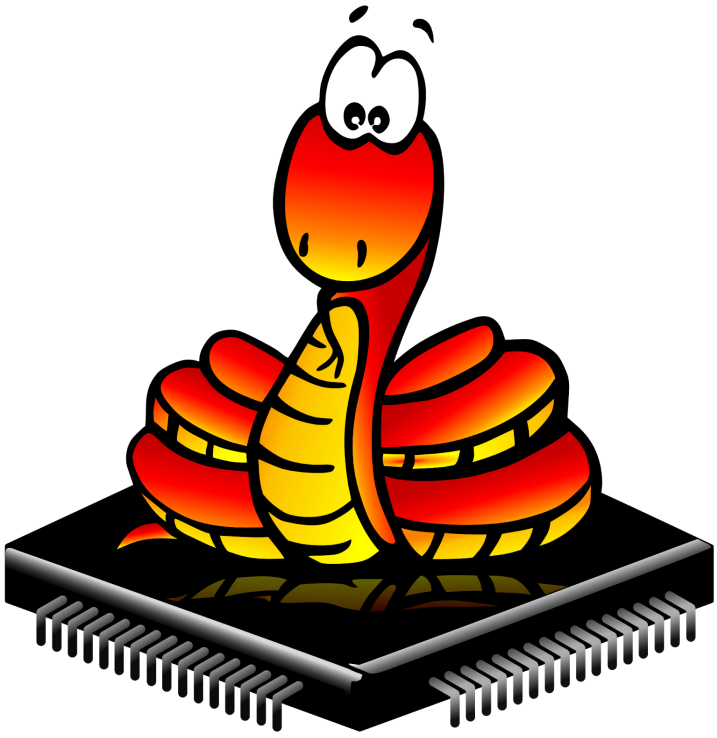


Micropython e a ESP32

2020

PET - Tecnologia em Eletrônica e Computação
Universidade Federal de Itajubá - UNIFEI

Itajubá Minas Gerais

O que é a ESP32?

A ESP32 é uma unidade de microprocessamento fabricado pela empresa Espressif. Possui conectividade Wi-Fi e Bluetooth para uma ampla gama de aplicações. Possui um co processador de ultra baixo consumo de energia, além de múltiplos periféricos.

Suas principais especificações são:

- CPU: Xtensa® Dual-Core 32-bit LX6
- ROM: 448 KBytes
- RAM: 520 KBytes
- Flash: 4 MB
- Clock Máximo: 240MHz
- Wireless padrão 802.11 b/g/n
- Conexão Wifi 2.4Ghz (máximo de 150 Mbps)
- Bluetooth BLE 4.2
- Portas GPIO: 11
- Conversor analógico digital (ADC)

O que é Python?

Python é uma linguagem de programação interpretada de alto nível. Possui tipagem dinâmica e foi desenhada para ajudar os programadores a escreverem um código limpo e conciso. Sua principal vantagem é sua simplicidade e o acesso a um imenso número de bibliotecas. Para a programação de microcontroladores pode se usar uma implementação do Python chamada MicroPython que é um subconjunto da linguagem otimizado para rodar

em dispositivos limitados.

Por que utilizar o MicroPython?

As principais vantagens ao se utilizar o MicroPython são:

- Acesso a praticamente todos os recursos da linguagem Python

Incluindo suporte a números inteiros de precisão arbitrária, unicode por padrão, além de um imenso conjunto de bibliotecas.

- Acesso a um REPL

Dessa forma é possível testar pequenos trechos de programas diretamente na placa, sem necessidade de upload de arquivos.

- Não há necessidade de compilar e linkar arquivos fontes

Os códigos fontes dos programas são gravados diretamente na placa.

- Acesso ao o WebREPL

Possibilidade de acessar o REPL através de uma rede Wi-Fi.

- Sistema de arquivos

MicroPython oferece por padrão o sistema de arquivos SPIFFS, tornando muito mais simples a organização dos dados no interior da placa.

Softwares necessários

Primeiramente é necessário obter o firmware do MicroPython que pode ser obtido no site oficial ou mais diretamente no link: https://micropython.org/download/esp32/. Em seguida deve-se instalar o Python em seu computador.

Para usuários de Linux e MacOs provavelmente o Python já está instalado em seu sistema, mas caso não esteja ele pode ser instalar com o gerenciador de pacotes de seu sistema. Para esses sistemas também pode ser necessário instalar o "pip" que é um gerenciador de pacotes do Python que pode ser instalado da mesma forma que o Python.

