Anotações ESP32

→ Possui vários protocolos de comunicação serial disponíveis (SPI, I2S, I2C e UART), o que possibilita que se comunique com diversos tipos de componentes

→ O modelo ESP32 WROOM-32 usa 6 pinos exclusivamente para se comunicar com a memória flash externa, que **não** podem ser usados para outro propósito (GPIO 6-11)

→ A versão do ESP32 que temos é um DevKit. A versão "padrão" é apenas o chip quadradinho que fica na placa que a gente tem. A placa serve para poder encaixar o chip na protoboard.

→ Os pinos no datasheet se referem aos pinos do chip sozinho. Para encontrar a pinagem do DevKit, é preciso achar um modelo específico

→ O ESP32 possui 2 botões na placa: EN para reset e BOOT (ver melhor esse).

→ CP2102 é um pequeno chip (tipo um CI msm) que realiza a conversão de sinais USB para UART (UART é um protocolo de comunicação serial (ver tópico 1 novamente)). Isso serve para podermos passar do PC pro micro via comunicação. Para usar essa funcionalidade, é necessário um driver. Se achar algo parecido com driver de CP2102, é isso.

→ Alguns pinos do tipo GPIO (general input output) não estão nem visíveis na placa. Podem estar sendo utilizados internamente pelo chip ou só não foram disponibilizados, por outros motivos, para o usuário (Ex: GPIO20 não aparece no diagrama de pinout).

→ O pino EN (canto superior esquerdo) serve para o reset. Sim, ele já está conectado automaticamente com o botão EN que vem com a placa. Pino em 1: placa ligada, pino em 0: desligada. O botão corta a corrente, logo, o botão reseta o micro.

→ Algumas nomenclaturas no esquema de pinout do DevKit que mandei no zape:

→ ADC: Conversão analógico digital

→ COMM. INTERFACE: Pinos que servem para fazer comunicação serial

→ DAC: Contrário de ADC, digital para analógico.

→ HS: Alguma coisa a ver com modo de alta velocidade. Por exemplo, alguns pinos tem comunicaçã em alta velocidade

→ TOUCH: Não entendi muito bem como funciona mas servem para implementar equipamentos sensíveis ao toque.

→ Para se comunicar com o ESP32 via UART (protocolo de comunicação serial), usamos os pinos RX e TX:

→ RX é o pino reception. Por meio desse pino, o micro **recebe** dados

→ TX é o pino transmission. Por meio dele, o micro **envia** dados

→ Por meio desses pinos, podemos carregar o nosso código no micro por comunicação serial, utilizando a conversão USB – SERIAL citada anteriormente.

→ Baud rate é a velocidade de envio de dados da comunicação serial em bits/s. O padrão do ESP32 é 115200b/s.

→ APIs no contexto do ESP32 são as bibliotecas de funções prontas para cada tipo de funcionalidade. Por exemplo, a API da utilização de wi-fi são as funções prontas que podem ser utilizadas para isso. Sem as APIS (vulgo bibliotecas), precisaríamos criar os códigos de baixo nível do zero.

→ As APIs são próprias do microcontrolador, desenvolvidas pelos fabricantes do chip. As IDEs precisam ser compatíveis com a utilização dessas APIs, ou seja, devem ser capazes de transformar os textos que escrevemos na informação binária correta. A ESP-IDF (original da espressif) possui as APIs “padrão”, sem modificação. A Arduino IDE pega essas APIs e cria, com as próprias APIs, outras funções ainda mais simplificadas para ajudar o programador. As APIs, porém, não deixam de ser do próprio hardware.

→ Existem dois canais diferentes de leitura analógica para digital (ADC). ADC1 e ADC2.

→ Em canais ADC o funcionamento é o seguinte: Se existem dois dispositivos conectados ao mesmo tempo, a conversão **não** é feita de forma simultânea. Ele lê sequencialmente (primeiro um pouquinho do primeiro, e depois um pouquinho do segundo), mas essa alternância é feita de forma tão rápida que na prática é como se fosse simultâneo. Esse processo se chama **multiplexação**

→ Os módulos Wifi e ADC2 não podem ser utilizados simultaneamente. Pelo que entendi, utilizam os mesmos dispositivos internos, então enquanto um usa, o outro fica indisponível. O ADC1 é independente do módulo Wifi. ADC1 possui 8 pinos e ADC2 possui 10 pinos.

→ A resolução dos ADC é de 12 bits (0 – 4095).

→ O ESP32 possui uma técnica/sistema chamado sinais PWM. A serventia desse sistema consiste em simular sinais analógicos por meio de um sinal digital. Funciona da seguinte forma:

> → O sinal PWM é digital, e ao invés de passar uma tensão maior ou menor para controlar a potência do sinal (como é feito em sinais analógicos) os sinais PWM só possuem nível alto ou baixo.
>
> → Para simular as variações, ele liga e desliga rapidamente. Dependendo de quanto tempo ele fica ligado ou desligado em cada ciclo, a potência final no dispositivo receptor vai ser diferente. Se por exemplo, em cada ciclo ele fica ligado 75% do tempo, o dispositivo vai receber 75% da potência.