

ESP8266 — nodeMCU

Noemi Rodriguez

Departamento de Informática
PUC-Rio

2019



microcontrolador ESP8266 - configuração D1 mini

- ▶ CPU de 32 bits
 - ▶ 64 KB RAM p/ instruções, 96 KB RAM p/ dados
 - ▶ 4MB flash
 - ▶ módulo wifi
-
- ▶ pode ser programado em Lua
 - ▶ firmware `nodeMCU`

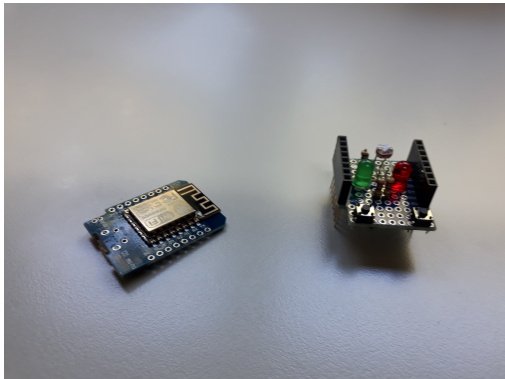
microcontrolador ESP8266 - configuração D1 mini

- ▶ CPU de 32 bits
- ▶ 64 KB RAM p/ instruções, 96 KB RAM p/ dados
- ▶ 4MB flash
- ▶ comparação: arduino uno

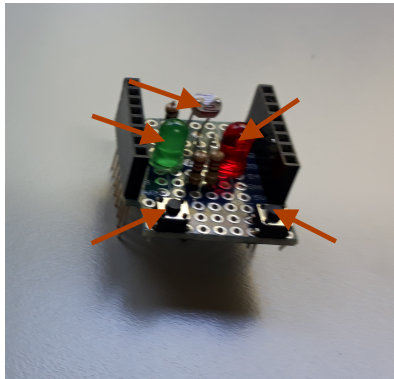
- processador ATmega328P
 - 8 bits
- memória flash: 32KB
 - programa (sketch)
- memória SRAM: 2K
 - dados
- clock: 16MHz
- EEPROM: 1KB
 - armazenamento



material que vamos usar



o que tem na placa

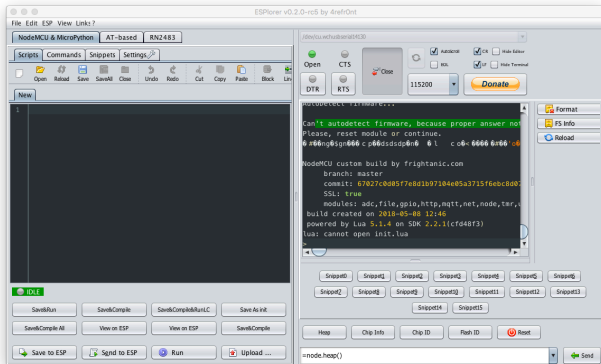


Programando o nodeMCU — ambiente

- ▶ documentação: <http://nodemcu.readthedocs.io>
- ▶ módulo de desenvolvimento:
<http://esp8266.ru/esplorer-latest/?f=ESPplorer.zip>
- ▶ controla carga e execução de programas no nodemcu



Programando o nodeMCU — ambiente



Programando o nodeMCU — ambiente

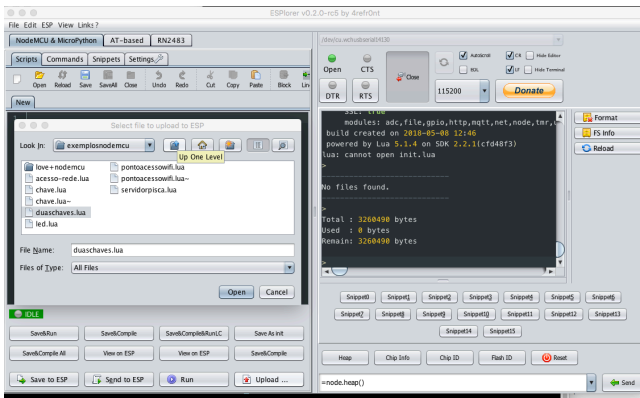
- ▶ documentação: <http://nodemcu.readthedocs.io>
- ▶ IDE para desenvolvimento: controla carga e execução de programas no nodemcu

arquivo `init.lua`

EVITAR usar este nome pois difícil de interromper em caso de bugs



Programando o nodeMCU — ambiente



The screenshot displays the ESPLoader v0.2.0-rc5 interface. The top menu bar includes File, Edit, ESP, View, and Links. The main window is titled "NodeMCU & MicroPython AT-based RN2483". Below the menu is a toolbar with icons for Open, Reload, Save, Save All, Close, Undo, Redo, Cut, Copy, Paste, and Block. A "New" button is also present.

