1ºEdição

PROGRAME A RASPBERRY PI COM PYTHON

Setembro 2019
www.embarcados.com.br

Ebook Programe Raspeberry Pi com Python

Olá,

Obrigado por baixar o nosso ebook: **Programe Raspeberry Pi com Python**. Esse ebook traz uma coleção de textos já publicados no Embarcados sobre a plataforma Raspeberry.

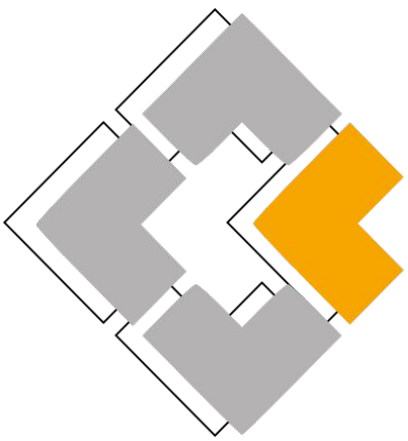
Fizemos um compilado de textos que consideramos importantes para os primeiros passos para programar a Raspberry Pi com Python.Esperamos que você aproveite esse material e lhe ajude em sua jornada.

Agradecemos a todos os autores pelo excelente conteúdo compartilhado.

Continuamos com a missão de publicar textos novos diariamente no site.

Um grande abraço.

Equipe Embarcados.



Raspberry PI GPIO output com Python

EMBARCADOS

Introdução

Vimos no artigo 'Python + Arduino – Comunicação Serial' como é simples e prático iniciar uma comunicação serial passo-a-passo. Mas Python possui um leque maior de recursos para os sistemas embarcados como, por exemplo, interagir com um GPIO (*General Purpose Input Output*), ou melhor, os pinos programáveis de entrada e saída da placa Raspberry Pi.

Se não conhece a Raspberry PI recomendo ler o artigo Raspberry Pi, do Thiago Lima. Se já conhece essa board mas não é muito familiarizado com Linux, Vinicius Maciel escreveu o artigo Raspberry Pi e o Linux que irá dar uma excelente base sobre os dois assuntos.

Configuração do ambiente host

- Board: Raspberry PI B
- Distribuição: Raspbian - Debian Wheezy (2014-01-07)
- Kernel 3.12.22
- Python 2.7
- RPi.GPIO 0.5.6

```
cleiton@vm03 ~ $  pip search RPi.GPIO
RPIO-    Advanced GPIO for the Raspsberry Pi. Extends RPi.GPIO with PWM, GPIO interrupts, TCP
socket interrupts, command line tools and more
RPi.GPIO-    A module to control Rapsberry Pi GPIO channels
...
```

Pesquisar nos repositórios de pacotes do Python qualquer pacote que no nome ou descrição contenha a expressão informada, no caso RPi.GPIO. Neste caso instalaremos a segunda opção RPi.GPIO, que neste primeiro contato é a que vamos trabalhar, para instalar o pacote RPi.GPIO execute o comando abaixo:

```
cleiton@vm03 ~ $  pip install RPi.GPIO
Downloading/unpacking RPi.GPIO
Downloading RPi.GPIO-0.5.6.tar.gz
Running setup.py (path:/home/cleiton/projetos/python/rpi/proj_rpi_gpio/build/RPi.GPIO/setup.py)
egg_info for package RPi.GPIO
Installing collected packages: RPi.GPIO
Running setup.py install for RPi.GPIO
building 'RPi.GPIO' extensionx86_64-linux-gnu-gcc -pthread -fno-strict-aliasing -DNDEBUG -g
-fwrapv -O2 -Wall -Wstrict-prototypes -fPIC -I/usr/include/python2.7 -c source/py_gpio.c -o
build/temp.linux-x86_64-2.7/source/py_gpio.o
...
Successfully installed RPi.GPIO
Cleaning up...
```

Caso durante a instalação do RPi.GPIO um erro como “fatal error: Python.h: No such fileor directory” surgir, não se assuste, é ausência dos headers, algumas bibliotecas necessárias e ferramentas do Python para a construção do pacotes, e que é facilmente resolvido instalando o python-dev.

```
cleiton@vm03 ~ $  apt-get install python-dev
```

Esquema da ligação

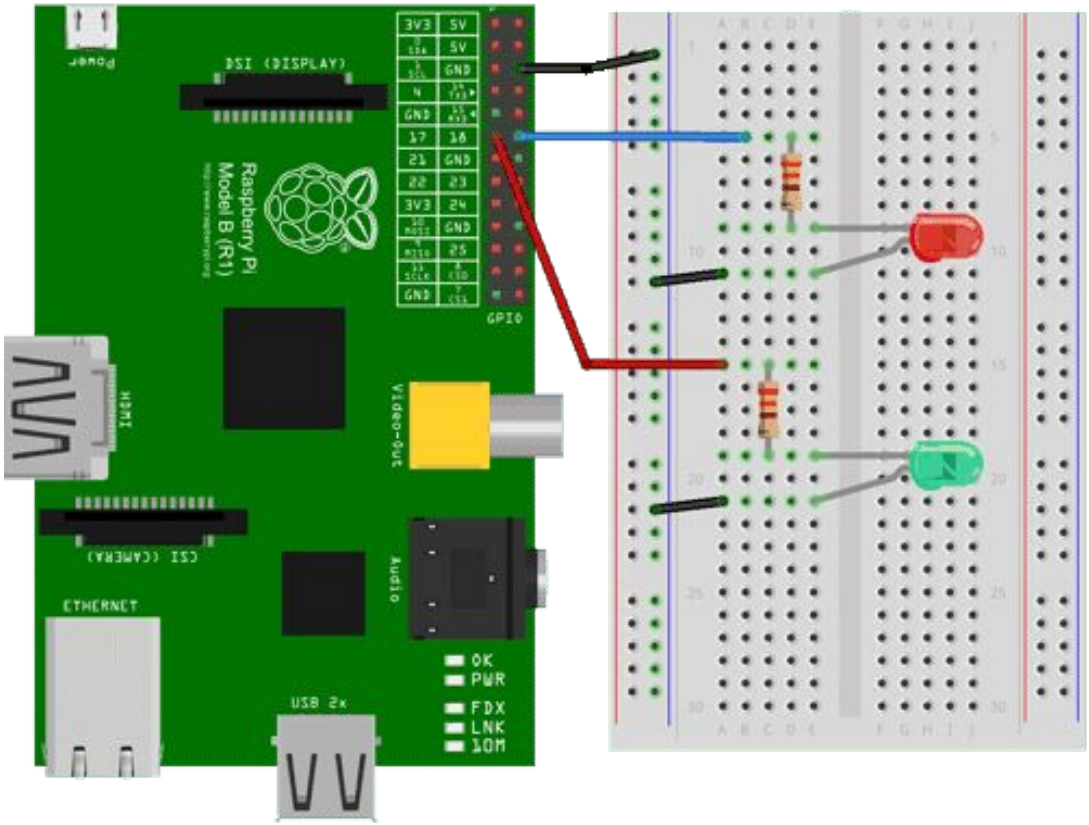


Figura 01 – Ligações Raspberry Pi GPIO

Ligação simples como podemos ver na Figura01, onde ligamos o led vermelho ao GND e usando um resistor conectamos ao pino 12 (BOARD) [GPIO18 (BCM)] e o led verde ao GND e pino 11 (BOARD) [GPIO17 (BCM)], logo mais será compreendido os termos BOARD e BCM.

Lembre-se que o GPIO do Raspberry PI utiliza níveis de tensão 3.3V e deve-se utilizar o resistor correto no LED para não causar maiores danos, o que estou usando é um de 330Ohms, visto que era o que tinha disponível e atende a necessidade para a prática deste artigo (limita a corrente em 10 mA.).

O Python por si só não sabe interagir com o GPIO (a menos que você utilize /sys/class/gpio/..., mas não é o foco neste artigo), então utilizaremos um módulo que já possui funções prontas para realizar todas as configurações e interagir com os leds. O mais interessante é que foi possível usar o Python com a filosofia Arduino e criar algo fácil de trabalhar.

O módulo utilizado é o RPi.GPIO, possui licença MIT, e em poucas linhas você configura o modo, pino, direção e já começa a funcionar, é um modulo em constante atualização e promete em breve recursos para comunicar e interagir com SPI, I2C e PWM, já existe um o RPIO mas a licença é LGPLv3 e irei falar dele em um artigo futuro.

Preparando o ambiente

Vamos preparar o Raspbian para utilizar este módulo, utilizando pip (Gerenciador de Pacotes do Python). Não sabe se possui o pip? Execute o comando abaixo:

```
cleiton@vm03 ~ $ pip -version
pip 1.1 from /usr/lib/python2.7/dist-packages (python 2.7)
```

Se for exibido um erro durante a etapa acima, execute o procedimento abaixo para instalar, caso seja como a saída acima, ignore a etapa abaixo

```
cleiton@vm03 ~ $ apt-get update && apt-get -f -y  install python-pip
```

Vamos verificar o módulo RPi.GPIO e instalar.

RPi.GPIO

Iremos ver agora as principais funções do RPi.GPIO e um detalhe importante ao interagir com o GPIO do Raspberry PI.

RPi.GPIO.setmode()=>Modo de acesso ao GPIO, BOARD (Posição física dos pinos) ou BCM (Número após GPIO, deve-se tomar cuidado com a revisão da placa pois esta informação pode mudar)
RPi.GPIO.setup()=>Configura o pino: (pino e direção [IN ou OUT], exemplo:
RPi.GPIO.setup(12, RPi.GPIO.OUT)

RPi.GPIO.output()=>Configurar como saide: (pino e valor [HIGH ou LOW]), exemplo:
RPi.GPIO.output(12, RPi.GPIO.HIGH)

RPi.GPIO.input()=>Configurar como entrada: (pino) e possui retorno, exemplo:
valor_pino = RPi.GPIO.input(12)

RPi.GPIO.cleanup()=>Restaura para o estado anterior as portas que foram modificadas no programa, deve ser a última linha antes de finalizar o programa.

3.3V	1	2	5V
I2C1 SDA	3	4	5V
I2C1 SCL	5	6	GROUND
GPIO4	7	8	UART TXD
GROUND		10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GROUND
GPIO 22	15	16	GPIO 23
3.3V	17	18	GPIO 24
SP10 MOSI	19	20	GROUND
SP10 MISO	21	22	GPIO 25
SP10 SCLK	23	24	SP10 CE0 N
GROUND	25	26	SP10 CE1 N

Figura 02 – Informações GPIO Raspberry Pi

Ao configurar o setmode() como BOARD ou BCM, a diferença é o número do pino, por exemplo, no esquema informado na Figura01 estamos utilizando o Led1(Vermelho) no ae o Led2(Verde) no GPIO17, que baseado na Figura02 temos a seguinte informação:

BOARD: BCM
11: GPIO17
12: GPIO18

Se o setmode() for setado como BOARD, devemos usar o número 11 e 12 em setup(). Se for setado como BCM, devemos usar 17 e 18 em setup(). É importante se atentar neste detalhe para não ter maiores problemas.

Código

Agora o código necessário para fazer com que o led verde pisque usando Python no modo BOARD, vamos chamar o código abaixo de led1.py.

```
import RPi.GPIO as gpio
import time

# Configurando como BOARD, Pinos Fisicos
gpio.setmode(gpio.BOARD)

# Configurando a direcao do Pino
gpio.setup(11, gpio.OUT) # Usei 11 pois meu setmode é BOARD, se estive usando BCM seria 17
while True:
gpio.output(11, gpio.HIGH)
time.sleep(2)
gpio.output(11, gpio.LOW)
time.sleep(2)

# Desfazendo as modificações do GPIO
gpio.cleanup()
```

Para executar o código led1.py:

```
pi@rpb01 ~/python/led $ sudo python led1.py
```

OBS: Deve ser executado como sudo ou permissão de super-usuario visto que RPi.GPIO acessa /dev/mem.

Novamente o mesmo código, alterando apenas o modo para BCM e trocando para o outro led, no led2.py.

```
import RPi.GPIO as gpio
import time

# Configurando como BCM, Numeracao do GPIO
gpio.setmode(gpio.BCM)

# Configurando a direcao do Pino
gpio.setup(18, gpio.OUT) # Usei 18 pois meu setmode é BCM, se estive usando BOARD seria 12
while True:
gpio.output(18, gpio.HIGH)
time.sleep(2)
gpio.output(18, gpio.LOW)
time.sleep(2)

# Desfazendo as modificações do GPIO
gpio.cleanup()
```

Para encerrar, vamos interagir com os dois leds, utilizando modo BOARD, no código led1_2.py.

```
import RPi.GPIO as gpio
import time

# Configurando como BOARD, identificacao fisica dos pinos
gpio.setmode(gpio.BOARD)
print " Configurando o modo de acesso ao GPIO - BOARD"
print

# Configurando a direcao do Pino
gpio.setup(11, gpio.OUT)
print "Setando modo OUTPUT no PINO11 GPIO17 [LED VERDE]"

gpio.setup(12, gpio.OUT)
print "Setando modo OUTPUT no PINO12 GPIO18 [LED VERMELHO]"
print

gpio.output(11, gpio.HIGH)
print "Led Verde aceso!"
time.sleep(2)

gpio.output(11, gpio.LOW)
print "Led Verde apagado!"
time.sleep(2)

print

gpio.output(12, gpio.HIGH)
print "Led Vermelho aceso!"
time.sleep(2)

gpio.output(12, gpio.LOW)
print "Led Vermelho apagado!"
time.sleep(2)

# Desfazendo as modificações do GPIO
gpio.cleanup()

print
print "Tchau..."
```

Executando o código, o led verde irá acender e apagar, na sequência o vermelho e no terminal será exibido como abaixo:

```
pi@rpb01 ~/python/led $ sudo python led1_2.py
Configurando o modo de acesso ao GPIO - BOARD

Setando modo OUTPUT no PINO 11 GPIO17 [LED VERDE]
Setando modo OUTPUT no PINO 12 GPIO18 [LED VERMELHO]

Led Verde aceso!
Led Verde apagado!
Led Vermelho aceso!
Led Vermelho apagado!
Voltando configuracoes default GPIO

Tchau...
```

Conclusão

Python é muito prático, prototipar essas ideias e exemplos é brincadeira. Abstrai otimização do código para melhorar a didática do exemplo e para melhor compreensão da linguagem e familiarizar com o Python, mas aos poucos, nos próximos irei partindo com ideias pythonicas.

O que vem por aí?

No próximo artigo vamos interagir com o GPIO usando um botão e ver alguns problemas e soluções que podem ser implementadas utilizando o RPi.GPIO.

Referências

https://pypi.python.org/pypi/RPi.GPIO

https://www.embarcados.com.br/arduino-vs-raspberry-pi/

http://pt.wikipedia.org/wiki/General_Purpose_Input/Output

http://www.hobbytronics.co.uk/raspberry-pi-gpio-pinout

http://cleitonbueno.wordpress.com/2014/08/25/python-solucionando-o-erro-fatal-error-python-h-no-such-file-or-directory-no-pip-install/

Este artigo foi escrito por **Cleiton Bueno**.

Raspberry Pi GPIO intup com Python

EMBARCADOS

Introdução

A primeira parte da série de interação com o GPIO (*General Purpose Input Output*) do Raspberry PI no modo Output possui os passos fundamentais para esta segunda etapa como a preparação do ambiente. Nesta segunda etapa vamos ver o modo Input, e para a prática será utilizado o Raspberry PI B, protoboard, pushbutton e alguns resistores. Serão mencionados os possíveis modos de trabalho com Input utilizando resistores de pull-up e pull-down, internos ou externos, e recursos como detecção de borda, debounce por software e uma "interrupção" baseada na detecção por borda.

Configuração do ambiente host

- Board: Raspberry PI B;
- Distribuição: Raspbian – Debian Wheezy (2014-01-07);
- Kernel 3.12.22;
- Python 2.7.3;
- RPi.GPIO 0.5.7.

RPi.GPIO

Já vimos na primeira parte da série as funções **RPi.GPIO.setmode()**, **RPi.GPIO.setup()**, **RPi.GPIO.output()**, **RPi.GPIO.input()** e **RPi.GPIO.cleanup()**, e agora vamos ver algumas funções e parâmetros que tornam o modo input muito interessante.

- **RPi.GPIO.RISING** => Borda de subida, quando passa de 0V para 3.3V;
- **RPi.GPIO.FALLING** => Borda de descida, quando passa de 3.3V para 0V;
- **RPi.GPIO.BOTH** => Os dois estados, ocorre ação tanto na subida [0V → 3.3V] quanto na descida [3.3V → 0V];
- **RPi.GPIO.LOW** => Nivel lógico baixo [low] ou 0V;
- **RPi.GPIO.HIGH** => Nivel lógico alto [high] ou 3.3V;
- **RPi.GPIO.PUD_UP** => Pino Input em modo pull-up;
- **RPi.GPIO.PUD_DOWN** => Pino Input em modo pull-down;
- **RPi.GPIO.PUD_OFF** => Padrão.
- **RPi.GPIO.wait_for_edge()** => Bloqueia a execução até que o botão seja pressionado ou a ação setada aconteça;
- **RPi.GPIO.event_detected()** => Guarda o estado de um evento no pino, e quando for verificado no loop, indica se houve ou não mudança no pino. Pode ser utilizado dentro do loop e não irá perder a informação;
- **RPi.GPIO.add_event_detect()** => Adiciona um evento baseado no pino, se o evento irá ocorrer na borda de descida [FALLING], subida [RISING] ou em ambos [BOTH]. Adiciona uma função para ser chamada e aceita parâmetro de tempo debounce;
- **RPi.GPIO.remove_event_detect()** => Permite remover um evento em qualquer parte do código se não for mais utilizado.

A Figura 1 ilustra os comportamentos que ocorrerão no decorrer do artigo ao acionar o pushbutton.

