

NodeMCU e IoT

Jair Guilherme Certório

Luis Felipe Favaro Soares



Jair Guilherme Certório

Engenheiro Eletricista pela UEM
Desenvolvedor de sistemas embarcados
Desenvolve projetos de sistemas
microcontrolados, IoT e de controle.

Luis Felipe Fávaro Soares

Aluno do 5 ano de Engenharia Elétrica Estagiário da área de engenharia de processos Experiência com placas de desenvolvimento como Arduino e nodeMCU além de microcontroladores como o PIC18F Atualmente desenvolve projetos de automação e IoT



Laboratório comunitário, mantido por doações e equipado para projetos DIY



Eletrônica, Computação, Marcenaria, Serralheria, Comunidade e Impressão 3D.

Movimento Maker

"Esta cultura moderna tem em sua base a idéia de que pessoas comuns podem construir, consertar, modificar e fabricar os mais diversos tipos de objetos e projetos com suas próprias mãos"

Aprender vendo, exemplos e tutoriais

IoT

- Conectividade
- Dados
- Serviço



"The Internet of things (IoT) is the network of physical devices, vehicles, home appliances and other items embedded with electronics, software, sensors, actuators, and connectivity which enables these objects to connect and exchange data."

ESP-01

Módulo capaz de comunicação Wi-Fi que respondia comandos AT

Péssima documentação, só em chinês

Chamou atenção para a criadora, Espressif

ESP-12

Melhor documentação e internacionalização

Um microcontrolador bom com WiFi embutido

Levou ao projeto NodeMCU



NodeMCU

NodeMCU is an <u>eLua</u> based firmware for the <u>ESP8266 WiFi SOC from Espressif</u>. The NodeMCU *firmware* is a companion project to the popular <u>NodeMCU dev kits</u>, ready-made open source development boards with ESP8266-12E chips."

Um firmware, tipo o do Arduino, para poder programar os ESP8266 em Lua, orientado a eventos.



Documentação e exemplos em: https://nodemcu.readthedocs.io/en/master/

Especificações técnicas

- Processador 32-bit RISC @ 80 MHz;
- >50 KiB de RAM/ROM;
- 10-bit ADC, aproximação sucessiva;
- 16 pinos GPIO;
- SPI.

Especificações técnicas

- Tensão de alimentação 2.5V ~3.6V;
- Corrente média de 80 mA;
- Capacidade de corrente por GPIO de 12mA;
- Wi-Fi b/g/n (WEP or WPA/WPA2) @ 2.5GHz;
- Station/SoftAP/SoftAP+Station;
- IPv4, TCP/UDP/HTTP/FTP.

Memória

- SRAM de 64 KiB para instruções;
- SRAM de 96 KiB para dados;
- Flash externas QSPI, tipicamente, 4 MiB.



Linguagens

Lua C++ (IDE Arduino) MicroPython

Funções base

node.bootreason() node.compile() node.dsleep()

node.heap()

node.input()

node.output()

node.restart()

node.setcpufreq()

node.sleep()

node.random()

Returns the boot reason and extended reset info.

Compiles a Lua text file into Lua bytecode, and saves it as .

Enters deep sleep mode, wakes up when timed out.

Returns the current available heap size in bytes.

Submits a string to the Lua interpreter.

Redirects the Lua interpreter output to a callback function.

Restarts the chip.

Change the working CPU Frequency.

Put NodeMCU in light sleep mode to reduce current consumption.

This behaves like math.



node.chipid()

node.flashid()

node.flashsize()

node.info()

node.restore()

node.stripdebug()

node.osprint()

node.egc.setmode()

node.task.post()

Returns the ESP chip ID.

Returns the flash chip ID.

Returns the flash chip size in bytes.

Returns NodeMCU version, chipid, flashid, flash size, flash mode, flash speed.

Restores system configuration to default

Controls the amount of debug information kept during node.

Controls whether the debugging output from the Espressif SDK is printed.

Sets the Emergency Garbage Collector mode.

Enable a Lua callback or task to post another task request.





