ESP8266 WiFi extension Design

Virgile Neu

May 10, 2017
version 1.0

Contents

1 Introduction

The ESP8266 chip is a WiFi module with a AT command mode. It can be used for single connection or multiple connections (server). It has 4 pins of interest plus VCC and GND : Chip Enable, RESET, Rx and Tx. The picture 1 below show the esp8266 board with it's connectivity.

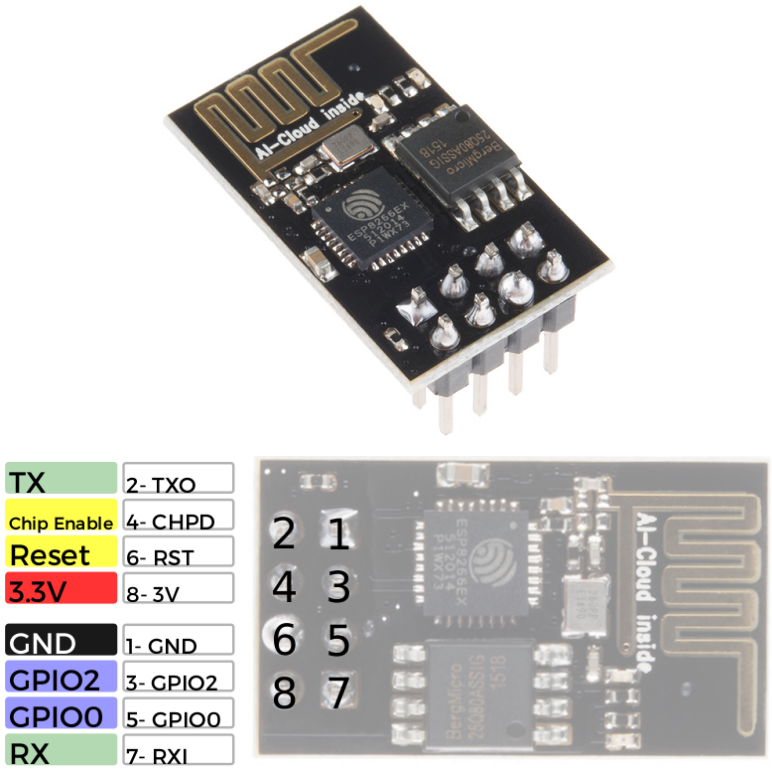

Figure 1: The ESP8266 module

The goal is to make available this WiFi module to use on the FPGA and to make it easily usable. Figure 2 depicts how to use it from the CPU point of view.

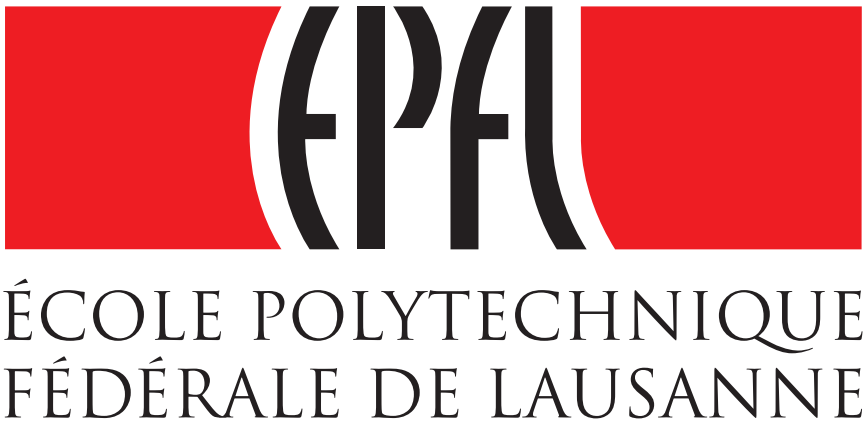

Figure 2: Uses flowchart seen by the CPU.

2 Parameters

2.1 Default configuration

2.2 Serial Parameters

The ESP8266 WiFi module uses UART communication to transmit information with the FPGA.
The UART works as described on figure 3. It starts with the start bit, always '0', then comes the data (here 8 bits), least significant bit first, then the parity bit (if set), and then 1 or 2 stop bit, always '1'.
The ESP8266 supports a lot of different settings :

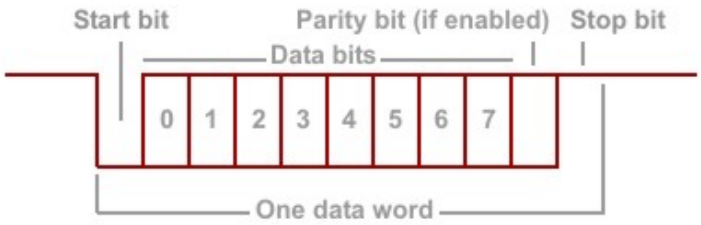

Figure 3: UART data transfert.

