

Iniciando no ESP32 a partir da plataforma Arduino

Renan Tesch

Palestrante

Formação:

- Técnico em Eletromecânica (SENAI)
- Estudante de Engenharia Elétrica (UniMetrocamp)

Experiencia profissional:

- Desenvolvedor IoT - ART IT (Atual)
- Desenvolvedor de sistemas mainframe -IBM
(Janeiro a Novembro de 2019) ...

Experiencias “outras”:

- Sistemas Microprocessados (Microchip, STM, Espressif)
- Internet das Coisas
- Sensores e protocolos Industriais
- Eletrônica de potência



ÍNDICE

1. Conhecendo o microcontrolador ESP32.
2. Instalando o ESP32 no Arduino IDE.
3. Entradas digitais
4. Saídas digitais
5. Conversores ADC
6. Conversores DAC
7. PWM

Conhecendo o microcontrolador ESP32

CPU principal: Tensilica Xtensa LX6 microprocessor LX6 32-bit Dual-core, operando 240 MHz.

CPU secundário: ULP (Ultra Low Power co-processor) 8MHz e consome 150uA.

FLASH: 4MB.

RAM: 520kB.

GPIO: 34, com 3.3V e 12mA recomendável, máximo 40mA.

ADC: 18, com resolução de 12-bit.

DAC: 2, com resolução 8-bit.

WiFi: 2,4 GHz, IEEE 802.11 b/g/n.

Protocolos :PWM, I2C, SPI, I2S, CAN e etc;

Bluetooth: Bluetooth Low Energy v4.2 (BLE).

Timers: 4 de 64-bit.

Watchdogs: 4.

Sensores de Touch Capacitivo: 10.

Sensor de temperatura interno: 1.

Sensor de efeito Hall: 1.



Instalando o ESP32 no Arduino IDE

Requisitos:

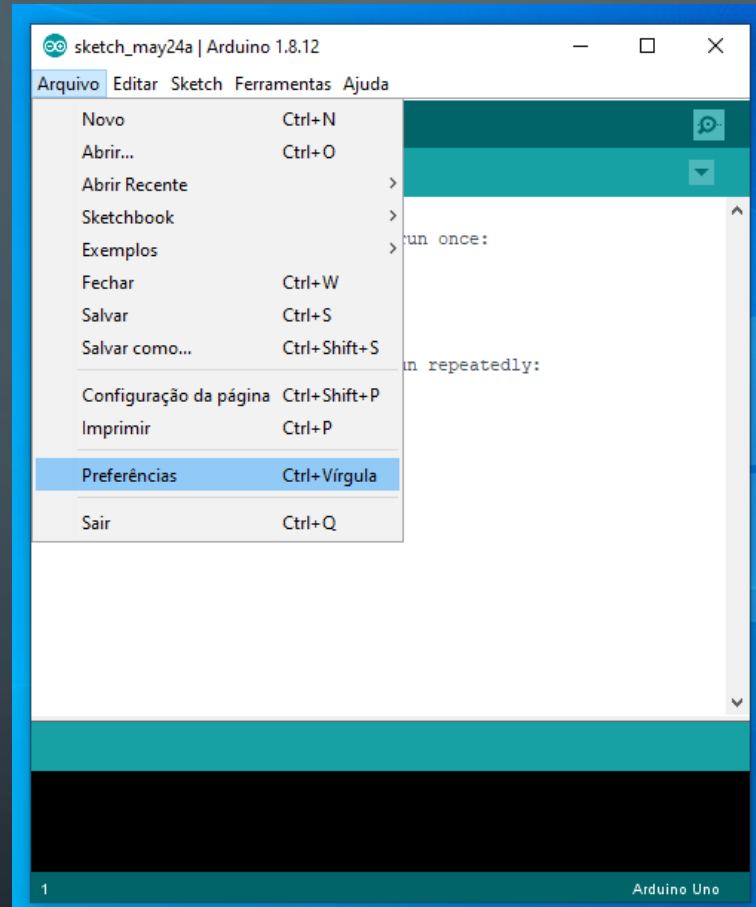
- Arduino IDE
- Microcontrolador ESP32
- Cabo USB tipo micro USB.
- PC

Instalar driver conversor USB-Serial “**Para Windows que não reconhece o ESP**”

1. Baixe acessando: [link do driver](#)
2. Baixe o driver correspondente a sua versão de S.O.
3. Descompacte os arquivos
4. Execute o instalador para a sua versão de S.O. (x64 para 64 bits e x86 para 32 bits)
5. Siga a instalação padrão

Instalando o ESP32 no Arduino IDE

Abra o Arduino IDE, clique em arquivo e depois em preferencias.

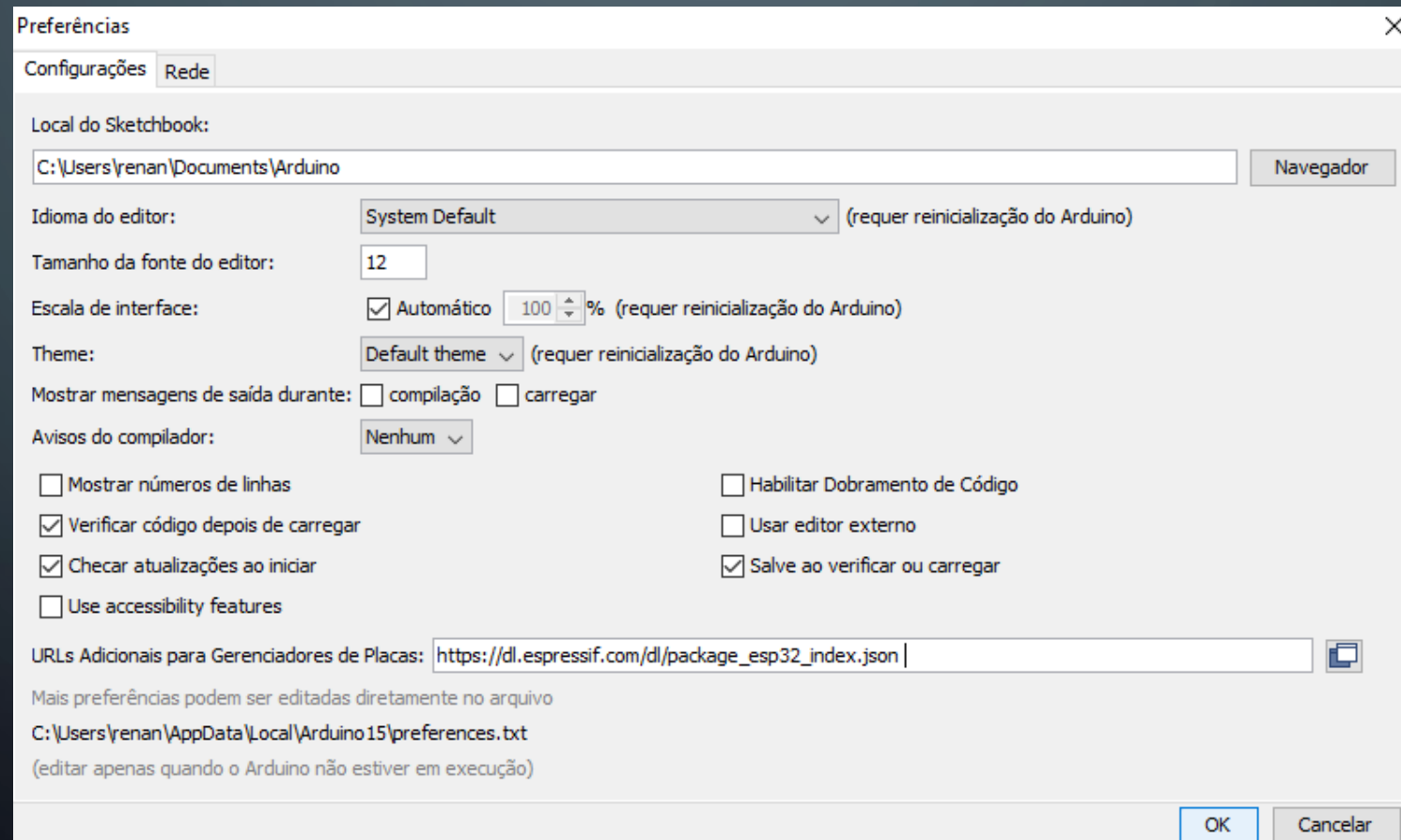


Instalando o ESP32 no Arduino IDE

Em seguida, adicione o seguinte link no campo de texto exibido conforme a imagem e clique em OK

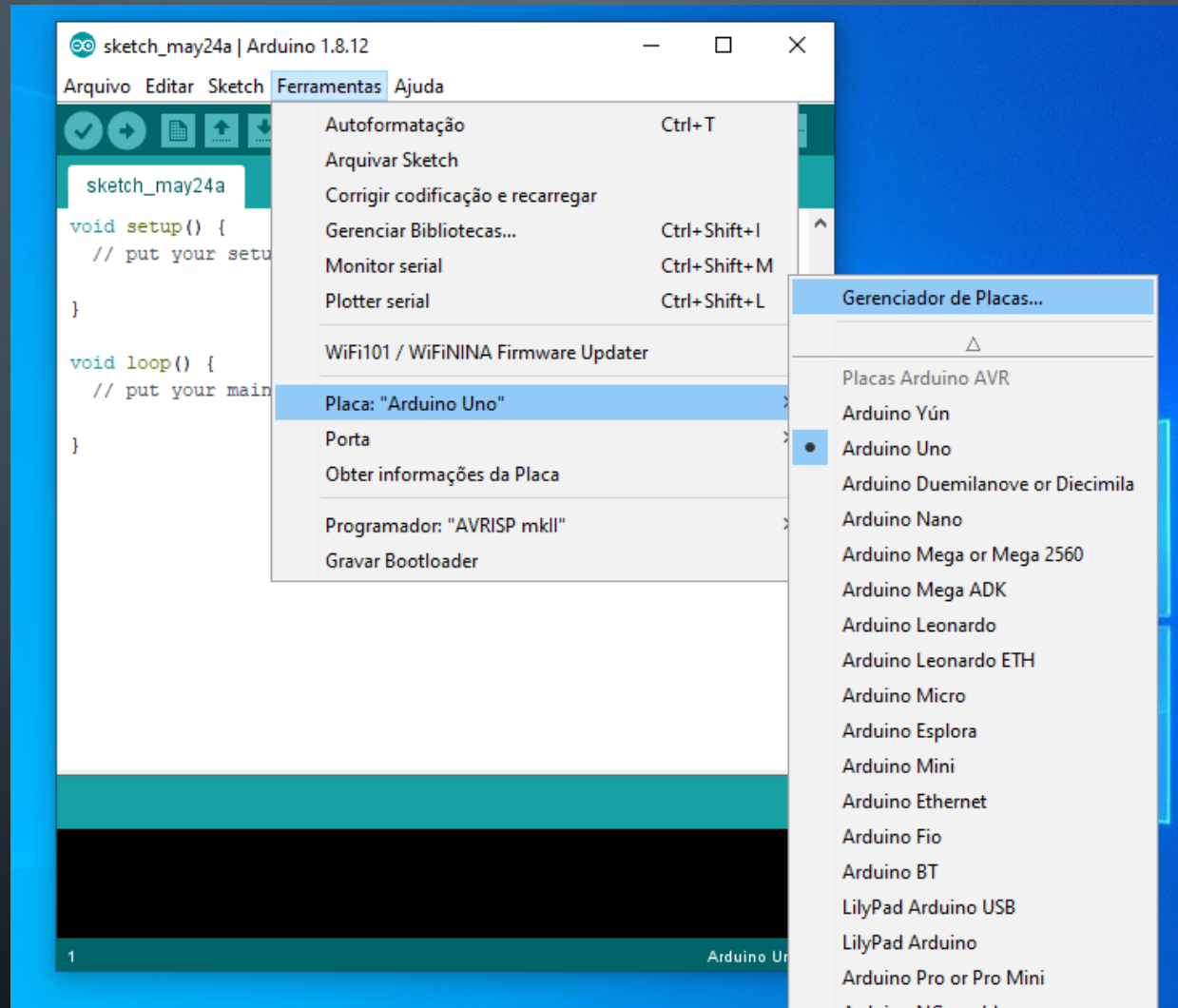
https://dl.espressif.com/dl/package_esp32_index.json