Para usuários de Windows basta acessar o site oficial do Python e efetuar o download do Instalador. Será instalado tanto Python quanto o pip.

Após a instalação do Python e do pip usaremos o seguinte comando em um terminal para instalar a ferramenta "esptool" :

```
pip install esptool
```

Essa ferramenta é necessária para escrever na memória flash da ESP32 e, dessa forma, instalar o firmware do MicroPython.

Outra ferramenta recomendada é o "ampy" que nos permite enviar arquivos de código fonte para placa. Para a instalação usaremos também o pip:

```
pip install adafruit-ampy
```

Por fim precisamos instalar um emulador de terminal que servirá para acessarmos o prompt do MicroPython. No Windows podemos instalar o Tera Term que pode ser encontrado facilmente em uma pesquisa online e no linux o picoterm que certamente estará nos repositórios de sua distribuição.

No Windows precisaremos instalar um driver que permita que o sistema operacional reconheça a placa. O driver mais recomendado é o “Silicon Labs CP2102 USB-to-UART bridge chip” que pode ser obtido através deste link.

Instalando o firmware do MicroPython na ESP32

Agora que temos todos os arquivos e programas necessários, podemos começar a instalar o firmware do MicroPython na placa. Porém antes de tudo devemos conectar a placa em uma das entradas USB de seu computador e colocar a placa em Bootloader Mode. Para isso basta pressionar o botão escrito “BOOT”, provavelmente ao lado da entrada de USB de sua placa.

No próximo passo devemos apagar toda a memória da placa para evitar possíveis conflitos quanto formos instalar o Micropython na placa. Para isso usaremos o esptool.

Se estiver usando linux é necessário adicionar o seu usuário ao grupo que possui acesso aos dispositivos USB. Para derivados do Debian como Ubuntu e Mint deve-se executar o comando a seguir. Para distribuições derivadas do Arch deve-se trocar “dialout” por “uucp”.

```
sudo usermod -a -G dialout username
```

Abra um terminal e entre com o comando a seguir, onde PORTA é a porta USB onde está conectada a sua placa. No linux e Mac é geralmente /dev/ttyUSB0 ou /dev/ttyUSB1. Já no Windows é provavelmente COM1 ou COM2.

```
esptool.py --port PORTA erase_flash
```

Agora já podemos instalar o Micropython, também utilizando o esptool:

```
esptool.py --chip esp32 --port PORTA write_flash -z 0x1000 PATH_DO_FIRMWARE
```

onde PATH_DO_FIRMWARE é o caminho do arquivo de firmware baixado.

Se nenhum erro aconteceu, então a instalação foi um sucesso! Já podemos acessar o prompt do MicroPython através de um emulador de terminal.

Acessando o Prompt do MicroPython

Se você estiver utilizando o picocom, para acessar o prompt, insira o seguinte comando em um terminal:

```
picocom PORTA -b115200
```

Se tudo ocorrer bem você será apresentado a um prompt precedido por “>>”. Parabéns você está acessando o MicroPython que está rodando diretamente da sua placa ESP32. Experimente digitar algum comando do Python, por exemplo 2+2.

Para os usuários de Windows que estiverem usando o Tera Term, após abrir o programa aparecerá uma janela pop-up. Nesta janela deve-se selecionar a opção “Serial” e selecionar a porta na qual a sua placa está conectada. Em seguida abra o menu “Setup” e selecione a opção “Serial Port...”, uma janela irá se abrir e nela modifique o campo “Speed” com o valor 115200 que é a taxa de bits da comunicação serial. Ao retornar para a janela principal pressione enter, se tudo deu certo deverá surgir um prompt começando com “>>”. Agora você pode digitar alguns comandos do Python que serão executados a partir da placa.

A IDE Upycraft

A IDE mais utilizada com o Micropython é a Upycraft que conta com um REPL interno, um editor de texto com syntax highlight e permite enviar e ler arquivos da ESP32 de forma simples.

Para instalar no Windows basta acessar a página no github do projeto, https://github.com/DFRobot/uPyCraft, e baixar o arquivo de executável de uma das versões.

No linux pode ser instalado através do gerenciador de pacotes de seu sistema ou compilando o código fonte disponível juntamente com as instruções em https://github.com/DFRobot/uPyCraft_src.