2.2.1 Data bits

- 5 bits
- 6 bits
- 7 bits
- 8 bits

2.2.2 Baud rates

Unlike the HC05, it supports a continuous range of baud rates, between 110 to 115200*40 bits/s.

2.2.3 Stop bit

- 1 bit
- 1.5 bit

- 2 bit

2.2.4 Parity bit

- None
- Odd parity
- Event parity

2.2.5 Flow control

- No flow control
- enable Request To Send
- enable Clear To Send
- enable both RTS and CTS

2.3 WiFi Parameters

2.3.1 Encryption

The ESP8266 supports several kind of encryption :

- No encryption
- WEP
- WPA_PSK
- WPA2_PSK
- WPA_WPA2_PSK
- WPA2_Entreprise

2.3.2 Connection mode

The ESP8266 has two WiFi connection modes :

- COMPLETE THIS

3 Design Choices

Here I will show how the extension will look like, see figure 4. It will consist of Four parts:

- Registers, to store configuration, status and other things,
- A FIFO_OUT to send data from the CPU to the UART custom interface,
- A FIFO_IN to receive data from the ESP8266,
- A custom UART interface to communicate to the ESP8266.

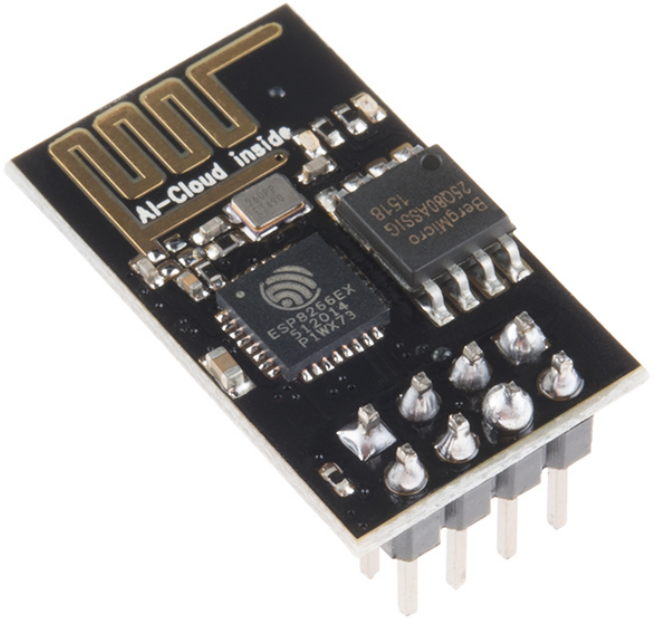

Figure 4: High level block diagram of the ESP8266 extension.

3.1 Registers

The registers will have height registers :

- A control register CTRL,
- A status register STATUS,

- A register for the UART waiting cycles (depends on the UART rate),
- The FIFO_out_data register,
- The FIFO_out_free_space register,
- The FIFO_in_data register,
- The FIFO_in_pending_data register.
- The reset_FIFO_in register.

Here is the register map in table 1 below.

Table 1: Register map of the Registers component.

#	addr	31..8	7	6	5	4	3	2	1	0	R/W
0	0x00	Unused			UART_CTRL			I_ENABLE		UART_ON	R/W
1	0x04	Unused							i_pending		R/W
2	0x08	UART_wait_cycles									R/W
3	0x0C	ignored	FIFO_out_data								W
4	0x10	FIFO_out_free_space									R
5	0x14	zeros	FIFO_in_data								R
6	0x18	FIFO_in_pending_data									R
7	0x1C	Unused							Reset_FIFO_out	Reset_FIFO_in	W

UART_CTRL		
5	4	3
Parity_bit		Stop_bit

I_ENABLE	
2	1
i_dropped	i_received

The role of each bit is described below :

- 0x00 :
  - UART_ON : Specifies if the UART will capture or send data or if it will stay off.
  - i_reveived : Specifies if the device can send interrupts request when receiving data from the ESP8266.
  - i_dropped : Specifies if the device can send interrupts request when some data is dropped.
  - stop_bit : Specifies the number of stop bit, '0' for 1, '1' for 2.
  - parity_bit : Specifies the parity bit, "00" for None, "10" for Even and "11" for Odd.

