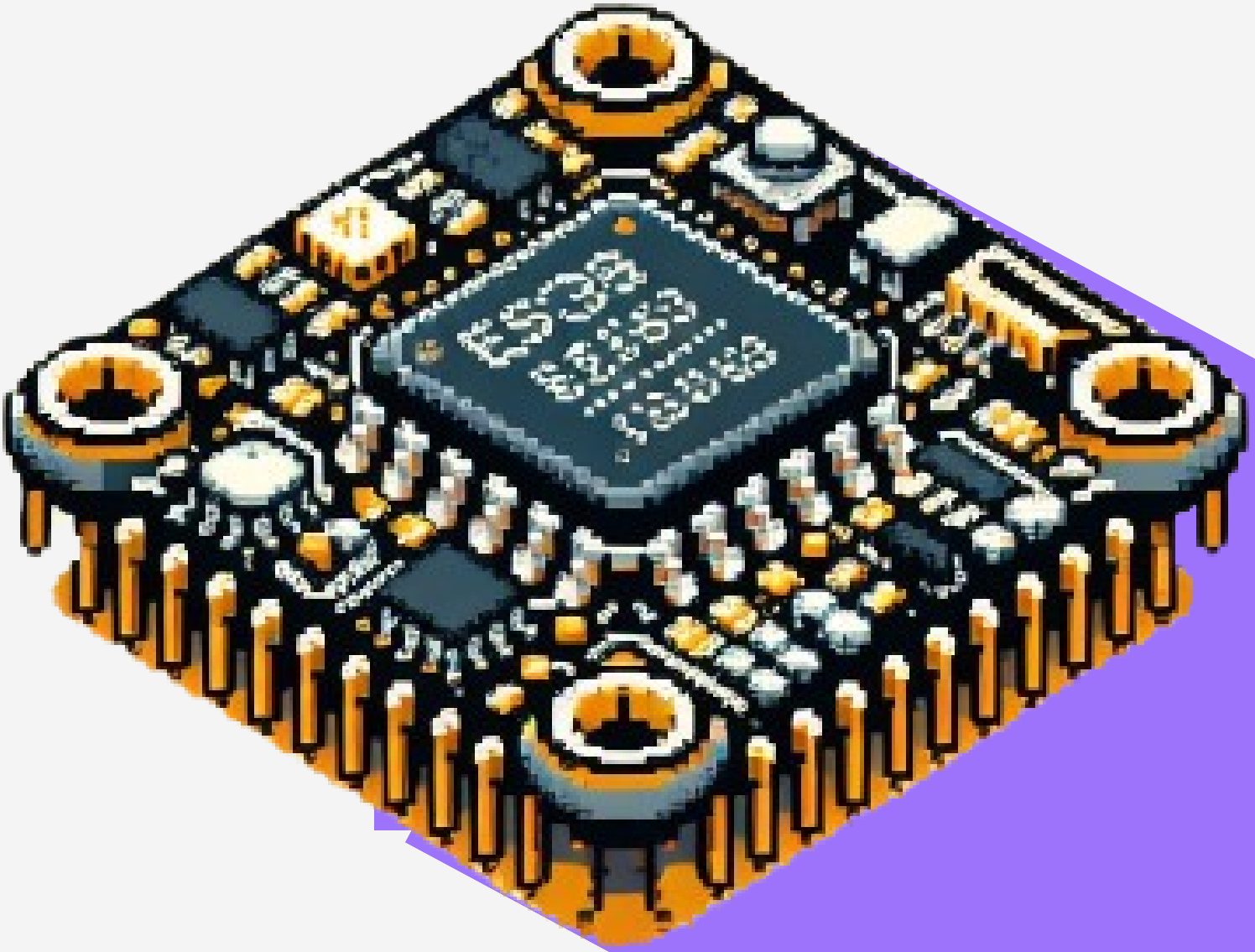
ESP32

GUIA PRÁTICO



SYNESTHESIA VISION

ILUMINANDO VIDAS

SYNESTHESIA VISION

ILUMINANDO VIDAS

Coordenação

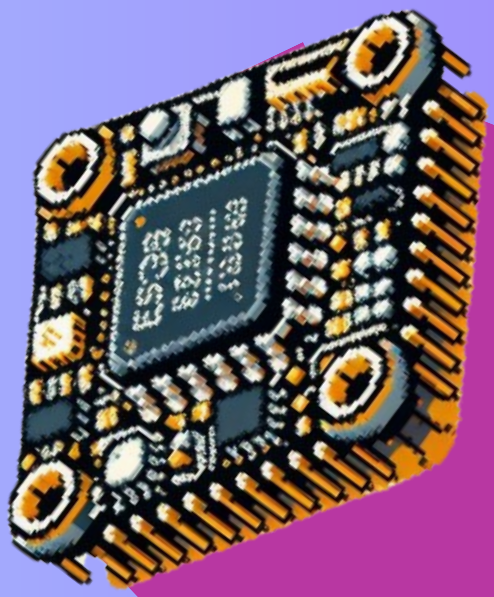
Prof. Dra. Aida Araújo Ferreira
Prof. Dr. Gilmar G. de Brito
Prof. Dra. Ioná Maria B. R. Barbosa
Prof. Dra. Vânia S. de Carvalho
Prof. Dr. Ronaldo R. B. de Aquino

Equipe

Carlos Alberto Leal de Barros
Estevão Pereira da Silva
Allamy Monteiro Pereira
Caroline Medeiros do Nascimento
Isaque Domingos Santana Silva
Felipe Santana de Oliveira
Jailson Pessoa de Souza Junior

ESP32
GUIA PRÁTICO

EDIÇÃO 1



SUMÁRIO

INTRODUÇÃO

Vídeo introdutório sobre a apostila clicando aqui!

Este trabalho tem por objetivo, orientar a todos que tem interesse em projetos com o SOC (System On a Chip) ESP32. Traremos diversas dicas que irão ajudar desde o início do desenvolvimento, inclusive, como fazer a escolha dos pinos mais adequados e evitar as possíveis armadilhas presentes nos Strapping pins. Também teremos capítulos dedicados a alguns dos recursos presentes no módulo, como timers, Interrupções, execução simultânea em dois núcleos, deep-sleep e função touch.
Esta apostila também conta com exercícios didáticos, a fim de que haja uma fixação mais relevante do conteúdo abordado, além disso, essas tarefas foram pensadas de forma que possam ser executadas tanto num ESP32 físico, quanto num simulador web-based chamado Wokwi disponível em: https://wokwi.com/. Com o propósito de facilitar os testes e o entendimento do conteúdo, iremos adotar a placa de de envolvimento ESP32 development board.

CAPÍTULO 1
ESCOLHENDO OS PINOS DO ESP32

CAPÍTULO 1: ESCOLHENDO PINOS DO ESP32

Por que escolher?

O ESP32 possui muitos pinos disponíveis (Fig.1) para os mais diversos usos. Diante disso, a fim de evitar problemas de funcionamento e operação nas aplicações, é necessário conhecer algumas funcionalidades desses pinos.

Neste capítulo, abordaremos as características e as indicações de utilização de pinos com portas analógicas, pinos conectados à memória Flash/SPI, pinos digitais apenas de entrada e pinos de Strapping.