ADC
ADS1115
ADXL345
AM2320
APA102
bit
BME280
BMP085
CoAP
Cron
crypto

DHT

DS18B20 encoder end user setup file gdbstub GPIO HDC1080 HMC5883L HTTP

HX711 I²C L3G4200D MCP4725 mDNS MQTT net node 1-Wire PCM perf PWM RC (no docs) rfswitch rotary RTC fifo RTC mem RTC time Si7021 Sigma-delta SJSON SNTP Somfy SPI struct Switec TCS34725 TM1829 timer TSL2561 U8G UART

UCG websocket WiFi WPS WS2801 WS2812 XPT2046



ADC
ADS1115
ADXL345
AM2320
APA102
bit
BME280
BMP085
CoAP
Cron
crypto

DHT

DS18B20 encoder end user setup file gdbstub GPIO HDC1080 HMC5883L HTTP HX711 I²C L3G4200D MCP4725 mDNS MQTT **net node** 1-Wire PCM perf PWM RC (no docs) rfswitch rotary RTC fifo RTC mem RTC time Si7021 Sigma-delta SJSON SNTP Somfy SPI struct Switec TCS34725 TM1829 timer TSL2561 U8G UART UCG websocket **WiFi** WPS WS2801 WS2812 XPT2046

As mais usadas

- file, GPIO, net, node, timer, UART, WiFi;
- 2. HTTP;
- MQTT;
- 4. I2C;
- 5. PWM;
- 6. DHT;
- 7. ADC;
- 8. SPI;
- 9. 1-Wire:
- 10. websocket;
- 11. DS18B20.



Outras interessantes

- TLS;
- SJSON;
- crypto.



LUA

"<u>lightweight, multi-paradigm programming language</u> designed primarily for embedded systems and clients <u>cross-platform</u>, since the interpreter is written in <u>ANSI C</u>, [3] and has a relatively simple CAPI

Criada no Tecgraf da PUC-Rio"

Exemplos

```
x = -10
while x<1 do
print("Hello World!")
x = x + 1
end
```





São a única estrutura de dados da linguagem

```
t = {}
t[0] = 1
t[10] = 2
T["valor"] = {}
```

for key,value in pairs(t) do print(key,value) end

Prática 1: Lua



https://github.com/jcert/curso_node





Programar uma função que acha o n=ésimo número de Fibonacci

Prática 1: Lua

```
function fibrec(n)
    if(n==0 or n==1) then
        return 1
    else
        return fibrec(n-1)+fibrec(n-2)
    end
end
```

```
function fib(n)

aux1=0

aux2=1

while n>0 do

tmp=aux1

aux1=aux2

aux2=aux2+tmp

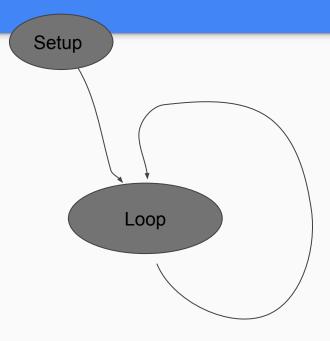
n=n-1

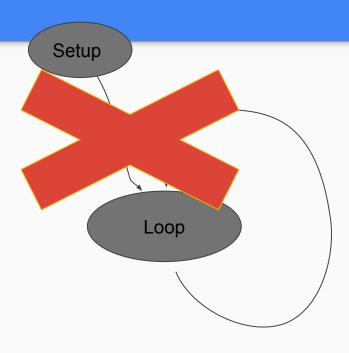
end

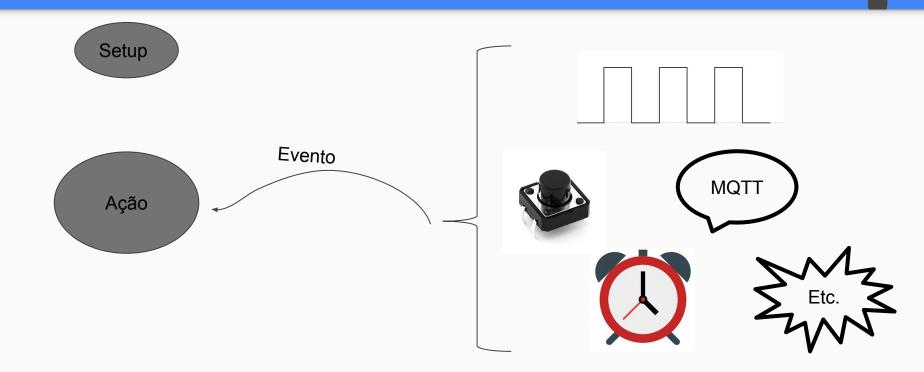
return aux2

end
```

O espírito do NodeMCU é fazer tudo de forma assíncrona, sendo um plataforma embarcada orientada a eventos. Resultando em vários bugs por concorrência e outras vantagens por cobrir vária situações sem muitos condicionais.









Pelo NodeMCU podemos enviar páginas HTML com javascript incluso



My First Heading

C'est ne pa un: Botão.

