WiFiMCU 参考手册

Doit

2015 年 8 月 15 日

修订记录

Date	Name	Detail	Version	Other
2015-7-24	yp	初版	V1.0.0	Null
2015-7-26	yp	增加 PWM、ADC 模块	V1.0.1	Null
2015-7-27	yp	增加 uart 模块	V1.0.2	Null
2015-7-28	yp	调整 GPIO 定义、新增 Lua 内置模块内容	V1.0.3	Null
2015-7-29	yp	调整 tmr 模块函数，新增 mcu.chipid()函数	V1.0.4	Null
2015-7-30	yp	新增 mcu.bootreason()函数	V1.0.5	Null
2015-8-15	yp	新增 tmr.stopall()函数 修改 net 模块函数 新增 WiFi 模块例子	V1.0.6	Null

目 录

Lua 内置模块

WiFiMCU 基于 Lua 5.3.1（发布于 2015-06-10），支持库见下表。

luaopen_base	支持
luaopen_package	支持
luaopen_coroutine	支持
luaopen_string	支持
luaopen_table	支持
luaopen_math	支持
luaopen_utf8	不支持
luaopen_io	不支持
luaopen_os	不支持
luaopen_debug	不支持

关于模块的使用方法可参考官方手册：
http://www.lua.org/manual/5.3/
或中文手册：
http://cloudwu.github.io/lua53doc/manual.html
https://github.com/cloudwu/lua53doc

MCU 模块

函数列表

mcu.ver()	查询 WiFiMCU 固件版本号
mcu.info()	查询 WiFi 模块库函数版本、MAC 地址、WiFi 驱动版本号
mcu.reboot()	重启 WiFi 模块
mcu.mem()	查询 WiFi 模块内存使用情况
mcu.chipid()	获取 MCU 的唯一产品身份标识（96 位）
mcu.bootreason()	获取 MCU 本次启动原因

常数

无

mcu.ver()

调用格式：

nv,bd=mcu.ver()

说明：

查询 WiFiMCU 固件版本号

参数：

nil

返回值：

nv：string 类型；WiFiMCU 版本号

bd：string 类型；编译日期

示例：

>nv,bd=mcu.ver()

>print(nv,bd)

> WiFiMCU 0.9.0 build 20150722

mcu.info()

调用格式：

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

说明：

查询 WiFi 模块库函数版本、MAC 地址、WiFi 驱动版本号

参数：

nil

返回值：

libv：string 类型模块函数库版本

mac：模块 MAC 地址

drv：WiFi 驱动版本

示例：

>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()

调用格式：

mcu.reboot()

说明：

重启 WiFi 模块

参数：

nil

返回值：

nil

示例：

>mcu.reboot()

mcu.mem()

调用格式：

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

说明：

查询 WiFi 模块内存使用情况

参数：

nil

返回值：

fm：number 类型，剩余内存(total free space)

tas：number 类型，总共可分配内存(total allocated space)

mtas：number 类型，最大分配内存(maximum total allocated space)

fc：number 类型，剩余的 chunk 数(number of free chunks)

示例：

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

>print(fm,tas,mtas,fc)

> 31832 54184 86016 27

mcu.chipid()

调用格式：

chipid= mcu.chipid()

说明：

获取 MCU 的唯一产品身份标识（96 位）。

参数：

nil

返回值：

chipid：string 类型，产品标识 id 号。

示例：

> mcu.chipid()

> 0200C000FDFFF42405E5F000

mcu.bootreason()

调用格式：

bootreason= mcu. bootreason()

说明：

获取 MCU 本次启动原因。

参数：

nil

返回值：

bootreason：string 类型，启动原因有如下：

"NONE"：未成功获取；

"SOFT_RST"：软重启；

"PWRON_RST"：上电重启；

"EXPIN_RST"：外部重启引脚引发重启；

"WDG_RST"：看门狗复位重启；

"WWDG_RST"：窗口看门狗复位重启；

"LOWPWR_RST"：低电压重启；

"BOR_RST"：BOR 欠压重启；

示例：

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

GPIO 模块

函数列表

gpio.mode()	定义 GPIO 管脚功能，可定义为输入、输出、中断三种基本模式
gpio.read()	读取指定 pin 的当前输入值
gpio.write()	读取指定 pin 的当前输入值
gpio.toggle()	反转指定 pin 的输出值