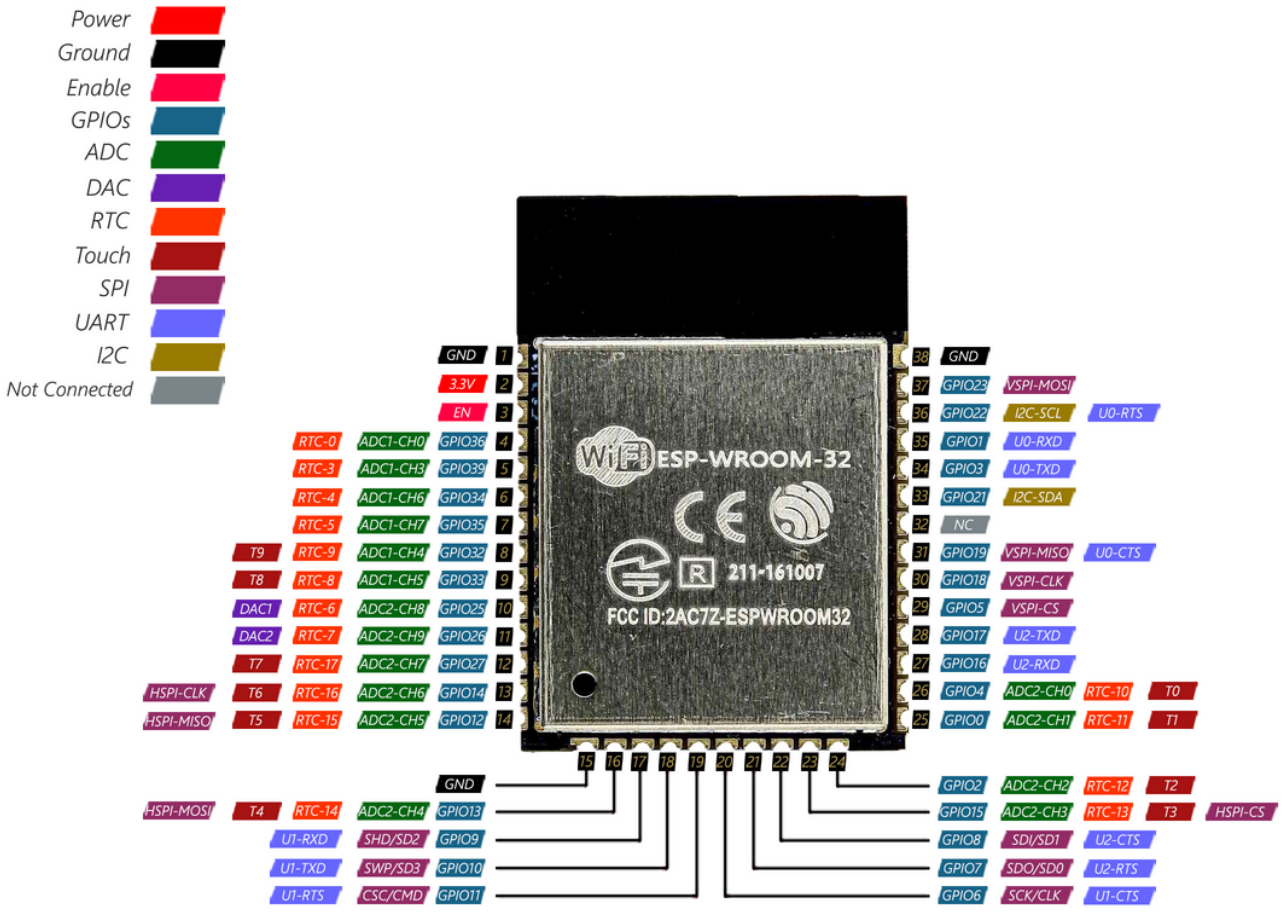


Fig. 1: Pinout do ESP-WROOM-32

CAPÍTULO 1: ESCOLHENDO PINOS

Pinos analógicos

O ESP32 possui 18 canais analógicos ADC Analog-to-Digital Converter) de 12 bits (Fig. 2), ou seja, pinos que podem converter sinais analógicos em digitais. Esses canais são utilizados na maioria das vezes para medir níveis de tensão de uma bateria ou sensor por exemplo. São eles:

- ADC1-CH0 (GPIO 36)
- ADC1-CH1 (GPIO 37)
- ADC1-CH2 (GPIO 38)
- ADC1-CH3 (GPIO 39)
- ADC1-CH4 (GPIO 32)
- ADC1-CH5 (GPIO 33)
- ADC1-CH6 (GPIO 34)
- ADC1-CH7 (GPIO 35)
- ADC2-CH0 (GPIO 4)
- ADC2-CH1 (GPIO 0)
- ADC2-CH2 (GPIO 2)
- ADC2-CH3 (GPIO 15)
- ADC2-CH4 (GPIO 13)
- ADC2-CH5 (GPIO 12)
- ADC2-CH6 (GPIO 14)
- ADC2-CH7 (GPIO 27)
- ADC2-CH8 (GPIO 25)
- ADC2-CH9 (GPIO 26)

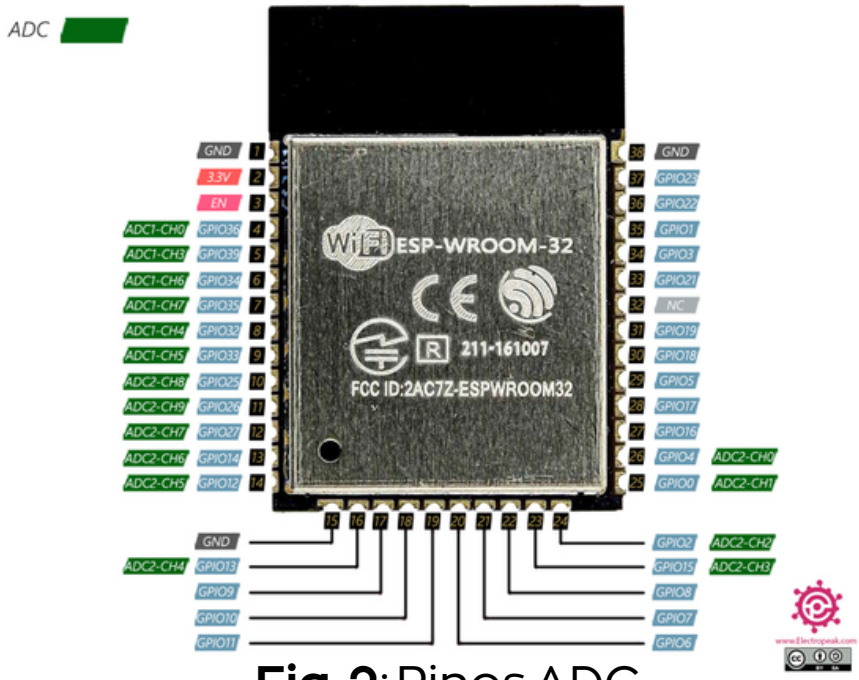


Fig. 2: Pinos ADC

CAPÍTULO 1: ESCOLHENDO PINOS

Pinos analógicos

Além dos pinos ADC, o ESP32 também dispõe de 2 canais DAC (Digital-to-Analog Converter) de 8 bits (Fig.3) que podem converter sinais digitais em analógicos digitais em analógicos, sendo úteis para gerar sinais e níveis de tensão. São eles:

- DAC1 (GPIO 25)
- DAC2 (GPIO 26)

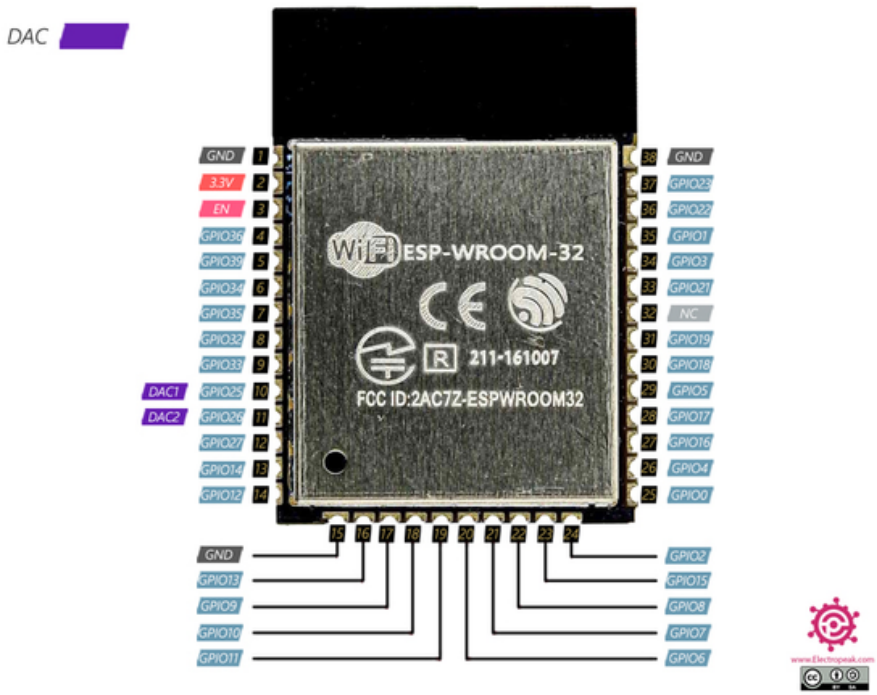


Fig. 3: Pinos DAC

CAPÍTULO 1: ESCOLHENDO PINOS

Pinos GPIO(Flash SPI)

Existem 6 pinos que não devem ser usados, pois são utilizados pelo chip para comunicação com a memória Flash/SPI (FIG.4) , ou seja, usar pinos poderá causar problemas no funcionamento :

- GPIO 6 (SCK/CLK)
- GPIO 7 (SDO/SD0)
- GPIO 8 (SDI/SD1)
- GPIO 9 (SHD/SD2)
- GPIO 10 (SWP/SD3)
- GPIO 11 (CSC/CMD)