Após abrir a IDE deve-se conectá-la à placa. Para isso basta ir no menu “tools”, “serials” e selecionar a porta onde a placa está conectada. Se tudo ocorrer um prompt deve aparecer no painel inferior da IDE.

No canto esquerdo um painel apresenta os arquivos do diretório de projeto (workspace e sd) , do sistema de arquivos interno da ESP32 (device) e algumas bibliotecas da própria IDE (uPy_lib).

Para criar um novo arquivo de código fonte deve-se ir em “file” e “new”.

Primeiro Programa

Nesta parte do tutorial será criado um programa simples que faz com que um dos leds embutidos na placa alterne entre ligado e desligado a cada 500ms.

Após criar um novo arquivo na uPyCraft copie o seguinte código:

```
from machine import Pin
import time
pin2 = Pin(2, Pin.OUT)

def alterna(pin):
  pin.value(not pin.value())

while True:
  alterna(pin2)
  time.sleep_ms(500)
```

Agora será explicado cada uma das linhas de código.

```
from machine import Pin
import time
```

Nas duas primeiras linhas são importados os módulos machine, que possui as classes que representam os GPIOs da placa, e time que possui a rotina que possui a rotina de atraso.

```
pin2 = Pin(2, Pin.OUT)
```

Em seguida define-se um objeto Pin que recebe como argumento o número do pino (no caso 2, que representa o led da placa) e se o mesmo será saída ou entrada.

```
def alterna(pin):
  pin.value(not pin.value())
```

Aqui é definida uma rotina para alternar o valor de um pino entre ligado e desligado. Para obter o valor de um pino usa-se o método value() sem nenhum parâmetro. Para setar o valor usa-se a mesma função porém com o valor desejado como parâmetro: value(valor). O valor pode ser 1 (ligado) ou 0 (desligado).

```
while True:
  alterna(pin2)
  time.sleep_ms(500)
```

Nestas linhas é definido um loop infinito. A cada iteração do loop a função alterna é chamada no pino 2 e em seguida a placa consome tempo durante 500ms.

Antes de rodar o programa deve-se salvar arquivo que pode ser feito através do menu “file” ou do atalho Ctrl+S. Assim o seu código fonte será salvo no workspace, não na placa.

Com arquivo salvo pode-se descarregá-lo e rodá-lo na placa com o atalho F5 ou através da opção Download and Run no menu Tools. Dessa forma a IDE irá copiar o código fonte do workspace para a memória interna da ESP32 e em seguida executá-lo.

Se tudo ocorrer bem, um led de cor azul deverá estar piscando na sua placa. Para interromper a execução do programa basta apertar Ctrl+C no REPL da IDE.

Conectando-se ao Wifi

O seguinte código conecta a ESP32 há uma rede Wifi.

```
import network

wlan = network.WLAN(network.STA_IF) # cria a interface Wifi
wlan.active(True) # ativa a interface

wlan.connect('<ESSID de sua rede>', '<a senha>') # conecta-se a ela
```

Após executar o código pode-se verificar se a conexão foi um sucesso entrando com o seguinte código no REPL:

```
>>> wlan.isconnected()
```

Também é possível obter informações sobre a rede com ifconfig.

```
>>> wlan.ifconfig()
```

Acessando a web com Sockets

Com o Wifi configurado pode-se acessar a rede através de sockets. Um socket é um software que representa um ponto final em uma rede de computadores tanto recebendo quanto enviando dados, é o bloco de construção dos protocolos de internet. Os sockets permitem que uma comunicação seja estabelecida entre dois dispositivos, desde que ambos usem o mesmo protocolo e saibam seus respectivos endereços de IP e número da porta.

No código a seguir é criado um socket para realizar um requisição GET HTTP.

```
import socket

def http_get(url):
    _, _, host, path = url.split('/', 3)
    addr = socket.getaddrinfo(host, 80)[0][-1]
    s = socket.socket()
    s.connect(addr)
    s.send(bytes('GET /%s HTTP/1.0\r\nHost: %s\r\n\r\n' % (path, host),
'utf8'))
    while True:
        data = s.recv(100)
          if data:
              print(str(data, 'utf8'), end='')
          else:
              break
    s.close()
```