常数

gpio.INPUT	输入模式（同输入上拉模式）
gpio.INPUT_PULL_UP	输入上拉模式
gpio.INPUT_PULL_DOWN	输入下拉模式
gpio.INPUT_INPUT_HIGH_IMPEDANCE_DOWN	输入高阻模式
gpio.OUTPUT	输出模式（同输出上拉模式）
gpio.OUTPUT_PUSH_PULL	输出上拉模式
gpio.OUTPUT_OPEN_DRAIN_NO_PULL	输出开漏无上拉
gpio.OUTPUT_OPEN_DRAIN_PULL_UP	输出开漏带上拉
gpio.INT	中断模式
gpio.HIGH	高电平
gpio.LOW	低电平

GPIO 对应关系表

WiFiMCU 定义	其他功能	说明
D0	BOOT	在启动时拉低本管脚将进入 Bootloader
D1	PWM/ADC	
D2		
D3	PWM	
D4		
D5		SWCLK
D6		SWDIO
D7		
D8	PWM	RX1
D9	PWM	TX1

D10	PWM	SCL 带上拉电阻
D11	PWM	SDA 带上拉电阻
D12	PWM	
D13	PWM/ADC	
D14	PWM	
D15	PWM/ADC	
D16	PWM/ADC	
D17	ADC	LED

gpio.mode

调用格式：

gpio.mode(pin, mode)

gpio.mode(pin, gpio.INT, intMode,function)

说明：

定义 GPIO 管脚功能，可定义为输入、输出、中断三种基本模式。输入可以定义为上拉电阻、下拉电阻、高阻模式；输出可定义为上拉电阻、开漏无上拉电阻、开漏带上拉电阻等模式；中断可定义上升沿触发（“rising”）、下降沿触发（“falling”）和上升及下降沿触发（“both”）。

参数：

pin：gpio 的 ID，范围 0~17。

mode：

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;

intMode: 触发方式。”rising”，”falling”，”both”

function: 中断处理函数

返回值：

nil

示例：

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

gpio.read()

调用格式：

b=gpio.read(pin)

说明：

读取指定 pin 的当前输入值。

参数：

pin：gpio 的 ID。范围 1~16。

返回值：

number 类型，1（高电平）或 0（低电平）

示例：

```
>print(gpio.read(0))
>1
```

gpio.write()

调用格式：

gpio.write(pin, level)

说明：

读取指定 pin 的当前输入值。

参数：

pin：gpio 的 ID。

level：高电平 gpio.HIGH；低电平 gpio.LOW

返回值：

nil

示例：

```
>gpio.write(0,gpio.HIGH)
>gpio.write(0,gpio.LOW)
```

gpio.toggle()

调用格式：

gpio.toggle(pin)

说明：

反转指定 pin 的输出值。

参数：

pin：gpio 的 ID。

返回值：

nil

示例：

```
>gpio.toggle(0)
```

TIMER 模块

函数列表

tmr.start()	启动一个定时器
tmr.stop()	停止指定定时器
tmr.stopall()	停止所有定时器
tmr.tick()	获取 MCU 自上电启动以来的运行时间，单位毫秒（ms）
tmr.delayms()	延时指定的时间，单位为 ms，延时时间内没有任何操作
tmr.wdclr()	喂狗函数，避免看门狗复位

常数

无

tmr.start()

调用格式：

tmr.start(id, interval, function)

说明：

启动一个定时器。

参数：

id: 定时器编号，范围 0~15。

interval：定时器触发周期。

function：定时器回调函数。

返回值：

nil

示例：

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

tmr.stop()

调用格式：

tmr.stop(id)