A B Enviar texto



```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<body>
<h1>My First Heading</h1>
C'est ne pa un: Botão.
<input type="button" value="A" onclick="toggleButton()">
<input type="button" value="B" onclick="b1ESP()">
<form>
<input type="text">
<button onclick="b1ESP()">Enviar texto</button>
</form>
</body>
</html>
<script>
function mudarTexto(){
 document.getElementById('texto').innerHTML = "isto não é um botão"
function toggleButton(){
 document.getElementById('texto').innerHTML = "o botão mexeu comigo"
</script>
```

Firmware

O firmware deve ser montado com as bibliotecas necessárias, pois mesmo elas sendo escritas em C ocupam bastante espaço na memória.

Existem ferramentas online para se montar o firmware

Firmware

Link que ensina a montar o firmware:

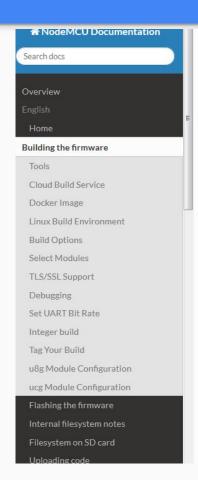
https://nodemcu.readthedocs.io/en/master/en/build/

Uma forma bem fácil é:

https://nodemcu-build.com/

Firmware





Docs » English » Building the firmware

C Edit on GitHub

There are essentially three ways to build your NodeMCU firmware: cloud build service, Docker image, dedicated Linux environment (possibly VM).

Tools

Cloud Build Service

NodeMCU "application developers" just need a ready-made firmware. There's a cloud build service with a nice UI and configuration options for them.

Docker Image

Occasional NodeMCU firmware hackers don't need full control over the complete tool chain. They might not want to setup a Linux VM with the build environment. Docker to the rescue. Give Docker NodeMCU build a try.

Linux Build Environment

NodeMCU firmware developers commit or contribute to the project on GitHub and might want to build their own full fledged build environment with the complete tool chain. There is a post in the esp8266.com Wiki that describes this.

Build Options

The following sections explain some of the options you have if you want to build your own NodeMCU firmware.

Select Modules

Firmware - nodemcu-build.com/



Select branch to build from master master

ADC 🕮	end user setup 💷	perf 🕮	Switec III
ADS1115 III	▼ file ■	PWM 🕮	TCS34725 III
ADXL345	gdbstub 🕮	RC (no docs)	■ TM1829 ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
■ AM2320 💷	☑ GPIO ■	rfswitch 🕮	▼ timer ■
■ APA102 💷		rotary 💷	☐ TSL2561 ☐ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■ ■
bit 🕮		RTC fifo III	■ U8G ■
Bloom filter	HTTP III	RTC mem	▼ UART ■
■ BME280 🕮		RTC time	UCG I
■ BME680 🕮	12C 111	Si7021 III	websocket III
■ BMP085 □■	■ L3G4200D ■	Sigma-delta	✓ WiFi
OoAP III		SJSON I	WiFi monitor
color utils 🕮	mDNS III	SNTP III	WPS III
Cron III	■ MQTT ■	Somfy III	■ WS2801 ■
crypto 💷	v net 🕮	SPI III	■ WS2812 ■
DHT 🕮	node 🕮	SQLite 3 🕮	WS2812 effects
DS18B20 🕮	1-Wire 🕮	struct 🕮	
encoder 🕮	PCM I		
Click the III to go to th	ne module documentation if yo	u're uncertain whether you	should include it or not.
	nodules will give you a basic fire firmware small. See the FAQ		t as few modules as

Firmware

- Esptool.py Python
- NodeMCU PyFlasher Python
- NodeMCU Flasher Windows

Gravar o binário no endereço 0x00000





Nodemcu-uploader

- Multiplataforma
- Feito em Python
- Envia LUA

Lendo a serial

Use o miniterm, screen ou hterm na porta serial do nodemcu

Prática 2: Upload a blink



```
pin=4
led_state=0
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.LOW)
function blink()
   if led_state==0 then
       led_state=1
       gpio.write(pin,gpio.LOW)
   else
       led_state=0
       gpio.write(pin,gpio.HIGH)
   end
end
tmr.alarm(1,1000, 1, function() blink() end )
```

Consumo de energia

Parameter	Typical	Unit
Tx 802.11b, CCK 11Mbps, P _{OUT} =+17dBm	170	mA
Tx 802.11g, OFDM 54Mbps, P _{OUT} =+15dBm	140	mA
Tx 802.11n, MCS7, P _{OUT} =+13dBm	120	mA
Rx 802.11b, 1024 bytes packet length , -80dBm	50	mA
Rx 802.11g, 1024 bytes packet length, -70dBm	56	mA
Rx 802.11n, 1024 bytes packet length, -65dBm	56	mA
Modem-Sleep	15	mA
Light-Sleep	0.5	mA
Power save mode DTIM 1	1.2	mA
Power save mode DTIM 3	0.9	mA
Deep-Sleep	10	uA
Power OFF	0.5	uA

