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.

Copyright © 2015 Espressif Systems. All rights reserved.
# Table of Contents

1. **Preambles** .................................................................................................................. 4  
   1.1. UART TX FIFO Working Process ........................................................................... 4  
   1.2. UART RX FIFO Working Process ........................................................................... 4  
   1.3. Application Context ................................................................................................. 4  
2. **Hardware Resources** .................................................................................................... 6  
3. **Parameters Configuration** ............................................................................................ 6  
   3.1. Baud Rate .................................................................................................................. 6  
   3.2. Parity ......................................................................................................................... 6  
   3.3. Number Bit ................................................................................................................ 6  
   3.4. Stop Bit ...................................................................................................................... 7  
   3.5. Reverse Operation .................................................................................................... 7  
   3.6. Switch the Output Terminal .................................................................................... 7  
   3.7. Read the Left Number Length of TX/RX Queue .................................................... 7  
   3.8. Loop-back ............................................................................................................... 8  
   3.9. Signal Interruption .................................................................................................... 8  
   3.10. Flow Control ........................................................................................................... 8  
   3.11. Miscellaneous ........................................................................................................ 9  
4. **Configuration of Interrupt** .......................................................................................... 10  
   4.1. Interrupt Control Register ....................................................................................... 10  
   4.2. Interfaces .................................................................................................................. 10  
   4.3. Types of Interrupt .................................................................................................... 10  
   4.3.1. RX Full Interrupt ................................................................................................. 10  
   4.3.2. RX Overflow Interrupt ....................................................................................... 11  
   4.3.3. RX FIFO TOUT Interrupt .................................................................................... 12  
   4.3.4. TX FIFO Empty Interrupt ................................................................................... 12  
   4.3.5. Error Detecting Interrupt ................................................................................... 13  
   4.3.6. Flow Control Interrupt ....................................................................................... 14  
5. **Demo of Interrupt Handler** ....................................................................................... 15  
6. **Print Information of Interrupt Mask** ......................................................................... 16
1. **Preambles**

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

<table>
<thead>
<tr>
<th>UART0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>U0TXD</td>
<td>pin26(U0TXD)</td>
</tr>
<tr>
<td>U0RXD</td>
<td>pin25(U0RXD)</td>
</tr>
<tr>
<td>U0CTS</td>
<td>pin12(MTCK)</td>
</tr>
<tr>
<td>U0RTS</td>
<td>pin13(MTDO)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UART1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>U1TXD</td>
<td>pin14(GPIO2)</td>
</tr>
</tbody>
</table>

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 by 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.
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**

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

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

```c
#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**