- 0x04 :
  - i_pending : Tells if there is an interrupt waiting to be served by the CPU. The CPU must clear it by software when serving the interrupt. Bit 0 is for i_received, bit 1 is for i_dropped. Writing '1' to any of the two bits has no effect.
- 0x08 : UART_wait_cycles : Specifies to the UART how many cycles it should wait before capturing the values during the transfert. The values to put are described in the table 2 below for a 50MHz clock.
- 0x0C : FIFO_out_data : Address to write to send data to the ESP8266 through the FIFO_out. The write must has the byte_enable signal equal to "0001".
- 0x10 : FIFO_out_free_space : Number of free words (8 bits) in the FIFO_out.
- 0x14 : FIFO_in_data : Address to read to receive data from the ESP8266 through the FIFO_in.
- 0x18 : FIFO_out_free_space : Number of waiting words (8 bits) in the FIFO_in.
- 0x1C :
  - Reset_FIFO_in : Write only bit to clear the FIFO_in.
  - Reset_FIFO_out : Write only bit to clear the FIFO_out.

The value to put in the UART_wait_cycles registers depend on the desired UART baud rate, and is computed with the following formula.

$$\begin{aligned} wait\_cycles &= \frac{time\_per\_bit}{time\_per\_cycles} \\ &= \frac{\frac{1}{baud\_rate}}{clk\_period} \\ &= \frac{clk\_freq}{baud\_rate} \end{aligned}$$

For 4800 bits/s of baud rate we have.

$$\begin{aligned} wait\_cycles &= \frac{clk\_freq}{baud\_rate} \\ &= \frac{50 \cdot 10^6}{4800} = 10416.667 \quad clk\_cycles \end{aligned}$$

The rounding doesn't matter.

Table 2: UART_wait_cycles values for a given UART.

UART_Rate	wait_cycles value (decimal)
4800 bits/s	10416 clk_cycles
9600 bits/s	5207 clk_cycles
19200 bits/s	2604 clk_cycles
38400 bits/s	1302 clk_cycles
57600 bits/s	868 clk_cycles
115200 bits/s	434 clk_cycles
230400 bits/s	217 clk_cycles
460800 bits/s	109 clk_cycles
921600 bits/s	54 clk_cycles
1382400 bits/s	36 clk_cycles

When using autoconnect, the device will simply boot with the last configuration used. When using AT command mode, the CPU will have to initiate itself the connection, and can change modes or settings.

The ports of the Registers component are described on figure 5 below.

