ESP8266 UART User Guide

Version 0.2

Espressif Systems IOT Team

http://bbs.espressif.com/

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The WiFi Alliance Member Logo is a trademark of the WiFi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Table of Contents

1. Preambles

ESP8266 has two UART interfaces, the pin definitions of which are described below:

UART0	
U0TXD	pin26(U0TXD)
U0RXD	pin25(U0RXD)
U0CTS	pin12(MTCK)
U0RTS	pin13(MTDO)
UART1	
U1TXD	pin14(GPIO2)

1.1. UART TX FIFO Working Process

Once data is filled in UART TX FIFO, the data transmission process will be invoked. However, data transmission is a relatively slow process, thus other data that requires to be transmitted needs to be filled in the TX FIFO simultaneously until it is stuffed. At this point, no data should be filled but has to wait, otherwise the data will be lost. TX FIFO will send out the data according to the sequence that they are filled in. Data transmission will be ended until all TX FIFO have been sent out. Data that have been sent out will be cleared automatically, so that there will be a vacancy in TX FIFO.

1.2. UART RX FIFO Working Process

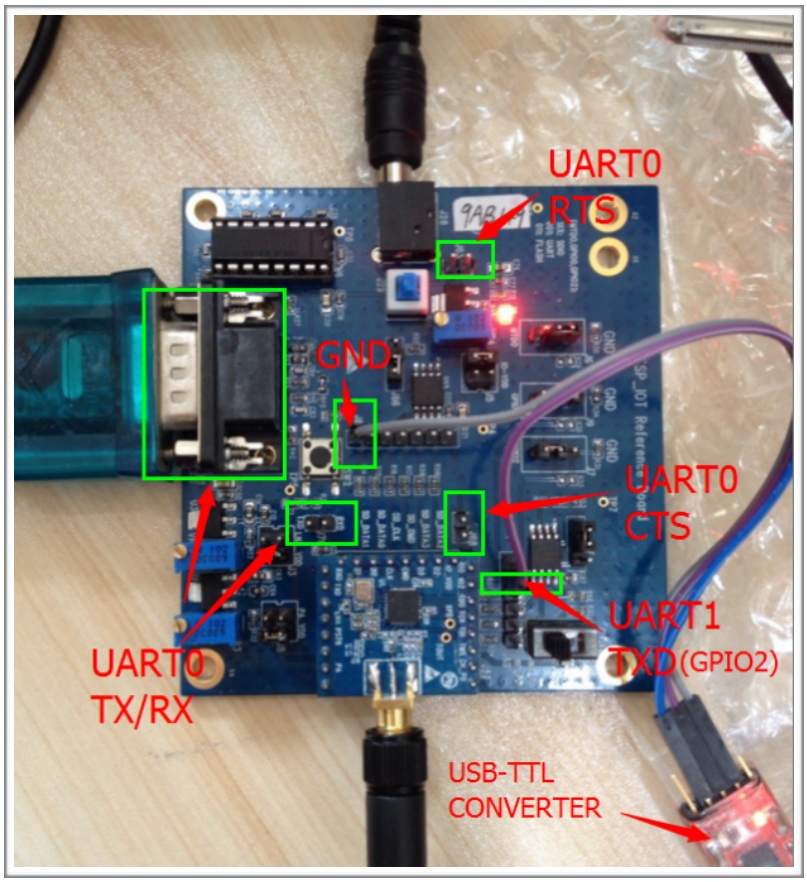
When data is received by the hardware logic, they will be filled in RX FIFO, and will be taken away be programs. Once all datum is taken away by the program, there will be a vacancy in RX FIFO. If the data received are not taken away timely, resulting in the stuffing of RX FIFO, then data losses will be caused since when there is no vacancy left for other data received.

1.3. Application Context

UART0 is used for data transmission and receiving, while UART1 is used for printing of debug information.