说明：

停止指定定时器。

参数：

id: 定时器编号，范围 1~16；

返回值：

nil

示例：

```
>tmr.stop(1)
```

tmr.stopall()

调用格式：

tmr.stopall()

说明：

停止所有定时器。

参数：

nil

返回值：

nil

示例：

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

tmr.tick()

调用格式：

tk=tmr.tick()

说明：

获取 MCU 自上电启动以来的运行时间，单位毫秒（ms）。

参数：

nil

返回值：

number 类型

示例：

```
>print(tmr.tick())
> 838207
```

tmr.delayms()

调用格式：

tmr.delayms(time)

说明：

延时指定的时间，单位为 ms，延时时间内没有任何操作。

参数：

time: 延迟时间，单位毫秒（ms）

返回值：

nil

示例：

>tmr.delayms(1000)

tmr.wdclr()

调用格式：

tmr.wdclr()

说明：

喂狗函数。在 Lua 程序中如果运算耗费较长时间，则有必要使用本函数主动喂狗，避免看门狗超时。

默认的看门狗超时时间为 10 秒钟。

参数：

nil

返回值：

nil

示例：

>tmr.wdclr()

WiFi 模块

函数列表

wifi.startap()	开启模块 AP 模式，自动开启 DHCP 功能
wifi.startsta()	开启模块 STA 模式，连接到指定的无线路由器
wifi.scan()	扫描热点
wifi.stop()	关闭所有的 WiFi 连接，包括 AP 模式和 STA 模式
wifi.powersave()	使能 WiFi 模块的 IEEE 低功耗节能模式
wifi.ap.getip()	获取 WiFi 模块 AP 模式下的 IP 地址
wifi.ap.getipadv()	获取 WiFi 模块 AP 模式下的详细 ip 信息，包括 DHCP 模式、ip 地址、网关、子网掩码、dns、mac、广播地址等
wifi.ap.stop()	关闭 AP 模式下所有的 WiFi 连接
wifi.sta.getip()	获取 WiFi 模块 STA 模式下的 IP 地址
wifi.sta.getipadv()	获取 WiFi 模块 STA 模式下的详细 ip 信息,包括 DHCP 模式、ip 地址、网关、子网掩码、dns、mac、广播地址等
wifi.sta.getlink()	获取 STA 模式下已经连接的 AP 信息，包括连接状态、wifi 信号强度、ssid、ap 的 bssid 地址等
wifi.sta.stop()	关闭 STA 模式下所有的 WiFi 连接，并停止 WiFi 重连

常数

无

wifi.startap()

调用格式：

wifi.startap(cfg)

wifi.startap(cfg,function)

说明：

开启模块 AP 模式，自动开启 DHCP 功能。

参数：

cfg 是 lua 表；

cfg.ssid：ap 的 ssid

cfg.pwd：密码

cfg.ip：模块 ip。参数可选，默认为 11.11.11.1

cfg.netmask：子网掩码。参数可选，默认为 255.255.255.0

cfg.gateway：网关。参数可选，默认为 11.11.11.1

cfg.dnsSrv：DNS 服务器地址，参数可选，默认为 11.11.11.1

cfg.retry_interval：重连间隔时间，单位为毫秒。参数可选，默认为 1000ms。

function:当 ap 建立完成时的回调函数。

返回值：

nil

示例：

```
>cfg={}
>cfg.ssid=""
>cfg.pwd=""
cfg.ip (optional,default:11.11.11.1)
cfg.netmask(optional,default:255.255.255.0)
cfg.gateway(optional,default:11.11.11.1)
cfg.dnsSrv(optional,default:11.11.11.1)
cfg.retry_interval(optional,default:1000ms)
wifi.startap(cfg,function(optional))
```

wifi.startsta()

调用格式：

wifi.startsta(cfg)

wifi.startsta(cfg,function)