Após descarregar o código na placa e rodá-lo podemos chamar a função http_get através do REPL:

```
>>> http_get('http://micropython.org/ks/test.html')
```

que deverá retornar a seguinte string.

```
HTTP/1.1 200 OK
Server: nginx/1.10.3
Date: Thu, 06 Aug 2020 21:57:47 GMT
Content-Type: text/html
Content-Length: 180
Last-Modified: Tue, 03 Dec 2013 00:16:26 GMT
Connection: close
Vary: Accept-Encoding
ETag: "529d22da-b4"
Accept-Ranges: bytes

<!DOCTYPE html>
<html lang="en">
      <head>
      <title>Test</title>
      </head>
      <body>
      <h1>Test</h1>
      It's working if you can read this!
      </body>
</html>
```

Agora será apresentada uma explicação dos pontos importantes do código.

```
_, _, host, path = url.split('/', 3)
```

A linha de código acima obtém o domínio e o path a partir da url simplesmente dividindo a string em strings menores separadas pelo caracter "/". O segundo argumento de *split* é a quantidade máxima de divisões que serão realizadas.

```
addr = socket.getaddrinfo(host, 80)[0][-1]
```

Aqui é utilizada a função *getaddinfo* para obter o endereço de IP do host. Nesse caso como se trata de um servidor HTTP é utilizada a porta 80 como segundo argumento da função. A função retorna uma lista de tuplas contendo informações sobre os possíveis sockets que poderão ser criado para se conectar as host. Nesse caso precisaremos apenas do primeiro

elemento da lista (índice 0) e o último elemento da tupla (índice -1) que contém o endereço de IP.

```
s = socket.socket()
s.connect(addr)
```

Nas duas linhas acima é criado um socket que se conecta ao endereço de IP do host.

```
 s.send(bytes('GET /%s HTTP/1.0\r\nHost: %s\r\n\r\n' % (path, host),
'utf8'))
```

Em seguida manda-se uma requisição GET para o servidor. Essa requisição é utilizada para retornar o conteúdo determinado arquivo, nesse caso é pedido o indicado pelo *path* da url. Também é preciso incluir o domínio do host. Observe que a mensagem deve ser uma sequência de bytes codificada em UTF-8, por isso o uso da função *bytes*.

```
while True:
       data = s.recv(100)
       if data:
           print(str(data, 'utf8'), end='')
       else:
            break
```

Para recuperar os dados que o servidor enviará para a placa usa-se o método *recv* do socket. Essa função obtém um determinado número de bytes, indicado por seu argumento, recebidos. Como não se sabe o tamanho dos dados que serão recebidos é construído um loop que recupera 100 bytes dados em cada iteração. Quando todos os dados já foram recebidos *recv* retorna *null* e o loop é quebrado, caso contrário os dados são exibidos no console.

Criando Um Cliente e Servidor TCP

Um servidor e um cliente TCP é muito útil para comunicar entre dispositivos através de uma rede WiFi por exemplo. O cliente TCP pode ser criado de forma semelhante ao cliente HTTP do exemplo anterior, no exemplo a seguir é criado um cliente que envia uma mensagem para um servidor

```
import socket
endereco_IP = "Coloque aqui o endereço de algum servidor"
porta = 1000 # a porta que o servidor está rodando
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((endereco_IP, porta))
msg = bytes("Olá tudo bem?", 'utf-8')
s.send(msg)
s.close()
```

Já o código a seguir cria um servidor TCP:

```
import socket
endereco_IP = ''       # para o servidor este campo pode ficar vazio
porta = 5000           # pode ser qualquer porta livre
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((endereco_IP, porta))
s.listen(1)
while True:
    connection, ip_cliente = s.accept()
    print('Concetado por', ip_cliente)
    while True:
        msg = connection.recv(1024)
        if not msg: break
        print(ip_cliente, msg)
    print('Finalizando conexao do cliente', ip_cliente)
    connection.close()
```

Observe que é semelhante a criação do Cliente porém usa-se os métodos *bind* e *listen* no lugar de *connect*, o primeiro cria o servidor e o segundo o prepara para aceitar conexões. Já dentro do *while* método *accept* do socket mantém o programa bloqueado enquanto um cliente não se conectar ao servidor, quando uma conexão é realizada o método retorna uma tupla contendo um objeto *connection* e o endereço do cliente, e o programa continua sua execução. A mensagem do cliente é lida através do método *recv* que recebe como argumento a quantidade de bytes que será lida. Por fim a conexão é encerrada com *connection.close().*

Para testar estes códigos é interessante rodar o cliente na placa e o servidor em um computador colocando os devidos endereços IPs e vice-versa.

