PADO

Labs

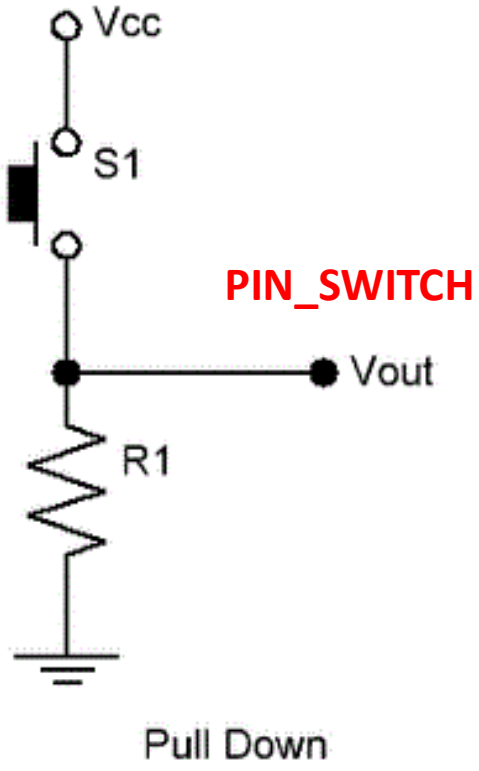
Microcontroladores Aplicados a IoT

Dilson Liukiti Ito

10 – Botão, PWM e Websocket

Microcontroladores Aplicados a IoT

Botão com Polling




```
gpio_pad_select_gpio(PIN_SWITCH);
gpio_set_direction(PIN_SWITCH, GPIO_MODE_INPUT);
gpio_pulldown_en(PIN_SWITCH);
gpio_pullup_dis(PIN_SWITCH);
```

```
while (true)
{
    ESP_LOGI(TAG, "Botão %d: %s", PIN_SWITCH,
                    gpio_get_level(PIN_SWITCH)?"pressionado":"solto");
    vTaskDelay(100);
}
```

https://www.researchgate.net/figure/Figura-4-Pulsador-en-pull-up-y-en-pull-down_fig4_310621668

gpio_config_t

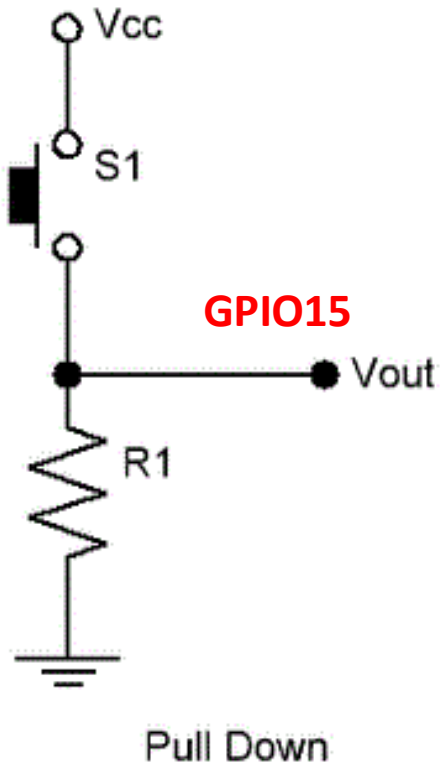

```
gpio_pad_select_gpio(PIN_SWITCH);
gpio_set_direction(PIN_SWITCH, GPIO_MODE_INPUT);
gpio_pulldown_en(PIN_SWITCH);
gpio_pullup_dis(PIN_SWITCH);
```

```
gpio_config_t io_conf = {
    .intr_type = GPIO_INTR_DISABLE,
    .pin_bit_mask = (1ULL<<PIN_SWITCH),
    .mode = GPIO_MODE_INPUT,
    .pull_up_en = 0,
    .pull_down_en = 1
};
gpio_config(&io_conf);
```

Exemplo GPIO Polling

- Baixe os arquivos da Aula10.zip
- Rode o arquivo polling.c
- Quais as vantagens/desvantagens do Polling?