By default, some information will be print out by UART0 when the chip is powered on and is booting up. During this period, the baud rate of print information is related to the frequency of the external crystal applied. When the crystal frequency is 40MHz, the baud rate of the print information will be 115200, while when the crystal frequency is 26MHz, the baud rate will be 74880.

If the print information exerts impact on the functionality of the application of the device, print information during the booting-up period can be shielded following the steps described in Chapter Four.


UART0
RTS
GND
UART0
CTS
UART1
TXD(GPIO2)
UART0
TX/RX
USB-TTL
CONVERTER

2. Hardware Resources

There is one 128 Byte hardware FIFO for both **UART0** and **UART1** respectively. Read-and-write FIFO is operated at the same address. The hardware registers for both UART module is the same, which are differentiated by the macro definitions of **UART0** or **UART1**.

3. Parameters Configuration

Attributes of UART parameters, which can be found by `uart_register.h`, can be configured by registers defined by `UART_CONF0`. That's to say, by changing the data bit of the registers, attributes of UART parameters can be configured.

3.1. Baud Rate

The serial baud rate range that can be supported by ESP8266 is between 300 to 115200*40.

Interface function: `void UART_SetBaudrate(uint8 uart_no,uint32 baud_rate);`

3.2. Parity

```
#define UART_PARITY_EN  (BIT(1))
```

check the enable function: 1: enable; 0: disable

```
#define UART_PARITY  (BIT(0))
```

check the type of setting: 1: odd; 0: even

Interface function: `void UART_SetParity(uint8 uart_no, UartParityMode Parity_mode);`

3.3. Number Bit

```
#define UART_BIT_NUM  0x00000003
```

Two bits are occupied by the length of Number Bit. The length of the number can be configured by setting the two bits, for example: 0: 5bit; 1: 6bit; 2: 7bit; 3: 8bit

```
#define UART_BIT_NUM_S  2
```

Deviation of the register is 2 (counting from the second bit)

Interface function: `void UART_SetWordLength(uint8 uart_no, UartBitsNum4Char len)`

3.4. Stop Bit

```
#define UART_STOP_BIT_NUM      0x00000003
```

Two bits are occupied by the length of Stop Bit. The length of Stop Bit can be configured by setting the two bits, for example: 1: 1bit; 2: 1.5bit; 3 : 2bit

```
#define UART_STOP_BIT_NUM_S    4
```

Deviation of the register is 4 (counting from the forth bit)

Interface function: `void UART_SetStopBits(uint8 uart_no, UartStopBitsNum bit_num);`

3.5. Reverse Operation

All the input and output signals can be configured to the reverse direction internally.

```
#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))
```

By setting the allocation of registers, the input and output of the corresponding signal lines can be reversed.

Interface function: `void UART_SetLineInverse(uint8 uart_no, UART_LineLevelInverse inverse_mask);`

3.6. Switch the Output Terminal

By default, function printed by the system (`os_printf`) will be exported by port UART0. Print information exported by UART0 or UART1 can be configured via interface function:

```
void UART_SetPrintPort(uint8 uart_no);
```

3.7. Read the Left Number Length of TX/RX Queue

TX FIFO Length:

```
(READ_PERI_REG(UART_STATUS(uart_no))>>UART_TXFIFO_CNT_S)
&UART_TXFIFO_CNT;
```

Interface function: `TX_FIFO_LEN(uart_no)`

RX FIFO Length:

```
(READ_PERI_REG(UART_STATUS(UART0))>>UART_RXFIFO_CNT_S)
&UART_RXFIFO_CNT;
```

Interface function: `RF_FIFO_LEN(uart_no)`

3.8. Loop-back

After configuring the register in UART_CONF0, UART TX/RX will cause short circuiting connection internally.

```
#define UART_LOOPBACK  (BIT(14))
```

Loop-back enable bit, 1: enable; 0: disable

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

Interface function: `ENABLE_LOOP_BACK(uart_no)`

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

Interface function: `DISABLE_LOOP_BACK(uart_no)`

3.9. Signal Interruption

By setting UART_TXD_BRK to be 1, signal interruption can be caused on the line. Consequently, when UART TX data on the queue are send out, a break signal will be transmitted (the TX voltage level is low). At that time, data transmission should be stopped and UART_TXD_BRK should be set to be 1.

```
#define UART_TXD_BRK  (BIT(8))
```