```c
#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

```c
#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 \( \text{UART\_INT\_ST}(\text{uart\_no}) \), 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:

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART_INT_RAW</td>
<td>Interrupt Raw Status Register.</td>
</tr>
<tr>
<td>UART_INT_ENA</td>
<td>Interrupt Enable Register. It is used to indicate that the current enabling function of the register is interrupted.</td>
</tr>
<tr>
<td>UART_INT_ST</td>
<td>Interrupt Status Register. It is used to indicate the current effective interrupt status.</td>
</tr>
<tr>
<td>UART_INT_CLR</td>
<td>Clear Interrupt Register. Interrupt status of a register will be cleared if status bit is set.</td>
</tr>
</tbody>
</table>

4.2. Interfaces

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{UART_ENABLE_INTR_MASK}(\text{uart_no},\text{ena_mask}) )</td>
<td>Enable interrupt</td>
</tr>
<tr>
<td>( \text{UART_DISABLE_INTR_MASK}(\text{uart_no},\text{disable_mask}) )</td>
<td>Disable interrupt</td>
</tr>
<tr>
<td>( \text{UART_CLR_INTR_STATUS_MASK}(\text{uart_no},\text{clr_mask}) )</td>
<td>Clear interrupt status</td>
</tr>
<tr>
<td>( \text{UART_GET_INTR_STATUS}(\text{uart_no}) )</td>
<td>Get interrupt status</td>
</tr>
</tbody>
</table>

4.3. Types of Interrupt

4.3.1. RX Full Interrupt

<table>
<thead>
<tr>
<th>RX Full Interrupt</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status Bit</td>
<td>( \text{UART_RXFIFO_FULL_INT_ST} )</td>
</tr>
<tr>
<td>Definition</td>
<td>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.</td>
</tr>
</tbody>
</table>
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 transferring the messages into buffer. For example, threshold value is 100, full interrupt is enabled, when data received via UART interface reaches 100 byte, 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 7 bit, 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.

RX Full 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 (128 bytes).

Application
Overflow occurs usually there is no flow control, because if there is flow control, stack overflow won’t 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

<table>
<thead>
<tr>
<th>Status Bit</th>
<th>UART_RXFIFO_TOUT_INT_ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>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.</td>
</tr>
<tr>
<td>Application</td>
<td>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.</td>
</tr>
<tr>
<td>define UART_RX_TOUT_THRHD_S 24</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>#define UART_RX_TOUT_EN (BIT(31))</td>
</tr>
<tr>
<td></td>
<td>Timeout enable, 1: enable; 0: disable</td>
</tr>
<tr>
<td></td>
<td>#define UART_RX_TOUT_THRHD 0x0000007F</td>
</tr>
<tr>
<td></td>
<td>There are 7 allocations available for configuration of timeout threshold value, ranging from 0 to 127.</td>
</tr>
<tr>
<td></td>
<td>#define UART_RX_TOUT_THRHD_S 24</td>
</tr>
<tr>
<td></td>
<td>Deviation of the register is 24 (counting from the 24th bit)</td>
</tr>
<tr>
<td>Interrupt Enable</td>
<td>Enable the interrupt using register UART_INT_ENA.</td>
</tr>
<tr>
<td></td>
<td>#define UART_RXFIFO_TOUT_INT_ENA (BIT(8)) tout</td>
</tr>
<tr>
<td></td>
<td>TOUT interrupt enable bit, 1: enable; 0: disable</td>
</tr>
<tr>
<td>Clear Interrupt Status</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 4.3.4. TX FIFO Empty Interrupt

<table>
<thead>
<tr>
<th>Status Bit</th>
<th>UART_RXFIFO_TOUT_INT_ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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

| Status Bit         | UART_CTS_CHG_INT_ST  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Flow control interrupt will be triggered when the power voltage level on pin CTS and/or DSR is changed.</td>
</tr>
<tr>
<td>Application</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### Interrupt Enable

Enable the interrupt using register UART_INT_ENA

```c
#define UART_PARITY_ERR_INT_ENA (BIT(2))
```

Parity error interrupt enable bit, 1: enable; 0: disable

```c
#define UART_BRK_DET_INT_ENA  (BIT(7))
```

Line-break error interrupt enable bit, 1: enable; 0: disable

```c
#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  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Flow control interrupt will be triggered when the power voltage level on pin CTS and/or DSR is changed.</td>
</tr>
<tr>
<td>Application</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### Interrupt Enable

Enable the interrupt using register UART_INT_ENA

```c
#define UART_CTS_CHG_INT_ST  
#define UART_DSR_CHG_INT_ST
```

```c
#define UART_CTS_CHG_INT_ENA  (BIT(6))  CTS
```

Line-break error interrupt enable bit, 1: enable; 0: disable

```c
#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

```c
Demo of Interrupt 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);`

<table>
<thead>
<tr>
<th>Calling Interface</th>
<th>void system_uart_swap(void);</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin definitions of UART0 before Initialization:</td>
<td>Pin definitions of UART0 after exchanging the pins:</td>
</tr>
<tr>
<td>U0TXD</td>
<td>pin26(U0TXD)</td>
</tr>
<tr>
<td>U0RXD</td>
<td>pin25(U0RXD)</td>
</tr>
<tr>
<td>U0CTS</td>
<td>pin12(MTCK)</td>
</tr>
<tr>
<td>U0RTS</td>
<td>pin13(MTDO)</td>
</tr>
</tbody>
</table>

**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.