Botão com Interrupção




```
in_conf.intr_type = GPIO_INTR_POSEDGE;
```

```
//install gpio isr service
gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT);
//hook isr handler for specific gpio pin
gpio_isr_handler_add(PIN_SWITCH, gpio_isr_handler, (void*) PIN_SWITCH);
```

```
static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    uint32_t gpio_num = (uint32_t) arg;
    xQueueSendFromISR(gpio_evt_queue, &gpio_num, NULL);
}
```

https://www.researchgate.net/figure/Figura-4-Pulsador-en-pull-up-y-en-pull-down_fig4_310621668

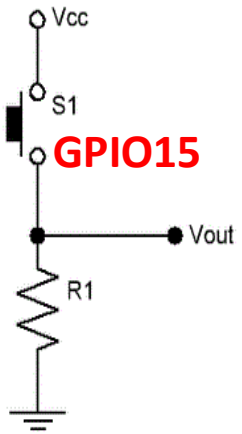
Exemplo GPIO Interrupção

- Rode o arquivo interrupt.c

- Há algum problema?

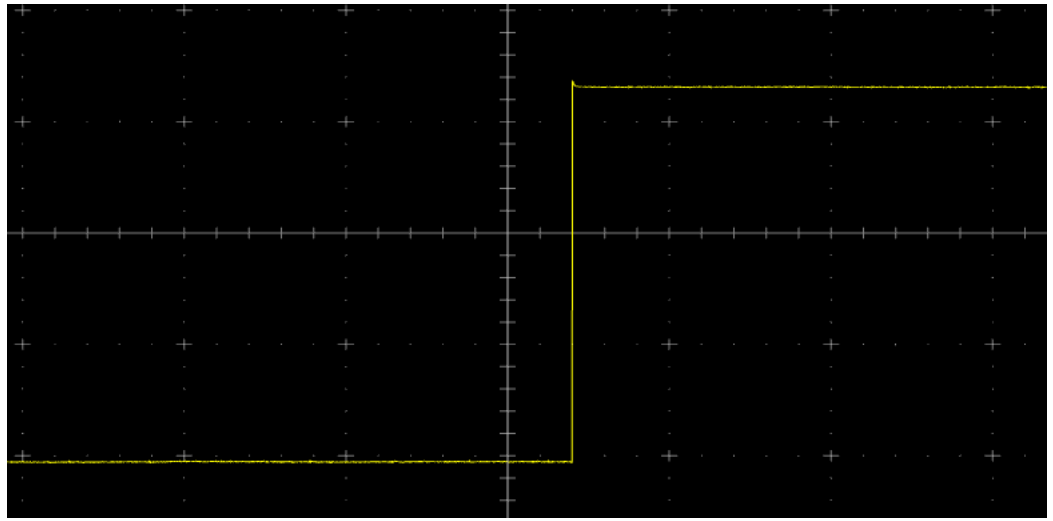
Bounce

- Expectativa

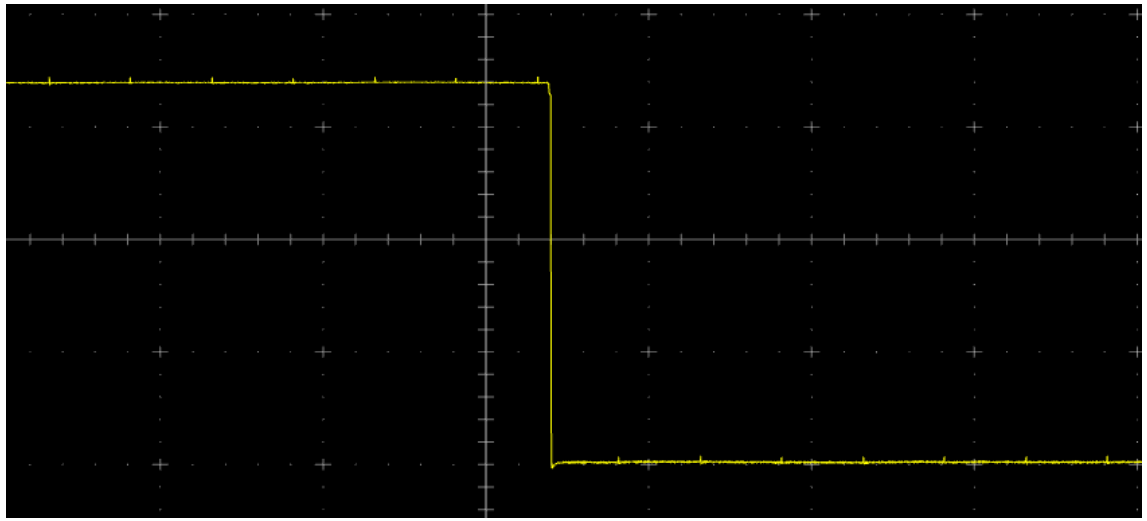


Pull Down

Botão sendo pressionado



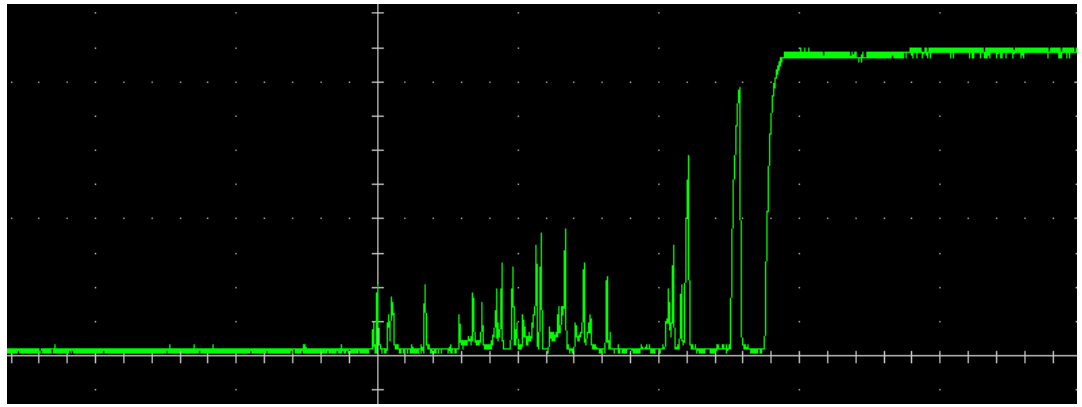
Botão sendo solto



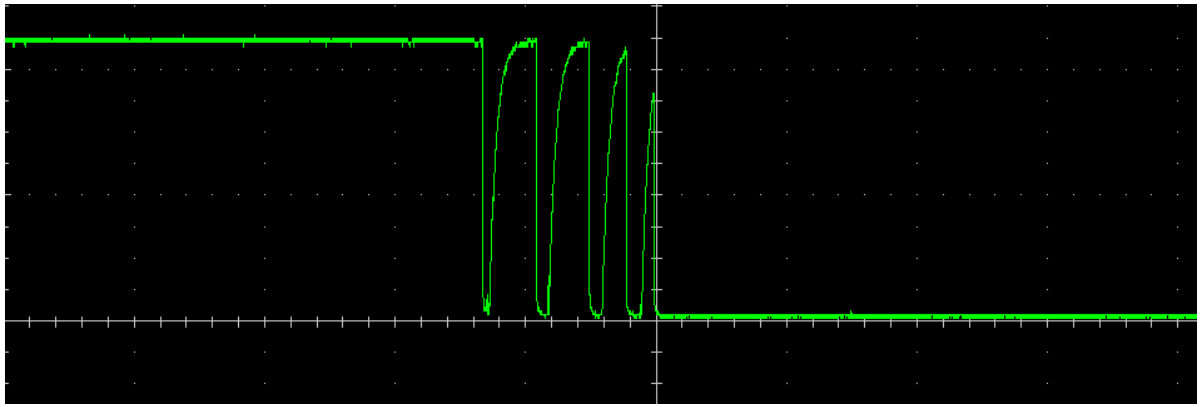
Bounce (cont.)

