

# **WiFiMCU Lua Reference Book**

DoIT / LoBo

2015-11-22

# Content

Lua Basic Modules.....	6
MCU Module.....	6
Function List.....	6
Constant.....	6
mcu.ver().....	7
mcu.info().....	7
mcu.reboot().....	7
mcu.mem().....	8
mcu.chipid().....	8
mcu.bootreason().....	8
GPIO Module.....	10
Function List.....	10
Constant.....	10
GPIO Pin Table.....	11
gpio.mode().....	11
gpio.read().....	12
gpio.write().....	12
gpio.toggle().....	13
TIMER Module.....	14
Function List.....	14
Constant.....	14
tmr.start().....	14
tmr.stop().....	15
tmr.stopall().....	15
tmr.tick().....	15
tmr.delayms().....	16
tmr.delayus().....	16
tmr.wdclr().....	16
WiFi Module.....	17
Function list.....	17

Constant.....	17
wifi.startap().....	17
wifi.startsta().....	18
wifi.scan().....	19
wifi.stop().....	19
wifi.powersave().....	19
wifi.ap.getip().....	20
wifi.ap.getipadv().....	20
wifi.ap.stop().....	21
wifi.sta.getip().....	21
wifi.sta.getipadv().....	21
wifi.sta.getlink().....	22
wifi.sta.stop().....	22
wifi.sta.ntptime().....	23
Net Module.....	24
Function list.....	24
Constant.....	24
net.new().....	24
net.start().....	25
net.on().....	25
net.send().....	26
net.close().....	26
net.getip().....	27
File Module.....	28
Function list.....	28
Constant.....	28
file.format().....	29
file.open().....	29
file.close().....	30
file.write().....	30
file.writeline().....	30
file.read().....	31

file.readline().....	31
file.list().....	32
file.slist().....	32
file.remove().....	32
file.seek().....	33
file.flush().....	33
file.rename().....	34
file.info().....	34
file.state().....	34
file.compile().....	35
dofile().....	35
PWM Module.....	37
Function list.....	37
Constant.....	37
Pin Table.....	37
pwm.start().....	37
pwm.stop().....	38
ADC Module.....	39
Function list.....	39
Constant.....	39
Pin Table.....	39
adc.read().....	39
adc.readmv().....	40
UART Module.....	41
Function list.....	41
Constant.....	41
uart.setup().....	41
uart.on().....	42
uart.send().....	42
Bit Module.....	43
Function List.....	43
Constant.....	43

bit.bnot()	43
bit.band()	44
bit.bor()	44
bit.bxor()	44
bit.lshift()	45
bit.rshift()	45
bit.arshift()	45
bit.bit()	46
bit.set()	46
bit.clear()	47
bit.isset()	47
bit.isclear()	47
Sensor Module	49
Function List	49
Constant	49
sensor.dht11.init()	49
sensor.dht11.get()	50
sensor.ds18b20.init()	50
sensor.ds18b20.search()	50
sensor.ds18b20.gettemp()	51
sensor.ds18b20.setres()	51
sensor.ds18b20.getres()	51
sensor.ds18b20.getrom()	52
sensor.ow.init()	52
sensor.ow.search()	52
SPI Module	54
Function List	54
Constant	54
spi.setup()	54
spi.write()	55
spi.read()	55
spi.deinit()	56

I2C Module.....	57
Function List.....	57
Constant.....	57
i2c.setup().....	57
i2c.start().....	58
i2c.stop().....	58
i2c.address().....	58
i2c.write().....	59
i2c.read().....	59
RTC Module.....	60
Function List.....	60
rtc.getasc().....	60
rtc.getstrf().....	60
rtc.get().....	62
rtc.set().....	62
rtc.standby().....	63
rtc.standbyUntil().....	63
OLED Module.....	65
Function List.....	65
oled.init().....	65
oled.clear().....	66
oled.write().....	66
oled.writechar().....	66
oled.fontsize().....	67
oled.charspace().....	67
oled.inverse().....	67
oled.fixedwidth().....	68

## Lua Basic Modules

The Lua interpreter in WiFiMCU is based on Lua 5.1.4.

The following modules are supported:

luaopen_base	Supported
luaopen_package	Supported
luaopen_string	Supported
luaopen_table	Supported
luaopen_math	Supported

‘io’ and ‘debug’ modules are not supported.

The functions description in supported modules can be found at: <http://www.lua.org/manual/5.1/>

## MCU Module

### Function List

mcu.ver()	Get the WiFiMCU firmware version
mcu.info()	Get the mxchipWNet library version, MAC address, WLAN driver version
mcu.reboot()	Reboot WiFiMCU
mcu.mem()	Get the memory status
mcu.chipid()	Get the stm32 chip ID (96 bits)
mcu.bootreason()	Get the WiFiMCU boot reason that caused its startup

### Constant

nil

## mcu.ver()

### Description

Get the WiFiMCU firmware version.

### Syntax

```
nv,bd=mcu.ver()
```

### Parameters

nil

### Returns

nv: string type, WiFiMCU firmware version

bd: string type, build date of the firmware

### Examples

```
>nv,bd=mcu.ver()
```

```
>print(nv,bd)
```

```
>WiFiMCU 0.9.7 build 20151122
```

## mcu.info()

### Description

Get the mxchipWNet library version, MAC address, WLAN driver version.

**Syntax** libv,mac,drv=mcu.info()

**Parameters** nil

### Returns

libv: mxchipWNet library version mac:

MAC address of the module drv: WLAN

driver version

### Examples

```
>libv,mac,drv=mcu.info()
```

```
>print(libv,mac,drv)
```

```
>31620002.031 C8:93:46:50:21:4C wl0: Dec 29 2014 14:07:06 version 5.90.230.10  
FWID 01-9bdaad4d
```

## mcu.reboot()

### Description

Reboot WiFiMCU immediately.

### Syntax

```
mcu.reboot()
```



**Parameters**

nil

**Returns**

nil

**Examples**

```
> mcu.reboot()
```

## mcu.mem()

**Description**

Get the memory status.

**Syntax** fm,tas,mtas,fc=mcu.mem()

**Parameters** nil

**Returns**

fm: Total free space

tas: Total allocated space

mtas: Maximum total allocated space fc: Number of free chunks

**Examples**

```
> fm,tas,mtas,fc=mcu.mem()
```

```
> print(fm,tas,mtas,fc)
```

```
> 35600 50416 86016 25
```

## mcu.chipid()

**Description**

Get the stm32 chip ID (96 bits).

**Syntax**

```
chipid= mcu.chipid()
```

**Parameters**

nil

**Returns**

chipid: the stm32 chip product ID

**Examples**

```
> chipid= mcu.chipid()
```

```
> print(chipid)
```

```
0200C000FDFFFAE005DFF000
```

## mcu.bootreason()

## Description

Get the WiFiMCU boot reason that cause its startup.