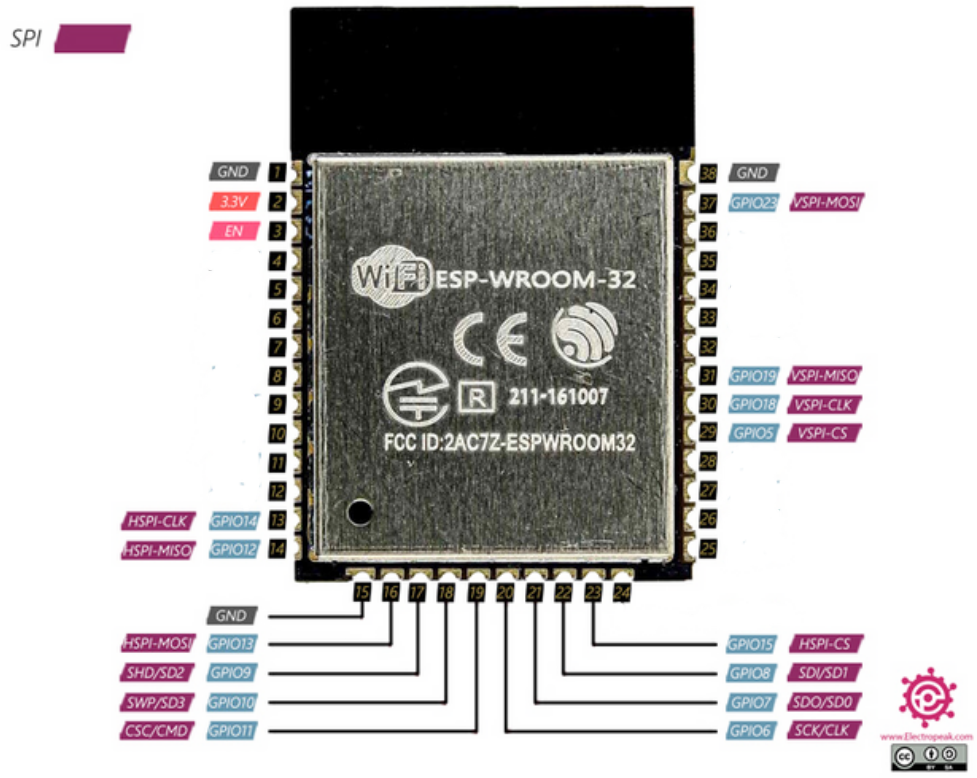


Fig. 4: Pinos de comunicação com a memória Flash/SPI

CAPÍTULO 1: ESCOLHENDO PINOS

Pinos GPI (apenas entrada)

O ESP32 possui 4 pinos que não podem ser utilizados como saídas, apenas entradas digitais (FIG.5). Além disso, não possuem resistores de pull-up ou pull-down. A função de PWM também não está disponível nesses pinos. São eles:

- GPI34;
- GPI35;
- GPI36;
- GPI39

Fig. 5: Pinos GPI (apenas entrada)

CAPÍTULO 1: ESCOLHENDO PINOS

Strapping pins

Os strapping pins definem algumas configurações do chip dependendo do nível lógico aplicado a eles durante a inicialização (FIG.6). São eles:

- GPIO0;
- GPIO2;
- GPIO5;
- GPIO12;
- GPIO15;

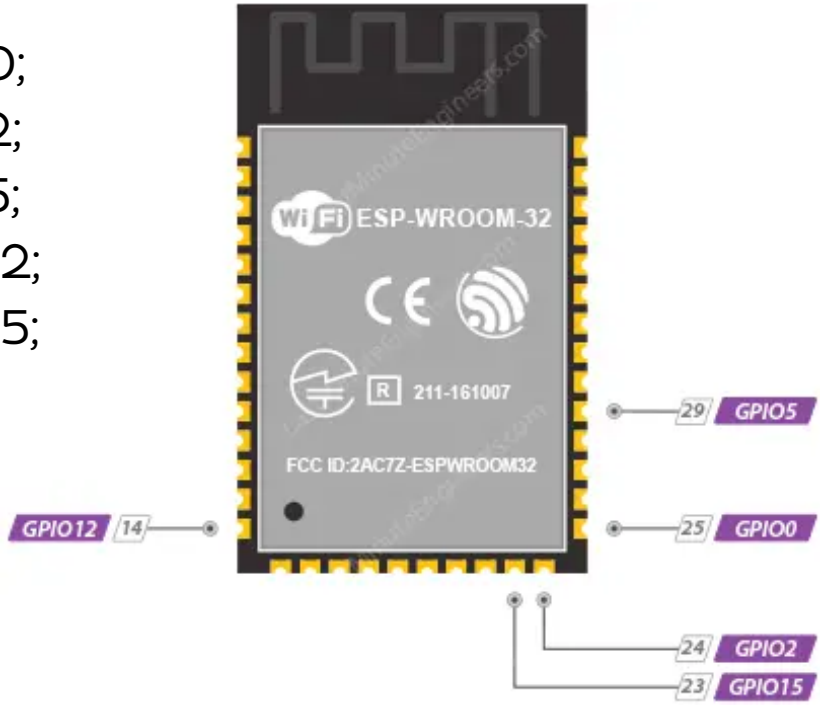


Fig. 6: Strapping pins

Comportamento dos Straping pins:

GPIO0 e GPIO2 (boot mode): Estes pinos definem o modo de boot. Para a operação normlal, basta que o nível GPIO0 esteja em nível alto. Para gravação do chip, é necessário aplicar nível baixo em ambos os pinos.

GPIO5(Timing of SDIO Slave): Controla configurações relativas a comunicação com SD cards junto com o GPIO15.

CAPÍTULO 1: ESCOLHENDO PINOS

Strapping pins

GPIO12(Voltage of Internal LDO): Controla a tensão de alimentação de alguns periféricos, se ele estiver em nível lógico alto, essa tensão será de 1,8V, em nível lógico baixo, essa tensão será de 3,3V. A tensão de 3,3 Volts é necessária para o funcionamento adequado na maioria das aplicações.

GPIO15(Debugging Log): Se esse pino estiver em nível lógico baixo, o chip não gerará log de depuração na inicialização.

Exercício 1:

Por que não é possível gravar o ESP32 com o circuito a seguir (Fig. 7)?

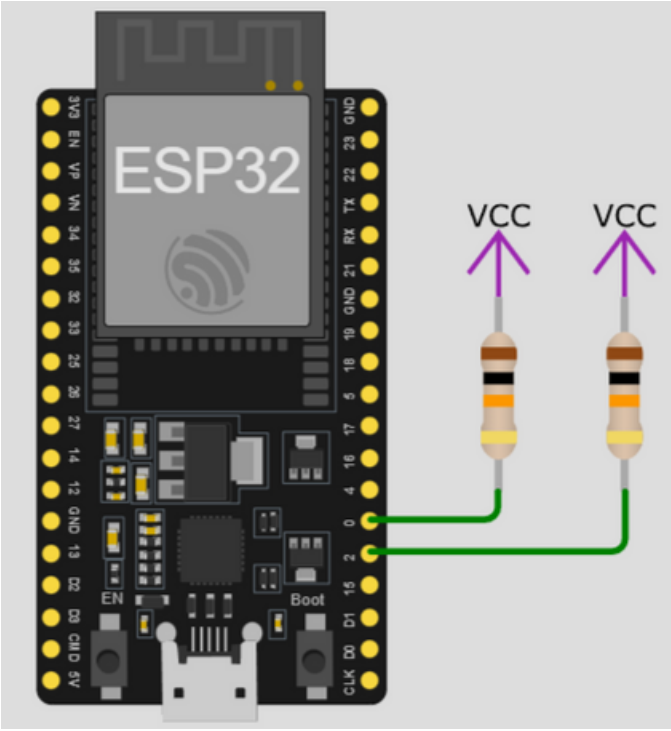


Fig. 7: GPIO0 e GPIO2 com resistores em 3,3 Volts

Resposta: Porque é necessário nível lógico baixo em ambos os pinos (GPIO0 e GPIO2) para entrar no modo de gravação.

CAPÍTULO 1: ESCOLHENDO PINOS

Exercício 2:

Qual será a principal consequência caso o GPIO12 esteja em nível lógico alto (Fig.8)?

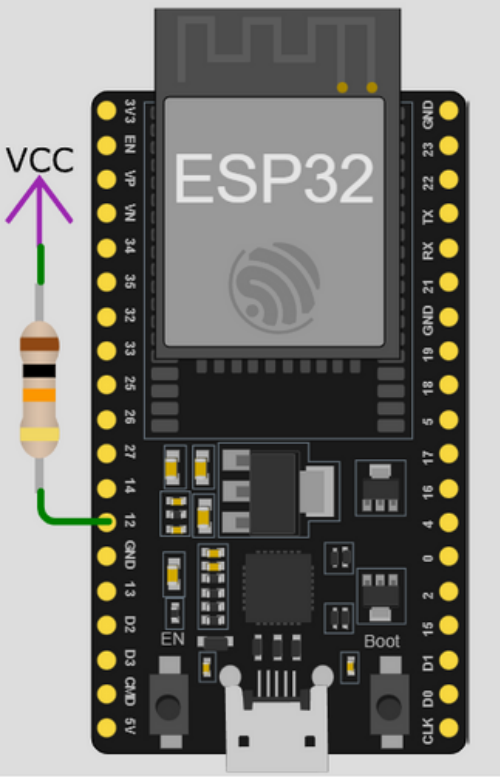


Fig. 8: GPI12 com resistor em 3,3V

Resposta: O GPIO12 controla a tensão do LDO interno e consequentemente da tensão externa do ESP32. Quando o nível nesse pino estiver em nível lógico baixo a tensão será de 3,3V, caso o nível seja alto, a tensão será de 1,8V, o que pode causar efeitos indesejados em alguns casos.