Consumo de energia - comparativo

Parameter	Typical	Unit
Tx 802.11b, CCK 11Mbps, P _{OUT} =+17dBm	170	mA
Tx 802.11g, OFDM 54Mbps, P _{OUT} =+15dBm	140	mA
Tx 802.11n, MCS7, P _{OUT} =+13dBm	120	mA
Rx 802.11b, 1024 bytes packet length , -80dBm	50	mA
Rx 802.11g, 1024 bytes packet length, -70dBm	56	mA
Rx 802.11n, 1024 bytes packet length, -65dBm	56	mA
Modem-Sleep	15	mA
Light-Sleep	0.5	mA
Power save mode DTIM 1	1.2	mA
Power save mode DTIM 3	0.9	mA
Deep-Sleep	10	uA
Power OFF	0.5	uA

Arduino 20 mA @ 20MHz 1/0

	Entrada	Saída
Digital	DIn (9)	DOut (9)
Analógica	Aln (1)	

Pwm

I/O Digital

gpio.mode(pino, modo)

• gpio.OUTPUT, gpio.INPUT

val = gpio.read(pino)

0, 1

gpio.write(pino, valor)

0, 1

gpio.trig(pino, tipo)

"up", "down", "both", "low", "high"



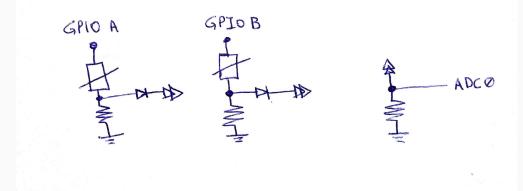
Entrada Analógica

Entrada do ADC

Só tem um :(

adc.read(0)

Dá para multiplexar assim, mas só dá pra ler um por vez

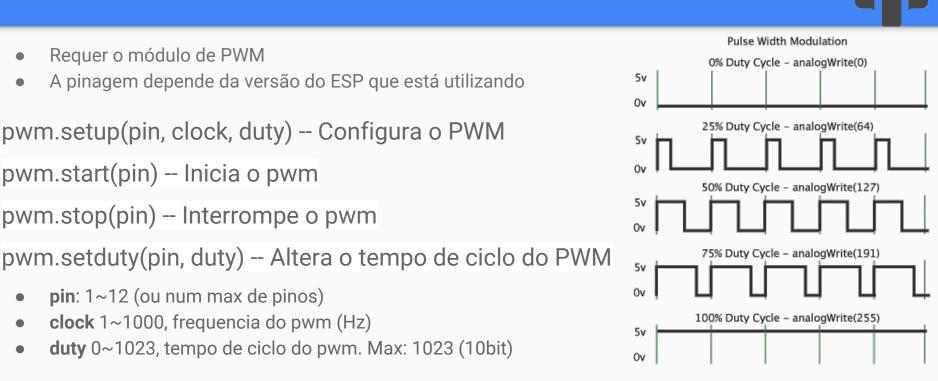


Saída PWM

- Requer o módulo de PWM
- A pinagem depende da versão do ESP que está utilizando

pwm.setup(pin, clock, duty) -- Configura o PWM pwm.start(pin) -- Inicia o pwm pwm.stop(pin) -- Interrompe o pwm

- **pin**: 1~12 (ou num max de pinos)
- **clock** 1~1000, frequencia do pwm (Hz)
- duty 0~1023, tempo de ciclo do pwm. Max: 1023 (10bit)



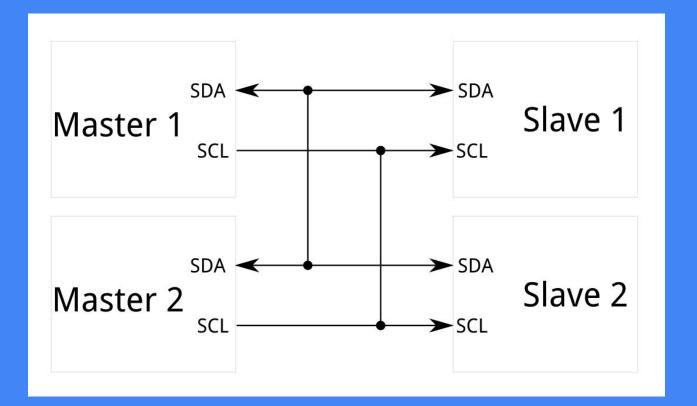
Comunicação

- UART
- 12C
- WiFi (IPv4 2,4GHz)