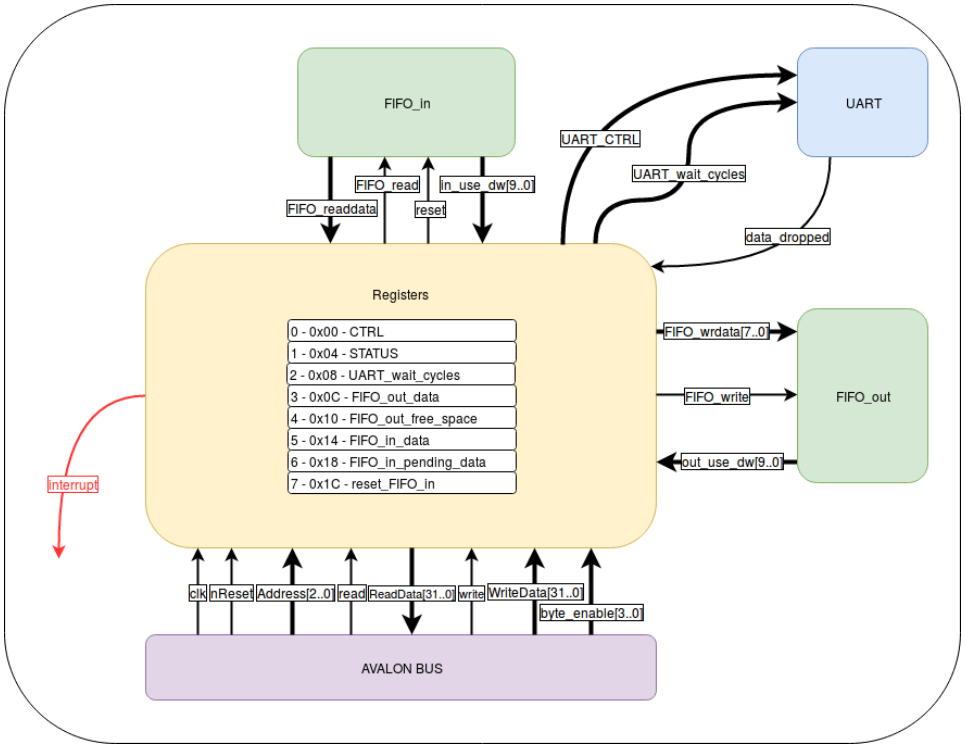

Figure 5: Ports description of the Registers component.

3.2 FIFO_out

For the FIFO_out we will use the FIFO available in the IP catalogue of Quartus with the following configurations :

- Width = 8 bits,
- Depth = 1024 (biggest size with only one M10k element),
- control signals :
  - use_dw[] (10 bits),
  - empty,
  - asynchronous clear;
- Normal synchronous FIFO mode,
- Auto memory block type,

- No optimisation or circuitry protection.

The ports of the FIFO_out component are described on figure 6.

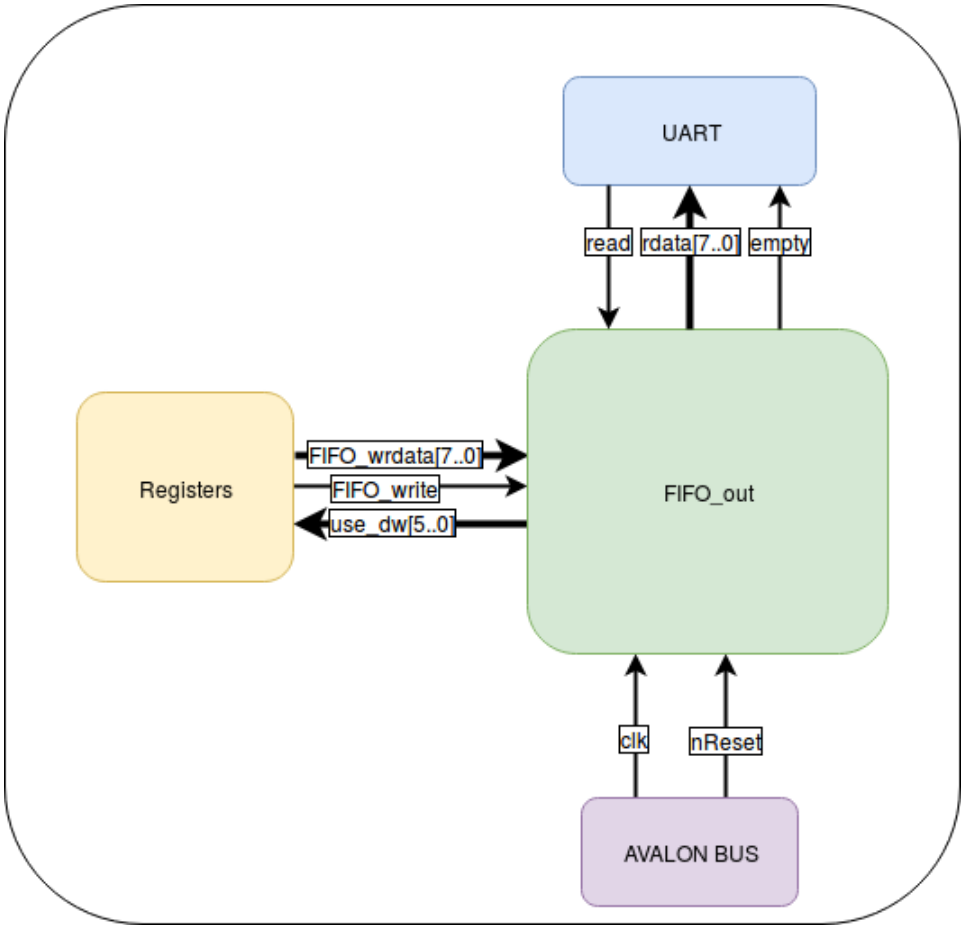

Figure 6: Ports description of the FIFO_out component.

