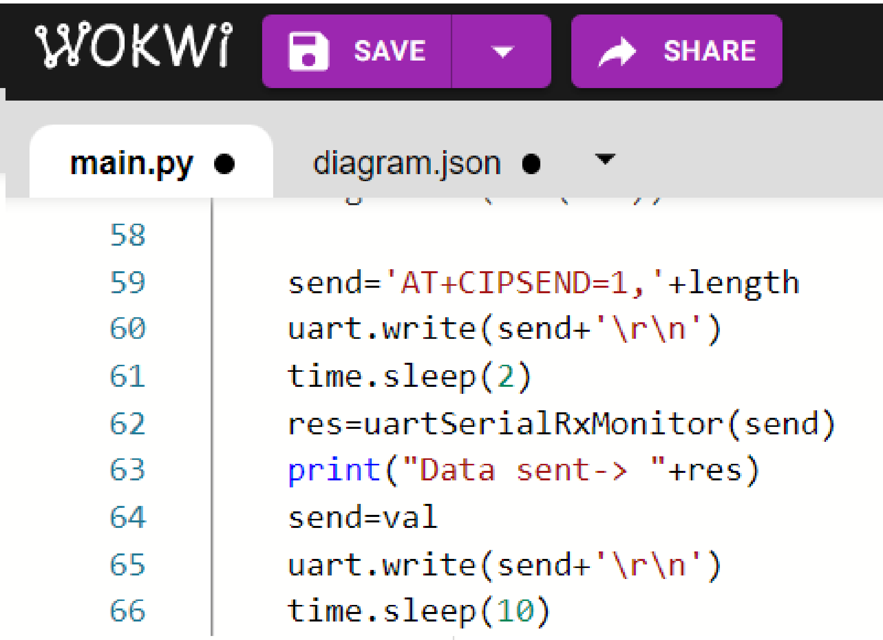
```python
from machine import UART
import machine
import _thread
import time

uart = UART(0,115200)
print('UART Serial')
print('>', end='')

def uartSerialRxMonitor(command):
    recv=bytes()
    while uart.any()>0:
        recv+=uart.read(1)
    res=recv.decode('utf-8')
    erase_len=len(command)+5
    res = res[erase_len:]
    return res

#configure as SoftAP+station mode
send='AT+CWMODE=3'
uart.write(send+'\r\n')
time.sleep(1)

#Set SoftAP name
send='AT+CWSAP="pos_softap","",11,0,3'
uart.write(send+'\r\n')
time.sleep(1)
res=uartSerialRxMonitor(send)
print(res)
```

THONNY

Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico

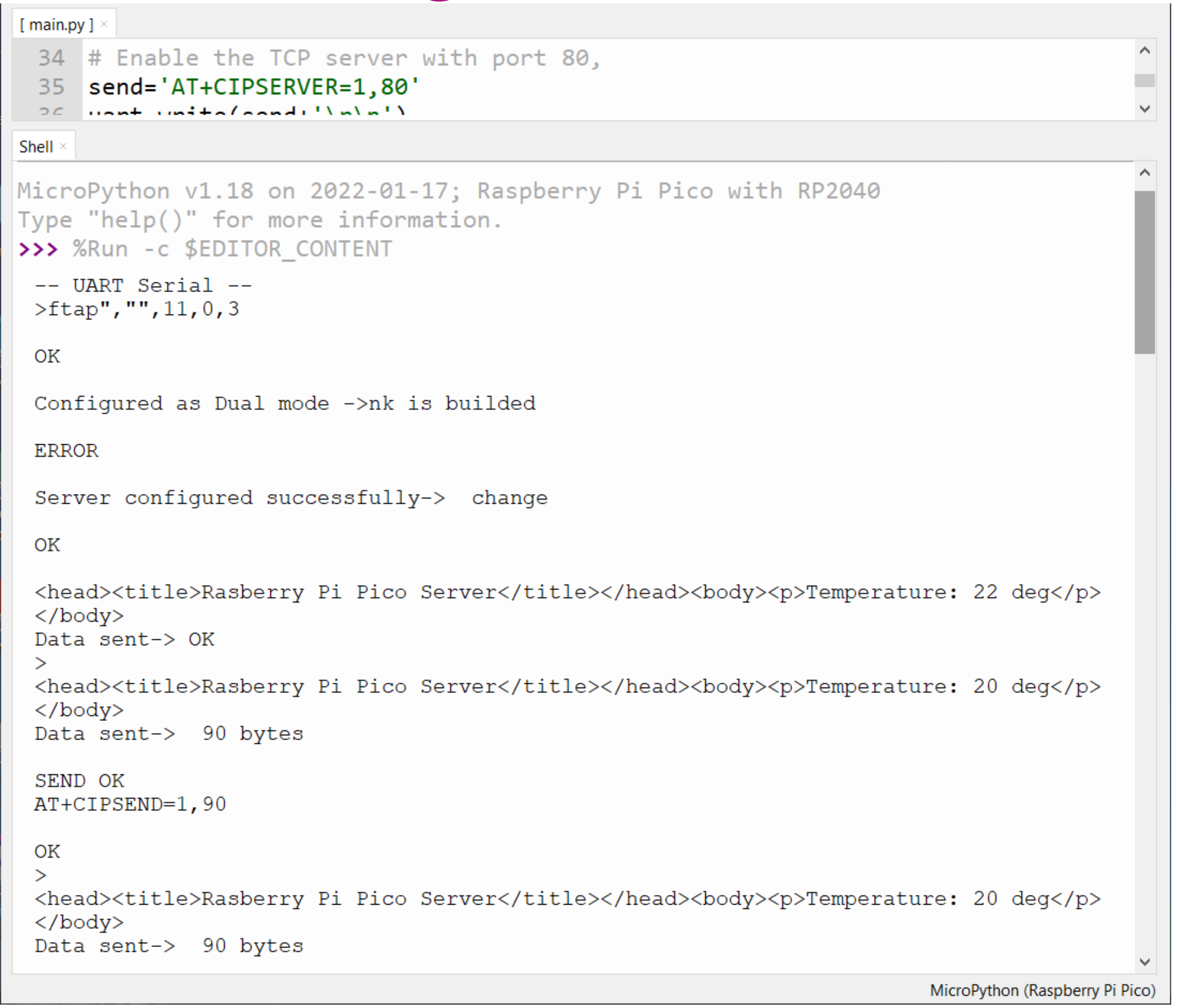


```python

#enable multi connection mode
send='AT+CIPMUX=1'
uart.write(send+'\r\n')
time.sleep(1)
res=uartSerialRxMonitor(send)
print("Configured as Dual mode ->" + res)

# Enable the TCP server with port 80,
send='AT+CIPSERVER=1,80'
uart.write(send+'\r\n')
time.sleep(2)
res=uartSerialRxMonitor(send)
print("Server configured successfully-> "+res)

#temperature reading
sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

#Here the code runs indefinitely
while True:
    #temperature reading
    reading_temp = sensor_temp.read_u16() * conversion_factor
    temperature = 27 - (reading_temp - 0.706)/0.001721
    #Place basic code for HTML page display
    val='<head><title>Pi Pico Server</title></head><body><p>Temperature: '+str(int(temperature))+' deg'+'</p></body>'
    print(val)
    length=str(len(val))
```



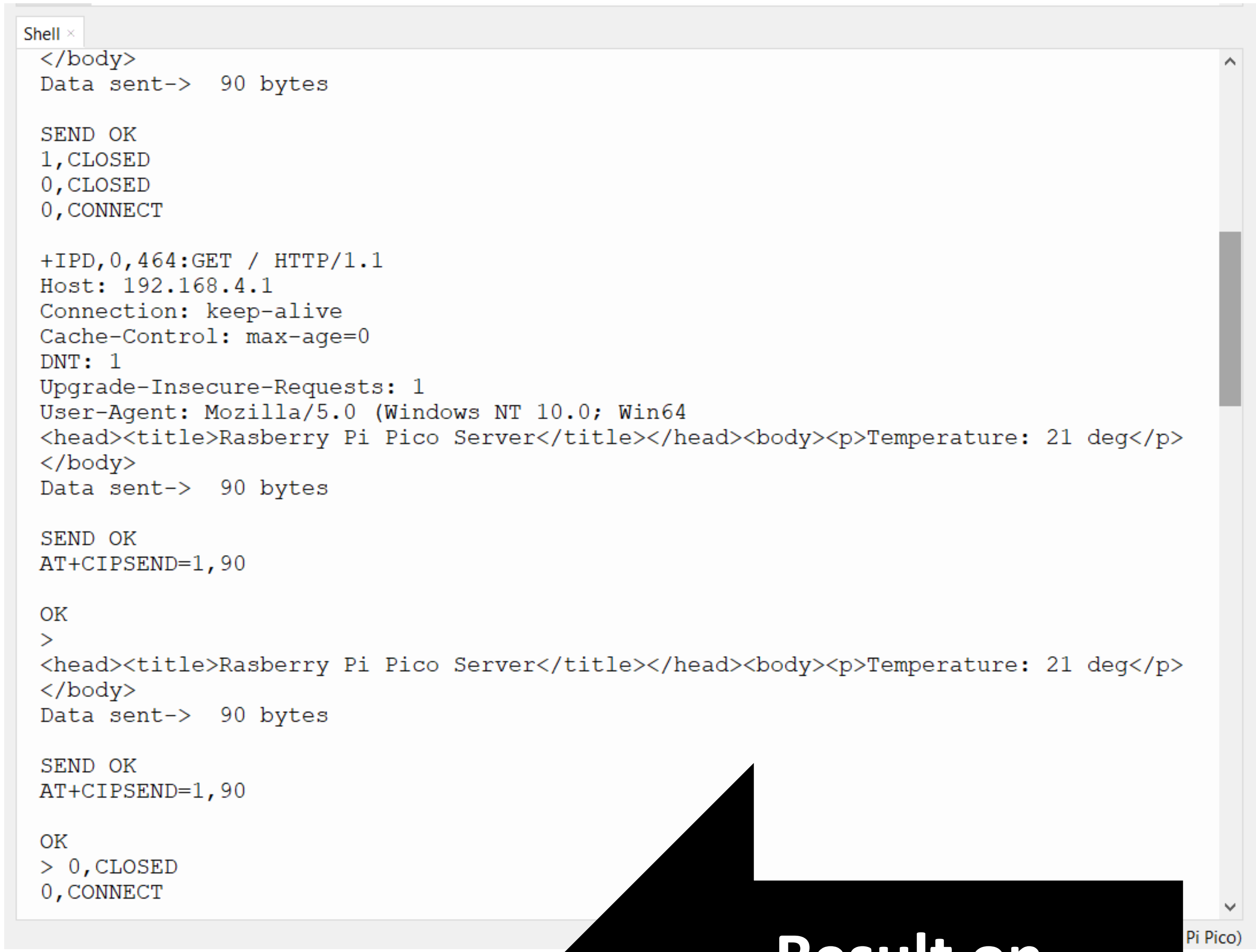
```python

    send='AT+CIPSEND=1,'+length
    uart.write(send+'\r\n')
    time.sleep(2)
    res=uartSerialRxMonitor(send)
    print("Data sent-> "+res)
    send=val
    uart.write(send+'\r\n')
    time.sleep(10)
```

Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico



```
# Enable the TCP server with port 80,
send='AT+CIPSERVER=1,80'
```

```
MicroPython v1.18 on 2022-01-17; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

 -- UART Serial --
 >ftap","",11,0,3

 OK

 Configured as Dual mode ->nk is builded

 ERROR

 Server configured successfully->  change

 OK

 <head><title>Rasberry Pi Pico Server</title></head><body><p>Temperature: 22 deg</p>
 </body>
 Data sent-> OK
 >
 <head><title>Rasberry Pi Pico Server</title></head><body><p>Temperature: 20 deg</p>
 </body>
 Data sent->  90 bytes

 SEND OK
 AT+CIPSEND=1,90

 OK
 >
 <head><title>Rasberry Pi Pico Server</title></head><body><p>Temperature: 20 deg</p>
 </body>
 Data sent->  90 bytes
```