说明：

开启模块 STA 模式，连接到指定的无线路由器。

参数：

cfg 是 lua 表；

cfg.ssid：无线路由器的 ssid

cfg.pwd：密码

cfg.dhcp：是否允许自动分配 ip。'enable'或'disable'，默认为'enable'。

cfg.ip：模块 ip。参数可选，若 cfg.dhcp 为'disable'，要求填写。

cfg.netmask：子网掩码。参参数可选，若 cfg.dhcp 为'disable'，要求填写。

cfg.gateway：网关。参数可选，若 cfg.dhcp 为'disable'，要求填写。

cfg.dnsSrv：DNS 服务器地址，参数可选，若 cfg.dhcp 为'disable'，要求填写。

cfg.retry_interval：重连间隔时间，单位为毫秒。参数可选，默认为 1000ms。

function:当模块成功连接无线了路由器时的回调函数。

返回值：

nil

示例：

```
>cfg={}
cfg.ssid=""
cfg.pwd=""
cfg.dhcp=enable/disable(default:enable)
cfg.ip (depends on dhcp)
cfg.netmask (depends on dhcp)
cfg.gateway (depends on dhcp)
cfg.dnsSrv (depends on dhcp)
cfg.retry_interval(default:1000ms)
```

wifi.startap(cfg)

wifi.scan()

调用格式：

wifi.scan(function(t))

说明：

扫描热点。

参数：

function(t):扫描完成后回调函数。t 为 lua 表，扫描结果。

扫描结果格式：

lua 表中 key 为 ssid，

value 为字符串（格式为：“mac 地址,加密方式,信号强度,信道编号”)

返回值：

nil

示例：

```
>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)
>CMCC-WEB              00:23:89:22:98:B0,90,11,OPEN
MERCURY_44B6      C0:61:18:21:44:B6,75,6,WPA2 AES
Tomato    8C:28:06:1E:01:54,100,11,WPA2 AES
ChinaNet-mALi     8C:E0:81:30:C1:95,65,10,WPA2 AES
Wireless              00:25:12:62:A6:36,57,6,OPEN
CMCC      00:23:89:22:98:B1,87,11,WPA2 AES
CMCC-FREE             00:23:89:96:02:03,60,11,OPEN
Doit      BC:D1:77:32:E7:2E,100,1,WPA2 AES
```

wifi.stop()

调用格式：

wifi.stop()

说明：

关闭所有的 WiFi 连接，包括 AP 模式和 STA 模式。

请参考：wifi.ap.stop()和 wifi.sta.stop()

参数：

nil

返回值：

nil

示例：

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

wifi.powersave()

调用格式：

wifi. powersave()

说明：

使能 WiFi 模块的 IEEE 低功耗节能模式。

参数：

nil

返回值：

nil

示例：

> wifi. powersave ()

wifi.ap.getip()

调用格式：

ip=wifi. ap.getip()

说明：

获取 WiFi 模块 AP 模式下的 IP 地址。

参数：

nil

返回值：

string：ip

示例：

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

wifi.ap.getipadv()

调用格式：

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

说明：

获取 WiFi 模块 AP 模式下的详细 ip 信息，包括 DHCP 模式、ip 地址、网关、子网掩码、dns、mac、广播地址等。

参数：

nil

返回值：

string 类型：
dhcp：“DHCP_Server”
ip：ip 地址
gw：网关地址

nm：子网掩码
dns：dns 地址
mac：MAC 地址
bip：广播地址

示例：

> dhcp,ip,gw,nm,dns,mac,bip=wifi. ap.getipadv()
>print(ip)
>11.11.11.1

wifi.ap.stop()

调用格式：

wifi.ap.stop()

说明：

关闭 AP 模式下所有的 WiFi 连接。
请参考：wifi.stop()和 wifi.sta.stop()

参数：

nil

返回值：

nil

示例：

> wifi.ap.stop()

wifi.sta.getip()

调用格式：

ip=wifi. sta.getip()

说明：

获取 WiFi 模块 STA 模式下的 IP 地址。

参数：

nil

返回值：

string：ip

示例：

> ip=wifi.sta.getip ()
>print(ip)
>192.168.1.101

wifi.sta.getipadv()