## Syntax

```
bootreason= mcu.bootreason()
```

## Parameters

nil

## Returns

bootreason: The boot reason should be one the followings:

"NONE":	Fail to get the boot reason
"SOFT_RST":	Software reset
"PWRON_RST":	Power on reset
"EXPIN_RST":	Pin reset
"WDG_RST":	Independent Watchdog reset
"WWDG_RST":	Window Watchdog reset
"LOWPWR_RST":	Low Power reset
"BOR_RST":	POR/PDR or BOR reset

## Examples

```
> mcu.bootreason() SOFT_RST
```

## GPIO Module

### Function List

gpio.mode()	Define the GPIO Pin mode, set the pin to input output or interrupt mode
gpio.read()	Read the pin value
gpio.write()	Set the pin value
gpio.toggle()	Toggle the pin's output value

### Constant

gpio.INPUT	Input with an internal pull-up resistor
gpio.INPUT_PULL_UP	Input with an internal pull-up resistor
gpio.INPUT_PULL_DOWN	Input with an internal pull-down resistor
gpio.INPUT_INPUT_HIGH_IMPEDANCE_DOWN	Input high impedance down
gpio.OUTPUT	Output actively driven high and actively driven low
gpio.OUTPUT_PUSH_PULL	Output actively driven high and actively driven low
gpio.OUTPUT_OPEN_DRAIN_NO_PULL	Output actively driven low but is high-impedance when set high
gpio.OUTPUT_OPEN_DRAIN_PULL_UP	Output actively driven low and is pulled high with an internal resistor when set high
gpio.INT	Interrupt
gpio.HIGH	High voltage level
gpio.LOW	Low voltage level

## GPIO Pin Table

WiFiMCU Index	Alternate function	Discription
D0	GPIO/BOOT	WiFiMCU would enter into Bootloader Mode, if D0 goes to LOW
D1	GPIO/PWM/ADC	
D2	GPIO	
D3	GPIO/PWM	
D4	GPIO	
D5	GPIO	SWD Flash Programming Pin: swclk
D6	GPIO	SWD Flash Programming Pin: swdio
D7	GPIO/SPI5_MISO	
D8	GPIO/PWM/SPI5_MOSI	Uart1 rx pin: RX1
D9	GPIO/PWM	Uart1 tx pin: TX1
D10	GPIO/PWM	I2C interface: SCL
D11	GPIO/PWM	I2C interface: SDA
D12	GPIO/PWM	
D13	GPIO/PWM/ADC	
D14	GPIO/PWM	
D15	GPIO/PWM/ADC	
D16	GPIO/PWM/ADC/SPI5_CLK	
D17	GPIO/ADC	BLUE LED on WiFiMCU board

## gpio.mode()

### Description

Define the GPIO Pin mode, set the pin to input output or interrupt mode.

### Syntax

```
gpio.mode(pin, mode)
gpio.mode(pin, gpio.INT, trigMode, func_cb)
```

### Parameters

pin: gpio ID, 0~17  
mode: Should be one of the followings: gpio.INPUT  
gpio.INPUT\_PULL\_UP gpio.INPUT\_PULL\_DOWN  
gpio.INPUT\_INPUT\_HIGH IMPEDANCE\_DOWN gpio.OUTPUT  
gpio.OUTPUT\_PUSH\_PULL gpio.OUTPUT\_OPEN\_DRAIN\_NO\_PULL  
gpio.OUTPUT\_OPEN\_DRAIN\_PULL\_UP gpio.INT  
trigMode: if mode is gpio.INT, trigMode should be:  
'rising': Interrupt triggered at input signal's rising edge  
'falling': Interrupt triggered at input signal's falling edge  
'both': Interrupt triggered at both rising and falling edge

func\_cb: if mode is gpio.INT, the interrupt call back function

**Note:** It's recommend that you DO NOT do too much time consuming operations in the func\_cb.

**Returns** nil

### Examples

```
>gpio.mode(0, gpio.OUTPUT)
>gpio.write(0, gpio.HIGH)
>gpio.mode(1,gpio.INPUT)
>print(gpio.read(1))
>0
```

## gpio.read()

### Description

Read the pin value.

**Syntax** value=gpio.read(pin)

### Parameters

pin: gpio ID, 0~17

### Returns

value: 0 - low, 1 - high

### Examples

```
> gpio.mode(0, gpio.INPUT)
> print(gpio.read(0))
> 0
```

## gpio.write()

### Description

Set the pin value.

### Syntax

gpio.write(pin, value)

### Parameters

pin: gpio ID, 0~17

value: 0 or 1 or gpio.HIGH or gpio.LOW

**Returns** nil

### Examples

```
> gpio.mode(0, gpio.OUTPUT)
> gpio.write(0,gpio.HIGH)
> gpio.write(0,0)
```

## gpio.toggle()

### Description

Toggle the pin's output value

**Syntax** gpio.toggle(pin)

### Parameters

pin: gpio ID, 0~17

**Returns** nil

### Examples

```
>gpio.mode(17, gpio.OUTPUT)
>gpio.toggle(17)
>gpio.toggle(17)
```

# TIMER Module

## Function List

tmr.start()	Start a timer with call back function
tmr.stop()	Stop a timer
tmr.stopall()	Stop all the timers
tmr.tick()	Get the current time tick of the MCU (ms) since startup
tmr.delayms()	Delay for a assigned time in millisecond
tmr.delayus()	Delay for a assigned time in microsecond
tmr.wdclr()	Clear the Independent watchdog counter

## Constant

nil

## tmr.start()

### Description

Start a timer with call back function.

### Syntax

```
tmr.start(tmrID, interval, func_cb)
```

### Parameters

tmrID: timer ID, 0~15. 16 timers are supported at present  
interval: interval time for the timer  
func\_cb: Callback function for the timer

**Returns** nil

### Examples

```
> tmr.start(1,1000,function() print("tmr1 is called") end)
> tmr1 is called
tmr1 is called tmr1 is called
```

## tmr.stop()

### Description

Stop a timer

### Syntax

```
tmr.stop(tmrID)
```

### Parameters

tmrID: timer ID, 0~15

### Returns

nil

### Examples

```
> tmr.start(1,1000,function() print("tmr1 is called") end)
> tmr1 is called
tmr1 is called tmr1 is called >tmr. stop(1)
```

## tmr.stopall()

### Description

Stop all the timer.

### Syntax

```
tmr.stopall(tmrID)
```

### Parameters

nil

### Returns

nil

### Examples

```
>tmr. stopall()
```

## tmr.tick()

### Description

Get the current time tick of the MCU (ms) since startup.

**Syntax** tick=tmr.tick()

**Parameters** nil

**Returns** nil

### Examples

```
>print(tmr.tick())
1072237
```



## tmr.delayms()

### Description

Delay for a assigned time in millisecond.

### Syntax

```
tmr.delayms(ms)
```

### Parameters

ms: The delay time in millisecond

### Returns

nil

### Examples

```
> tmr.delayms(1000)
```

## tmr.delayus()

### Description

Delay for a assigned time in microsecond.

### Syntax

```
tmr.delayus(us)
```

### Parameters

us: The delay time in microsecond

### Returns

nil

### Examples

```
> tmr.delayus(1000)
```

## tmr.wdclr()

### Description

Clear the independent watchdog counter.

The default independent watchdog time is 10 senconds.

**Note:** This function should be called if some operations takes more than 10 seconds to complete

### Syntax

```
tmr. wdclr ()
```

### Parameters

nil

### Returns

nil

### Examples

```
> tmr.wdclr()
```

# WiFi Module

## Function list

wifi.startap()	Setup wifi in soft Access Point (AP) Mode, enable DHCP function
wifi.startsta()	Setup wifi in Station Mode (STA), begin to connect a AP
wifi.scan()	Scan APs
wifi.stop()	Close all the Wi-Fi connections, Both in station mode and soft ap mode
wifi.powersave()	Enable IEEE power save mode
wifi.ap.getip()	Get ip address in soft AP mode
wifi.ap.getipadv()	Get advanced net information in soft AP mode: DHCP mode, ip address, gateway, netmask, dns, MAC, broad cast address
wifi.ap.stop()	Close all the Wi-Fi connections in soft ap mode
wifi.sta.getip()	Get ip address in STA mode
wifi.sta.getipadv()	Get advanced net information in STA mode: DHCP mode, ip address, gateway, netmask, dns, MAC, broad cast address
wifi.sta.getlink()	Get the connected AP information in STA mode: Connect status, WiFi signal strength, ssid, bssid.
wifi.sta.stop()	Close all the Wi-Fi connections in STA mode
wifi.sta.ntptime()	Set RTC datetime from ntp server

## Constant

nil

## wifi.startap()

### Description

Setup wifi in soft Access Point (AP) Mode, enable DHCP function.

### Syntax

wifi.startap(cfg)

wifi.startap(cfg,func\_cb)

### Parameters

cfg: lua table, contains the configurations for soft AP mode.

cfg.ssid: soft AP's ssid

cfg.pwd: soft AP's password. It will be an open WiFi if cfg.pwd is empty

cfg.ip: optional. The local ip address of the module, Default: "11.11.11.1"  
 cfg.netmask: optional. Default: "255.255.255.0"  
 cfg.gateway: optional. Default: "11.11.11.1"  
 cfg.dnsSrv: optional. DNS server address. Default: "11.11.11.1"  
 cfg.retry\_interval: optional. retry interval in micro seconds. Default: 1 sec  
 func\_cb: The callback function when the soft AP is setup successfully or the soft AP is shut down.