Comunicação

- UART Serial
- I2C
- WiFi (IPv4 2,4GHz)

12C



12C



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

i2c.start(id)

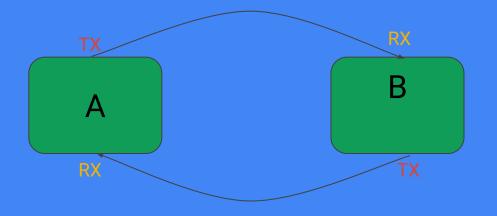
i2c.stop()

i2c.address(id, device_addr, direction)

i2c.read(id, len)

i2c.write()

UART





UART

Comunicação bidirecional com um outro dispositivo (ou unidirecional com vários)

UART

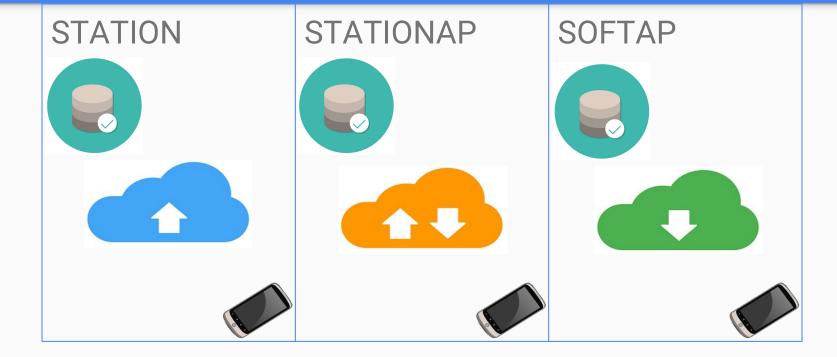
```
uart.on("data",[number/end_char], [function], [run_input])
uart.setup(id, baud, databits, parity, stopbits[, echo])
baud databits parity stopbits = uart.getconfig(id)
uart.write(id, data1)
```

STA - Modo station

AP - Modo access point

Protocolos: IPv4 - TCP, UDP - HTTP, MQTT, IMAP, TLS

Modos do WiFi



wifi.getchannel() Gets the current WiFi channel.
wifi.getcountry() Get the current country info.
wifi.getdefaultmode() Gets default WiFi operation mode.
wifi.getmode() Gets WiFi operation mode.
wifi.getphymode() Gets WiFi physical mode.
wifi.nullmodesleep() Configures whether or not WiFi automatically goes to sleep in NULL_MODE.
wifi.resume() Wake up WiFi from suspended state or cancel pending wifi suspension.
wifi.setcountry() Set the current country info.
wifi.setmode() Configures the WiFi mode to use.
wifi.setphymode() Sets WiFi physical mode.

wifi.setmaxtxpower() Sets WiFi maximum TX power.
wifi.startsmart() Starts to auto configuration, if success
set up SSID and password automatically.
wifi.stopsmart() Stops the smart configuring process.
wifi.suspend() Suspend Wifi to reduce current
consumption.

<u>wifi.sta.autoconnect()</u> Auto connects to AP in station mode.

wifi.sta.changeap() Select Access Point from list returned by wifi.

wifi.sta.clearconfig() Clears the currently saved WiFi station configuration, erasing it from the flash.
wifi.sta.config() Sets the WiFi station configuration.
wifi.sta.connect() Connects to the configured AP in station mode.

<u>wifi.sta.disconnect()</u> Disconnects from AP in station mode.

wifi.sta.getmac() Gets MAC address in station mode. wifi.sta.getrssi() Get RSSI(Received Signal Strength Indicator) of the Access Point which ESP8266 station connected to.

wifi.sta.setaplimit() Set Maximum number of Access Points to store in flash.

<u>wifi.sta.sethostname()</u> Sets station hostname. <u>wifi.sta.setip()</u> Sets IP address, netmask, gateway address in station mode.

<u>wifi.sta.setmac()</u> Sets MAC address in station mode. <u>wifi.sta.sleeptype()</u> Configures the WiFi modem sleep type to be used while station is connected to an Access Point.

wifi.sta.status() Gets the current status in station mode. wifi.ap.config() Sets SSID and password in AP mode.

wifi.sta.getap() Scans AP list as a Lua table into callback function.

wifi.sta.getapindex() Get index of current Access Point stored in AP cache.

wifi.sta.getapinfo() Get information of APs cached by ESP8266 station.