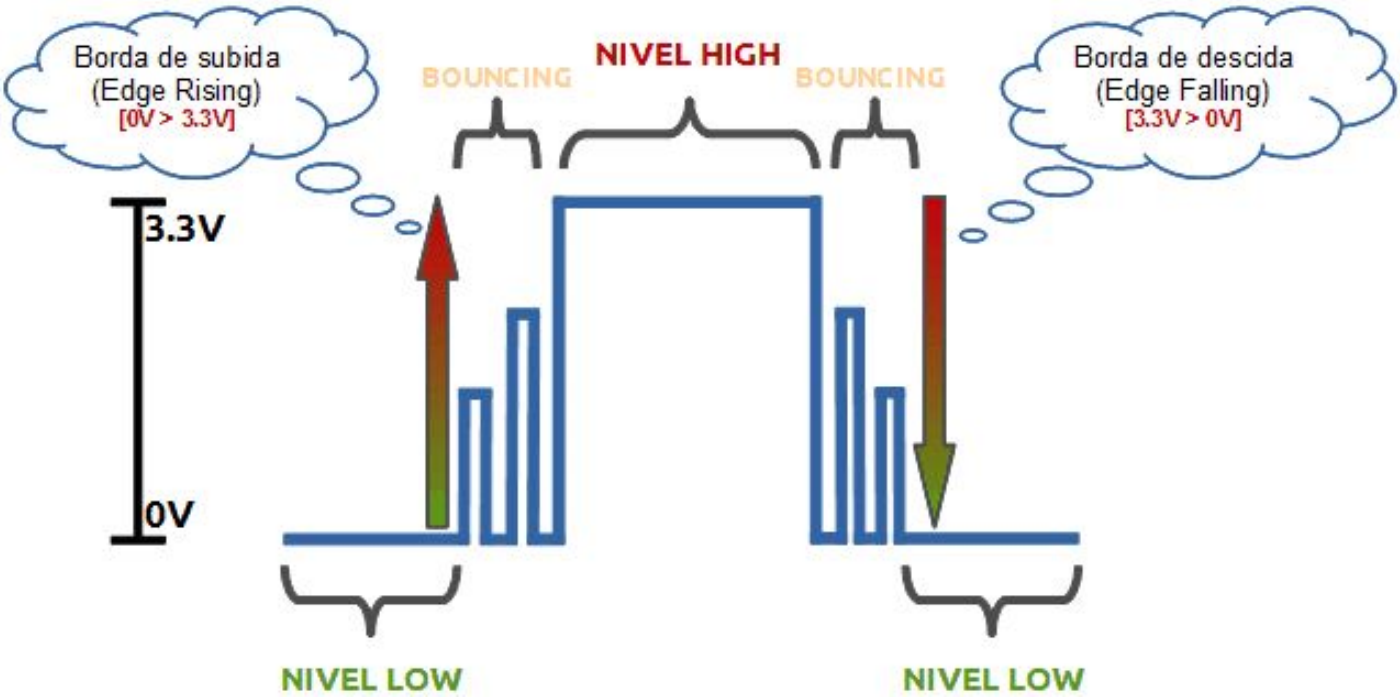


Figura 1 - Detalhes sobre a mudança do nível lógico com chaves mecânicas

Olhando a **Figura 1**, fica fácil entender os parâmetros LOW (representado por 0V), HIGH (como 3.3V), Edge Rising ou RISING (transição de 0V a 3.3V), Edge Falling ou FALLING (transição de 3.3V a 0V) e o efeito bouncing. Esse último, se você não conhece, pode ver o artigo Leitura de chaves mecânicas e o processo de debounce, escrito pelo Rodrigo Almeida, indicando como contornar o problema via hardware e software, e neste artigo iremos estudar a solução via software.

PushButton com pull-down interno

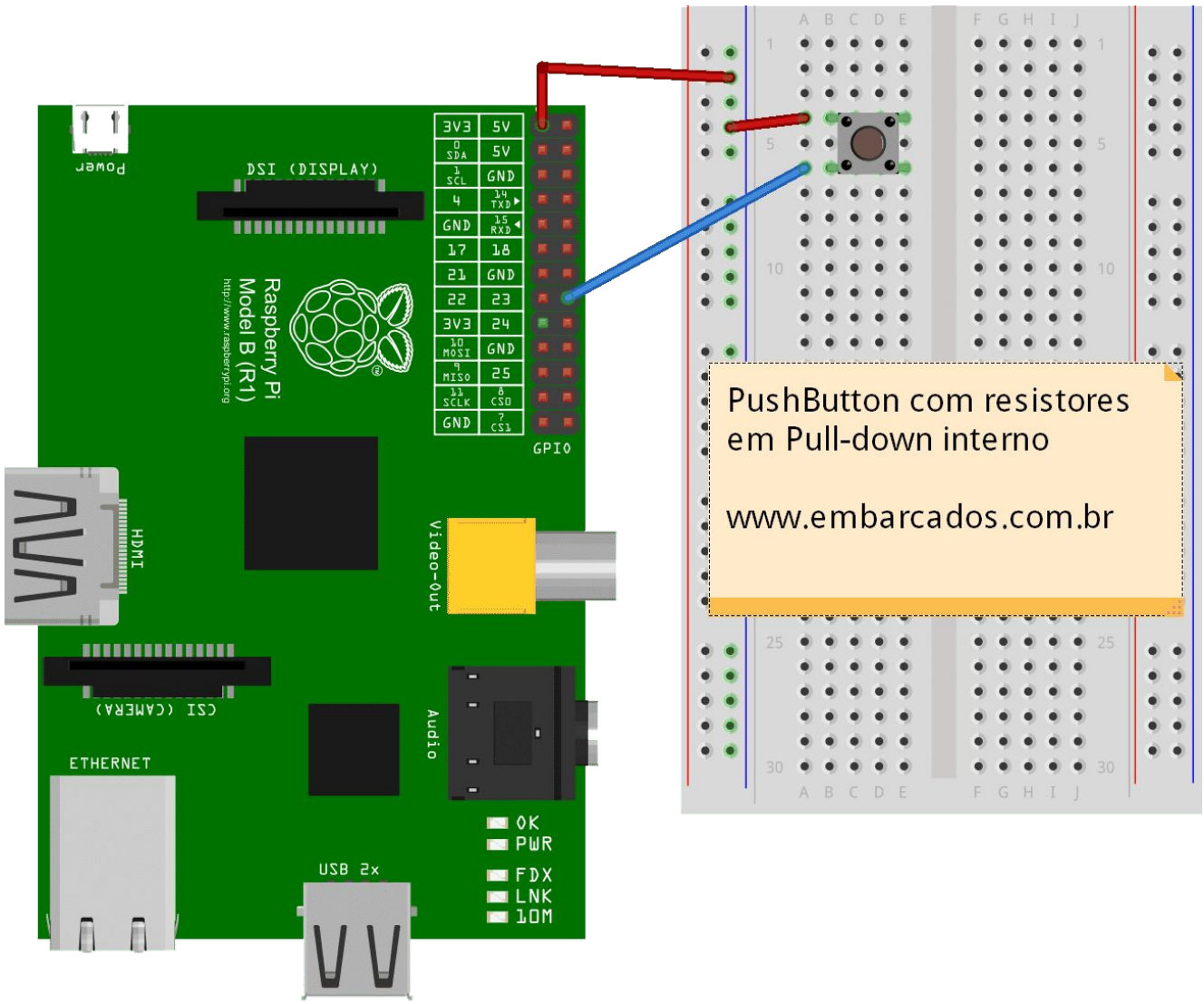


Figura 2 - Esquema de ligação do PushButton em pull-down interno

O primeiro exemplo prático será adicionar um pushbutton. Uma de suas extremidades será conectada ao 3.3V (pino 1) e a outra ao pino 16 (ou GPIO 23) da Raspberry Pi no modo BCM, como na **Figura 2.** Configurados via software utilizando RPi.GPIO o pino 16 (GPIO 23) no modo INPUT e com pull-down interno.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import RPi.GPIO as gpio
import time

gpio.setmode(gpio.BCM)

gpio.setup(23, gpio.IN, pull_up_down = gpio.PUD_DOWN)

while True:
if(gpio.input(23) == 1):
print("Botão pressionado")
else:
print("Botao desligado")
time.sleep(1)

gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_pulldown_interno.py
Botão desligado
Botão desligado
Botão desligado
Botão pressionado
Botão pressionado
Botão desligado
Botão desligado
```

PushButton com pull-up interno

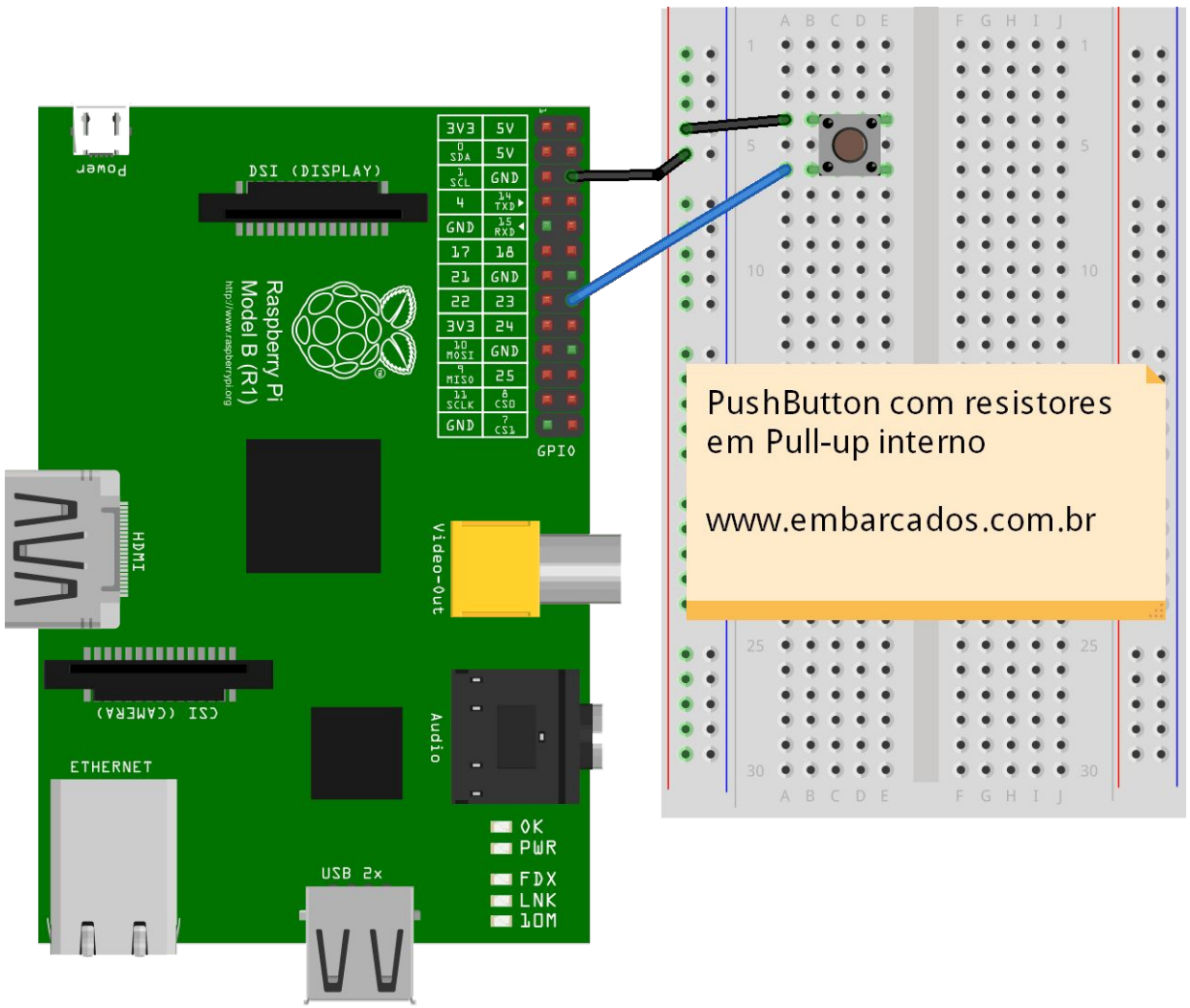


Figura 3 - Esquema de ligação do PushButton em pull-up interno

No segundo exemplo prático o mesmo PushButton será utilizado, trocando apenas a extremidade de 3.3V (pino 1) para 0V GND (pino 6), como na **Figura 3.** Configuraremos via software utilizando RPi.GPIO o pino 16 (GPIO 23) no modo INPUT e com pull-up interno.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import RPi.GPIO as gpio
import time

gpio.setmode(gpio.BCM)

gpio.setup(23, gpio.IN, pull_up_down = gpio.PUD_DOWN)

while True:
if(gpio.input(23) == 1):
print("Botão pressionado")
else:
print("Botao desligado")
time.sleep(1)

gpio.cleanup()
exit()
```

O nosso código acima é praticamente o mesmo do exemplo anterior com algumas alterações como: o setup() do GPIO 23 como pull-up; o teste que verifica o estado do pino faz uso da constante gpio.LOW, que poderia ser 0, seguindo a analogia do exemplo anterior; o botão, quando pressionado, faz o programa parar.

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_pullup_interno.py
Botao desligado
Botao desligado
Botao desligado
Botao pressionado
```

PushButton com pull-down externo

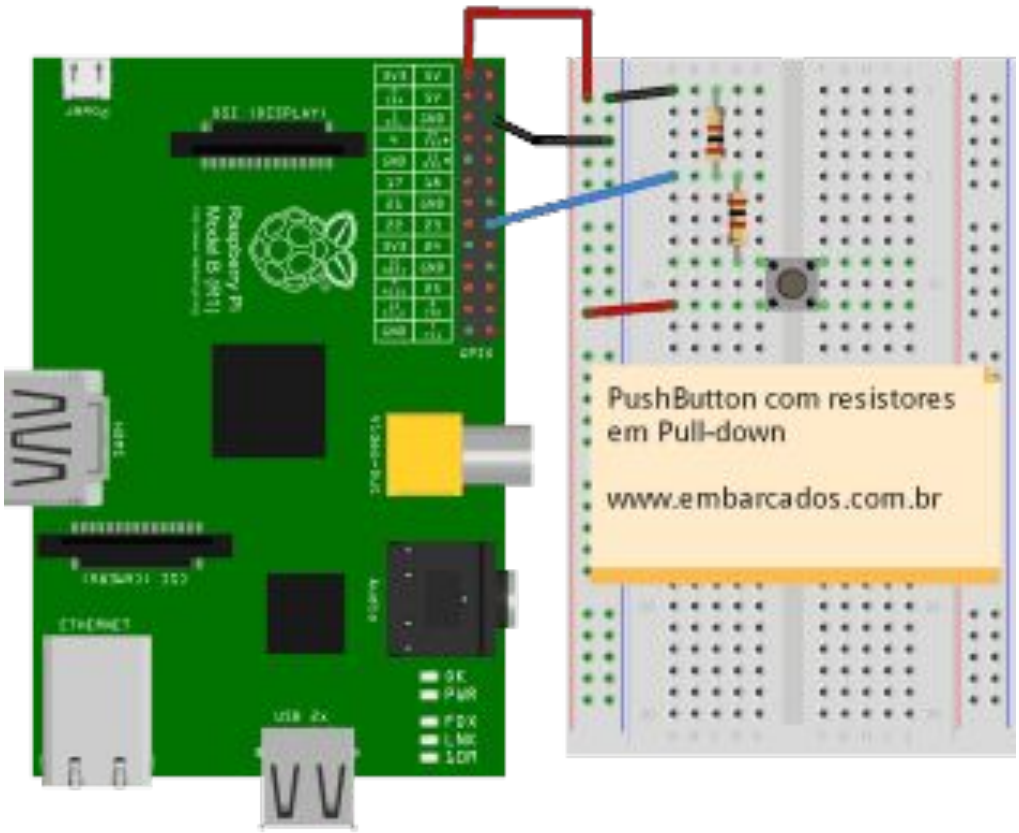


Figura 4 - Esquema de ligação do PushButton em pull-down externo

O próximo exemplo é utilizando resistores externos para configurar como pull-up ou pull-down e, como na **Figura 4**, vamos utilizar 2 resistores: um de 1k e outro de 10k. A configuração que foi utilizada é o resistor de 1k entre o GPIO e o PushButton e o 10k entre GPIO e o GND, caracterizando ser pull-down.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import RPi.GPIO as gpio
import time


""" Global """
PIN=23 # Usando modo BCM


""" Funcoes """
def action_press_button(gpio_pin):
print "O botão no pino %d foi pressionado!" % gpio_pin
print "Saindo..."


""" Configurando GPIO """
# Configurando o modo dos pinos como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN como INPUT
gpio.setup(PIN, gpio.IN)


while True:
if gpio.input(PIN) == gpio.HIGH:
action_press_button(PIN)
break
else:
print "Botão desligado"
time.sleep(1)


gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_pulldown_externo.py
Botão desligado
Botão desligado
Botão desligado
O botão no pino 23 foi pressionado!
Saindo...
```

PushButton com pull-up externo

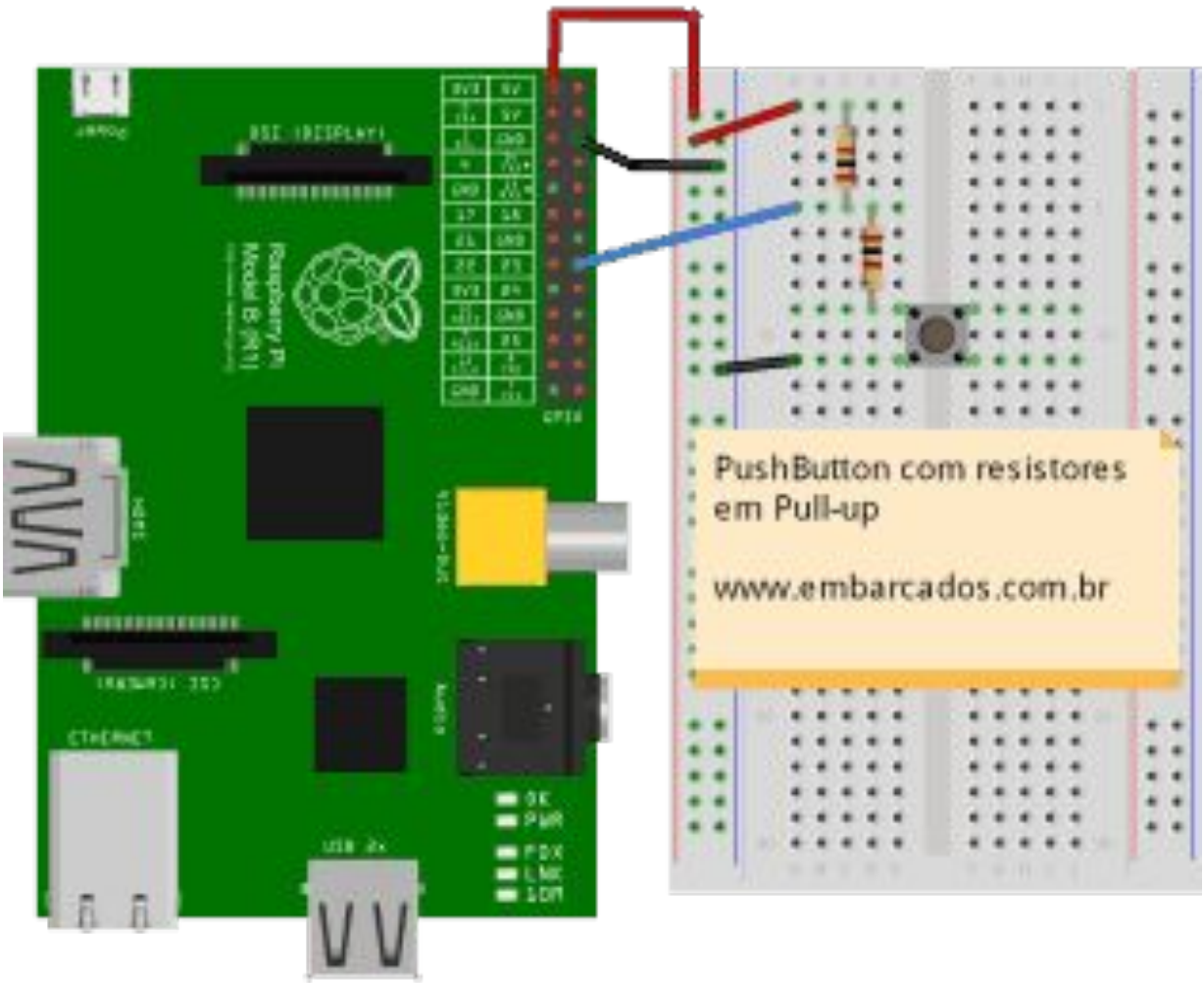


Figura 5 - Esquema de ligação do PushButton em pull-up externo

O esquema final da Figura 5, praticamente o mesmo que o anterior, apenas com a alteração do resistor 10k que é entre VCC (3.3V) com GPIO e o PushButton agora é entre o pino do GPIO e o GND, caracterizando ser pull-up. A mesma ideia deve-se seguir no código.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import RPi.GPIO as gpio
import time


""" Global """
PIN=23 # Usando modo BCM


""" Funcoes """
def action_press_button(gpio_pin):
print "O botão no pino %d foi pressionado!" % gpio_pin
print "Saindo..."


