Monday, July 19, 2021 8:40 PM

ESP8266/ESP32自动下载电路原理分析

[青山]

专业配置寄存器，吉他爱好者

背景

前一段时间需要自己制作一片ESP32单板，成本和封装考虑，计划选择CH340E作为USB转串口芯片，ESP8266/ESP32的单板一般都有自动下载电路，用户无需按钮即可令单板自动进入下载模式实现固件烧录，然而自动下载电路需要串口芯片支持DTR和RTS，CH340E却只有RTS信号，没有DTR信号，于是研究学习了一下自动下载电路的原理，准备用一些奇淫技巧解决CH340E的自动下载问题。

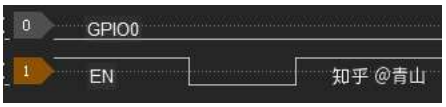
遗憾的是，目前的中文互联网上，关于ESP8266/ESP32自动下载原理，所有能搜索到的解释**全部都是错误**的，差之毫厘，谬以千里，如果随意的假设，自以为是，最终得出的只能是一厢情愿的结论。

以下分析的硬件和源码基于手头的一块ESP32-S2开发板，对于其他ESP开发板，原理也是一样的。

下载模式

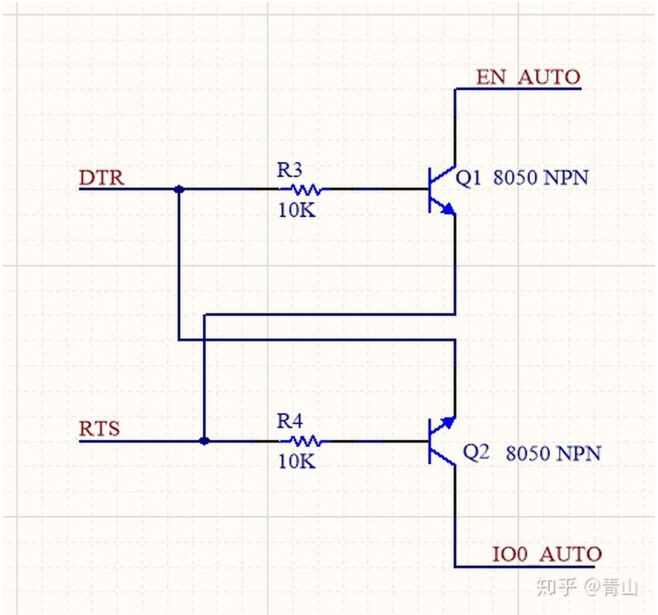
ESP8266/ESP32进入下载模式的条件很简单：

EN（也称为RST）上升沿时候GPIO0保持为低电平，如图所示，



分析-1

下载电路如下所示，其结构与RS触发器比较类似，注意EN和IO0信号均连接在三极管集电极，通过控制三极管只能拉低此信号，若三极管截止，则此信号的状态由其他电路决定（一般来说，此类信号会默认接电阻上拉到VCC）



逻辑关系如下

DTR = 0; RTS = 0, 此时Q1截止，Q2截止， EN = 1; IO0 = 1

DTR = 0; RTS = 1，此时Q1截止，Q2导通, EN = 1; IO0 = 0

DTR = 1; RTS = 0, 此时Q1导通，Q2截止, EN = 0; IO0 = 1

DTR = 1; RTS = 1, 此时Q1截止，Q2截止, EN = 1; IO0 = 1

列表如下

DTR RTS EN IO0
0 0 1 1
0 1 1 0
1 0 0 1
1 1 1 1

简单总结：当DTR和RTS同时为0或者同时为1时，三极管Q1和Q2均为截止状态，此时EN和IO0的状态由其他电路决定（内部/外部上拉电阻）。

当不同时为0或者1时：

EN = RTS
IO0 = DTR

注意这种逻辑下 EN和IO0是不可能同时为0的，然而进入下载模式则需要如下的序列

1. IO = 0; EN = 0
2. IO = 0; EN 0 -> 1

从逻辑表上看是根本无法正常进入下载模式的，此为疑惑1。

分析-2

再来继续分析一下esptool.py里下载相关的代码

```
    # issue reset-to-bootloader:
    # RTS = either CH_PD/EN or nRESET (both active low = chip in reset
    # DTR = GPIO0 (active low = boot to flasher)
    #
    # DTR & RTS are active low signals,
    # ie True = pin @ 0V, False = pin @ VCC.
    if mode != 'no_reset':
      self._setDTR(False)  # IO0=HIGH
    1) self._setRTS(True)   # EN=LOW, chip in reset
      time.sleep(0.1)
    2) self._setDTR(True)   # IO0=LOW
    3) self._setRTS(False)  # EN=HIGH, chip out of reset
      time.sleep(0.05)
    4) self._setDTR(False)  # IO0=HIGH, done
```

