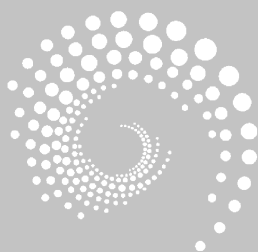


ESP32的串口通信

CHAPTER 5



目

前,嵌入式系统已经不是简单的独立工作的设备,往往需要与其他设备或用户进行互动,这就涉及数据的传输。在嵌入式系统中,典型的通信方式就是串口通信。本章主要讲解异步串行通信的原理,ESP32 中串口初始化、串口数据的输出、串口数据的查询输入方法,并在 Wokwi 仿真平台和 DIY 开发板上实现呼吸灯占空比数据输出和输入控制案例的调试与运行。

学习目标:

- (1) 掌握 ESP32 中串口基本特性。
- (2) 掌握 ESP32 串口的初始化和输入、输出使用方法。
- (3) 掌握 Wokwi 仿真平台和 DIY 开发板上串口调试流程和方法。

5.1 串口通信原理



5.1.1 串行通信介绍

1. 什么是串行通信

所谓通信,对于设备而言就是它们之间的数据交换,这种数据交换是通过传输电信号来实现的。在以二进制为基础的计算机中,高电位(数据“1”)是一种状态,低电位(数据“0”)则是另一种状态,通过若干高低电位的组合状态进行传递就实现了数据交换。

终端与计算机之间或者计算机与计算机之间交换信息时,通常采用的是并行通信的方式,但是微型计算机相互通信,特别是远距离通信中,并行通信已显得无能为力。因此随着微型计算机技术的发展,除了采用并行通信方式外,还经常采用串行通信方式。

并行通信是指数据的各位同时进行传输,其优点是传输数据速率快,缺点是有多少位数据就需要多少根传输线,这在数据位数较多,传输距离较远时就不宜采用。

串行通信是指数据一位一位地按顺序传输,其突出优点是无论数据有多少位只需一根传输线,特别适应于远距离传输,缺点是传输速率较慢。

2. 单工、半双工与全双工

在串行通信中,数据传输有3种工作方式:单工方式、半双工方式和全双工方式,如图5.1所示。单工方式只允许数据按一个固定的方向传输;半双工方式中,数据在一个时刻只能进行一个方向的传输;全双工方式下,同一时刻数据可以在两个方向上相互传输。

3. 同步与异步传输

串行通信分为同步传输和异步传输两种方式,如图5.2所示。

同步传输方式要求通信双方以相同的速率进行,而且要准确地协调。它通过共享一个单一的时钟或定时脉冲源以保证发送方和接收方准确同步。其特点是允许连续发送一组字符序列(而非单个字符),每个字符数据位数相同,且不增加任何附加位,没有起始位和停止位,效率高。

异步传输方式不要求通信双方同步,其优点是收发双方不需要严格的位同步。也就是说,在这种通信方式下,每个字符作为独立的通信单元,可以随机地出现在数据流中,而每个字符出现在数据流中的相对时间是随机的。在异步通信中,“异步”是指字符与字符之间异步,而在字符内部,仍然是同步传输。发送方和接收方可以有各自的时钟源。为了能够实现通信,双方必须都遵循异步通信协议。在异步通信中,通信双方必须规定两件事:一是采用的字符格式,即规定字符各部分所占的位数,是否采用奇偶校验,以及校验的方式;二是采用的波特率,以及时钟率与波特率之间的比例关系。由此可见,异步通信方式的传输效率比同步通信方式低,但它对通信双方的同步要求大幅降低,因而成本也比同步通信方式低。