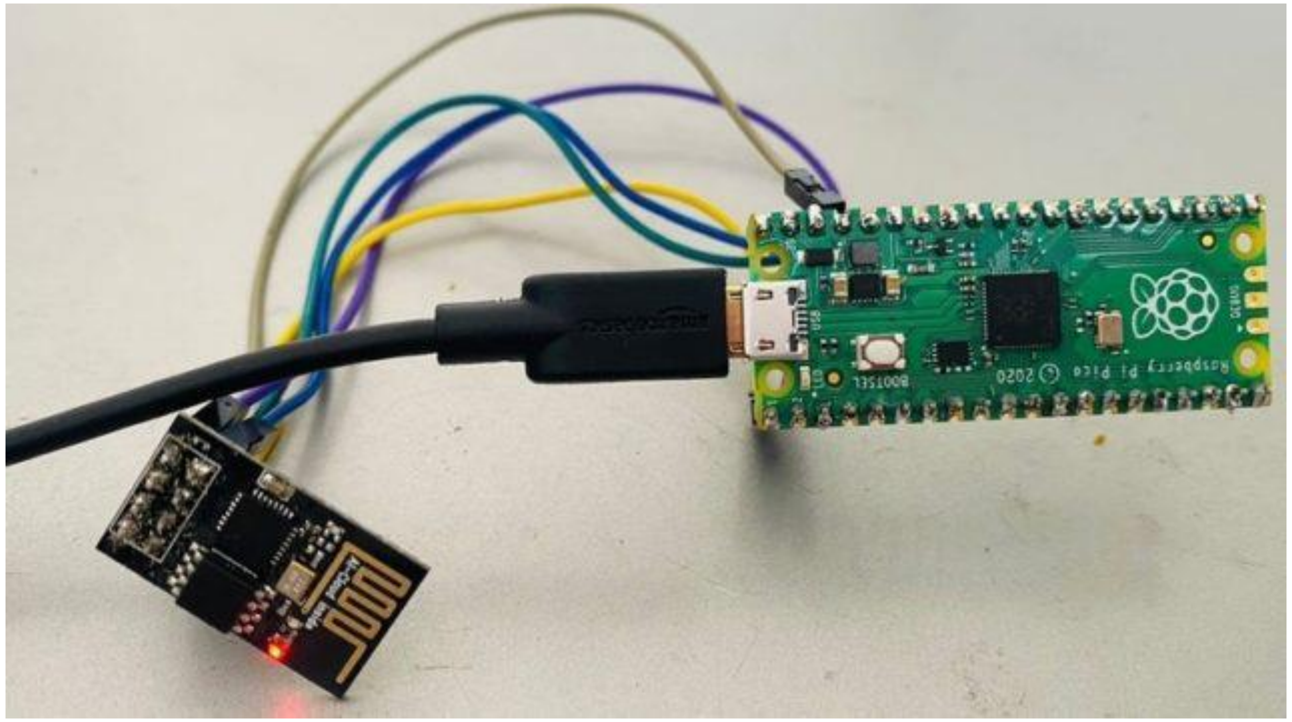


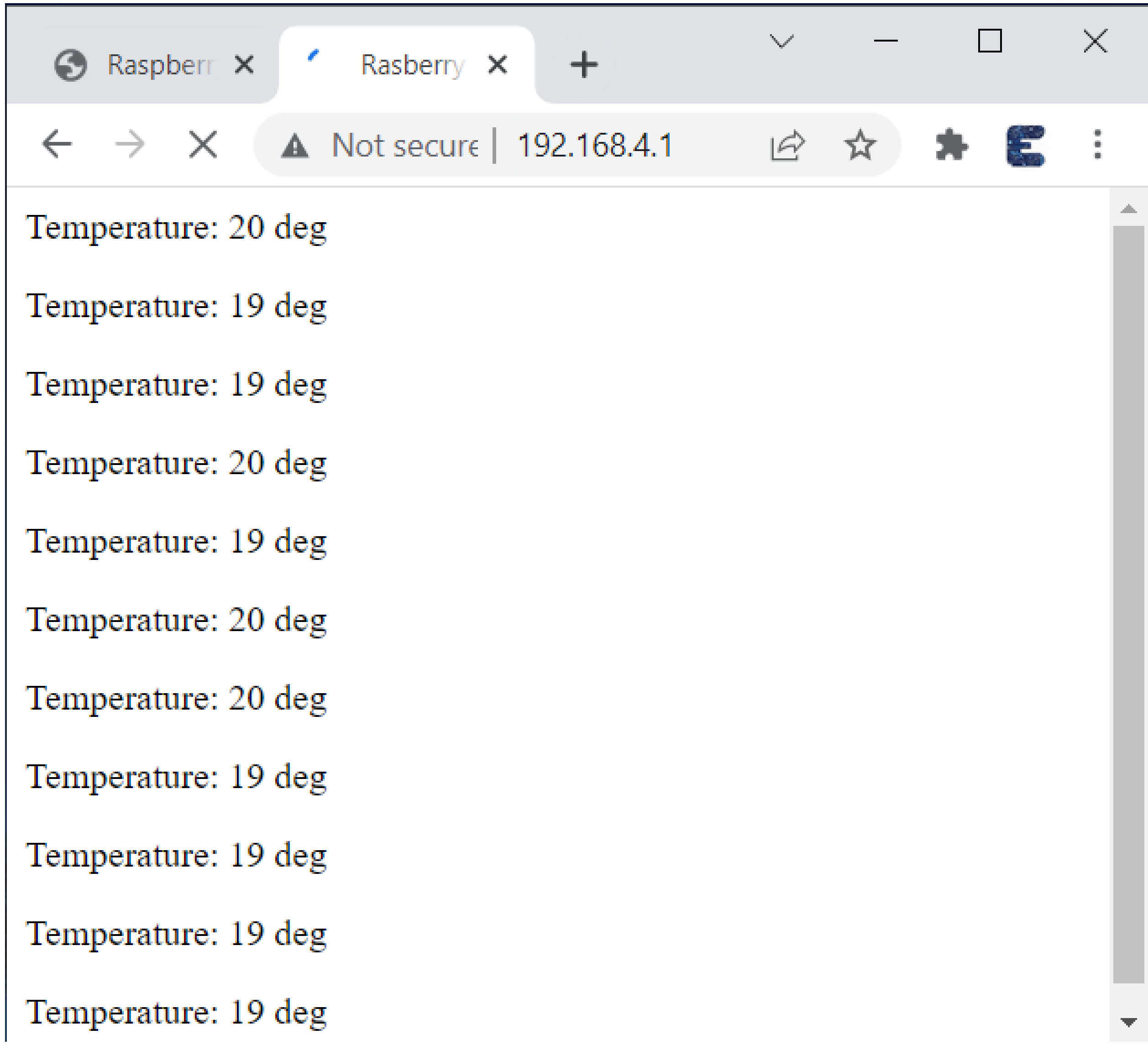
```
</body>
Data sent->  90 bytes

SEND OK
1,CLOSED
0,CLOSED
0,CONNECT

+IPD,0,464:GET / HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Cache-Control: max-age=0
DNT: 1
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64
<head><title>Rasberry Pi Pico Server</title></head><body><p>Temperature: 21 deg</p>
</body>
Data sent->  90 bytes

SEND OK
AT+CIPSEND=1,90

OK
>
<head><title>Rasberry Pi Pico Server</title></head><body><p>Temperature: 21 deg</p>
</body>
Data sent->  90 bytes

SEND OK
AT+CIPSEND=1,90

OK
> 0,CLOSED
0,CONNECT
```

Result on THONNY

Then open Google Chrome, type 192.168.4.1 in the address field. Then press Enter. The browser will start to display the temperature data. Since the connection of the server is not closed, it will continuously print data on the page.

Interfacing of ESP8266 WiFi Module with Raspberry Pi Pico





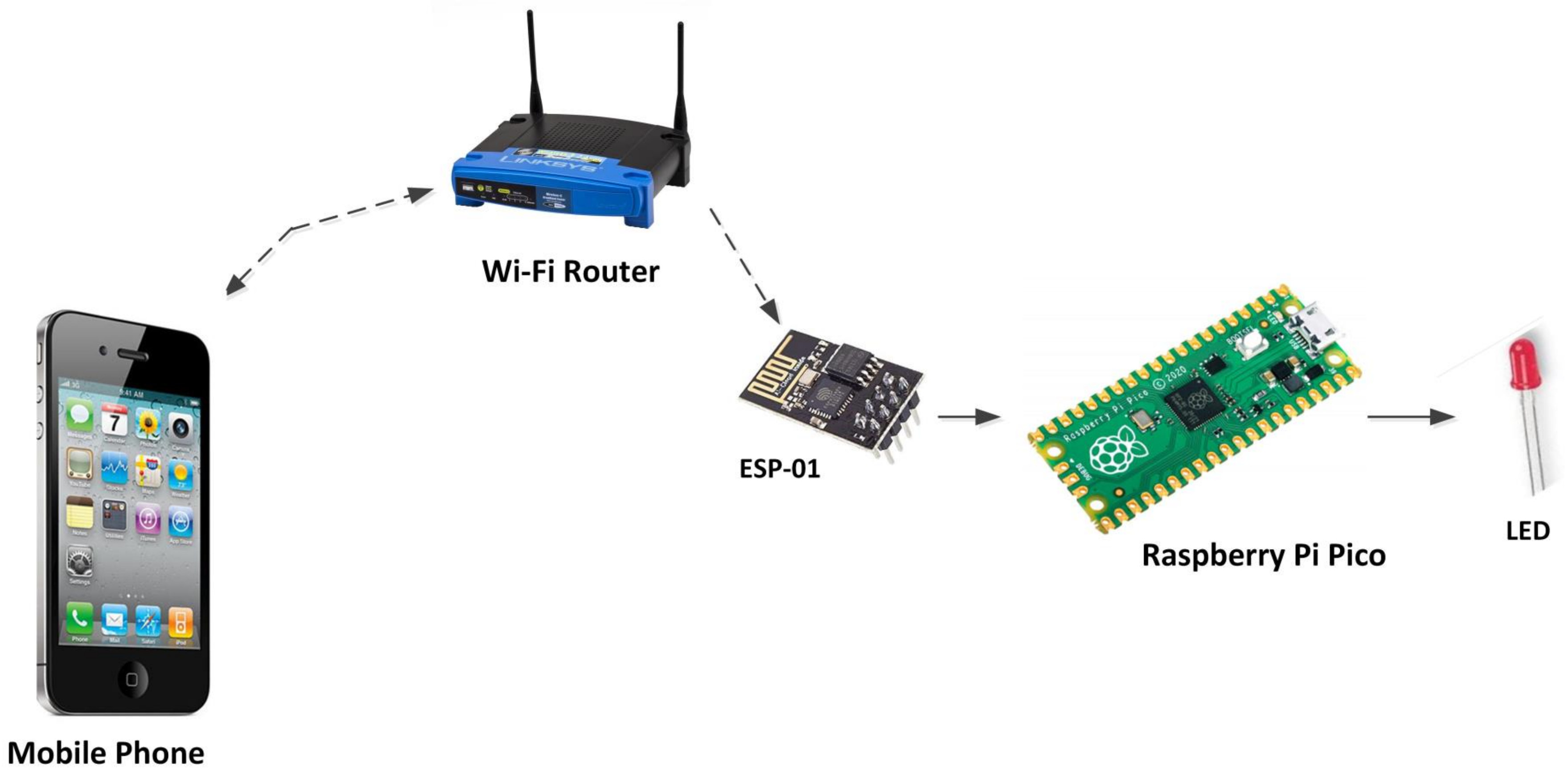
```
Temperature: 20 deg

Temperature: 19 deg

Temperature: 19 deg

Temperature: 20 deg

Temperature: 19 deg

Temperature: 20 deg

Temperature: 20 deg

Temperature: 19 deg

Temperature: 19 deg

Temperature: 19 deg

Temperature: 19 deg
```

Controlling an LED from a Smartphone Using Wi-Fi

Objective:

Sending commands over the Wi-Fi link from a mobile phone to control an LED (the LED can be replaced with a relay, for example, to control a piece of equipment) connected to the Raspberry Pi Pico. Commands must be terminated with a Return (CR/LF or 'newline'). Valid commands include:

LON	Turn LED ON
LOFF	Turn LED OFF



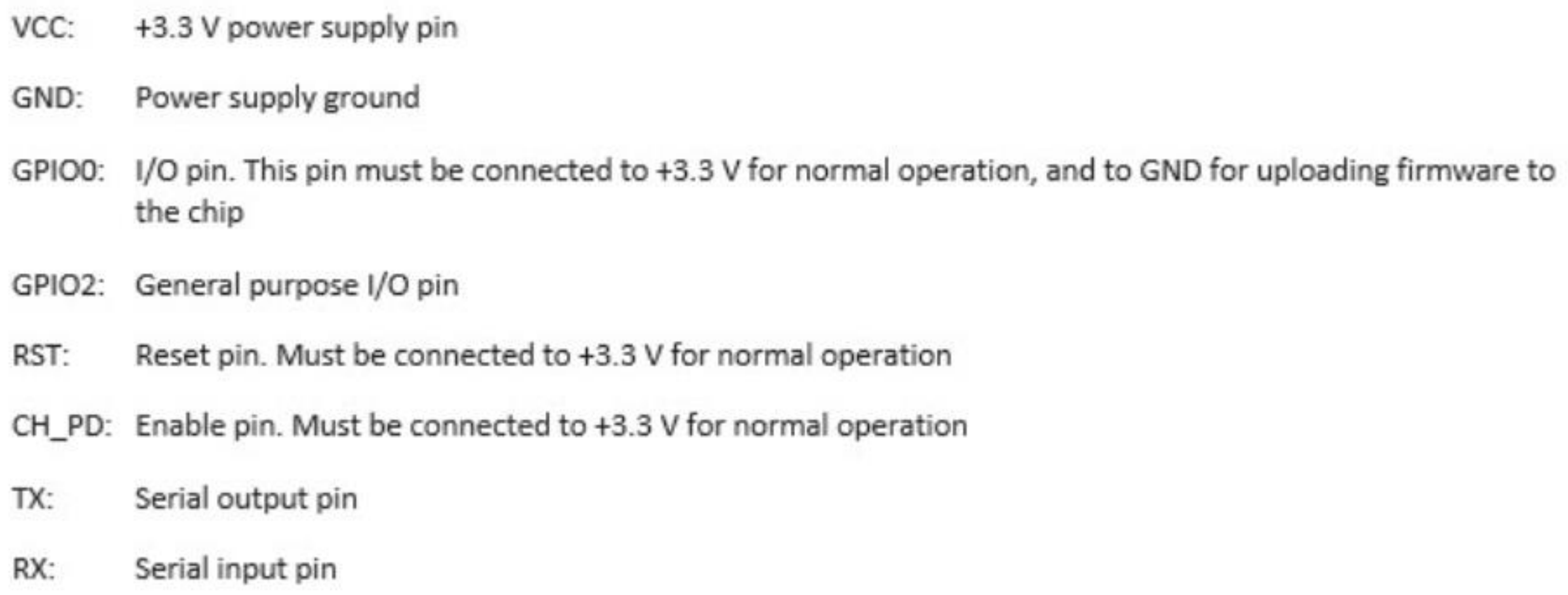
Pico Wi-Fi Connectivity

Objective:

Sending commands over the Wi-Fi link from a mobile phone to control an LED (the LED can be replaced with a relay, for example, to control a piece of equipment) connected to the Raspberry Pi Pico. Commands must be terminated with a Return (CR/LF or ‘newline’). Valid commands include:

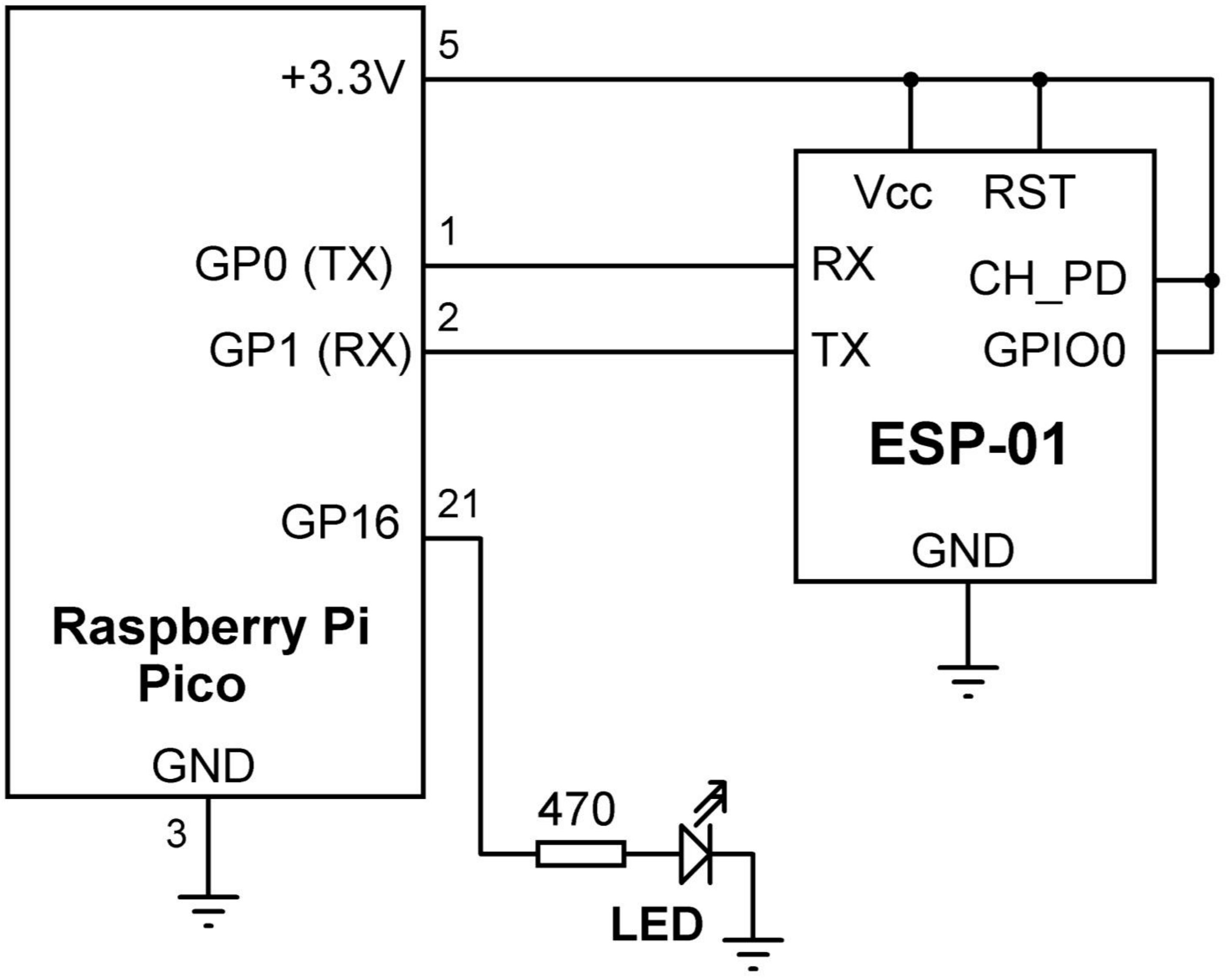
LON	Turn LED ON
LOFF	Turn LED OFF

The ESP-01 communicates with the host processor through its TX and RX serial port pin. It is an 8-pin board with pin names as follows:

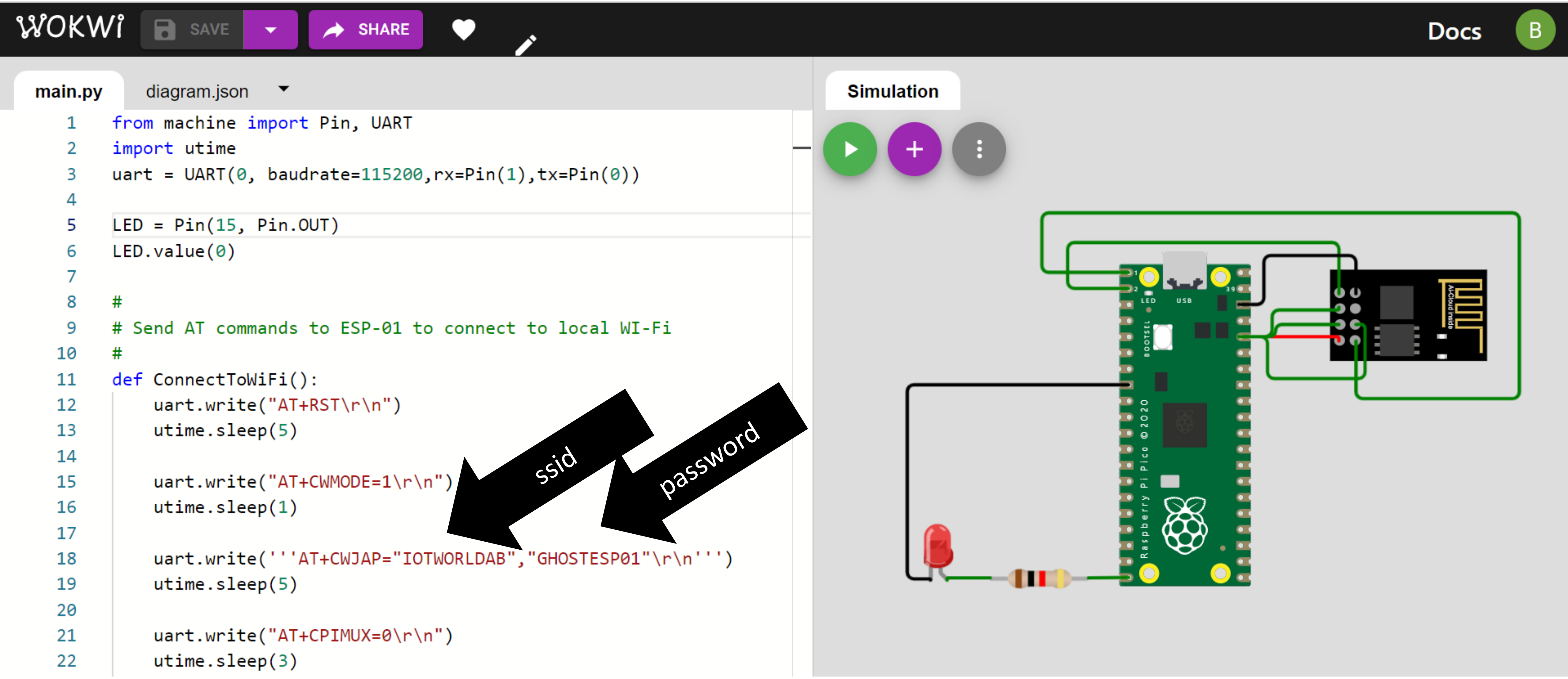


VCC:	+3.3 V power supply pin
GND:	Power supply ground
GPIO0:	I/O pin. This pin must be connected to +3.3 V for normal operation, and to GND for uploading firmware to the chip
GPIO2:	General purpose I/O pin
RST:	Reset pin. Must be connected to +3.3 V for normal operation
CH_PD:	Enable pin. Must be connected to +3.3 V for normal operation
TX:	Serial output pin
RX:	Serial input pin

Controlling an LED from a Smartphone Using Wi-Fi

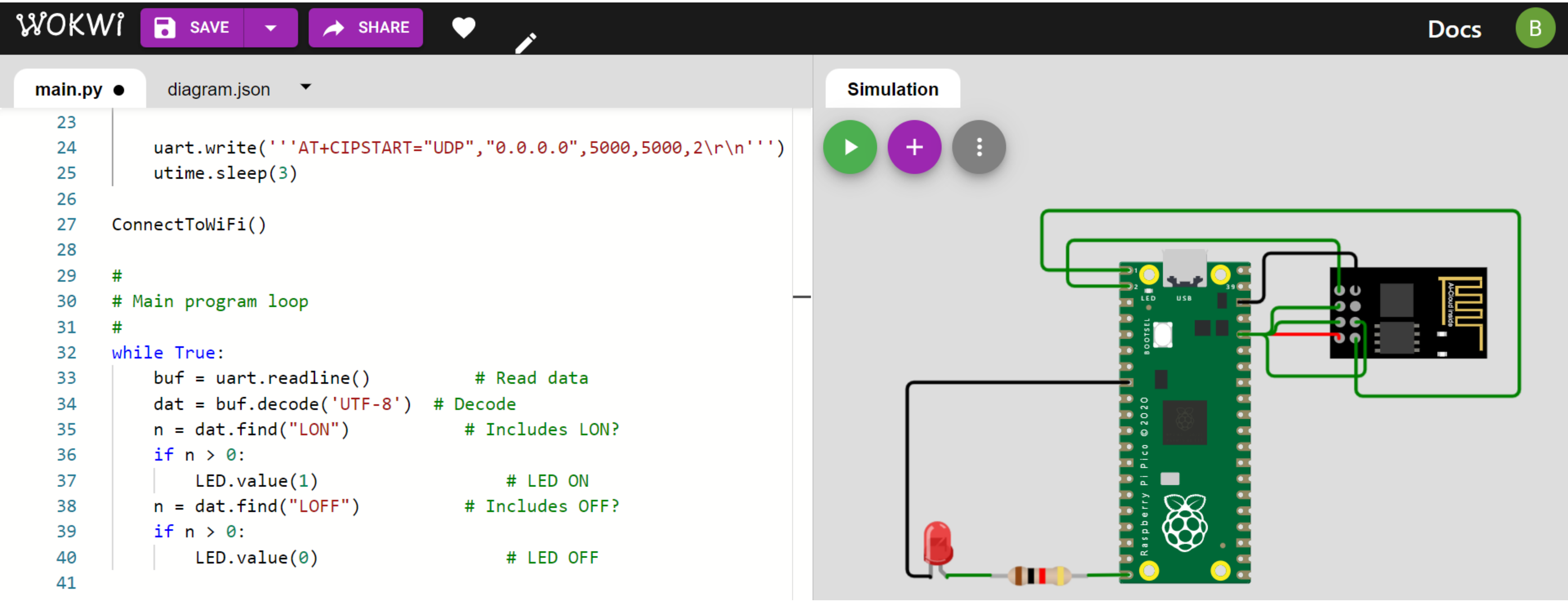


Controlling an LED from a Smartphone Using Wi-Fi



```python
from machine import Pin, UART
import utime
uart = UART(0, baudrate=115200,rx=Pin(1),tx=Pin(0))

LED = Pin(15, Pin.OUT)
LED.value(0)

#
# Send AT commands to ESP-01 to connect to local WI-Fi
#
def ConnectToWiFi():
    uart.write("AT+RST\r\n")
    utime.sleep(5)

    uart.write("AT+CWMODE=1\r\n")
    utime.sleep(1)

    uart.write('''AT+CWJAP="IOTWORLDAB","GHOSTESP01"\r\n''')
    utime.sleep(5)

    uart.write("AT+CPIMUX=0\r\n")
    utime.sleep(3)
```

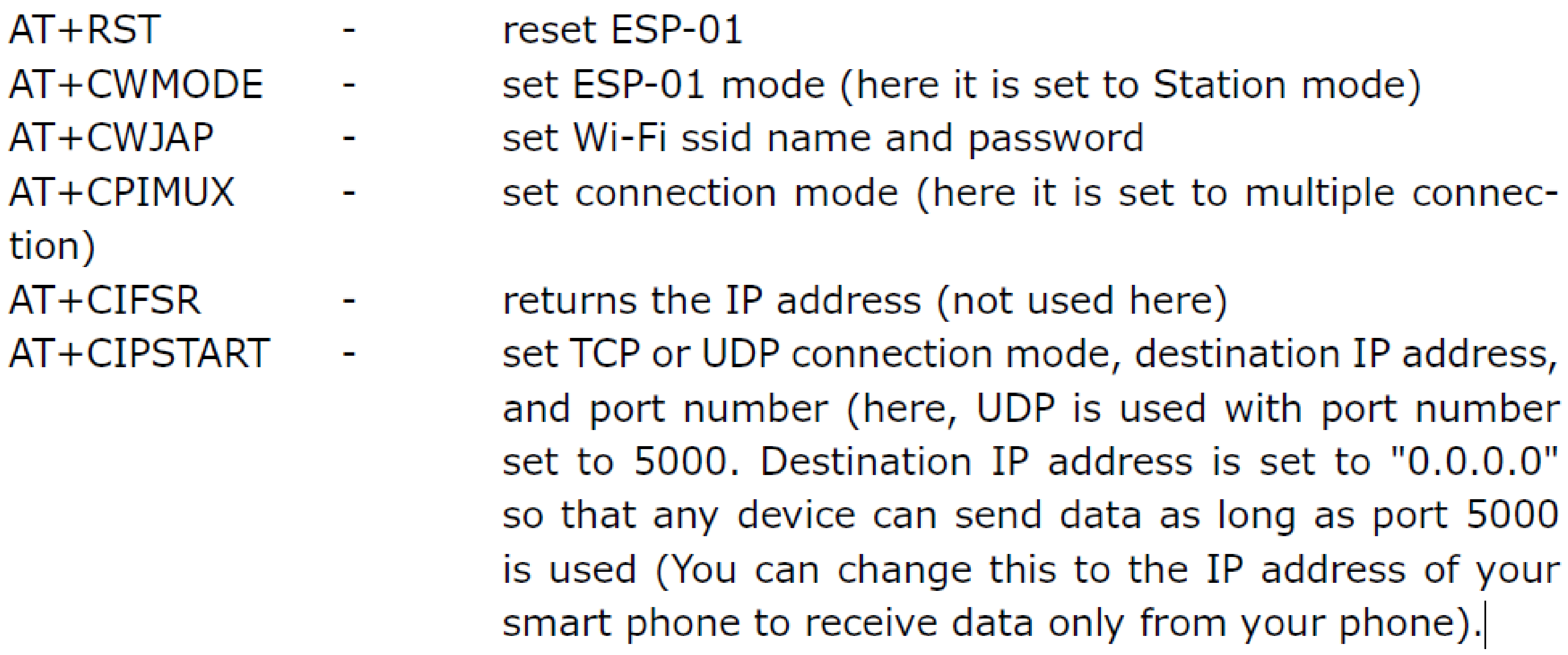
Controlling an LED from a Smartphone Using Wi-Fi



```
    uart.write('''AT+CIPSTART="UDP","0.0.0.0",5000,5000,2\r\n''')
    utime.sleep(3)

ConnectToWiFi()

#
# Main program loop
#
while True:
    buf = uart.readline()          # Read data
    dat = buf.decode('UTF-8')  # Decode
    n = dat.find("LON")          # Includes LON?
    if n > 0:
        LED.value(1)                 # LED ON
    n = dat.find("LOFF")         # Includes OFF?
    if n > 0:
        LED.value(0)                 # LED OFF
```

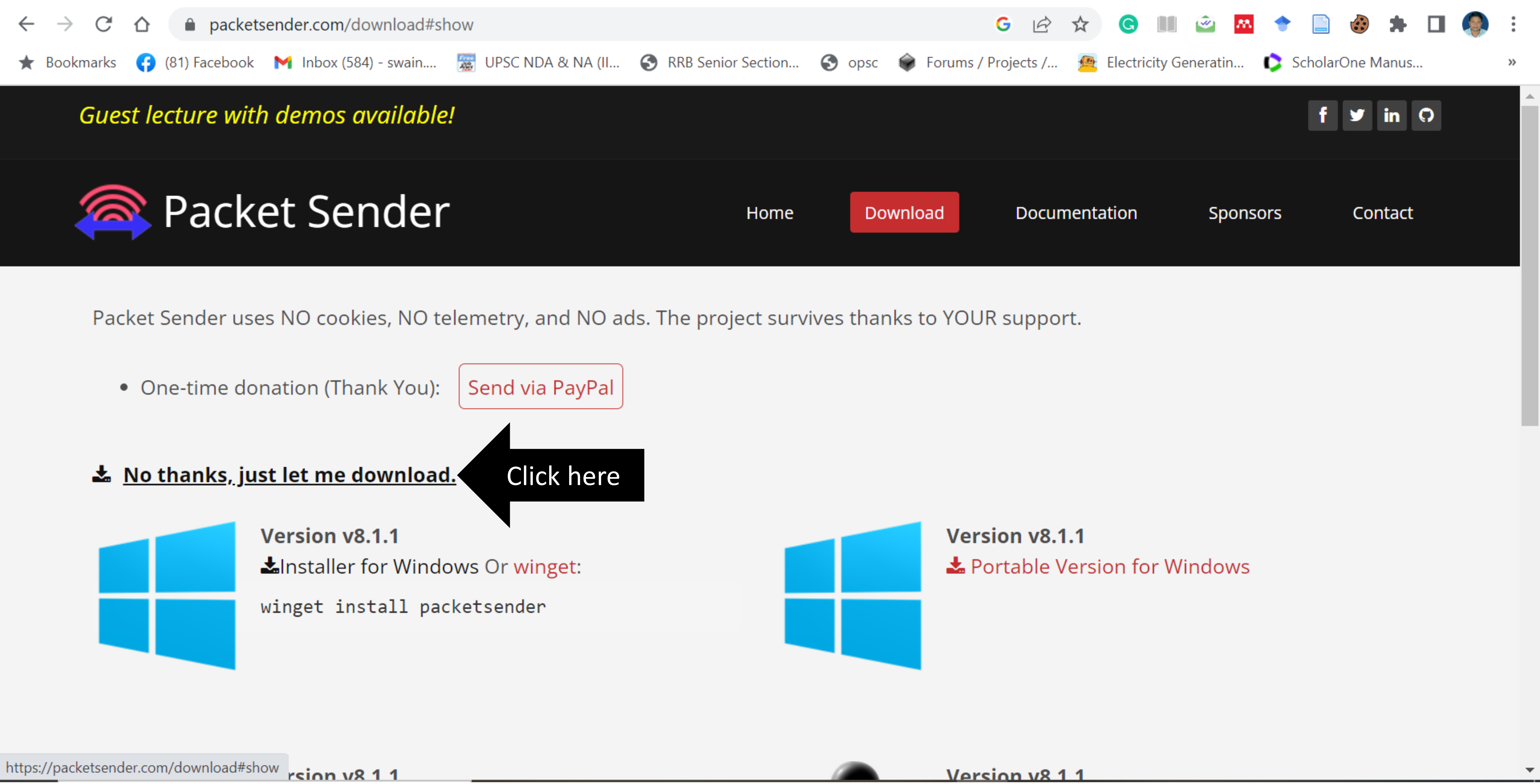
Testing the Program

Function ConnectToWiFi sends the following commands to the ESP-01 to connect to the Wi-Fi network:



AT+RST	-	reset ESP-01
AT+CWMODE	-	set ESP-01 mode (here it is set to Station mode)
AT+CWJAP	-	set Wi-Fi ssid name and password
AT+CPIMUX	-	set connection mode (here it is set to multiple connection)
AT+CIFSR	-	returns the IP address (not used here)
AT+CIPSTART	-	set TCP or UDP connection mode, destination IP address, and port number (here, UDP is used with port number set to 5000. Destination IP address is set to "0.0.0.0" so that any device can send data as long as port 5000 is used (You can change this to the IP address of your smart phone to receive data only from your phone).