**Returns** nil

#### Examples

```

>cfg={}
>cfg.ssid="WiFiMCU_Wireless"; cfg.pwd=""
>wifi.startap(cfg)
  
```

## wifi.startsta()

### Description

Setup wifi in Station Mode (STA), begin to connect a AP.

### Syntax

```

wifi.startsta(cfg)
wifi.startsta(cfg, func_cb)
  
```

### Parameters

cfg: lua table, contains the configurations for soft AP mode.  
 cfg.ssid: AP's ssid  
 cfg.pwd: AP's password  
 cfg.dhcp: optional. Set dhcp function: 'enable' is to enable the dhcp function. WiFiMCU will get ip automatically. 'disable' is to disable the dhcp function. It's 'enable' in default.  
 cfg.ip: optional. The local ip address of the module. If cfg.dhcp is 'disable' this parameter must be assigned.  
 cfg.netmask: optional. Netmask. If cfg.dhcp is 'disable' this parameter must be assigned. cfg.gateway: optional. Gateway. If cfg.dhcp is 'disable' this parameter must be assigned. cfg.dnsSrv: optional. DNS server address. If cfg.dhcp is 'disable' this parameter must be assigned.  
 cfg.retry\_interval: optional. retry interval in micro seconds. If cfg.dhcp is 'disable' this parameter must be assigned.  
 func\_cb: The callback function when WiFiMCU had connected to the AP successfully, or WiFiMCU is disconnected from the AP.

**Returns** nil

#### Examples

```

>cfg={}
>cfg.ssid="Doit"; cfg.pwd="123456789"
>wifi.startsta(cfg)
  
```

## wifi.scan()

### Description

Scan AP list and return a Lua table containing the results.

**Syntax** wifi.scan(fun\_cb(t))

### Parameters

func\_cb(t): The callback function when scan is finished. 't' is a Lua table in which the keys are the APs' ssid and values are strings in format (" mac, signal strength, channel, authmode")

**Returns** nil

### Examples

```
> function listap(t) if t then for k,v in pairs(t) do print(k..'t'..v);end else print('no ap') end
end; wifi.scan(listap)
> LoBoInternet 9C:C7:A6:45:B9:E7,100,11,WAP2 MIXED
hHyVEd 58:98:35:B8:3E:17,45,11,WAP2 MIXED
B.net_98796 58:23:8C:83:69:D3,70,1,WPA TKIP
B.net_11651 88:F7:C7:9A:CE:B0,72,6,WPA TKIP
3c3f1e 4C:72:B9:89:0C:FE,40,9,WPA AES
ISKONOVAC-4016CC 2A:28:5D:40:16:CC,30,1,WAP2 MIXED
```

## wifi.stop()

### Description

Close all the Wi-Fi connections, Both in station mode and soft ap mode.

**Syntax** wifi.stop()

**Parameters** nil

**Returns** nil

**See also** wifi.ap.stop() wifi.sta.stop()

### Examples

```
> wifi.stop()
```

## wifi.powersave()

### Description

Enable IEEE power save mode.

### Syntax

```
wifi.powersave ()
```

**Parameters**

nil

**Returns**

nil

### Examples

```
> wifi.powersave ()
```

## wifi.ap.getip()

### Description

Get ip address in AP mode

### Syntax

```
ip=wifi. ap.getip()
```

### Parameters

nil

### Returns

ip: The module ip in soft AP mode.

### Examples

```
> ip=wifi.ap.getip ()
> print(ip)
11.11.11.1
```

## wifi.ap.getipadv()

### Description

Get advanced net information in soft AP mode: DHCP mode, ip address, gate way, net mast, dns, MAC, broadcast address.

### Syntax

```
dhcp,ip,gw,nm,dns,mac,bip =wifi. ap.getipadv()
```

### Parameters

nil

### Returns

dhcp: DHCP mode. in soft AP mode, it will be always "DHCP\_Server"  
ip: ip address.  
gw: gateway address. nm: netmask.  
dns: dns address. mac: MAC address.  
bip: broadcast ip address.

### Examples

```
> dhcp,ip,gw,nm,dns,mac,bip =wifi.ap.getipadv()
> print(dhcp,ip,gw,nm,dns,mac,bip)
DHCP_Server 11.11.11.1 11.11.11.1 255.255.255.0 208.67.222.222 c89346501a62
255.255.255.255
```

## wifi.ap.stop()

### Description

Close all the Wi-Fi connections in soft ap mode.

**Syntax** wifi.ap.stop()

**Parameters** nil

**Returns** nil

**See also** wifi.stop()

wifi.sta.stop()

### Examples

```
> wifi.ap.stop()
```

## wifi.sta.getip()

### Description

Get ip address in STA mode.

### Syntax

```
ip=wifi. sta.getip()
```

**Parameters**

nil

**Returns**

ip: The module ip in STA mode.

### Examples

```
> ip=wifi.sta.getip ()
> print(ip)
192.168.1.108
```

## wifi.sta.getipadv()

### Description

Get advanced net information in STA mode: DHCP mode, ip address, gateway, netmask, dns, MAC, broad cast address.

### Syntax

```
dhcp,ip,gw,nm,dns,mac,bip =wifi. sta.getipadv()
```

**Parameters** nil

### Returns

dhcp: DHCP mode. in STA mode, "DHCP\_Server" or "DHCP\_Client" or DHCP\_Disable ip: ip address.  
gw: gateway address.  
nm: netmask.  
dns: dns address. mac: MAC address.  
bip: broadcast ip address.

### Examples

```
> dhcp,ip,gw,nm,dns,mac,bip =wifi.sta.getipadv() >print(dhcp,ip,gw,nm,dns,mac,bip)
DHCP_Client 192.168.1.108 192.168.1.1 255.255.255.0 192.168.1.1 c89346501a62
255.255.255.255
```

## wifi.sta.getlink()

### Description

Get the connected AP information in STA mode:Connect status, WiFi signal strength, ssid, bssid.

**Syntax** status,strength,ssid,bssid=wifi.sta.getlink()

**Parameters** nil

### Returns

status: The connecting status. if connected it's "connected" else it's "disconnected". It will be nil for strength/ssid/bssid if it's "disconnected".  
strength: The signal strength. ssid: The connected AP's ssid. bssid: The connected AP's bssid.

### Examples

```
> status,strength,ssid,bssid=wifi.sta.getlink() >print(status,strength,ssid,bssid)
connected 62 Doit BC:D1:77:32:E7:2E
```

## wifi.sta.stop()

### Description

Close all the Wi-Fi connections in STA mode.

**Syntax** wifi.sta.stop()

**Parameters** nil

**Returns** nil

**See also** wifi.stop() wifi.ap.stop()

### Examples

```
> wifi.sta.stop()
```

## wifi.sta.ntptime()

### Description

Set RTC datetime from ntp server.

### Syntax

```
wifi.sta.ntptime()  
wifi.sta.ntptime(timezone)  
wifi.sta.ntptime(timezone,ntpserver)
```

### Parameters

timezone: optional, use specified time zone offset from UTC (-12 - +14), default=0  
ntpserver: optional, specify ntp server to use, default="time1.google.com"

### Returns

status: disconnected if no wifi connection detected

### Examples

```
> wifi.sta.ntptime(1)
```



# Net Module

## Function list

net.new()	Create a new socket, set the socket and transmission protocol
net.start()	Start the socket, set remote port, remote ip address, or local port according to the socket and transmission protocol
net.on()	Register the callback functions for socket events
net.send()	Send data
net.close()	Close socket
net.getip()	Get the ip address and port of the client socket.

## Constant

net.TCP	TCP protocol
net.UDP	UDP protocol
net.SERVER	Server type
net.CLIENT	Client type

## net.new()

### Description

Create a new socket, set the socket and protocol type.