调用格式：

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

说明：

获取 WiFi 模块 STA 模式下的详细 ip 信息，包括 DHCP 模式、ip 地址、网关、子网掩码、dns、mac、广播地址等。

参数：

nil

返回值：

string 类型：

dhcp：“DHCP_Server”、“DHCP_Client”或“DHCP_Disable”

ip：ip 地址

gw：网关地址

nm：子网掩码

dns：dns 地址

mac：MAC 地址

bip：广播地址

示例：

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

>print(ip)

>192.168.1.102

wifi.sta.getlink()

调用格式：

connect,strength,ssid,bssid=wifi.sta.getlink()

说明：

获取 STA 模式下已经连接的 AP 信息，包括连接状态、wifi 信号强度、ssid、ap 的 bssid 地址等。

参数：

nil

返回值：

connect：string 类型，如果已经连接返回“connected”，否则返回“disconnected”。如果尚未连接，后面的返回值（信号强度、ssid、bssid 等）均为 nil。

strength：number 类型。信号强度。

ssid：AP 的 SSID。

Bssid：AP 的 bssid。

示例：

> wifi.sta.getlink()

wifi.sta.stop()

调用格式：

wifi.sta.stop()

说明：

关闭 STA 模式下所有的 WiFi 连接，并停止 WiFi 重连。

请参考：wifi.stop()和 wifi.sta.stop()

参数：

nil

返回值：

nil

示例：

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

Net 模块

函数列表

net.new()	新建一个 socket，指定使用的协议及其类型
net.start()	启动 socket，指定远程端口、远程 ip 或本地绑定端口
net.on()	注册回调事件函数
net.send()	向指定的 socket 发送数据
net.close()	关闭指定 socket，释放相应资源
net.getip()	获取指定客户端的 ip 地址和端口号

常数

net.TCP	使用 TCP 协议
net.UDP	使用 UDP 协议
net. SERVER	服务器
net.CLIENT	客户端

net.new()

调用格式：

skt=net.new(protocal,type)

说明：

新建一个 socket，指定使用的协议及其类型。WiFiMCU 最多可以同时建立 4 个 socket server、4 个 socket client。如果是 net.SERVER 类型，则该 socket 最多可以同时连接 5 个客户端。

参数：

protocal：net.TCP 或 net.UDP

type：net.SERVER 或 net.CLIENT

返回值：

skt：socket 值。用于标识本 sokcet。

示例：

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

net.start()

调用格式：

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

说明：

启动 socket，指定远程端口、远程 ip 或本地绑定端口。

参数：

socket：number 类型，通过 net.new()获得的 sokcet 值

port：number 类型，需要绑定的端口。

domain：string 类型，远程服务器的地址，可以是 ip 地址或者域名。

[local port]：可选参数，number 类型，当建立 TCP 或者 UDP CLIENT 的时候，用于指定本地绑定的端口。若不填写本参数，模块自动分配一个可用的本地端口。

返回值：

nil

示例：

```
>net.start(skt,9005,'11.11.11.2')
>net.start(skt2,9000,'11.11.11.2', 8000)
```

net.on()

调用格式：

net.on(socket,event,function)

说明：

定义 socket 对象的回调处理函数，根据 socket 类型不同，使用不同的事件回调。包括连接完成、dns 完成、连接断开、接收到数据、数据发送完成等。

参数：

socket：通过 net.new()获得的 sokcet 值

event：string 类型；取值包括：“connect”、“receive”、“sent”、“disconnect”、“dnsfound”。

function：回调函数，不同的 event 触发时，回调函数传递的参数不一样。

“accept”事件：仅限 TCP SERVER 类型套接字使用。当 tcp server 建立并开始监听，有客户端连接时触发。本事件的回调函数 function(sktclt, ip, port):sktclt 为客户端 socket，ip 为客户端 ip 地址，port 为客户端端口。