Você pode acrescentar mais links separando-os com uma vírgula ou quebra de linha



Instalando o ESP32 no Arduino IDE

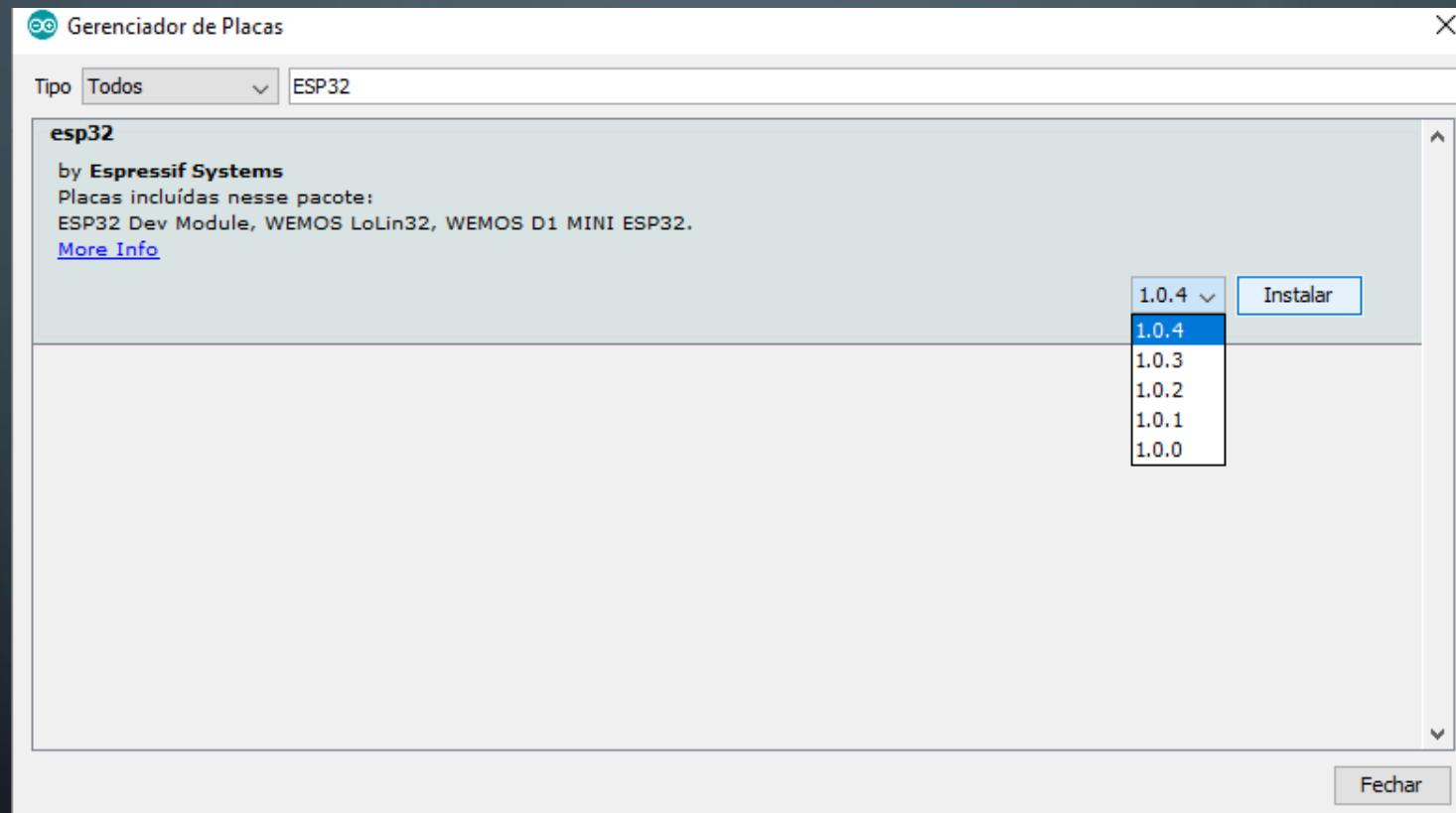
Agora clique na aba Ferramentas e em seguida **Gerenciador de Placas...**



Instalando o ESP32 no Arduino IDE

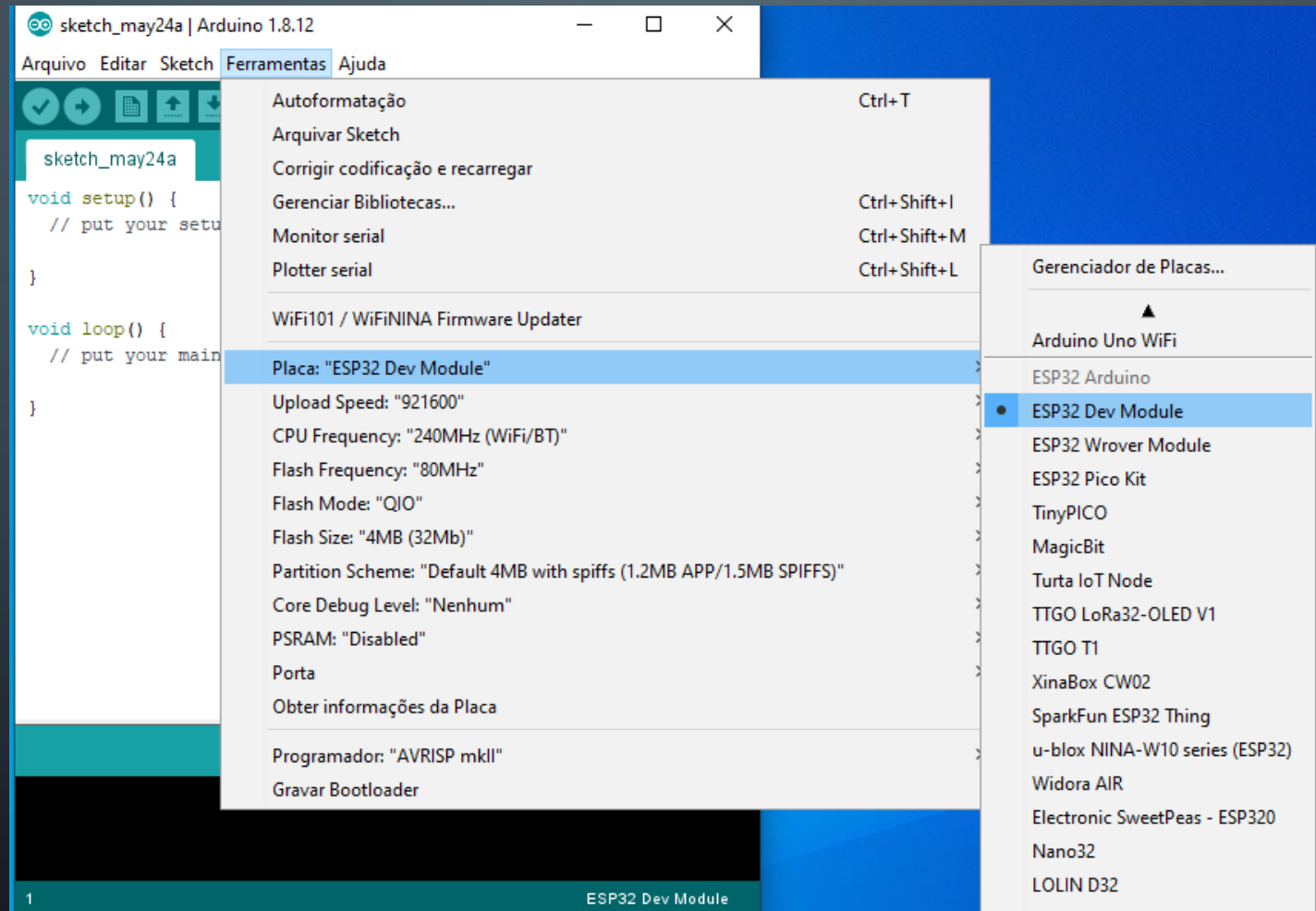
Na próxima etapa irá aparecer uma tela na qual você deve pesquisar “ESP32”, conforme a imagem a seguir, selecione a ultima versão e clique em instalar.

Obs. Este processo pode demorar e esta totalmente relacionado com a velocidade de conexão de sua internet.



Instalando o ESP32 no Arduino IDE

Após isso basta selecionar em Ferramentas, placa, o modelo **ESP32 Dev Module**, após isso o processo esta finalizado.



GitHub dos Exemplos

https://github.com/TeschRenan/PEX_ESP32

Entradas digitais

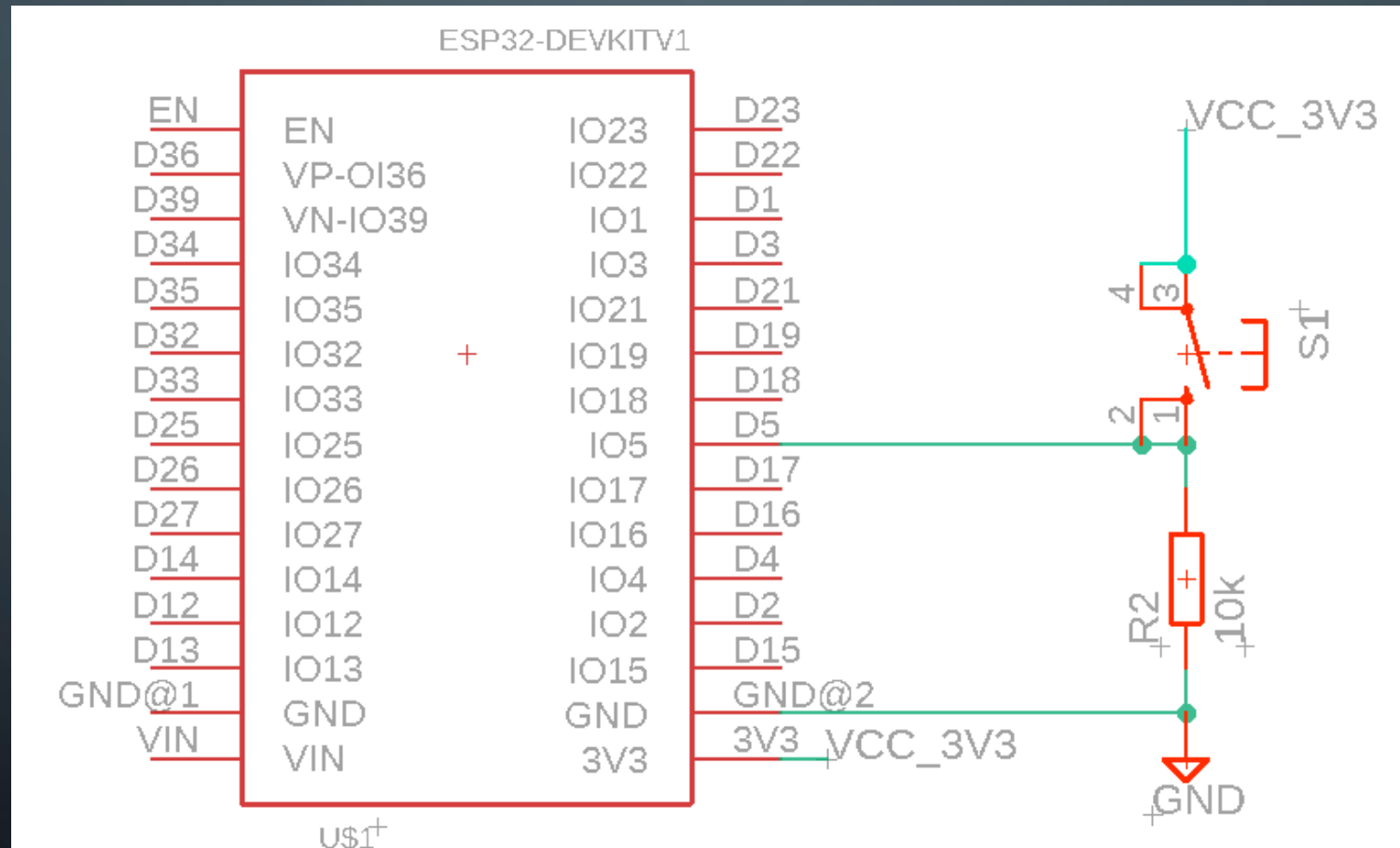
GPIO 0 ao 33 podem ser usados como Entrada e saídas

GPIO 34 ao 39 só podem ser usados como entradas.