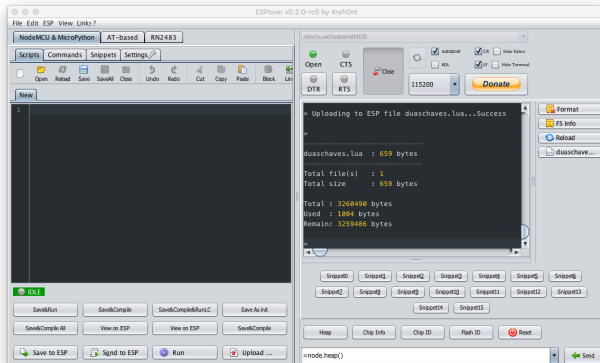
A file selection dialog is open, showing a directory structure under "exemplisnodemcu". The files listed include: love+nodemcu, acesso-rede.lua, chave.lua, duaschaves.lua, led.lua, pontoacessowifi.lua, pontoacessowifi.lua~, and servidorpisca.lua. The "File Name" field is set to "duaschaves.lua" and "Files of Type" is set to "All Files". Buttons for "Open" and "Cancel" are at the bottom.

The main terminal window shows the following output:

```
./bin/cu.wchusbserial@04130
Open CTS Close
DTR RTS 115200 Donate
modules: adc,file,gpio,http,nqmt,net,node,tmr,
build created on 2018-05-08 12:46
powered by Lua 5.1.4 on SDK 2.2.1(cf4df3)
lua: cannot open init.lua
No files found.
Total : 3260490 bytes
Used : 0 bytes
Remain: 3260490 bytes
```

Below the terminal are several buttons for Snippet0 through Snippet15, a "Heap" button, "Chip Info", "Chip ID", "Flash ID", and a "Reset" button. At the bottom, there is a text input field containing "=node.heap()" and a "Send" button.

Programando o nodeMCU — ambiente



Programando o nodeMCU — entrada e saída

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2

-- pinos de leds são de saída
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)

-- apaga os leds
gpio.write(led1, gpio.LOW);
gpio.write(led2, gpio.LOW);
```



Programando o nodeMCU — modelo de programação

- ▶ programador define *callbacks* para eventos
 - ▶ como *keypressed* em löve mas programador diz qual função deve ser chamada



Programando o nodeMCU — acendendo e apagando led

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2
local apagado = true
...

-- coloca os pinos dos leds em modo de saida
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
-- apaga os leds
gpio.write(led1, gpio.LOW)
gpio.write(led2, gpio.LOW)
-- coloca um sinal estavel nas chaves 1 e 2
gpio.mode(sw1,gpio.INT,gpio.PULLUP)
gpio.mode(sw2,gpio.INT,gpio.PULLUP)
```



Programando o nodeMCU — acendendo e apagando led

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2
local apagado = true
...

-- coloca os pinos dos leds em modo de saida
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
-- apaga os leds
gpio.write(led1, gpio.LOW)
gpio.write(led2, gpio.LOW)
-- coloca um sinal estavel nas chaves 1 e 2
gpio.mode(sw1,gpio.INT,gpio.PULLUP)
gpio.mode(sw2,gpio.INT,gpio.PULLUP)

-- cadastra a funcao cbchave para ser chamada se
-- a chave 1 for apertada
gpio.trig(sw1, "down", cbchave)
```



nodeMCU — callback para interrupção

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2

local apagado = true

local function cbchave (level, timestamp)
  apagado = not apagado -- muda estado global
  if apagado then
    gpio.write(led1, gpio.LOW);
  else
    gpio.write(led1, gpio.HIGH);
  end
end

-- coloca os pinos dos leds em modo de saida
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
-- apaga os leds
gpio.write(led1, gpio.LOW)
gpio.write(led2, gpio.LOW)
-- coloca um sinal estavel nas chaves 1 e 2
gpio.mode(sw1,gpio.INT,gpio.PULLUP)
gpio.mode(sw2,gpio.INT,gpio.PULLUP)
-- cadastra a funcao cbchave para ser chamada se
-- a chave 1 for apertada
gpio.trig(sw1, "down", cbchave)
```



callback para interrupção — mais “cara” de Lua

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2

local criacontroleled = function (sw,led)
  local apagado = true
  local function cbchave (level, timestamp)
    apagado = not apagado -- muda estado global
    if apagado then
      gpio.write(led, gpio.LOW);
    else
      gpio.write(led, gpio.HIGH);
    end
  end
  -- coloca pino do led em modo de saida
  gpio.mode(led, gpio.OUTPUT)
  -- apaga o led
  gpio.write(led, gpio.LOW)
  return cbchave
end

-- coloca um sinal estavel nas chaves
gpio.mode(sw1,gpio.INT,gpio.PULLUP)
gpio.mode(sw2,gpio.INT,gpio.PULLUP)
-- cadastra as funcoes de callback para cada chave
gpio.trig(sw1, "down", criacontroleled(sw1, led1))
gpio.trig(sw2, "down", criacontroleled(sw2, led2))
```



nodeMCU — Interface timer

- ▶ exemplo de objeto

interface

```
t = tmr.create()  
t:register()  
t:start()  
t:stop()  
t:unregister()  
...
```



nodeMCU — pisca-pisca