CAPÍTULO 2
INTERRUPÇÕES DO ESP32

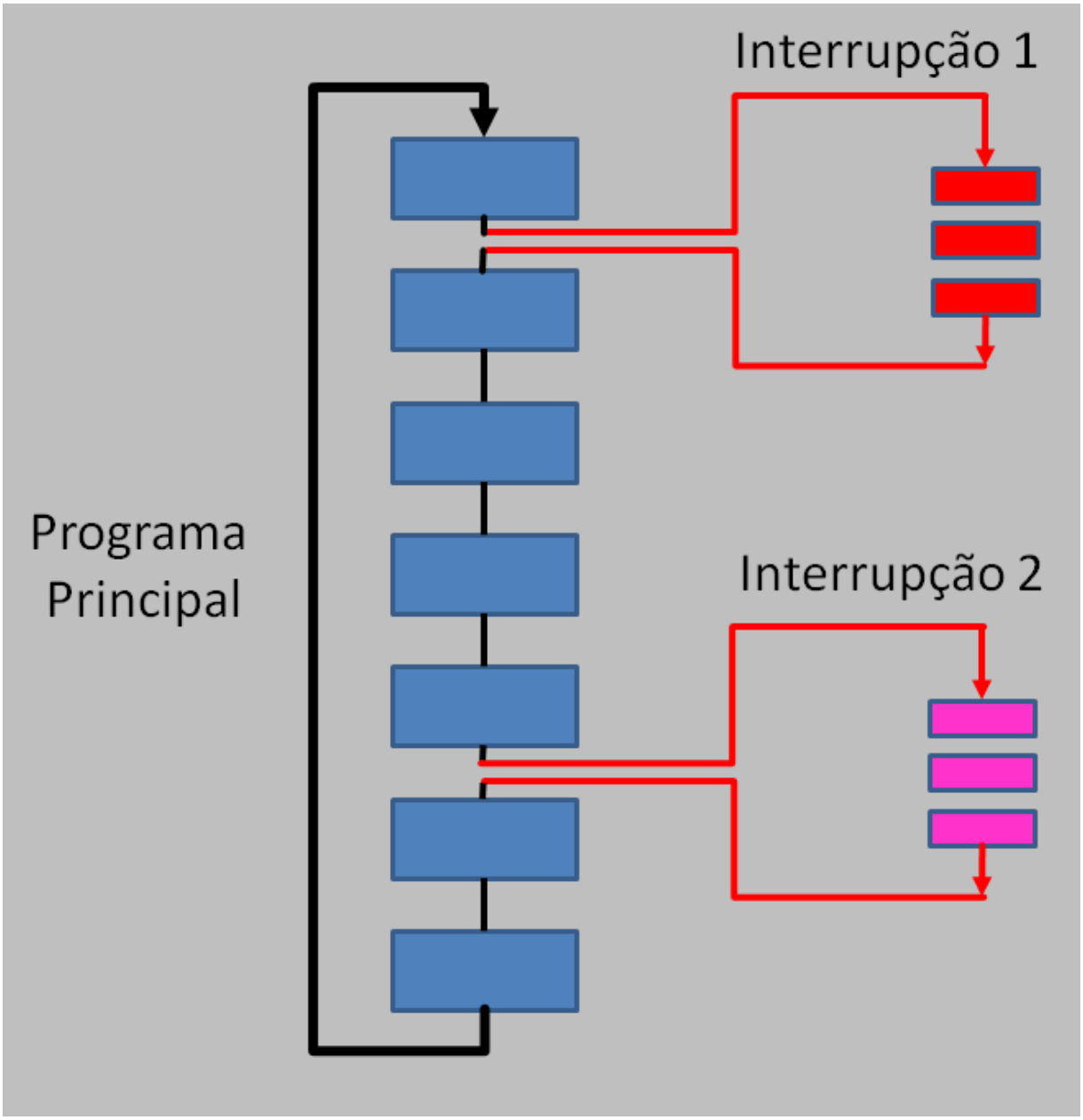


Fig. 9:Ilustração da lógica de interrupção

CAPÍTULO 2: INTERRUPÇÕES DO ESP32

O que são interrupções?

Os programas em um microcontrolador, seguem uma ordem de execução conforme a sequência de comandos escritos. Entretanto, existem aplicações que precisam que funções sejam executadas fora do fluxo básico do programa, quando alguma situação específica for encontrada.

No que diz respeito a isso, as interrupções são utilizadas para realizar esse papel. Este serviço é chamado após uma fonte de interrupção ser acionada. O microcontrolador parará a execução da tarefa atual e trabalhará em um trecho de código definido para essa interrupção e então voltará ao ponto em que parou (Fig. 9).

Fontes de interrupção

As interrupções precisam ser geradas através de uma fonte, o ESP32 possui diversas fontes para as mais diversas aplicações:

- **Interrupções externas:** os pinos do controlador podem ser configurados para gerar interrupções dependendo da transição ou do nível lógico neles.
- **Timers:** O ESP32 possui temporizadores que podem gerar interrupções em um período definido de tempo, esses serão explicados melhor em um capítulo posterior.

CAPÍTULO 2: INTERRUPÇÕES DO ESP32

Interrupções externas

Para usar interrupções externas no ESP32 é necessário utilizar uma função específica para isso, a seguir, temos um exemplo:

attachInterrupt(PINO, FUNÇÃO, MODO);

Os argumentos dessa função necessitam dos seguintes parâmetros:

- **PINO:** o pino que será monitorado;
- **FUNÇÃO:** a função que será chamada quando a interrupção for detectada;
- **MODO:** a condição para que a interrupção seja detectada:
  - **RISING:** borda de subida;
  - **FALLING:** borda de descida;
  - **LOW:** nível lógico baixo;
  - **HIGH:** nível lógico alto;
  - **CHANGE:** mudança no nível lógico;

Obs: A função chamada como parâmetro, deve ser declarada com a seguinte sintaxe:

```
void IRAM_ATTR nomeDaFunção()
{
  //código a ser executado
}
```

CAPÍTULO 2: INTERRUPÇÕES DO ESP32

Exemplo: LED aceso por interrupção

O seguinte código faz com que um LED seja aceso quando o usuário apertar um botão. Teste em um simulador (como o Wokwi) ou em um ESP32.

```
#define button 14  //pino do botão
#define LEDPIN 5  //pino do LED

void IRAM_ATTR changeLed()
//função de interrupção
{
  digitalWrite(LEDPIN, HIGH);
}

 void setup()
{
  pinMode(LEDPIN, OUTPUT);
  pinMode(button, INPUT);
  attachInterrupt(button, changeLed, FALLING);
  / *Definição da função do serviço de interrupção,
  onde o pino 14, quando em nível baixo, chama a
  função changeLed() */

  digitalWrite(LEDPIN, LOW);
}

void loop()
{
}
```