Para utilizar com a placa de desenvolvimento é necessário validar o uso de cada pino, é recomendado acessar o link: <https://randomnerdtutorials.com/esp32-pinout-reference-gpios/>, para a validação dos pinos do ESP32.

Entradas digitais

Esquema de ligação:



Entradas digitais

Código usando funções do Arduino IDE.

Nome do arquivo: input_Arduino

```
#define pinoBotao 5
#define ledPin2 2

int statusBotao = 0;

void setup() {

    pinMode(ledPin2, OUTPUT);

    pinMode(pinoBotao, INPUT);
}

void loop() {

    statusBotao = digitalRead(pinoBotao);

    if (statusBotao == 1) {

        digitalWrite(ledPin2, HIGH);
    } else {

        digitalWrite(ledPin2, LOW);
    }
}
```

Entradas digitais

Código usando funções da IDF.

Nome do arquivo: input_IDF

```
#define pinoBotao GPIO_NUM_5
#define ledPin2 GPIO_NUM_2

int statusBotao = 0;

void setup() {

    gpio_pad_select_gpio(ledPin2);
    gpio_pad_select_gpio(pinoBotao);

    gpio_set_direction(ledPin2, GPIO_MODE_OUTPUT);
    gpio_set_direction(pinoBotao, GPIO_MODE_INPUT);

}

void loop() {

    statusBotao = gpio_get_level(pinoBotao);

    if (statusBotao == 1) {

        gpio_set_level(ledPin2, 1);
    } else {

        gpio_set_level(ledPin2, 0);
    }
}
```

Entradas digitais

Código usando funções do Arduino IDE com interrupções externas.

Nome do arquivo: input_Arduino_ISR

```
#define pinoBotao 5
#define ledPin2 2

bool botaoFlag = false;

void setup() {

    pinMode(ledPin2, OUTPUT);

    pinMode(pinoBotao, INPUT);

    /*
    Define quando a interrupção será acionada.
    Abaixo seguem as constantes predefinidas:
    LOW : aciona a interrupção sempre que o pino estiver baixo.
    CHANGE: aciona a interrupção sempre que o pino muda de estado.
    RISING: aciona a interrupção quando o pino vai de baixo para alto (LOW > HIGH).
    FALLING: para acionar a interrupção quando o pino vai de alto para baixo (HIGH > LOW)
    HIGH : aciona a interrupção sempre que o pino estiver alto.
    */
    attachInterrupt(pinoBotao, interrupcaoFunction, FALLING);
}
```

```
void interrupcaoFunction(){
    botaoFlag = true;
}

void loop() {

    if (botaoFlag == true){

        delay(1000);

        digitalWrite(ledPin2, HIGH);

        delay(1000);

        digitalWrite(ledPin2, LOW);
    }
}
```

Saídas digitais

Código usando funções do Arduino IDE.

Nome do arquivo: outputDigital_Arduino

```
|  
#define ledPin2 2  
  
void setup() {  
  pinMode(ledPin2, OUTPUT);  
}  
  
void loop() {  
  
  digitalWrite(ledPin2, HIGH);  
  delay(1000);  
  digitalWrite(ledPin2, LOW);  
}
```


Saídas digitais

Código usando funções da IDF.

Nome do arquivo: outputDigital_IDF

```
#define ledPin2 GPIO_NUM_2

void setup() {

    gpio_pad_select_gpio(ledPin2);

    gpio_set_direction(ledPin2, GPIO_MODE_OUTPUT);

}

void loop() {

    gpio_set_level(ledPin2, 1);
    delay(1000);
    gpio_set_level(ledPin2, 0);
    delay(1000);

}
```

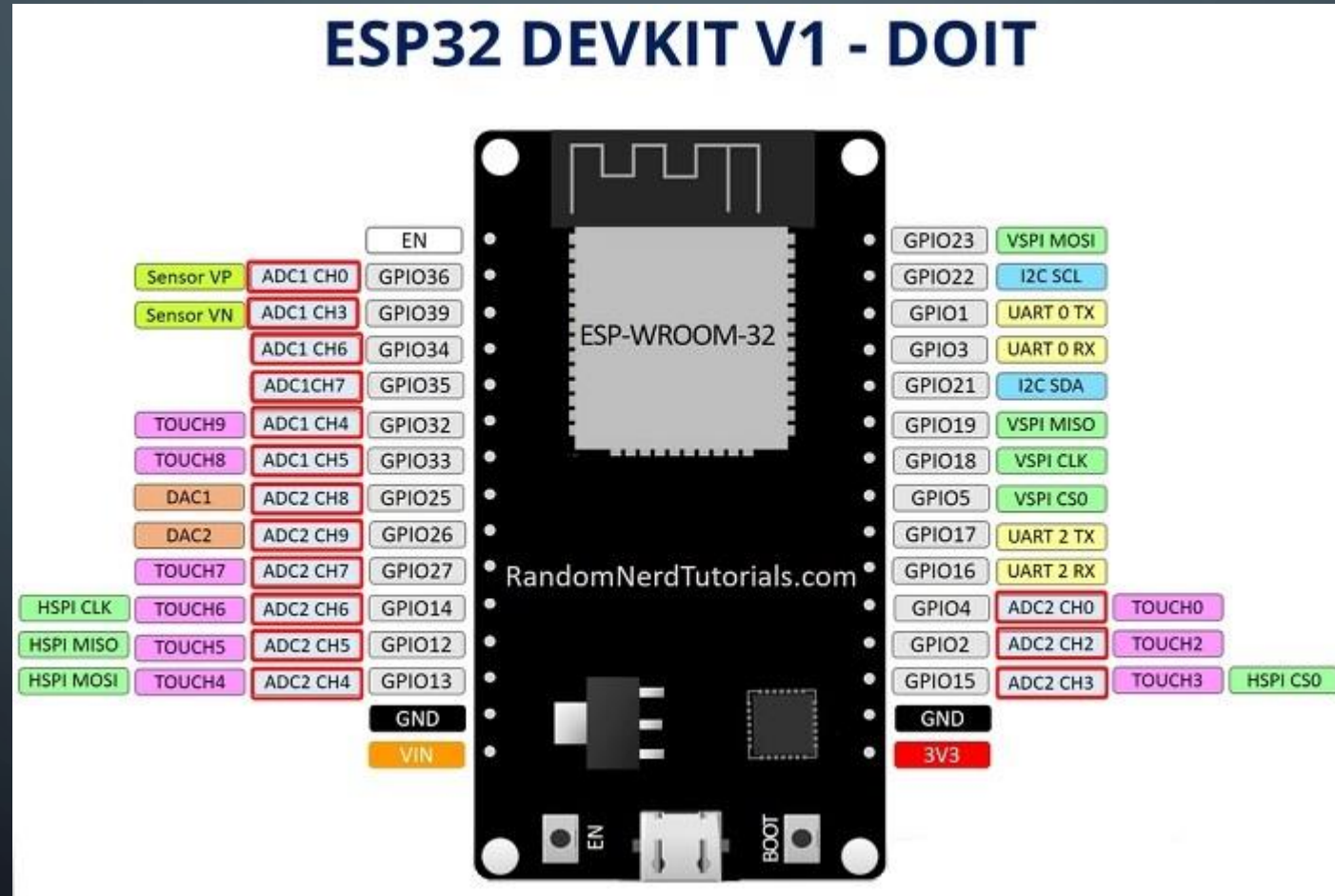
ADC

Características:

- Dividido em 2 canais, ADC1 GPIO 32 - 39 e ADC2 GPIO 0, 2, 4, 12 -15 e 25 – 27
- **ADC2** é usado pelo driver Wi-Fi. Portanto, o aplicativo pode usar o ADC2 apenas quando o driver Wi-Fi não foi iniciado.
- Alguns dos **ADC2** pinos são usados como pinos de inicialização “emitem PWM na inicialização” (GPIO 0, 2, 15), portanto, não podem ser usados livremente.
- ADC1 pode ser usado pelo ULP
- Resolução de amostragem de 9 a 12bits.
- Atenuação configurável de 0 dB (1,1 V), 2,5 dB (1,5 V), 6 dB (2,2 V), 11 dB (3,9 V limitado pelo VDD_A)