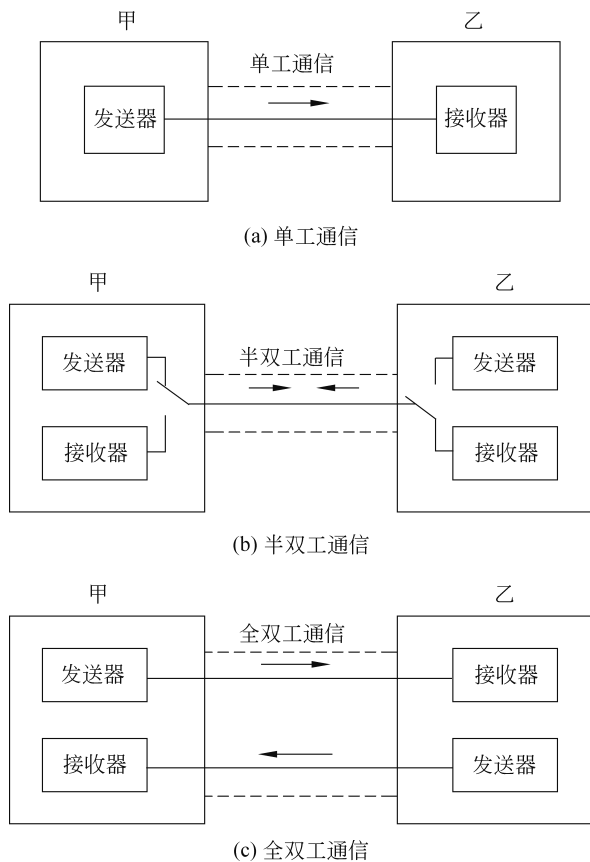


图 5.1 串行工作方式

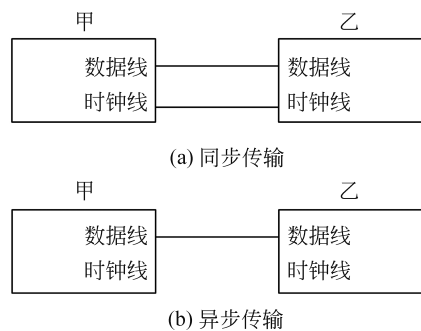


图 5.2 同步传输与异步传输示意图

4. ESP32 的全双工异步串行通信

目前在嵌入式设备串口通信中,大多数使用全双工异步传输通信方式。ESP32 支持全

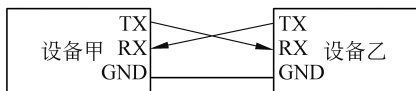


图 5.3 全双工异步传输通信

双工异步传输通信方式,其硬件连接方式如图 5.3 所示。两个设备之间只要使用 3 条线路即可,其中 TX 为发送数据,RX 为接收数据,GND 为地信号。

TX 和 RX 之间一般采用交叉连接方式。从图中可

以看出,在全双工硬件连接方式下,也可以通过软件控制实现单工或半双工工作方式。

在异步串行通信中使用的通信协议格式如图 5.4 所示。数据是以字符(例如 1 字节即 8 位)为单位一个个地发送和接收的。发送的 1 个字符为 1 帧数据,其由 4 部分组成:

(1) 起始位: 1 位,用逻辑“0”低电平表示,用于通知接收设备新字符到达,并复位接收设备以准备接收。

(2) 有效数据位(字符位): 可选 5~8 位,表示发送的 1 个字符数据。

(3) 奇偶校验位: 0 位或 1 位,可选奇校验、偶校验或无校验。

(4) 停止位: 1 位、1.5 位或 2 位,用逻辑“1”高电平表示。

上述格式中规定异步串行通信 1 帧数据的格式,但是没有规定具体每一位二进制数据占用的时间,而这个参数用串口参数波特率来确定,串口通信的双方只有采用相同的波特率

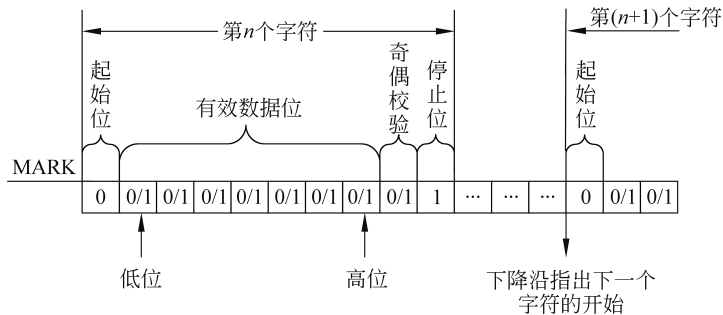


图 5.4 在异步串行通信中使用的通信协议格式

才能够正常通信。如果不采用相同波特率,往往接收到的就都是乱码数据。

所谓波特率就是每秒传输二进制数码的位数,以位/秒(b/s)为单位,亦称“波特”。

典型的波特率有 9600b/s、38400b/s 和 115200b/s 等。

5. TTL 串口、RS-232 串口与 RS-485 总线

串口通信虽然适合远距离通信,但是随着两个设备间距离的增大,信号在电缆中衰减加大,同时所受干扰也会增多,这样就会导致信号发送错乱,因此不同方式的串口通信传输距离一定有一个上限显示。目前,经常使用的串口通信连接方式有 3 中,分别为 TTL 串口、RS-232 串口与 RS-485 总线。