3.3 FIFO_in

For the FIFO_in we will also use the FIFO available in the IP catalogue of Quartus with almost the same configurations :

- Width = 8 bits,
- Depth = 1024 (biggest size with only one M10k element),
- control signals :

  - use_dw[] (10 bits),
  - full,
  - asynchronous clear;
- Normal synchronous FIFO mode,
- Auto memory block type,
- No optimisation or circuitry protection.

The ports of the FIFO_in component are described on figure 7.

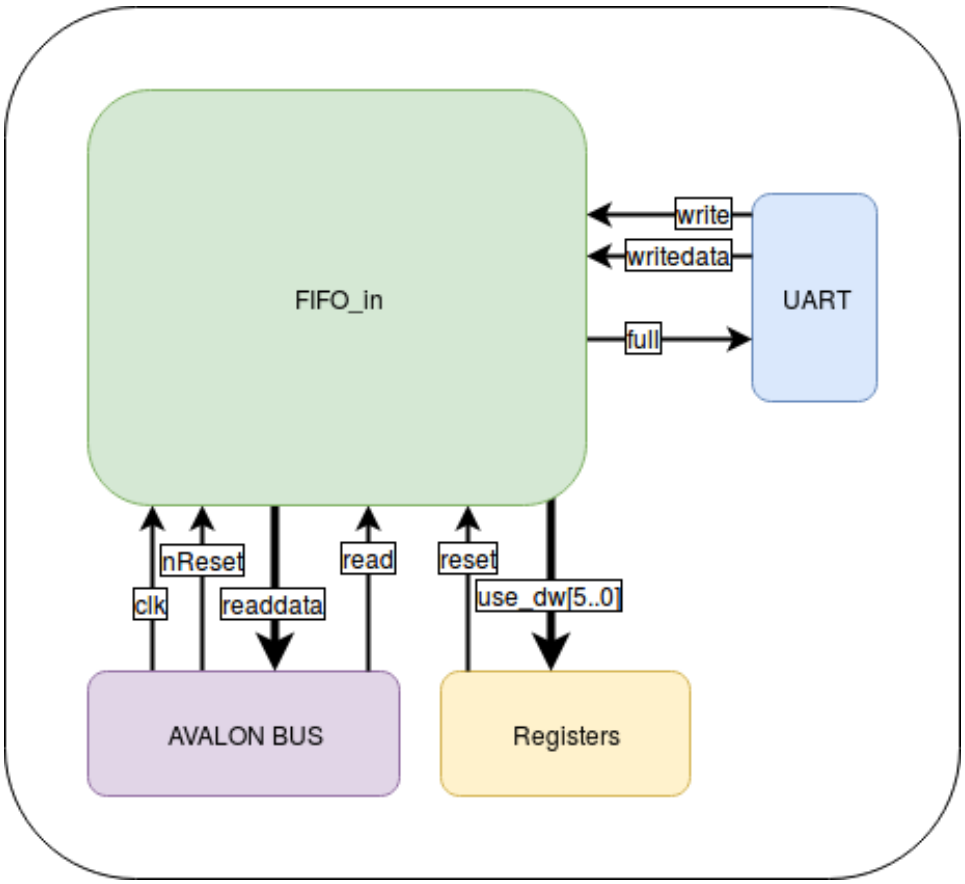

Figure 7: Ports description of the FIFO_in component.

3.4 UART

The UART will be the part communicating with the ESP8266 module. It will send whenever it can while the FIFO_out isn't empty, and whenever it receives information, it will recompose the words, perform the parity check (if set) and send the correct words to the FIFO_in.
The ports of the UART component are described on figure 8.