Max 4 server and max 4 client can be setup in Wi-Fi MCU.

If the socket type is Server, max number of 5 clients are allowed to connect.

**Syntax** skt=net.new(protocol,type)

### Parameters

protocol: The transmission protocol, must be one of the two: net.TCP, net.UDP

type: socket type, must be one of the two: net.SERVER, net.CLIENT

### Returns

skt: the handle for this socket

### Examples

```
>skt = net.new(net.TCP,net.SERVER) >skt2
```

```
= net.new(net.UDP,net.CLIENT)
```

## net.start()

### Description

Start the socket, set remote port, remote ip address, or local port according to the socket and transmission protocol.

### Syntax

```
net.start(socket, localport)
net.start(socket, remoteport, "domain", [local port])
```

### Parameters

**socket:** The socket handle returned from net.new()  
**localport:** If the socket type is net.SERVER, It's the local binded port for this socket. **remoteport:** If the socket type is net.CLIENT, It's the remote server port.  
**"domain":** If the socket type is net.CLIENT, it's the domain name string for remote server. The remote server's ip address can be used too.  
**local port:** Optional, if the socket type is net.CLIENT, [local port] set the local binded port for the socket. If ignored, a random port would be assigned.

**Returns** nil

### Examples

```
>skt = net.new(net.TCP,net.SERVER)
>skt2 = net.new(net.UDP,net.CLIENT)
>net.start(skt, 80) >net.start(skt2,9000,'11.11.11.2', 8000)
```

## net.on()

### Description

Register the callback functions for socket events.

**Syntax** net.on(socket,event,func\_cb)

### Parameters

**socket:** The socket handle returned from net.new()  
**event:** If the socket type is net.SERVER, event should be one of the following:  
"accept"(TCP server socket only), "receive", "sent", "disconnect".  
If the socket type is net.CLIENT, event should be one of the following:  
"connect"(TCP client socket only)", "receive", "sent", "disconnect", "dnsfound".  
**func\_cb:** Callback function for different events. The function parameters diff from events.  
**"accept":** TCP server socket only. If the tcp server accept a tcp client connection request, the function will be called. Function prototype is: func\_cb(clt, ip, port). "clt" is the tcp client socket handle, "ip" is the client ip address, "port" is the client's port.

“receive”: If data arrived on the assigned socket, the function will be called. Function prototype is: `func_cb(clt, data)`. “clt” is the socket handle, “data” is the received data.

“sent”: When data had sent successfully on the assigned socket, the function will be called. Function prototype is: `func_cb(clt)`. “clt” is the socket handle.

“disconnect”: If the client socket is disconnected from server or some errors happened, the function will be called. Function prototype is: `func_cb(clt)`. “clt” is the socket handle.

“connect”: TCP Client socket only. When the client socket connects to the remote server successfully, the function will be called. Function prototype is: `func_cb(clt)`. “clt” is the socket handle.

“dnsfound”: TCP or UDP Client socket only. When the DNS operations has finished, the function will be called. Function prototype is: `func_cb(clt, ip)`. “clt” is the socket handle, “ip” is the ip address for the domain.

**Returns** nil

### Examples

```
>clt = net.new(net.TCP,net.CLIENT)
>net.on(clt,"dnsfound",function(clt,ip) print("dnsfound clt: "..clt.." ip: "..ip) end)
>net.on(clt,"connect",function(clt) print("connect:clt: "..clt) end)
>net.on(clt,"disconnect",function(clt) print("disconnect:clt: "..clt) end)
>net.on(clt,"receive",function(clt,d) print("receive:clt: "..clt.."data: "..d) end)
>net.start(clt,9003,"11.11.11.2")
```

## net.send()

### Description

Send data.

### Syntax

```
net.send(socket, data, [func_cb])
```

### Parameters

socket: The socket handle returned from `net.new()`

data: Data to be sent.

func\_cb: Optional, “sent” eventcall back function. When data had sent successfully on the assigned socket, the function will be called. Function prototype is: `func_cb(clt)`. “clt” is the socket handle.

**Returns** nil

### Examples

```
>net.send(clt,"hello")
```

## net.close()

### Description

Close socket, release the resource of the socket.

**Syntax**

```
net.close(socket)
```

**Parameters**

socket: The socket handle returned from net.new()

**Returns**

nil

**Examples**

```
>skt = net.new(net.TCP,net.SERVER) >net.close(skt)
```

## net.getip()

**Description**

Get the ip address and port of the client socket.

**Syntax**

```
ip, port = net.getip(socket)
```

**Parameters**

socket: The socket handle returned from net.new(). The socket handle should be a client socket.

**Returns**

ip: the ip address for the socket. port: the port for the socket.

**Examples**

```
>ip, port = net.getip(clt)
```

# File Module

The file system is based on spi flash embedded in Wi-Fi MCU.  
The total storage capacity is ~1550K

## Function list

file.format()	Format file system, all stored data will be lost after format
file.open()	Open or create a file
file.close()	Close an opened file
file.write()	Write data to an opened file
file.writeline()	Write data to an opened file, with a '\n' added at the tailed of data
file.read()	Read data from an opened file
file.readline()	Read a line data from an opened file
file.list()	Get the file name and size list in file system
file.slist()	Print the file name and size list on terminal
file.remove()	Remove file
file.seek()	Set the position of file pointer
file.flush()	Clear file buffer
file.rename()	Rename the file
file.info()	Get the file system storage status
file.state()	Get the opened file's name and size
file.compile()	Compile a Lua scripts file to lc file.
dofile()	Run a file

## Constant

nil

## file.format()

### Description

Format file system, all stored data will be lost after format. It's recommended not to do anythings while formatting.

### Syntax

```
file.format()
```

**Parameters** nil

**Returns** nil

If formatting is done successfully, "format done" will be printed, else "format error" will be printed.

### Examples

```
>file.format() format  
done
```

## file.open()

### Description

Open or create a file.

### Syntax

```
ret = file.open(filename,mode)
```

### Parameters

filename: filename string to be created or opened. Directories are not supported yet.

mode: open type:

"r":	read mode (the default parameter)
"r+":	update mode, all previous data is preserved
"w":	write mode
"w+":	update mode, all previous data is erased
"a":	append mode
"a+":	append update mode, previous data is preserved, writing is only allowed at the end of file

### Returns

ret: true if succeed, else nil.

### Examples

```
>file.open("test.lua","w+")  
>file.write("This is a test") >file.close()
```

## file.close()

### Description

Close an opened file.

### Syntax

```
file.close()
```

### Parameters

nil

### Returns

nil

### Examples

```
>file.open("test.lua","w+")
>file.write("This is a test")
>file.close()
```

## file.write()

### Description

Write data to an opened file.

### Syntax

```
ret=file.write(data)
```

### Parameters

data: The data to be wrote.

### Returns

ret: true if succeed, else nil.

### Examples

```
>file.open("test.lua","w+")
>file.write("This is a test")
>file.close()
```

## file.writeline()

### Description

Write data to an opened file, with a '\n' added at the tailed of data.

### Syntax

```
ret=file.writeline(data)
```

### Parameters

data: The data to be wrote. A char '\n' will be added at the end of data.

### Returns

ret: true if succeed, else nil.

### Examples

```
>file.open("test.lua","w+")
```

```
>file.writeline("This is a test")
>file.close()
```

## file.read()

### Description

Read data from an opened file.

### Syntax

```
ret=file.read()
ret=file.read(num)
ret=file.read(endchar)
```

### Parameters

if the parameter is nil, read all byte in file.

num: if a number is assigned, read the num bytes from file, or all rest data in case of end of file.

endchar: read until endchar or EOF is reached.

### Returns

ret: the file data if succeed, else nil.

### Examples

```
>file.open("test.lua","r")
>data=file.read()
>file.close()
>print(data)
This is a test
>file.open("test.lua","r")
>data=file.read(10)
>file.close()
>print(data)
This is a
>file.open("test.lua","r")
>data=file.read('e')
>file.close()
>print(data)
This is a te
```

## file.readline()

### Description

Read a line data from an opened file.