""" Configurando GPIO """
# Configurando o modo dos pinos como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN como INPUT
gpio.setup(PIN, gpio.IN)


while True:
#if gpio.input(PIN) == gpio.LOW:
if gpio.input(PIN) == 0:
action_press_button(PIN)
break
else:
print "Botão desligado"
time.sleep(1)


gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_pullup_externo.py
Botão desligado
Botão desligado
Botão desligado
Botão desligado
Botão desligado
O botão no pino 23 foi pressionado!
Saindo...
```

Não errei o código não, é que utilizando resistores externos altera-se apenas a linha do *if* de gpio.HIGH para gpio.LOW. No caso ainda dei uma valorizada, onde troquei gpio.LOW por 0.

Eventos e Interrupções

Os exemplos utilizados até agora foram legais e interessantes, porém na prática sabemos que é um pouco inconveniente aguardar o loop ou, como no exemplo que fizemos, utilizarmos um delay de 1s. Em alguns casos isso é um absurdo e pode ser que a ação de pressionar o botão não seja capturada, sendo uma falha grave e que pode ser facilmente contornada com as funções de "detecção de eventos" *event_detect* do RPi.GPIO, que tem uma grande relação com *Edge Rising* (Borda de Subida), *Edge Falling* (Borda de Descida) ou *Both* (Ambos sentidos). Utilizaremos o esquemático da Figura 2 por ser mais simples de montar, o PushButton com pull-down e vamos ao código.

Código:

```
# -*- coding: utf-8 -*-
#! /usr/bin/python

import RPi.GPIO as gpio
import time

""" Global """
PIN=23

""" Funcoes """
def action_press_button(gpio_pin):
print "O botão no pino %d foi pressionado!" % gpio_pin
print "Saindo..."


""" Configurando GPIO """
# Configurando o modo dos pinos como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN como INPUT e modo pull-donw interno
gpio.setup(PIN, gpio.IN, pull_up_down = gpio.PUD_DOWN)

# Adicionando um evento ao GPIO 23 na mudança RISING 0V[LOW] - > 3.3V[HIGH]
gpio.add_event_detect(PIN, gpio.RISING)

while True:
print "Polling..."

if gpio.event_detected(PIN):
action_press_button(PIN)
break

time.sleep(1)


gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_event_detect01.py
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
O botão no pino 23 foi pressionado!
Saindo...
```

Você pode testar, e pode ver que independente do momento que pressionar o pushbutton, na ação dele ser pressionado (RISING), indo de 0V para 3.3V, como estamos com pull-down, ele irá "guardar" esta mudança de estado. Na próxima vez que for executado o loop, este evento será tratado, no nosso caso sendo chamada a função **action_press_button()** e saindo do programa.

E se por acaso quiséssemos que a ação ocorresse ao soltar o pushbutton e não ao pressionar o mesmo?

Código:

```
# -*- coding: utf-8 -*-
#! /usr/bin/python

import RPi.GPIO as gpio
import time

""" Global """
PIN=23

""" Funcoes """
def action_press_button(gpio_pin):
print "O botão no pino %d foi pressionado!" % gpio_pin
print "Saindo..."


""" Configurando GPIO """
# Configurando o modo dos pinos como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN como INPUT e modo pull-donw interno
gpio.setup(PIN, gpio.IN, pull_up_down = gpio.PUD_DOWN)

# Adicionando um evento ao GPIO 23 na mudança FALLING 3.3V[HIGH] - > 0V[LOW]
gpio.add_event_detect(PIN, gpio.FALLING)

while True:
print "Polling..."

if gpio.event_detected(PIN):
action_press_button(PIN)
break

time.sleep(1)

gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_event_detect02.py
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
O botão no pino 23 foi pressionado!
Saindo...
```

Não mudou nada na saída do código, tirando que o programa só parou quando pressionamos e soltamos o PushButton. Enquanto o botão estiver pressionado, nada acontece. O mesmo pode ser feito com BOTH, que a ação ocorrerá ao pressionar e soltar.

Outra opção interessante no event_detect() é poder adicionar uma função a ser chamada assim que ocorrer a mudança de estado, utilizando o parâmetro callback. Seria a correspondência de uma "Interrupção", porém eu não gosto muito deste termo e prefiro aceitar o de uma chamada em paralelo ou execução já que uma segunda thread é disparada neste caso. Vamos ver!

Código:

```
# -*- coding: utf-8 -*-
#! /usr/bin/python

import RPi.GPIO as gpio
import time

""" Global """
PIN=23

""" Funcoes """
def action_press_button_loop(gpio_pin):
print "O botão no pino %d foi pressionado!" % gpio_pin
print "Saindo..."

def action_press_button(gpio_pin):
print "Tratando o botão no pino %d que foi pressionado!" % gpio_pin

""" Configurando GPIO """
# Configurando o modo dos pinos como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN como INPUT e modo pull-donw interno
gpio.setup(PIN, gpio.IN, pull_up_down = gpio.PUD_DOWN)

# Adicionando um evento ao GPIO 23 na mudança FALLING 0V[LOW] - > 3.3V[HIGH]
gpio.add_event_detect(PIN, gpio.RISING, callback=action_press_button)

# Junto com o parametro callback podemos utilizar ainda o bouncetime
# na linha abaixo estamos dizendo para ignorar nos primeiro 300ms
# gpio.add_event_detect(PIN, gpio.RISING, callback=action_press_button, bouncetime=300)

while True:
print "Polling..."

if gpio.event_detected(PIN):
action_press_button_loop(PIN)
break

time.sleep(1)

gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_event_detect03.py
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Polling...
Tratando o botão no pino 23 que foi pressionado!
Polling...
O botão no pino 23 foi pressionado!
Saindo...
```

Usando callback no add_event_detect(), podemos ver que existem “prioridades” na chamada, porque assim que o evento ocorre é lançada uma segunda thread para a chamada. Após isso vem a continuação do loop e depois é atendida nossa mudança de estado no próprio loop como antes. Assim conseguimos prioridade no tratamento de input independente do tamanho e tempo de nosso loop, caracterizando a interrupção dita. Comentando sobre a opção de utilizar o debounce via software, como na linha 30, com ou sem o callback, podemos definir o parâmetro bouncetime, que é um valor de tempo em ms para ser ignorado na detecção da borda.

A função remove_event_detect(), pode ser chamada e passado como parâmetro o pino. A qualquer momento o evento sobre o mesmo pino passa a não ser mais válido e você pode remover a ação sobre a borda naquele pino. Vamos ver no próximo código.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import Rpi.GPIO as gpio
import time


""" Global """
PIN=23

""" Funcoes """
def action_press_button(gpio_pin):
print "Você pressionou o botão do pino %d e agora ele sera desativado" % gpio_pin


""" Configurando GPIO """
# Configurando o modo do GPIO como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN's como INPUT e modo pull-down interno
gpio.setup(PIN, gpio.IN, pull_up_down = gpio.PUD_DOWN)

# Adicionando um evento ao GPIO 23 na mudança RISING 0V[LOW] -> 3.3V[HIGH]
gpio.add_event_detect(PIN, gpio.RISING)


while True:
try:
if gpio.event_detected(PIN):
    action_press_button(PIN)
    gpio.remove_event_detect(PIN)
else:
                    print("Botão Desligado")

time.sleep(1)
except (KeyboardInterrupt):
print("Saindo...")
gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_event_detect04.py
Botão desligado
Botão desligado
Botão desligado
Você pressionou o botão no pino 23 e agora ele sera desativado
Botão desligado
Botão desligado
Botão desligado
^CSaindo...
```

E por último o **wait_for_edge()**, que podemos parar a execução do programa até que a mudança no pino ocorra, também sendo válido para a mudança RISING, FALLING ou BOTH.

Código:

```
#! /usr/bin/python
# -*- coding: utf-8 -*-

import Rpi.GPIO as gpio
import time


""" Global """
PIN1=23

""" Configurando GPIO """
# Configurando o modo do GPIO como BCM
gpio.setmode(gpio.BCM)

# Configurando PIN's como INPUT e modo pull-down interno
gpio.setup(PIN1, gpio.IN, pull_up_down = gpio.PUD_DOWN)



while True:
try:
print "Inicio loop..."
gpio.wait_for_edge(PIN1, gpio.RISING)
print "Botão foi pressionado"
gpio.wait_for_edge(PIN1, gpio.FALLING)
print "Botão foi liberado"
print "Continua loop..."
time.sleep(1)
except (KeyboardInterrupt):
print "Saindo..."
gpio.cleanup()
exit()
```

Executando o código:

```
pi@raspberrypi ~/python/rpio/ $ sudo python pushbutton_event_detect05.py
Inicio loop...
Botão foi pressionado
Botão foi liberado
Continua loop...
Inicio loop...
Botão foi pressionado
Botão foi liberado
Continua loop...
Inicio loop...
Botão foi pressionado
Botão foi liberado
Continua loop...
Inicio loop...
Botão foi pressionado
Botão foi liberado
Continua loop...
^CSaindo...
```

Conclusão

Podemos ver mais uma vez como Python é uma ferramenta fantástica para prototipagem. Com poucas linhas e utilizando o RPi.GPIO conseguimos manipular de diversas maneiras o GPIO do Raspberry Pi no modo input.

O que vem por aí?

No próximo artigo vamos continuar brincando com o GPIO do Raspberry Pi, só que desta vez partindo para PWM.

Este artigo foi escrito por **Cleiton Bueno**.

Participe da comunidade no Facebook

RFID com Raspberry Pi e Python

EMBARCADOS

Introdução

Identificação por radiofrequência, como podemos chamar o RFID, não é uma tecnologia recente e há especulações que já era utilizada na segunda guerra mundial. É a comunicação por sinal de rádio onde um dos dispositivos é energizado e fica ativo, e o outro lado, a TAG (chaveiro ou cartão RFID), não necessariamente é energizado. Quando aproximados, realizasse a captura da informação do TAG, tecnologia essa que deu o passo inicial para NFC (*Near Field Communication*).

Utilizando o computador de bolso mais popular, o Raspberry Pi, a linguagem de programação Python, que é o canivete suíço para prototipar ideias, e um módulo RFID comercial vamos ver como é simples, direto e didático fazer a comunicação via SPI com essa turma.

Não conhece muito bem o Raspberry Pi? Veja o artigo **Raspberry Pi** do Thiago Lima. E como vamos usar o Raspberry Pi B com Linux, para conhecer melhor essa dupla veja o artigo **A Raspberry Pi e o Linux** de Vinicius Maciel. Se isso não te assusta, continue lendo o artigo.

Módulo RFID

O Mifare MFRC522 (NXP RC522), ou comercialmente RFID-RC522, é um dos mais populares leitores RFID do mercado, no modo comprar, ligar e usar, uma excelente forma para prototipar.

Figura 1 - Módulo RFID com cartão e chaveiro

O RFID-RC522 já vem pronto para ser utilizado, fazendo uso da comunicação SPI. Essa comunicação é nova pra você? Então recomendo o artigo **Comunicação SPI** do Francesco Sacco. Além disso o CI RC522 possui os recursos I2C e UART para comunicação e realiza comunicação RFID a 13.56 MHz.

Na figura 1 temos uma imagem do nosso módulo leitor RFID (RFID-RC522) e junto o que o acompanha na compra, um chaveiro e um cartão.
Segue a relação dos pinos disponíveis para alimentação e comunicação na figura 2.

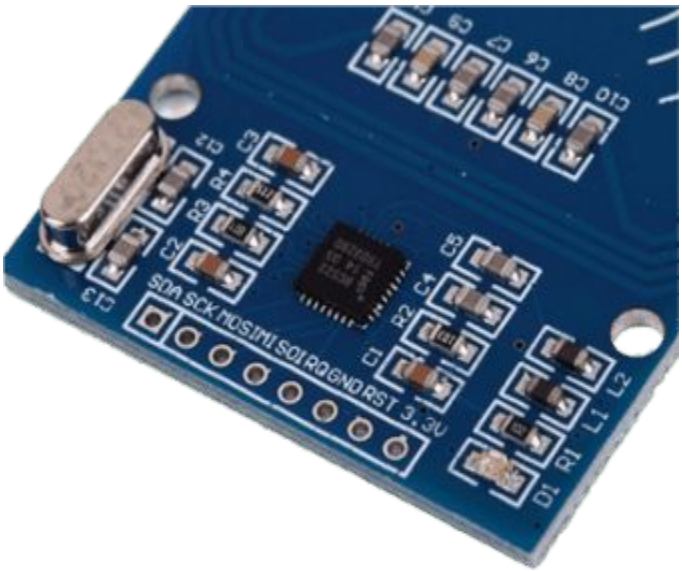


Figura 2 - Relação de pinos de alimentação e comunicação do módulo RFID

Esquema de ligação

Na sequência, na figura 3, o esquema de ligação do módulo RFC522 com o Raspberry Pi.

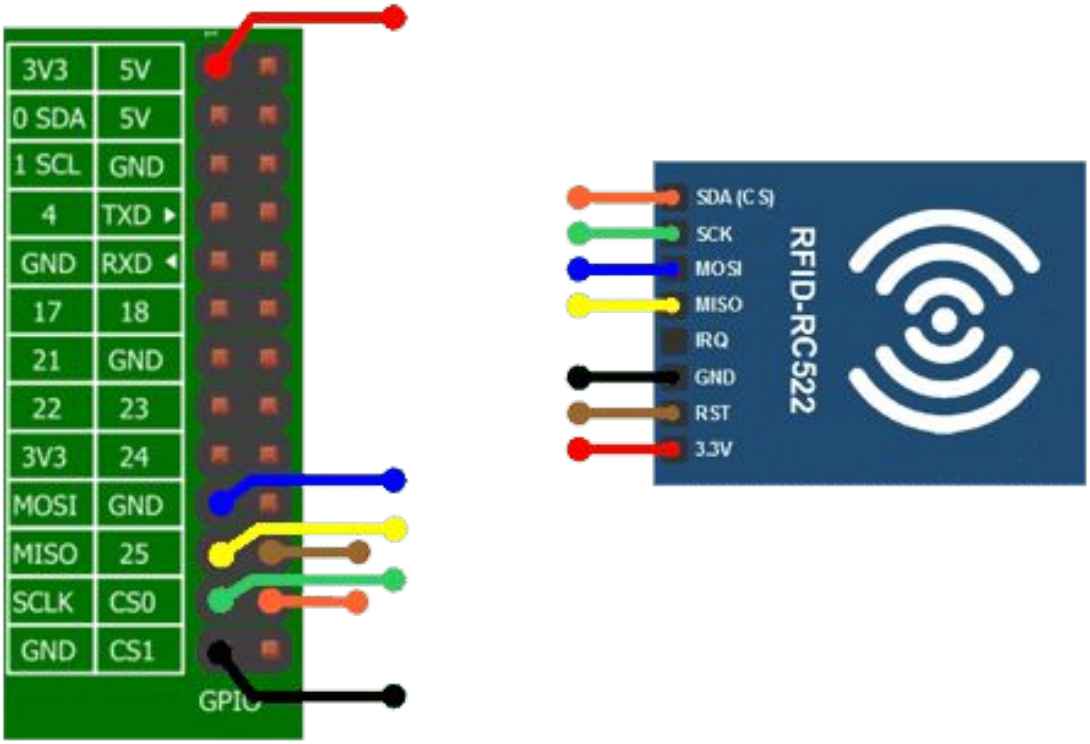


Figura 3 - Esquema de ligação do módulo RFID com o GPIO do Raspberry Pi B

Apenas alguns detalhes no esquema de ligação é a alimentação 3,3V utilizando o pino 1 do barra de pinos de GPIO, e a comunicação SPI, sendo o SDA no módulo RFID o sinal CS (*Chip Select*) ou SS (*Slave Select*), que depende da adoção do fabricante, sendo o correto uso do nome SDA para comunicação I2C.

Configuração do ambiente host

- Board: Raspberry Pi B;
- Distribuição: Raspbian – Debian Wheezy (2014-01-07);
- Kernel 3.12.22;
- Python 2.7.