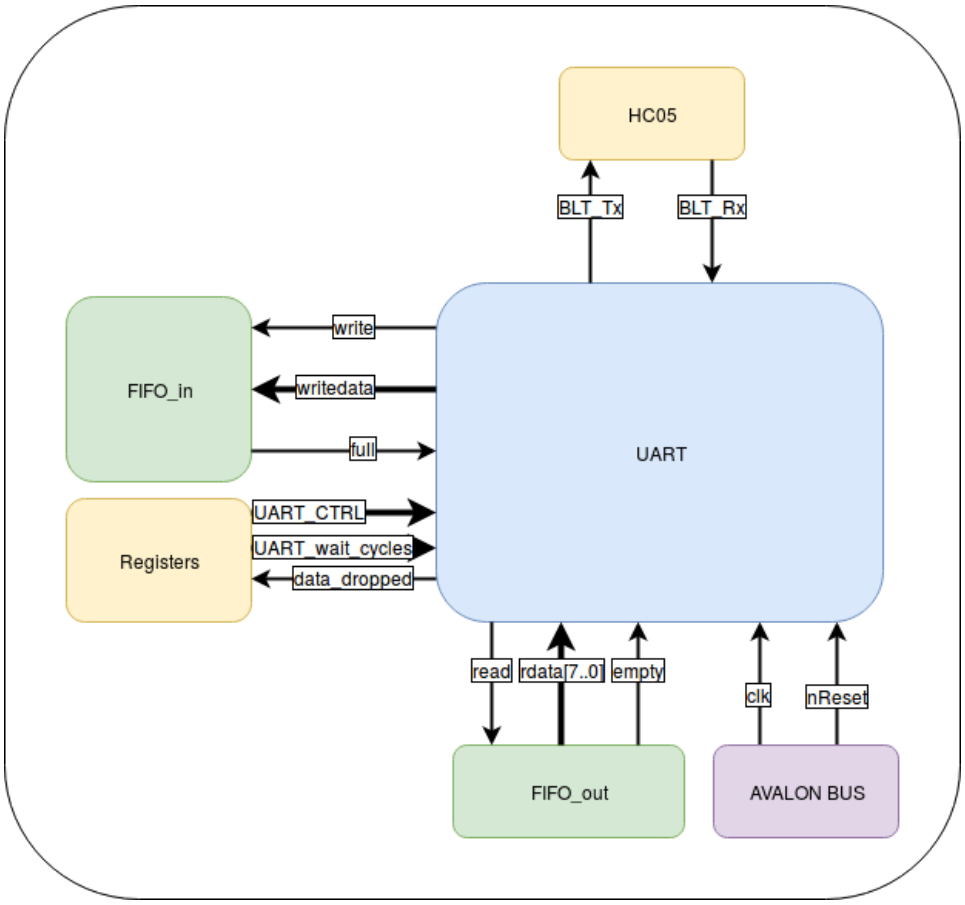

Figure 8: Ports description of the UART component.

4 Pinout

The external connectivity of the device is described on table 3.

Table 3: Pinout table of the device.

signal name	connectivity
BLT_RxD	GPIO_1 8 – FPGA PIN_AE22
BLT_TxD	GPIO_1 6 – FPGA PIN_AH24
BLT_State	PCA9673 via Avalon Bus
BLT_EN	
BLT_ATSel	

5 States Machines

This section describes the several states machines used in the extension.

5.1 UART

5.1.1 Transmitting State Machine

The figure 9 below describe the state machine used for transmitting data. It consists of 5 states : WAITING, START, SENDING, PARITY and STOP states. It starts at the WAITING states, and wait for data to be available in the FIFO_out. Once data is available, it issue a read to the FIFO_out and go to the start states. During the start state, it outputs the '0' value, as specified in the UART protocol, and store the data from the FIFO_out_readdata during the first cycle in this state. Once it has waited enough, it goes to sending. During sending state, it will send bit after bit, every time waiting the good amount of time. Oncei all the 8 bit of data are sent, it will either go to STOP if the parity is disabled (Parity_bit = "00") or to PARITY if it is enable. In the PARITY state, it will output the parity value (odd or even) for the right amount of time, and then go to the STOP state. In the STOP state, it will output 1 or 2 bit at '1', depending on the settings of the Stop_bit, and then go to the WAITING state, ready to transfer again.