**Syntax** ret=file.readline ()

**Parameters** nil

### Returns

ret: the file data if succeed, else nil.



## Examples

```
>file.open ("test.lua","w+")
>file.writeline("this is a test")
>file.close()
>file.open("test.lua","r")
>data=file.readline()
>print(data)
This is a test
>file.close()
```

## file.list()

### Description

Get the file name and size list in file system.

**Syntax** ft=file.list()

**Parameters** nil

### Returns

ft: a Lua table, in which the filename is the key, file size is the value.

### Examples

```
>for k,v in pairs(file.list()) do print("name:"..k.." size(bytes):"..v) end
name:test.lua size(bytes):15
```

## file.slist()

### Description

Print the file name and size list on terminal.

**Syntax** file.slist()

**Parameters** nil

**Returns** nil

### Examples

```
>file.slist()
test.lua size:15
```

## file.remove()

### Description

Remove file.

### Syntax

file.remove(filename)

**Parameters**

filename: filename string to be removed.

**Returns**

nil

**Examples**

```
>file.remove("test.lua")
```

## file.seek()

**Description**

Set the position of file pointer.

**Syntax**

```
fi = file.seek(whence, offset)
```

**Parameters**

whence: should be one of the following:

"set": base is position 0 (beginning of the file);

"cur": base is current position;(default value)

"end": base is end of file;

offset: default 0.

**Returns**

fi: the file pointer final position if succeed, else nil.

**Examples**

```
>file.open ("test.lua","r")
>file.seek("set",10)
>data=file.read()
>file.close()
>print(data)
test
```

## file.flush()

**Description**

Clear file buffer.

**Syntax**

```
ret = file.flush()
```

**Parameters** nil**Returns**

ret: true if succeed, else nil.

**Examples**

```
>file.open ("test.lua","r")
>file.flush ()
```

```
>file.close()
```

## file.rename()

### Description

Rename the file.

### Syntax

```
ret=file.rename(oldname,newname)
```

### Parameters

oldname: File name to be changed.

newname: New file name.

### Returns

ret: true if succeed, else nil.

### Examples

```
> file.slist()
test.lua size:14
>file.rename('test.lua',' testNew.lua')
>file.slist()
testNew.lua size:14
```

## file.info()

### Description

Get the file system storage status.

### Syntax

```
last,used,total = file.info()
```

### Parameters

nil

### Returns

last: free storage left in bytes. used: used storage in bytes.

total: all allocated storage for file system in bytes.

### Examples

```
> last,used,total = file.info()
> print(last,used,total)
1140500 2750 1143250
```

## file.state()

### Description

Get the opened file's name and size

**Syntax**

```
fn,sz = file.state()
```

**Parameters** nil**Returns**

fn: filename.

sz: file size in bytes.

**Examples**

```
>file.open("testNew.lua","r")
>fn,sz = file.state()
>file.close()
>print(fn,sz)
testNew.lua 14
```

## file.compile()

**Description**

Compile a Lua scripts file to lc file. The lc file will be named as the same name as the Lua file.

**Syntax** file.compile('filename.lua')**Parameters**

filename.lua: file name of the Lua scripts.

**Returns** nil.**Examples**

```
>file.open("test.lua","w+")
>file.write("print('Hello world!')")
>file.close()
>file.compile("test.lua")
>file.slist()
test.lua size:21 test.lc size:100
```

## dofile()

**Description**

Run a file. The file can be either a Lua scripts or a lc format file.

**Syntax**

```
dofile('filename.lua')
dofile('filename.lc')
```

**Parameters**

filename.lua: Lua scripts file.

filename.lc: a lc file

**Returns** nil.

## Examples

```
>dofile("test.lua")
```

```
Hello world!
```

```
>dofile("test.lc")
```

```
Hello world!
```

# PWM Module

## Function list

pwm.start()	Start pwm function at assigned gpio pin
pwm.stop()	Stop pwm

## Constant

nil

## Pin Table

Plaese refer: "GPIO Table" for detail.

## pwm.start()

### Description

Start pwm function at assigned gpio pin.

### Syntax

```
pwm.start(pin, freq, duty)
```

### Parameters

pin: gpio pin ID. There are 11 PWM ports supported in WiFiMCU:  
D1, D3, D4, D9, D10, D11, D12, D13, D14, D15, D16.

freq: PWM output frequency in Hz,  $0 < \text{freq} < 10\text{KHz}$

duty: Duty of PWM output, must be  $0 \leq \text{duty} \leq 100$

**Returns** nil.

### Examples

```
>i=1;pin=1;  
>tmr.start(1,1000,function() i=i+10;if i>=100 then i=1 end pwm.start(pin,10000,i)  
end)  
>
```

## **pwm.stop()**

### **Description**

Stop pwm.

**Syntax** `pwm.stop(pin)`

### **Parameters**

pin: gpio pin ID. There are 11 PWM ports supported in WiFiMCU: D1, D3, D4, D9, D10, D11, D12, D13, D14, D15, D16.

**Returns** nil.

### **Examples**

```
>pwm.stop(1)
```

# ADC Module

## Function list

<code>adc.read()</code>	Read the ADC result at assigned pin
<code>adc.readmv()</code>	Read the ADC result in mV at assigned pin

## Constant

`nil`

## Pin Table

Plaese refer: "GPIO Table" for detail.

## `adc.read()`

### Description

Read the ADC result at assigned pin.

### Syntax

```
data= adc.read(pin)
```

### Parameters

pin: gpio pin ID.

There are 5 ADC ports supported in WiFiMCU: D1, D13, D15, D16, D17.

### Returns

data: if succeed, data between 0~4095 is returned, else nil.

Note that: 0 presents 0V, 4095 presents 3.3V.

### Examples

```
>=adc.read(1)
1
>=adc.read(1)
4095
```



## adc.readmv()

### Description

Read the ADC result in mV (mili volts) at assigned pin.

### Syntax

```
data= adc.readmv(pin)
```

### Parameters

pin: gpio pin ID.

There are 5 ADC ports supported in WiFiMCU: D1, D13, D15, D16, D17.

### Returns

data: if succeed, data between 0~3300 is returned, else nil.

### Examples

```
>=adc.read(1)
1
>=adc.read(1)
4095
```

# UART Module

Only one uart is supported in Wi-Fi MCU so far. The GPIO pin is D8(RX1), D9(TX1).

## Function list

uart.setup()	Setup uart parameters: baudrate, databits, parity, stopbits.
uart.on()	Register the callback functions for uart events
uart.send()	Send data via uart

## Constant

null

**Warning:** UART shares Rx pin with hardware SPI5. You can't use both at the same time.

## uart.setup()

### Description

Setup uart parameters: baudrate, databits, parity, stopbits.

### Syntax

```
uart.setup(id, baud, parity, databits, stopbits)
```

### Parameters

id: uart ID, always 1 at present.  
baud: baudrate, such as: 4800, 9600, 115200.  
parity: 'n': no parity, 'o': odd parity, 'e': even parity.  
databits: data bits, '5', '6', '7', '8', '9'.  
stopbits: stop bits, '1', '2'

**Returns** nil

### Examples

```
>uart.setup(1,9600,'n','8','1')
```

## uart.on()

### Description

Register the callback functions for uart events.

### Syntax

```
uart.on(id, event ,func_cb)
```

### Parameters

id: uart ID, always 1 at present.  
event: always "data".  
func\_cb: Callback function for the event. When data arrived, the function will be called. Function prototype is: func\_cb(data). "data" is the data received.

**Returns** nil

### Examples

```
>uart.on(1, 'data',function(t) len=string.len(t) print(len.."  "..t) uart.send(1,t) end)
```

## uart.send()

### Description

Send data via uart.

### Syntax

```
uart.send(1, string1,[number],...[stringn])
```

### Parameters

id: uart ID, always 1 at present.  
string1: string ready to send.  
[number]: Optional, number ready to send.  
[stringn]: Optional, The nth string ready to be send.