CAPÍTULO 2: INTERRUPÇÕES DO ESP32

Exercício

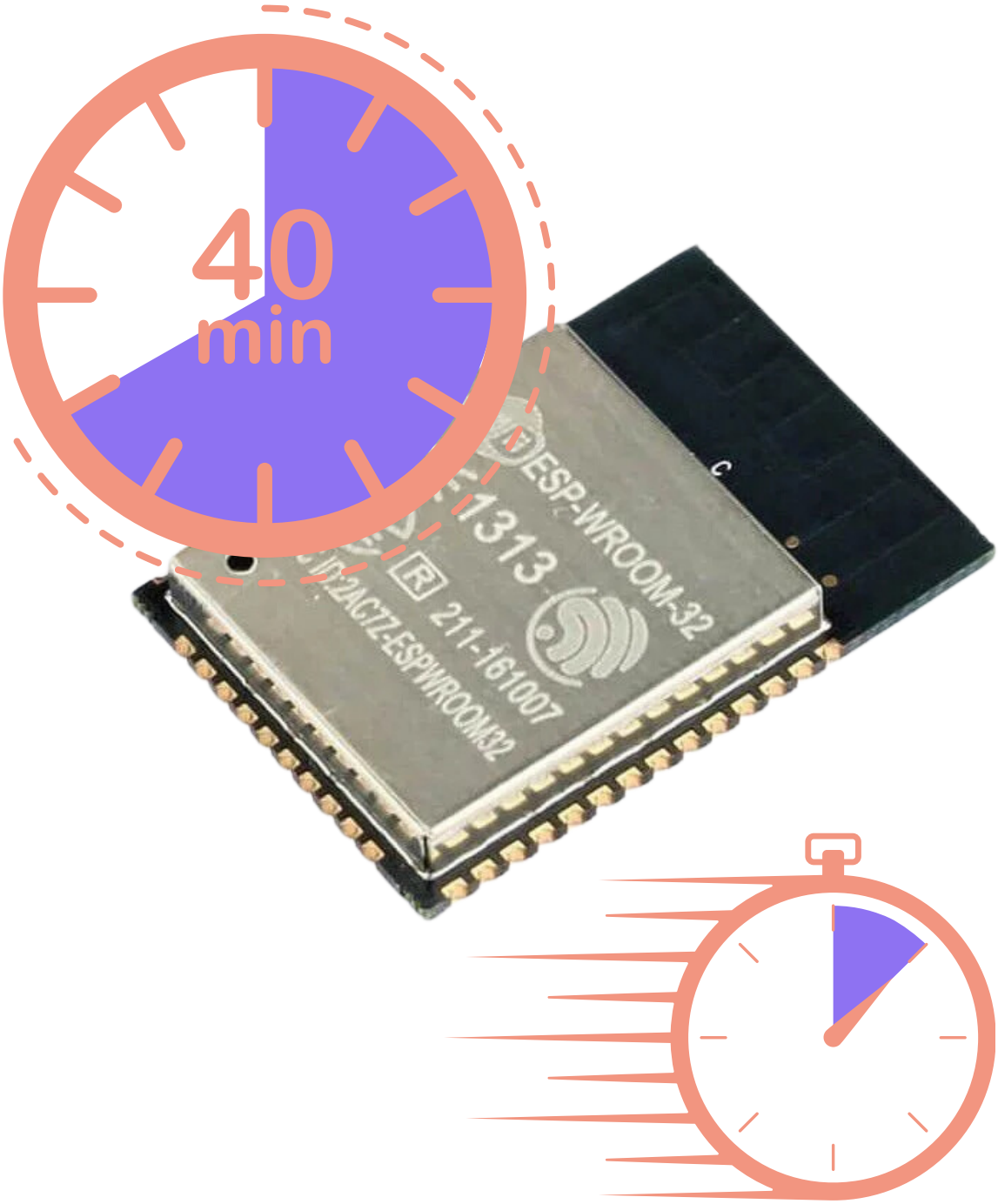
1)Usando uma interrupção externa, exiba uma mensagem no monitor serial quando um botão for pressionado.

2)Após isso, exiba a quantidade de vezes que o botão foi pressionado.

Para aprofundar: pesquise o que é “debounce” no contexto de interrupções e implemente em seu código. Registre a diferença entre o código com debounce e sem debounce.

Sugestão de solução disponível clicando aqui.

CAPÍTULO 3
TIMERS DO ESP32



CAPÍTULO 3: TIMERS DO ESP32

O que são timers?

Os timers, ou temporizadores, desempenham um papel fundamental ao programar eventos relacionados com o tempo de maneira precisa. São constituídos por cronômetros e podem estar ligados aos serviços de interrupções do MCU. Os temporizadores são essencialmente relógios simples que medem e controlam eventos temporais. Essa funcionalidade é aplicada em diversas situações, sendo controlada por registradores de função especiais no microcontrolador, como é o caso do ESP32. Os timers são ferramentas essenciais para controlar o tempo na execução de tarefas sem causar grandes bloqueios no fluxo do cógido como a função delay() geralmente o faz.

CAPÍTULO 3: TIMERS DO ESP32

Timers no ESP32

O ESP32 conta com 2 ou 4 temporizadores por hardware, dependendo do modelo, veja a Tabela 1. Todos eles são temporizadores genéricos de 64 bits (Fig. 10) (54 bits para ESP32-C3), baseados em pré-escalonadores de 16 bits, contadores que pode ser progressivos ou regressivos e que são capazes de ser recarregados automaticamente.

ESP32 SoC	Número de temporizadores
ESP32	4
ESP32-S2	4
ESP32-S3	4
ESP32-C3	2
ESP32-C6	2
ESP32-H2	2

Tabela 1: Quantidade de timers de cada modelo ESP32

Figura. 10: Timers ESP32

CAPÍTULO 3: TIMERS DO ESP32

Interrupções do temporizador

As interrupções por temporizador no ESP32 são um recurso fundamental para garantir eventos cronometrados precisos. Essas interrupções são acionadas pelo hardware do temporizador, permitindo a execução de tarefas em intervalos específicos, independentemente do código principal. A velocidade do temporizador, baseada no relógio do temporizador e no pré-escalonador, é crucial para determinar a precisão temporal. Configurando uma interrupção de temporizador e associando-a a uma ISR específica, o microcontrolador pode realizar tarefas não bloqueantes em intervalos definidos, interrompendo o loop principal e retornando a ele após a conclusão da ISR (Interrupt Service Routine). Essa abordagem é eficaz para operações como modulação por largura de pulso (PWM) , controle de LEDs e outros eventos assíncronos aleatórios que poderiam paralisar a execução de parte do código.

Os temporizadores no ESP32 são baseados em hardware, e o tempo está vinculado ao relógio do temporizador. A velocidade do temporizador (em Hertz) pode ser calculada pela fórmula:

$$\text{Velocidade do temporizador (Hz)} = \frac{\text{Frequência do relógio do temporizador (MHz)}}{\text{Pré-escalonador}}$$

Fórmula 1: Velocidade do timer

CAPÍTULO 3: TIMERS DO ESP32

Por exemplo, se o ESP32 estiver operando a uma frequência de clock de 80MHz, a velocidade do temporizador será de 80MHz (ou 8,000,000Hz) com um pré-escalonador definido como 1. Se o pré-escalonador for ajustado para 80, a velocidade do temporizador será de 1MHz (ou 1,000,000Hz).

O pré-escalonador é responsável por dividir a frequência acima para criar um "tick" do temporizador, que incrementa o contador do temporizador. A ISR é então configurada para disparar após um número específico de ticks.

Assim como as interrupções de hardware, as interrupções de temporizador no ESP32 são uma maneira eficiente de executar funções sem bloqueio em intervalos específicos. Para isso, é necessário configurar e associar uma interrupção de temporizador específica a uma Rotina de Serviço de Interrupção (ISR) durante a fase de configuração. Em uma aplicação comum, o microcontrolador executa continuamente o loop principal, sempre que uma interrupção de temporizador é gerada, o microcontrolador interrompe temporariamente o loop principal, executa a ISR e, após a conclusão, retoma o loop principal de onde foi pausado.

CAPÍTULO 3: TIMERS DO ESP32

Os temporizadores do ESP32 utilizam um contador que opera a uma determinada velocidade, dependendo da frequência do relógio e do valor do pré-escalonador. Essa contagem pode ser incremental ou decremental, pode ser parada e retomada e também pode ser auto recarregável, reiniciado quando atinge um valor predefinido, nesse momento pode haver o acionamento de uma interrupção. Os intervalos do temporizador podem ser ajustados durante a execução inclusive.

Exemplo de interrupção do temporizador do ESP32:

Neste exemplo, demonstraremos um programa onde ao pressionar um botão, um LED acenderá depois de um tempo, utilizando interrupções de temporizador no lugar da função delay(). Para implementar isso, realize as conexões conforme indicado no diagrama de circuito abaixo (Fig. 11):

Fig. 11: Led no GPIO14 e botão no GPIO4 e GND

CAPÍTULO 3: TIMERS DO ESP32

```
#define LED 12
#define BUTTON 2

//Instanciação do timer
hw_timer_t *My_timer = NULL;

//Função chamada pela interrupção
void IRAM_ATTR onTimer()
{
  digitalWrite(LED, HIGH); // Liga a LED
}

//Inicialização
void setup()
{
  pinMode(LED, OUTPUT);
  digitalWrite(LED, LOW);
  pinMode(BUTTON, INPUT_PULLUP);
  //Timer definido com clock dividido por 80 = 1MHz
  My_timer = timerBegin(0, 80, true);

  //Executa onTimer ao chegar no limite do tempo
  timerAttachInterrupt(My_timer, &onTimer, true);

  //Define o tempo de espera em 3 segundos
  timerAlarmWrite(My_timer, 3000000, true);
}
```

CAPÍTULO 3: TIMERS DO ESP32

```
//Loop com a função de monitorar um botão
void loop()
{
  if (!digitalRead(BUTTON))
  {
    timerAlarmEnable(My_timer);
   }
}
```

Como podemos ver no código acima, a única função que é executada continuamente no loop principal é a verificação do estado do botão. Caso o botão seja acionado, um temporizador começará a contar o tempo, até chegar em 3 segundos. Após isso, uma interrupção é detectada e uma função altera o estado do LED sem que utilizemos nenhuma função de delay();

Código completo clicando aqui.