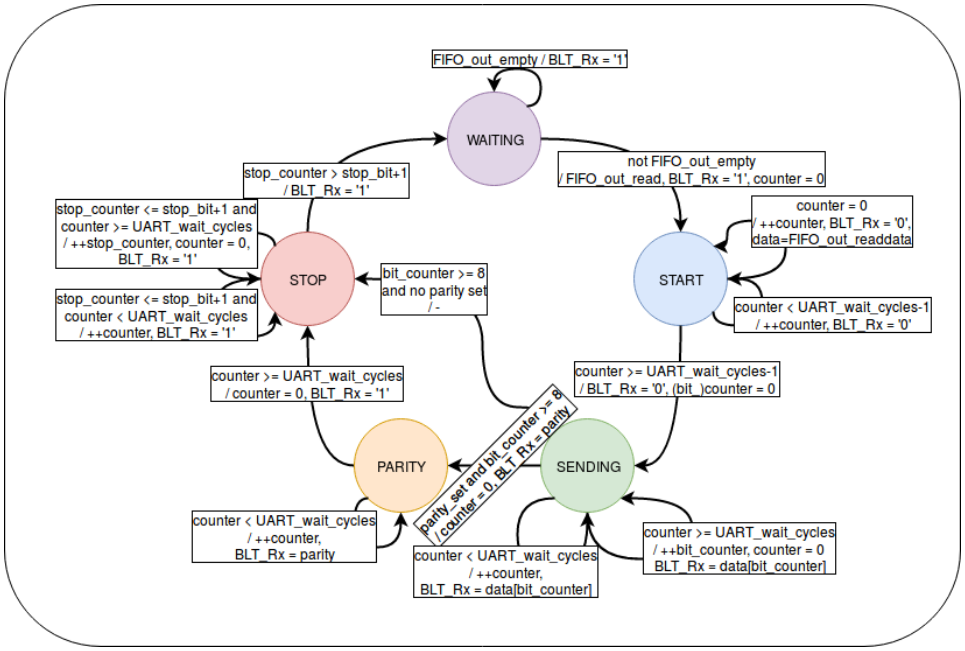

Figure 9: State machine used for sending one word (8 bits) to the ESP8266.

5.1.2 Receiving State Machine

The figure 10 below describe the state machine used for receiving data. It has 4 states : WAITING, START, RECEIVING and PARITY. It starts at the WAITING states, and wait until the BLT_Tx is '0' (start bit). Then we wait for half the cycles to wait in the START state in order to capture each bit in correctly and not just when they are supposed to go up (in order to avoid wrong bits), continuously checking that the start bit is still on (BLT_Tx = '0'). Then we go to the RECEIVING state, where we wait for a full wait before capturing each bit. There is a transition back to the WAITING state with a big condition, it is to catch an error in the start bit during the first half of the first wait round. Once we received all the bits, we either go to the parity check in the PARITY state if enable or directly to the WAITING state and writing the data to the FIFO_in if it is not full. If we go to the PARITY state, we check if the parity of the data we received is correct, and if it is we write it to the FIFO_in if it is not full, and else we discard it. Then we go back to WAITING.

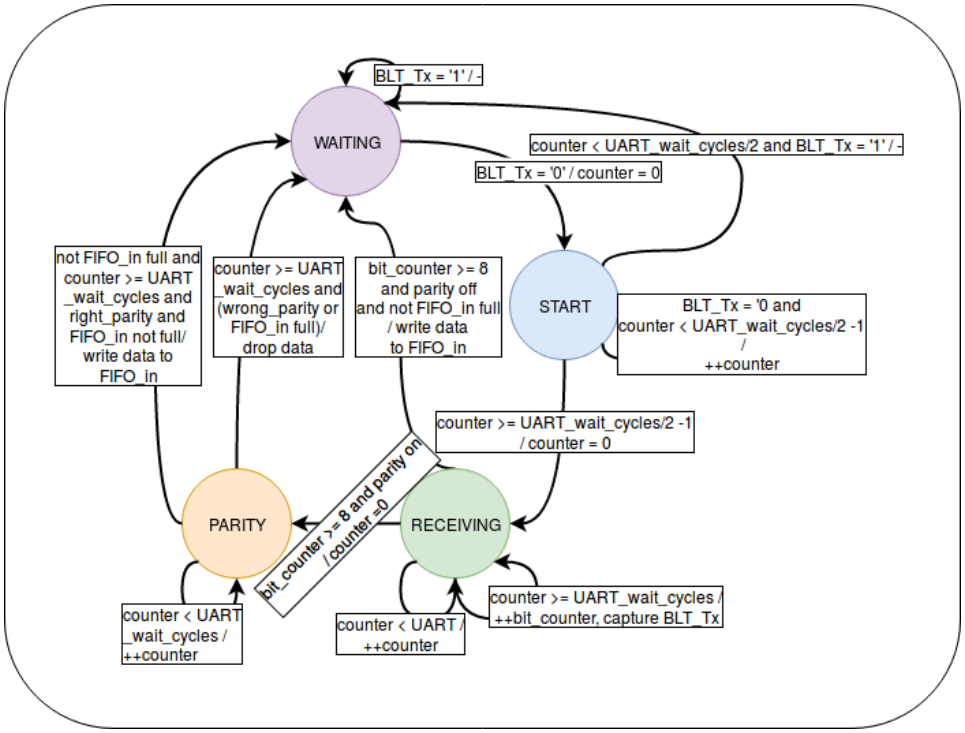

Figure 10: State machine used for receiving one word (8 bits) from the ESP8266.