Signal interruption enable bit: 1: enable; 0: disable

3.10. Flow Control

Configuration process:

a. First configure pin 12 and pin 13 of UART0, multiplex them to allow U0CTS and U0RTS function.

```
#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. Hardware flow control at the data receiving side can be configured by setting a threshold value at the data receiving terminal. When the length of RX FIFO is longer than the threshold value set before, the voltage level of U0RTS will be pulled high, thus data transmission from the transmission side will be prevented.

Configure the threshold value of flow control at the data receiving side:

Configurations of corresponding threshold values can be set using registers defined by UART_CONF1.

```
#define UART_RX_FLOW_EN  (BIT(23))
```

Data receiving flow control of the 23th bit data: 0: disable; 1: enable

```
#define UART_RX_FLOW_THRHD  0x0000007F
```

Threshold value, occupies 7bit, the value range is between 0 and 127

```
#define UART_RX_FLOW_THRHD_S  16
```

Deviation of the register is 16 (counting from the 16th bit)

c. Flow control of data at the data transmitting side can be realized by configuring the enable function of data flow control only. This register is in UART_CONF0.

```
#define UART_TX_FLOW_EN  (BIT(15))
```

Enable the transmission of flow control: 0: disable; 1: enable

d. Interface

```
Void UART_SetFlowCtrl(uint8 uart_no,UART_HwFlowCtrl flow_ctrl,uint8
rx_thresh);
```

e. Connection of the demo board

Please connect J68(U0CTS) with J63(U0RTS) using jumper wire.

3.11. Miscellaneous

```
TX_FIFO_LEN(uart_no)
```

Macro definition, the current length of transmit queue