“connect”事件：仅限 TCP CLIENT 类型套接字使用。当 tcp client 成功连接到远程服务器时触发。本事件的回调函数 function(sktclt): sktclt 为 tcp client 自身的 socket 值。

“receive”事件：所有的套接字都可使用。当指定的 socket 上有数据时触发。本事件的回调函数 function(skt, data)，skt 为连接对象，data 为接收到的数据。

“sent”事件：所有的套接字都可使用。当指定的 socket 发送数据完成时触发。本事件的回调函数 function(skt)，skt 为发送数据的 sokcet。

“disconnect”事件：所有的套接字都可使用。当使用 socket 被迫关闭时触发。关闭的原因根据 sokcet 类型不同而不同主要包括：远程服务器断开、本地连接出错、本地发送数据出错、本地接受数据出错等。本事件的回调函数 function(skt)，skt 为发送数据的 sokcet。

“dnsfound”事件：当套接字类型为 TCP 或者 UDP 客户端时使用。当本地客户的发起连接，指定的域名解析完成时触发。本事件的回调函数 function(skt,ip)，skt 为客户端 socket 值，ip 为域名解析获得的 ip 地址。

返回值：

nil

示例：

```
>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,data) print("receive:clt:"..clt.."data:"..data) end)
>net.start(clt,9003,"11.11.11.2")
```

net.send()

调用格式：

net.send(socket, data, [function])

说明：

向指定的 skt 发送数据。并可以指定发送完成后的回调函数。

参数：

socket：通过 net.new()获得的 sokcet 值。

data：string 类型，期望发送的数据。

[function]：发送完成后的回调函数。可选参数。同 net.on()函数中注册的“sent”事件。

返回值：

nil

示例：

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

net.close()

调用格式：

net.close(socket)

说明：

关闭指定的 socket。释放其占用的资源。其后如果需要再次启用，需要重新调用 net.new()分配。

参数：

socket：通过 net.new()获得的 sokcet 值。

返回值：

nil

示例：

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

net.getip()

调用格式：

ip, port = net.getip(socket)

说明：

获取指定的客户端 sokcet 对应的 ip 地址和端口。

参数：

socket：通过 net.new()获得的 sokcet 值。Sokce 值仅限连接到本地服务器的客户端 sokcet 或者本地客户端 sokcet。否则返回 nil。

返回值：

ip：string 类型，ip 地址

port：number 类型，端口。

示例：

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

File 模块

文件系统总存储容量为 1M 字节（测试版本），地址映射到 SPI Flash 中。

函数列表

file.format()	格式化文件系统
file.open()	创建/打开文件
file.close()	关闭已经打开的文件
file.write()	向已打开的文件写入内容
file.writeline()	向已打开的文件写入内容，并在内容后自动添加换行符“\n”
file.read()	读取已经打开的文件内容
file.readline()	读取一行已经打开的文件内容
file.list()	获取文件列表名称以及每个文件大小
file.slist()	在控制台打印文件列表名称以及每个文件大小
file.remove()	删除指定文件名的文件
file.seek()	设置文件指针位置
file.flush()	清文件缓冲
file.rename()	更改文件名称
file.info()	获取文件系统存储使用情况
file.state()	获取当前打开文件的文件名称和文件大小
file.compile()	将 filename 指定的 lua 文件编译为 lc 文件，并自动命名为 filename.lc
dofile()	运行指定文件

常数

无

file.format()

调用格式：

file.format()

说明：

格式化文件系统。如果成功，在控制台打印“format done”，否则打印出错信息“format error”。

参数：

nil

返回值：

nil

示例：

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

file.open()

调用格式：

ret = file.open(filename,mode)

说明：

创建/打开文件。

参数：

filename：string 类型，文件名称

mode：文件打开类型。

“r”：只读打开

“r+”：读写方式打开

“w”：只写方式打开，如果不存在则创建文件，覆盖原文件

“w+”：读写方式打开，如果不存在则创建文件，覆盖原文件

“a”：只写方式打开，在文件末尾新增内容，如果不存在则创建文件

“a+”：读写方式打开，在文件末尾新增内容，如果不存在则创建文件

返回值：

若打开成功，返回 true。否则返回 nil

示例：

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

file.close()