UDP Server



Packet Sender uses NO cookies, NO telemetry, and NO ads. The project survives thanks to YOUR support.

- One-time donation (Thank You): Send via PayPal

No thanks, just let me download.

Click here

Version v8.1.1
Installer for Windows Or winget:

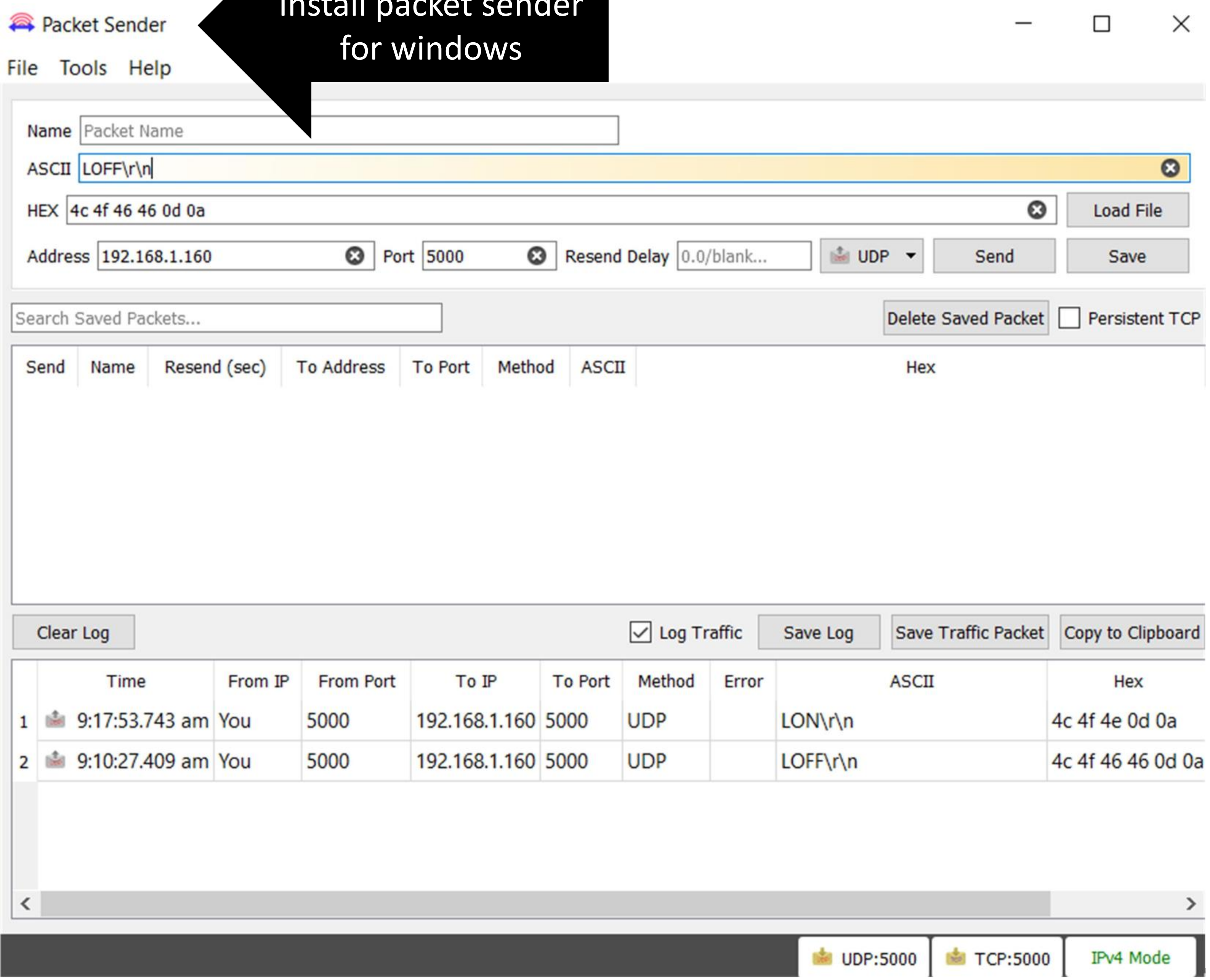
```
winget install packetsender
```

Version v8.1.1
Portable Version for Windows

https://packetsender.com/download#show

Testing the Program

Install packet sender for windows



Packet Sender

File Tools Help

Name: Packet Name

ASCII: LOFF\r\n

HEX: 4c 4f 46 46 0d 0a

Address: 192.168.1.160 Port: 5000 Resend Delay: 0.0/blank... UDP

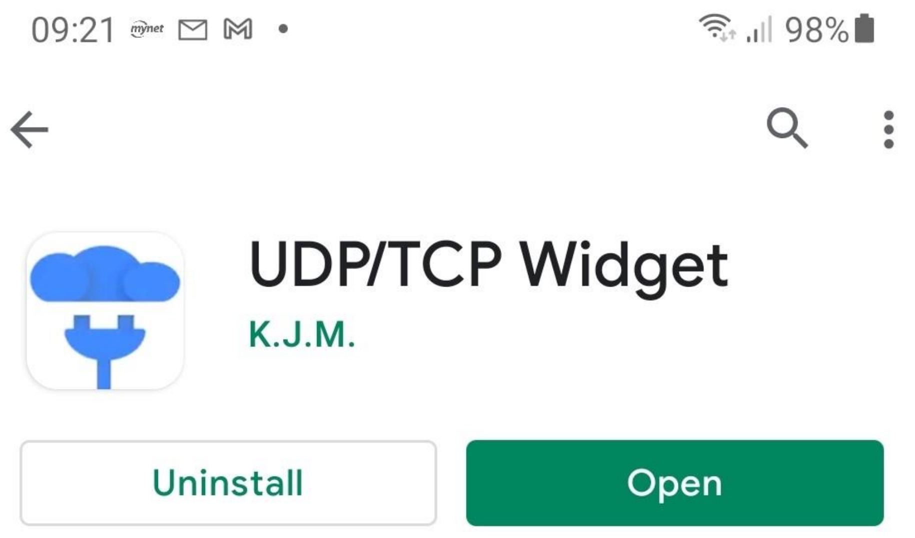
Send	Name	Resend (sec)	To Address	To Port	Method	ASCII	Hex

	Time	From IP	From Port	To IP	To Port	Method	Error	ASCII	Hex
1	9:17:53.743 am	You	5000	192.168.1.160	5000	UDP		LON\r\n	4c 4f 4e 0d 0a
2	9:10:27.409 am	You	5000	192.168.1.160	5000	UDP		LOFF\r\n	4c 4f 46 46 0d 0a

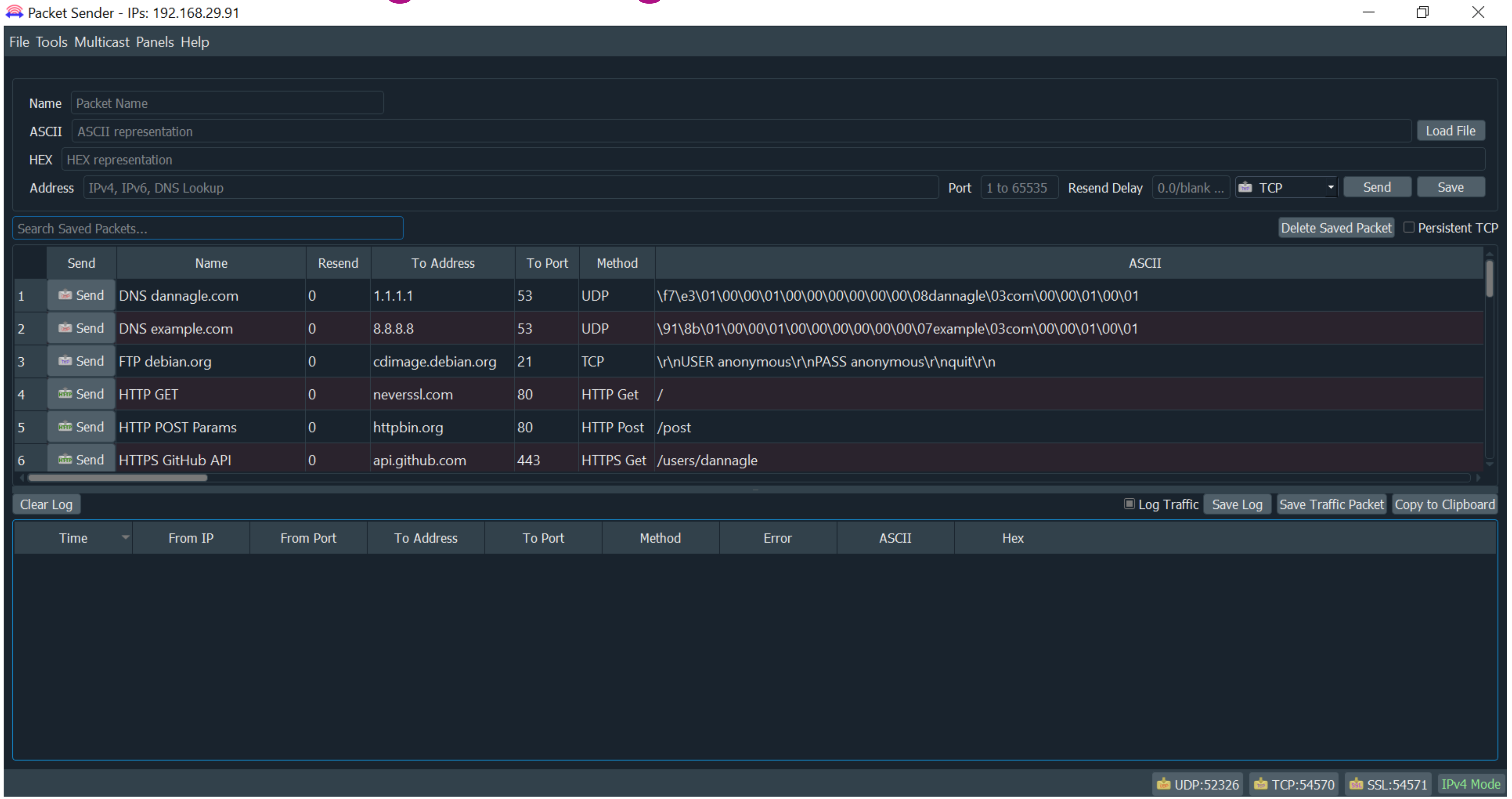
UDP:5000 TCP:5000 IPv4 Mode

Install a UDP Server app on your Android mobile phone

freely available UDP apps in the *Play Store*



Testing the Program: Packet sender


Packet Sender - IPs: 192.168.29.91

File Tools Multicast Panels Help

Name: Packet Name
ASCII: ASCII representation — Load File
HEX: HEX representation
Address: IPv4, IPv6, DNS Lookup — Port: 1 to 65535 — Resend Delay: 0.0/blank ... — TCP — Send — Save

Search Saved Packets... — Delete Saved Packet — Persistent TCP

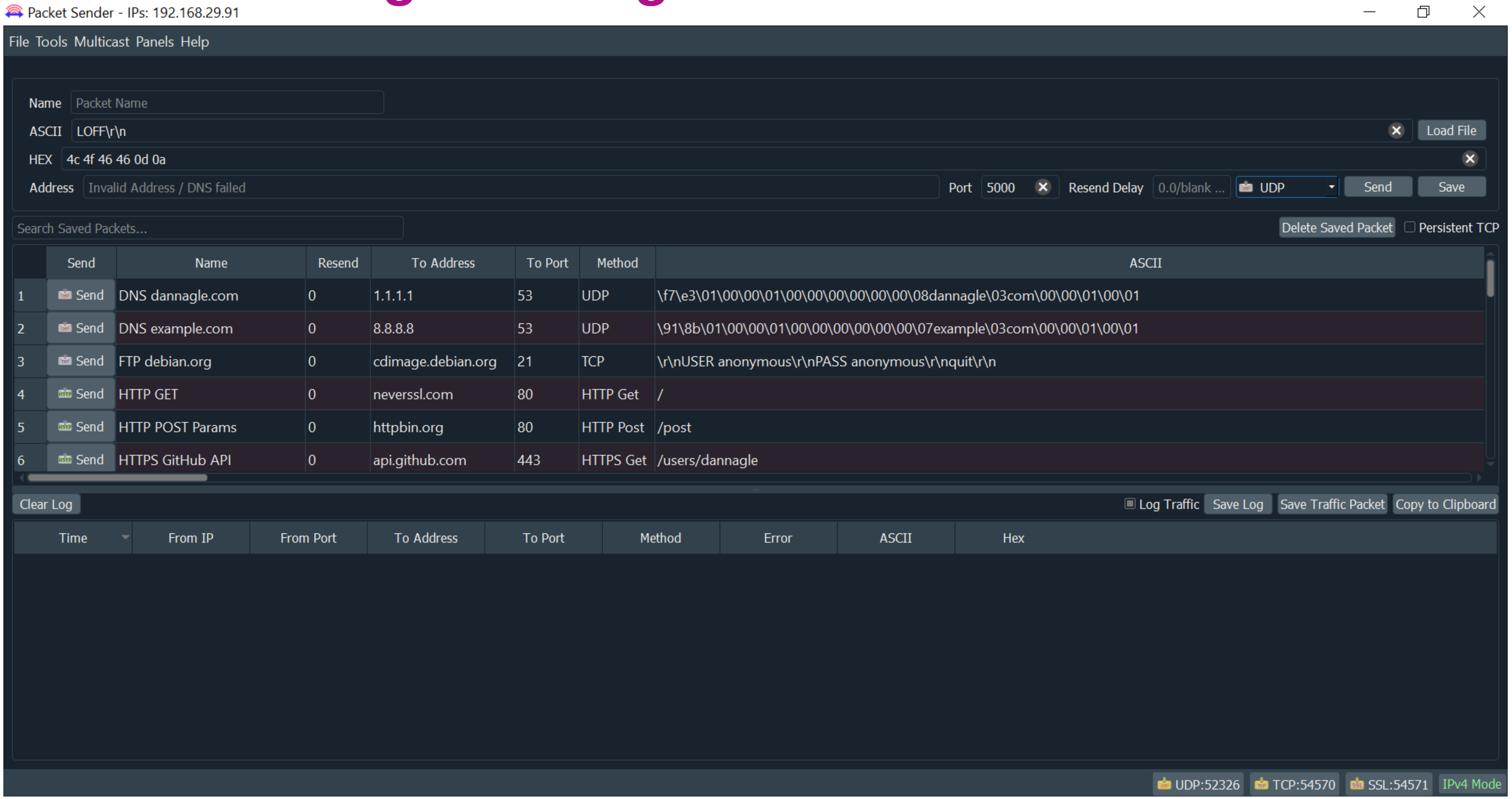
	Send	Name	Resend	To Address	To Port	Method	ASCII
1	Send	DNS dannagle.com	0	1.1.1.1	53	UDP	\f7\e3\01\00\00\01\00\00\00\00\00\00\08dannagle\03com\00\00\01\00\01
2	Send	DNS example.com	0	8.8.8.8	53	UDP	\91\8b\01\00\00\01\00\00\00\00\00\00\07example\03com\00\00\01\00\01
3	Send	FTP debian.org	0	cdimage.debian.org	21	TCP	\r\nUSER anonymous\r\nPASS anonymous\r\nquit\r\n
4	Send	HTTP GET	0	neverssl.com	80	HTTP Get	/
5	Send	HTTP POST Params	0	httpbin.org	80	HTTP Post	/post
6	Send	HTTPS GitHub API	0	api.github.com	443	HTTPS Get	/users/dannagle

Clear Log — Log Traffic — Save Log — Save Traffic Packet — Copy to Clipboard

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII	Hex

UDP:52326 — TCP:54570 — SSL:54571 — IPv4 Mode

Testing the Program: Packet sender



Packet Sender - IPs: 192.168.29.91

File Tools Multicast Panels Help

Name: Packet Name

ASCII: LOFF\r\n

HEX: 4c 4f 46 46 0d 0a

Address: Invalid Address / DNS failed

Port: 5000

Resend Delay: 0.0/blank ...