Preparando o ambiente

Agora a parte legal, vamos preparar o Raspbian para liberar o uso device SPI no Linux, resolver as dependências de software e testar com uma aplicação padrão de teste. Para isso, o primeiro passo é verificar se o dispositivo SPI está liberado, ou melhor, se o módulo do kernel subiu para fornecer o acesso.

```
pi@raspberrypi ~ $ ls /dev/spi*
ls: cannot access /dev/spi*: No such file or directory
pi@raspberrypi ~ $ lsmod | grep spi_bcm*
pi@raspberrypi ~ $ dmesg | grep spi
pi@raspberrypi ~ $
```

Se na sua Raspberry Pi obteve as mesmas saídas acima, o módulo SPI está desabilitado. Caso contrário, pule a próxima etapa. No Raspbian isso quer dizer que ele está no black list do modprobe, e para habilitar o módulo SPI siga o passo a passo abaixo e reinicie a Raspberry Pi.

```
pi@raspberrypi ~ $ sudo vim /etc/modprobe.d/raspi-blacklist.conf

# blacklist spi and i2c by default (manu users don't need them)

#blacklist spi-bcm2708
blacklist i2c-bcm2708

:wq
pi@raspberrypi ~ $ sudo reboot
```

Verificando novamente:

```
pi@raspberrypi ~ $ ls /dev/spi*
/dev/spidev0.0 /dev/spidev0.1
pi@raspberrypi ~ $ lsmod | grep
spi_bcm spi_bcm2708 4808 0
pi@raspberrypi ~ $ dmesg | grep spi
[ 5.078766] bcm2708_spi bcm2708_spi.0: master in unqueued, this is deprecated
[ 5.230200] bcm2708_spi bcm2708_spi.0: SPI Controller at 0x2020400 (irq 80)
pi@raspberrypi ~ $
```

Para ler/escrever no barramento SPI precisamos de um módulo no Python. Um que possui rotinas e funções bem escritas é o SPI-Py do Louis Thiery. Vamos clonar o repositório do GitHub, compilar e instalar. Mas antes, é necessário confirmar se possui o python-dev. Você pode verificar com o comando abaixo:

```
pi@raspberrypi ~ $ dpkg -list | grep -E 'ii.*python.*dev'
ii python-dbus-dev 1.1.1-1 all main loop integration development files
ii python2.7-dev 2.7.3-6+deb7u2 armhf Headers files and static library for Python
(v2.7)
```

No caso acima temos instalado. Caso não aparecesse python2.7-dev ou python-dev, poderia executar a sua instalação com o comando abaixo:

```
pi@raspberrypi ~ $ sudo apt-get update && sudo apt-get install python-dev -f
```

Agora vamos clonar o repositório do SPI-Py, acessar o diretório e instalar.

```
pi@raspberrypi ~ $ git clone https://github.com/lthiery/SPI-Py spi-py
Cloning into 'spi-py'...
remote: Counting objects: 64, done.
remote: Total 64 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (64/64), done.
pi@raspberrypi ~ $
pi@raspberrypi ~ $ cd spi-spy
pi@raspberrypi ~/spi-py $ sudo python setup.py install
running install
running build
running build_ext
building 'spi' extension
...
running install_lib
copying build/lib.linux-arm6l-2.7/spi.so → /usr/local/lib/python2.7/dist-packages
running install egg_info
Writing /usr/local/lib/python2.7/dist-packages/SPI_Py-1.0.egg-info
pi@raspberrypi ~/spi-py $
```

O módulo para comunicação via SPI está pronto. Agora vamos baixar de outro repositório um código exemplo pronto para comunicar com RC522, que depende do módulo SPI-Py para funcionar, no caso é o MFRC522-python que, além do SPI-Py, utiliza também o RPi.GPIO.

```
pi@raspberrypi ~/spi-py $cd ~
pi@raspberrypi ~ $ mkdir -p python_rfid
pi@raspberrypi ~ $ cd python_rfid
pi@raspberrypi ~/python_rfid $ git clone https://github.com/mxgxw/MFRC522-python
Cloning into 'MFRC522-python'...
remote: Counting objects: 60, done.
remote: Total 60 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (60/60), done.
pi@raspberrypi ~/python_rfid $ cd MFRC522-python
pi@raspberrypi ~/python_rfid/MFRC522-python $
```

Testando a aplicação RFID

Dos arquivos clonados do repositório e que estão no diretório MFRC522-python, o importante neste momento é saber que toda programação e tratamento do SPI com o RF522 está no arquivo MFRC522.py, e que no arquivo Read.py há um exemplo pronto para testar a comunicação e o Write.py é outro exemplo para gravação no RC522.

```
pi@raspberrypi ~/python_rfid/MFRC522-python $ sudo python Read.py
Welcome to the MFRC522 data read example
Press Ctrl+C to stop.
```

No momento que aproxime o cartão:

```
pi@raspberrypi ~/python_rfid/MFRC522-python $ sudo python Read.py
Welcome to the MFRC522 data read example
Press Ctrl+C to stop.
Card detected
Card read UID: 227, 41, 93, 116
Size: 8
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
^Cctrl+C captured, ending read.
```

Agora, vamos utilizar o módulo MFRC522.py e criar uma aplicação nossa baseada no Read.py. Primeiro vou abrir o código do Read.py e comentá-lo, logo em seguida irei escrever nossa aplicação (veja os comentários para mais detalhes).

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import RPi.GPIO as GPIO
import MFRC522
import signal

continue_reading = True

"""
Função que é chamada assim que Ctrl+C é pressionado
"""
# Capture SIGINT for cleanup when the script is aborted

def end_read(signal,frame):
    global continue_reading
    print "Ctrl+C captured, ending read."
    continue_reading = False
    GPIO.cleanup()

"""
Responsável por capturar o SIGINT gerado (Ctrl+C) e chamar a função end_read
"""
# Hook the SIGINT
signal.signal(signal.SIGINT, end_read)

"""
Criando um objecto da class MFRC522
"""
# Create an object of the class MFRC522
MIFAREReader = MFRC522.MFRC522()

# Welcome message
print "Welcome to the MFRC522 data read example"
print "Press Ctrl-C to stop."

"""
```

```
Entra em um loop infinito baseado na variavel global :( continue_reading, que ira mudar o seu
valor assim que
a aplicação for interrompida e a função end_read() irá setar como False
"""
# This loop keeps checking for chips. If one is near it will get the UID and authenticate
while continue_reading:
    """
    Maioria das chamadas abaixo falam por si só, irei comentar as que achar interessante
    """
    # Scan for cards
    (status,TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)

    # If a card is found
    if status == MIFAREReader.MI_OK:
        print "Card detected"

    """
    Agora iremos receber o ID RFID e o status, e o nome Anticoll é baseado no proprio protocolo
    no Registrador ErrorReg bit3 verifica se houve colisão dos dados
    """
    # Get the UID of the card
    (status,uid) = MIFAREReader.MFRC522_Anticoll()


    # If we have the UID, continue
    if status == MIFAREReader.MI_OK:

        # Print UID
        print "Card read UID: "+str(uid[0])+","+str(uid[1])+","+str(uid[2])+","+str(uid[3])

        """
        Uma camada de segurança a mais, onde além de verificar o ID que recebemos podemos enviar a
key dos chaveiros e tags e aguardar a autenticação
        Veja no datasheet MF1S70yyX.pdf Pagina: 11 - 8.6.3 Sector trailer
        E as rotinas para esta checagem segue abaixo.
```

```
Usando a key abaixo é a chave de 6 bytes utilizada para
        autenticar a comunicação maneira esta usando o MFAuthent
        endereço 0x60 a 0x61

        OBS: Não irei utilizar isso em nossa aplicação!
        """
        # This is the default key for authentication
        key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]

        # Select the scanned tag
        MIFAREReader.MFRC522_SelectTag(uid)

        """
        Aqui é enviado a key do "fabricante", modo de autenticação 0x60 ou 0x61, Block Address e
nosso uid
        """
        # Authenticate
        status = MIFAREReader.MFRC522_Auth(MIFAREReader.PICC_AUTHENT1A, 8, key, uid)

        # Check if authenticated
        if status == MIFAREReader.MI_OK:
            MIFAREReader.MFRC522_Read(8)
            MIFAREReader.MFRC522_StopCrypto1()
        else:
            print "Authentication error"
```

O código foi exposto e adicionei alguns comentários para facilitar. As informações detalhadas sobre o protocolo e o modo de autenticação pode ser abstraído no datasheet do cartão RFID. No meu caso o cartão é um MIFARE Classic 4k e do código do MFRC522.py.

Agora vamos à nossa aplicação **rfid_embarcados.py**, onde utilizaremos a estrutura pronta do Read.py, removendo a parte de autenticação.

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

import os
import sys
import signal

"""
    Lista com relação dos IDs autorizados
"""
acessos_autorizados = [[227,41,93,116,227], [201,39,92,115,201], [225,95,12,103,225]]

"""
    Vou tentar importar os modulos abaixo, caso algum problema ocorra,
    sera lançada a exceção na sequencia
"""
try:
    import MFRC522
    import RPi.GPIO as GPIO
except ImportError as ie:
    print("Problema ao importar modulo {0}").format(ie)
    sys.exit()

"""
    Funcao que irá garantir que o root ou usuario com permissão de
    super-usuario irá executar a aplicação
"""
def check_user():
    if os.geteuid() != 0:
        print("Você deve executar o programa como super-usuario!")
        #print "Exemplo:\nsudo python {0}".format(os.path.realpath(__file__))
        print("Exemplo:\nsudo python {0}").format(__file__)
        sys.exit()


"""
    Captura o sinal gerado, no caso o que nos interessa é o sinal
    SIGINT(Interrupção do Terminal ou processo) e irá encerrar a aplicação
"""
def finalizar_app(signal,frame):
    global continue_reading
    print("\nCtrl+C pressionado, encerrando aplicação...")
    continue_reading = False
    GPIO.cleanup()
continue_reading = True
```

```
def main():

    check_user()

    # Handler do sinal SIGINT
    signal.signal(signal.SIGINT, finalizar_app)

    # Cria o objeto da class MFRC522
    MIFAREReader = MFRC522.MFRC522()

    print("Portal Embarcados - Python é sucesso!")
    print("Pressione Ctrl-C para encerrar a aplicação.")

    while continue_reading:
        # Scan for cards
        #(status,TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)
        MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)


        # Get the UID of the card
        (status,uid) = MIFAREReader.MFRC522_Anticoll()

        # If we have the UID, continue
    if status == MIFAREReader.MI_OK:
            if uid in acessos_autorizados:
                    print("Acesso liberado!")
            else:
                    print("Sem acesso!")



if __name__ == "__main__":
    main()
```

No caso, para um controle simples, criei uma lista chamada acessos_autorizados=[]que contém listas de ID's RFID. Na linha 73 verifico se o uid recebido está contido ou não da minha lista. Em caso positivo, irá imprimir "Acesso Liberado!", caso contrário, "Sem Acesso!". Agora vamos ver nossa aplicação em funcionamento.

```
pi@raspberrypi ~/rfid-python $ sudo python rfid_embarcados.py
Portal Embarcados - Python é sucesso!
Pressione Ctrl-C para encerrar a aplicação.
Acesso liberado!
Acesso liberado!
Acesso liberado!
Sem acesso!
Acesso liberado!
^C
Ctrl+C pressionado, encerrando aplicação...
```

A aplicação funcionou como previsto, porém, algumas implementações novas surgiram na aplicação como a função **check_user()** e um except **ImportError**. Nesses casos, estou garantindo que a aplicação só deve continuar se estiver sendo executada com permissão de super-usuário e é obrigatório ter MFRC522 e RPi.GPIO para prosseguir. Agora vamos ver o que ocorre se executar a aplicação como usuário comum e logo em seguida, executar se não tivermos o MFRC522.py.

```
pi@raspberrypi ~/rfid-python $ python rfid_embarcados.py
Você deve executar o programa como super-usuario!
Exemplo:
sudo python rfid_embarcados.py
pi@raspberrypi ~/rfid-python $
pi@raspberrypi ~/rfid-python $ sudo python rfid_embarcados.py
Problema ao importar modulo No module named MFRC522
pi@raspberrypi ~/rfid-python $
```

Para melhorar essa aplicação, poderia utilizar um banco de dados que contém os IDs com alguns nomes e informações, realizar consultas e, em caso de sucesso, poderia acionar algum GPIO e abrir uma porta ou liberar algo. Implemente, invente e faça seus testes e qualquer dúvida envie pelos comentários.0

Conclusão

Vimos como é fácil realizar a configuração do módulo ao Raspberry Pi B, como é prático preparar o Linux com Python e realizar os testes, e como é simples utilizar o módulo que baixamos para criar uma aplicação própria para ler os dados RFID. E utilizando um arquivo exemplo mais os recursos do python conseguimos criar uma aplicação simples e com recursos interessantes, como por exemplo verificar se é o root que está executando e se o módulo existe.

Referências

http://www.nxp.com/documents/data_sheet/MFRC522.pdf

https://github.com/lthiery/SPI-Py

https://github.com/mxgxw/MFRC522-python

https://pypi.python.org/pypi/RPi.GPIO

https://www.nxp.com/documents/data_sheet/MF1S70YYX.pdf

Este artigo foi escrito por **Cleiton Bueno**.

Raspberry Pi Display LCD com Python

EMBARCADOS

Este artigo apresentará uma breve teoria sobre display de LCD, quais são os seus modos de utilização, como realizar as conexões com a Raspberry Pi e principalmente como desenvolver um código em linguagem Python de forma que se consiga utilizar todos os recursos disponíveis neste componente tão importante em sistemas embarcados.

Ao final da leitura, aplicando os conceitos teóricos e práticos apresentados sobre o display de LCD na Raspberry Pi, o leitor será capaz de realizar os seus próprios projetos e aplicações com o dispositivo.

Este artigo dá continuidade à série de artigos sobre entrada e saída de dados e utilização do PWM publicado pelo Cleiton Bueno e por mim, Roniere Rezende.

O que é um Display de LCD

O display de LCD é uma interface de comunicação visual alfanumérica muito utilizada em diversos equipamentos eletrônicos, como equipamentos industriais, eletrodomésticos, brinquedos e muitos outros. Abaixo é mostrado um exemplo desse dispositivo.

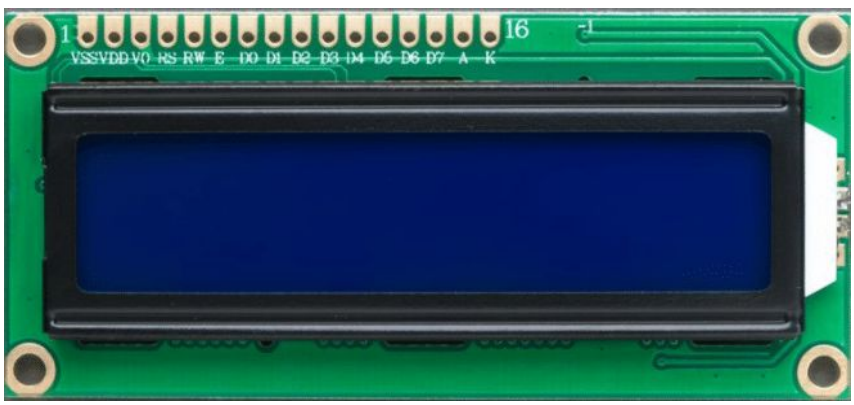


Figura 1 - Display de LCD

Eles podem ser especificados pelo número de caracteres que podem existir em cada linha e também pelo número de linhas. Por exemplo 16x2, 20x2, 40x4, entre outros.

Existem dois modos de se conectar e configurar o display de LCD à Raspberry Pi: modo 8 bits, que utiliza 10 portas GPIO para se realizar a manipulação do LCD; e modo 4 bits, que utiliza 6 portas GPIO para se realizar a manipulação do LCD.

Neste artigo será abordado o display de LCD 16x2 e trabalhando em modo 4 bits para fins de economia de portas GPIO. Para maiores informações sobre esses dispositivos, recomendo a leitura do artigo Módulo de Display LCD publicado por Henrique Puhlmann.

Instalação da Biblioteca do Display LCD

Para a instalação da biblioteca utilizada para se trabalhar com o display LDC, deve-se primeiramente conectar a Raspberry Pi à internet. Em seguida, abre-se o LX Terminal e executa-se os comandos a seguir para atualizar e se certificar que a biblioteca GPIO está instalada:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip git
sudo pip install RPi.GPIO
```

Desconsidere caso apareça alguma informação de erro, pois a biblioteca provavelmente já estava instalada anteriormente. A biblioteca do display LCD que vamos trabalhar é a Adafruit_Python_CharLCD. Deve-se realizar o download e executá-la, utilizando os seguintes comandos:

```
cd ~
git clone https://github.com/adafruit/Adafruit_Python_CharLCD.git
cd Adafruit_Python_CharLCD
sudo python setup.py install
```

Terminando a instalação, as bibliotecas já estarão disponíveis para o seu uso. A seguir são definidas as principais funções disponíveis.

Comandos

Os comandos Python utilizados para realizar a comunicação entre a Raspberry e o display LDC fazem parte da biblioteca Adafruit_Python_CharLDC e devemos usar o comando import para poder utilizar esses comandos nos nossos códigos. Como utilizar o comando import é demonstrado na seção “Código”. Lembrando que estes comandos são configurados para serem aplicados nos displays que têm conexões e configurações similares ao HD44780 Character LCD da Hitachi:

- lcd.home():

Move o cursor para a primeira linha e primeira coluna.

- lcd = LCD.Adafruit_CharLCD(rs, e, d4, d5, d6, d7, bl)

Cria um objeto que nesse exemplo é chamado de “lcd”, mas poderia ser qualquer outro nome. Também inicializa e configura os pinos que enviaram os dados para o display LCD. Esta função configura o modo de funcionamento em 4 bits.

- lcd.clear():

Limpa a tela do LCD, o deixando sem mostrar qualquer informação.

- lcd.set_cursor(col, row):

Posiciona o cursor na posição definida pelos valores que são definidos por “col” e por “row”. Como o display de LCD que estamos usando é 16 x 2, então o valor de “col” varia entre 0 a 15 e “row” varia entre 0 ou 1.

- lcd.enable_display(enable):

Habilita ou desabilita o display.

- lcd.show_cursor(show):

Mostra ou esconde o cursor. O cursor é mostrado se o valor de “show” for igual a true, caso contrário, se for false, não é mostrado (esse é o valor padrão).

- lcd.blink(blink):

Habilita ou desabilita a cintilação do cursor. Se “blink” for igual a true a cintilação do cursor é habilitada e caso for false a cintilação é desabilitada (esse é o valor padrão).

- lcd.move_left():

Move o cursor para a esquerda uma posição.

- lcd.move_right():

Move o cursor para a direita uma posição.

- lcd.set_left_to_right():

Configura a direção do texto da esquerda para a direita.

- lcd.set_right_to_left():

Configura a direção do texto da direita para a esquerda.

- lcd.autoscroll(autoscroll):

Essa função formata o texto justificado à direita se “autoscroll” recebe o valor true, caso contrário o texto será justificado à esquerda.

- lcd.message(‘texto”):

Insere um texto no display LCD, incluindo o comando newline “/”.

- lcd.backlight(backlight):

Habilita ou desabilita o backlight do display LCD. Se “backlight” for igual a true o “backlight” será habilitado, sendo este o valor padrão, e caso for false o “backlight” será desabilitado.

- lcd.write8(value, char_mode = false):

Recebe um valor de 8 bits e escreve no display LCD um caractere ou um dado. O “value” deve ser um valor inteiro de 0 a 255. Se “char_mode” é true o valor retornado é um caractere, caso contrário será um dado.

- lcd.create(location, pattern):

Pode-se criar ou personalizar um novo caractere preenchendo cada um dos 8 bits nas 8 linhas que formam um caractere no display LCD. Para maior entendimento de como criar um novo caractere, entrem neste endereço.

Existem mais funções dentro da biblioteca mas que não são tão utilizadas. Devido a isso, não foram apresentadas neste artigo. Caso haja interesse de aprofundar o estudo sobre ela, pode-se acessar o endereço eletrônico no GitHub.

Circuito

O circuito é basicamente o mesmo circuito utilizado na publicação sobre PWM, mas agora são adicionadas as conexões com o display de LCD em modo de 4 bits. O botão S1 incrementa e botão S2 decrementa o valor do duty cycle. O valor do duty cycle é apresentado na tela do display de LCD e a luminosidade do led é alterada de acordo com este valor. O trimpot ajusta o contraste do backlight do display LCD.

A seguir é apresentado o esquema elétrico e a montagem do circuito no protoboard.

Esquema Elétrico

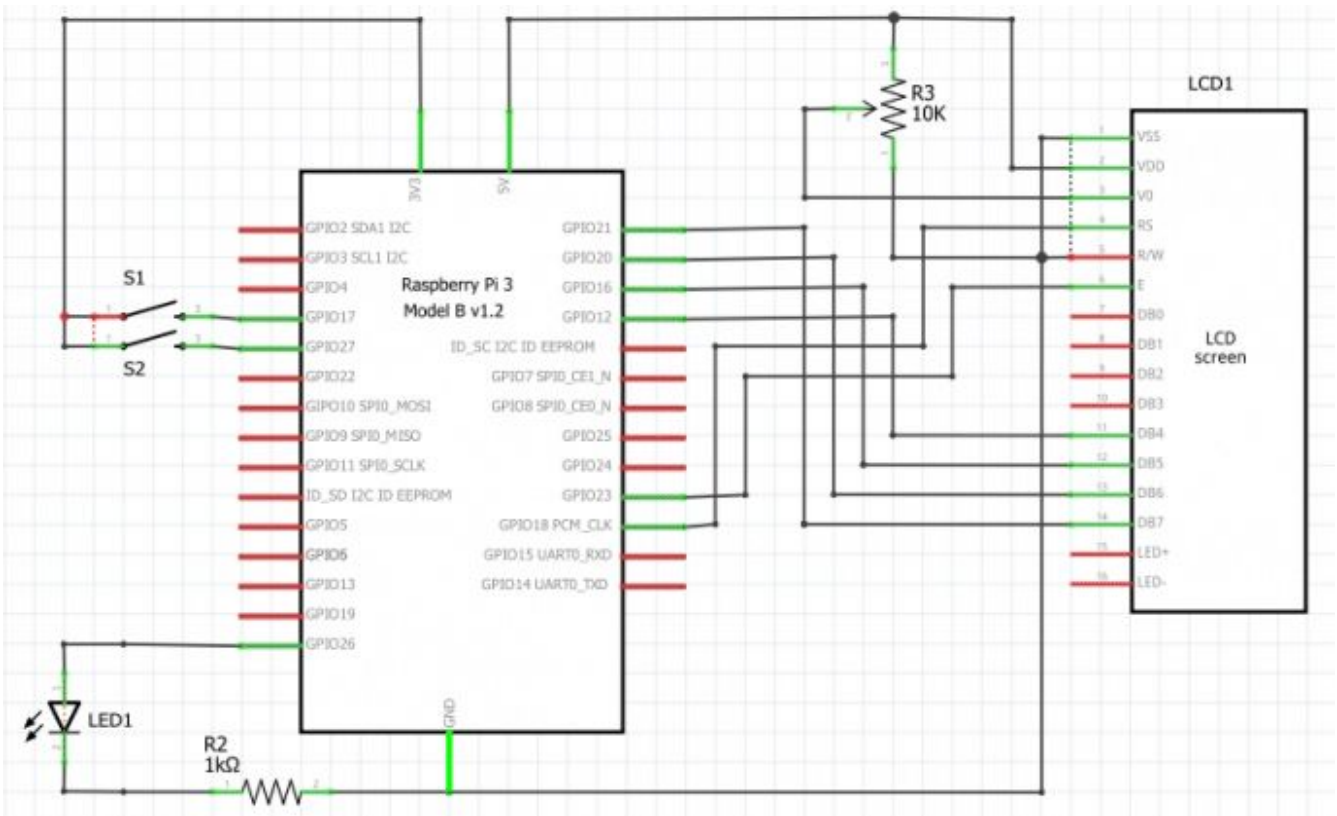


Figura 2 - Esquema elétrico

Montagem

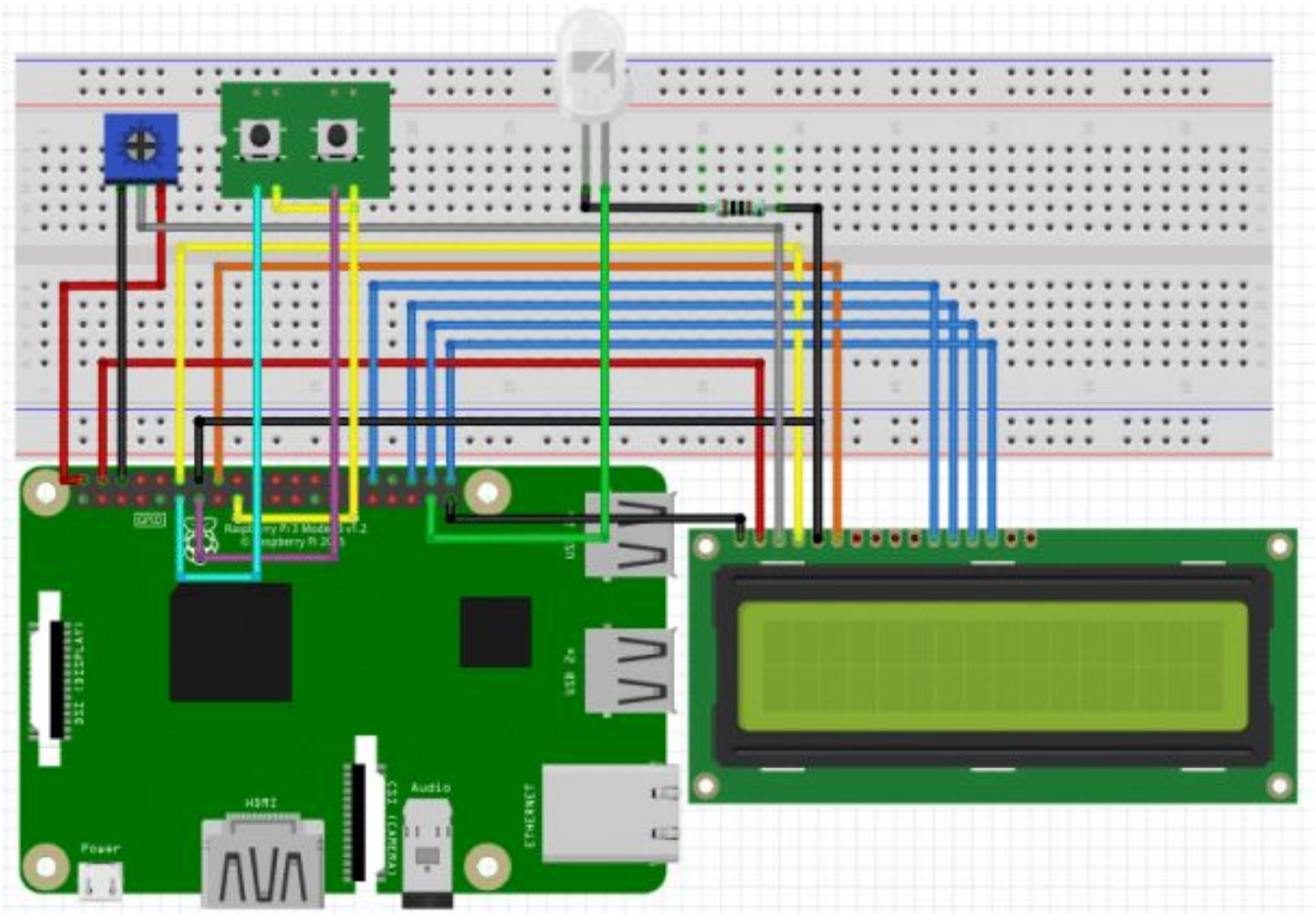


Figura 3 - Montagem no Protoboard

Código

O código deste projeto é similar ao código do artigo anterior, PWM com python, mas neste são acrescentadas funções que realizam a manipulação do display LCD apresentando as informações referentes ao valor no duty cycle, que por consequência realiza alteração na luminosidade do LED.

Deem uma recapitulada nesse artigo, por ser um continuação e para não ficar repetitivo não será realizada superficialmente a descrição do código referente ao artigo anterior. Então vamos lá!

```
# Define Libraries
import RPi.GPIO as gpio
import Adafruit_CharLCD as LCD
import time

# Functions
# This function inserts string and manipulates
# its position on LCD display screen
def screen_control(dc):
    print("Duty Cycle:",dc)
    lcd.clear()
    lcd.set_cursor(3,0)
    lcd.message('PWM CONTROL')
    lcd.set_cursor(0,1)
    lcd.message('Duty Cycle:')
    dc_s = str(dc)
    lcd.set_left_to_right()
    lcd.set_cursor(12,1)
    lcd.message(dc_s)
    if dc < 10:
        lcd.set_cursor(13,1)
    elif dc >= 10 and dc < 100:
        lcd.set_cursor(14,1)
    elif dc == 100:
        lcd.set_cursor(15,1)
    lcd.message('%')
    pwm.ChangeDutyCycle(dc)

    return

# Pinos LCD x Raspberry (GPIO)
lcd_rs        = 18
lcd_en        = 23
lcd_d4        = 12
lcd_d5        = 16
lcd_d6        = 20
lcd_d7        = 21
lcd_backlight = 4
```

```
#Configuring GPIO
gpio.setwarnings(False)
gpio.setmode(gpio.BCM)
gpio.setup(17,gpio.IN, pull_up_down = gpio.PUD_DOWN)
gpio.setup(27,gpio.IN, pull_up_down = gpio.PUD_DOWN)
gpio.setup(26,gpio.OUT)

# Define numero de colunas e linhas do LCD
lcd_colunas = 16
lcd_linhas  = 2

# Configuracao para display 20x4
# lcd_colunas = 20
# lcd_linhas  = 4

# Inicializa o LCD nos pinos configurados acima
lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5,
                           lcd_d6, lcd_d7, lcd_colunas, lcd_linhas,
lcd_backlight)

# Configuring GPIO as PWM
pwm = gpio.PWM(26,100)

# Initializing PWM
dc = 50
pwm.start(dc)

# Print the Initial on LCD display
screen_control(dc)

#Defining the detection in rising edge
gpio.add_event_detect(17,gpio.RISING,bouncetime = 300)
gpio.add_event_detect(27,gpio.RISING,bouncetime = 300)

while True:

    # Increasing the duty cycle of the PWM waveform
    if gpio.event_detected(17):
        dc = dc + 5
        if dc > 100:
            dc = 0
        screen_control(dc)
```

```
  # Decreasing the duty cycle of the PWM waveform
    elif gpio.event_detected(27):
        dc = dc - 5
        if dc < 0:
            dc = 100
        screen_control(dc)

    time.sleep(0.1)