TTL 串口是嵌入式设备端口一般默认支持的,不需要外加器件就可以实现两个 TTL 串口之间的连接。TTL 全称为 Transistor-Transistor Logic,即逻辑门电路,其输出高电平为 2.4~5V,输出低电平为 0~0.4V。使用 TTL 串口连接的距离一般限制在 2 米,连接方式如图 5.3 所示。

RS-232 串口是一种差分电路信号,规定逻辑“1”的电平为 -15~-5 V,逻辑“0”的电平 5V~15V。而在嵌入式芯片中一般工作电压为 0~3.3V,因此如果需要 RS-232 通信,一般需把芯片的 TTL 串口引脚连接到 TTL 转 RS-232 通信芯片上,从而支持 R2-232 通信。典型 RS-232 通信转换芯片有 MAX232 等,如图 5.5 所示。RS-232 串口通信距离一般限定在 15 米以内。

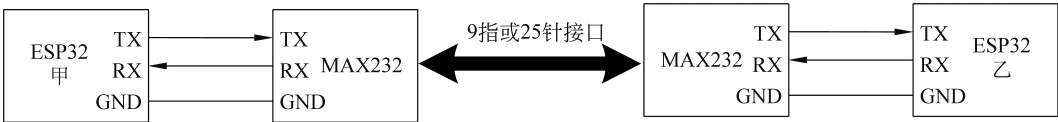


图 5.5 RS-232 通信示意图

RS-485(EIA-485 标准)总线是一种多设备、远距离数据传输方案。上述 TTL 串口和 RS-232 串口只能进行两个设备之间的直接连接,不能同时连接多个设备,如果需要用串口方式连接多个设备,可以采用 RS-485 模式。RS-485 通信只需要两条线路即可,不需要共地信号。其信号也采用差分方式,两个线缆的电压差大于 1.5V 即可进行通信,共模电压为 -7~12V。由于采用差分信号,因此只能采用半双工工作模式,通信效率相对较低。传输距离一般限定在 1.5 千米以内。此种方式也需要外接电路支持,典型的 RS-485 转换芯片为 MAX485、MAX1487 等。总线上最多连接设备数量根据芯片不同数据不同,一般为 32~

256 个。连接方式如图 5.6 所示。

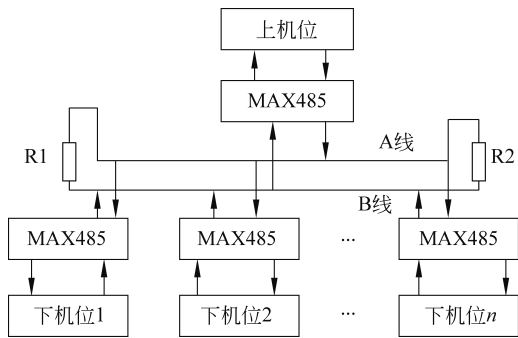


图 5.6 RS-485 连接示意图

提示：上述 3 种通信方式的限定距离一般随着波特率的增大而缩短。同时要注意，这 3 种方式的通信设备之间不能连接通信。如果连接了，由于工作电压不一致，有可能导致设备通信端口损坏。

5.1.2 ESP32-S3 开发板串口介绍

根据 ESP32-S3 开发板官方手册，ESP32-S3 芯片共有 3 个串口，分别是 UART0、UART1 和 UART2 通信速率可达到 5Mb/s。这 3 个串口原理上可以选择芯片任意通用 GPIO 引脚配置为 TX 和 RX 引脚。但是在 MicroPython 中，需要利用 1 个串口进行 REPL 通信，系统采用的是 UART0 进行 REPL 通信，因此供用户使用的只有 UART1 和 UART2。