<u>wifi.sta.getbroadcast()</u> Gets the broadcast address in station mode.

wifi.sta.getconfig() Gets the WiFi station configuration. wifi.sta.getdefaultconfig() Gets the default WiFi station configuration stored in flash.

<u>wifi.sta.gethostname()</u> Gets current station hostname. <u>wifi.sta.getip()</u> Gets IP address, netmask, and gateway address in station mode.

wifi.ap.deauth() Deauths (forcibly removes) a client from the ESP access point by sending a corresponding IEEE802.

<u>wifi.ap.getbroadcast()</u> Gets broadcast address in AP mode.

wifi.ap.getclient() Gets table of clients connected to device in AP mode.

<u>wifi.ap.getconfig()</u> Gets the current SoftAP configuration.

wifi.ap.getdefaultconfig() Gets the default SoftAP configuration stored in flash.

wifi.ap.getip() Gets IP address, netmask and gateway in AP mode.

<u>wifi.ap.getmac()</u> Gets MAC address in AP mode. <u>wifi.ap.setip()</u> Sets IP address, netmask and gateway address in AP mode. wifi.ap.setmac() Sets MAC address in AP mode.
wifi.ap.dhcp.config() Configure the dhcp service.
wifi.ap.dhcp.start() Starts the DHCP service.
wifi.ap.dhcp.stop() Stops the DHCP service.
wifi.eventmon.register() Register/unregister callbacks
for WiFi event monitor

<u>wifi.eventmon.unregister()</u> Unregister callbacks for WiFi event monitor.

<u>Wifi.eventmon.reason</u> Table containing disconnect reasons.



wifi.getmode() Gets WiFi operation mode. wifi.setmode() Configures the WiFi mode to use. wifi.suspend() Suspend Wifi to reduce current consumption.

<u>wifi.resume()</u> Wake up WiFi from suspended state or cancel pending wifi suspension.

<u>wifi.sta.config()</u> Sets the WiFi station configuration. <u>wifi.sta.connect()</u> Connects to the configured AP in station mode.

wifi.sta.disconnect() Disconnects from AP in station mode.

wifi.sta.sethostname() Sets station hostname.
wifi.ap.config() Sets SSID and password in AP mode.
wifi.sta.getbroadcast() Gets the broadcast address in station mode.

<u>Wifi.eventmon.reason</u> Table containing disconnect reasons.

Envios

TCP

- Garante o envio de pacotes;
- Ordem dos pacotes é determinada;
- Mantém conexão ativa.

UDP

- Não garante o envio de pacotes;
- Ordem dos pacotes não é determinada;
- Não cria conexão.

HTTP



- Pode usar o módulo de http ou,
- escrever na mão os headers e mandar pelo módulo net





GET

Dado vai na URL, dentro do header

GET /?say=Hi&to=Mom HTTP/1.1 Host: foo.com

POST

- Dado vai no corpo da requisição
- Cabe mais dados

POST / HTTP/1.1

Host: foo.com

Content-Type: application/x-www-form-urlencoded

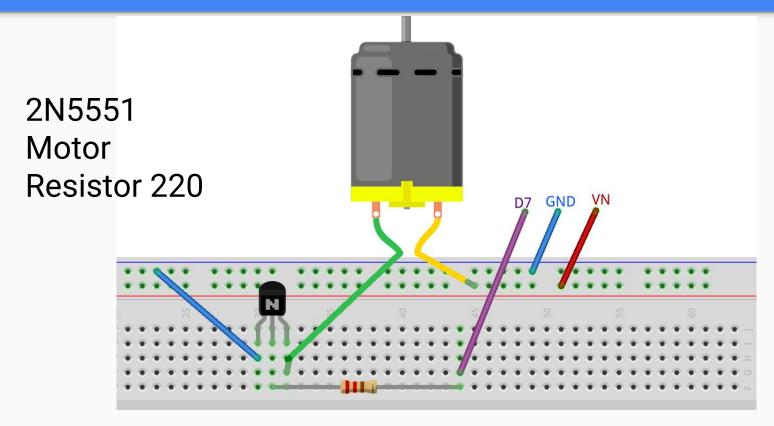
Content-Length: 13

say=Hi&to=Mom

Prática 3: WiFi - HTML

Diagrama feito no Fritzing







Criar um template, dar pra galera e fazer eles controlarem a velocidade de um motor ou algo assim

```
station_cfg={}
station_cfg.ssid="certorio"
station_cfg.pwd= "jair"
station_cfg.auto=true
set = wifi.sta.config(station_cfg)
wifi.sta.connect()
- enquanto não tiver a biblioteca de pwm
pwm={['setup']=function(x,y,z) return x end,['start']=function(x) return x
end,['setduty']=function(x,y) return x end}
pwm.setup(p,500,0)
pwm.start(p)
simpleHTMLpage = [[
<head></head>
<body>
my text is simple
</body>
function onReceive(sck, data)
       and and aimple LITM negal
```

