ESP8266 UART使用说明

Version 0.2

Espressif Systems IOT Team

免责申明和版权公告

本文中的信息，包括供参考的URL地址，如有变更，恕不另行通知。

文档"按现状"提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。

Wi-Fi联盟成员标志归Wi-Fi联盟所有。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

Table of Contents

1. 配置属性参数

ESP8266共有两组UART接口，分别为：

UART0:

U0TXD: pin26(U0TXD)

U0RXD: pin25(U0RXD)

U0CTS: pin12(MTCK)

U0RTS: pin13(MTDO)

UART1:

U1TXD: pin14(GPIO2)

发送FIFO的基本工作过程：

只要有数据填充到发送FIFO 里，就会立即启动发送过程。由于发送本身是个相对缓慢的过程，因此在发送的同时其它需要发送的数据还可以继续填充到发送 FIFO 里。当发送 FIFO 被填满时就不能再继续填充了，否则会造成数据丢失，此时只能等待。发送 FIFO 会按照填入数据的先后顺序把数据一个个发送出去，直到发送 FIFO 全空时为止。已发送完毕的数据会被自动清除，在发送FIFO 里同时会多出一个空位。

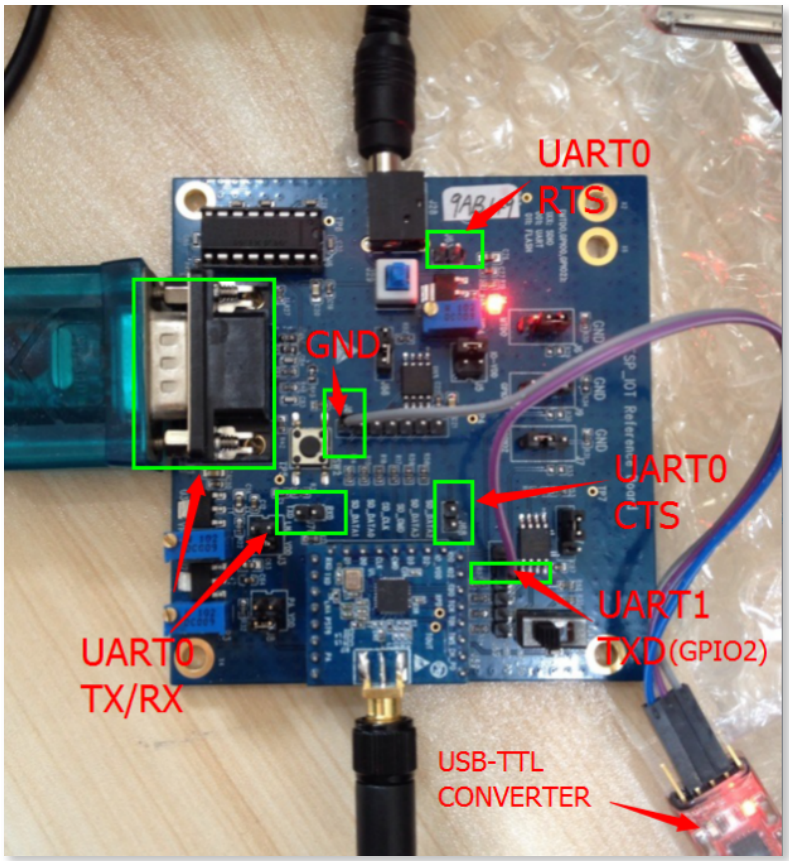
接收FIFO的基本工作过程：

当硬件逻辑接收到数据时，就会往接收FIFO 里填充接收到的数据。程序应当及时取走这些数据，数据被取走也是在接收FIFO 里被自动删除的过程，因此在接收 FIFO 里同时会多出一个空位。如果在接收 FIFO 里的数据未被及时取走而造成接收FIFO 已满，则以后再接收到数据时因无空位可以填充而造成数据丢失。

应用场景：

UART0作为数据通信接口，UART1作为debug信息的打印。

UART0默认情况会在上电booting期间输出一些打印，此期间打印内容的波特率与所用的外部晶振频率有关。使用40M晶振时，该段打印波特率为115200。使用26M晶振时，该段打印波特率为74880。



