



# SISTEMAS EMBARCADOS

PROF. JOSENALDE OLIVEIRA

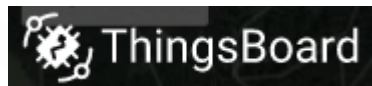
[josenalde.oliveira@ufrn.br](mailto:josenalde.oliveira@ufrn.br)

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - UFRN

-

# PLATAFORMAS ESPXXXX - DASHBOARDS

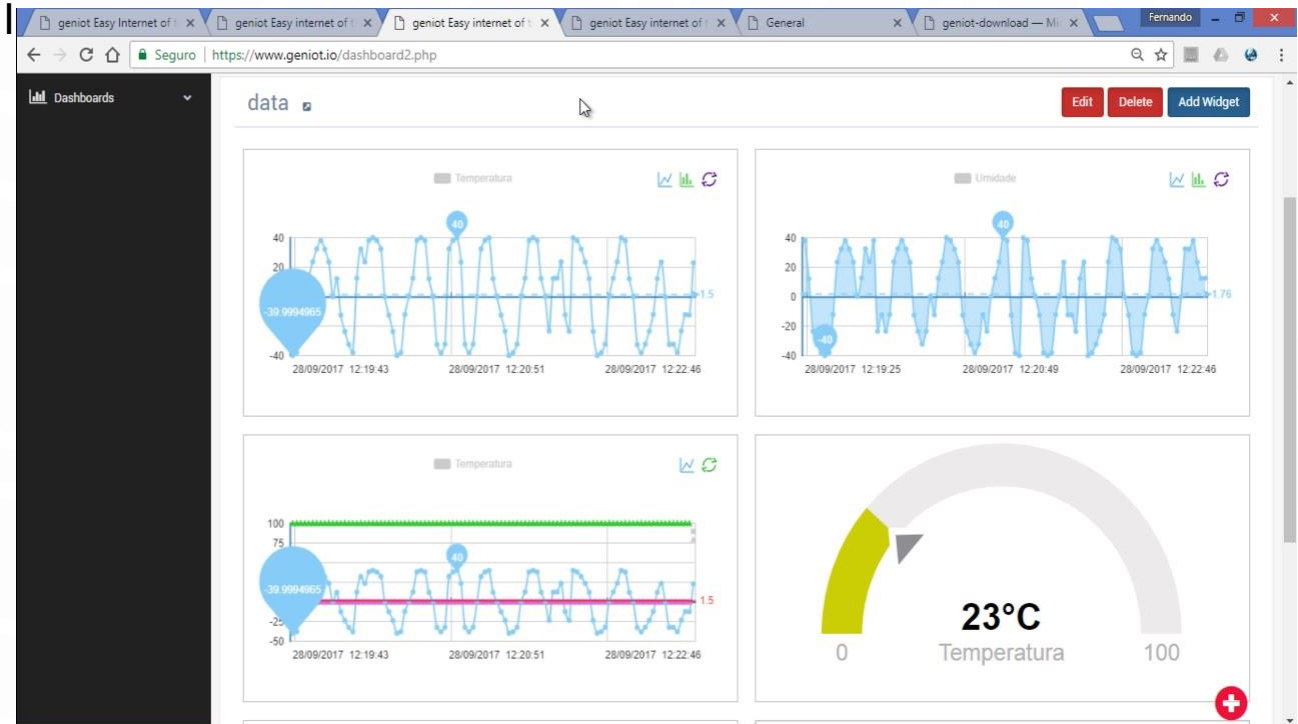
- GENIOT (MICROGENIOS): automação industrial



<https://thingsboard.io/>



<https://dojot.com.br/>



# PLATAFORMAS ESP8266/ESP12-E , ESP32

- Comparativo (Espressif)

Apenas com RTOS (por exemplo, FREERTOS nativo), pode-se usar o potencial de dividir tarefas em 02 núcleos

	ESP32	ESP8266	ARDUINO UNO R3
Cores	2	1	1
Arquitetura	32 bits	32 bits	8 bits
Clock	160MHz	80MHz	16MHz
WiFi	Sim	Sim	Não
Bluetooth	Sim	Não	Não
RAM	512KB	160KB	2KB
FLASH	16Mb	16Mb	32KB
GPIO	36	17	14
Interfaces	SPI / I2C / UART / I2S / CAN	SPI / I2C / UART / I2S	SPI / I2C / UART
ADC	18	1	6
DAC	2	0	0