**Returns** nil

### Examples

```
>uart.send(1,'hello wifimcu')  
>uart.send(1,'hello wifimcu','hi',string.char(0x32,0x35))  
>uart.send(1,string.char(0x01,0x02,0x03))
```

## Bit Module

### Function List

bit.bnot	Bitwise negation
bit.band	Bitwise AND
bit.bor	Bitwise OR
bit.bxor	Bitwise XOR
bit.lshift	Logical left shift a number
bit.rshift	Logical right shift a number
bit.arshift	Arithmetic right shift a number
bit.bit	Generate a number with a 1 bit (used for mask generation)
bit.set	Set bits in a number
bit.clear	Clear bits in a number
bit.isset	Test if a given bit is set
bit.isclear	Test if a given bit is cleared

### Constant

nil

### bit.bnot()

#### Description

Bitwise negation.

**Syntax** num=bit.bnot(val)

#### Parameters

val: the number to negation, value is 32 bit width.

#### Returns

num: the bitwise negated value of the number.

#### Examples

```
>print("result: "..bit.bnot(0x00000000))  
result: -1
```

## bit.band()

### Description

Bitwise AND.

### Syntax

```
num= bit.band(val1, val2, ... valn)
```

### Parameters

val1: the first number to AND val1: the second number to AND  
valn: the nth number to AND

### Returns

num: the bitwise AND of all the arguments.

### Examples

```
> print("result: "..bit.band(0xffffffff, 0x000000ff, 0x000000f))  
result: 15
```

## bit.bor()

### Description

Bitwise OR.

### Syntax

```
num= bit.bor(val1, val2, ... valn)
```

### Parameters

val1: the first number to OR val1: the second number to OR  
valn: the nth number to OR

### Returns

num: the bitwise OR of all the arguments.

### Examples

```
> print("result: "..bit.bor(0x00000000, 0x000000ff, 0x000000f)) result: 255
```

## bit.bxor()

### Description

Bitwise XOR.

### Syntax

```
num= bit.bxor(val1, val2, ... valn)
```

### Parameters

val1: the first number to XOR  
val1: the second number to XOR

valn: the nth number to XOR

### Returns

num: the bitwise XOR of all the arguments.

### Examples

```
> print("result: "..bit.bxor(0x00000000, 0x000000ff, 0x000000f))  
result: 240
```

## bit.lshift()

### Description

Logical left shift a number.

### Syntax

```
num= bit.lshift(val, shift)
```

### Parameters

val: the value to shift shift: positions to shift

### Returns

num: the number shifted left.

### Examples

```
> print("result: "..bit.lshift(0x00000001,8))  
result: 256
```

## bit.rshift()

### Description

Logical right shift a number.

### Syntax

```
num= bit.rshift(val, shift)
```

### Parameters

val: the value to shift shift: positions to shift

### Returns

num: the number shifted right.

### Examples

```
> print("result: "..bit.rshift(0x00000080,1))  
result: 64
```

## bit.arshift()

### Description

Arithmetic right shift a number.

### Syntax

```
num= bit.arshift(val, shift)
```

**Parameters**

val: the value to shift shift: positions to shift

**Returns**

num: the number arithmetically shifted right.

**Examples**

```
> print("result: "..bit.arshift(0x00000080,1))  
result: 64
```

## bit.bit()

**Description**

Generate a number with a 1 bit (used for mask generation).

**Syntax**

```
num= bit.bit(pos)
```

**Parameters**

pos: position of the bit that will be set to 1.

**Returns**

num: the number that only one bit is set to 1 and 0 for the rests.

**Examples**

```
> print("result: "..bit.bit(8))  
result: 256
```

## bit.set()

**Description**

Set bits in a number.

**Syntax**

```
num= bit.set(val, pos1,pos2,...,posn)
```

**Parameters**

val: the base number.

pos1: first position to be set. pos2: second position to be set. posn: nth position to be set.

**Returns**

num: the number with the bit(s) set in the given position(s)..

**Examples**

```
> print("result: "..bit.set(0x00000000, 0, 1, 2, 3))  
result: 15
```

## bit.clear()

### Description

Clear bits in a number.

### Syntax

```
num= bit.clear (val, pos1,pos2,...,posn)
```

### Parameters

val: the base number.

pos1: first position to be cleared. pos2: second position to be cleared. posn: nth position to be cleared.

### Returns

num: the number with the bit(s) cleared in the given position(s).

### Examples

```
> print("result: "..bit.clear(0x0000000f, 0, 1, 2, 3))  
result: 0
```

## bit.isset()

### Description

Test if a given bit is set.

### Syntax

```
res= bit.isset (val, pos)
```

### Parameters

val: the value number to be test pos: bit position.

### Returns

res: true if the bit at the given position is 1, false otherwise.

### Examples

```
>=bit.isset(0x0000000f, 1) true >=bit.isset(0x0000000f, 5)  
false
```

## bit.isclear()

### Description

Test if a given bit is cleared.

### Syntax

```
res= bit.isclear (val, pos)
```

### Parameters

val: the value number to be test



pos: bit position.

**Returns**

res: true if the bit at the given position is 0, false otherwise.

**Examples**

```
>=bit.isclear(0x0000000f, 1)
false
>=bit.isclear (0x0000000f, 5)
true
```

# Sensor Module

## Function List

sensor.dht11.init	Init dht11, Assign the GPIO Pin for dht11.
sensor.dht11.get	Get the dht11 temperature and humidity value
sensor.ds18b20.init	Init DS18B20, Assign the GPIO Pin for ow.
sensor.ds18b20.gettemp	Start temperature measurement and get the temperature
sensor.ds18b20.search	Search for DS18B20 1-wire devices
sensor.ds18b20.setres	Set DS18B20 resolution (9,10,11,12 bit)
sensor.ds18b20.getres	Get current DS18B20 resolution (9,10,11,12 bit)
sensor.ds18b20.getrom	Get DS18B20 ROM values (returns 8 element table)
sensor.ow.init	Init 1-wire device, Assign the GPIO Pin for ow.
sensor.ow.search	Search for 1-wire devices

## Constant

```
sensor.ds18b20.DS18B20_RES9    DS18B20 9 bit resolution
sensor.ds18b20.DS18B20_RES10   DS18B20 9 bit resolution
sensor.ds18b20.DS18B20_RES11   DS18B20 9 bit resolution
sensor.ds18b20.DS18B20_RES12   DS18B20 9 bit resolution
```

## sensor.dht11.init()

### Description

Init dht11 sensor. Assign the GPIO Pin for dht11.

### Syntax

```
res = sensor.dht11.init(pin)
```

### Parameters

pin: gpio ID, 0~17.

### Returns

res: true if dht11 initialization successfully, false otherwise.

### Examples

```
>=sensor.dht11.init(1) true
```

## sensor.dth11.get()

### Description

Get the dth11 temperature and humidity value.

### Syntax

```
temp, hum = sensor.dth11.get()
```

### Parameters

nil

### Returns

temp: temperature measured by dth11.

hum: humidity measured by dth11.

### Examples

```
>=sensor.dth11.get (1) 26  
65
```

## sensor.ds18b20.init()

### Description

Init ds18b20 sensor. Assign the GPIO Pin for 1-wire.

### Syntax

```
res = sensor.ds18b20.init(pin)
```

### Parameters

pin: gpio ID, 0~17.

### Returns

res: true if ds18b20 initialization successfully, false otherwise.

### Examples

```
>=sensor.ds18b20.init(7)  
true
```

## sensor.ds18b20.search()

### Description

Search for DS18B20 1-wire devices.

### Syntax

```
res = sensor.ds18b20.search()
```

### Parameters

nil

### Returns

res: Number of found DS18B20 devices.

### Examples

```
>=sensor.ds18b20.search()  
1
```

## sensor.ds18b20.gettemp()

### Description

Start temperature measurement and get the temperature.