如果这段打印对应用的功能产生影响，可以用第四节的方法间接屏蔽上电时期的打印输出。

2. 硬件资源

UART0和UART1各有一个长度为128Byte的硬件FIFO，读写FIFO都在同一个地址操作。

两个UART模块的硬件寄存器相同，通过UART0/UART1的宏定义来区分。

3. 参数配置

UART属性参数都在UART_CONF0定义的寄存器中，可以在uart_register.h中找到。修改该寄存器下的不同对应位，可以配置UART属性。

3.1. 波特率

ESP8266的串口波特率范围从300到115200*40都可以支持。

接口：void UART_SetBaudrate(uint8 uart_no,uint32 baud_rate);

3.2. 校验位

#define UART_PARITY_EN (BIT(1)) 校验使能：1：enable； 0：disable

#define UART_PARITY (BIT(0)) 校验类型设置 1：奇校验；0：偶校验

接口：void UART_SetParity(uint8 uart_no, UartParityMode Parity_mode);

3.3. 数据位

#define UART_BIT_NUM 0x00000003 //数据位长度占用两个bit：

设置这两个bit可以配置数据长度0: 5bit ; 1: 6bit ; 2: 7bit ; 3: 8bit

#define UART_BIT_NUM_S 2 //寄存器偏移为2（第2bit开始）

接口：void UART_SetWordLength(uint8 uart_no, UartBitsNum4Char len)

3.4. 停止位

#define UART_STOP_BIT_NUM 0x00000003 //数据位长度占用两个bit

设置这两个bit可以配置停止位长度 1 : 1bit ; 2 : 1.5bit ; 3 : 2bit

#define UART_STOP_BIT_NUM_S 4 //寄存器偏移为4（第4bit开始）

接口：void UART_SetStopBits(uint8 uart_no, UartStopBitsNum bit_num);

3.5. 反相

UART各个信号输入与输出信号，可在内部进行反向配置。

#define UART_DTR_INV (BIT(24))

#define UART_RTS_INV (BIT(23))

#define UART_TXD_INV (BIT(22))

#define UART_DSR_INV (BIT(21))

#define UART_CTS_INV (BIT(20))

#define UART_RXD_INV (BIT(19))

将对应寄存器置位，可以将对应信号线反向输出/输入。

接口：void UART_SetLineInverse(uint8 uart_no, UART_LineLevelInverse inverse_mask);

3.6. 切换打印函数输出端口

默认情况下，系统打印函数os_printf从uart0口输出内容，通过以下接口可以设置从uart0或者uart1口输出打印。

void UART_SetPrintPort(uint8 uart_no);

3.7. 读取tx/rx 队列内当前剩余的字节数

Tx fifo length:

(READ_PERI_REG(UART_STATUS(uart_no))>>UART_TXFIFO_CNT_S)

&UART_TXFIFO_CNT;

接口：TX_FIFO_LEN(uart_no)

Rx fifo length:

(READ_PERI_REG(UART_STATUS(UART0))>>UART_RXFIFO_CNT_S)

&UART_RXFIFO_CNT;

接口：RF_FIFO_LEN(uart_no)

3.8. 回环操作（loop-back)

在UART_CONF0寄存器中,配置后，uart tx/rx在内部短接。

#define UART_LOOPBACK (BIT(14)) //回环使能位，1: enable;0: disable

ENABLE: SET_PERI_REG_MASK(UART_CONF0(UART0), UART_LOOPBACK);

接口：ENABLE_LOOP_BACK(uart_no)

DISABLE:CLEAR_PERI_REG_MASK(UART_CONF0(UART0), UART_LOOPBACK);

接口：DISABLE_LOOP_BACK(uart_no)

3.9. 线中止信号

要产生线上中止信号，可以将UART_TXD_BRK置1，这样在uart发送队列发送完成后，输出一个break信号（tx输出低电平），需要停止输出将该位置0。

#define UART_TXD_BRK (BIT(8)) //线中止信号，1:enable ;0: disable

3.10. 流量控制

配置过程:

a. 先配置uart0的pin12,、pin13脚复用为U0CTS,和U0RTS功能。

#define FUNC_U0RTS 4

#define FUNC_U0CTS 4
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDO_U, FUNC_U0RTS);
PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTCK_U, FUNC_U0CTS);