Specifications	ESP8266	ESP32
MCU	Xtensa® Single-Core 32-bit L106	Xtensa® Dual-Core 32-bit LX6 600 DMIPS
802.11 b/g/n Wi-Fi	Yes, HT20	Yes, HT40
Bluetooth	None	Bluetooth 4.2 and below
Typical Frequency	80 MHz	160 MHz
SRAM	160 kBytes	512 kBytes
Flash	SPI Flash , up to 16 MBytes	SPI Flash , up to 16 MBytes
GPIO	17	36
Hardware / Software PWM	None / 8 Channels	1 / 16 Channels
SPI / I2C / I2S / UART	2/1/2/2	4/2/2/2
ADC	10-bit	12-bit
CAN	None	1
Ethernet MAC Interface	None	1
Touch Sensor	None	Yes
Temperature Sensor	None	Yes
Working Temperature	- 40°C ~ 125°C	40°C ~ 125°C

Saída analógica “pura”, não PWM, entre 0 e 3,3VDC (dacWrite(0-255)), 255 = 3,3V e 128 = 1,65V

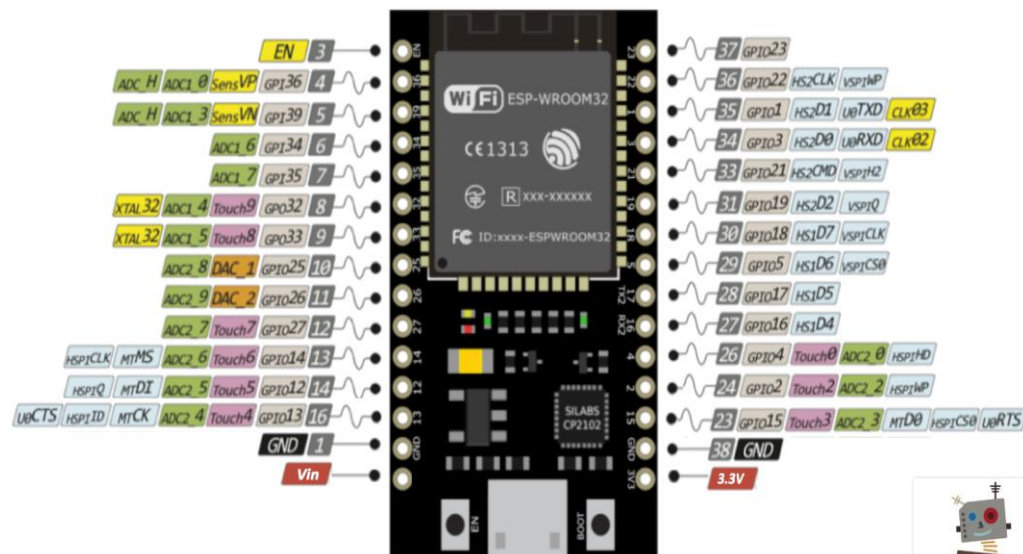


# PLATAFORMAS ESP32

- OBS: em algumas placas, problema no UPLOAD do código. **Colocar cap. 10uF entre o EN e GND**