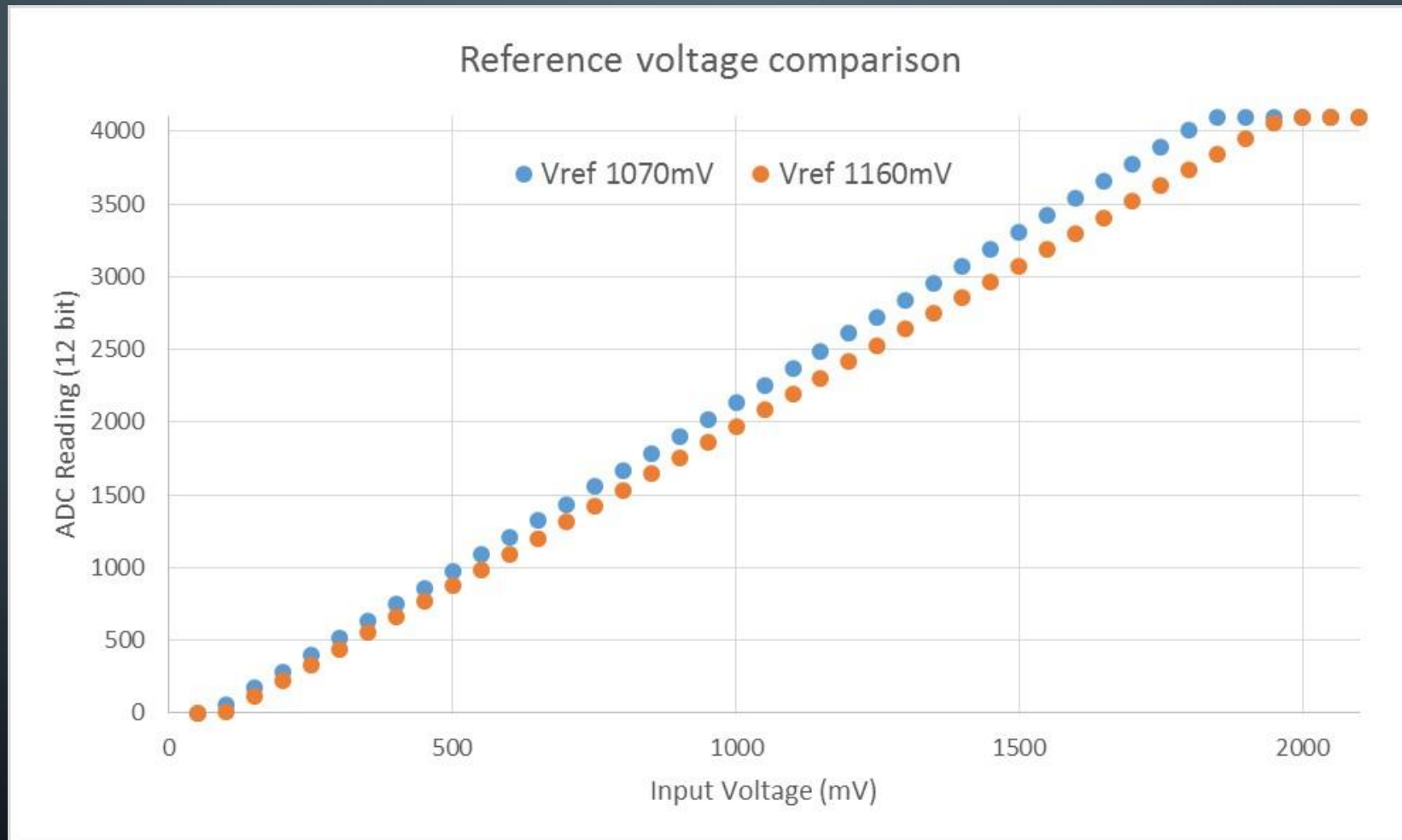
ADC

Disposições dos pinos no ESP32 DevKit V1



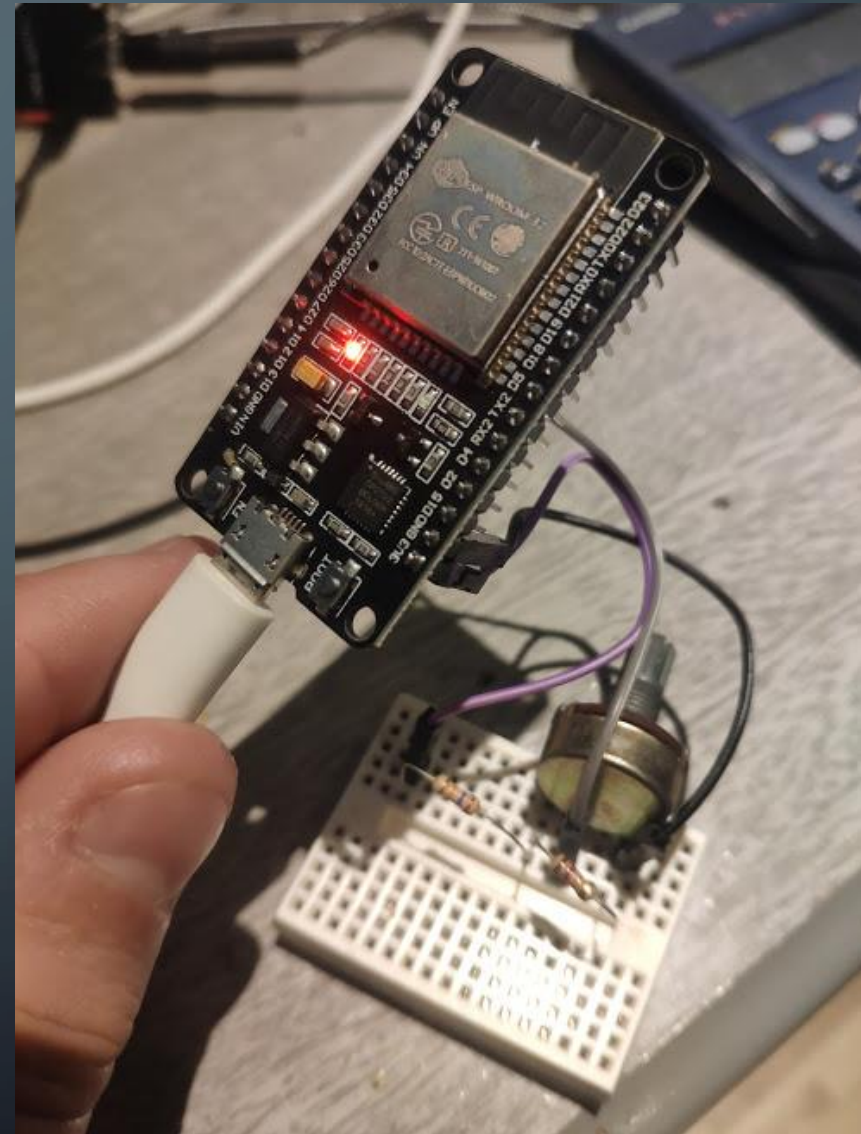
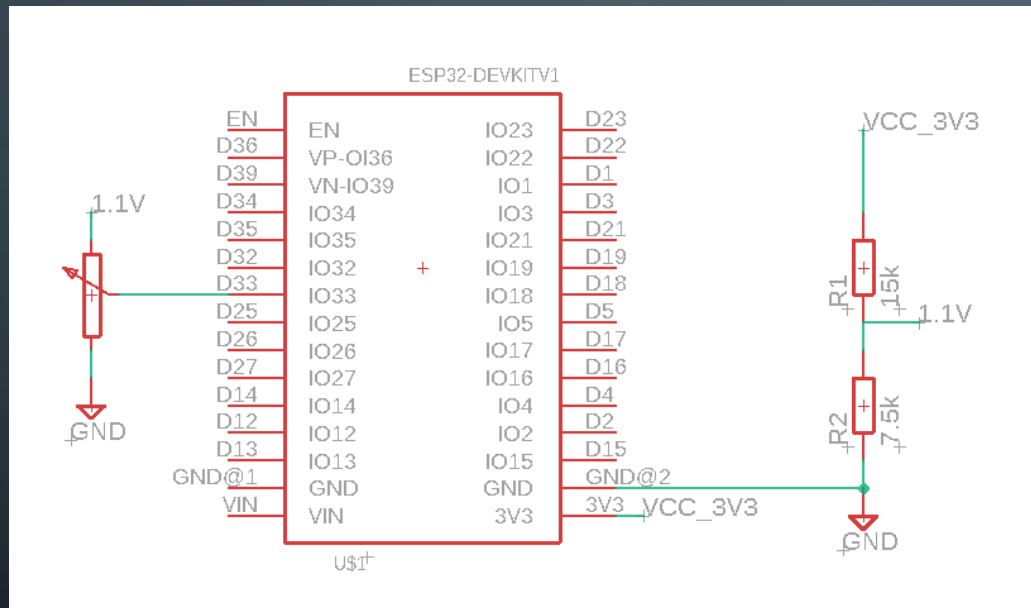
ADC

Comparações de amostragens ADC com dois ESP32



ADC

Esquema de ligação:



ADC

Código usando funções do Arduino IDE.

Nome do arquivo: analog_Arduino

```
// Entrada analogica conectada no GPIO 33 (Analog ADC1_CH5)
#define analogIn 33

//Variavel para guardar a somatoria das medições da entrada analogica
uint32_t analogValue = 0;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12);
  analogSetPinAttenuation(analogIn, ADC_0db);
}

void loop() {

  for (int i = 0; i < 100; i++)
  {
    analogValue += analogRead(analogIn);
    ets_delay_us(30);
  }
  analogValue /= 100;

  Serial.println(analogValue);
  delay(500);
}
```

COM4

```
21:06:05.885 -> 2034
21:06:06.395 -> 2017
21:06:06.878 -> 2001
21:06:07.396 -> 2018
21:06:07.875 -> 2021
21:06:08.389 -> 2007
21:06:08.902 -> 2015
21:06:09.384 -> 2013
21:06:09.902 -> 2015
21:06:10.383 -> 2000
21:06:10.898 -> 2014
21:06:11.382 -> 2015
21:06:11.893 -> 2003
21:06:12.373 -> 1776
21:06:12.886 -> 1422
21:06:13.400 -> 1379
21:06:13.879 -> 1370
21:06:14.395 -> 1366
21:06:14.878 -> 1443
21:06:15.394 -> 1371
21:06:15.878 -> 1392
21:06:16.391 -> 1369
21:06:16.902 -> 1369
21:06:17.384 -> 1378
21:06:17.902 -> 1371
21:06:18.385 -> 1376
21:06:18.901 -> 1364
21:06:19.377 -> 1374
21:06:19.899 -> 1378
21:06:20.405 -> 1369
21:06:20.895 -> 1360
21:06:21.382 -> 1369
21:06:21.904 -> 1376
21:06:22.402 -> 1378
21:06:22.886 -> 1377
21:06:23.398 -> 1376
21:06:23.885 -> 1372
```

ADC

Código usando funções da IDF.

Nome do arquivo: analog_IDF

Includes e informações do código.

```
#include <driver/adc.h>
#include <esp_adc_cal.h>

esp_adc_cal_characteristics_t adc_cal;//Estrutura que contem as informacoes para calibracao

//Codigo fonte extraido da Espressif IDF para medição do ADC1_CHANNEL_5 = GPIO 33, com atenuação de 0dB, faixa de leitura de 0 a 1.1V
```

ADC

Código usando funções da IDF.

Nome do arquivo: analog_IDF.

Função do programa: setup.

```
void setup() {  
  
    Serial.begin(115200);  
  
    adc1_config_width(ADC_WIDTH_BIT_12);  
    adc1_config_channel_atten(ADC1_CHANNEL_5, ADC_ATTEN_DB_0);  
  
    esp_adc_cal_value_t adc_type = esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_0, ADC_WIDTH_BIT_12, 1100, &adc_cal); // Inicializa a estrutura de calibração  
  
    if (adc_type == ESP_ADC_CAL_VAL_EFUSE_VREF)  
    {  
        Serial.println("ADC CALV ref eFuse encontrado: ");  
        Serial.print(adc_cal.vref);  
        Serial.print("mV");  
    }  
    else if (adc_type == ESP_ADC_CAL_VAL_EFUSE_TP)  
    {  
        Serial.println("ADC CAL Two Point eFuse encontrado");  
    }  
    else  
    {  
        Serial.println("ADC CAL Nada encontrado, utilizando Vref padrão: ");  
        Serial.print(adc_cal.vref);  
        Serial.print("mV");  
    }  
}
```

ADC

Código usando funções da IDF.

Nome do arquivo: analog_IDF.

Função do programa: loop.