UDP

Send Save Load File

Search Saved Packets...

Delete Saved Packet

Persistent TCP

	Send	Name	Resend	To Address	To Port	Method	ASCII
1	Send	DNS dannagle.com	0	1.1.1.1	53	UDP	\f7\e3\01\00\00\01\00\00\00\00\00\00\08dannagle\03com\00\00\01\00\01
2	Send	DNS example.com	0	8.8.8.8	53	UDP	\91\8b\01\00\00\01\00\00\00\00\00\00\07example\03com\00\00\01\00\01
3	Send	FTP debian.org	0	cdimage.debian.org	21	TCP	\r\nUSER anonymous\r\nPASS anonymous\r\nquit\r\n
4	Send	HTTP GET	0	neverssl.com	80	HTTP Get	/
5	Send	HTTP POST Params	0	httpbin.org	80	HTTP Post	/post
6	Send	HTTPS GitHub API	0	api.github.com	443	HTTPS Get	/users/dannagle

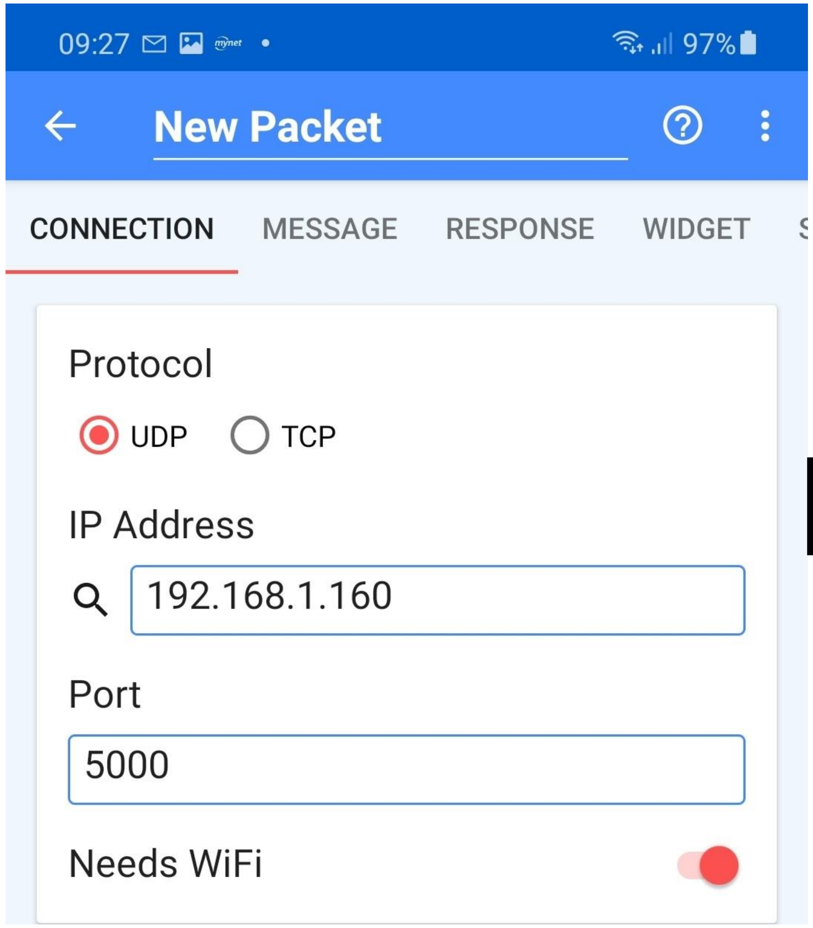
Clear Log

Log Traffic Save Log Save Traffic Packet Copy to Clipboard

Time	From IP	From Port	To Address	To Port	Method	Error	ASCII	Hex

UDP:52326 TCP:54570 SSL:54571 IPv4 Mode

Testing the Program in Smartphone


09:27 97%

New Packet

CONNECTION MESSAGE RESPONSE WIDGET

Protocol

[✓ UDP] TCP

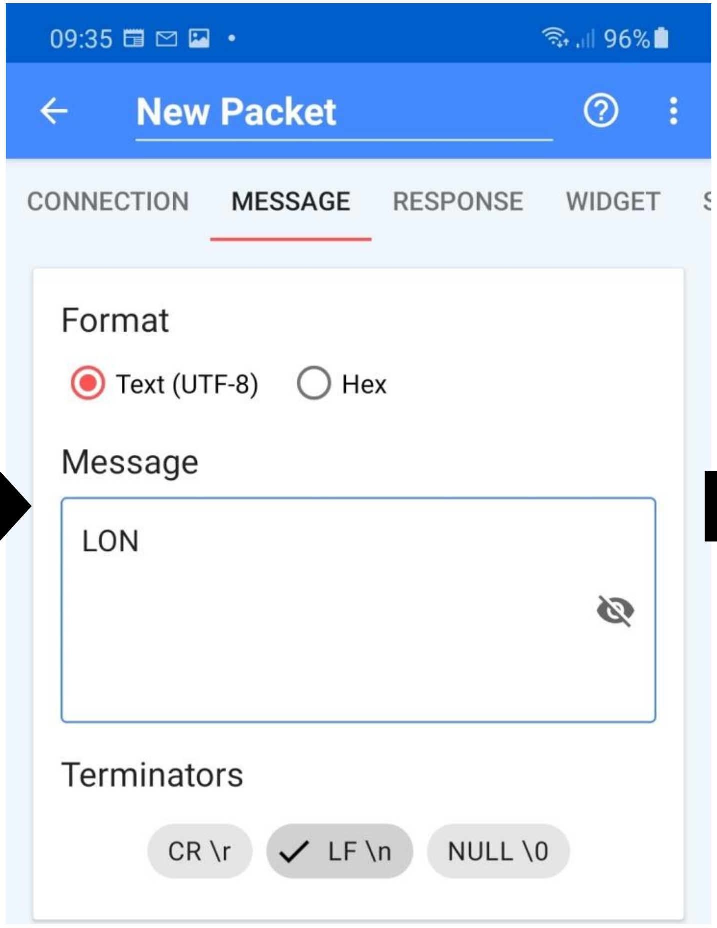
IP Address

192.168.1.160

Port

5000

Needs WiFi


09:35 96%

New Packet

CONNECTION MESSAGE RESPONSE WIDGET

Format

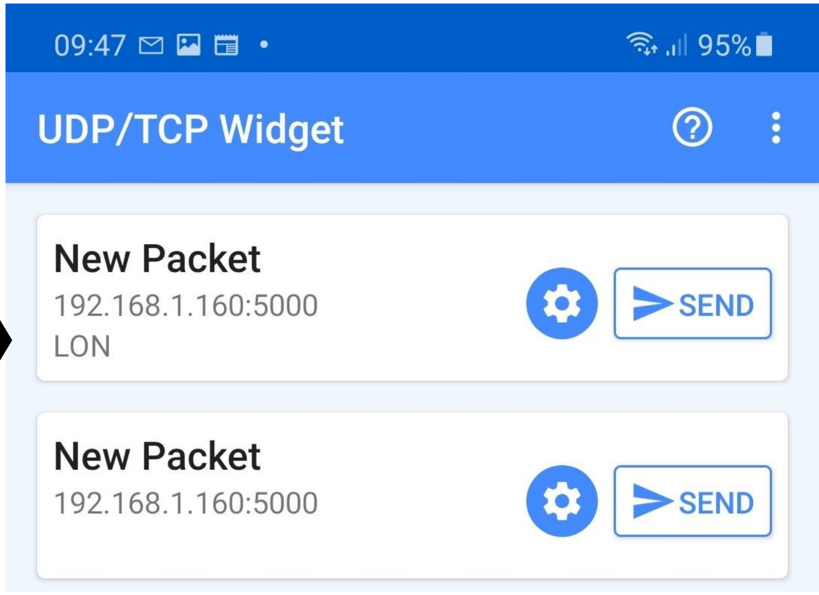
[✓ Text (UTF-8)] Hex

Message

LON

Terminators

CR \r [✓ LF \n] NULL \0

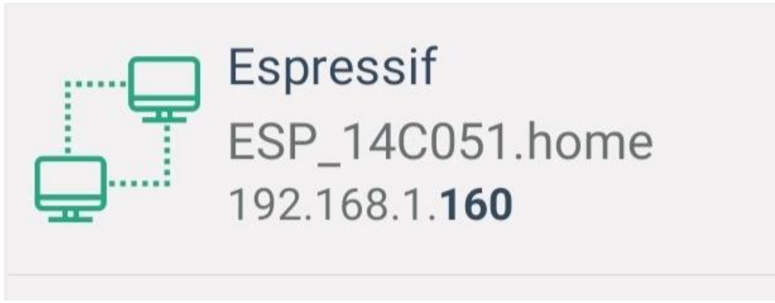

09:47 95%

UDP/TCP Widget

New Packet
192.168.1.160:5000
LON
SEND

New Packet
192.168.1.160:5000
SEND

IP address of the ESP-01 can be obtained by scanning all the devices on the local Wi-Fi router. For example, the Android app called Who Uses My WiFi – Network Scanner by Phuongpn can be used to see the IP addresses of all the devices connected to your router.


Espressif
ESP_14C051.home
192.168.1.160

Control an LED from the Android application using Bluetooth wireless communication

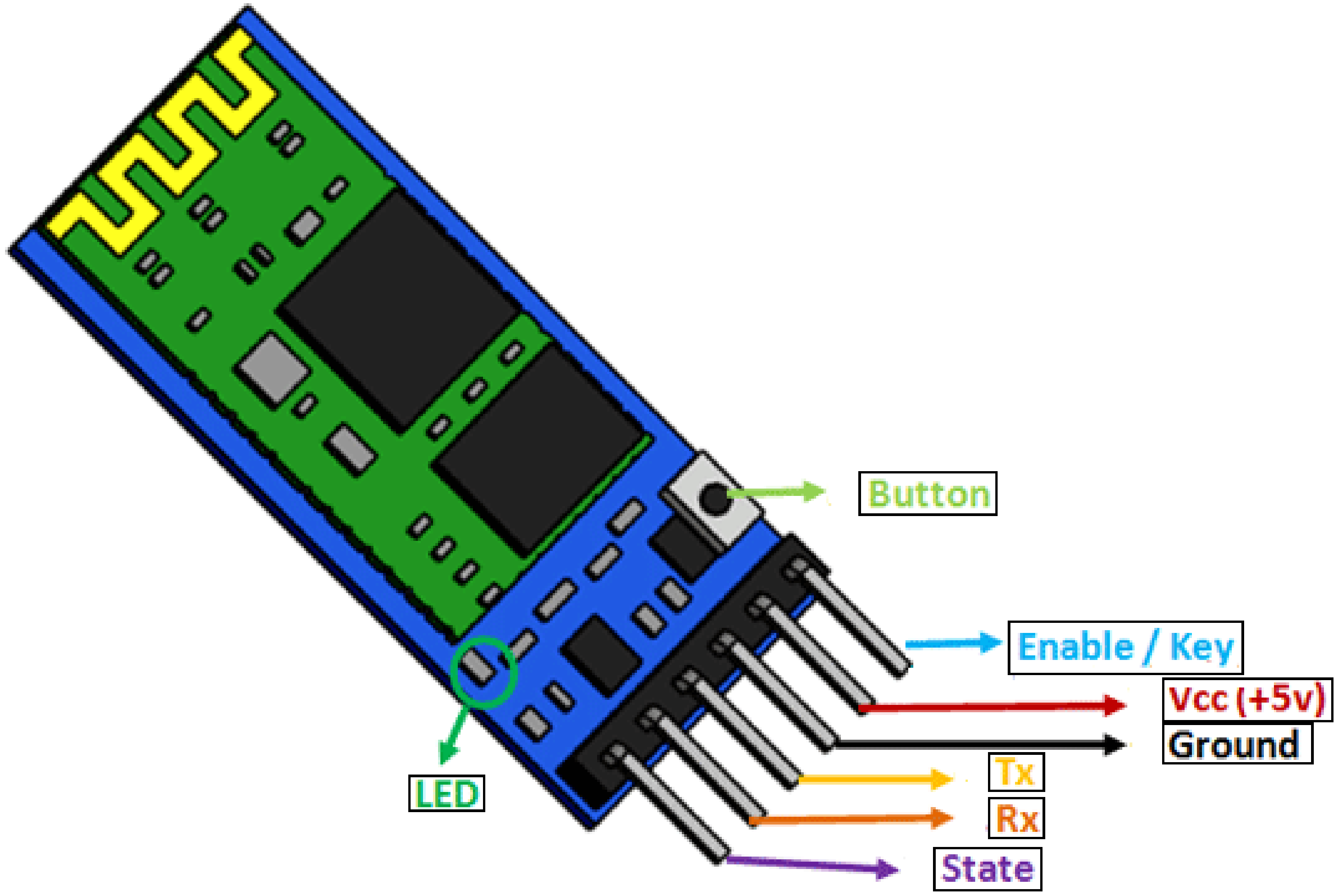
Bluetooth Module HC-05

HC-05 is one of the commonly used Bluetooth device that uses a UART communication protocol. The module normally operates at UART serial communication with TX and RX pins at 9600 baud rates. However, it can be programmed with AT commands and it supports 9600,19200,38400,57600,115200,230400,460800 baud rates.