```
local led1 = 3
local led2 = 6

local apagado = true

local function piscapisca (t)
    ...
end

-- inicia estado dos leds
...

local mytimer = tmr.create()
mytimer:register(500, tmr.ALARM_AUTO, piscapisca)
mytimer:start()
```



nodeMCU — pisca-pisca

```
local led1 = 3
local led2 = 6

local function disparapiscapisca (led, tempo)
  local apagado = true
  local function piscapisca(timer)
    if apagado then
      gpio.write(led, gpio.HIGH);
    else
      gpio.write(led, gpio.LOW);
    end
    apagado = not apagado
  end
  -- coloca o pino dos leds em modo de saida
  gpio.mode(led, gpio.OUTPUT)
  -- apaga o led
  gpio.write(led, gpio.LOW);
  local mytimer = tmr.create()
  mytimer:register(tempo, tmr.ALARM_AUTO, piscapisca)
  mytimer:start()
end

disparapiscapisca (led1, 500)
disparapiscapisca (led2, 750)
```



revido exemplo chave

- ▶ comportamento do programa não é muito estável
- ▶ chave algumas vezes não funciona corretamente

de novo bouncing!

- ▶ sinal flutua quando circuito se fecha
- ▶ função de *callback* é chamada várias vezes

revido exemplo chave e debouncing

```
local led1 = 3
local led2 = 6
local sw1 = 1
local sw2 = 2

local apagado = true

-- só queremos chamar se já decorreu
-- um tempo mínimo desde a última chamada
local function cbchave ()
    apagado = not apagado -- muda estado global
    if apagado then gpio.write(led1, gpio.LOW);
    else gpio.write(led1, gpio.HIGH);
    end
end
end
```



debouncing com verificação de tempo decorrido

```
local ultimavez = 0
local tolerancia = 300000 -- microsegundos
...
cbchave = function ()
  apagado = not apagado
  if apagado then gpio.write(led1, gpio.LOW);
  else gpio.write(led1, gpio.HIGH);
  end
end
cbint = function (nivel, quando)
-- quando é um contador em microsegundos
  if quando - ultimavez > tolerancia then
    ultimavez = quando
    cbchave()
  end
end
...
gpio.trig(sw1, "down", cbint)
```



debouncing: outra maneira

usada por alguns no projeto arduino

- ▶ podemos mudar o *estado* do programa!
- ▶ ... e deixá-lo não reagir ao botão por algum tempo

chaves: *debouncing com timer* (1)

```
...
reestabelece = function ()
  gpio.trig(sw1, "down", cbint)
end
cbint = function ()
  -- suspende tratamento de sinais da chave por um tempo
  gpio.trig(sw1)
  alarme:register(tolerancia, tmr.ALARM_SINGLE,
                 reestabelece)

  alarme:start()
  -- trata chave
  cbchave()
end
cbchave = function ()
  -- mesma de antes
end
...
gpio.trig(sw1, "down", cbint)
alarme = tmr.create()
```

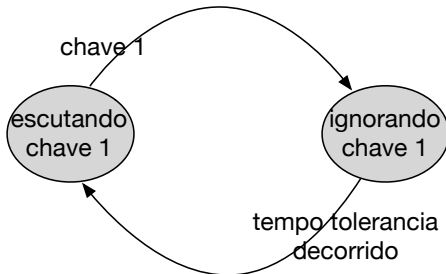


chaves: *debouncing com timer* (2)

```
local tolerancia = 300 -- !!!! agora em milisegundos!!
local apagado = true
local alarme, reestabelece, cbchave, cbint
reestabelece = function ()
  gpio.trig(sw1, "down", cbint)
end
cbint = function ()
  -- suspende tratamento de sinais da chave por um tempo
  gpio.trig(sw1)
  alarme:register(tolerancia, tmr.ALARM_SINGLE, reestabelece)
  alarme:start()
  -- trata chave:
  cbchave()
end
cbchave = function ()
  -- mesma de antes
end
...
gpio.trig(sw1, "down", cbint)
alarme = tmr.create()
```



Máquina de estados



Exercício

- ▶ baixar o ESPlorer
(<http://esp8266.ru/esplorer-latest/?f=ESPlorer.zip> – já disponível nas máquinas do labgrad)
- ▶ baixar o programa <http://www.inf.puc-rio.br/~noemi/sr-19/code/nodemcu/sogeraseq.lua> e criar um joguinho que espera que o usuário repita a sequência mostrada pelo nodemcu usando a chave 1 para o led vermelho e a chave 2 para o led verde:
 - ▶ Quando usuário pressiona a chave 1, o nodemcu mostra uma sequência de 5 piscadas verdes e vermelhas.
 - ▶ Em seguida, o usuário pressiona as chaves repetindo a sequência (chave 1 – led vermelho, chave 2 – led verde).
 - ▶ Quando o usuário termina uma sequência de 5 pressionamentos, o nodemcu acende luz verde se a sequência foi repetida corretamente e a luz vermelha se houve erro. A partir daí o programa não deve reagir a pressionamentos de chave.