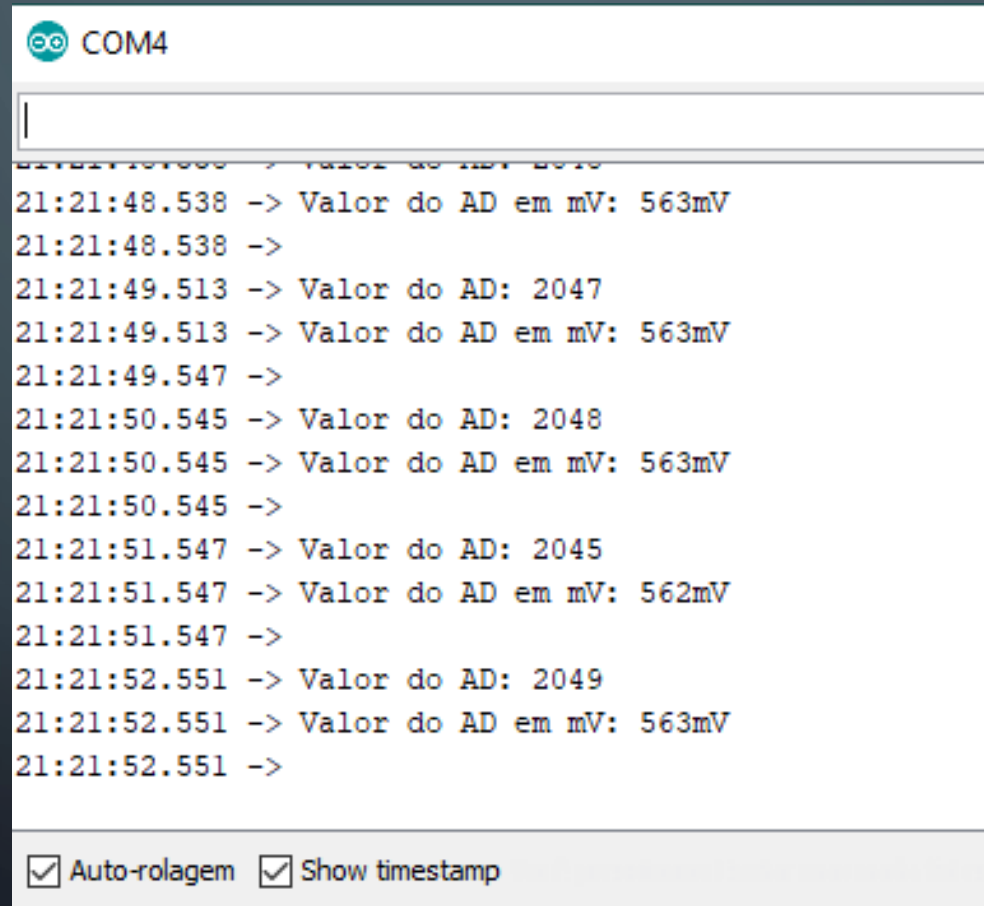
```
void loop() {  
    uint32_t AD = 0;  
    for (int i = 0; i < 100; i++)  
    {  
        AD += adc1_get_raw(ADC1_CHANNEL_5); //Obtem o valor RAW do ADC  
        ets_delay_us(30);  
    }  
    AD /= 100;  
  
    Serial.print("Valor do AD: ");  
    Serial.print(AD);  
    Serial.println("");  
    AD = esp_adc_cal_raw_to_voltage(AD, &adc_cal); //Converte e calibra o valor lido (RAW) para mV  
  
    Serial.print("Valor do AD em mV: ");  
    Serial.print(AD);  
    Serial.println("mV");  
    Serial.println("");  
    delay(1000);  
}
```

ADC

Código usando funções da IDF.

Nome do arquivo: analog_IDF.

Print da Serial.



The screenshot shows a serial terminal window with the title 'COM4'. The output consists of multiple lines of text, each starting with a timestamp followed by an arrow and a message. The messages alternate between 'Valor do AD: [integer]' and 'Valor do AD em mV: [float]'. The integer values are 2047, 2048, 2045, and 2049. The float values are 563mV, 563mV, 562mV, and 563mV. At the bottom of the window, there are two checkboxes: 'Auto-rolagem' and 'Show timestamp', both of which are checked.

```
COM4
21:21:48.538 -> Valor do AD em mV: 563mV
21:21:48.538 ->
21:21:49.513 -> Valor do AD: 2047
21:21:49.513 -> Valor do AD em mV: 563mV
21:21:49.547 ->
21:21:50.545 -> Valor do AD: 2048
21:21:50.545 -> Valor do AD em mV: 563mV
21:21:50.545 ->
21:21:51.547 -> Valor do AD: 2045
21:21:51.547 -> Valor do AD em mV: 562mV
21:21:51.547 ->
21:21:52.551 -> Valor do AD: 2049
21:21:52.551 -> Valor do AD em mV: 563mV
21:21:52.551 ->
☒ Auto-rolagem ☒ Show timestamp
```

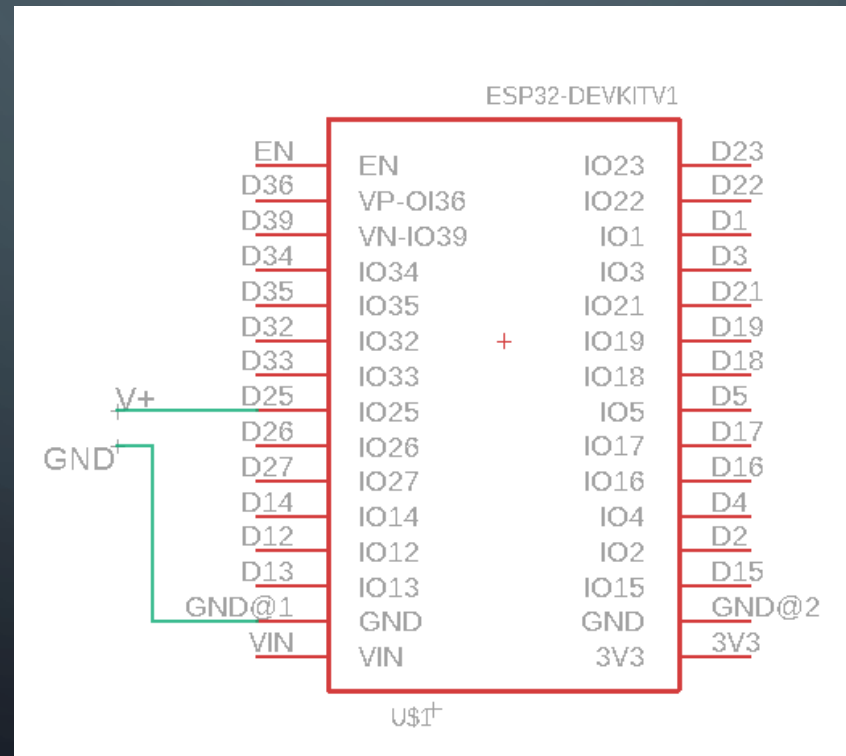

DAC

Características:

2 canais com resolução máxima de 8 bits e frequência máxima de 2MHz utilizando I2S.

Pinos disponíveis: DAC1 (GPIO25),DAC2 (GPIO26)

Esquemático:



DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: include e setup.

```
#define Dac1Pin 25
int32_t data[360] = {0};
int32_t data2[255] = {0};
int32_t inicio = 0;
int32_t fim = 0;

//16.67 / n bits da senoide

void setup() {

  Serial.begin(115200);

  //Monta o vetor da senoide de 0 a 360 Graus
  for (int16_t deg = 0; deg < 360; deg = deg + 1){
    data[deg] = (128 + 64 * (sin(deg*PI/180)));
    //OFFSET //Amplitude de sinal //graus para radianos
  }
  //Monta o vetor da onda triangular de 0 a 255, de 0 a 127 é a dente de serra.
  for (int16_t i = 0; i < 128; i++){
    data2[i] = i * 2;
  }
  for (int16_t i = 128; i < 255; i++){
    data2[i] = (255 - i) * 2;
  }
  inicio = micros();
  dacWrite(Dac1Pin, data[180]);
  fim = micros();
  Serial.print(fim-inicio);
}
```

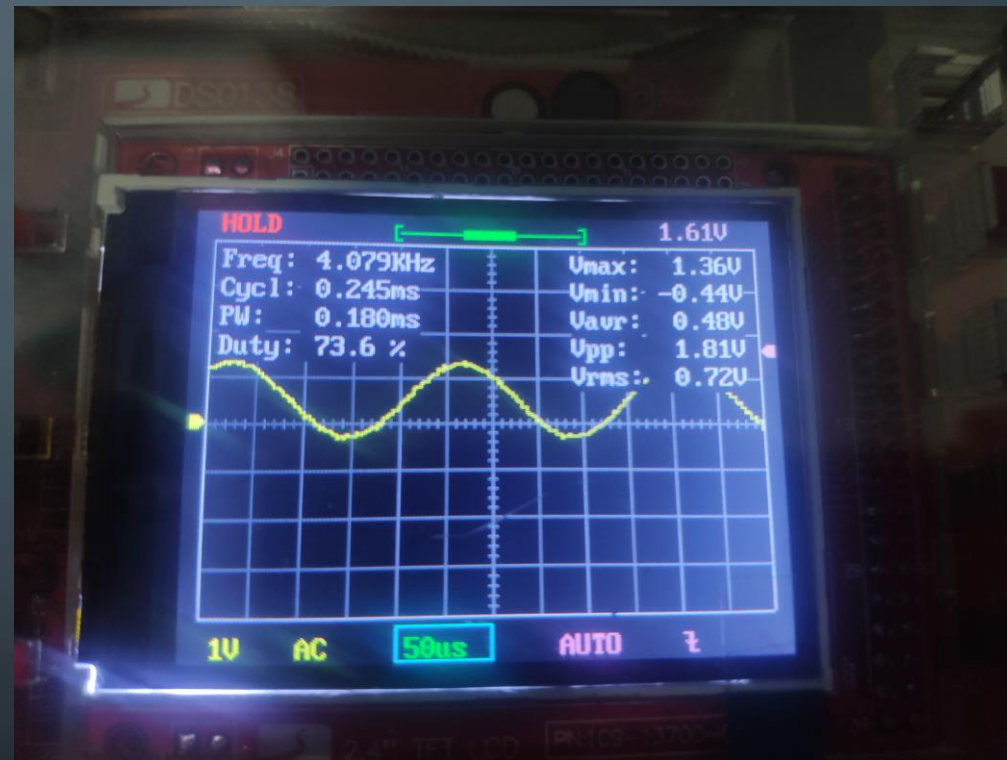
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, senoide de alta frequência e baixa resolução.

```
//Senoide de baixa resolução, frequência maxima 4kHz  
// a cada 8° um incremento  
  
for (int16_t deg = 0; deg < 360; deg = deg + 8){  
  dacWrite(Dac1Pin, data[deg]);  
}
```



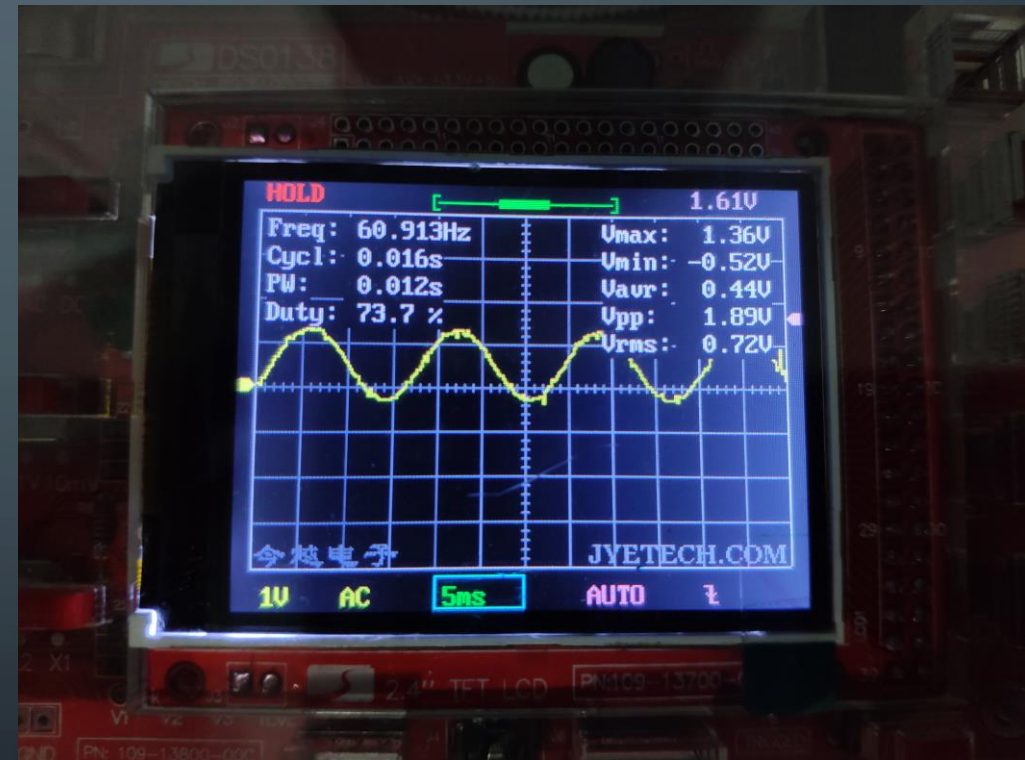
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, senoide de 60Hz baixa frequência e alta resolução.

```
//Senoide de alta resolução Frequencia maxima 160Hz  
// a cada 1º um incremento  
  
for (int16_t deg = 0; deg < 360; deg = deg + 1){  
    digitalWrite(Dac1Pin, data[deg]);  
    ets_delay_us(40);  
}
```