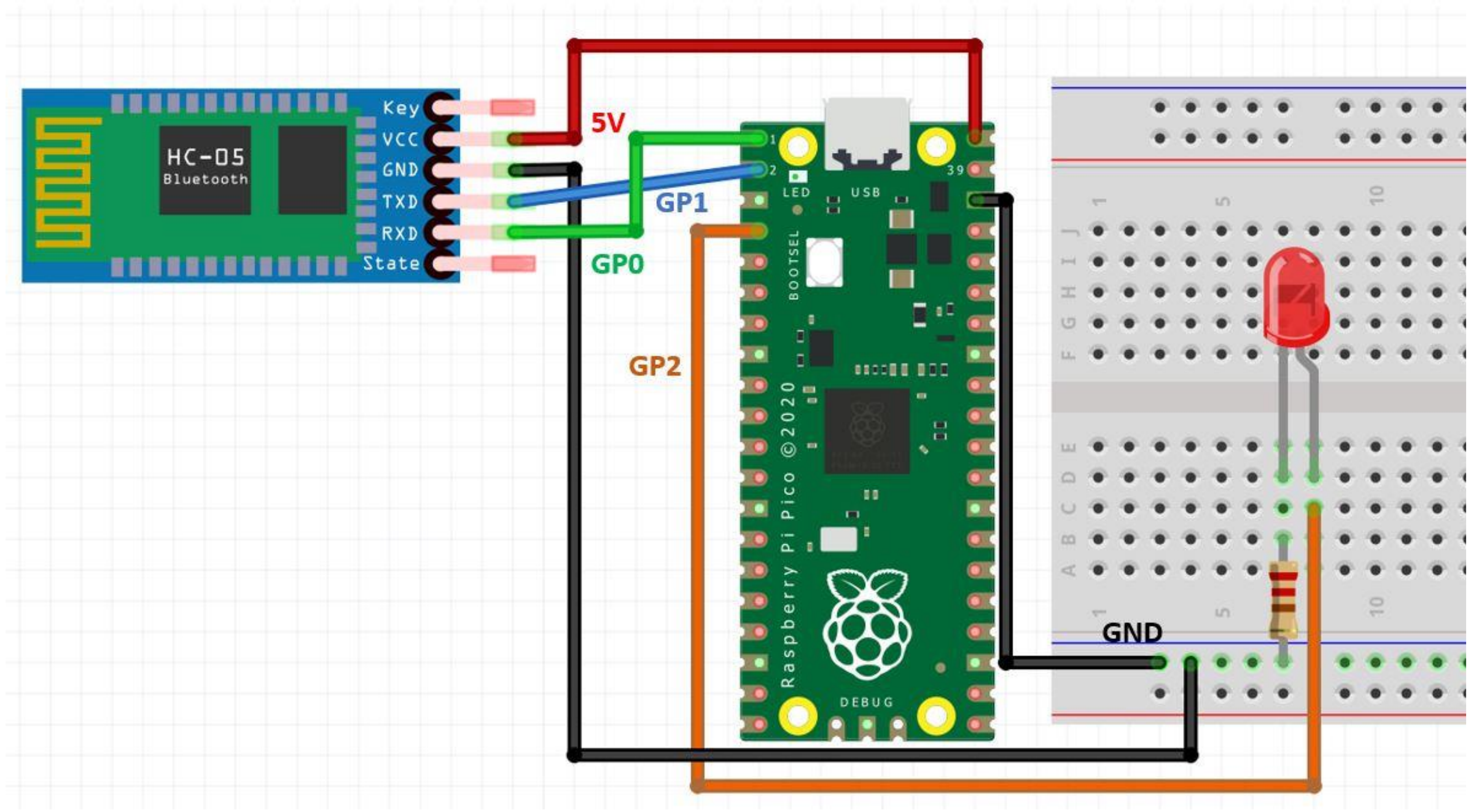
Bluetooth Module HC 05 Pinout

Bluetooth Module HC-05

HC-05 is one of the commonly used Bluetooth device that uses a UART communication protocol. The module normally operates at UART serial communication with TX and RX pins at 9600 baud rates. However, it can be programmed with AT commands and it supports 9600,19200,38400,57600,115200,230400,460800 baud rates.

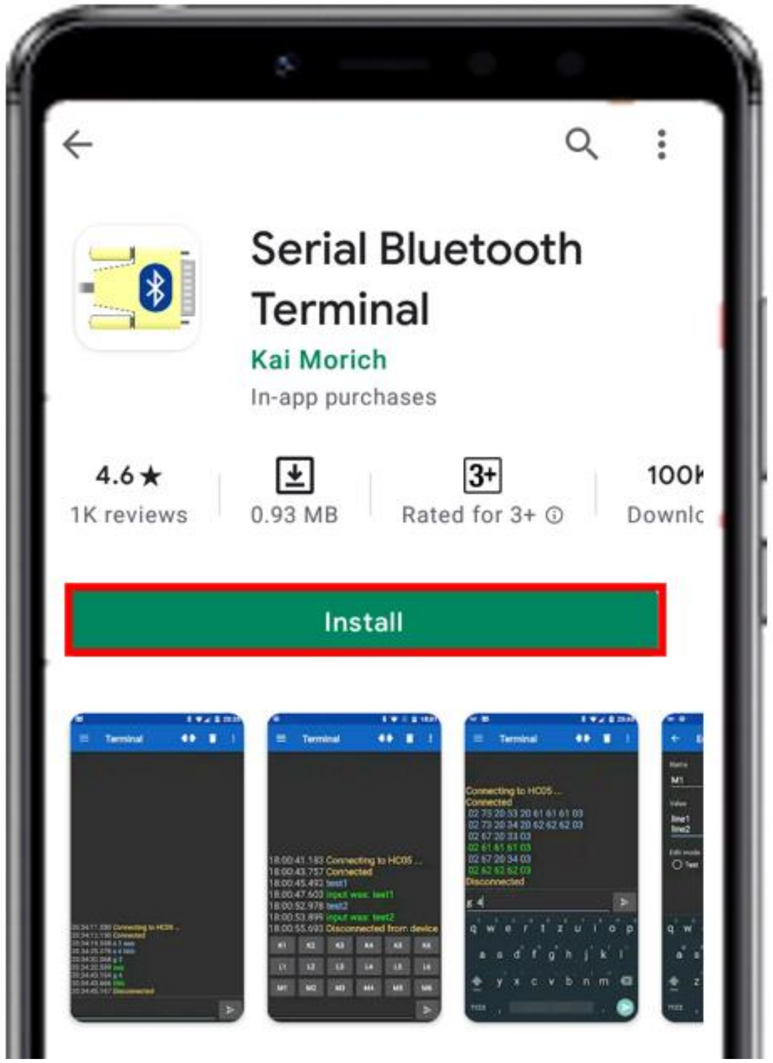


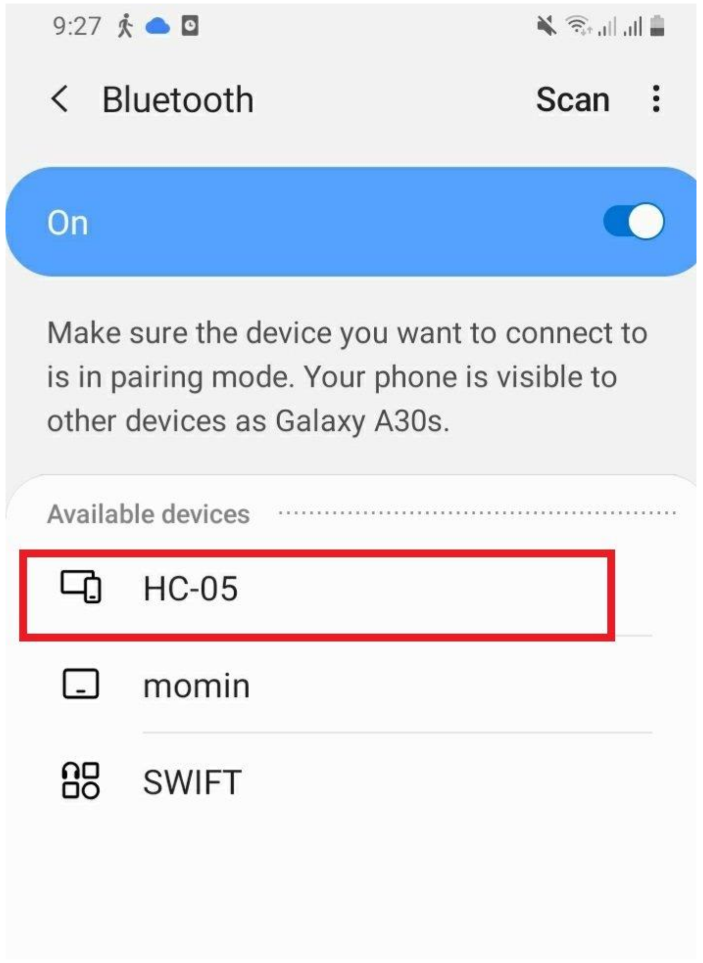
Schematic Raspberry Pi Pico and HC-05



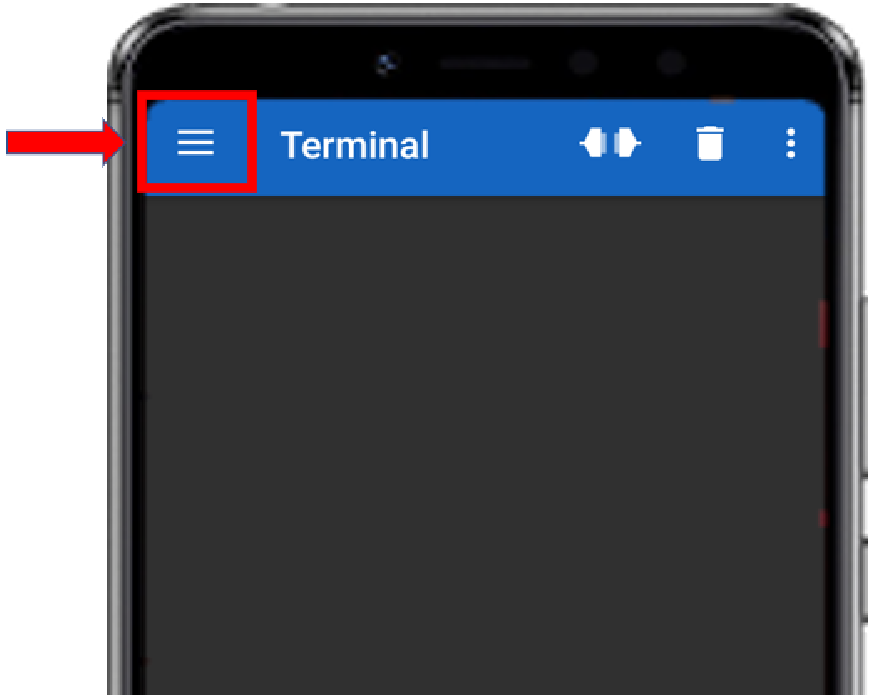
Bluetooth Terminal Application

Go to the Play Store and download the application by the name: Serial Bluetooth terminal.

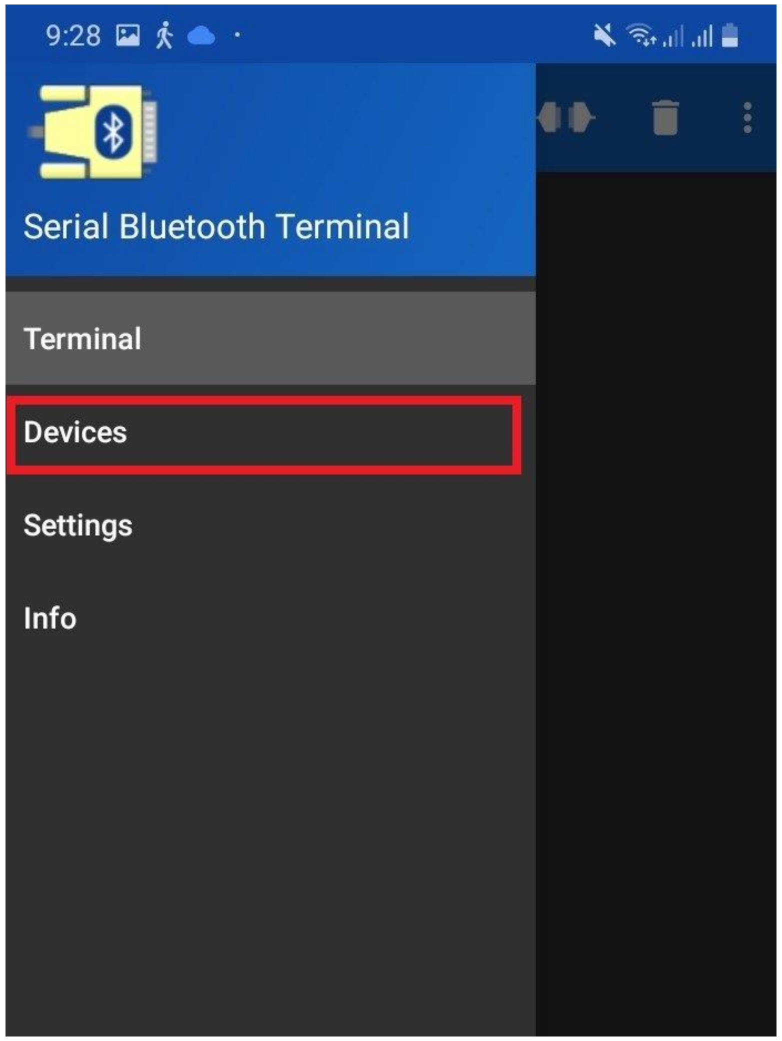




Make sure the device you want to connect to is in pairing mode. Your phone is visible to other devices as Galaxy A30s.