在 YD-ESP32-S3 开发板中，共提供了两个 Type-C 类型的 USB 转串口端口(USB 和 COM 端口)。其中 Type-C 的 COM 端口连接 ESP32-S3 芯片，如图 5.7 所示，通过 CH343P 芯片实现 USB 转 TTL 串口的功能，连接到 ESP32-S3 芯片的 U0RXD(GPIO44) 和 U0TXD(GPIO43) 引脚。注意，在程序中使用这两个引脚时，要使用引脚号 44 和 43 进行配置，且只能配置为 UART1 或 UART2 的引脚，不能配置为 UART0 的引脚。

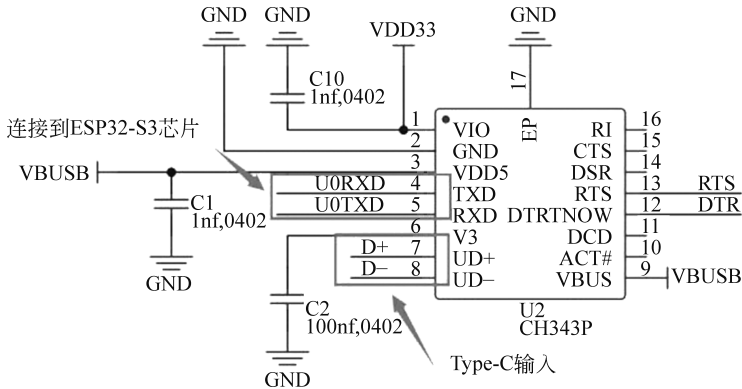


图 5.7 COM 端口

Type-C 的 USB 端口连接 ESP32-S3 芯片，如图 5.8 所示。外部信号经过 TYPEC-304-BCP16 芯片后连接到 ESP32-S3 的 GPIO19 和 GPIO20 引脚。这两个引脚启用的是 USB

功能,默认是在 debug 中进行 REPL 通信,用户不能修改,直接调用 print()和 input()函数使用即可。

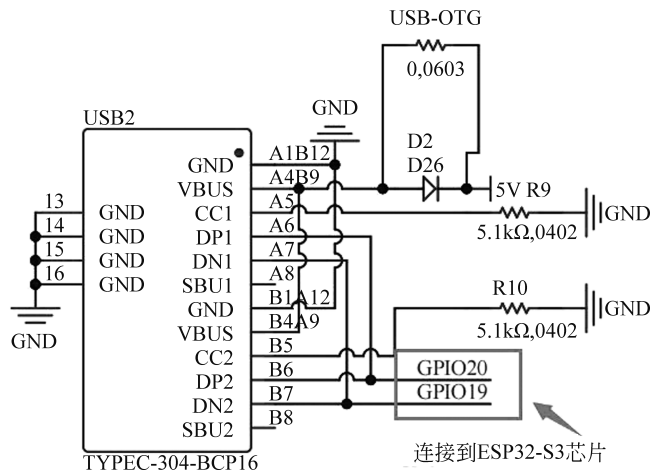


图 5.8 USB 端口

5.1.3 MicroPython 中串口 UART 类

在 MicroPython 中与串口相关的库为 UART 库,使用此库时需要使用代码 from machine import UART 导入库。UART 库中主要包括串口的初始化和串口发送与接收方法,UART 类构造函数与方法如表 5.1 所示。

表 5.1 UART 类构造函数与方法

类 方 法	说 明	示 例
machine.UART(id,baudrate=9600,bits=8,parity=None,stop=1,tx,rx)	构造给定 id 的 UART 对象。参数 id 的取值为 1 或 2;baudrate 为波特率,默认为 9600b/s;bits 为数据有效位数,取值为 7、8 或 9,默认为 8;parity 是奇偶校验,取值为无校验 None、偶校验 0 或奇校验 1;stop 是停止位的数量,取值为 1 或 2;tx 为发送数据引脚号;rx 为接收数据引脚号	myuart1=UART(1, baudrate=9600,bits=8, parity=None,stop=1, tx=43,rx=44)
UART.init(baudrate=9600, bits=8,parity=None, stop=1,tx,rx)	使用给定的参数初始化 UART 总线	myuart1.init(baudrate=9600,bits=8,parity=None, stop=1,tx=43,rx=44)
UART.deinit()	关闭 UART 总线	myuart1.deinit()
UART.any()	返回一个整数,计算可以在不阻塞的情况下读取的字符数。如果没有可用字符,则返回 0;如果有字符,则返回大于 0 的整数。即使有多个字符可供读取,该方法也可能返回 1	x=myuart1.any()
UART.read([nbytes])	读取字符。如果 nbytes 指定,则最多读取 nbytes 字节;否则,读取尽可能多的数据。如果超时,则返回 None。注意读取的结果为 bytes 类型	data1=myuart1.read()