```
RF_FIFO_LEN(uart_no)
```

Macro definition, the current length of receive queue

4. Configuration of Interrupt

Before interruptive events are sent to an interrupt controller, OR operation will be executed, therefore, only one interrupt request will be initiated by UART. By checking interrupt status function `UART_INT_ST(uart_no)`, software will be able to process several interruptive events (composed by a couple of "If" sentences) in one interrupt server function simultaneously.

4.1. Interrupt Control Register

There are several interrupt control registers via UART interface:

UART_INT_RAW	Interrupt Raw Status Register.
UART_INT_ENA	Interrupt Enable Register. It is used to indicate that the current enabling function of the register is interrupted.
UART_INT_ST	Interrupt Status Register. It is used to indicate the current effective interrupt status.
UART_INT_CLR	Clear Interrupt Register. Interrupt status of a register will be cleared if status bit is set.

4.2. Interfaces

UART_ENABLE_INTR_MASK(uart_no,ena_mask)	Enable interrupt
UART_DISABLE_INTR_MASK(uart_no,disable_mask)	Disable interrupt
UART_CLR_INTR_STATUS_MASK(uart_no,clr_mask)	Clear interrupt status
UART_GET_INTR_STATUS(uart_no)	Get interrupt status

4.3. Types of Interrupt

4.3.1. RX Full Interrupt

RX Full Interrupt	
Status Bit	`UART_RXFIFO_FULL_INT_ST`
Definition	When threshold value is configured, and interrupt control is enabled, an interrupt will be triggered when the data length of RX FIFO is longer than the threshold value.

RX Full Interrupt	
Application	RX full interrupt is mainly used to process data received via UART, and to achieve flow control by directly processing or posting messages directly, or transfering the messages into buffer. For example, threshold value is 100, full interrupt is enabled, when data received via UART interface reaches 100byte, full interrupt will be triggered.

Config Threshold Value	Config the threshold value of full interrupt function in register UART_CONF1
	`#define UART_RXFIFO_FULL_THRHD 0x0000007F`
	Threshold value is mask, data length is 7bit, data range is between 0 and 127
	`#define UART_RXFIFO_FULL_INT_ENA (BIT(0))`
	Deviation of the register is 0 (counting from 0 bit)

Interrupt Enable	Enable the interrupt using register UART_INT_ENA `#define UART_RXFIFO_FULL_INT_ENA (BIT(0))` Full interrupt enable bit, 1: enable;0: disable
Clear Interrupt Status	All data in RX FIFO must be wiped out before clearing the status of interrupt register, otherwise the interrupt status will remain.

4.3.2. RX Overflow Interrupt

Receive Overflow Interrupt	
Status Bit	`UART_RXFIFO_OVF_INT_ST`
Definition	When Rx FIFO overflow is enabled, FIFO overflow interrupt will be triggered when the data length of RX FIFO queue is longer than the total length of the queue (128bytes).
Application	Overflow occurs usually there is no flow control, because if there is flow control, stack overflow wont happen. The difference between full interrupt and overflow interrupt is that threshold value of full interrupt is manually configured, and the data won't be obliterated, while data obliteration can largely happen in overflow interrupt. Overflow interrupt can be used for debugging.
Interrupt Enable	Config interrupt enable function using register UART_INT_ENA `#define UART_RXFIFO_OVF_INT_ENA (BIT(4))` Status bit of Rx FIFO overflow interrupt: 1: enable; 0: disable
Clear Interrupt Status	Get queue information, make sure that the length of the queue is less than 128 bytes, then set UART_INT_CLR value so as to clear the register status.

4.3.3. RX FIFO TOUT Interrupt

RX FIFO TOUT Interrupt	
Status Bit	`UART_RXFIFO_TOUT_INT_ST`
Definition	When TOUT threshold value is configured, and interrupt control is enabled, a TOUT interrupt will be triggered when the time data transmitting and receiving via UART interface is longer than the set threshold value.
Application	RX FIFO TOUT interrupt is mainly used to process instructions or data transmitted via UART interface, process the data directly or post messages, or transfer the data into buffer.

define UART_RX_TOUT_THRHD_S 24	Config TOUT threshold value using register **UART_CONF1**
	The unit of TOUT threshold value is equal to the time of transmitting 8 byte data via UART, almost a byte.
	`#define UART_RX_TOUT_EN (BIT(31))`
	Timeout enable, 1: enable; 0: disable
	`#define UART_RX_TOUT_THRHD 0x0000007F`
	There are 7 allocations available for configuration of timeout threshold value, ranging from 0 to 127.
	`#define UART_RX_TOUT_THRHD_S 24`
	Deviation of the register is 24 (counting from the 24th bit)

Interrupt Enable	Enable the interrupt using register **UART_INT_ENA** `#define UART_RXFIFO_TOUT_INT_ENA (BIT(8)) tout` TOUT interrupt enable bit, 1: enable;0: disable