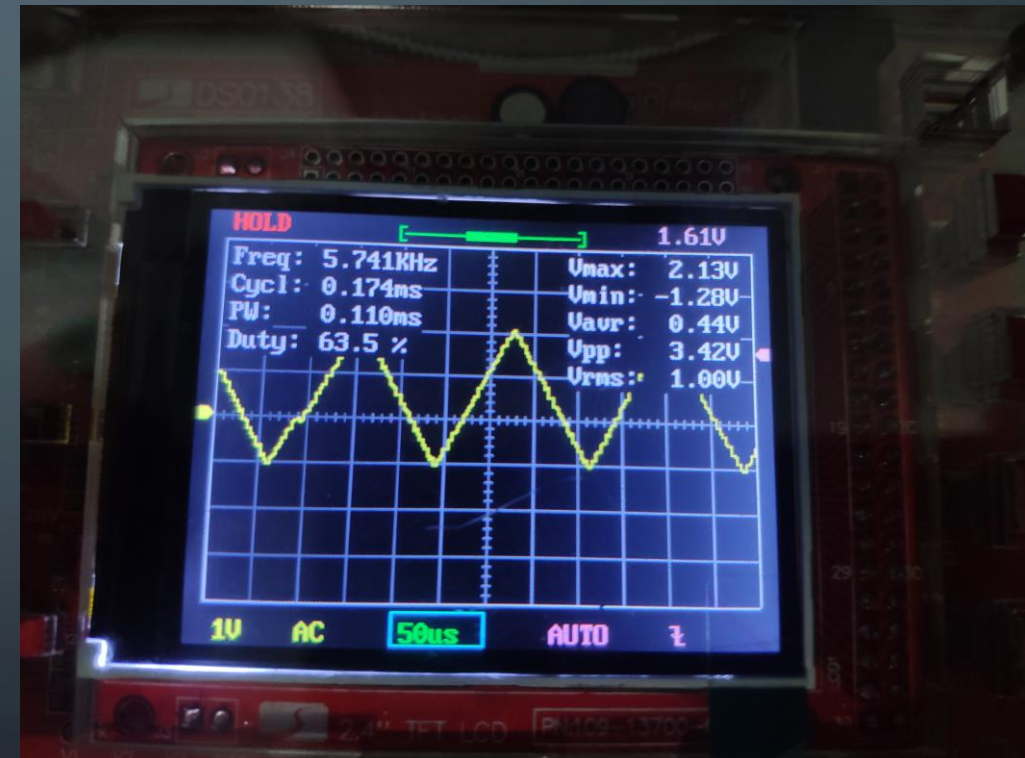
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, onda triangular de alta frequência e baixa resolução.

```
//Onda triangular de baixa resolução, Frequencia Maxima de 5.8KHz  
// a cada 8 bits de incremento  
  
for (int16_t i = 0; i < 255; i = i + 8){  
  dacWrite(Dac1Pin, data2[i]);  
}
```



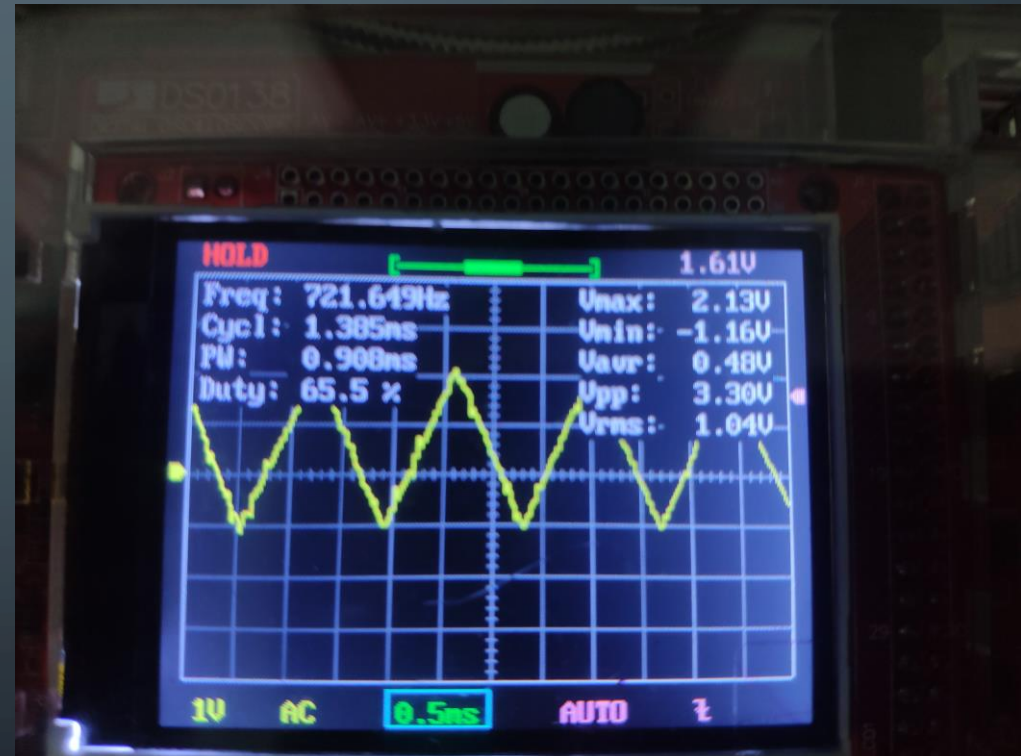
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, onda triangular de baixa frequência e alta resolução

```
//Onda triangular de alta resolução, Frequencia Maxima de 720Hz  
// a cada 1 bits de incremento  
  
for (int16_t i = 0; i < 255; i++){  
  dacWrite(Dac1Pin, data2[i]);  
}
```



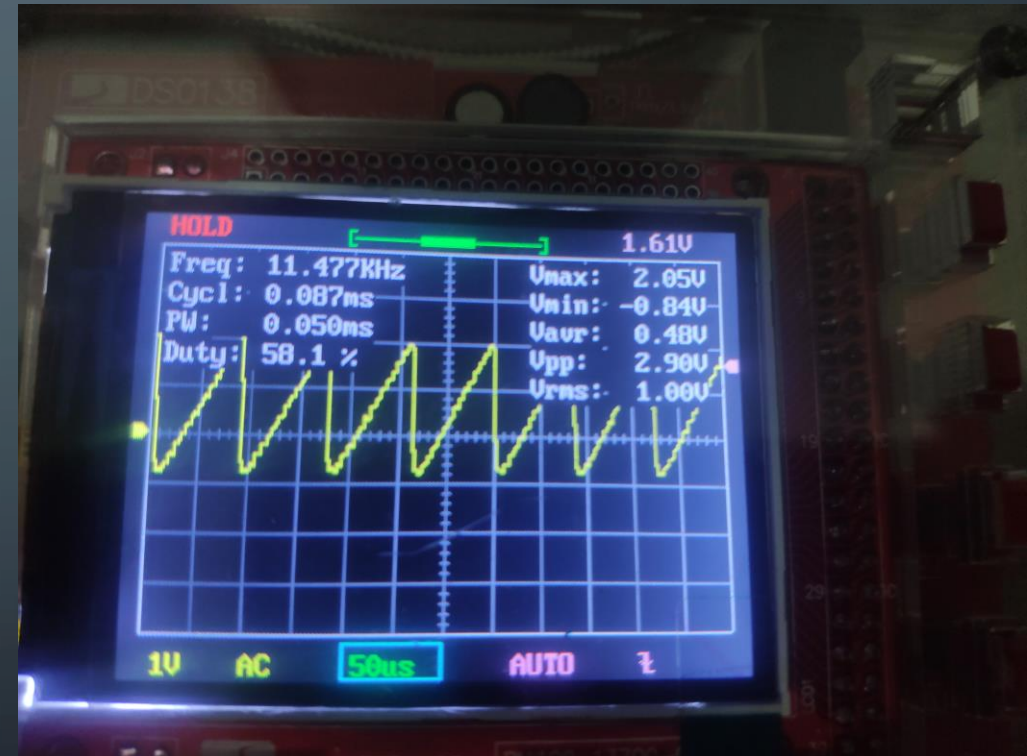
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, onda dente de serra de alta frequência e baixa resolução

```
//Dente de Serra de baixa resolução, Frequencia Maxima de 11.5KHz  
// a cada 8 bits de incremento  
  
for (int16_t i = 0; i < 128; i = i+8){  
  digitalWrite(Dac1Pin, data2[i]);  
}
```



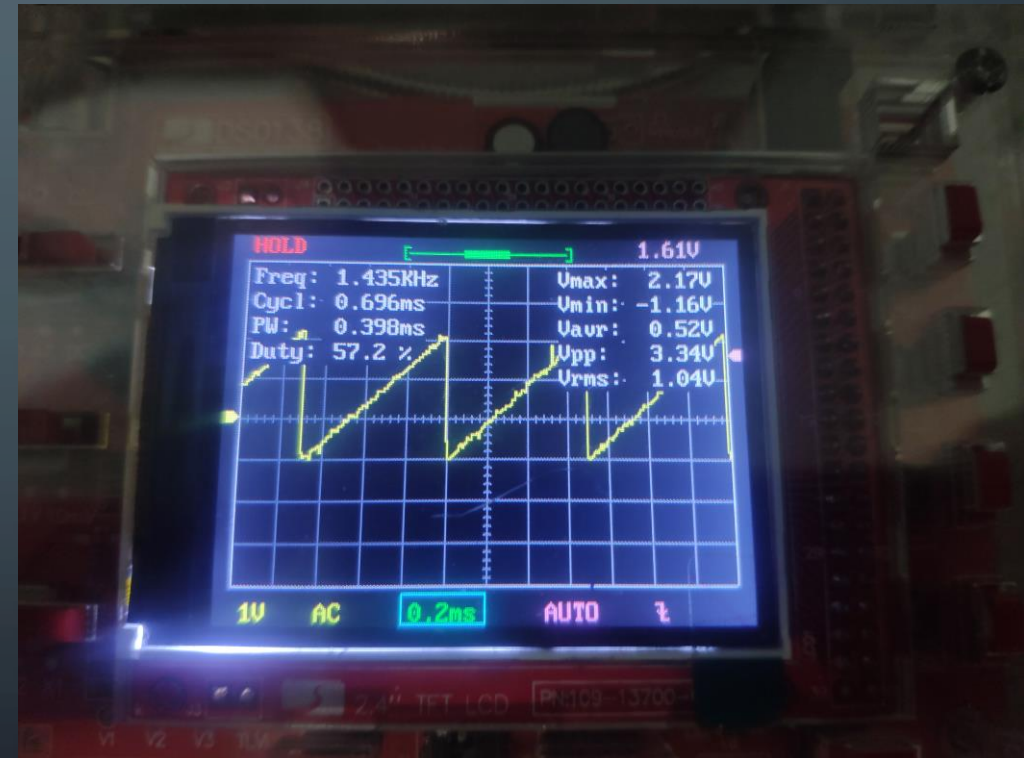
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_Arduino

Função do programa: loop, onda dente de serra de baixa frequência e alta resolução

```
//Dente de Serra de alta resolução, Frequencia Maxima de 1.45KHz  
// a cada 1 bits de incremento  
  
for (int16_t i = 0; i < 128; i++){  
  digitalWrite(Dac1Pin, data2[i]);  
}
```



DAC

Código usando funções da IDF.