续表

类 方 法	说 明	示 例
UART.readinto(buf[,nbytes])	将字节读入 buf 中。如果 nbytes 指定,则最多读取 nbytes 字节。否则,最多读取 len(buf)字节。如果超时,则返回 None	myuart1.readinto(buf01)
UART.readline()	读取一行,以换行符结尾。如果超时,则返回 None	data2=myuart1.readline()
UART.write(buf)	将字节缓冲区 buf 内容写入总线,进行发送。返回值为写入字节数或 None	myuart1.write("abc")

注意:上述 UART 库中的方法,在 ESP32-S3 中只能用在 UART1 和 UART2 中。而在 MicroPython 中,print()和 input()函数也可以用于串口通信,但这两个函数使用的是在 REPL 窗口中的数据。



5.2 UART 串口输出

5.2.1 DIY 开发板硬件原理图分析

在 DIY 开发板上,除去 Type-C 的 COM 端口外,没有单独引出特殊引脚作为串口使用,如果不使用 Type-C 的 COM 端口进行连接,可以利用 YD-ESP32-S3 核心开发板外接引脚中的通用 GPIO 引脚作为串口使用。

本节案例可以利用 Type-C 接口 USB 数据线连接 PC 和 YD-ESP32-S3 核心开发板的 COM 端口,同时为了下载程序和使用 REPL 环境进行调试,也需要连接 YD-ESP32-S3 核心开发板的 USB 端口,就是说 DIY 开发板共连接了两个 USB 端口,在 PC 中产生两个串口号,如图 5.9 所示。需要注意区分两个串口号对应的端口,通过插拔 USB 数据线可以去除和再产生串口号,其中标识“CH343”文字的是 COM 串口号,另一个为 USB 串口号。

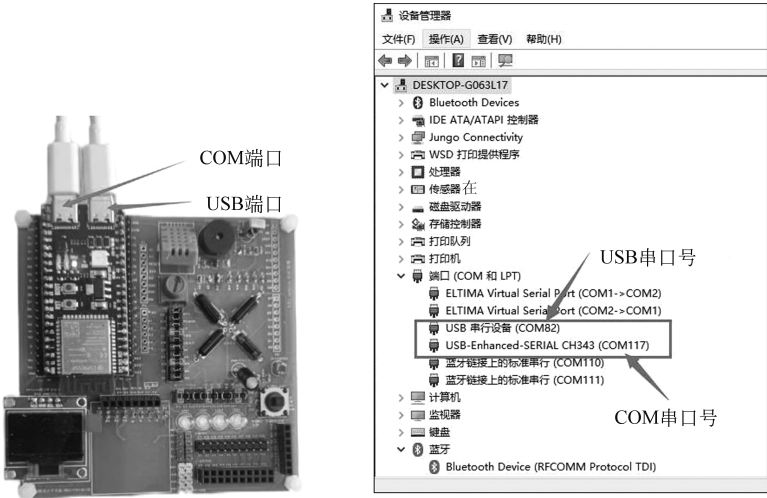


图 5.9 DIY 开发板串口连接

5.2.2 UART 串口初始化与输出使用

UART 串口的使用主要包括串口配置参数初始化,然后是串口的写和读。根据表 5.1 编写如下代码完成写串口操作,利用串口每 1 秒循环输出一次计数值。

```
from machine import Pin,UART
from utime import sleep
# 串口号使用 1 或 2,不能使用 0
# 串口 1 初始化,引脚为 43 和 44(U0RXD(GPIO44) 和 U0TXD(GPIO43))
myuart1=UART(1,baudrate=9600,bits=8,parity=None,stop=1,tx=43,rx=44)
myuart1.write("start game\r\n") #写串口
i=0
while True:
    i=i+1 #计数累计
    s1 = "i={}\r\n".format(i)
    myuart1.write(s1) #写串口,或者 myuart1.write(bytearray(s1,"utf-8"))
    sleep(1)
```