Capítulo 3: Timers do ESP32

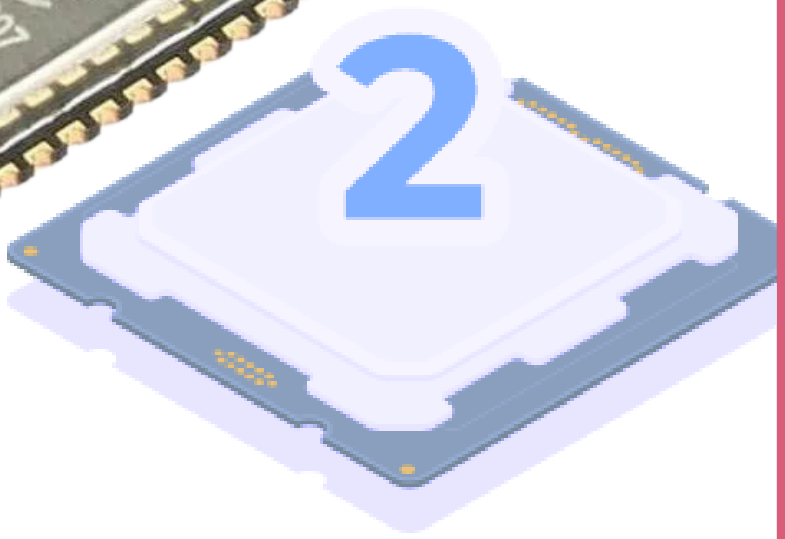
Exercício 1:

Usando os timers do ESP32, implemente um programa que conte o tempo e mostre no monitor serial..

Sugestão de solução disponível clicando aqui.

CAPÍTULO 4

EXECUÇÃO EM DUAL CORE



2.4 GHz

CAPÍTULO 4: EXECUÇÃO EM DUAL CORE

O que é uma execução em dual core?

A execução em dual core envolve um processador contendo dois núcleos de processamento em um único chip (Fig.12). Permitindo a realização de tarefas simultaneamente, oferecendo um aumento no poder de processamento sem depender exclusivamente do aumento da frequência do clock. Com essa abordagem, é possível otimizar o desempenho geral do sistema, executando mais de um código por vez.

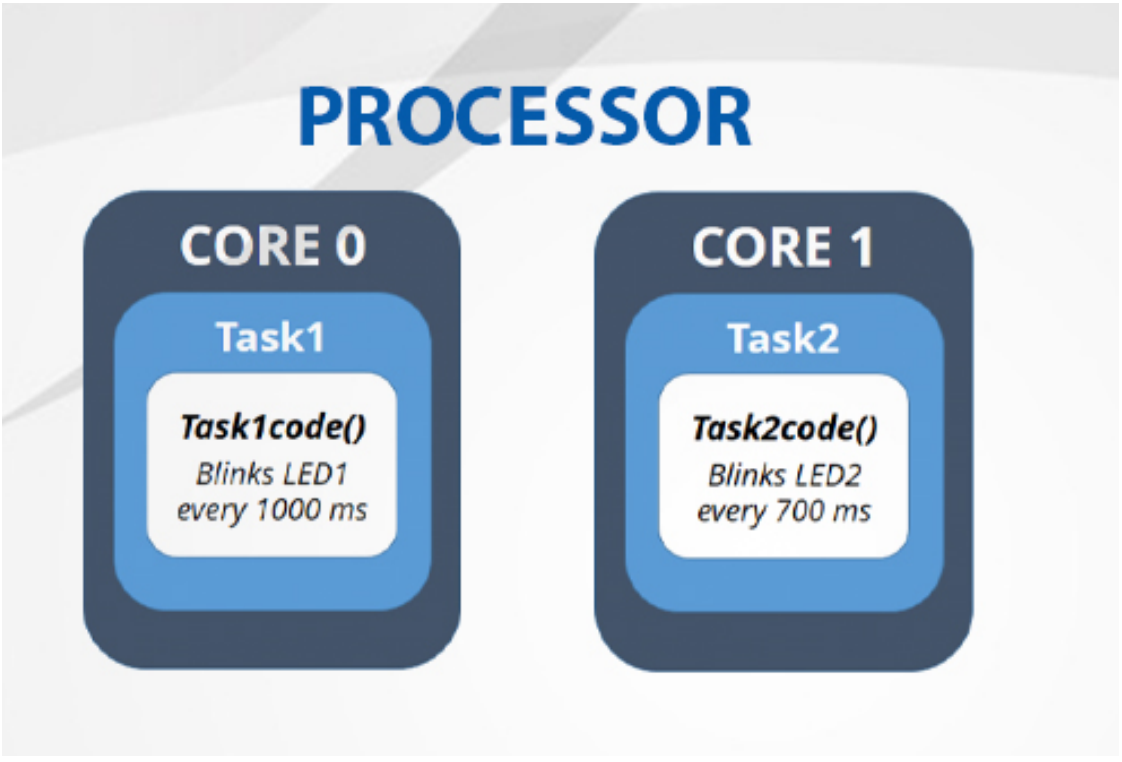


Fig. 12: Exemplo dos dois núcleos.

CAPÍTULO 4: EXECUÇÃO EM DUAL CORE

Dual core & ESP32

Como dito anteriormente, o ESP32 se destaca pela variedade de pinos, mas não só isso como também pela presença de dois núcleos de processamento (Fig.13).

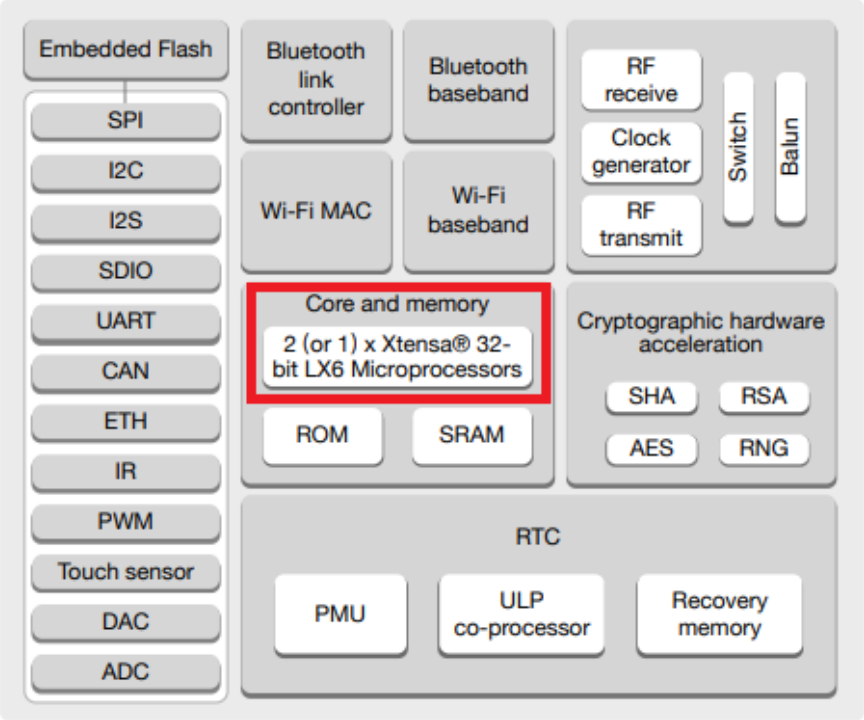


Fig. 13: ESP32 schematic (Random Nerd Tutorials)

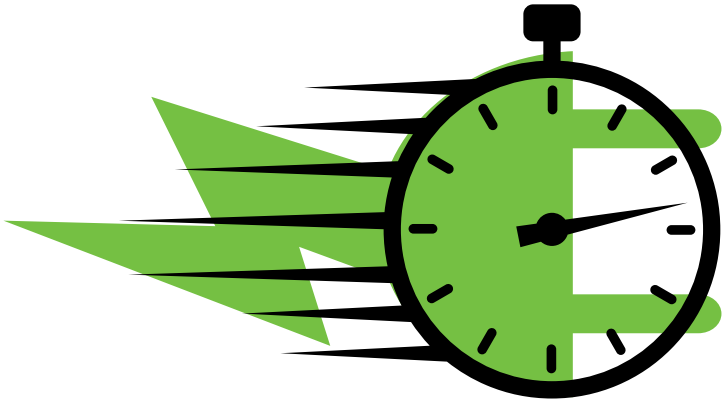
Com isso em mente, podemos listar algumas das diversas vantagens que acompanham essa tecnologia....