File

```
file.chdir()
file.exists()
file.format()
file.fscfg ()
file.fsinfo()
file.list()
file.mount()
file.on()
file.open()
file.remove()
file.rename()
file.stat()
```

```
x = file.open('dados.txt','wb+')
x:close()
x:flush()
x:read()
x:readline()
x:seek()
x:write()
x:writeline()
```

>4MB

A flash do ESP proporciona muito mais espaço que a EEPROM do Arduino

File

- Possui um sistema de arquivos
- Não precisa ser escrito direto na FLASH (sem formatação)
- Pode ser feito o download com a nodemcu-uploader e outras ferramentas
- A FLASH aguenta cerca de 10000 escritas

Prática 4: Arquivos e dados



A partir do template, recebam os dados do post html com texto e armazenem em um arquivo, retornar o texto armazenado para as conexões.

Os dados em arquivos não são perdidos quando você desliga o ESP!

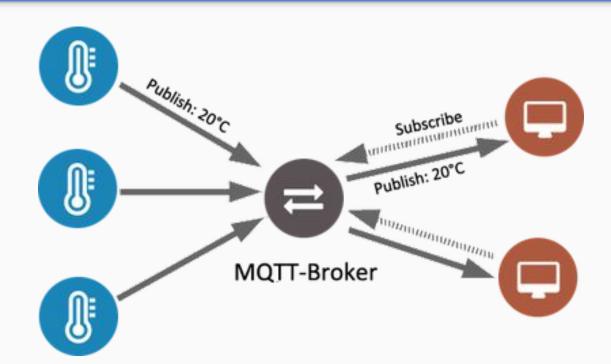
```
station_cfg={}
station_cfg.ssid="certorio"
station_cfg.pwd= "jair"
station_cfg.auto=true
set = wifi.sta.config(station_cfg)
wifi.sta.connect()
- enquanto não tiver a biblioteca de pwm
pwm={['setup']=function(x,y,z) return x end,['start']=function(x) return x
end,['setduty']=function(x,y) return x end}
pwm.setup(p,500,0)
pwm.start(p)
simpleHTMLpage = {[[
<head></head>
<body>11.
[[</body>]]}
function onReceive(sck, data)
         sck:send(simpleHTMLpage)
         aux = string.gmatch(data,"(%w+)=(%w+)&*")
```

for ky in our do

MQTT

- Sigla para Message Queuing Telemetry Transport, é um protocolo utilizado para o recebimento e envio de mensagens no sistema publish/subscribe;
- Desenvolvido para minimizar o uso de largura de banda e recursos do dispositivo além assegurar uma confiabilidade na entrega dos dados (QoS);
- Já existem implementações em linguagens, como:
 - Lua, Ruby on Rails, Node.js C, C#, C++, Python, Java, JavaScript, Go, etc.
- Além de empresas como:
 - o Adafruit, Arduino, Eclipse, IBM, Amazon, etc.
- Utilizado principalmente em aplicações onde é necessário uma comunicação M2M (Machine to Machine) e em situações onde existam poucos recursos de hardware e sistemas dependentes de baterias;

MQTT





MQTT - Funções

- mqtt.Client() Creates a MQTT client.
 - Sintaxe: m = mqtt.Client("SeuNome", 120, "user", "password")
- mqtt.client:connect() Connects to the broker specified by the given host, port, and secure options.
 - Sintaxe: m:connect("192.168.25.96", 1883, 0, function(client)
 print("connected") end, function(client, reason) print("failed reason: " ...
 reason) end

MQTT - Funções

- mqtt.client:on() Registers a callback function for an event.
 - Sintaxe: m:on("message", function(client, topic, data)
- mqtt.client:publish() Publishes a message.
 - Sintaxe: m:publish("/topic", msg, 0, 0, function(client) print("sent") end)
- mqtt.client:subscribe()
 Subscribes to one or several topics.
 - Sintaxe: m:subscribe("/inData", 0, function(client) print("success") end)
- mqtt.client:unsubscribe()
 - Sintaxe: m:unsubscribe("/inData", 0, function(client) print("success") end)

MQTT - Funções

- mqtt.client:close() Closes connection to the broker.
 - Sintaxe: m:close()
- mqtt.client:lwt() Setup Last Will and Testament (optional).
 - m:lwt("/lwt", "offline", 0, 0)