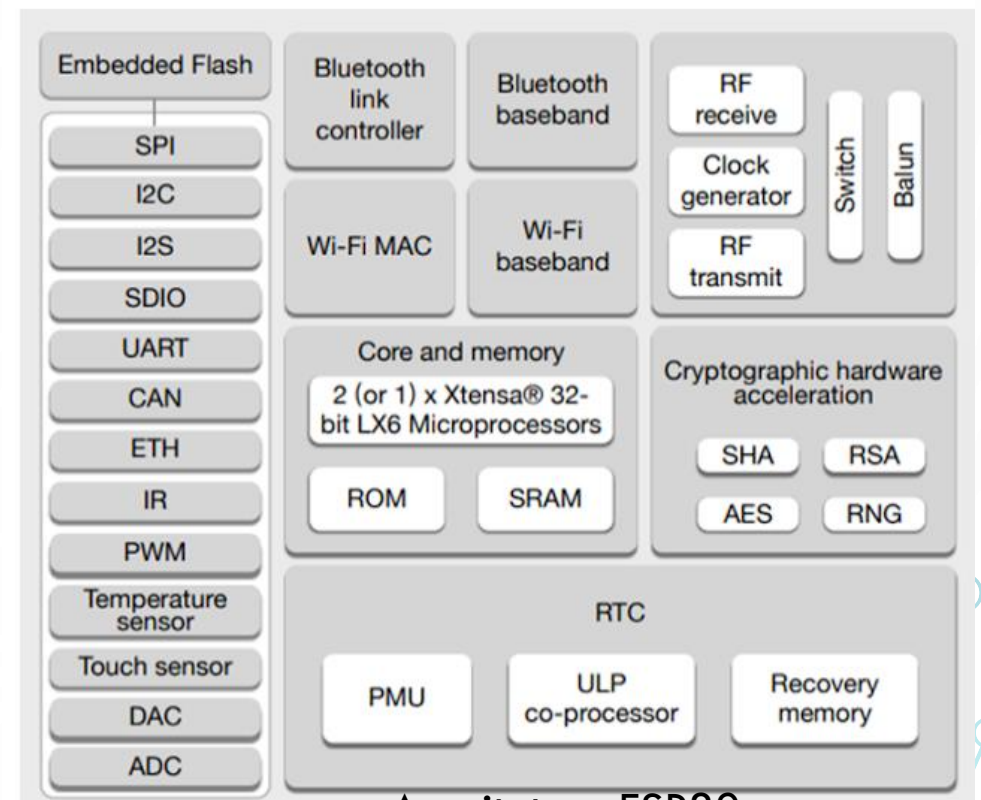
## ESP32 PINOUT

ESP32 DEVELOPMENT BOARD DUAL CORE ESP-32 & ESP-32S BOARD



Freely adapted by <https://MJRoBot.org>

PINAGEM – Node32s – 30 pinos



Arquitetura ESP32

**Se encontrar problemas com a exibição de valores no Serial Monitor, desabilitar DTS, RTS control**

# PLATAFORMAS ESP32

- Um diferencial é o DAC de 8 bits (2 canais) (gerar sinal no ESP32 e visualizar no osciloscópio):



<https://www.fernandok.com/2018/08/esp32-voce-sabe-o-que-e-dac.html>

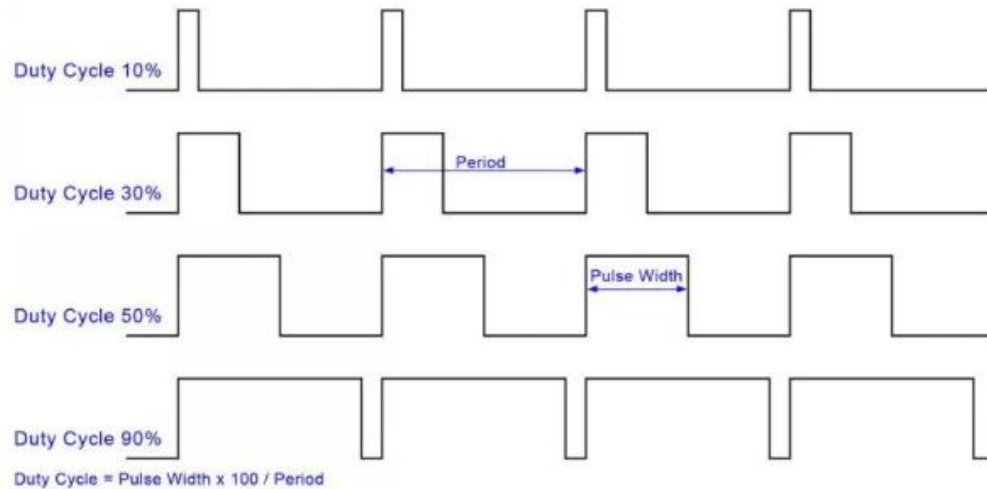
<https://github.com/G6EJD/ESP32-DAC-Examples>

- **Proposta: adaptar código do PID\_v1.h/.cpp para o ESP32 com o dacwrite**

# PLATAFORMAS ESP32

- O gerador PWM permite maior flexibilidade –

## Pulse Width Modulation



$$V_{cc} * duty\ cycle = V_m$$

$$3.3 * 0.32\% = 1.056V$$

Canal: 0 – 15.