- Realidade

Botão sendo pressionado



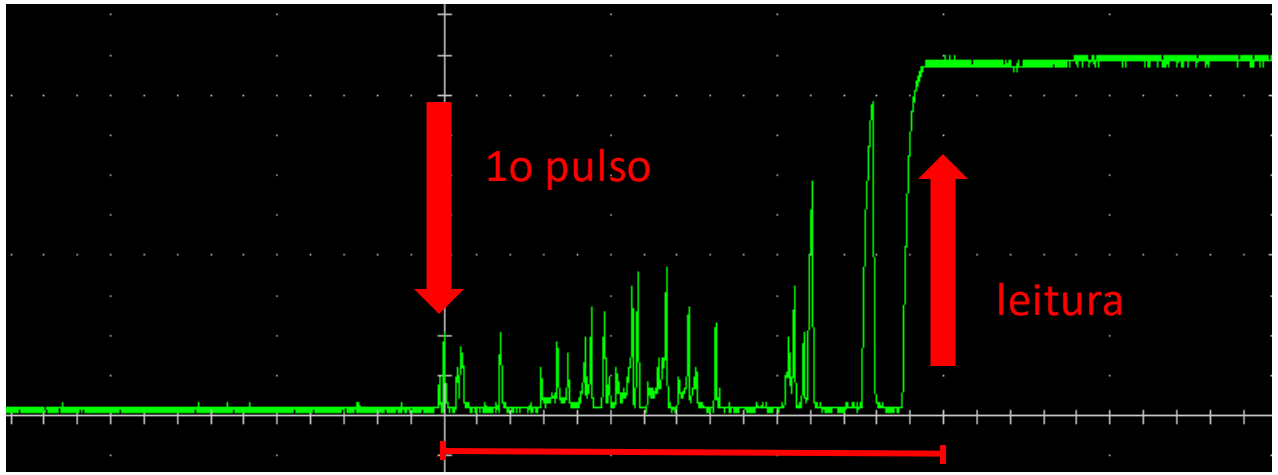
Botão sendo solto



https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

Debounce por Software

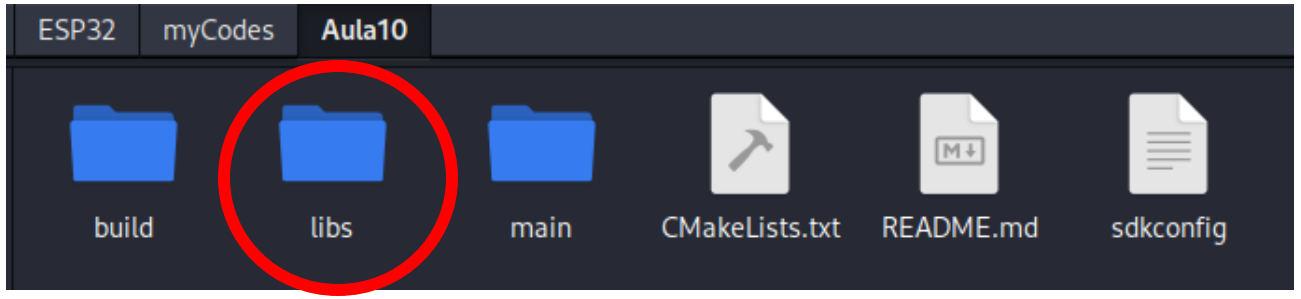
- Interrupção é acionada no primeiro pulso;
- Desabilita interrupção e inicia timer de debounce;
- Após timer estourar, fazer leitura do pino para determinar estado;
- Habilita interrupção;



https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

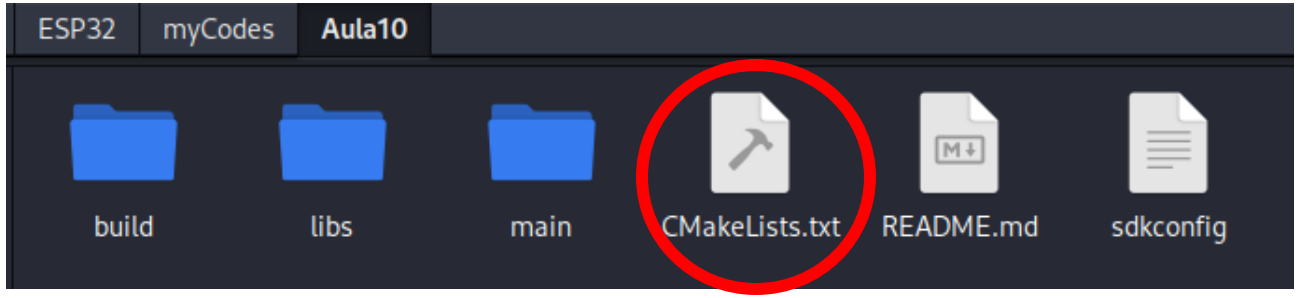
Lib button

- Para adicionar a lib button:
  - Se ainda não existir, crie uma pasta "libs" no diretório raiz do projeto



Verifique CMakeLists.txt da pasta principal

- Se ainda não existir, adicione o seguinte comando ao arquivo CMakeLists.txt do diretório raiz do projeto




```
# For more information about build system see
# https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/build-system.html
# The following five lines of boilerplate have to be in your project's
# CMakeLists in this exact order for cmake to work correctly
cmake_minimum_required(VERSION 3.5)

set(EXTRA_COMPONENT_DIRS "libs/.")

include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(Aula10)
```