提示: 上述代码 myuart1.write(s1) 中,s1 为字符串,系统会自动把字符串(str 类型)转换为字节串(bytes 类型)输出。write() 函数中参数可以是字节串,会直接输出,同时 MicroPython 也支持串口中文输出,编码方式为 UTF-8 格式。

5.2.3 Wokwi 仿真串口输出呼吸灯占空比案例

【案例 5.1】 利用 UART1 循环输出案例 4.2 中呼吸灯占空比。

在 Wokwi 仿真中系统提供了 serialMonitor 虚拟调试设备,其为串口模拟调试器。可以通过配置工程中 diagram.json 文件来修改 serialMonitor 的连接引脚,连接到对应串口引脚即可完成串口监测,注意 serialMonitor 与 ESP32 串口引脚的连接采用交叉连接方式,即是 TX 连接 RX;同时 serialMonitor 支持的波特率只能为 9600b/s,无法在其他波特率下工作。

由于本案例中需要模拟 DIY 开发板 COM 端口输出,在 5.1.2 节中已经介绍 COM 端口连接的是开发板的 U0RXD(GPIO44) 和 U0TXD(GPIO43) 引脚,因此在程序代码中使用引脚号 44 和 43。但是在 diagram.json 文件中,这两个引脚比较特殊,不能直接使用引脚号,需要替换为对应的标识名字 TX 和 RX。基于案例 4.2 的 Wokwi 仿真工程,只要在 diagram.json 文件中修改 connections 属性中的 serialMonitor:RX 和 serialMonitor:TX 值即可,如图 5.10 所示。

编写如下代码在仿真环境中运行。

```
from machine import Pin,PWM,UART
import time
# 初始化 PWM 引脚为 GPIO11,频率为 1000Hz,占空比初值为 0
led1=PWM(Pin(11), freq=1000, duty_u16=0)
myuart1=UART(1,baudrate=9600,bits=8,parity=None,stop=1,tx=43,rx=44)
duty_value=0 # 占空比变量
fx=1 # 控制占空比增大和减小的方向
mycyctime=2000 # 一次亮灭变化的总时间为 2000ms
```

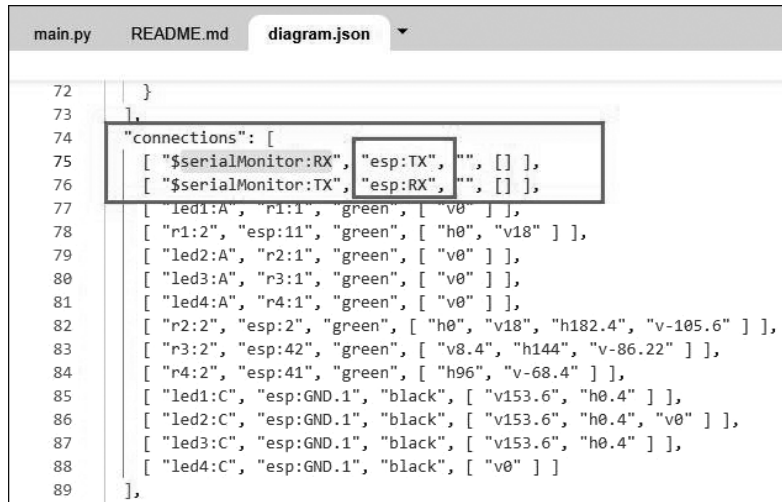



图 5.10 serialMonitor 参数配置

```
# 每次脉宽变化量为 820, 65535/820=80 次, 每个显示周期时间为 2000ms/80=25ms
mydelaytime=int(mycyctime/80)      # 显示周期时间
while True:
    if fx==1:                        # 变亮
        duty_value+=820              # 每次增加占空比 820, LED 变亮
        if duty_value>65535:
            duty_value=65535
            fx=0                      # 变暗方向
    else:
        duty_value-=820              # 每次减少占空比 820, LED 变暗
        if duty_value<0:
            duty_value = 0
            fx=1                      # 变量方向
    led1.duty_u16(duty_value)         # 修改占空比
    # 输出占空比(百分数形式, 保留 2 位小数)
    myuart1.write("占空比={:.2f}% \r\n".format(duty_value/65535 * 100))
    print("占空比={:.2f}% , mydelaytime={}\r\n".format(duty_value/65535 * 100,
mydelaytime))
    time.sleep_ms(mydelaytime)       # 延时 25ms
```