Frequência: 1 – 40MHz.

Resolução: 1 – 16 bits.

```
1 void setup()
2 {
3   pinMode(2, OUTPUT); // Definimos o pino 2 (LED) como saída.
4
5   ledcAttachPin(2, 0); // Atribuímos o pino 2 ao canal 0.
6   ledcSetup(0, 1000, 10); // Atribuímos ao canal 0 a frequência de 1000Hz com resolução de 10bits
7 }
8
9 void loop()
10 {
11   for (int i = 0; i < 1024; i++)
12   {
13     ledcWrite(0, i); // Escrevemos no canal 0, o duty cycle "i".
14     delay(2);
15   }
16   for (int i = 1023; i > 0; i--)
17   {
18     ledcWrite(0, i);
19     delay(2);
20   }
21 }
22
23
24
```

$$\frac{Clock}{Resolução} = F_m$$

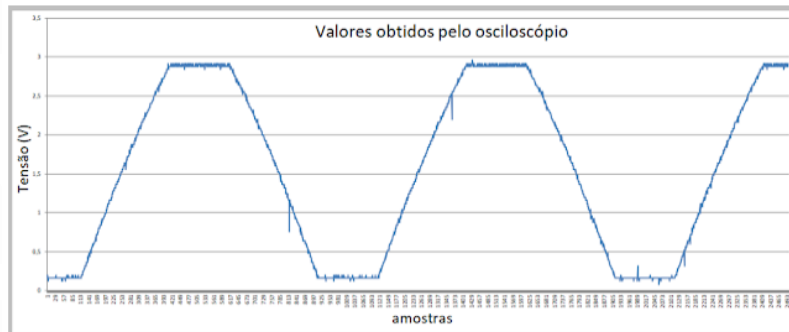
# PLATAFORMAS ESP32

- Canais ADC

Segundo dados da **Espressif**, os **chips ESP32** podem apresentar uma **diferença de +/- 6% de um chip para outro nos resultados medidos**.

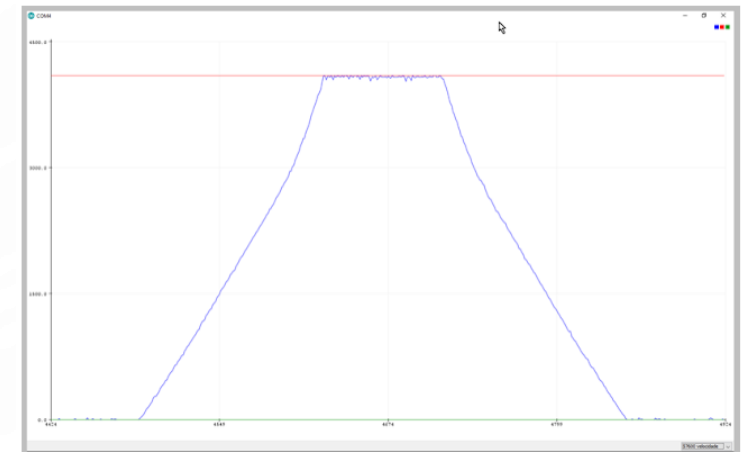
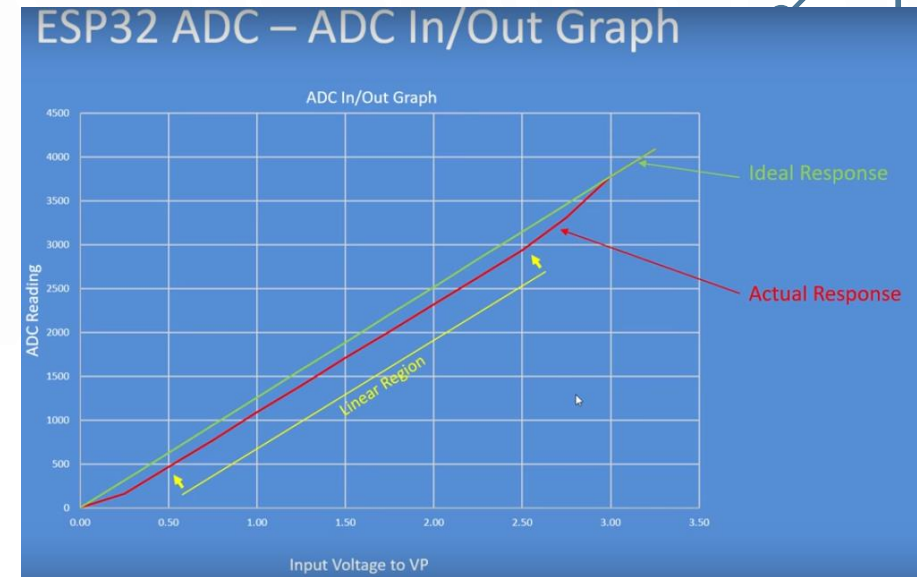
Além disso, a conversão **NÃO** possui uma resposta linear por toda faixa disponível para leitura.