CAPÍTULO 4: EXECUÇÃO EM DUAL CORE

Vantagens

Algumas das vantagens adquiridas ao adicionar a funcionalidade de dual core aos nossos projetos, são:

- Melhor Desempenho com Multitarefa: Com capacidade de executar códigos em núcleos separados, conseguimos realizar tarefas simultaneamente.
- Eficiência Energética: A distribuição eficaz de tarefas entre os núcleos nos dá a capacidade de maior controle/economia de energia, resultando em maior vida útil da bateria.
- Maior rapidez na resposta do sistema como um todo.



Com o conhecimento em mãos, precisamos de um pouco de prática para fixar melhor.

CAPÍTULO 4: EXECUÇÃO EM DUAL CORE

Exercício 1

Teste a função de execução paralela em dois núcleos da ESP32 fazendo dois LEDs piscarem independentemente utilizando o diagrama da Fig.14.

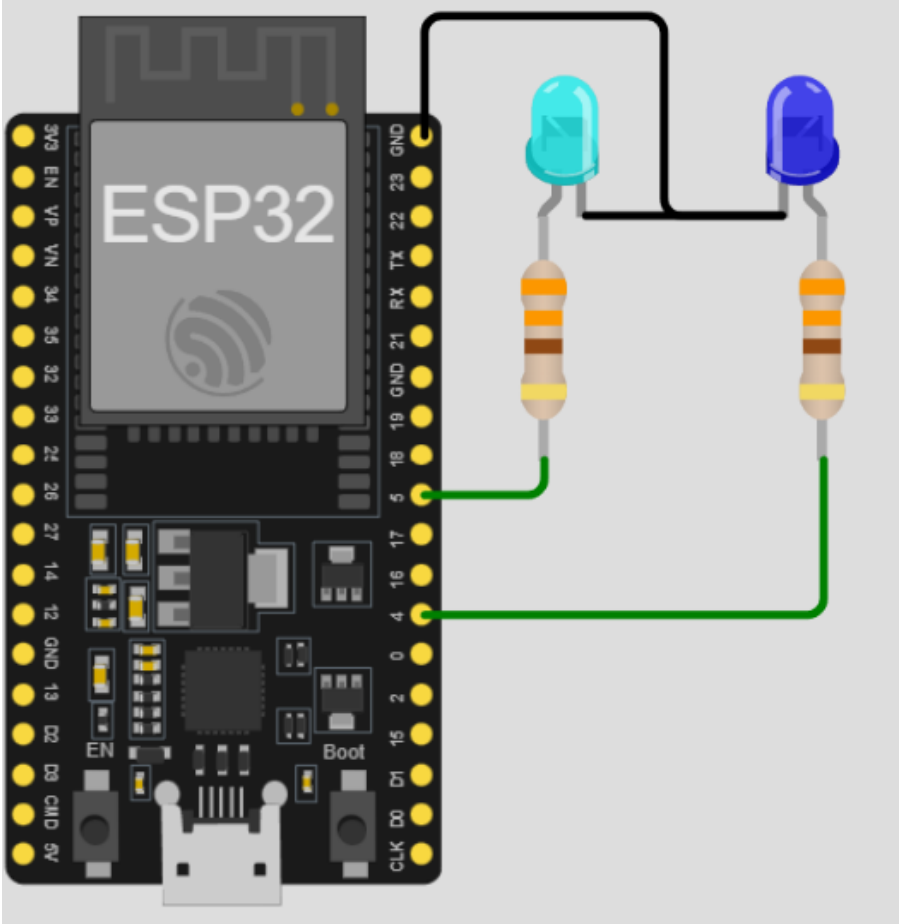
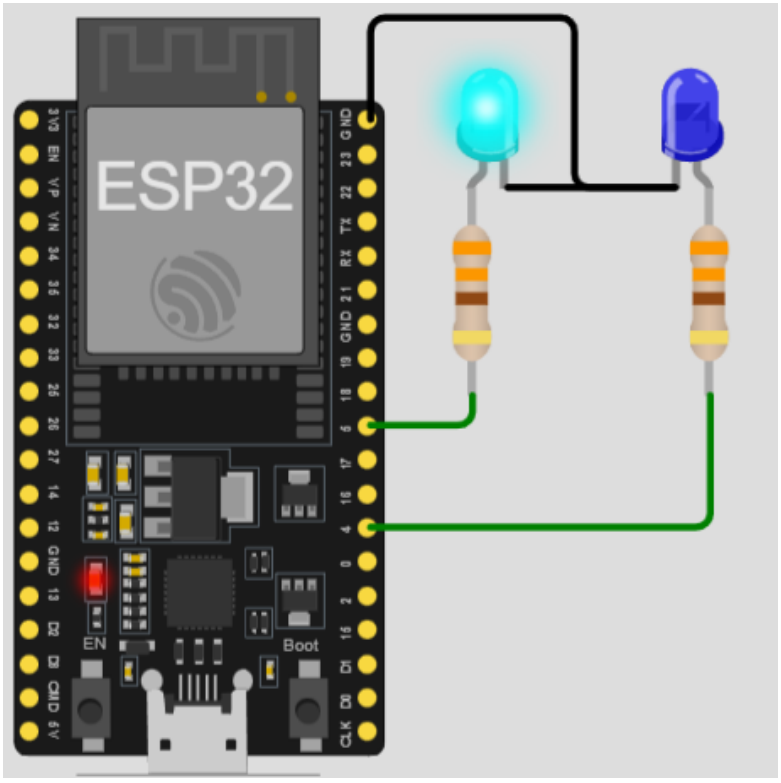
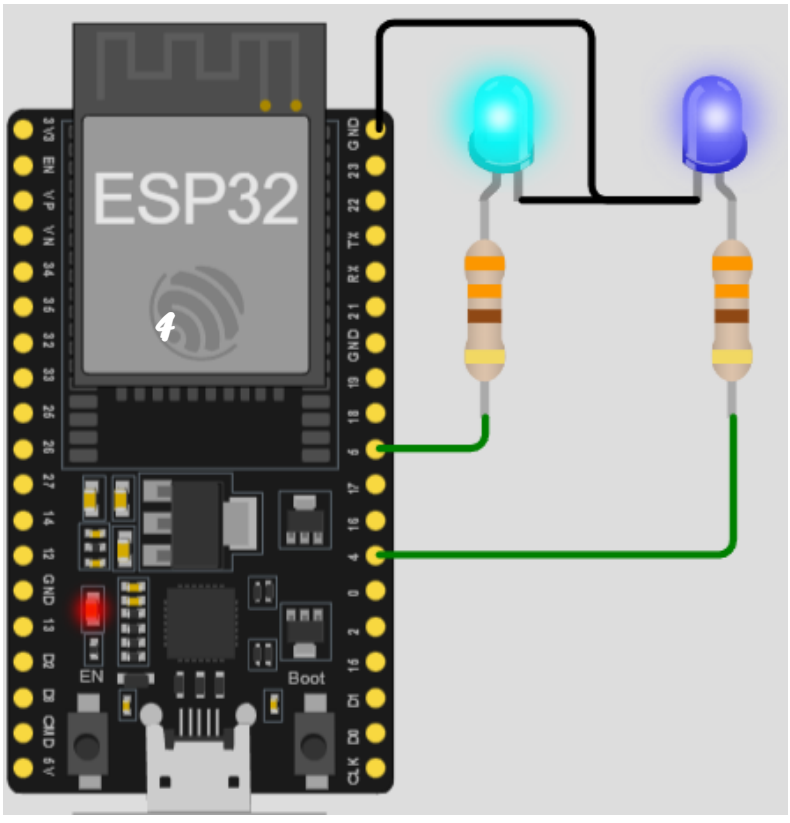


Fig. 14: Diagrama de exemplo para o exercício. LED's conectados nos pinos 4 e 5

O resultado esperado deve ser como o das imagens abaixo.

Sugestão de solução clicando aqui.





CAPÍTULO 5

SLEEP MODES

CAPÍTULO 5: SLEEP MODES

O que são sleep modes?

O microcontrolador ESP32 já mostrou-se poderoso e com diversas funções úteis, como seu Wi-Fi e Bluetooth. Entretanto, essas aplicações têm um alto gasto de energia, mesmo quando estão em repouso. Logo, para projetos que não necessitem desses ou de outros recursos, esses tornam-se apenas gasto desnecessário de bateria.

O ESP32 possui 5 modos de atuação diferentes, cada um com suas peculiaridades e com diferente disponibilidade de recursos. Assim, permitindo que a gestão de energia de um projeto seja realizada de forma eficiente.