Trabalhando com GPIOs

Uma das tarefas mais comuns executadas por microcontroladores é obter informações do mundo externo e modificá-lo. Para isso pode ser utilizado os pinos de entrada e saída da placa para realizar leitura e alteração do nível lógico de tensão em seus terminais. Nem todos os pinos da placa podem ser utilizados com essa função, somente os pinos 1, 2, 3, 4, 5, 12, 13, 14, 15, 16. Tenha em mente que esses números estão relacionados a associação

Para acessá-los primeiramente é necessário importar o módulo *machine:*

```
import machine
```

E para criar um pino de saída:

```
pin = machine.Pin(0, machine.Pin.OUT)
```

Onde "0" indica o número do pino desejado e o segundo argumento indica que um pino de saída. Para modificar o seu valor usa-se o método value indicando no argumento o valor desejado.

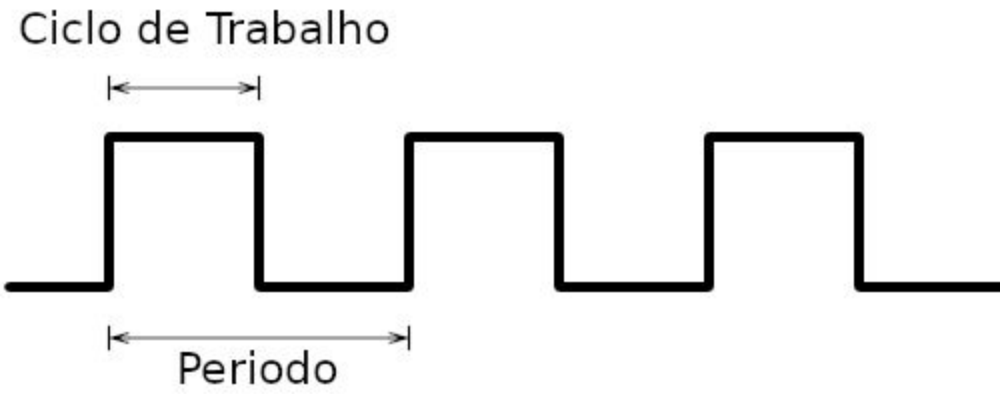
```
pin.value(1)   # O pino terá nível lógico um, poderia ser utilizado para
acender um LED
```

Caso queira-se ler o valor de um pino é necessário declará-lo como entrada e utilizar o método *value* sem argumentos.

```
pin = machine.Pin(0, machine.Pin.IN)
pin.value()  # retorna 1 ou 0, poderia para fazer leitura de um botão por
exemplo
```

Pinos PWM

Pulse Width Modulation (PWM) é uma técnica utilizada para simular uma saída analógica em uma saída digital. Consiste em variar rapidamente a saída gerando uma onda quadrada, conforme a figura a seguir. A onda possui um período e um ciclo de trabalho, isto é, porcentagem do ciclo em valor 1.



O PWM pode ser utilizado apenas nos pinos 0, 2, 4, 5, 12, 13, 14 e 15. Um pino pode ser configurado para PWM da seguinte forma:

```
import machine
pin0 = machine.Pin(0, machine.Pin.OUT)
pwm0 = machine.PWM(pin0)
```

Para configurar a frequência e o ciclo de trabalho usa-se os métodos *freq* e *duty* respectivamente.

```
pwm0.freq(500)
pwm0.duty(512)
```

É importante ressaltar que a frequência é compartilhada por todos os pinos, portanto dois pinos não podem ter frequências diferentes ao mesmo tempo. E o valor passado ao método *duty* deve estar entre 0 e 1023, o primeiro corresponde a 0% e o segundo a 100% e assim 512 seria 50%, por exemplo.

Leitura Analógica

Alguns pinos também ser utilizados para realizar leitura de valores analógicos. São disponíveis os pinos de 32 a 39 e para utilizá-los é preciso fazer o seguinte:

```
import machine
pin32 = machine.Pin(23, machine.Pin.OUT)
pwm32 = machine.ADC(pin32)
```

O valor pode ser lido com o método read.

```
pwm32.read()
```

Por padrão, entradas de zero volts são lidas como zero e entradas maiores ou iguais a 1V são lidas como 1023.