gpio.cleanup()
exit()
```

Entre as linhas 1 e 4 são definidas as bibliotecas de funções que serão utilizadas no código fonte. Observe que na linha 3 é onde foi declarada a biblioteca Adafruit_CharLCD, onde estão definidas as funções que manipulam as informações no LCD.

Da linha 6 até a linha 29, é realizada a declaração da função screen_control(), que realiza a manipulação dos dados na tela do display LCD, posicionando e escrevendo cada informação em uma posição pré-definida. Também é nela que recebe as informações sobre o duty cycle e posiciona o valor na tela de uma forma mais legível.

Entre as linhas 31 e 38 definem as GPIOs que serão utilizadas nas conexões entre a Raspberry Pi e os conectores do display de LCD. Essa definição deve estar compatível com o desejo apresentado no esquema elétrico e na montagem das conexões mostradas nas figuras 2 e 3. Caso haja alguma alteração nas GPIOs, deve-se ficar atento às conexões também.

Da linha 40 à linha 46 são definidas as configuração dos pinos, se serão GPIOs. Sendo assim, as GPIOs 17 e 27 são definidas como entrada, e possuindo um resistor interno de pull-down em cada uma delas e a GPIO 26 como saída.

As linhas 48 e 49 definem que o display LCD será de 16 colunas por 2 linhas. Se ele for utilizar de outra configuração, esses valores devem ser alterados, como é citado nas linhas 52 e 53.

Nas linhas 56, 57 e 58 é definido o objeto chamado “ldc”. Poderia ser dado outro nome, e este recebe os valores definidos entre as linhas 31 e 38.

A linha 61 define que GPIO 26 será a saída PWM com frequencia de 100 Hz.

As linhas 64 e 65 inicializam a saída PWM com um duty cycle em 50% e a linha 68 manipula o valor inicial do PWM para ser mostrado na tela do display LCD.

Entre as linhas 70 e 71 são detectados os eventos de borda de subida das chaves S1 e S2 que são responsáveis pelo incremento e decremento do duty cycle.

Das linhas 74 até a 90 é definido o loop infinito que realiza o incremento e o decremento do duty cycle e manipulam o seu resultado no display LCD através da função screen_control(dc).

E nas linhas 92 e 93, é onde o código é finalizado.

Conclusão

Com as informações obtidas neste artigo o leitor é capaz de desenvolver as suas próprias e inúmeras aplicações utilizando o display de LCD, que é um dispositivo de fácil manipulação e muito utilizado na área de eletrônica.

Continuaremos no próximo artigo utilizando o display de LCD, mas agora para uma aplicação bem interessante, que será um termômetro digital, definindo como se usar o sensor de temperatura e umidade DHT-22.

Este artigo foi escrito por **Roniere Rezende.**

Raspberry PI PWM com Python

EMBARCADOS

Este artigo apresentará os conceitos básicos da Modulação por largura de Pulso, PWM (Pulse Width Modulation) sendo a sua sigla em inglês, e como utilizar e manipular os comandos GPIO da linguagem de programação Python no Raspberry Pi 3.

Ao final da leitura e aplicando os conceitos teóricos e práticos apresentados sobre o PWM na Raspberry, o leitor será capaz de realizar as seus próprios projetos e aplicações.

Este artigo é continuidade aos dois artigos sobre entrada e saída de dados no Raspberry publicado pelo Cleiton Bueno.

O que é PWM?

A Modulação por largura de Pulso ou *Pulse Width Modulation*, é uma maneira de se manipular uma forma de onda quadrada mantendo a frequência constante e se variando intervalo de tempo em que se permanece em nível lógico alto durante o período de frequência.

Existem inúmeras aplicações como, controle de velocidade rotação ou torque em motores DC, controle de luminosidade, controle de atuadores e etc.

A razão entre o tempo em estado lógico alto pelo período da frequência é chamado de *duty cycle*. A equação 1, apresenta essa definição:

$$Duty\ Cycle = \frac{tempo\ em\ nivel\ lógico\ alto}{período\ de\ frequência}$$ Equação 01

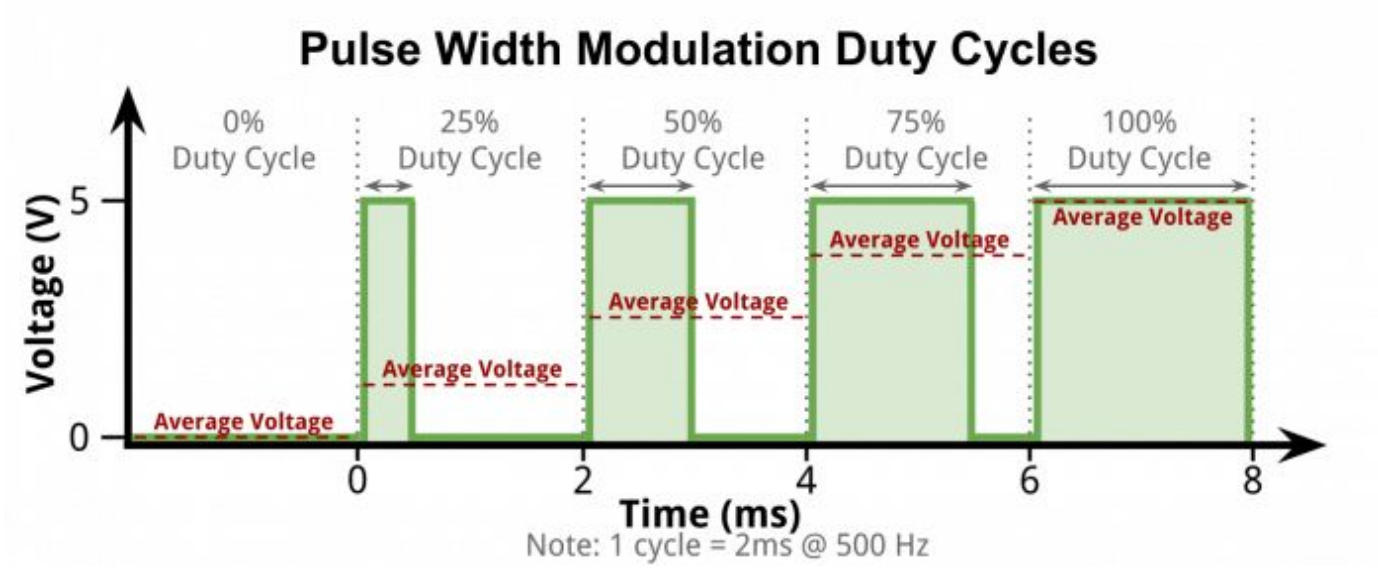


Figura 1 - Duty Cycle no PWM

Vamos analisar a situação apresentado na figura 01 para melhor entendermos o conceito de PWM e o *duty cycle*. A frequência do sinal neste exemplo é 500 Hz e foram fornecidos 5 valores diferentes de *duty cycle*, 0%, 25%, 50%, 75% e 100%, lembrando que o valores podem variar entre 0% e 100%. Vamos realizar os cálculos utilizando a equação 01 e iremos encontrar o valor do tempo em nível lógico alto.

Antes de tudo, deve-se calcular o valor do período da frequência desejada que é dado pela equação 02.

$$Período\ da\ Frequência = \frac{1}{Frequência\ do\ Sinal}$$ Equação 02

Para um sinal com frequência de 500 Hz, temos um período de 2 ms como calculado abaixo:

$$Período\ da\ Frequência = \frac{1}{500} = 0.002\ s = 2\ ms$$

Agora então vamos realizar o cálculo do tempo em nível lógico alto para as cinco situações.

Para *duty cycle* de 0%:

$$0\% = \frac{tempo\ em\ nivel\ lógico\ alto}{2 \times 10^{-3}}$$

$$Tempo\ em\ nivel\ lógico\ alto = 0,00 * 2 \times 10^{-3} = 0\ s$$

Pode-se ver que quando o *duty cycle* é de 0%, o tempo em nível lógico alto é de 0s e comprovando o que é apresentado na Figura 01 no intervalo de tempo anterior a 0 s.

$$25\% = \frac{tempo\ em\ nivel\ lógico\ alto}{2 \times 10^{-3}}$$

$$Tempo\ em\ nivel\ lógico\ alto = 0,25 * 2 \times 10^{-3} = 500 \times 10^{-6}\ s$$

Pode-se ver que quando o *duty cycle* é de 25%, o tempo em nível lógico alto é de 500 us e comprovando o que é apresentado na Figura 01 no intervalo de tempo entre 0 e 2 ms .

Para *duty cycle* de 50%:

$$50\% = \frac{tempo\ em\ nivel\ lógico\ alto}{2 \times 10^{-3}}$$

$$Tempo\ em\ nivel\ lógico\ alto = 0,50 * 2 \times 10^{-3} = 1 \times 10^{-3}\ s$$

Pode-se ver que quando o *duty cycle* é de 50%, o tempo em nível lógico alto é de 1 ms e comprovando o que é apresentado na Figura 01 no intervalo de tempo entre 2 e 4 ms.

Para *duty cycle* de 75%:

$$75\% = \frac{tempo\ em\ nivel\ lógico\ alto}{2 \times 10^{-3}}$$

$$Tempo\ em\ nivel\ lógico\ alto = 0,75 * 2 \times 10^{-3} = 1,5 \times 10^{-3}\ s$$

Pode-se ver que quando o *duty cycle* é de 75%, o tempo em nível lógico alto é de 1,5 mse comprovando o que é apresentado na Figura 01 no intervalo de tempo entre 4 e 6 ms.

Para *duty cycle* de 100%:

$$100\% = \frac{tempo\ em\ nivel\ lógico\ alto}{2 \times 10^{-3}}$$

$$Tempo\ em\ nivel\ lógico\ alto = 1 * 2 \times 10^{-3} = 2 \times 10^{-3}\ s$$

Pode-se se ver que quando o *duty cycle* é de 100%, o tempo em nível lógico alto é de 2 ms, que é o mesmo tempo do período da freqüência, e comprovando o que é apresentado no intervalo de tempo entre 6 e 8 ms.

Comandos Python para PWM na Raspberry Pi

Os comandos Python utilizados para realizar a manipulação PWM da forma de onda quadrada fazem parte da biblioteca RPi.GPIO e devemos usar o comando *import* para poder utilizar esses comandos nos nossos códigos.

Há seguir serão apresentados as funções que nos ajudaram na manipulação PWM da forma de onda:

- pwm =GPIO.PWM(canal, frequência):

Nesta função se cria um objeto chamado “pwm”, mas poderia ser dado qualquer outro nome a este objeto, como poderemos ver no exemplo 01.O argumento “canal” é o pino em que será utilizado como saída PWM e o argumento “frequência” define a frequência do sinal aplicada na saída do pino definido no argumento “canal”.

- pwm.start(DCInicio):

Esta função inicializa o sinal PWM com um valor do *duty cycle*, deevendeo estar entre 0 a 100, respectivamente representando 0% a 100% .

- pwm.ChangeDutyCycle(DC):

Esta função é utilizada durante a execução do código para variar o valor do *duty cycle*entre 0% e 100%. Nos exemplos apresentados mais abaixo, entenderemos melhor o seu uso.

Exemplos

Para ilustrar tudo que foi apresentada até o momento, vamos apresentar dois códigos simples desenvolvidos que se utilizam dos conceitos e definições da PWM e como as funções da biblioteca RPi.GPIO do Python são aplicadas no decorrer do código. Vamos então a eles:

Exemplo 01

Neste exemplo é realizada a variação da luminosidade de dois leds, um vermelho e outro azul, conectados na saídas dos terminais do Raspberry Pi 3. Vamos então analisar o código passo-a -passo para melhor entendimento do que está sendo realizando no código Python.

Circuito

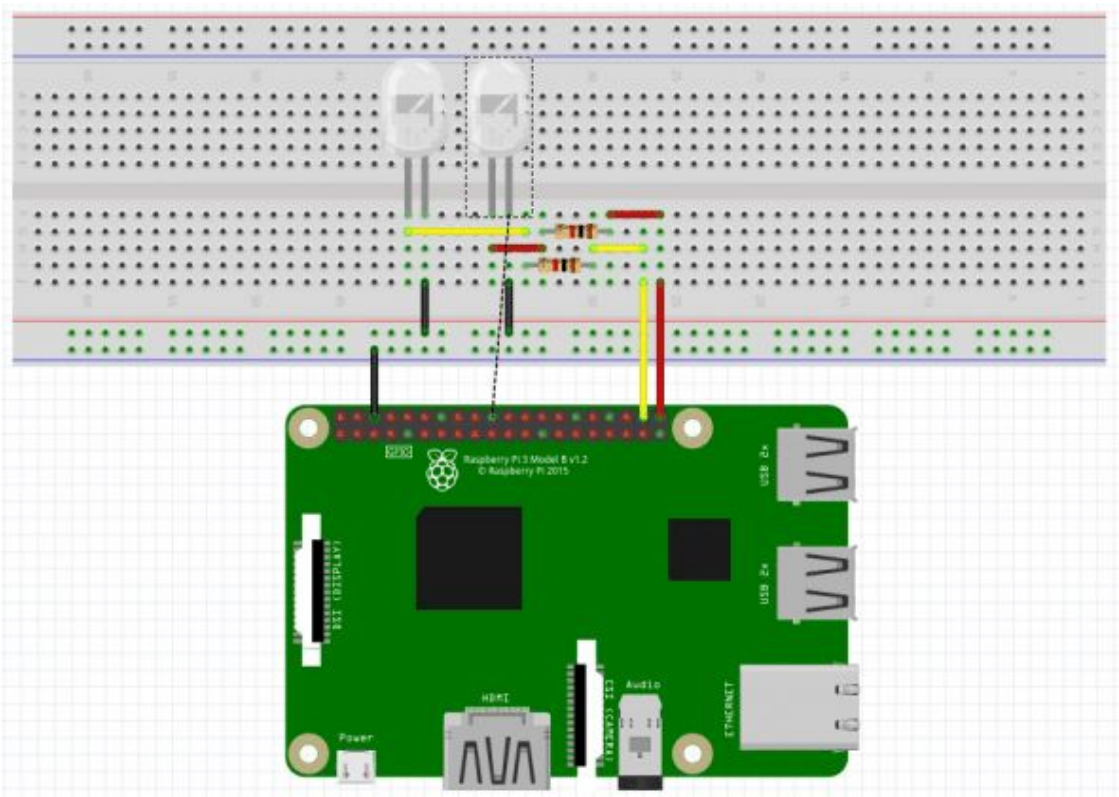


Figura 02 - Circuito Exemplo 01

O circuito acima apresenta as conexões a serem realizadas em um protoboard para que o circuito definido no exemplo 02 funcione corretamente. Os dois resistores de 1 kΩ em das suas extremidades conectadas respectivamente ao pino 38 e 40 da Raspberry Pi 3. Nas duas extremidades cada uma está conectada ao anodo do led. Os catodos dos led estão conectados ao pino 6 de GND Raspberry Pi.

Código Python

```
#Define Libraries
import RPi.GPIO as gpio
import time

#Configuring don't show warnings
gpio.setwarnings(False)

#Configuring GPIO
pio.setmode(gpio.BOARD)
gpio.setup(38,gpio.OUT)
gpio.setup(40,gpio.OUT)

#Configure the pwm objects and initialize its value
pwmBlue = gpio.PWM(38,100)
pwmBlue.start(0)

pwmRed = gpio.PWM(40,100)
pwmRed.start(100)
#Create the dutycycle variables
dcBlue = 0
dcRed  = 100

#Loop infinite
while True:

    #increment gradually the luminosity
    pwmBlue.ChangeDutyCycle(dcBlue)
    time.sleep(0.05)
    dcBlue = dcBlue + 1
    if dcBlue == 100:
        dcBlue = 0

    #decrement gradually the luminosity
    pwmRed.ChangeDutyCycle(dcRed)
    time.sleep(0.05)
    dcRed = dcRed - 1
    if dcRed == 0:
        dcRed = 100

#End code
gpio.cleanup()
exit()
```

As linhas 2 e 3, importam para o código as bibliotecas que permitem o uso das funções manipulação das GPIOs e de tempo.

A linha 6, faz que não se mostre os *warnings* no *shell* do Python.

As linhas 9, 10 e 11 definem respectivamente que configuração dos pinos se dará pelo números destes e que os pinos 38 e 40 serão saídas.

As linhas 14 e 17 criam dois objetos chamados pwmBlue e pwmRed, define que eles irão trabalhar na frequência de 100 Hz e respectivamente definem osn seus canais sendo os pinos 38 e 40.

As linhas 15 e 18 definem os valores iniciais do *duty cycle* dos objetos pwmBlue como 0% e pwmRed como 100%.

As linhas 21 e 2 definem as variáveis dcBlue e dcRed recebendo respectivamente os valores 0% e 100%.

Alinha 25 defini o *loop* infinito.

Entre as linhas 28 e 32 são executados os comandos que realizam ativam e incrementam o *duty cycle* do pino 38, e quando se alcança o valor máximo este é retornado a 0%. Isso faz com que o led azul aumente a sua luminosidade gradualmente.

Entre as linhas 35 e 39 são executados os comandos que realizam ativam e decrementam o *duty cycle* do pino 40, e quando se alcança o valor mínimo este é retornado a 100%. Isso faz com que o led vermelho diminua a sua luminosidade gradualmente.

As linhas 43 e 44 finalizam o código.

Exemplo 02

Neste exemplo, o controle do *duty cycle* é realizado por duas chaves, uma para incrementar e outra decrementar, e este representado por led que tem a sua luminosidade alterada de acordo com o seu valor. Vamos então analisar o código passo-a -passo para melhor entendimento do que está sendo realizando no código Python.

Circuito

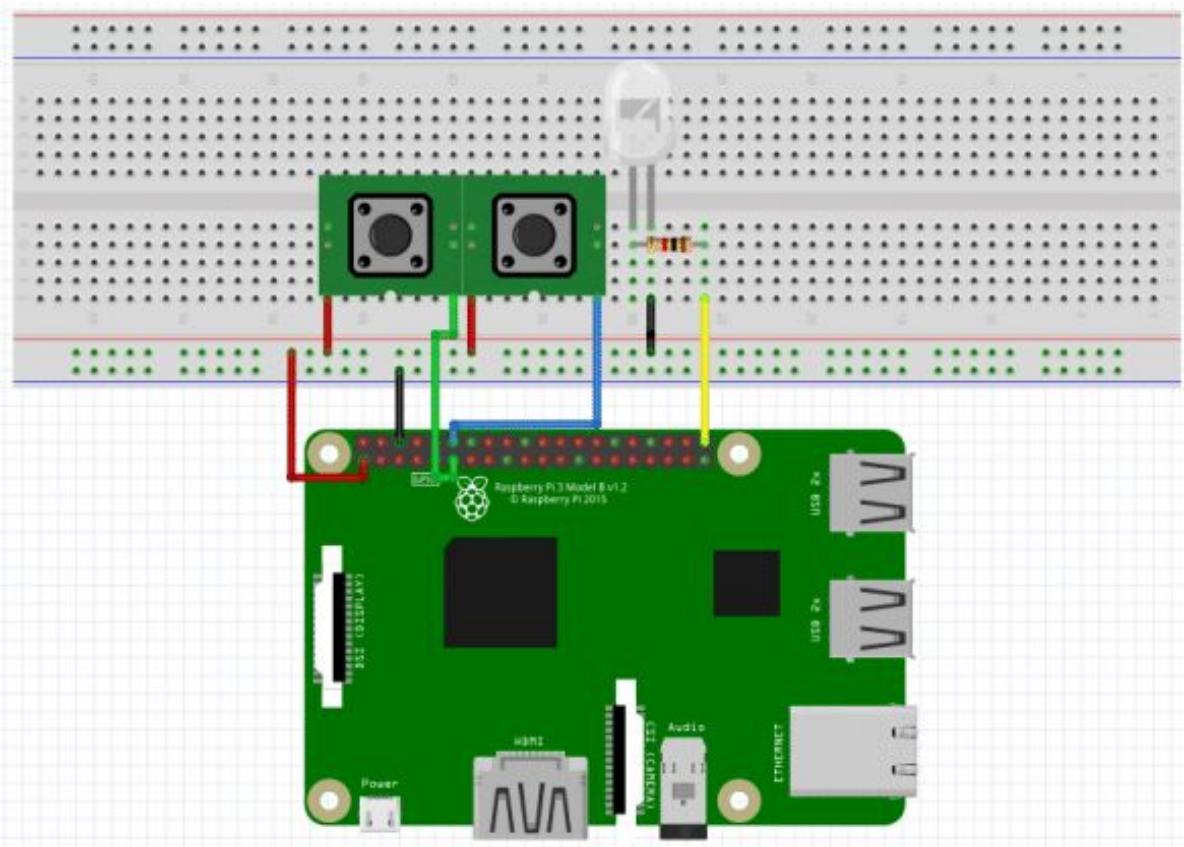


Figura 03 - Circuito Exemplo 02

O circuito acima apresenta as conexões a serem realizadas em um protoboard para que o circuito definido no exemplo 02 funcione corretamente. As duas chaves push-button tem um dos seus terminais conectados ao pino 1 que fornece a tensão de 3,3 V e o outro terminal de cada uma deve ser conectados respectivamente nos pinos 11 e 12. No pino 40 é conectado um dos terminais do resistor de 1 kΩ e no outro terminal deve ser conectado ao terminal do anodo do led. O terminal cátodo do led deve ser conectado ao terminal do pino 6 que é o GND da Raspberry.

Código Python

```
#Define Libraries
import RPi.GPIO as gpio
import time

#Configuring GPIO
gpio.setwarnings(False)
gpio.setmode(gpio.BOARD)
gpio.setup(11,gpio.IN, pull_up_down = gpio.PUD_DOWN)
gpio.setup(12,gpio.IN, pull_up_down = gpio.PUD_DOWN)
gpio.setup(40,gpio.OUT)

#Configuring GPIO as PWM
pwm = gpio.PWM(40,100)

#Initializing PWM
pwm.start(50)
dc = 50
print("Duty Cycle:",dc)
pwm.ChangeDutyCycle(dc)

#Defining the detection in rising edge
gpio.add_event_detect(11,gpio.RISING,bouncetime = 300)
gpio.add_event_detect(12,gpio.RISING,bouncetime = 300)

while True:
```

```
# Increasing the duty cycle of the PWM waveform
    if gpio.event_detected(11):
        dc = dc + 10
        if dc == 110:
            dc = 0
        print("Duty Cycle:",dc)
        pwm.ChangeDutyCycle(dc)