b. 接收方向的硬件流控可以配置阈值，当rx fifo中的长度大于所设的阈值，U0RTS脚就会拉高，阻止对方发送。

配置接收流控阈值：
阈值相关的配置一般都在UART_CONF1定义的寄存器中。
#define UART_RX_FLOW_EN (BIT(23)) 第23bit使能接收流控: 0: disable; 1: enable
#define UART_RX_FLOW_THRHD 0x0000007F //门限值，占用7bit，范围0-127
#define UART_RX_FLOW_THRHD_S 16 //寄存器偏移为16（第16bit开始）

c. 发送方向的流控只需配置使能，该寄存器在UART_CONF0中：
#define UART_TX_FLOW_EN (BIT(15)) 使能发送流控: 0: disable ; 1: enable

d. 接口：
Void UART_SetFlowCtrl(uint8 uart_no,UART_HwFlowCtrl flow_ctrl,uint8 rx_thresh);

e. demo板硬件连接：
需要将J68(U0CTS)与J63(U0RTS)的跳线接上。

3.11. 其他接口

TX_FIFO_LEN(uart_no) //宏定义，发送队列当前长度
RF_FIFO_LEN(uart_no) //宏定义，接收队列当前长度

4. 配置中断

由于所有中断事件在发送到中断控制器之前会一起进行"或运算"操作，所以任意时刻 UART 只能向中断产生一个中断请求。通过查询中断状态函数UART_INT_ST(uart_no)，软件可以在同一个中断服务函数里处理多个中断事件（多个并列的if 语句）。

4.1. 中断寄存器

Uart的中断寄存器有：

UART_INT_RAW 中断原始状态寄存器

UART_INT_ENA 中断使能寄存器：表示当前使能的uart中断。

UART_INT_ST 中断状态寄存器：表示当前有效的中断状态

UART_INT_CLR 清除中断寄存器：置对应位来清除中断状态寄存器

4.2. 接口

打开中断使能：UART_ENABLE_INTR_MASK(uart_no,ena_mask);

关闭中断使能：UART_DISABLE_INTR_MASK(uart_no,disable_mask);

清除中断状态：UART_CLR_INTR_STATUS_MASK(uart_no,clr_mask);

获取中断状态：UART_GET_INTR_STATUS(uart_no);

4.3. 中断类型

1. 接收full中断

中断状态位：UART_RXFIFO_FULL_INT_ST

定义：当配置阈值并使能中断后，当rx fifo中的数据长度大于阈值后，触发该中断。

应用：比较多用于处理uart接收的数据，配合流量控制，直接处理或者post出消息,或者转存入buffer。比如，配置阈值为100，并使能full中断，当串口收到100字节后，会触发full中断。

配置阈值：

full中断阈值（或门限值）

在UART_CONF1寄存器

#define UART_RXFIFO_FULL_THRHD 0x0000007F //门限值mask，7bit长，范围0-127

#define UART_RXFIFO_FULL_THRHD_S 0 //寄存器偏移为0（第0bit开始）

设置中断使能：
在UART_INT_ENA寄存器
#define UART_RXFIFO_FULL_INT_ENA (BIT(0)) //full中断使能位，1: enable;0: disable

清除中断状态：
对于full中断比较特殊，需要先将接收fifo中的数据全部读空，然后写清除中断状态寄存器。否则退出后中断状态位还是会被置上。
详见中断处理实例。

2. 接收溢出中断

中断状态位：UART_RXFIFO_OVF_INT_ST

定义：当使能接收溢出中断后，当接收队列的长度大于队列总长度(128bytes)时，会触发该中断信号。

触发场景：一般只在没有设置流控的情况下，因为有流量控制时候不会发生溢出。区别于full中断，full中断是人为设置阈值并且数据不会丢失。溢出中断触发则一般都会存在数据丢失。可用于程序调试与验错。

设置中断使能：
在UART_INT_ENA寄存器
#define UART_RXFIFO_OVF_INT_ENA (BIT(4)) //溢出中断使能位: 1: enable; 0: disable