Clone git button

- Dentro da pasta libs, faça o clone da biblioteca button

```
┌──(dilson㉿kali)-[~/…/ESP32/myCodes/Aula10/libs]
└─$ git clone git@github.com:liukiti/button.git
```

buttonInit


```
button_init_t button = {
    .buttonEventHandler = button_handler,
    .pin_bit_mask = (1ULL<<PIN_SWITCH),
    .pull_up_en = 0,
    .pull_down_en = 1
};

buttonInit(&button);
```

button_handler()


```
void button_handler (button_queue_t param) {
   static uint32_t cnt = 0;
   char tag[30]  = {0};
   sprintf(tag, "button_handler %d", cnt++);

   switch(param.buttonState)
   {
      case BTN_PRESSED:
      ESP_LOGI(tag, "botão %d %s", param.buttonPin, "pressionado");
      break;

      case BTN_RELEASED:
      ESP_LOGI(tag, "botão %d %s", param.buttonPin, "solto");
      break;

      case BTN_HOLD:
      ESP_LOGI(tag, "botão %d %s", param.buttonPin, "mantido pressionado");
      break;
     }
}
```

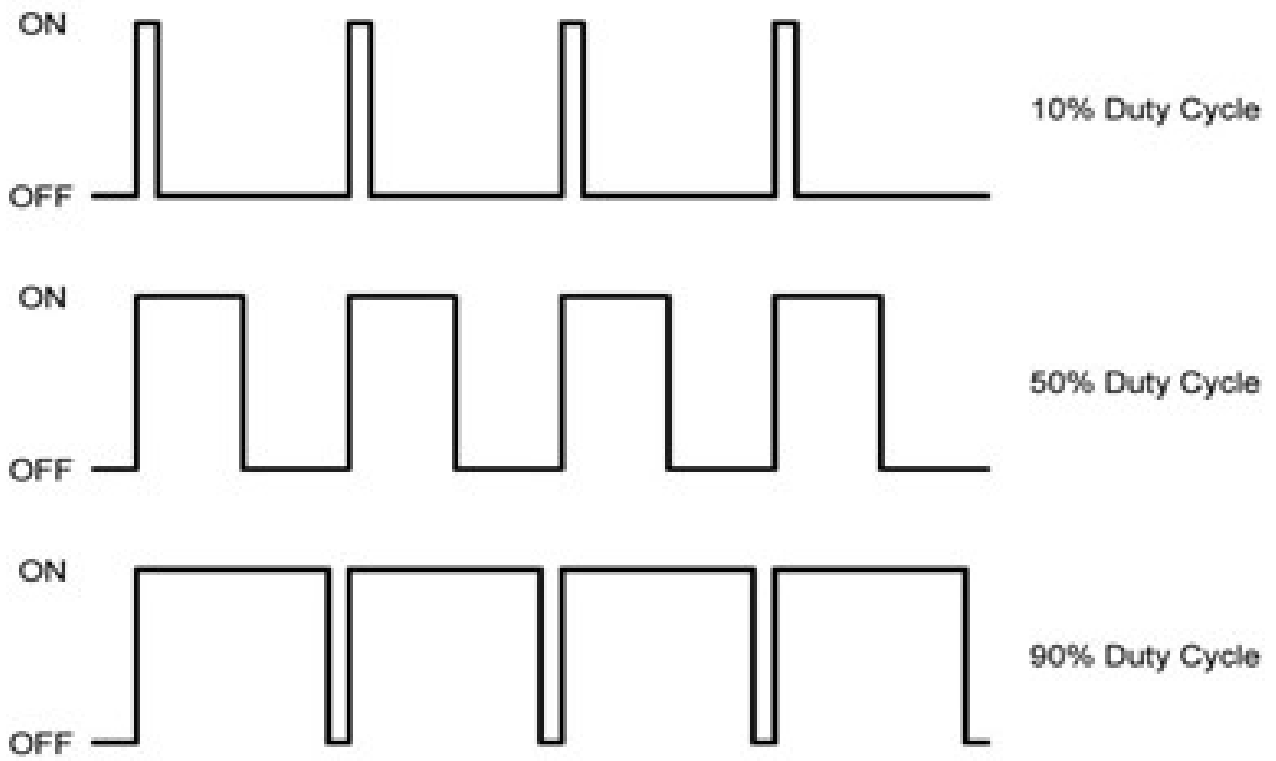
Exemplo debounce

- Rode o arquivo debounce.c

- Mantenha o botão pressionado por mais de 3 segundos

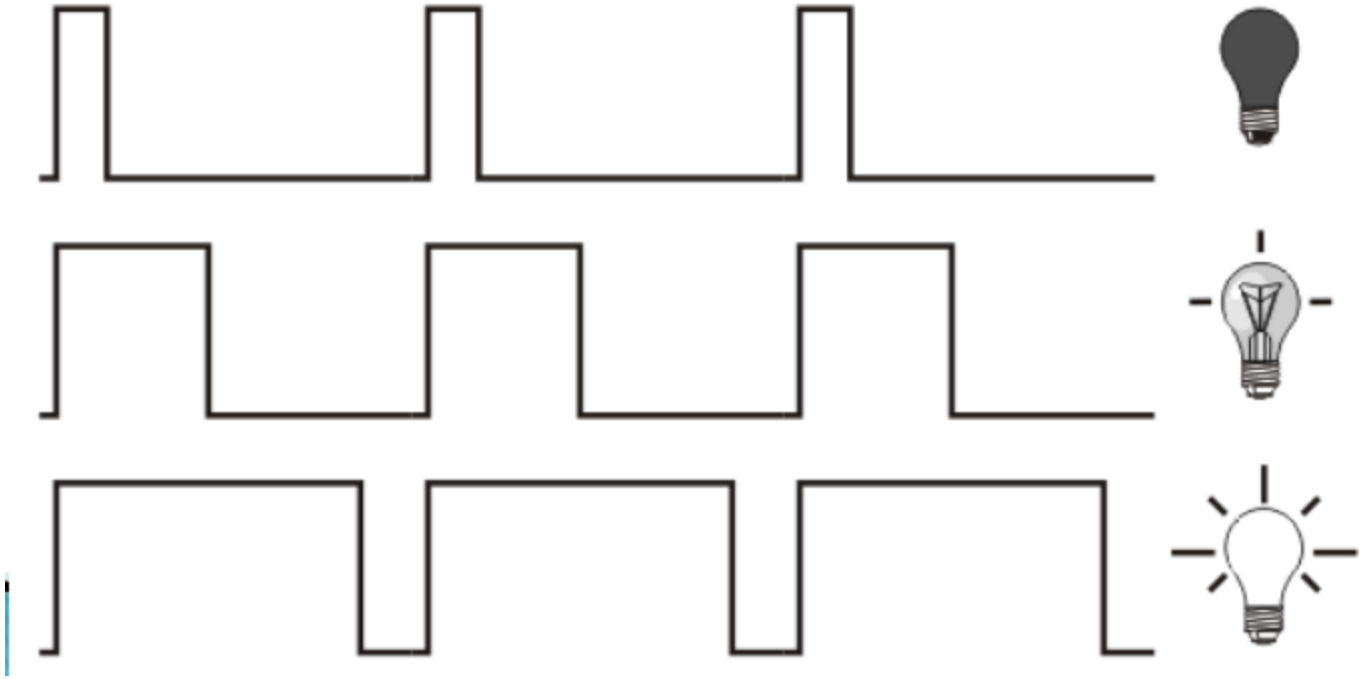
PWM

- Pulse Width Modulation ou Modulação por largura de pulso



https://www.mundodaeletrica.com.br/pwm-o-que-e-para-que-serve/

PWM com LED



https://wiki.ifsc.edu.br/mediawiki/index.php/AULA_6_-_Microcontroladores_-_T%C3%A9cnico