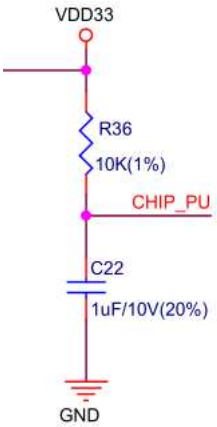
注意True是低电平，False为高电平，另外代码中的setDTR()和setRTS()两条语句之间虽然看上去紧挨着没有延时，然而由于这里是高级语言python，两条语句之间的延时并不能忽略，因此分析的时候必须依次的进行状态分析，以下分为四个阶段依次分析

1) 设置DTR = 1; RTS = 0, 此时Q1导通，Q2截止, EN = 0; IO0 = 1
2) 设置DTR = 0; RTS = 0, 此时Q1截止，Q2截止, EN = 1; IO0 = 1
3) 设置DTR = 0; RTS = 1, 此时Q1截止，Q2导通, EN = 1; IO0 = 0
4) 设置DTR = 1; RTS = 1, 此时Q1截止，Q2截止, EN = 1; IO0 = 1

如果按照上面的代码分析来做结论，不论如何系统也是不可能进入下载模式的：EN和IO0首先不可能同时为0，EN由0->1的上升沿IO0也并不为0，再次确认之前的疑惑，那么系统究竟是如何进入下载模式的呢?

答案

问题的答案实际在另外一部分电路，原理其实非常简单：EN信号连接在一个电容充放电电路上



CHIP_PU即EN，代码中23阶段之后会延时一段时间，而EN由于电容充电，电平并不会立马变为高电平，而是缓慢上升，以如上参数为例计算，同时参考芯片电气参数特性

3.4 DC Characteristics (3.3 V, 25 °C)

Table 9: DC Characteristics (3.3 V, 25 °C)

Symbol	Parameter	Min	Typ	Max	Unit
C_{IN}	Pin capacitance	—	2	—	pF
V_{IH}	High-level input voltage	0.75 × VDD	—	VDD + 0.3	V
V_{IL}	Low-level input voltage	–0.3	—	0.25 × VDD	V
I_{IH}	High-level input current	—	—	50	nA
I_{IL}	Low-level input current	—	—	50	nA
V_{OH}	High-level output voltage	0.8 × VDD	—	—	V
V_{OL}	Low-level output voltage	—	—	0.1 × VDD	V
I_{OH}	High-level source current (VDD = 3.3 V, V_{OH} >= 2.64 V, PAD_DRIVER = 3)	—	40	—	mA
I_{OL}	Low-level sink current (VDD = 3.3 V, V_{OL} = 0.495 V, PAD_DRIVER = 3)	—	28	—	mA
R_{PU}	Pull-up resistor	—	45	—	kΩ
R_{PD}	Pull-down resistor	—	45	—	kΩ
V_{IH_nRST}	Chip reset release voltage	0.75 × VDD	—	VDD + 0.3	V
V_{IL_nRST}	Chip reset voltage	–0.3	—	0.25 × VDD	V

Note:
VDD is the I/O voltage for a particular power domain of pins.

高电平为0.75VDD，则达到高电平按照如下公式计算：

解得t = 14ms，即EN经过14ms上升到电平1，在实际代码中延时了50ms的等待时间，以确保延时后EN处于电平1的状态。
另外需要提的是，阶段1需要等待一段时间，让电容放电，保证EN电平下降到电平0，才能保证系统正常复位，在代码中预留了100ms的等待时间，同样可以通过电容放电公式计算出放电到电平0需要的时间，感兴趣的朋友可以自行根据公式计算确认。

参考

https://github.com/espressif/esptool/issues/136
https://www.esp32.com/viewtopic.php?t=5731

编辑于 2020-06-02