Clear Interrupt Status	Similar to FIFO full interrupt, all data in RX FIFO must be wiped out before clearing the status of TOUT interrupt register, otherwise the interrupt status will remain.

4.3.4. TX FIFO Empty Interrupt

TX FIFO Empty Interrupt	
Status Bit	`UART_RXFIFO_TOUT_INT_ST`
Definition	When empty threshold value is configured, and interrupt control is enabled, a FIFO empty interrupt will be triggered when the data in UART TX FIFO is less than the set threshold value.

TX FIFO Empty Interrupt	
Application	TX FIFO empty interrupt can be used to transpond data in buffer to UART. For example, set the empty threshold value as 5, then if the data length of TX FIFO is less than 5 bytes, empty interrupt will be triggered. Consequently, interrupt handler will transmit the data in buffer to TX FIFO until it is filled (the data processing speed of FIFO is faster than the data transmitting speed of TX FIFO). Through such recursion, empty interrupt will close until the transmitting process is completed.

Config Threshold Value	Config empty threshold value using register UART_CONF1
	`#define UART_TXFIFO_EMPTY_THRHD 0x0000007F`
	There are 7 allocations available for configuration of empty threshold value of TX queue, ranging from 0 to 127.
	`#define UART_TXFIFO_EMPTY_THRHD_S 8`
	Deviation of the register is 8 (counting from the 8th bit)

Interrupt Enable	Enable the interrupt using register UART_INT_ENA `#define UART_TXFIFO_EMPTY_INT_ENA (BIT(1))` Empty interrupt enable bit, 1: enable;0: disable
Clear Interrupt Status	FIFO empty interrupt status will be cleared when the data length filled in TX queue is larger than the set threshold values. If there is no more data to fill TX FIFO, then close the interrupt enabler.

4.3.5. Error Detecting Interrupt

Error Detecting Interrupt	
Status Bit	Parity error interrupt: `UART_PARITY_ERR_INT_ST` Line-break error interrupt: `UART_BRK_DET_INT_ST` RX frame error interrupt: `UART_FRM_ERR_INT_ST`
Definition	Parity interrupt occurs when errors exist in parity check of bytes received. Line-break error interrupt occurs when break signal is received, or when the received initial conditions is incorrect (RX line remains low voltage level). RX frame error interrupt occurs when stop bit is not 1.
Application	This type of interrupt is usually used to detect errors.

Error Detecting Interrupt	
Interrupt Enable	Enable the interrupt using register UART_INT_ENA `#define UART_PARITY_ERR_INT_ENA (BIT(2))` Parity error interrupt enable bit, 1: enable;0: disable
	`#define UART_BRK_DET_INT_ENA (BIT(7))` Line-break error interrupt enable bit, 1: enable; 0: disable
	`#define UART_FRM_ERR_INT_ENA (BIT(3))` RX frame error interrupt enable bit: 1: enable; 0: disable
Clear Interrupt Status	Clear the interrupt status bit after the errors have been corrected.

4.3.6. Flow Control Interrupt

Error Detecting Interrupt	
Status Bit	`UART_CTS_CHG_INT_ST` `UART_DSR_CHG_INT_ST`
Definition	Flow control interrupt will be triggered when the power voltage level on pin CTS and/or DSR is changed.
Application	This type of interrupt is usually used for flow control. When triggered, please check the status of corresponding flow control lines, if the voltage level is high, then data writing into TX queue should be stopped.

Interrupt Enable	Enable the interrupt using register UART_INT_ENA
	`#define UART_CTS_CHG_INT_ST (BIT(6))`
	`#define UART_DSR_CHG_INT_ST (BIT(5))`

	`#define UART_CTS_CHG_INT_ENA (BIT(6)) CTS` Line-break error interrupt enable bit, 1: enable; 0: disable
	`#define UART_DSR_CHG_INT_ENA (BIT(5)) DSR` Line-break error interrupt enable bit: 1: enable; 0: disable
Clear Interrupt Status	Clear the interrupt status bit after the errors have been corrected.

5. Demo of Interrupt Handler

```
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. Print Information of Interrupt Mask

By default, UART0 will output some printed information when the device is powered on and is booting up. If this issue exerts influence on some specific applications, users can exchange the inner pins of UART when initializing, that is to say, exchange U0TXD, U0RXD with U0RTS, U0CTS.

Calling interface: `void system_uart_swap(void);`

Calling Interface	`void system_uart_swap(void);`		
Pin definitons of UART0 before Initialization:		Pin definitions of UART0 after exchanging the pins:	
U0TXD	pin26(U0TXD)	U0TXD	pin13(MTDO)
U0RXD	pin25(U0RXD)	U0RXD	pin12(MTCK)
U0CTS	pin12(MTCK)	U0CTS	pin25(U0RXD)
U0RTS	pin13(MTDO)	U0RTS	pin26(U0TXD)

Note ⚠: pin13 and pin12 is the transmit-receive pin of UART0, thus no information will be printed out when the chip is powered on in the initialization stage. However, users should make sure that pin13(MTDO) should NOT be pulled up externally in the initialization stage.