清除中断状态：
读取队列值，使队列长度小于128，然后置清除中断状态寄存器即可。

3. 接收超时中断tout

中断状态位：UART_RXFIFO_TOUT_INT_ST

定义：当配置tout阈值并使能中断后，当uart开始接收数据后，停止传输的时间超过所设定的门限值，就会触发tout中断。

应用：较多用于处理串口命令或者数据，直接处理数据或者post出消息,或者转存入buffer。

配置阈值与功能使能：
tout中断阈值（或门限值）在UART_CONF1寄存器中。
Tout阈值的单位为8个uart数据比特的时间（近似一个byte）。
#define UART_RX_TOUT_EN (BIT(31)) //超时功能使能位: 1: enable;0: disable
#define UART_RX_TOUT_THRHD 0x0000007F //超时阈值配置位，共7位，范围0-127

#define UART_RX_TOUT_THRHD_S 24 //寄存器偏移为24（第24bit开始）

设置中断使能：

在UART_INT_ENA寄存器

#define UART_RXFIFO_TOUT_INT_ENA (BIT(8)) tout //中断使能位，1: enable;0: disable

清除中断状态：

与full中断类似，tout中断也需要先将接收fifo中的数据全部读空，然后写清除中断状态寄存器。否则退出后中断状态位还是会被置上。

详见中断处理实例。

4. 发送fifo空中断

中断状态位：UART_TXFIFO_EMPTY_INT_ST

定义：当配置empty阈值并使能中断后，当uart发送fifo内的数据小于所设阈值时，会触发empty中断。

应用：可用于实现自动将buffer中的数据转发至uart。需要中断处理配合。 例如，将empty门限值设为5，则tx fifo长度小于5个字节时，触发empty中断，在empty中断的中断处理中，从buffer取数把tx fifo填满(操作fifo的速度远大于tx fifo发送的速度）。这样继续循环，直到 buffer的数据都被发送完毕，将empty中断关闭即可。

配置阈值：

empty中断阈值（或门限值）在UART_CONF1寄存器中

#define UART_TXFIFO_EMPTY_THRHD 0x0000007F //发送队列空中断阈值配置位，共7位，范围0-127

#define UART_TXFIFO_EMPTY_THRHD_S 8 //寄存器偏移为8（第8bit开始）

设置中断使能：

在UART_INT_ENA寄存器

#define UART_TXFIFO_EMPTY_INT_ENA (BIT(1)) //empty中断使能位，1: enable;0: disable

清除中断状态：

向发送队列填数，高于门限值，并清除对应的中断状态位。如果没有数据需要发送，需要关闭此中断使能位。

详见中断处理实例。

5. 错误检测类中断

中断状态位：

奇偶校验错误中断：UART_PARITY_ERR_INT_ST

线终止错误中断(line-break)：UART_BRK_DET_INT_ST

接收帧错误中断： UART_FRM_ERR_INT_ST

定义：

奇偶校验错误中断(parity_err):接收到的字节存在奇偶校验错误。

线终止错误中断(BRK_DET):接收到break信号，或者接收到错误的起始条件(rx线一直为低电平）。

接收帧错误中断(frm_err): 停止位不为1。

应用：一般都用于错误检测。

设置中断使能：

在UART_INT_ENA寄存器，

#define UART_PARITY_ERR_INT_ENA (BIT(2)) //奇偶校验错误中断使能位，1:enable;0:disable

#define UART_BRK_DET_INT_ENA (BIT(7)) //线终止错误中断使能位，1: enable;0: disable

#define UART_FRM_ERR_INT_ENA (BIT(3)) //接收帧错误中断使能位，1: enable;0: disable

清除中断状态：

对错误进行相应处理后，清除中断状态位即可。

6. 流量控制状态中断

中断状态位：

UART_CTS_CHG_INT_ST

UART_DSR_CHG_INT_ST

定义：当CTS、DSR引脚线上电平改变时触发该中断。

应用：一般配合流量控制使用，当触发该中断后，检查对应流控线状态，如为高电平，则停止向tx队列写数。

#define UART_CTS_CHG_INT_ST (BIT(6))

#define UART_DSR_CHG_INT_ST (BIT(5))