调用格式：

file.close()

说明：

关闭已经打开的文件。

参数：

nil

返回值：

nil

示例：

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

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

file.write()

调用格式：

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

说明：

向已打开的文件写入内容

参数：

data：string 类型。要写入的数据

返回值：

写入成功返回 true，否则返回 nil

示例：

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

file.writeline()

调用格式：

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

说明：

向已打开的文件写入内容，并在内容后自动添加换行符“\n”

参数：

data：string 类型。要写入的数据

返回值：

写入成功返回 true，否则返回 nil

示例：

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

file.read()

调用格式：

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

说明：

读取已经打开的文件内容。有三种方式。file.read()读取并返回所有的内容。file.read(num)读取 num 个字节的内容并返回。file.read(endchar)读取所有内容，直到遇到结束字符 endchar。读取完成后，文件指针自动移动到读取内容尾部的下一个位置。

参数：

num：number 类型，指定读取字节数

endchar：读取结束字符。

返回值：

成功返回读取的数据，否则返回 nil

示例：

```
>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()

调用格式：

ret=file.readline ()

说明：

读取一行已经打开的文件内容。

参数：

nil

返回值：

成功返回读取内容，否则返回 nil

示例：

```
>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()

调用格式：

ft=file.list()

说明：

获取文件列表名称以及每个文件大小。

参数：

nil

返回值：

Lua 表；key 为文件名称，value 为文件大小（单位为字节）

示例：

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

> name:test.lua size(bytes):15

file.slist()

调用格式：

ft=file.slist()

说明：

在控制台打印文件列表名称以及每个文件大小。

参数：

nil

返回值：

nil

示例：

>file.slist()

 test.lua size:15

file.remove()

调用格式：

file.remove(filename)

说明：

删除指定文件名的文件

参数：

filename：string 类型，文件名称

返回值：

nil

示例：

>file.remove (“test.lua”)

file.seek()

调用格式：

fi = file.seek(whence, offset)

说明：

设置文件指针位置

参数：

whence：string 类型，可选择：“set”、“cur”、“end”

“set”：基值为文件开头，即 0

“cur”：基值为文件当前位置（默认值）

“end”：基值为文件末尾。

offset：number 类型，偏移量

返回值：

成功返回文件当前指针位置，否则返回 nil

示例：

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

file.flush()

调用格式：

ret = file.flush()

说明：

清文件缓冲

参数：

nil

返回值：

成功返回 true，否则返回 nil

示例：

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

file.rename()

调用格式：

ret=file.rename(oldname,newname)

说明：

更改文件名称。

参数：

oldname：原文件名。string 类型。

newname：新文件名。string 类型。

返回值：

成功返回 true，否则返回 nil

示例：

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

file.info()

调用格式：

last,used,total = file.info()

说明：

获取文件系统存储使用情况。

参数：

nil

返回值：

last：剩余字节数

used：已使用字节数

total：文件系统存储总量

示例：

```
> last,used,total = file.info()
> print(last,used,total)
888750    500         889250
```

file.state()

调用格式：

fn,sz = file.state()

说明：

获取当前打开文件的文件名称和文件大小。

参数：

nil

返回值：

fn：string 类型，文件名

sz：number 类型，文件大小

示例：

```
>file.open("testNew.lua","r")
```

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

file.compile()

调用格式：

file.compile(‘filename.lua’)

说明：

将 filename 指定的 lua 文件编译为 lc 文件，并自动命名为 filename.lc。

可以使用 dofile(“filename.lc”)执行。

参数：

filename.lua：文件名称

返回值：

nil

示例：

```
>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()

调用格式：

dofile(‘filename.lua’)

dofile(‘filename.lc’)

说明：

运行 filename 指定的 Lua 文件或者 lc 文件。

参数：

filename.lua：Lua 文件名称

filename.lc：Lua 文件名称

返回值：

nil

示例：