    # Decreasing the duty cycle of the PWM waveform
    elif gpio.event_detected(12):
        dc = dc - 10
        if dc == -10:
            dc = 100
        print("Duty Cycle:",dc)
        pwm.ChangeDutyCycle(dc)

    time.sleep(0.1)

gpio.cleanup()
exit()
```

As linhas 2 e 3 importam para o código as bibliotecas que permitem o uso das funções manipulação das GPIOs e de tempo.

Entre as linhas 6 e 10 são definidos que os pinos 11 e 12 serão entradas e utilizam um resistor de *pull-down* interno e que o pino 40 é uma saída.

A linha 13 é definido o objeto “pwm” e também que o seu canal será o pino 40 e a frequência será de 100 Hz.

Na linha 16 define que *duty cycle* iniciará em 50%.

Entre as linhas 17 e 19, é criada uma variável chamada “dc” que recebe o valor 50 representando o valor de *duty cycle* de 50% e esse valor é mostrada na tela do *shell* do Python. E este valor é aplicado na pino 40 onde está conectado o led.

Entre as linhas 22 e 23 definem que os pinos 11 e 12 interpretem como nível lógico alto uma borda de subida e é aplicado um *debounce* de 300 ms.

A linha 25 implementa o *loop* infinito.

Entre as linhas 28 e 33 é interpretado se o evento da borda de subida no pino 11 aconteceu, sendo verdadeiro, a variável "dc" é incrementada em 10 unidades causando um aumento no *duty cycle* de 10%, também é mostrado no *shell* essa alteração e consequentemente a luminosidade do led aumenta. Caso o valor do variável "dc" ultrapasse o valor 100, ela é recebe ao valor 0.

Entre as linhas 36 e 41 é interpretado se o evento da borda de subida no pino 12 aconteceu, sendo verdadeiro, a variável "dc" é decrementada em 10 unidades causando um diminuição no *duty cycle* de 10%, também é mostrado no *shell* essa alteração e consequentemente a luminosidade do led aumenta. Caso o valor do variável "dc" seja menor o valor 0, ela é recebe ao valor 100.

A linha 43 causa um atraso de 100 ms segundo na execução do próxima iteração do *loop* infinito.

As linhas 45 e 46 finalizam o código.

Conclusão

Pode-se observar que a PWM é uma aplicação que é utilizada para inúmeras aplicações e que as funções contidas na biblioteca Python para o Raspberry Pi 3 são simples de serem manipuladas e aplicadas, o que pode ser observado pelos exemplos de códigos apresentados neste artigo.

Este artigo foi escrito por **Roniere Rezende.**

Raspberry Pi Projeto Termômetro Digital com Python - Teoria

EMBARCADOS

Este artigo apresentará a parte teórica do desenvolvimento de um projeto prático de um **termômetro digital** com o código fonte escrito em **Python**. A parte prática será oferecida no próximo artigo. Os dados de temperatura e umidade relativa do ar serão obtidos, codificados e transmitidos para a Raspberry Pi pelo módulo **AM2302**.

Será realizada uma breve definição e explicação da forma de se baixar e instalar a biblioteca Python dos sensores **DHT-11** e **DHT-22**, além de ensinar a realizar a sua conexão para transmitir as informações corretamente à Raspberry Pi.

Ao final da leitura deste artigo, aplicando os conceitos teóricos e práticos apresentados sobre o projeto do termômetro digital, o leitor será capaz de realizar a manipulação dos dados gerados pelo módulo AM2302 e também poderá, com a sua criatividade, expandir e melhorar o projeto apresentado e criar os seus próprios projetos.
Este artigo dá continuidade à série de artigos sobre Raspberry Pi com Python, escrita pelo Cleiton Bueno e por mim, Roniere Rezende.

Termômetro Digital

Para o desenvolvimento do projeto de aplicação do termômetro digital serão necessários os seguintes componentes:

- 1 Raspberry Pi;
- 1 Sensor AM2302 da AOSONG;
- 1 Display de LCD de 16 colunas e 2 linhas;
- 2 botões de *push-botton*.

O módulo sensor de temperatura e umidade AM2302 obtém os dados, os processa e transmite à Raspberry Pi. Esta manipula e envia os dados recebidos para serem apresentados na tela do display de LCD. A definição destas funções são apresentadas neste artigo. Dois botões *push-botton* são utilizados para apresentar na tela do display de LCD os valores de temperatura máxima e mínima registradas durante o período de análise da medição, um para cada registro.

A seguir, na Figura 1, são apresentadas as telas no display de LCD que serão desenvolvidas neste artigo.

Tela de Medição

Tela de Temperatura Máxima

Tela de Temperatura Mínima

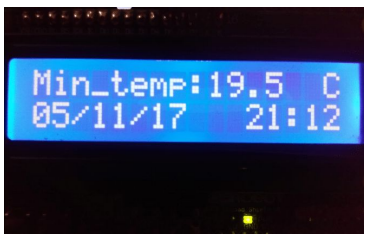


Figura 1 - Telas do Sensor de Temperatura

Sensor de Temperatura e Umidade

O dispositivo utilizado no projeto do termômetro digital para realizar as medidas de temperatura e umidade é o módulo sensor **AM2302** da AOSONG. Veja o seu datasheet aqui.

O módulo digital AM2302 é um módulo composto que realiza a medição, digitalização das variáveis físicas de temperatura e umidade e disponibiliza essas informações no seu terminal de dados. Tem baixo consumo e pode transmitir os dados com qualidade até 20 mts de distância do receptor. Realiza a calibração completamente automática, faz uso de sensor capacitivo de umidade, saída digital de barramento simples, apresenta excelente estabilidade por longo tempo e alta exatidão na medida de temperatura. A Figura 2 mostra módulo sensor AM2302.

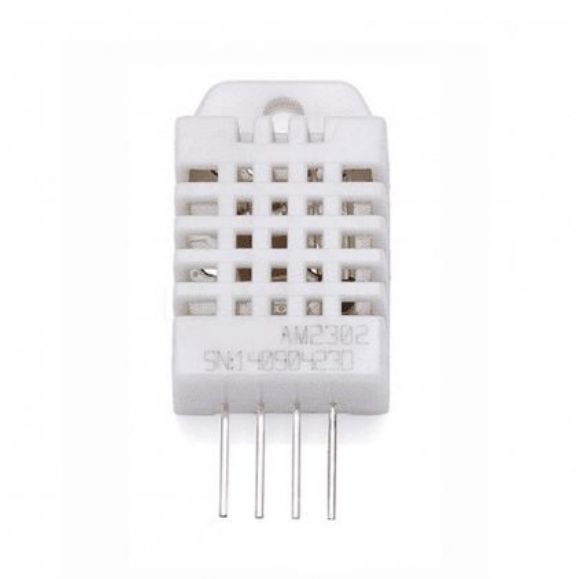


Figura 2 - Módulo AM2302

As aplicações para o módulo AM2302 são para aquecimento, ventilação e ar condicionado (sigla em inglês HVAC), desumidificador, equipamento de teste e inspeção, bens de consumo, automotivo, controle automático, registradores de dados, aplicações caseiras, regulador de umidade, equipamentos médicos, estações climáticas, controle e medida de umidade e, mais recentemente, pode-se utiliza-lo em aplicações IoT.

A pinagem do módulo AM2302 é definida na tabela abaixo:

Tabela 1 - Pinagem do módulo AM2302

Pino	Nome	Descrição
1	VDD	Alimentação (3,3V até 5,5V)
2	SDA	Porta bidirecional de dados seriais
3	NC	Não utilizável
4	GND	Referência de terra

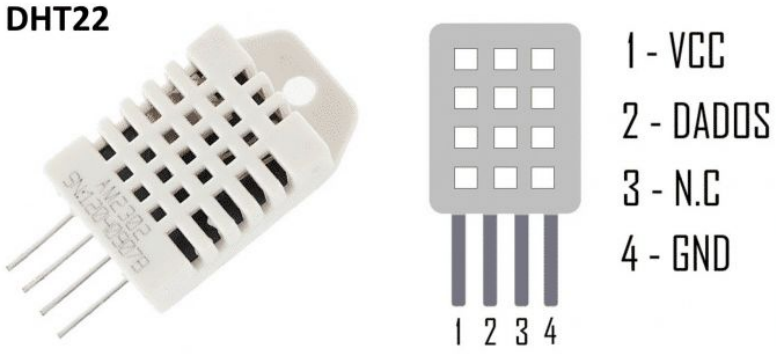


Figura 3 - Pinagem do módulo AM2302

Como pode ser visto na tabela 1, a alimentação do módulo AM2302 deve ser entre 3,3 V até 5V, sendo 5V o valor recomendado no *datasheet*. O pino SDA é uma estrutura de leitura e escrita de dados. Detalhes sobre o protocolo de comunicação serão definidos mais abaixo.

A seguir serão apresentadas as tabelas 2 e 3 com as características de medição de temperatura e umidade do módulo AM2302.

Tabela 2 - Desempenho de Umidade Relativa do AM2302

Parâmetros	Condições	Mínimo	Típico	Máximo	Unidade
Resolução			0,1		%RH
Faixa		0		99,9	%RH
Precisão	25°		±2		%RH
Repetibilidade			±0,3		%RH
Intercâmbio	Completamento Intercambiável				
Resposta	1/e(63%)		< 5		%RH
Lentidão			< 0,3		%RH
Escorregamento	Típico		< 0,5		%RH

Tabela 3 - Desempenho de Temperatura Relativa do AM2302

Parâmetros	Condições	Mínimo	Típico	Máximo	Unidade
Resolução			0,1		°C
			16		bit
Faixa		-40		80	°C
Precisão			±0,5	±1	°C
Repetibilidade			±0,2		°C
Intercâmbio	Completamento Intercambiável				
Resposta	1/e(63%)		<10		S
Escorregamento			±0,3		°C/yr

As características elétricas, tais como consumo, nível de tensão em alto e em baixa, e tensão de entrada e saída dependem da alimentação. Na tabela 4 são apresentadas essas características.

Tabela 4 - Características DC do Módulo 2302

Parâmetros	Condições	Mínimo	Típico	Máximo	Unidade
Tensão		3,3	5	5,5	V
Consumo de potência	Inativo	10	15		µA
	Medindo		500		µA
	Média		300		µA
Tensão em Nível baixo de Saída	IOL	0		300	mV
Tensão Alta de Saída	Rp < 25 kΩ	90%		100%	VDD
Tensão Baixa de Entrada	Descida	0		30%	VDD
Tensão Alta de Entrada	Subida	70%		100%	VDD
Rpu	VDD = 5V VIN = VSS	30	45	60	kΩ
Corrente de Saída	Ligado		8		mA
	Desligado	10	20		µA
Periodo de Amostragem		2			S

Para se realizar a comunicação entre o módulo AM2302 e a Raspberry Pi devem ser seguidas algumas recomendações, citadas no seu *datasheet*. Elas são listadas a seguir também:

- Para circuitos de aplicações típicas é recomendado o comprimento do cabo de 30 metros e o uso de um resistor de *pull-up* de 5,1 kΩ, conectado entre o terminal de Vcc e o barramento de dados;
- Utilizando uma alimentação de 3,3 V, o comprimento do cabo deve ser menor que 1 metro;
- O intervalo de leitura do sensor deve ser 2 segundos. Com um tempo menor que este, a leitura das medidas de temperatura e umidade podem não ser realizadas com sucesso;
- Os valores de temperatura e umidade são resultados da última medida realizada. Para dados em tempo-real, que precisam ser lidos continuadamente, é recomendado ler o sensor repetidamente com um intervalo de leitura maior que 2 segundos para obter resultados exatos.

O protocolo de barramento simples no AM2302 utiliza somente um terminal de dados com o sistema para a trocas de dados. A definição de como se realiza a comunicação é demonstrada na figura 4.

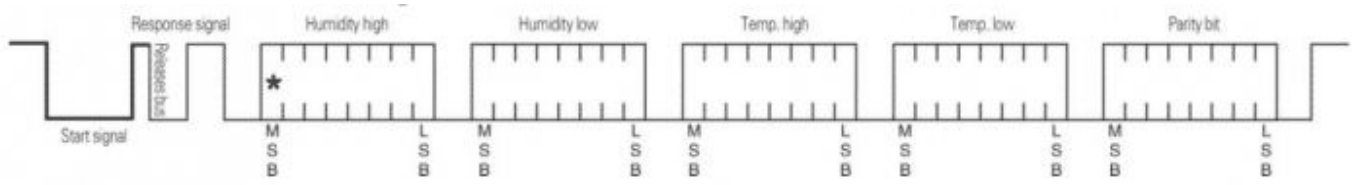


Figura 4 - Protocolo de Comunicação de Barramento Simples

O SDA (*Serial Data Line*) é utilizado para comunicação e sincronização entre a Raspberry Pi e o AM2302 e o formato de dados do barramento simples é formado por 40 bits, como mostrado na Figura 3. A sua descrição é apresentada na Tabela 5.

Tabela 05 - Definição do formato de comunicação do AM2302

Nome	Definição do Formato do Barramento Único
Sinal Inicial	Barramento de dados microcontrolador (SDA) deixa o sinal em nível baixo lógico baixo por um período de tempo (pelo menos 800 µs) notificando o sensor para preparar os dados.
Sinal de Resposta	O barramento de dados do sensor (SDA) é colocado em nível baixo por 80 µs e depois é colocado em nível lógico alto por 80 µs para receber o sinal de início.
Formato de dados	O receptor recebe sinal de início, o sensor envie a string de dados de 40 bits e termina deixando a saída em nível lógico alto.
Umidade	Resolução da umidade de 16 bits, o anterior em nível lógico alto, o valor da string do sensor de umidade é 10 vezes os valores atuais de umidade.
Temperatura	Resolução da temperatura de 16 bits, o anterior em nível lógico alto, o valor da string do sensor de temperatura é 10 vezes os valores atuais de temperatura. O nível lógico do bit mais significativo (bit 15) é igual a 1 para indicar temperatura negativa e será 0 para indicar temperatura positiva. O valor de temperatura é indicado entre os bits 0 e 14.
Bit de Paridade	Bit de paridade = umidade em alta + umidade em baixa + temperatura em alta + temperatura em baixa

A temporização da comunicação do barramento simples é realizada da seguinte forma. O dispositivo do usuário envia um sinal de início (o barramento de dados permanece em nível lógico baixo por pelo menos 800 µs), depois o AM2302 passa do Modo de Inatividade para o Modo de Alta Velocidade. O dispositivo do usuário indica ao módulo AM2302 enviando-lhe um sinal de resposta através do barramento de dados, no caso um byte em nível lógico alto. O módulo envia os dados (umidade alta, umidade baixa, temperatura alta, temperatura baixa e bit de paridade) ao final da coleta de informações de *trigger*, e no fim do processo o sensor é automaticamente transferido para o Modo de Inatividade, esperando até a próxima comunicação.

Na Figura 5 são apresentadas as características do sinal de temporização do módulo AM2302.

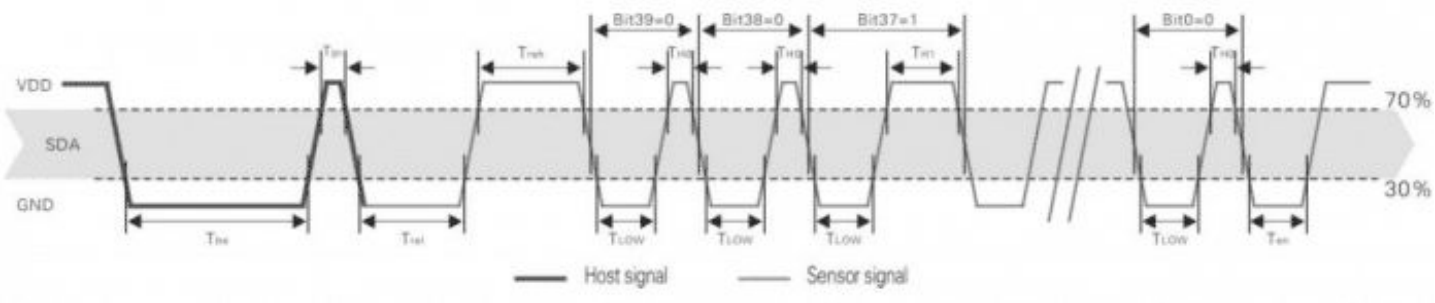


Figura 5 - Temporização do Barramento Único de Comunicação

Tabela 06 - Características do sinal do barramento único

Símbolo	Parâmetros	Mínimo	Típico	Máximo	Unidade
Tbe	Tempo de inatividade sinal inicial	0,8	1	20	ms
Tgo	Tempo de liberação do barramento	20	30	200	µs
Trel	Resposta ao tempo em baixa	75	80	85	µs
Treh	Resposta ao tempo em alta	75	80	85	µs
TLOW	Sinal “0”, tempo em baixo “1”	48	50	55	µs
TH0	Sinal “0” tempo em alta	22	26	30	µs
TH1	Sinal “1” tempo alto	68	70	75	µs
Ten	Tempo ao liberar o tempo do barramento	45	50	55	µs

Na próxima parte do artigo será apresenta a instalação da biblioteca do módulo DHT-22, como realizar a comunicação entre a Raspberry Pi e o módulo por meio de comando, o circuito/montagem realizado e o código. Não percam o próximo artigo!

Este artigo foi escrito por **Roniere Rezende.**

Raspberry Pi Projeto Termômetro Digital com Python - Prática

EMBARCADOS

Continuando o artigo anterior da série, trazemos agora a parte prática do projeto, apresentando a instalação da biblioteca do módulo DHT-22, a comunicação entre a Raspberry Pi e o módulo por meio de comando, o circuito/montagem realizado e o código.

Instalação da Biblioteca do Módulo DHT-22

No projeto do termômetro digital é utilizado o módulo **AM2302**, similar ao módulo **DHT-22**. Por este motivo é utilizada a **biblioteca Python do Módulo DHT-11/22**.

Para a instalação dessa biblioteca, deve-se primeiramente conectar a Raspberry Pi à internet. Em seguida, abre-se o LX Terminal e executa-se os comandos a seguir:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
```

Sem seguida, acesse a pasta *Adafruit_Python_DHT* com o comando:

```
cd Adafruit_Python_DHT
```

Deve-se atualizar o Raspbian e baixar o python-dev para que a biblioteca funcione corretamente. Execute os comandos abaixo:

```
sudo apt-get update
sudo apt-get install build-essential python-dev
```

Agora é o momento de instalar a biblioteca, portanto execute o comando a seguir:

```
Sudo python setup.py install
```

Comandos

Os comandos Python utilizados para realizar a comunicação entre a Raspberry Pi e o módulo AM2302 fazem parte da biblioteca *Adafruit_Python_DHT-22* e devemos usar o comando *import* para poder utilizar esses comandos nos nossos códigos. Como utilizar o comando *import* é demonstrado na seção "Código".

A função para se realizar a leitura dos dados de temperatura e umidade é mostrada abaixo. Maiores informações podem ser obtidas na referência da biblioteca.

```
umid, temp = Adafruit_DHT.read_retry(sensor, pino_sensor);
```

As duas próximas declarações definem qual o modelo de sensor irá ser usado no código.

```
sensor = Adafruit_DHT.DHT11
sensor = Adafruit_DHT.DHT22
```

Para um melhor entendimento dos comandos referentes ao display de LCD, recomendo a leitura do artigo publicado sobre este assunto, aqui.

Circuito

A montagem do circuito do termômetro digital é similar ao mostrado no artigo anterior sobre o display de LCD, agora é acrescentado o módulo AM2302. A pinagem desse módulo pode ser vista na Figura 1. Para esta montagem, o módulo é alimentado por uma tensão de 3,3V para se evitar que a tensão no barramento de dados supere a máxima tensão suportada pelas portas GPIO da Raspberry Pi e, por consequência, as danifique. Um resistores de pull-up é colocado entre o VCC e o barramento de dados do módulo AM2302 para mantê-lo em nível lógico alto quando não estiver sendo utilizado o barramento. Para se evitar ruídos e interferências na medição do módulo 2302, é colocado um capacitor cerâmico de 100 nF mais próximo possível dos terminais de VCC e GND do módulo AM2302. Os dois botões presentes na montagem são utilizados para que se possa visualizar a máxima e a mínima temperatura registrada durante o período de medição que são pressionados.