Nome do arquivo: DAC_IDF

Função do programa: include e setup.

```
#include <driver/dac.h>
int32_t data[360] = {0};
int32_t data2[255] = {0};
int32_t inicio = 0;
int32_t fim = 0;

//16.67uS / n bits da senoide
```

```
void setup() {

    Serial.begin(115200);
    //Ativa o DAC
    dac_output_enable(DAC_CHANNEL_1);
    |
    //Monta o vetor da senoide de 0 a 360 Graus
    for (int16_t deg = 0; deg < 360; deg = deg + 1){
        data[deg] = (128 + 64 * (sin(deg*PI/180)));
        //OFFSET //Amplitude de sinal //graus para radianos
    }
    //Monta o vetor da onda triangular de 0 a 255, de 0 a 127 é a dente de serra.
    for (int16_t i = 0; i < 128; i++){
        data2[i] = i * 2;
    }
    for (int16_t i = 128; i < 255; i++){
        data2[i] = (255 - i) * 2;
    }

    inicio = micros();
    dac_output_voltage(DAC_CHANNEL_1, 255);
    fim = micros();
    Serial.print(value); //4uS
}
```

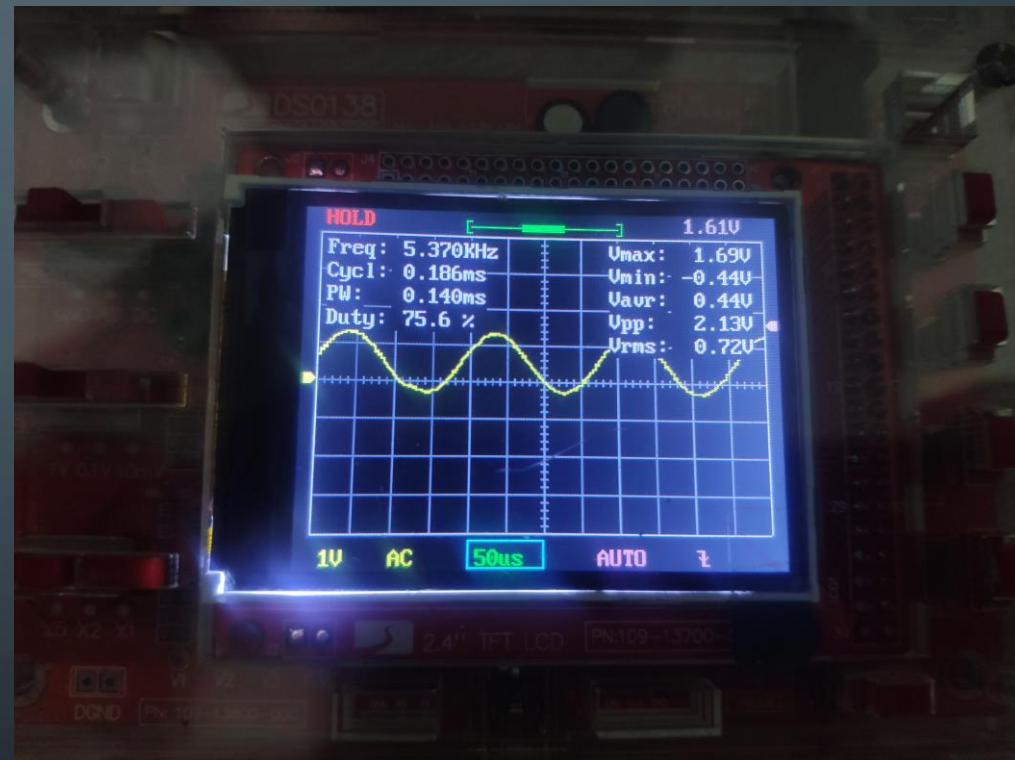
DAC

Código usando funções da IDF.

Nome do arquivo: DAC_IDF

Função do programa: loop, senoide de alta frequência e baixa resolução.

```
//Senoide de baixa resolução, frequencia maxima 5.4kHz  
// a cada 8° um incremento  
  
for (int16_t deg = 0; deg < 360; deg = deg + 8){  
    dac_output_voltage(DAC_CHANNEL_1, data[deg]);  
}
```



DAC

Código usando funções da IDF.

Nome do arquivo: DAC_IDF

Função do programa: loop, senoide de 60Hz baixa frequência e alta resolução.

```
//Senoide de alta resolução Frequencia maxima 670Hz
// a cada 1° um incremento

for (int16_t deg = 0; deg < 360; deg = deg + 1){
  dac_output_voltage(DAC_CHANNEL_1, data[deg]);
  ets_delay_us(42); //delay para 60HZ
}
```



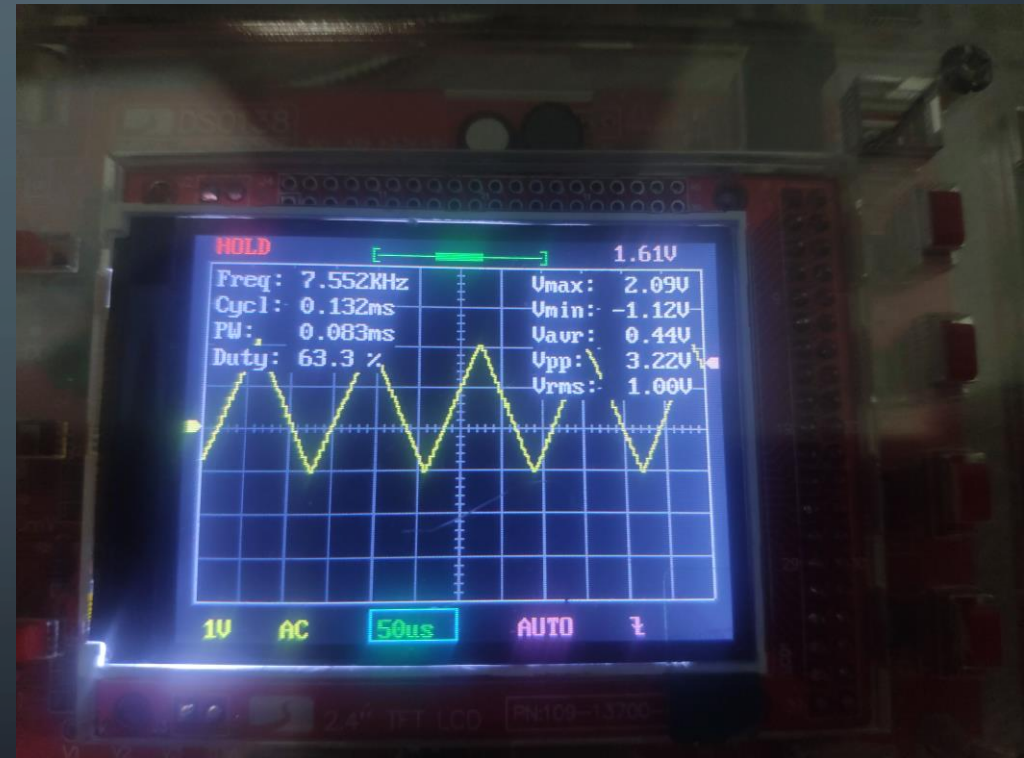
DAC

Código usando funções da IDF.

Nome do arquivo: DAC_IDF

Função do programa: loop, onda triangular de alta frequência e baixa resolução.

```
//Onda triangular de baixa resolução, Frequencia Maxima de 7.7KHz  
// a cada 8 bits de incremento  
  
for (int16_t i = 0; i < 255; i = i + 8){  
    dac_output_voltage(DAC_CHANNEL_1, data2[i]);  
}
```



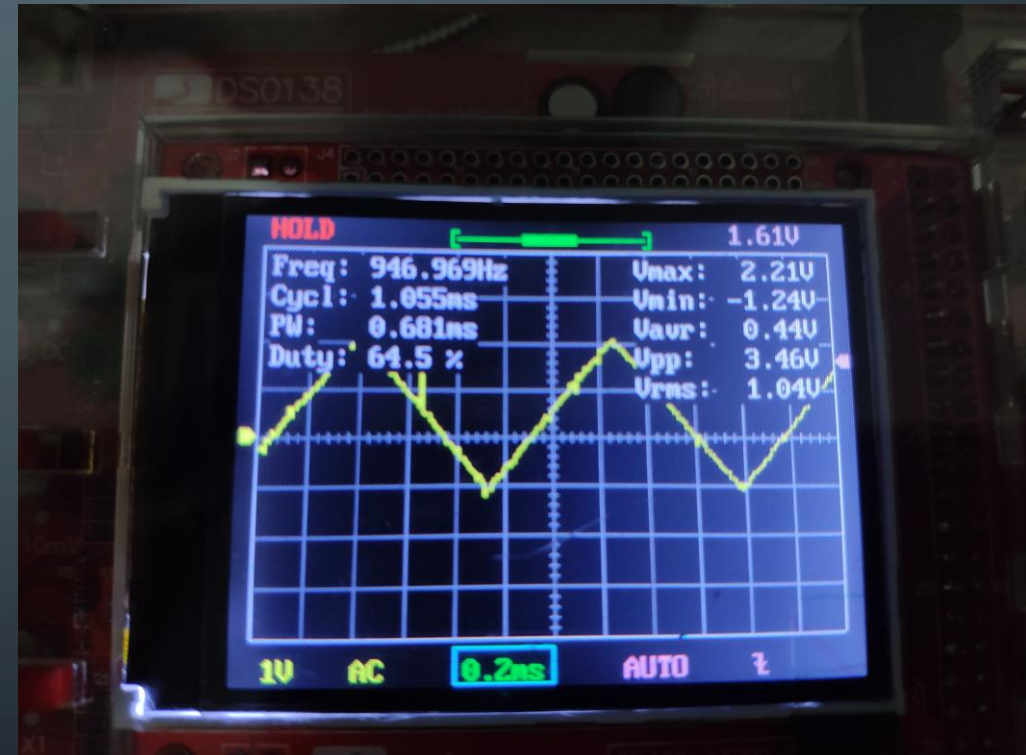
DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_IDF

Função do programa: loop, onda triangular de baixa frequência e alta resolução

```
//Onda triangular de alta resolução, Frequencia Maxima de 950Hz  
// a cada 1 bits de incremento  
  
for (int16_t i = 0; i < 255; i++){  
  dac_output_voltage(DAC_CHANNEL_1, data2[i]);  
}
```



DAC

Código usando funções do Arduino IDE.