### Syntax

```
tmp, n = sensor.ds18b20.gettemp(dev)
```

### Parameters

dev: ds18b20 device number, 1~num of detected devices with sensor.ds18b20.search()

### Returns

tmp: temperature

n: duration of the measurement in msec (depends on current ds18b20 resolution)

### Examples

```
> = sensor.ds18b20.gettemp(1)
22.1875 591
```

## sensor.ds18b20.setres()

### Description

Set DS18B20 resolution (9,10,11,12 bit).

### Syntax

```
sensor.ds18b20.setres(dev, res)
```

### Parameters

dev: ds18b20 device number, 1~num of detected devices with sensor.ds18b20.search()

res: resolution (9,10,11,12 bit)

### Returns

nil

### Examples

```
> = sensor.ds18b20.setres(1,10)
```

## sensor.ds18b20.getres()

### Description

Get DS18B20 current resolution.

### Syntax

```
Res = sensor.ds18b20.getres(dev)
```

### Parameters

dev: ds18b20 device number, 1~num of detected devices with sensor.ds18b20.search()

### Returns

res: resolution (9,10,11,12 bit)

### Examples

```
> = sensor.ds18b20.getres(1)
10
```

## sensor.ds18b20.getrom()

### Description

Get DS18B20 ROM values (returns 8 element table).

### Syntax

```
rom = sensor.ds18b20.getrom(dev)
```

### Parameters

dev: ds18b20 device number, 1~num of detected devices with sensor.ds18b20.search()

### Returns

rom: Table with 8 ROM values

### Examples

```
> rom=sensor.ds18b20.getrom(1); for i=1,9,1 do print(dsrom[i]) end
40
142
106
200
0
0
0
110
```

## sensor.ow.init()

### Description

Init 1-wire device. Assign the GPIO Pin for 1-wire.

### Syntax

```
res = sensor.ow.init(pin)
```

### Parameters

pin: gpio ID, 0~17.

### Returns

res: true if 1-wire initialization successfully, false otherwise.

### Examples

```
>=sensor.ow.init(7)
true
```

## sensor.ow.search()

### Description

Search for any 1-wire devices.

### Syntax

```
res = sensor.ow.search()
```

**Parameters**

nil

**Returns**

res: Number of found 1-wire devices.

**Examples**

```
>=sensor.ds18b20.search()  
1
```

# SPI Module

## Function List

spi.setup	Init spi, assign GPIO pins
spi.write	Write data via spi interface, data can be multi numbers, string or lua table
spi.read	Read data from spi interface
spi.deinit	Deinitializes the SPI, free gpio pins

## Constant

spi.BITS_8	8 Bits data length
spi.BITS_16	16 Bits data length

**Warning:** UART shares Rx pin with hardware SPI5. You can't use both at the same time.

## spi.setup()

### Description

Initialize SPI. SPI module works in MASTER mode.

### Syntax

```
spi.setup(id, config)
```

### Parameters

id: 0 for software SPI; 2 for hardware SPI5

config: Lua table with spi configuration parameters:

**mode**=spi\_mode 0,1,2,3

**speed**=spi\_speed spi clock frequency in kHz; 100~5000 for software spi, 400~50000 for hardware spi

**cs**=pin gpio ID, 0~17

**rw**=flag *optional*; 1 reads from MOSI while writing; 1 no read while write

*The following parameters are only for software SPI:*

**sck**=pin gpio ID, 0~17 used for SCK

**mosi**=pin gpio ID, 0~17 used for MOSI

**miso**=pin *optional*; gpio ID, 0~17 used for MISO

**Returns**

0 is succes; error code if not

**Examples**

```
-- hardware SPI5
>res = spi.setup(2,{mode=3, cs=12, speed=15000})
-- sftware SPI
>res = spi.setup(0,{mode=3, cs=12, speed=1000,sck=2, mosi=4})
```

## spi.write()

**Description**

Write data via spi interface. Data can be multi numbers, string or lua table

**Syntax**

```
ret = spi.write(id, databits, data1, [data2],...,[datan] )
```

**Parameters**

id: 0 for software SPI; 2 for hardware SPI5  
databits: write databits. spi. BITS\_8 or spi. BITS\_16.  
data1: should be 0<data1< 255 in spi. BITS\_8 mode  
or 0<data2<65535 in spi. BITS\_16 mode.  
data2: optional.  
datan: optional.

**Returns**

ret: The number of data written.

**Examples**

```
>res = spi.setup(0,{mode=3, cs=12, speed=1000,sck=2, mosi=4})
>ret = spi.write(0, 0xAA)
```

## spi.read()

**Description**

Read data via spi interface.

**Syntax**

```
ret = spi.read(id, databits, num)
```

**Parameters**

id: 0 for software SPI; 2 for hardware SPI5  
databits: write databits. spi. BITS\_8 or spi. BITS\_16.  
num: the number of data to read.

**Returns**

ret: the Lua table of read data.

**Examples**

```
> ret = spi.read(0, 2)
>print(ret[1]); print(ret[2])
```



## **spi.deinit()**

### **Description**

Deinitializes the SPI, free gpio pins.

### **Syntax**

```
ret = spi.deinit(id)
```

### **Parameters**

id: 0 for software SPI; 2 for hardware SPI5

### **Returns**

ret: 0 on success; err code if error

### **Examples**

```
> ret = spi.deinit(2)
```

## I2C Module

### Function List

i2c.setup	Init i2c, assign GPIO pin
i2c.start	Send start condition
i2c.stop	Send stop condition
i2c.address	Send i2c device address and set transmission direction
i2c.write	Write data via i2c interface, data can be multi numbers, string or lua table
i2c.read	Read data from i2c interface

### Constant

nil

### i2c.setup()

#### Description

Init i2c, assign GPIO pin.

#### Syntax

```
i2c.setup(id, pinSDA, pinSCL)
```

#### Parameters

id: i2c id, should be 0.

pinSDA: GPIO Pin 0~17

pinSCL: GPIO Pin 0~17

**Returns** nil

#### Examples

```
>i2c.setup(0, 1, 2)
```

## i2c.start()

### Description

Send start condition

### Syntax

```
i2c.start(id)
```

### Parameters

nil

### Returns

nil

### Examples

```
>i2c.start(0)
```

## i2c.stop()

### Description

Send stop condition

### Syntax

```
i2c.stop(id)
```

### Parameters

id: i2c id, should be 0.

### Returns

nil

### Examples

```
>i2c.stop(0)
```

## i2c.address()

### Description

Send i2c device address and set transmission direction.

### Syntax

```
ack = i2c.address(id, dev_id, mode)
```

### Parameters

id: i2c id, should be 0. dev\_id: device id address.  
mode: transmission direction. Should ether 'r' or 'w'.

### Returns

ack: if slave device acknowledge success, return true, else return nil.

### Examples

```
> i2c.address(0, 0x3C)
```

## i2c.write()

### Description

Write data via i2c interface, data can be multi numbers, string or lua table

### Syntax

```
ret = i2c.write(id, data1, [data2],...,[datan] )
```

### Parameters

id:            should be 0.  
data1:        should be  $0 < \text{data1} < 255$ .  
data2:        optional.  
datan:        optional.

### Returns

ret: The number of data wrote.

### Examples

```
> ret = i2c.write(0, 0x00)
```

## i2c.read()

### Description

Read data from i2c interface

### Syntax

```
ret = i2c.read(id, num)
```

### Parameters

id:            should be 0.  
num:           the number of data ready to read.

### Returns

ret: the string of read data.