A Espressif disponibiliza um método para calibração e sugere que os usuários implementem outros métodos caso julguem necessário, afim de alcançar a precisão desejada.



Osciloscópio

analogRead





# PLATAFORMAS ESP32

- Canais ADC

```
1  const int pin_leitura = 36; //GPIO usado para captura analógica
2
3  void setup() {
4      Serial.begin(1000000); //Iniciando a porta serial somente para debug
5      pinMode(pin_leitura, INPUT); //Pino utilizado para captura analógica
6  }
7
9  void loop() {
10
11      int valor_analogico = analogRead(pin_leitura);
12      //Serial.print(valor_analogico + f(valor_analogico));
13      Serial.print(valor_analogico); //imprime os valores
14      Serial.print(",");
15      Serial.print(4095); //cria uma linha para marcar
16      Serial.print(",");
17      Serial.println(0); //cria uma linha para marcar
18  }
19
20  /*
21   Mode: normal
22   Polynomial degree 6, 2365 x,y data pairs
23   Correlation coefficient (r^2) = 9,907187626418e-01
24   Standard error = 1,353761109831e+01
25   Output form: C/C++ function:
26
27   Copyright © 2012, P. Lutus -- http://www.arachnoid.com. All Rights Reserved.
28   */
29
30  double f(double x) {
31      return 2.202196968876e+02
32          + 3.561383996027e-01 * x
33          + 1.276218788985e-04 * pow(x, 2)
34          + -3.470360275448e-07 * pow(x, 3)
35          + 2.082790802069e-10 * pow(x, 4)
36          + -5.306931174991e-14 * pow(x, 5)
37          + 4.787659214703e-18 * pow(x, 6);
38  }
```

<https://www.fernandok.com/2018/09/voce-nao-sabia-ajuste-de-adc-do-esp32.html>

# PLATAFORMAS ESP32

- Canais ADC

Resolução ADC: 12 bits = 0..4095 DEC

-----  
 $3.3 \text{ VDC} / 4096 = 0,80566 \text{ mV}$  Resolução

AnalogRead(36); // ou pino VP GPIO36  
Ex.  $464 * 0,80566 = 0,375\text{V}$

analogSetClockDiv(1) 433 us - default  
adcAttachPin(13)

analogSetClockDiv(255) 775 us

```
double ReadVoltage(byte pin) {  
    double reading = analogRead(pin);  
    if (reading < 1 OR reading >= 4095) return 0;  
    return -0.0000000000000016*pow(reading,4) +  
           0.000000000118171*pow(reading,3) -  
           0.000000301211691*pow(reading,2) +  
           0.001109019271794*reading + 0.034143524634089;  
}
```