Bluetooth Terminal Application



9:28

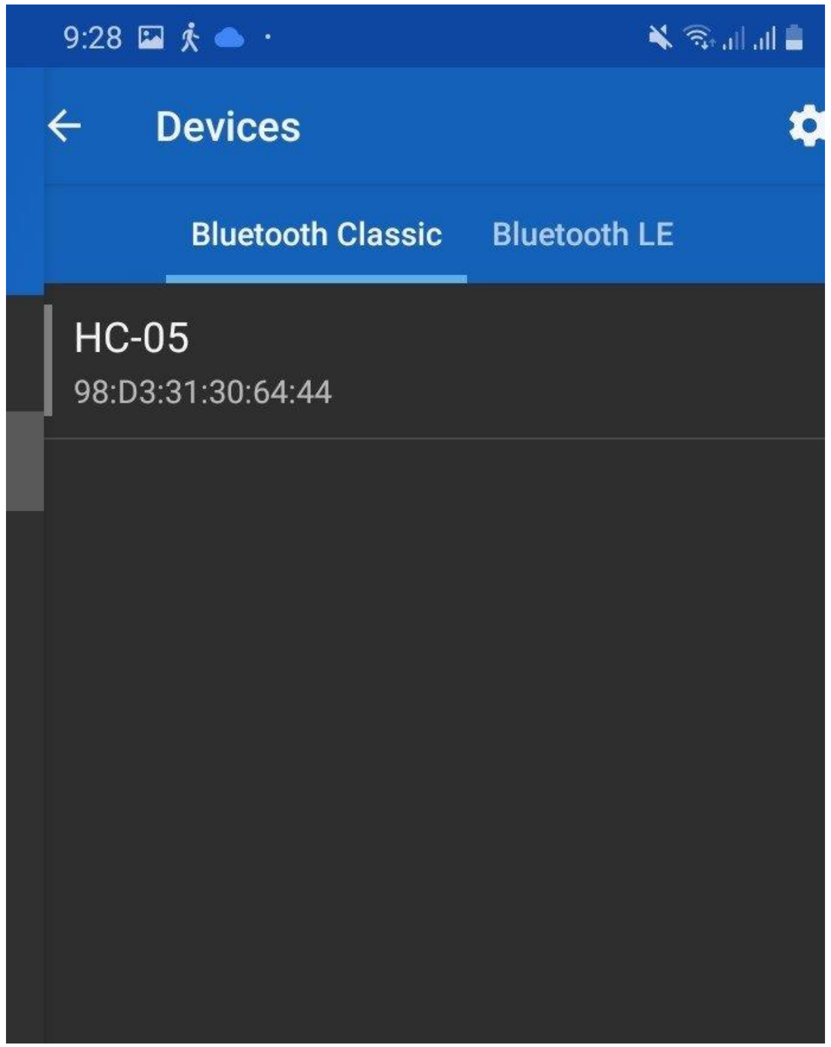
Serial Bluetooth Terminal

Terminal

Devices

Settings

Info



9:28

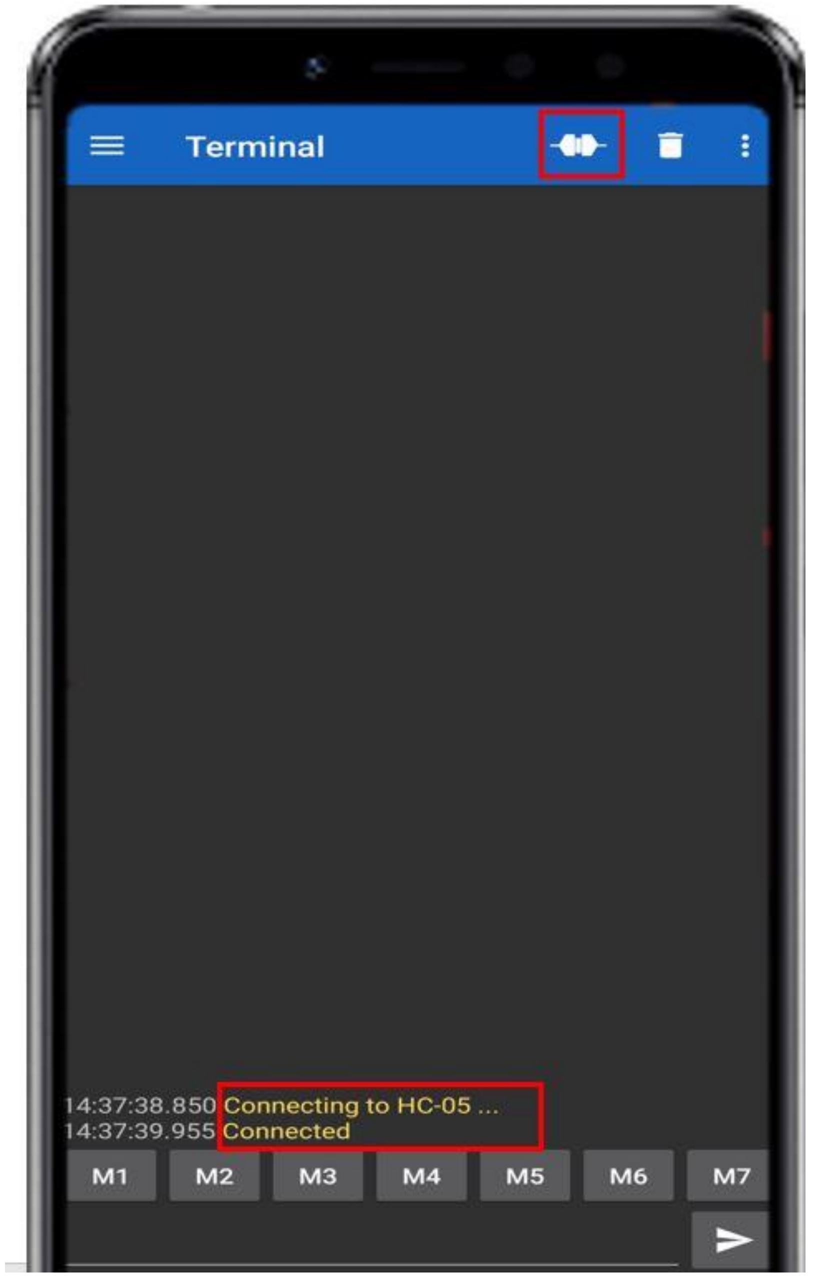
Devices

Bluetooth Classic

Bluetooth LE

HC-05

98:D3:31:30:64:44

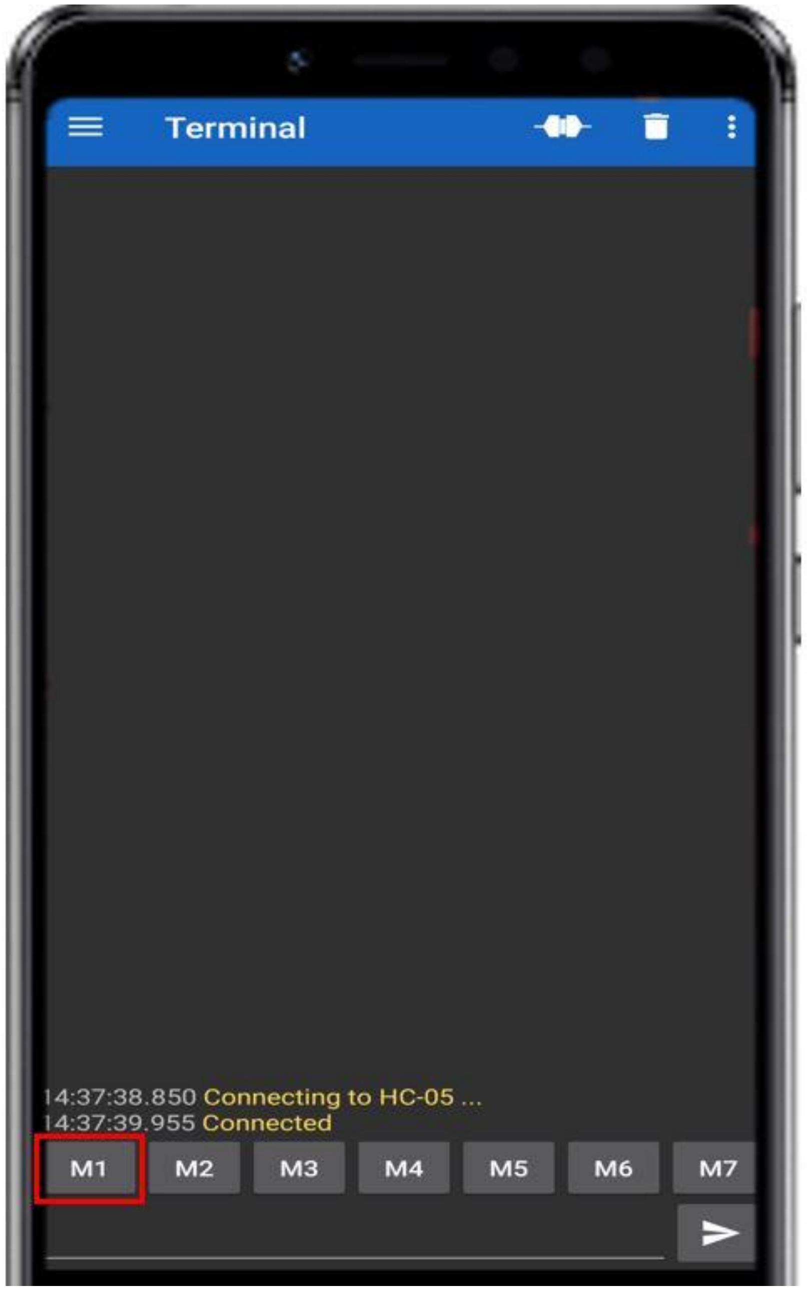


Terminal

```
14:37:38.850 Connecting to HC-05 ...
14:37:39.955 Connected
```

M1 M2 M3 M4 M5 M6 M7

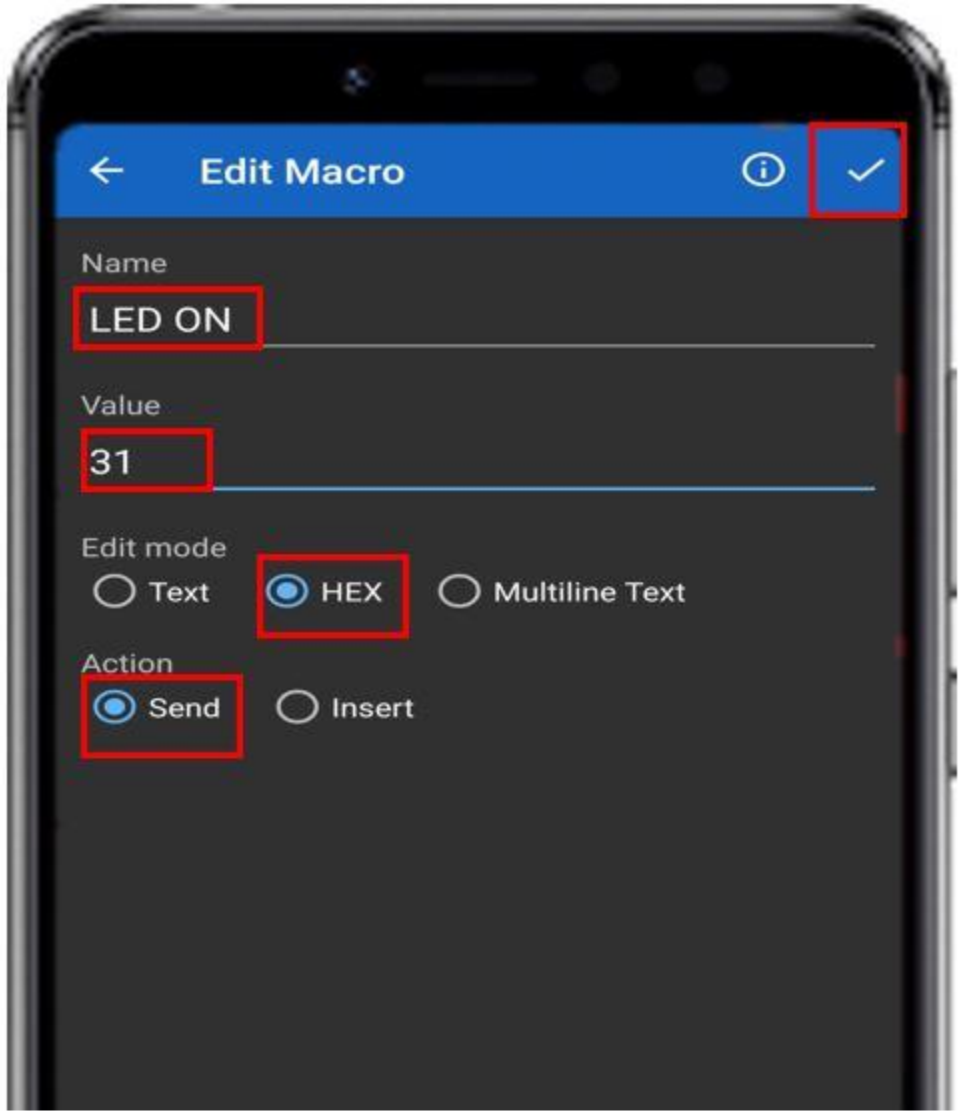
Bluetooth Terminal Application



Terminal

```
14:37:38.850 Connecting to HC-05 ...
14:37:39.955 Connected
```

M1 as ON



Edit Macro

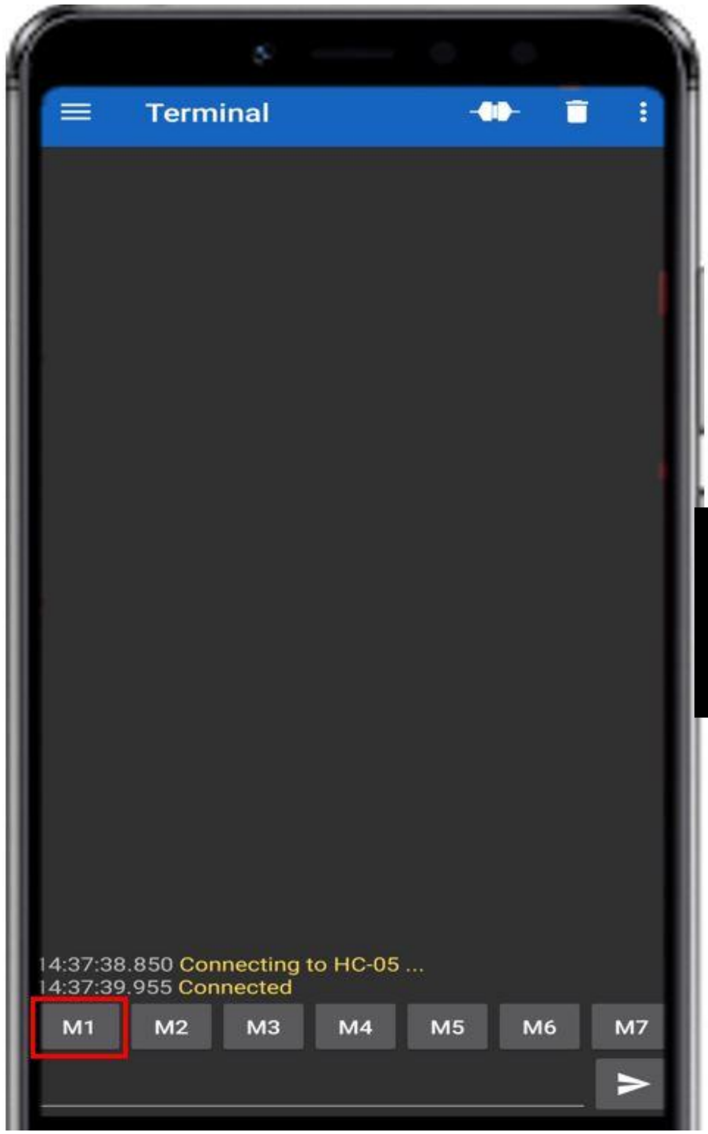
Name: LED ON

Value: 31

Edit mode: Text [✓ HEX] Multiline Text

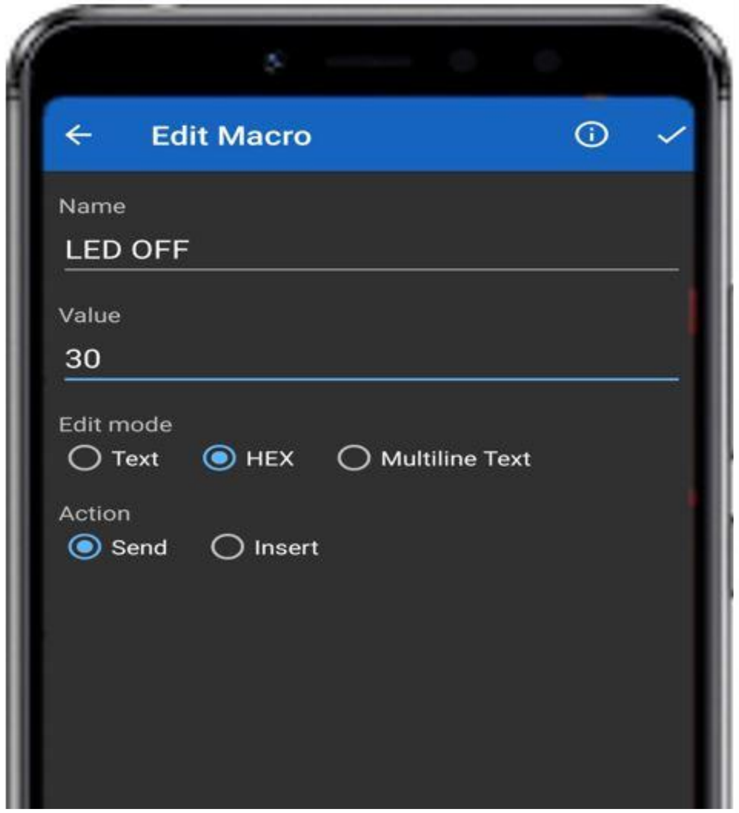
Action: [✓ Send] Insert

Bluetooth Terminal Application



```
14:37:38.850 Connecting to HC-05 ...
14:37:39.955 Connected
```