Nome do arquivo: DAC_IDF

Função do programa: loop, onda dente de serra de alta frequência e baixa resolução

```
//Dente de Serra de baixa resolução, Frequencia Maxima de 15KHz  
// a cada 8 bits de incremento  
  
for (int16_t i = 0; i < 128; i = i+8){  
  dac_output_voltage(DAC_CHANNEL_1, data2[i]);  
}
```



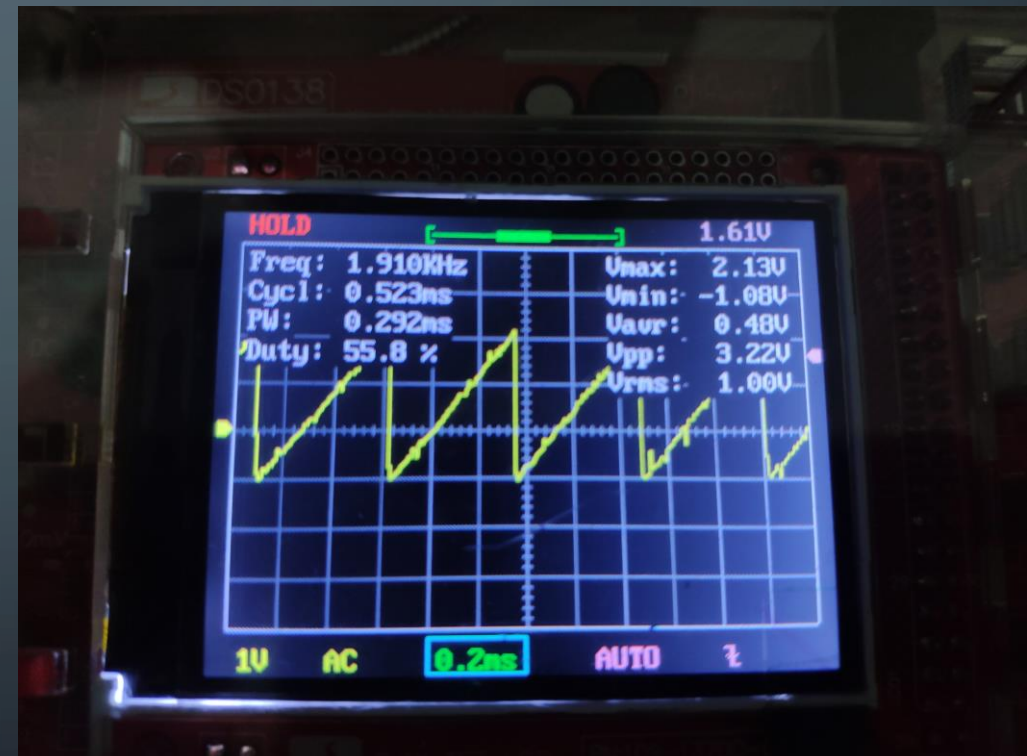
DAC

Código usando funções da IDF.

Nome do arquivo: DAC_IDF

Função do programa: loop, onda dente de serra de baixa frequência e alta resolução

```
//Dente de Serra de alta resolução, Frequencia Maxima de 1.9KHz  
// a cada 1 bits de incremento  
  
for (int16_t i = 0; i < 128; i++){  
    dac_output_voltage(DAC_CHANNEL_1, data2[i]);  
}
```



PWM “LED Control”

Características:

16 pinos separados por 2 canais de 8 GPIO's cada com resolução máxima de 13 bits e frequência máxima de 40MHZ com 1bit de resolução.

2 timers sendo um de alta velocidade implementado em hardware e outro de baixa velocidade implementado em software.

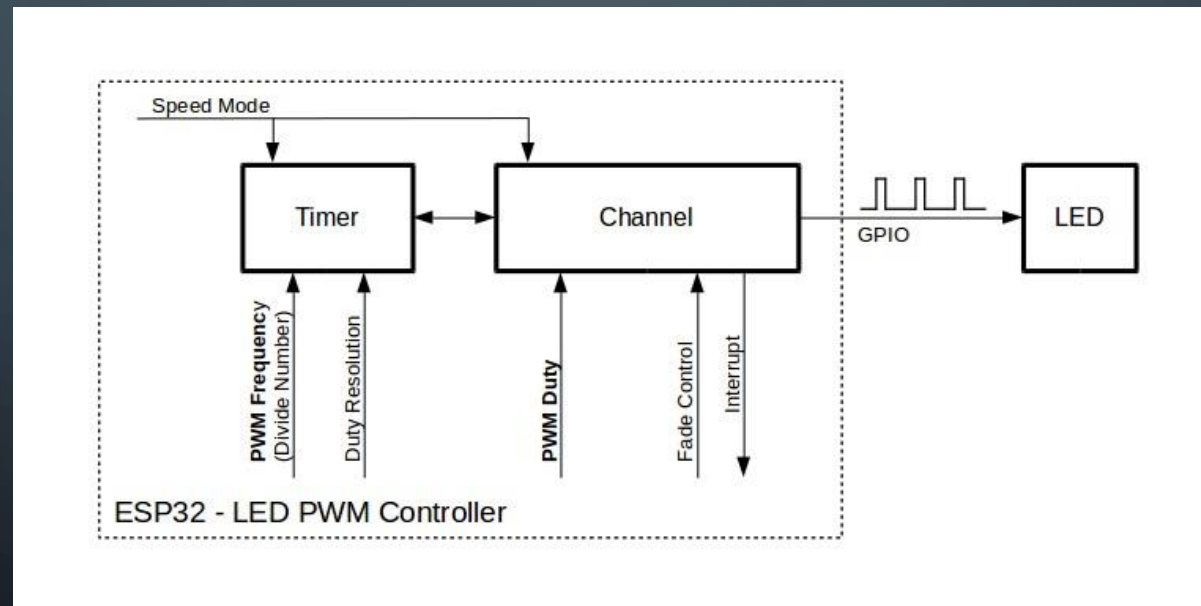
PWM “LED Control”

Procedimentos de uso:

Configure o Timer especificando o a frequência, duty cycle e resolução

Configure o Canal associe o timer ao GPIO para gerar a saída o sinal **PWM**.

Mudança de Sinal PWM Isso pode ser feito sob o controle total do software ou com as funções controle por software.



DAC

Código usando funções do Arduino IDE.

Nome do arquivo: PWM_Arduino

```
#define ledPin 15 // 15 == GPIO15

//Propriedades do PWM
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup(){
  //Configura o canal, frequencia e resolução do PWM.
  ledcSetup(ledChannel, freq, resolution);

  // Configura o pino para o Canal 0.
  ledcAttachPin(ledPin, ledChannel);
}

void loop(){
  //Incrementa o duty
  for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }

  //decrementa o duty
  for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
  }
}
```


DAC

Código usando funções da IDF.

Nome do arquivo: PWM_IDF

Função do programa: include e setup.

```
#include "driver/ledc.h"

void setup() {

    ledc_timer_config_t ledc_timer;
    ledc_timer.duty_resolution = LEDC_TIMER_8_BIT;    // resolução em Bits do PWM
    ledc_timer.freq_hz        = 5000;                // Frequencia do sinal de PWM
    ledc_timer.speed_mode     = LEDC_HIGH_SPEED_MODE; // Timer a ser usado, high speed ou low speed
    ledc_timer.timer_num      = LEDC_TIMER_0;         // Indexador do Timer
    // ledc_timer.clk_cfg      = LEDC_AUTO_CLK;        // Auto select the source clock

    // Configura o Timer 0 para high speed
    ledc_timer_config(&ledc_timer);

    ledc_channel_config_t ledc_channel;
    ledc_channel.channel = LEDC_CHANNEL_0;
    ledc_channel.duty    = 0;
    ledc_channel.gpio_num = 15;
    ledc_channel.speed_mode = LEDC_HIGH_SPEED_MODE;
    ledc_channel.timer_sel = LEDC_TIMER_0;
    ledc_channel.hpoint    = 1;
    // Configura o Canal 0 para iniciar.
    ledc_channel_config(&ledc_channel);

    ledc_set_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0, 0);
    ledc_update_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0);
}
```

DAC

Código usando funções da IDF.

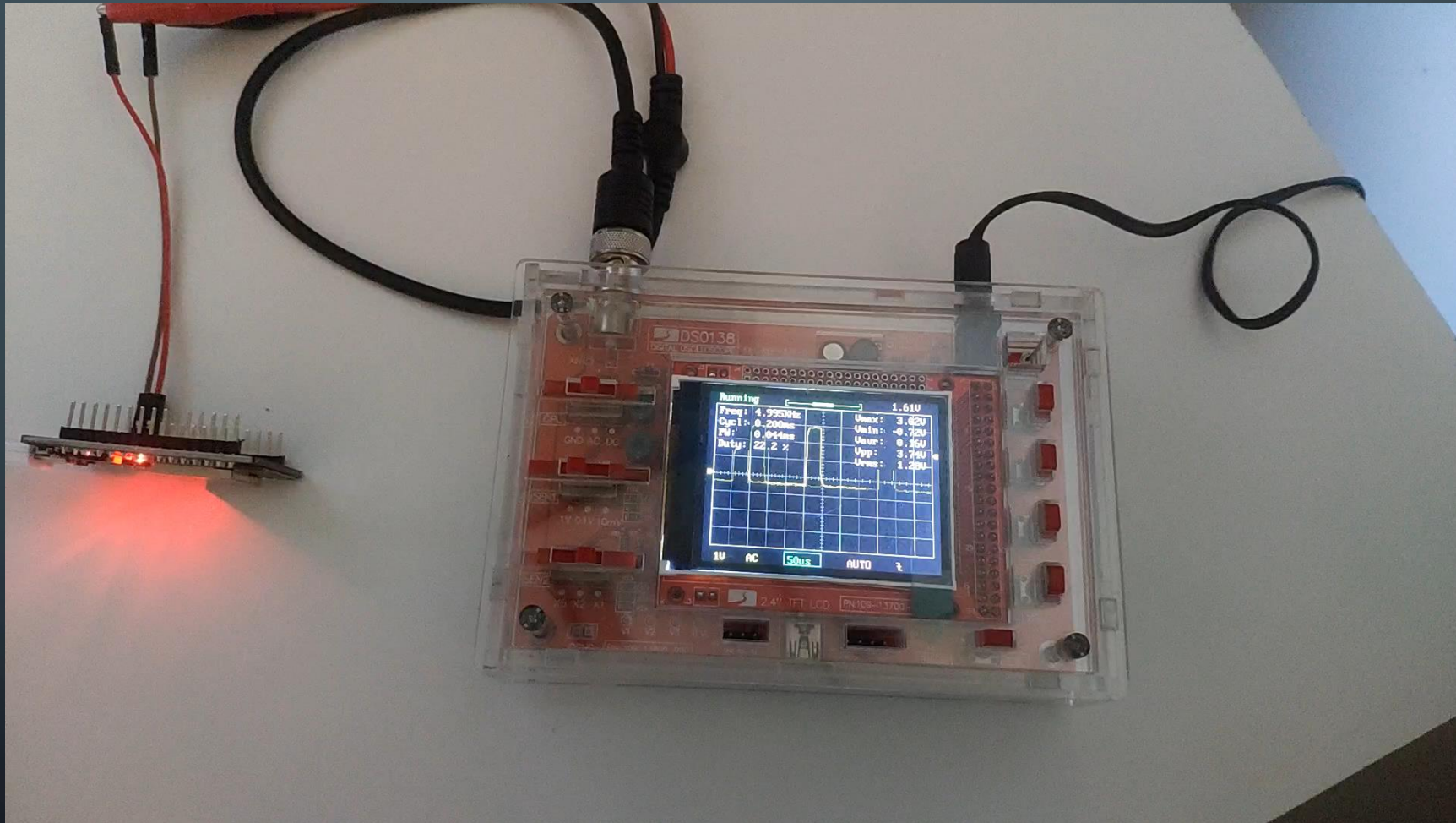
Nome do arquivo: PWM_IDF

Função do programa: loop.

```
void loop(){  
    //Incrementa o duty  
    for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){  
        ledc_set_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0, dutyCycle);  
        ledc_update_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0);  
        delay(15);  
    }  
  
    //decrementa o duty  
    for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){  
        ledc_set_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0, dutyCycle);  
        ledc_update_duty(LEDC_HIGH_SPEED_MODE, LEDC_CHANNEL_0);  
        delay(15);  
    }  
}
```

DAC

Vídeo demonstrativo do DAC



Referencias

Entrada e saída digitais: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/gpio.html?highlight=gpio>

ADC: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html?highlight=adc>

DAC: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2s.html?highlight=dac>

PWM : <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html?highlight=pwm>

The background is a dark blue gradient with a large, faint, light blue circle in the center. In the four corners, there are decorative white line art elements resembling circuit boards or neural network connections, with lines and small circles.

Perguntas?