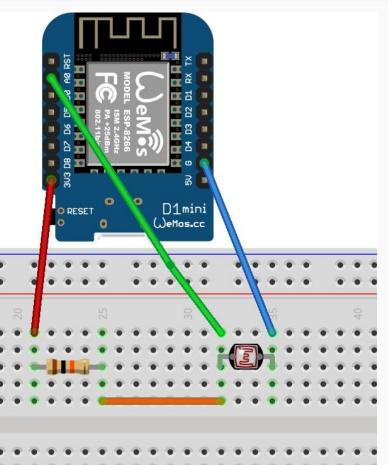
Prática 5: MQTT - sensor

Diagrama no Fritzing:



1 Resistor 10k Ohms

1 LDR



Prática 5: MQTT - sensor

Objetivo:

- 1. Leia o valor em ADO do NodeMCU
- Crie um Cliente MQTT e conecte-se ao Servidor (Broker) (192.168.25.96)
- 3. Envie uma mensagem com o valor lido para o tópico: "SECOMP/SeuNome"
- 4. Receba o valor de um colega ao lado e imprima na tela

Funções recomendadas:

- Funções de conexão Wifi
- m = mqtt.Client("SeuNome", 120, "user", "password")
- m:on("message", Function)
- m:connect("192.168.25.96", 1883, 0, function suc, function fail)
- m:publish("/topic", msg, 0, 0, function(client) print("sent") end)

Prática 6: WiFi - AP



Criar um wifi ap que serve uma página de HTML

Contagem de Tempo

Duas bibliotecas principais:

- Timer (Tmr)
- RTC Time

Permitem

Timers

Requer o módulo Timer (Tmr)

Implementa Timers simples via software $(0 \sim 6)$

Utilizado em tarefas que ocorrem em um períodos determinados de tempo

Se a tarefa necessitar de uma contagem de tempo mais precisa, o módulo

rtctime é mais indicado.

Timers - funções

tmr.alarm() This is a convenience function combining tmr.

tmr.create() Creates a dynamic timer object.

tmr.delay() Busyloops the processor for a specified number of microseconds.

tmr.interval() Changes a registered timer's expiry interval

tmr.now() Returns the system counter, which counts in microseconds.

tmr.register() Configures a timer and registers the callback function to call on expiry.

tmr.resume() Resume an individual timer.

tmr.resume_all() Resume all timers.

tmr.softwd() Provides a simple software watchdog, which needs to be re-armed or disabled before it expires, or the system will be restarted.

tmr.start() Starts or restarts a previously configured timer.

tmr.state() Checks the state of a timer. tmr.stop() Stops a running timer, but does not unregister

it.

tmr.suspend() Suspend an armed timer.

tmr.suspend_all() Suspend all currently armed timers.

tmr.time() Returns the system uptime, in seconds. tmr.unregister() Stops the timer (if running) and

unregisters the associated callback.

tmr.wdclr() Feed the system watchdog.

RTC Time

Suporta contagem de Tempo avançada incluindo a contagem de tempo através de modos de baixo consumo (Deep sleep)

Compatível com o NTP (num. de segundos depois de 01/01/1970)

Acurácia menor que 1ms

RTC Time - Funções

rtctime.dsleep(): Puts the ESP8266 into deep sleep mode, like node.

rtctime.epoch2cal(): Converts a Unix timestamp to calendar format.

rtctime.get(): Returns the current time.

rtctime.set(): Sets the rtctime to a given timestamp in the Unix epoch.

rtctime.adjust_delta(): This takes a time interval in 'system clock

microseconds' based on the timestamps returned by tmr. This function is

useful in some precision timing applications.

Exemplo: Timers & RTC

```
local start = tmr.now()
-- do something
local end = tmr.now()
print ('Duration', rtctime.adjust_delta(end -
start))
```

Prática 7: Timers & RTC



Objetivos:

- 1. Conecte-se ao servidor MQTT (Só precisa conectar uma vez)
- 2. A cada 5 Seg exibir "Cuco"
- 3. A cada 10 seg enviar o valor em AD0 via MQTT
- 4. Ajustar o relógio: Utilize https://www.epochconverter.com/ ou: 1536785107
- 5. A cada 30 seg escrever na serial o dia/mes/ano hora:min:seg





Exemplo de uso das funções

rtctime.set(1536785107, 0)

tm = rtctime.epoch2cal(rtctime.get())

print(string.format("%04d/%02d/%02d %02d:%02d:%02d", tm["year"], tm["mon"], tm["day"], tm["hour"], tm["min"], tm["sec"]))