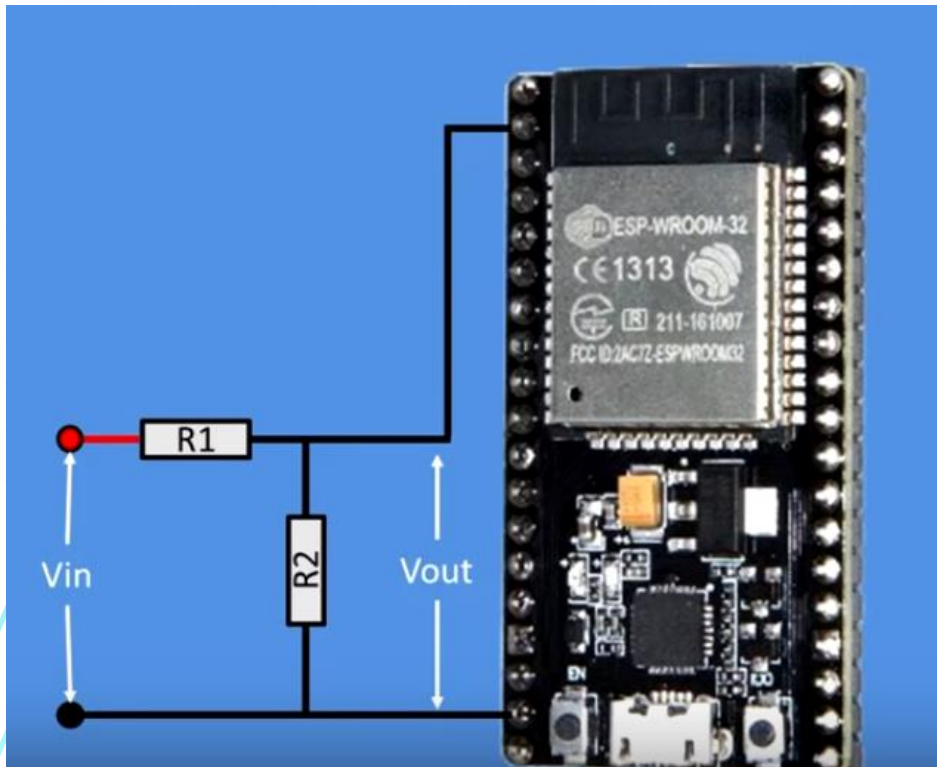
Erro < 1%, de 0 a 3V, região linear, caso contrário +- 7% precisão

## ESP32 ADC – ADC Advanced Programme Controls

Function	Description
<code>analogReadResolution(12);</code>	Sets the sample bits and read resolution, default is 12-bit (0 - 4095), range is 9 - 12 bits
<code>analogSetWidth(12);</code>	Sets the sample bits and read resolution, default is 12-bit (0 - 4095), range is 9 - 12 bits 9-bit = 0-511, 10-bit = 0-1023, 11-bit = 0-2047 and 12-bit = 0-4095
<code>analogSetCycles(8);</code>	Set number of cycles per sample, default is 8 and provides an optimal result, range is 1 - 255
<code>analogSetSamples(1);</code>	Set number of samples in the range, default is 1, it has an effect of increasing sensitivity
<code>analogSetClockDiv(1);</code>	Set the divider for the ADC clock, default is 1, range is 1 - 255
<code>analogSetAttenuation(attenuation);</code>	Sets the input attenuation for ALL ADC inputs, default is ADC_11db, range values are ADC_0db, ADC_2_5db, ADC_6db and ADC_11db
<code>analogSetPinAttenuation(pin, attenuation);</code>	Sets the input attenuation, default is ADC_11db, values are ADC_0db, ADC_2_5db, ADC_6db, ADC_11db  ADC_0db sets no attenuation e.g. 1.0 volt input = ADC reading of 1088, max voltage range = 3 ADC_2_5db sets an attenuation of 1.34 e.g. 1.0 volt input = ADC reading of 2086 ADC_6db sets an attenuation of 1.5 e.g. 1.0 volt input = ADC reading of 2975 ADC_11db sets an attenuation of 3.6 e.g. 1.0 volt input = ADC reading of 3959
<code>adcAttachPin(pin);</code>	Attach a pin to ADC (also clears any other analogue mode that could be on), returns TRUE/FALSE result
<code>adcStart(pin);</code> <code>adcBusy(pin);</code> <code>resultadcEnd(pin);</code>	Starts an ADC conversion on attached pin's bus; Check if conversion on the pin's ADC bus is currently running, returns TRUE/FALSE result or Get the result of the conversion (will wait if ADC has not finished), returns 16-bit integer

# DICAS PRÁTICAS: LENDO MAIS QUE 3,3V

- É possível usar divisores de tensão para ler sinais maiores que 3,3V. Exemplo:



Tensão de entrada	<input type="text" value="5"/>	Volts	<input type="button" value="Calcular"/>
R1	<input type="text" value="100"/>	ohms	<input type="button" value="Limpar"/>
R2	<input type="text" value="194.12"/>	ohms	
Tensão de Saída	<input type="text" value="3.3"/>	Volts	