### Examples

```
> ret = i2c.read(0, 1)  
> print(ret)
```

## RTC Module

### Function List

rtc.getasc	Get text representation of current date&time from RTC
rtc.get	Get Lua Table with second, minute, hour, weekday, date, month, year from RTC
rtc.getstrf	Get formatted string representing the current datetime
rtc.set	Set RTC second, minute, hour, weekday, date, month, year
rtc.standby	Terminate all threads, put CPU to standby for specified number of seconds
rtc.standbyUntil	Put CPU to standby until specified time

### rtc.getasc()

#### Description

Get text representation of current date&time from RTC

#### Syntax

```
strtime = rtc.getasc()
```

#### Parameters

nil

#### Returns

string: Current date & time

#### Examples

```
> =rtc.getasc()  
Wed Nov 4 15:56:06 2015  
>
```

### rtc.getstrf()

#### Description

Get formatted string representing the current datetime from RTC

#### Syntax

```
strtime = rtc.getstrf(format)
```

## Parameters

Format: format string, default: "%Y-%m-%d %H:%M:%S"

fmt	Replaced by	Example
%a	Abbreviated weekday name *	Thu
%A	Full weekday name *	Thursday
%b	Abbreviated month name *	Aug
%B	Full month name *	August
%c	Date and time representation *	Thu Aug 23 14:55:02 2001
%C	Year divided by 100 and truncated to integer (00-99)	20
%d	Day of the month, zero-padded (01-31)	23
%D	Short MM/DD/YY date, equivalent to %m/%d/%y	08/23/01
%e	Day of the month, space-padded ( 1-31)	23
%F	Short YYYY-MM-DD date, equivalent to %Y-%m-%d	2001-08-23
%g	Week-based year, last two digits (00-99)	01
%G	Week-based year	2001
%h	Abbreviated month name * (same as %b)	Aug
%H	Hour in 24h format (00-23)	14
%I	Hour in 12h format (01-12)	02
%j	Day of the year (001-366)	235
%m	Month as a decimal number (01-12)	08
%M	Minute (00-59)	55
%n	New-line character ('\n')	
%p	AM or PM designation	PM
%r	12-hour clock time *	02:55:02 pm
%R	24-hour HH:MM time, equivalent to %H:%M	14:55
%S	Second (00-61)	02
%t	Horizontal-tab character ('\t')	
%T	ISO 8601 time format (HH:MM:SS), equivalent to %H:%M:%S	14:55:02

%u	ISO 8601 weekday as number with Monday as 1 (1-7)	4
%x	Date representation *	08/23/01
%X	Time representation *	14:55:02
%y	Year, last two digits (00-99)	01
%Y	Year	2001
%%	A % sign	%

### Returns

string: Formated current date & time

### Examples

```
> =rtc.getstrf("%c")
Sun Nov 22 18:12:02 2015
> =rtc.getstrf("%H:%M:%S Date: %m/%d")
18:14:25 Date: 11/22
```

## rtc.get()

### Description

Get second, minute, hour, weekday, date, month, year from RTC

### Syntax

```
curtime = rtc.get()
```

### Parameters

nil

### Returns

Curtime: Lua table with current second, minute, hour, weekday, date, month, year

### Examples

```
> ct=rtc.get()
> for i=1,7,1 do print(ct[i]); end
11
59
17
0
22
11
2015
```

## rtc.set()

### Description

Set RTC second, minute, hour, weekday, date, month, year

**Syntax**

```
res=rtc.set(timetbl)
```

**Parameters**

timetbl: Lua table with second, minute, hour, weekday, date, month, year

**Returns**

res: 1 if date&time are set or 0 if error

**Examples**

```
> =rtc.set(53,57,15,3,4,11,15)
OK
>
```

## rtc.standby()

**Description**

Terminate all threads, put CPU to sleep for specified number of seconds

**Note: MCU is reset on wakeup.**

**Syntax**

```
rtc.standby(numsec)
```

**Parameters**

numsec: number of seconds to stay in standby

**Returns**

nil, after wake up CPU resets

**Examples**

```
> rtc.standby(5)
[5922990][Platform: platform_mcu_powersave.c: 430] Wake up in 5 seconds

WiFiMCU Lua starting...(Free memory 65544 bytes)
[5][Platform: mico_platform_common.c: 99] Platform initialised, build by IAR
Current Time: Wed Nov 4 16:11:47 2015

[ Ver. 0.9.6_lobo_0.1 WiFiMCU Team, modified by LoBo @2015 ]

Executing init.lua...
>
```

## rtc.standbyUntil()

**Description**

Put CPU to sleep until specified time

**Note: MCU is reset on wakeup.**



**Syntax**

`rtc.standbyUntil(time)`

**Parameters**

Time: Lua table with hour, minute, seconds to wake up at

**Returns**

nil, after wake up CPU resets

**Examples**

```
> rtc.standbyUntil(16,16,5)
Standby until 16:16:05
```

```
WiFiMCU Lua starting...(Free memory 65544 bytes)
[5][Platform: mico_platform_common.c: 99] Platform initialised, build by IAR
Current Time: Wed Nov 4 16:16:05 2015
```

```
[ Ver. 0.9.6_lobo_0.1 WiFiMCU Team, modified by LoBo @2015 ]
```

```
Executing init.lua...
>
```

# OLED Module

## Function List

oled.init	Initialize the oled display,
oled.clear	Clear the screen
oled.write	Write strings and/or numbers to display
oled.writechar	Write one character to display
oled.fontsize	Select the font size
oled.charspace	Define the space between characters
oled.inverse	Select normal or inverted write
oled.fixedwidth	Set fixed width or proportional character printing

The module supports operations with small (0.96" ~ 1.3") oled displays based on SSD1306 controller, using the 4-wire SPI interface.

## oled.init()

### Description

Initialize the oled display and clear the screen.  
You must initialize the SPI interface first.

### Syntax

```
res = oled.init(spi_id, DCpin)
```

### Parameters

spi\_id: id of the SPI interface to be used for oled  
DCpin: gpio ID, 0~17 used for DC control

### Returns

res: 0 on success, error code on error

### Examples

```
-- hardware spi
>spi.setup(2,{mode=3, cs=12, speed=15000})
>res = oled.init(2,14)
-- software spi
>spi.setup(0,{mode=3, cs=12, mosi=9, sck=16, speed=500})
>oled.init(0,14)
```

## oled.clear()

### Description

Clear screen.

### Syntax

```
oled.clear()
```

### Parameters

nil

### Returns

nil

### Examples

```
>oled.clear()
```

## oled.write()

### Description

Write strings and/or numbers to display.

### Syntax

```
oled.write(x, y, ndec, data1, [data2], ... [datan])
```

### Parameters

x: x position (column; 0~127)

y: y position (row; 0~7)

ndec: number of decimal places if number data is float; 0 to print integer)

data1: number or string to write to the display

data2: optional

datan: optional

### Returns

nil

### Examples

```
>oled.write(0,0,0,"WiFiMCU")
```

```
>t=2.3456
```

```
>oled.write(8,2,1,"Temp=", t)
```

## oled.writechar()

### Description

Write single character to display.

### Syntax

```
oled.write(x, y, char)
```

**Parameters**

x: x position (column; 0~127)  
y: y position (row; 0~7)  
char: character code

**Returns**

nil

**Examples**

```
>oled.writechar(16,5,0x42)
```

## oled.fontsize()

**Description**

Set the font size (height). At the moment can be only 8 or 16

**Syntax**

```
oled.fontsize(size)
```

**Parameters**

size: new font size (8 or 16)

**Returns**

nil

**Examples**

```
>oled.fontsize(16)
```

## oled.charspace()

**Description**

Set additional space between characters in pixels.

**Syntax**

```
oled.charspace(chr_spc)
```

**Parameters**

chr\_spc: new intercharacter space (0~8)

**Returns**

nil

**Examples**

```
>oled.charspace(1)
```

## oled.inverse()

**Description**

Set normal (light on dark) or inverse (dark on light) display.

**Syntax**

```
oled.inverse(flag)
```

**Parameters**

flag: 0 for normal, 1 for inverse

**Returns**

nil

**Examples**

```
>oled.inverse(1)
```

## oled.fixedwidth()

**Description**

Set fixed width or proportional character printing.

**Syntax**

```
oled.fixedwidth(flag)
```

**Parameters**

flag: 0 to print proportional character, 1 for fixed width

**Returns**

nil

**Examples**

```
>oled.fixedwidth(1)
>oled.write(0,0,0,"III\r\nMMMM")
III
MMMM
>oled.fixedwidth(0)
>oled.write(0,0,0,"III\r\nMMMM")
III
MMMM
```