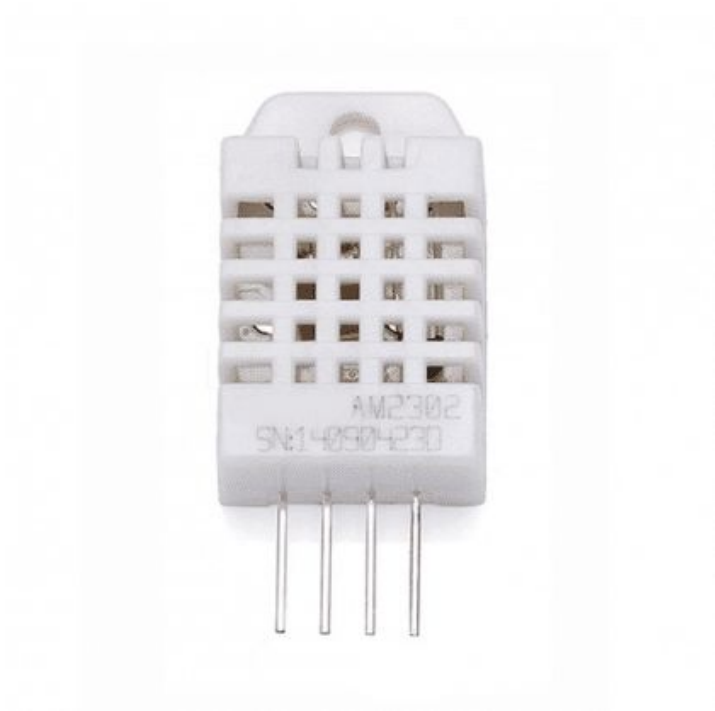


Figura 1 - Módulo AM2302

Esquema Elétrico

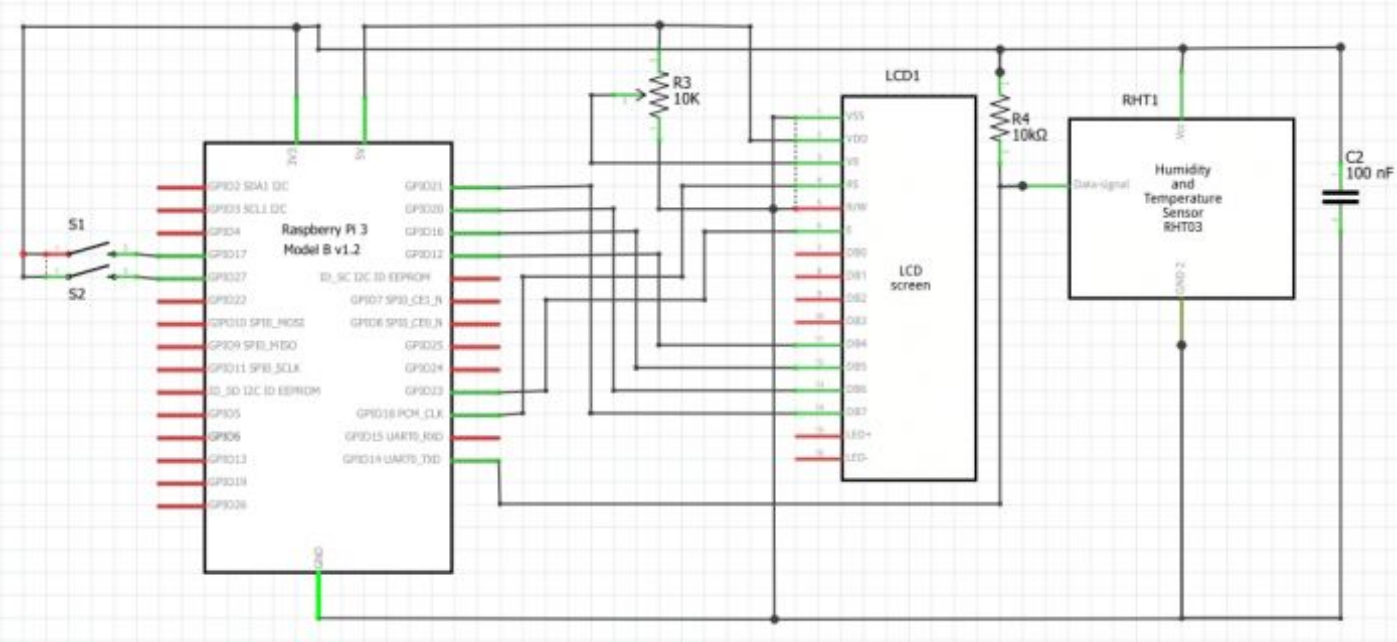


Figura 2 - Esquema Elétrico do Projeto Termômetro com Python

Montagem

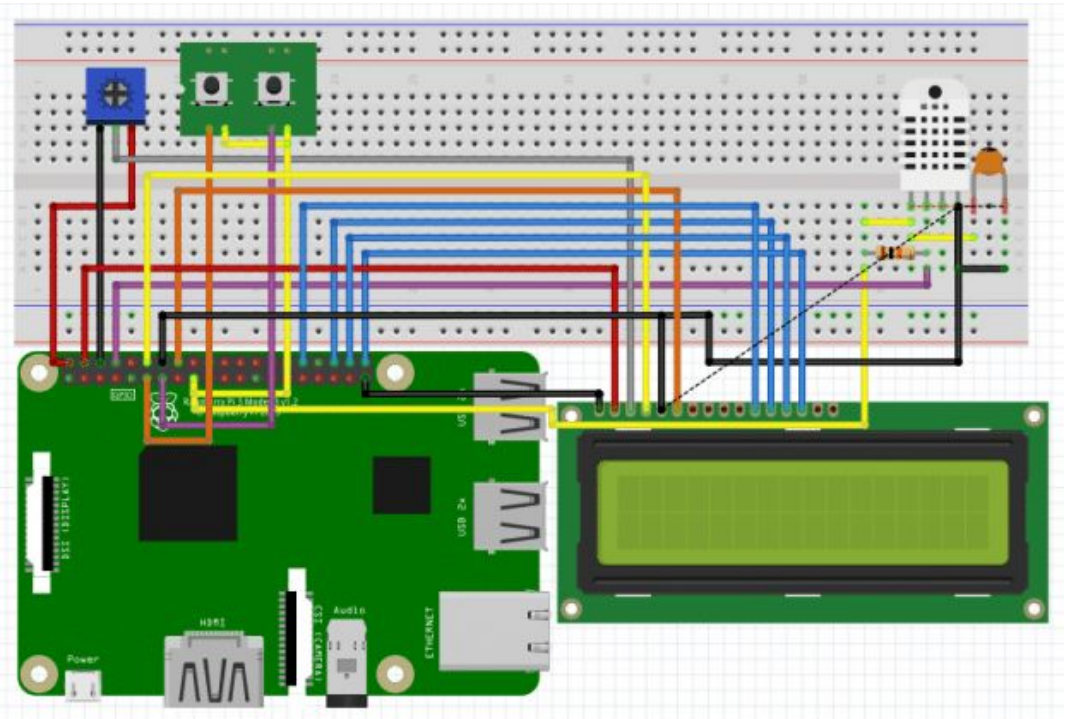


Figura 3 - Montagem no Protoboard do Projeto Termômetro com Python

Código

O código em linguagem Python descrito abaixo faz com que os dados de temperatura e umidade relativa do ar obtidos do módulo AM2302 sejam interpretados pela Raspberry Pi e ela mostre as informações na tela do display de LCD. O código também faz o registro de máxima e mínima temperaturas obtidas durante o intervalo de medição, apresentando o instante que o evento ocorreu.

Aconselho para um bom entendimento deste código em Python dar uma recapitulada nos artigos anteriores desta série. Então vamos lá.

```
# Carrega as bibliotecas
from datetime import datetime
import Adafruit_CharLCD as LCD
import RPi.GPIO as GPIO
import Adafruit_DHT
import time

 ## Define o tipo de sensor
# sensor = Adafruit_DHT.DHT11
sensor = Adafruit_DHT.DHT22


# Pinos LCD x Raspberry (GPIO)
lcd_rs        = 18
lcd_en        = 23
lcd_d4        = 12
lcd_d5        = 16
lcd_d6        = 20
lcd_d7        = 21
lcd_backlight = 4


# Pino de entrada de dados do DHT-22
DHT_IN  = 14
KEY_MAX =  4
KEY_MIN = 17
```

```
# Desabilita os Warnings
GPIO.setwarnings(False)


# Configura o modo de utilização das portas definidas pelas GPIOs

GPIO.setmode(GPIO.BCM)


# Define o pino conectado ao push-button
GPIO.setup(DHT_IN, GPIO.IN)
GPIO.setup(KEY_MAX, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
GPIO.setup(KEY_MIN, GPIO.IN, pull_up_down = GPIO.PUD_DOWN)


# Define número de colunas e linhas do LCD
lcd_colunas = 16
lcd_linhas  = 2


# Define variables of maximum and minimum temperature
max_temp = 0
min_temp = 100


# Inicializa o LCD nos pinos configurados acima
lcd = LCD.Adafruit_CharLCD(lcd_rs, lcd_en, lcd_d4, lcd_d5,
                           lcd_d6, lcd_d7, lcd_colunas, lcd_linhas,
                           lcd_backlight)


while True:
    #Variáveis que recebem os valores de data e hora
    tdy = datetime.today()
    now = datetime.now()
    time.sleep(0.1)
    #Variáveis que recebem os valores de umidade e temperatura
    umid, temp = Adafruit_DHT.read_retry(sensor, DHT_IN);

    #Testa de os dados são válidos. Se sim, executa o código
    if umid is not None and temp is not None:
        #Configura os dados de temperatura no LCD
        lcd.clear()
        lcd.set_cursor(0,0)
        lcd.message('Temp:')
        temp_s = str(temp)
        lcd.set_cursor(9,0)
        lcd.message(temp_s.ljust(5)[:5])
        lcd.set_cursor(15,0)
        lcd.message('C')
```

```
#Configura os dados de umidade no LCD
        lcd.set_cursor(0,1)
        lcd.message('Humid:')
        umid_s = str(umid)
        lcd.set_cursor(9,1)
        lcd.message(umid_s.ljust(5)[:5])
        lcd.set_cursor(15,1)
        lcd.message('%')

# Salva os dados do maior valor de temperatura
        if temp > max_temp:
            max_temp   = temp
            year_max    = str((datetime.now().year)-2000)
            month_max = str(datetime.now().month)
            day_max       = str(datetime.now().day)
            hour_max    = str(datetime.now().hour)
            minute_max = str(datetime.now().minute)


        # Salva os dados do menor valor de temperatura
        if temp < min_temp:
            min_temp   = temp
            year_min   = str((datetime.now().year)- 2000)
            month_min  = str(datetime.now().month)
            day_min    = str(datetime.now().day)
            hour_min   = str(datetime.now().hour)
            minute_min = str(datetime.now().minute)


        # Acessa os dados salvos da temperatura máxima aperta um botão
        if GPIO.input(KEY_MAX) == GPIO.HIGH:
            flag = False
            while True:
                if GPIO.input(KEY_MAX) == GPIO.LOW:
                    flag = True
                lcd.clear()
                max_temp_s = str(max_temp)
                lcd.set_cursor(0,0)
                lcd.message('Max_temp:')
                lcd.set_cursor(9,0)
                lcd.message(max_temp_s.ljust(5)[:5])
                lcd.set_cursor(15,0)
                lcd.message('C')
```

```
# Define a posição dos dados do dia que ocorreu o evento
                # o máximo valor na tela
                if int(day_max) < 10:
                    lcd.set_cursor(0,1)
                    lcd.message('0')
                    lcd.set_cursor(1,1)
                    lcd.message(day_max)
                else:
                    lcd.set_cursor(0,1)
                    lcd.message(day_max)

                lcd.set_cursor(2,1)
                lcd.message('/')

                # Define a posição dos dados do mês que ocorreu o evento
                # o máximo valor na tela
                if int(month_max) < 10:
                    lcd.set_cursor(3,1)
                    lcd.message('0')
                    lcd.set_cursor(4,1)
                    lcd.message(day_max)
                else:
                    lcd.set_cursor(3,1)
                    lcd.message(month_max)


                # Configura a posição do ano e hora do valor máximo no display de LCD
                lcd.set_cursor(5,1)
                lcd.message('/')
                lcd.set_cursor(6,1)
                lcd.message(year_max)
                lcd.set_cursor(11,1)
                lcd.message(hour_max)
                lcd.set_cursor(13,1)
                lcd.message(':')

 # Define a posição na tela dos dados do minuto que ocorreu o evento
                if minute_max < 10:
                    lcd.set_cursor(14,1)
                    lcd.message('0')
                    lcd.set_cursor(15,1)
                    lcd.message(minute_max)
                else:
                    lcd.set_cursor(14,1)
                    lcd.message(minute_max)
```

```
# Sai do loop infinito de temperatura máxima
                if ((GPIO.input(KEY_MAX) == GPIO.HIGH) and flag == True) == True:
                    break
                time.sleep(0.5)

        # Acessa os dados salvos da temperatura mínima aperta um botão
        if GPIO.input(KEY_MIN) == GPIO.HIGH:
            flag = False
            while True:
                if GPIO.input(KEY_MIN) == GPIO.LOW:
                    flag = True
                lcd.clear()
                min_temp_s = str(min_temp)
                lcd.set_cursor(0,0)
                lcd.message('Min_temp:')
                lcd.set_cursor(9,0)
                lcd.message(min_temp_s.ljust(5)[:5])
                lcd.set_cursor(15,0)
                lcd.message('C')

                # Define a posição dos dados do dia que que ocorreu o evento
                # o mínimo valor na tela
                if int(day_min) < 10:
                    lcd.set_cursor(0,1)
                    lcd.message('0')
                    lcd.set_cursor(1,1)
                    lcd.message(day_min)
                else:
                    lcd.set_cursor(0,1)
                    lcd.message(day_min)


                lcd.set_cursor(2,1)
                lcd.message('/')

                # Define a posição dos dados do mês que aconteceu
                # o mínimo valor na tela
                if int(month_min) < 10:
                    lcd.set_cursor(3,1)
                    lcd.message('0')
                    lcd.set_cursor(4,1)
                    lcd.message(day_min)
                else:
                    lcd.set_cursor(3,1)
                    lcd.message(month_min)
```

```
# Configura a posiçõa do ano e hora do valor mínimo do display LCD
                    lcd.set_cursor(5,1)
                    lcd.message('/')
                    lcd.set_cursor(6,1)
                    lcd.message(year_min)
                    lcd.set_cursor(11,1)
                    lcd.message(hour_min)
                    lcd.set_cursor(13,1)
                    lcd.message(':')


                    # Define a posição dos dados do minuto que aconteceu
                    # o máximo valor na tela
                if minute_min < 10:
                    lcd.set_cursor(14,1)
                    lcd.message('0')
                    lcd.set_cursor(15,1)
                    lcd.message(minute_min)
                else:
                    lcd.set_cursor(14,1)
                    lcd.message(minute_min)


                # Sai do loop infinito de temperatura mínima
                if ((GPIO.input(KEY_MIN) == GPIO.HIGH) and flag == True) == True:
                    break
                time.sleep(0.5)


    # Caso os dados não sejam válidos,
    # é apresentado na tela do LCD um notificação de erro de leitura dos dados
    else:
        lcd.clear()
        lcd.set_cursor(1,0)
        lcd.message('Reading Failed')

gpio.cleanup()
exit()
```

Descrição do Código

Entre as linhas 2 e 6 são declaradas as bibliotecas das funções que serão usadas no código.

Na linha 10 é definido qual modelo de sensor que iremos utilizar, no caso usar o modelo DHT-22.

Entre as linhas 13 a 19 é definida a pinagem do display de LCD.

Nas linhas 22, 23 e 24 são definidas as variáveis que definem quais são os GPIOs que serão utilizados como entrada de dados.

Na linha 27 são desabilitados os avisos de warnings.

Na linha 30 é configurado o modo de utilização das portas definidas pelo número dos GPIOs.

Nas linhas 33, 34 e 35 é configurada a pinagem dos GPIOs que receberam os dados.

Nas linhas 38 e 39 define-se que o display de LCD utilizado será 16 colunas e 2 linhas.

Na linha 42 define-se um valor inicial para a variável que será utilizada para armazenar o valor de temperatura máxima e na linha 43 define-se um valor inicial para a variável que será utilizada para armazenar o valor de temperatura mínima.

Nas linhas 46, 47 e 48 é declarado o objeto “lcd” a partir da função que define quais GPIOs irão ser utilizados para manipular o display de LCD.
Na linha 50 é onde o loop infinito principal deste código fonte é definido.

Nas linhas 52 e 53 as variáveis “tdy” e “now” recebem a informação de tempo, respectivamente dia e hora.

Na linha 56 as variáveis “umid” e “temp” recebem, respectivamente, as informações de umidade e temperatura geradas pelo módulo AM2302.

Na linha 59 é testado se os dados gerados pelo módulo AM2302 são válidos. Sendo válidos, o código é executado. Caso contrário, é gerada uma mensagem de erro presente entre as linhas 229 e 232.

Entre as linhas 61 e 68 é configurada a forma de apresentação dos dados de temperatura e entre as linhas 71 e 77 é configurada a forma de apresentação dos dados de umidade no display de LCD.

Entre as linhas 80 e 95 são realizados os registros de máxima e mínima temperatura que são realizados durante o período de medição.
Entre as linhas 98 e 160 são acessados os dados referentes à maior temperatura registrada durante o intervalo de medição, e se realiza a manipulação desses dados para sua apresentação no display de LCD. Os dados são de hora, dia, mês e ano que este evento ocorreu. Para se mostrar esses dados, quando se é pressionado o botão que dá acesso a estas informações, se entra num loop infinito que só é interrompido pressionando o botão novamente, fazendo que se retorne a tela inicial.

Entre as linhas 163 e 226 são acessados os dados de menor temperatura registrada durante o intervalo de medição. A manipulação dos dados é realizada de forma similar ao que é realizado para temperatura máxima.

Conclusão

Com o desenvolvimento do projeto do termômetro digital, pode-se adquirir conhecimento teórico e prático sobre o módulo sensor de temperatura e umidade, no caso o modelo AM2302. Também foi apresentado como instalar e utilizar as funções da biblioteca Python para o módulo DHT-11/22, que nos permite conseguir manipular os dados obtidos.

Com essas informações em mãos, pode-se observar que é simples utilizar o módulo AM2303 ou DHT-22 para outras aplicações onde se há a necessidade de analisar as informações de temperatura e umidade relativa ar e realizar algumas atuações a partir desses dados.

O leitor agora é capaz de desenvolver os seus próprios projetos utilizando o módulo DHT-22 (composto de um sensor de temperatura e outro de umidade) e similares.

Este artigo foi escrito por **Roniere Rezende.**

Considerações Finais

Chegamos ao final do nosso ebook.

Caso você tenha alguma dúvida, não deixe ela sem resposta. Você pode entrar em contato com o autor através da seção de comentários do artigo, ou então participar da nossa comunidade, interagindo com outros profissionais da área:

Comunidades Embarcados

Caso você tenha encontrado algum problema no material ou tenha alguma sugestão, por favor, entre em contato conosco. Sua opinião é muito importante para nós:

contato@embarcados.com.br

Siga o Embarcados na Redes Sociais


facebook.com/osembarcados/


instagram.com/portalembarcados/


youtube.com/embarcadostv/


linkedin.com/company/embarcados/


twitter.com/embarcados