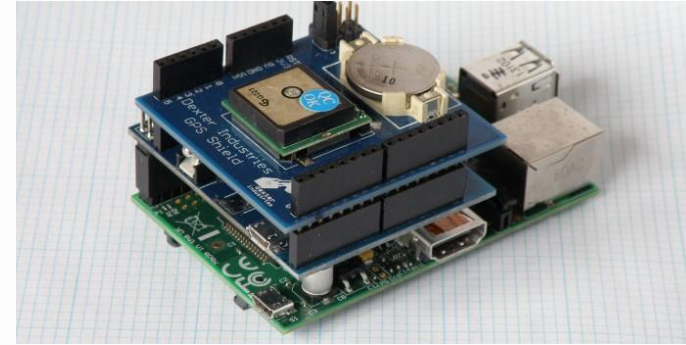
<https://www.arduinoecia.com.br/p/calculador-divisor-de-tensao-function.html>

```
float voltage = (float)analogRead(36)/4096*3.3;
```

```
float voltage = (float)analogRead(36)/4096*5*(194.12/200)
```

onde 5 é a tensão máxima de entrada e 194.12 é o valor teórico do divisor e 200 o resistor prático

# RASPBERRY PI



- **Análise para v3 B+**
- **SCP (SMALL COMPUTER BOARD):** periféricos integrados (wi-fi, bluetooth, ethernet, usb-host, vídeo-out, GPIO): permite desenvolver sistemas embarcados com interação IN/OUT com hardware, e inclui **SISTEMA OPERACIONAL** baseado em Linux (raspbian default) a ser instalado no **SD CARD**, com vários aplicativos (python, C/C++ Geany IDE). A IDE Arduino pode ser instalada versão **LINUX ARM 64 bits – ESP8266/ESP32 validar!**
- **Muito usado em projetos como servidor (arquivos, gateway, broker, web etc.) em configuração STACK**

## Especificações:

- Raspberry Pi 3 Model B+ Anatel
- Processador Broadcom BCM2837B0 64bits ARM Cortex-A53 Quad-Core
- Clock 1.4 GHz
- Memória RAM: 1GB
- Adaptador Wifi 802.11 b/g/n/AC 2.4GHz e 5GHz integrado
- Bluetooth 4.2 BLE integrado
- Conector de vídeo HDMI
- 4 portas USB 2.0
- Conector Gigabit Ethernet over USB 2.0 (throughput máximo de 300 Mbps)
- Alimentação: recomendamos uma fonte [DC chaveada 5V 3A](#)
- Interface para câmera (CSI)
- Interface para display (DSI)
- Slot para cartão microSD
- Conector de áudio e vídeo
- GPIO de 40 pinos



# RASPBERRY PI

- Pesquisas em redes de computadores (clusters)





# RASPBERRY PI

- **GPIO** (similar ao GPIO da placa FPGA DE2 voltagem)
- A corrente máxima no GPIO é 50 mA, para correntes

Maiores: Driver, transistor etc.; o GPIO não é tolerante a 5V

- Existem pull-up e pull-down internos
- Para ligar um LED por exemplo, usar um resistor 220R
- Baixar a última imagem do SO raspbian:

<https://www.raspberrypi.org/downloads/raspbian/> ou o

Gerenciador NOOBS: <https://www.raspberrypi.org/downloads/noobs/> e gravar

No SD CARD com o Etcher.io, por exemplo.

Raspberry Pi B+ J8 Header

Pin#	NAME		NAME	Pin#
01	3.3v DC Power	⬛ ⬛	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	⬛ ⬛	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	⬛ ⬛	Ground	06
07	GPIO04 (GPIO_GCLK)	⬛ ⬛	(TXD0) GPIO14	08
09	Ground	⬛ ⬛	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	⬛ ⬛	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	⬛ ⬛	Ground	14
15	GPIO22 (GPIO_GEN3)	⬛ ⬛	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	⬛ ⬛	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	⬛ ⬛	Ground	20
21	GPIO09 (SPI_MISO)	⬛ ⬛	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	⬛ ⬛	(SPI_CE0_N) GPIO08	24
25	Ground	⬛ ⬛	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	⬛ ⬛	(I2C ID EEPROM) ID_SC	28
29	GPIO05	⬛ ⬛	Ground	30
31	GPIO06	⬛ ⬛	GPIO12	32
33	GPIO13	⬛ ⬛	Ground	34
35	GPIO19	⬛ ⬛	GPIO16	36
37	GPIO26	⬛ ⬛	GPIO20	38
39	Ground	⬛ ⬛	GPIO21	40