设置中断使能：

在UART_INT_ENA寄存器，

#define UART_CTS_CHG_INT_ENA (BIT(6)) CTS //线状态中断使能位，1:enable;0:disable

#define UART_DSR_CHG_INT_ENA (BIT(5)) DSR //线状态中断使能位，1:enable;0:disable

清除中断状态：

对错误进行相应处理后，清除中断状态位即可。

5. 中断处理函数示例流程

```
LOCAL void
uart0_rx_intr_handler(void *para)
{
    /* uart0 and uart1 intr combine togther, when interrupt occur, see reg 0x3ff20020, bit2, bit0 represents
    * uart1 and uart0 respectively
    */
    uint8 RcvChar;
    uint8 uart_no = UART0;//UartDev.buff_uart_no;
    uint8 fifo_len = 0;
    uint8 buf_idx = 0;
    uint32 uart_intr_status = READ_PERI_REG(UART_INT_ST(uart_no)) ;//get uart intr status
    while (uart_intr_status != 0x0) { //while intr status is not cleared
        if (UART_FRM_ERR_INT_ST == (uart_intr_status & UART_FRM_ERR_INT_ST)) { //if it is caused by a frm_err interrupt
            WRITE_PERI_REG(UART_INT_CLR(uart_no), UART_FRM_ERR_INT_CLR);
        } else if (UART_RXFIFO_FULL_INT_ST == (uart_intr_status & UART_RXFIFO_FULL_INT_ST)) {//if it is caused by a fifo_full interrupt
            fifo_len = (READ_PERI_REG(UART_STATUS(UART0)) >> UART_RXFIFO_CNT_S)&UART_RXFIFO_CNT;  //read rf fifo length
            buf_idx = 0;
            //os_printf("full len:%d\n\r",fifo_len);//for dbg
            while (buf_idx < fifo_len) {                                                    //read all the data in rx fifo
                uart_tx_one_char(UART0, READ_PERI_REG(UART_FIFO(UART0)) & 0xFF);
                buf_idx++;
            }
            WRITE_PERI_REG(UART_INT_CLR(UART0), UART_RXFIFO_FULL_INT_CLR);              //clear full interrupt state
        } else if (UART_RXFIFO_TOUT_INT_ST == (uart_intr_status & UART_RXFIFO_TOUT_INT_ST)) {//if it is caused by a time_out interrupt
            fifo_len = (READ_PERI_REG(UART_STATUS(UART0)) >> UART_RXFIFO_CNT_S)&UART_RXFIFO_CNT;  //read fifo length
            buf_idx = 0;
            //os_printf("tout len:%d\n\r",fifo_len);//for dbg
            while (buf_idx < fifo_len) {                                                //read all the data in rx fifo
                uart_tx_one_char(UART0, READ_PERI_REG(UART_FIFO(UART0)) & 0xFF);
                buf_idx++;
            }
            WRITE_PERI_REG(UART_INT_CLR(UART0), UART_RXFIFO_TOUT_INT_CLR);              //clear rx tout interrupt state
        } else if (UART_TXFIFO_EMPTY_INT_ST == (uart_intr_status & UART_TXFIFO_EMPTY_INT_ST)) {//if it is caused by a tx_empty interrupt
            //uart1_sendStr_no_wait("empty\n\r");//for dbg
            WRITE_PERI_REG(UART_INT_CLR(uart_no), UART_TXFIFO_EMPTY_INT_CLR);
            CLEAR_PERI_REG_MASK(UART_INT_ENA(UART0), UART_TXFIFO_EMPTY_INT_ENA);
        } else {
            //skip
        }
        uart_intr_status = READ_PERI_REG(UART_INT_ST(uart_no)) ;                        //update interrupt status
    } ? end while uart_intr_status!=0x0 ?
} ? end uart0_rx_intr_handler ?
```

6. 关于屏蔽上电打印

Esp8266在上电时候，uart0默认会输出一些打印信息，如果对此敏感的应用，可以使用uart的内部引脚交换功能，在初始化的时候，将U0TXD、U0RXD分别与U0RTS，U0CTS交换.

调用接口：void system_uart_swap(void);

初始化前：
UART0:
U0TXD: pin26(u0txd)
U0RXD:pin25(u0rxd)
U0CTS:pin12(mtck)
U0RTS: pin13(mtdo)

在初始化执行pin脚交换后，
U0TXD:pin13(mtdo)

U0RXD:pin12(mtck)
U0CTS: pin25(u0rxd)
U0RTS: pin26(u0txd)
硬件上pin13与pin12作为uart0的收发脚，在上电 启动阶段不会有打印输出，但要注意保证pin13（mtdo）在esp8266启动阶段不能被外部拉高。