Em síntese, o presente capítulo expõe quais são esses modos e seus impactos no consumo de energia do chip. Além disso, ensinará como acessá-los. Os sleep modes são os seguintes:

- **Active mode**
- **Modem sleep**
- **Light sleep**
- **Deep sleep**
- **Hibernation**

CAPÍTULO 5: SLEEP MODES

Active mode

Esse é o modo padrão de operação da ESP32. Ele conta com todas as funções ativas. Portanto, é o modo com maior consumo de energia e não é recomendado para aplicações mais simples e que não usam os componentes de comunicação sem fio do chip, visto que possuem alto gasto de potência.
Segundo o datasheet do chip, esse modo consome entre 80 a 260 mA, sendo assim, o menos eficiente disponível.

Modem sleep

- No modo modem sleep, a comunicação sem fio, WiFi/Bluetooth é desativada quando não requisitada.
- Enquanto o modem está neste modo, a CPU e outras partes do dispositivo podem continuar operando normalmente.
- A comunicação pode ser ativada novamente quando há a necessidade de enviar ou receber dados, o que geralmente é acionado por um temporizador ou por eventos específicos de transmissão/recepção.
- É uma estratégia eficaz para otimizar o consumo de energia em dispositivos que precisam estar prontos para se comunicar, mas não estão constantemente transmitindo ou recebendo dados. Esse modo consome de 3 a 20 mA.

CAPÍTULO 5: SLEEP MODES

Light sleep

- Neste modo, a CPU é desligada, assim como o componente de comunicação sem fio, mas a memória RAM permanece ativa.
- A CPU pode ser acordada por interrupções externas, como GPIO, timer ou UART.
- É adequado para situações em que a CPU precisa ser despertada rapidamente em resposta a eventos específicos. Esse modo consome em média 0.8 mA.

Deep sleep

- Este é o modo mais eficiente em termos de economia de energia, consumindo de 10 a 150uA.
- A CPU, bem como a RAM, são desligadas.
- O chip pode ser acordado por timers, eventos externos, pelo ULP Coprocessor e pelos pinos touch.
- O consumo de energia é extremamente baixo, adequado para dispositivos que precisam operar com baterias por longos períodos de tempo.

CAPÍTULO 5: SLEEP MODES

Hibernation mode

- Este é um modo de economia de energia mais profundo do que o Deep Sleep.
- Além de desligar a CPU e a RAM, a maior parte do RTC também é desligado.
- Apenas um timer RTC e alguns GPIO RTC são mantidos para ativar o chip.
- Não é possível armazenar nenhum dado nesse modo.
- O consumo de energia é mínimo, aproximadamente 2,5uA, tornando-o adequado para aplicações que precisam de baixo consumo.

Como acessar os modos

- Active
  - Modo padrão do chip
- Modem sleep
  - Funções para acessar modem sleep
- Light sleep
  - Como acessar light sleep
- Deep sleep
  - Sono profundo da ESP32
- Hibernation
  - É acessado pelo deep sleep, porém sem guardar nada no RTC.(O exemplo fornecido para o deep sleep, também serve para o hibernation)

CAPÍTULO 6
FUNÇÃO TOUCH

CAPÍTULO 6: FUNÇÃO TOUCH

O que são os pinos touch?

O microcontrolador ESP32 possui suporte nativo a botões de toque capacitivos . Esses pinos funcionam detectando a presença de um objeto condutor, como um dedo humano. Portanto, a capacitância do pino varia, possibilitando a detecção através do código.

Quais são os pinos touch?

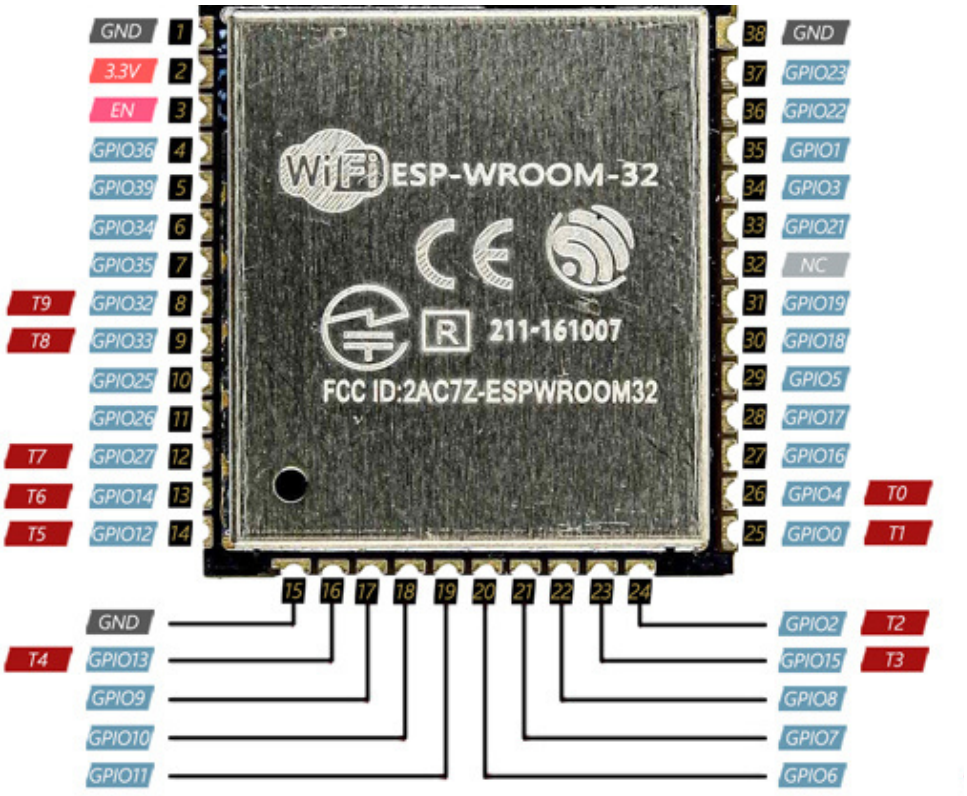


Fig. 15: Pinos touch do ESP32, em vermelho.

- T0(GPIO 4)
- T1(GPIO 0)
- T2(GPIO 2)
- T3(GPIO15)
- T4(GPIO13)
- T5(GPIO12)
- T6(GPIO14)
- T7(GPIO27)
- T8(GPIO33)
- T9(GPIO32)

CAPÍTULO 6: FUNÇÃO TOUCH

Como utilizar

A informação desses pinos pode ser acessada por meio da função touchRead(), que funciona da seguinte forma:

touchRead(GPIO Touch);

Atenção: O argumento deve ser o número do pino e não do touch, como por exemplo: touchRead(15); acessa o T3, se usar touchRead(3); um erro será retornado.

Essa função retorna o valor capacitivo lido pela porta.

Exercício

Utilizando algum pino touch da ESP32, mostre o valor lido por ele no monitor serial e faça com que um LED acenda quando detectar o toque de seu dedo.

Sugestão de solução clicando aqui.

Observação: Para aqueles que estão utilizando o simulador Wokwi para fazer os exercícios, ele ainda não possui suporte para a função touch. Logo, essa tarefa deve ser feita com um ESP32 development board.

CONCLUSÃO

Ao longo desta apostila, exploramos diversos recursos fundamentais oferecidos pelo microcontrolador ESP32, com foco na escolha dos pinos, interrupções, timers, execução em dual core, modos de economia de energia e função touch capacitiva. Desde a compreensão básica dos pinos e suas funcionalidades até a exploração do potencial de execução paralela em dual core, cada tópico foi cuidadosamente elaborado para proporcionar uma compreensão abrangente e prática.

Em conjunto, esses recursos formam uma poderosa ferramentas para o desenvolvimento de uma ampla variedade de aplicações embarcadas. Seja no domínio da automação residencial, IoT, dispositivos vestíveis ou projetos industriais, o ESP32 oferece uma plataforma flexível e poderosa para transformar ideias em realidade.

Esperamos que esta apostila tenha fornecido uma base sólida para explorar os recursos do microcontrolador ESP32, inspirando você a criar projetos inovadores e impactantes. Continuamos à disposição para auxiliá-lo em sua jornada de aprendizado e desenvolvimento.

REFERÊNCIAS

CAMERON, N. ESP32 Formats and Communication. Berkeley, CA: Apress, 2023.
646 p. ISBN 978-1-4842-9378-2.

Embarcados. Wokwi: Simulador de ESP32. Disponível em: <https://embarcados.com.br/wokwi-simulador-de-esp32/ >. Publicado em: 13 abr. 2022. Acesso em: 28 fev. 2024.

Espressif Systems. ESP32-WROOM-32 Datasheet. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf >. Acesso em: 27 de Fev. 2024

Wokwi. (2022). Simulação de Arduino Online. Wokwi. <https://wokwi.com/ >.

SYNESTHESIA<https://synesthesiavision.com/>. Acesso em : 12 de mar. 2024.