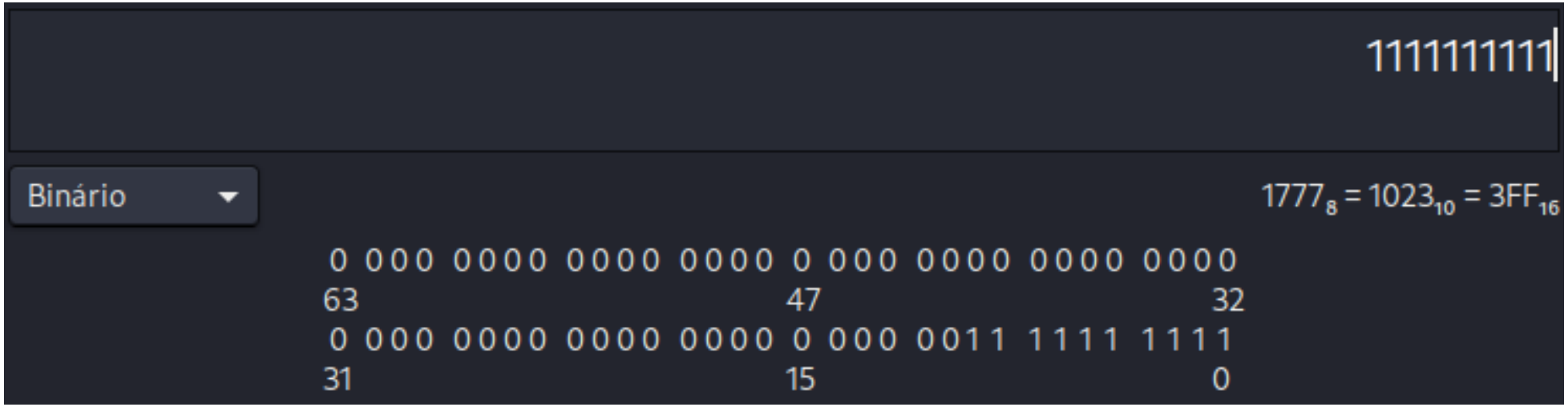
LEDC: PWM para LED no ESP32


```
ledc_timer_config_t timer = {
    .speed_mode = LEDC_LOW_SPEED_MODE,
    .duty_resolution = LEDC_TIMER_10_BIT, //10 bits-> 0 ~ 1023
    .timer_num = LEDC_TIMER_0,
    .freq_hz = 5000,
    .clk_cfg = LEDC_AUTO_CLK};

ledc_timer_config(&timer);
```

```
ledc_channel_config_t channel = {
    .gpio_num = PIN_LED,
    .speed_mode = LEDC_LOW_SPEED_MODE,
    .channel = LEDC_CHANNEL_0,
    .timer_sel = LEDC_TIMER_0,
    .duty = 0,
    .hpoint = 0}; //https://www.esp32.com/viewtopic.php?t=6532

ledc_channel_config(&channel);
```

https://inside.mines.edu/~coulston/courses/EENG383/lecture/lecture08.html

10 bits -> 0 ~ 1023



set_duty

- Estabelecer um valor de duty_cycle

```
ledc_set_duty_and_update(LEDC_LOW_SPEED_MODE,LEDC_CHANNEL_0,duty_cycle,0);
```

- "Varrer" por todos os valores de duty cycle (10 bits)

```
for (int i = 0; i < 1024; i++)
{
    ledc_set_duty_and_update(LEDC_LOW_SPEED_MODE,LEDC_CHANNEL_0,i,0);
    vTaskDelay(10 / portTICK_PERIOD_MS);
}
```

fade

- Variar duty cycle em um determinado período de tempo

```
ledc_fade_func_install(0);
```

```
ledc_set_fade_time_and_start(LEDC_LOW_SPEED_MODE,LEDC_CHANNEL_0,
                             0,2000,LEDC_FADE_WAIT_DONE);
```

valor duty cycle final

período de tempo (ms)

```
ledc_set_fade_time_and_start(LEDC_LOW_SPEED_MODE,LEDC_CHANNEL_0,
                             1023,4000,LEDC_FADE_WAIT_DONE);
```

Exemplo LEDC

- Rode o arquivo pwm.c

- Veja as possíveis maneiras de se variar o duty_cycle ao longo do tempo

Exercício

- Faça este exercício em dupla:
  - Um ESP32 será um servidor websocket
    - O servidor será o WiFi AP;
    - Deverá receber um comando de acionar o LED com PWM em formato json;
    - O LED deverá transicionar por cinco estados: apagado, 25%, 50%, 75% e 100% do brilho

Exercício (cont.)

- O outro ESP32 será um cliente websocket
  - O cliente deverá ser um WiFi STA;
  - Conectar ao WiFi do servidor (credenciais salvas em memória) e enviar um comando json websocket para acionar o LED de acordo com o pressionar de um botão;
  - O botão deverá ser monitorado quando pressionado e quando mantido pressionado:
    - Pressionado: o LED do servidor deverá transicionar pelos estados Apagado->25%->50%->75%->100%->Apagado...
    - Mantido pressionado: o LED do servidor deverá apagar

PADO

Labs