```
>dofile("test.lua")
Hello world!
>dofile("test.lc")
Hello world!
```

PWM 模块

函数列表

pwm.start()	启动 PWM
pwm.stop()	停止 PWM 输出

常数

null

GPIO 对应关系表

WiFiMCU 定义	其他功能
D0	
D1	PWM
D2	
D3	PWM
D4	
D5	
D6	
D7	
D8	PWM
D9	PWM
D10	PWM
D11	PWM
D12	PWM
D13	PWM
D14	PWM
D15	PWM
D16	PWM
D17	

pwm.start()

调用格式：

pwm.start(pin, freq, duty)

说明：

开启管脚 pin 的 pwm 输出功能，需要指定输出频率 freq 以及输出占空比 duty。

参数：

pin：gpio 的 ID，WiFiMCU 共支持 11 路 PWM 输出，分别是：
D1/D3/D4/D9/D10/D11/D12/D13/D14/D15/D16。
freq：输出频率，单位 Hz，范围 0<freq<10KHz
duty：输出占空比，范围 0<duty<10KHz

返回值：

nil

示例：

```
>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()

调用格式：

pwm.stop(pin)

说明：

停止 pin 管脚的 pwm 输出。

参数：

pin：gpio 的 ID，WiFiMCU 共支持 11 路 PWM 输出，分别是：
D1/D3/D4/D9/D10/D11/D12/D13/D14/D15/D16。

返回值：

nil

示例：

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

ADC 模块

函数列表

adc.read()	读取指定管脚 ADC 值

常数

null

GPIO 对应关系表

WiFiMCU 定义	其他功能
D0	
D1	ADC
D2	
D3	
D4	
D5	
D6	
D7	
D8	
D9	
D10	
D11	
D12	
D13	ADC
D14	
D15	ADC
D16	ADC
D17	ADC

adc.read()

调用格式：

result = adc.read(pin)

说明：

读取指定管脚的 ADC 值，并返回。ADC 精度为 12bit。

参数：

pin：gpio 的 ID，WiFiMCU 共支持 5 路 ADC 输入，分别是：

D1/D13/D15/D16/D17。

返回值：

result：若 ADC 读取正常，返回 0~4095。（0 对应 0V，4095 对应 3.3V）；
若出现错误，返回 nil

示例：

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

UART 模块

当前支持一个串口（id 号为 1）。GPIO 口为 D9（RX）、D10（TX）。

函数列表

uart.setup()	设置串口参数
uart.on()	设置接收到数据时的回调函数
uart.send()	通过串口发送数据

常数

null

GPIO 对应关系表

WiFiMCU 定义	其他功能
D8	RX1
D9	TX1

uart.setup()

调用格式：

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

说明：

设置串口参数。

参数：

id：number 类型，串口号，固定为 1。

baud：number 类型，波特率，例如 4800，9600,115200 等。

parity：string 类型，奇偶校验，'n'：无校验；'o'：奇校验；'e'：偶校验。

databits：string 类型，数据位数；

'5'：5 位；'6'：6 位；'7'：7 位；'8'：8 位；'9'：9 位；

stopbits：string 类型，停止位；'1'：1 位；'2'：2 位；

返回值：

nil

示例：

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

uart.on()

调用格式：

uart.on(id, event ,function(d))

说明：

设置接收到数据时的回调函数。

参数：

id：number 类型，串口号，固定为 1。

event：string 类型，回调事件，固定为'data'。

function(d)：回调函数。d 为接收到的数据。

返回值：

nil

示例：

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

uart.send()

调用格式：

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

说明：

通过串口发送数据。

参数：

id：number 类型，串口号，固定为 1。

string1：string 类型，待发送的字符串数据。

[number]：number 类型，待发送的数据。

[stringn]：第 n 个待发送字符串

返回值：

nil

示例：

> uart.send(1,'hello wifimcu')

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

>uart.send(1,string.char(0x01,0x02,0x03))

SPI 模块

待续…

I2C 模块

待续…