M2 as OFF

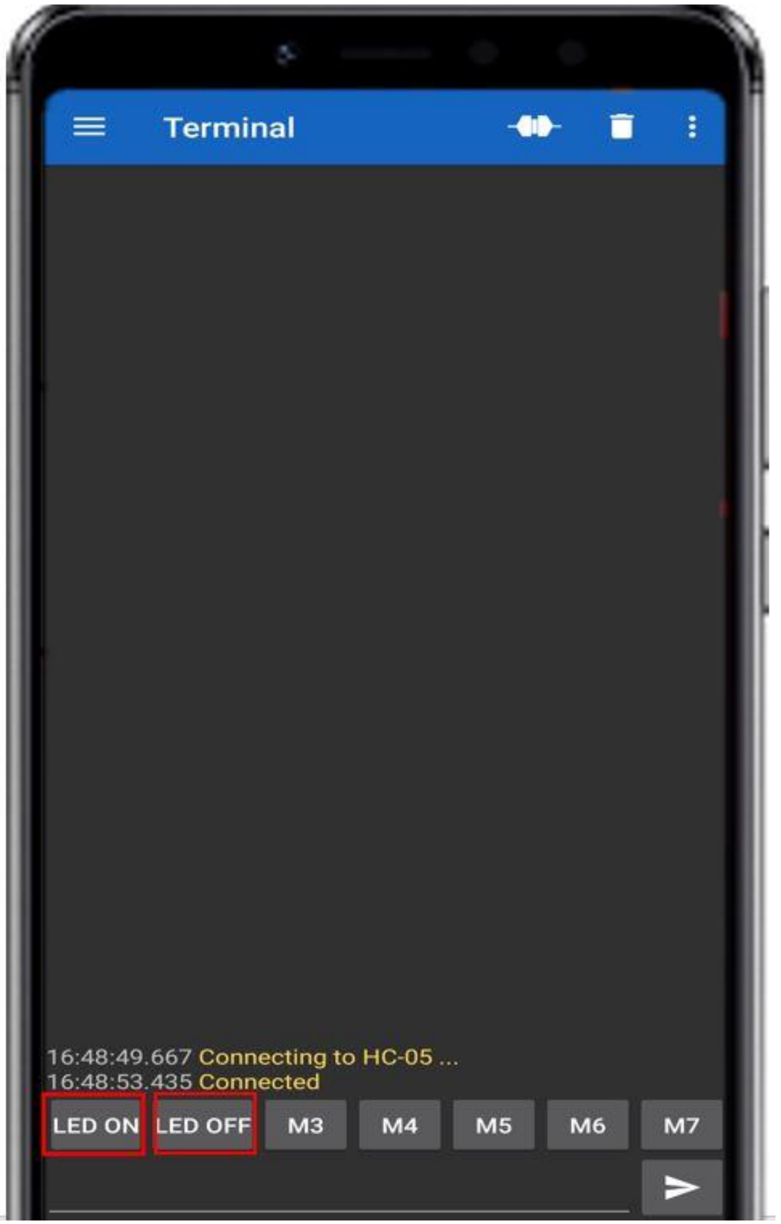


Name: LED OFF

Value: 30

Edit mode: Text [✓ HEX] Multiline Text

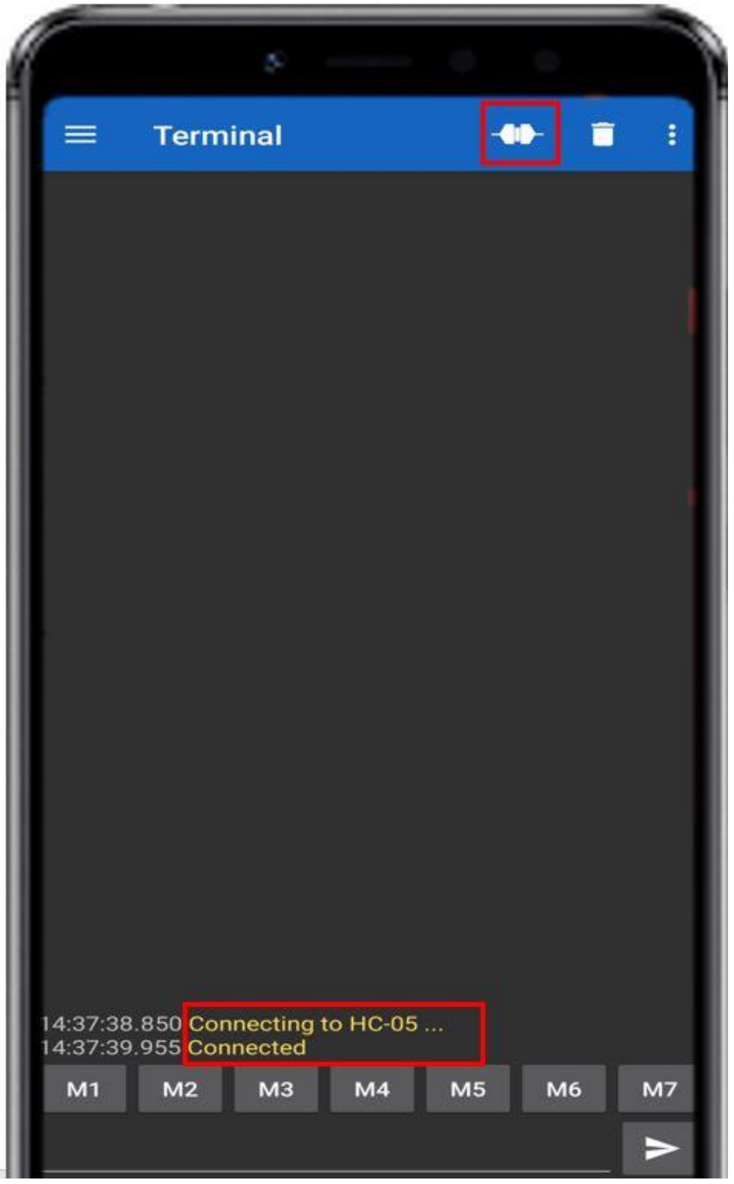
Action: [✓ Send] Insert



```
16:48:49.667 Connecting to HC-05 ...
16:48:53.435 Connected
```

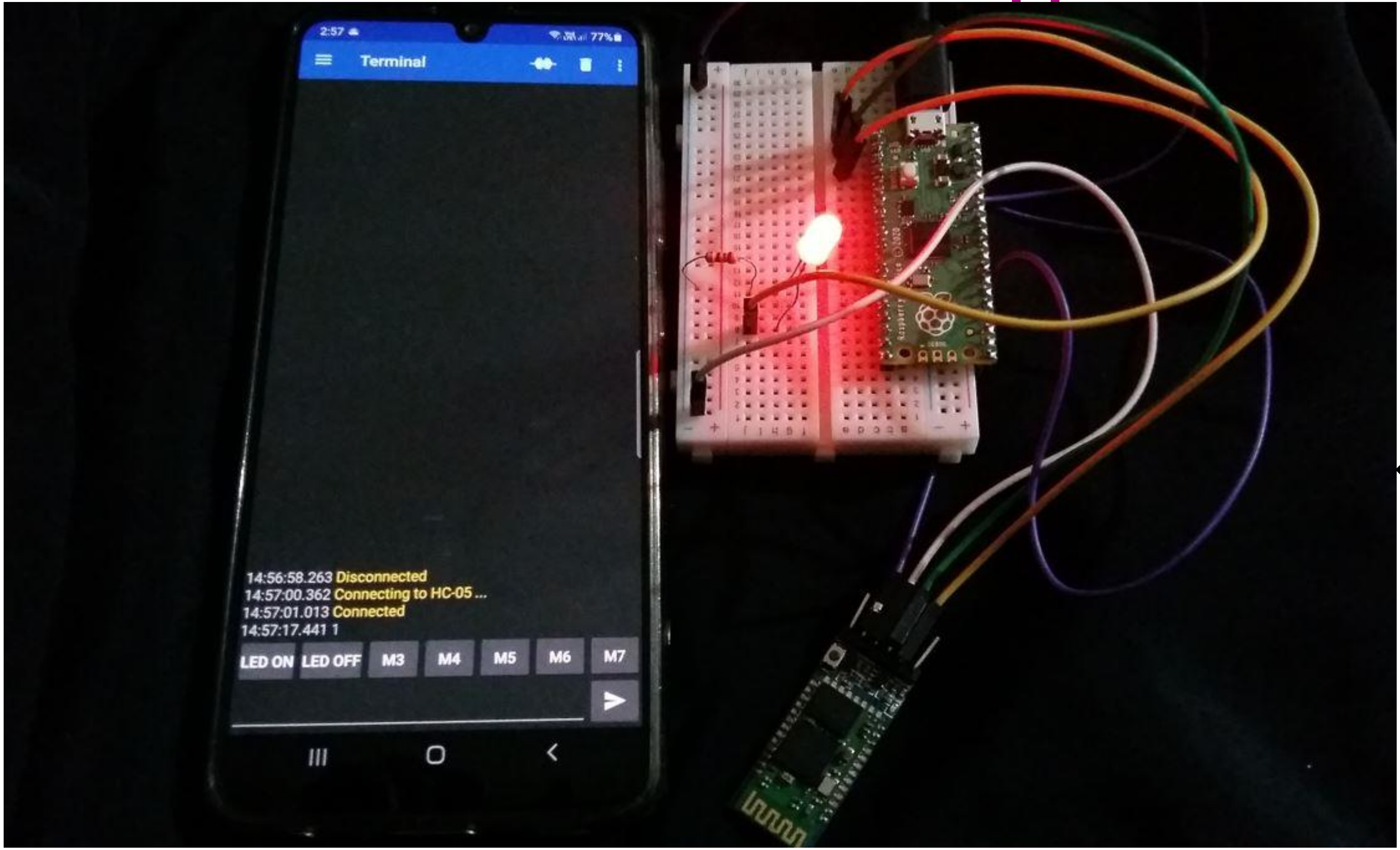
Bluetooth Terminal Application

```python
from machine import Pin,UART
uart = UART(0,9600)

Led_pin = 2
led = Pin(Led_pin, Pin.OUT)

while True:
    if uart.any():
        data = uart.readline()
        print(data)
        if data== b'1':
            led.high()
            print("LED is now ON!")
        elif data== b'0':
            led.low()
            print("LED is now OFF!")
```



```
14:37:38.850 Connecting to HC-05 ...
14:37:39.955 Connected
```

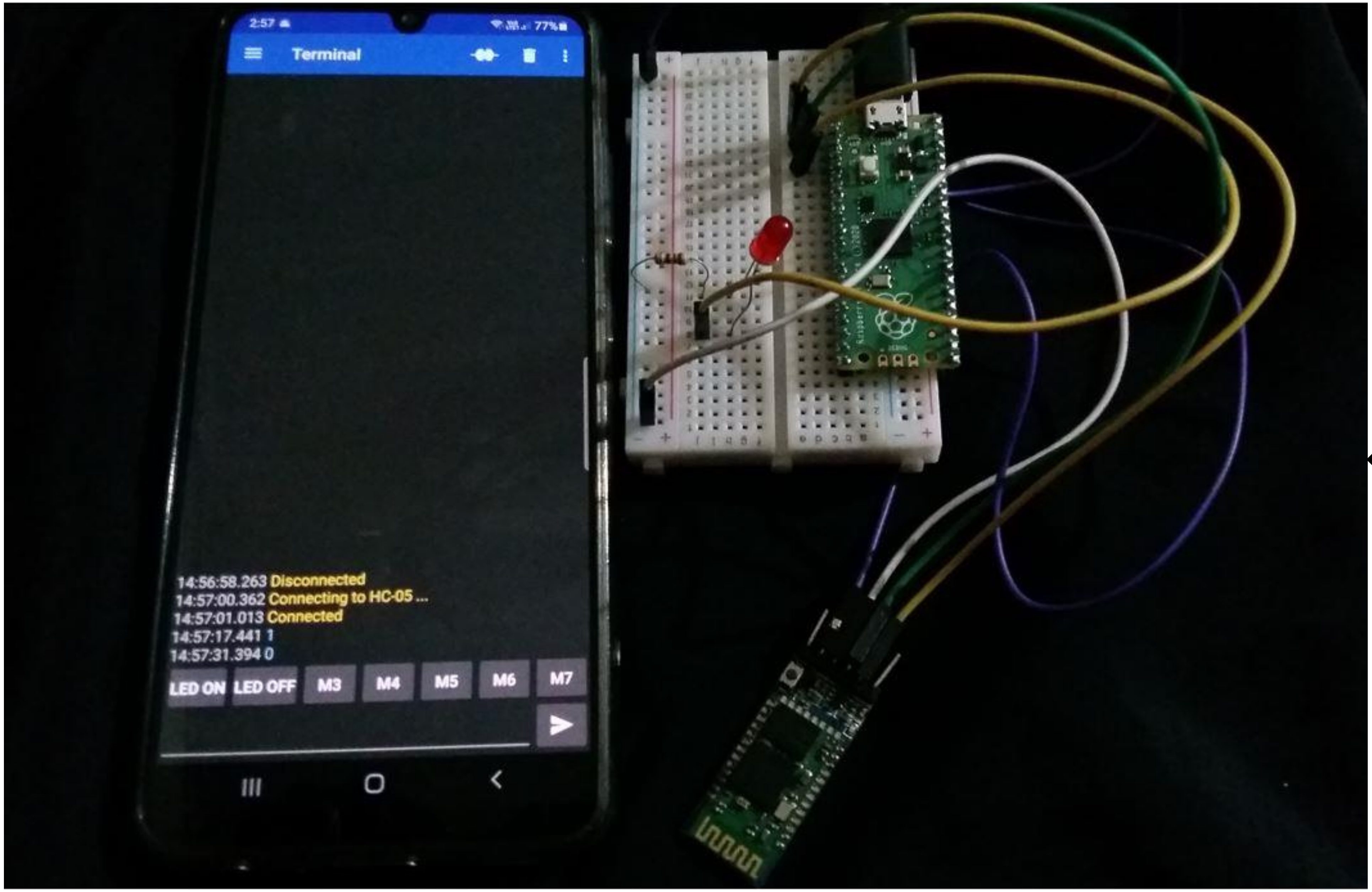
Bluetooth Terminal Application



```
14:56:58.263 Disconnected
14:57:00.362 Connecting to HC-05 ...
14:57:01.013 Connected
14:57:17.441 1
```

Press LEDON

Bluetooth Terminal Application



```
14:56:58.263 Disconnected
14:57:00.362 Connecting to HC-05 ...
14:57:01.013 Connected
14:57:17.441 1
14:57:31.394 0
```

Press LEDOFF