6 Power consumption

The ESP8266 device has a different power consumption depending on its mode and if it is transmitting or receiving data. The values have been measured with a constant 5V input for all the baud rates and it appears that it has no effect on the power consumption. The results can be found on the table 4 below.

Table 4: Power consumption of the ESP8266 device

Mode	I(mA)	P(mW)	Notes
Not enable	5	20	
AT command mode	15	65	Receiving or not has no impact
Slave mode not connected	40	200	
Slave mode connected	20	100	Spikes at 60mA/300mW every 130ms
Receiving data	60-80	300-400	Not constant, spikes every 0.5ms
Transmitting data	60-80	300-400	

7 WiFi transfer rate

The data transfer rate between the ESP8266 and another WiFi device have been measured. The baud rate has a significant impact on it, especially at low rates. The results are presented below on table 5 and figure 11.
The data throughput is almost constant when higher than 57600 b/s of baud rate, and the only thing changing is the packet size when above 460800 b/s.

Table 5: Throughput between the ESP8266 and another WiFi device

Baud rate	Useful payload(Byte)	Time between packets(ms)	Throughput(B/s)
4800	5	1.246	4012.84
9600	10	1.251	7993.61
19200	20	1.251	15987.21
38400	49	1.157	42350.91
57600	54	1.132	47703.18
115200	54	1.141	47326.91
230400	54	1.031	52376.3
460800	127	2.507	50658.16
921600	127	2.501	50779.69
1382400	127	2.694	47141.8

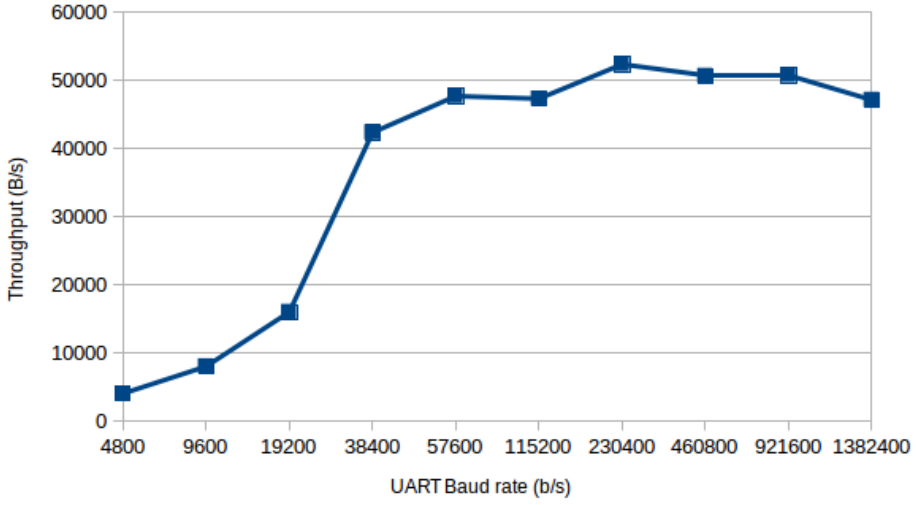

Figure 11: Throughput of the ESP8266 depending on the Baud rates.

8 WiFi protocol

The ESP8266 uses the L2CAP WiFi protocol to transmit data over a WiFi connection. This protocol supports segmentation and reassembly of packets, with a max packet payload of 64 kB. It also supports flow control and retransmission of packets. In theory, this protocol could also be used to do group-oriented communication, with different communication channels possible, but it is not used by the ESP8266.

ESP8266 AT Command Set

Function	*AT Command*	*Response*
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1) +CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1) OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): + IPD, <len>: (CIPMUX=1): + IPD, <id>, <len>: <data>
Watchdog Enable*	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable
Watchdog Disable*	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

* New in V0.9.2.2 (from http://www.electrodragon.com/w/Wi07c)