仿真程序正常运行后, 会在右下侧串口监测窗口输出串口 1 的输出结果, 如图 5.11 所示。

提示: 在上述案例中, 使用 ESP32 的 GPIO44 和 GPIO43 引脚是特殊引脚, 在 Wokwi 仿真中是 ESP32 的 REPL 仿真使用的引脚。如果在串口初始化中使用了这两个引脚, 就不能再用函数 print() 输出数据。读者也可以修改串口的引脚为其他引脚进行仿真。例如, 上述代码中串口的初始化代码可修改如下:

```
myuart1=UART(1,baudrate=9600,bits=8,parity=None,stop=1,tx=14,rx=13)
```

此时也需要对应修改 Wokwi 仿真案例中的“diagram.json”文件, 如下所示:

```
[ "$ serialMonitor:RX", "esp:13", "", [] ],
[ "$ serialMonitor:TX", "esp:14", "", [] ],
```

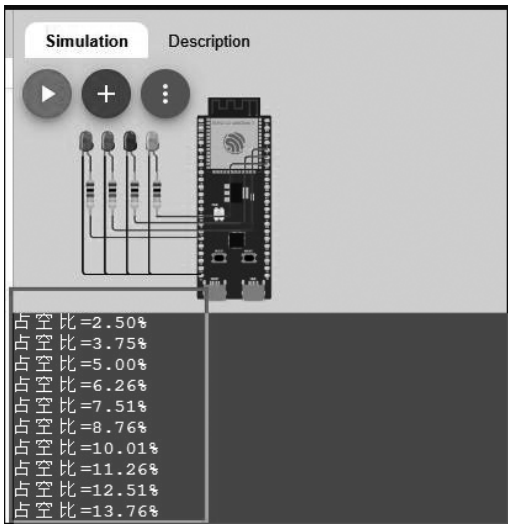


图 5.11 serialMonitor 串口仿真输出结果

读者可以观察 serialMonitor 监测窗口中的数据变化与之前的不一样之处。

5.2.4 DIY 开发板串口输出呼吸灯占空比案例

上述程序仿真通过后,可以把此代码下载到 DIY 开发板中运行,但是要注意,此程序要放在 main.py 中作为主程序运行。在此案例中,UART1 串口要发送数据给 PC,因此在 PC 中需要一个串口调试工具来接收此数据,推荐使用串口调试助手(UartAssist)软件。在 DIY 开发板运行程序后,在 PC 中接收的串口数据如图 5.12 所示。

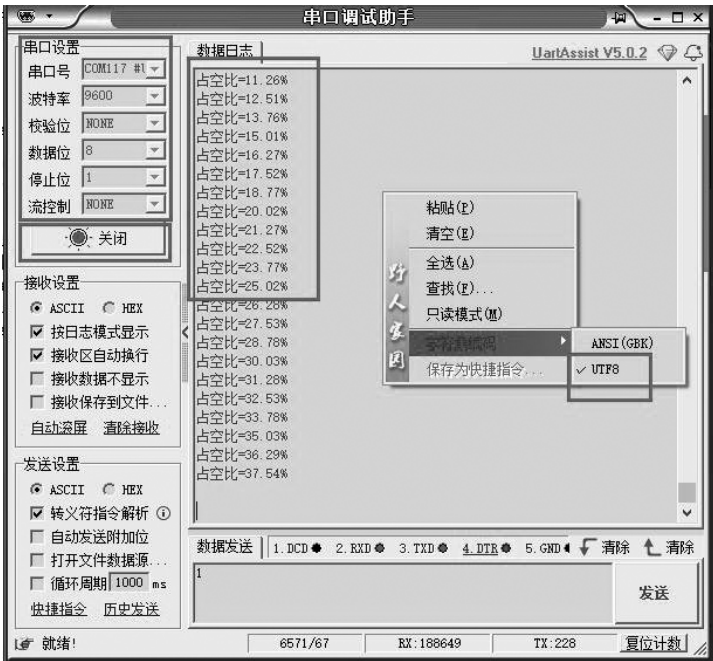


图 5.12 串口调试助手接收数据