Rev. 1.1  
16/07/2014

<http://www.element14.com>

# RASPBERRY PI

- Pode-se manipular diretamente o GPIO pelo terminal do SO, usando o SYSFS: pseudo sistema virtual de arquivos que faz interface entre os drivers, hardware e espaço do usuário. Ou seja, o usuário faz read/write num arquivo virtual, que faz a interação com o hardware ou o driver do dispositivo
- A GPIO é configurada por meio do comando export/unexport e pela definição da direção (INPUT, OUTPUT)
- Por exemplo, seja a GPIO21, pino 40, com um resistor e LED conectados. O comando abaixo

```
1 | $ echo "21" > /sys/class/gpio/export
```

Cria o diretório gpio21 dentro de /sys/class/gpio e os arquivos de interface

Para OUTPUT:

```
1 | $ echo "out" > /sys/class/gpio/gpio21/direction
```

Para INPUT:

```
1 | $ echo "in" > /sys/class/gpio/gpio21/direction
```

Os arquivos criados são:

```
1 | /sys/class/gpio/gpio21/direction
```

Determina se é GPIO de entrada ou saída

```
1 | /sys/class/gpio/gpio21/value
```

É o estado da GPIO

```
1 | /sys/class/gpio/gpio21/edge
```

Determina o momento da troca do estado

```
1 | /sys/class/gpio/gpio21/active_low
```

Inverte a lógica de leitura da GPIO

# RASPBERRY PI

Para escrever:

```
1 | $ echo "0" >/sys/class/gpio/gpio21/value
```

ou

```
1 | $ echo "1" >/sys/class/gpio/gpio21/value
```

Para ler:

```
1 | $ cat /sys/class/gpio/gpio21/value
```

<https://www.embarcados.com.br/raspberry-pi-ios-python/>

<https://www.filipeflop.com/blog/linguagem-c-com-raspberry-pi/>

- Existem contudo várias bibliotecas python, C, etc.

```
1 import RPi.GPIO as gpio
2 import time
3
4 # Configurando como BOARD, Pinos Fisicos
5 gpio.setmode(gpio.BOARD)
6
7 # Configurando a direcao do Pino
8 gpio.setup(11, gpio.OUT) # Usei 11 pois meu setmode é BOARD, se estive usando BCM seria 17
9 while True:
10 gpio.output(11, gpio.HIGH)
11 time.sleep(2)
12 gpio.output(11, gpio.LOW)
13 time.sleep(2)
14
15 # Desfazendo as modificações do GPIO
16 gpio.cleanup()
```

```
1 #include <stdio.h>
2 #include <wiringPi.h>
3
4 int main()
5 {
6     wiringPiSetup(); // inicia a biblioteca WiringPi
7     pinMode(1, OUTPUT); // configura o pino 1 como saída
8
9     printf("Exemplo 1 - Pisca LED\n"); //imprime mensagem na tela
10
11     while(1)
12     {
13         digitalWrite(1, HIGH); // liga o pino 1
14         delay(1000); // espera 1 segundo
15         digitalWrite(1, LOW); // desliga o pino 1
16         delay(1000); // espera 1 segundo
17     }
18     return 0;
19 }
```

**importante!** O pino 1 se refere ao pino da biblioteca que na verdade é o pino físico 12 da barra de pinos da Ras

# CONCLUSÕES

- As plataformas para desenvolvimento de sistemas embarcados crescem e atualizam com grande rapidez, incluindo mais recursos (memória, núcleos de processamento, periféricos etc.) e simultaneamente o número de sensores compatíveis e bibliotecas disponibilizadas para abstrair a complexidade do baixo nível.
- Ao desenvolvedor cabe escolher a melhor plataforma direcionada à determinada aplicação, ou mesmo avaliar a integração de plataformas para o melhor resultado que atenda às especificações do projeto, cliente, e ao custo total do projeto
- Desenvolver sistemas integrados hardware/software é motivador e um desafio com cada vez mais demanda no contexto IoT e de dispositivos móveis.