Conversão AD, Conversão DA e PWM no ESP32 com MicroPython

1. Conversão Analógico-Digital (AD)

A conversão AD permite que o ESP32 transforme sinais analógicos em valores digitais, possibilitando a leitura de sensores. O ESP32 possui ADCs de 12 bits, resultando em valores entre **0 e 4095**.

Atenuação no ADC do ESP32

O ADC do ESP32 pode medir a entrada em diferentes faixas, dependendo da **atenuação** definida. A atenuação define o intervalo de tensão que pode ser convertido pelo ADC. O ESP32 possui **4 níveis de atenuação** , permitindo a compreensão em diferentes faixas:

Atenuação	Faixa de Tensão Máxima
`ADC.ATTN_0DB`	0 V - 1,1 V
`ADC.ATTN_2_5DB`	0 V - 1,5 V
`ADC.ATTN_6DB`	0 V - 2,2 V
`ADC.ATTN_11DB`	0 V - 3,3 V

Por padrão, o ESP32 só mede até 1,1V . Se você precisar medir mais, é necessário aumentar a atenuação.

Exemplo: Leitura de Temperatura com LM35

O **LM35** fornece uma saída analógica proporcional à temperatura em graus Celsius (10mV/°C). Como o ESP32 tem uma tensão de referência de **3.3V** , podemos calcular a temperatura usando as seguintes soluções:

$$Temperatura\ (^\circ C) = \frac{V_{ADC} \times 3300}{4095 \times 10}$$

Código - Leitura do LM35

```
from machine import ADC, Pin
import time

# Configuração do pino ADC (GP34 - ADC6 no ESP32)
sensor_temp = ADC(Pin(34))
sensor_temp.atten(ADC.ATTN_11DB)  # Permite leitura até 3.3V

while True:
    valor_adc = sensor_temp.read()  # Lê o valor digital (0 a 4095)
    tensao = (valor_adc / 4095) * 3.3  # Converte para tensão (V)
    temperatura = (tensao * 100)  # Conversão para °C
    print(f"Temperatura: {temperatura:.2f} °C")
    time.sleep(1)
```

Explicação:

- O pino **34** está configurado como entrada analógica.
- **atten(ADC.ATTN_11DB)** ajusta a escala para até 3,3V.
- O valor lido pelo ADC é convertido para tensão e depois para temperatura.

2. Conversão Digital-Analógica (DA)

A conversão DA permite gerar sinais analógicos a partir de valores digitais. O ESP32 possui dois canais DAC (GPIO25 e GPIO26) , com resolução de 8 bits (0 a 255).

Exemplo: Gerando uma tensão analógica

O código abaixo gera uma tensão proporcional ao valor inserido pelo usuário (0-3,3V).

```
from machine import DAC, Pin
import time

# Configuração do DAC no pino 25
dac = DAC(Pin(25))

while True:
    valor = int(input("Digite um valor (0-255) para saída DAC: "))
    if 0 <= valor <= 255:
        dac.write(valor)
        tensao = (valor / 255) * 3.3
        print(f"Valor DAC: {valor} - Tensão de saída: {tensao:.2f} V")
    else:
        print("Digite um valor entre 0 e 255!")
```

Explicação:

- O DAC converte um valor entre **0 e 255** em uma tensão entre **0 e 3,3V** .
- A conversão da escala digital para tensão é feita por:

$$Vsaída = \frac{ValorDigital}{255} * 3{,}3$$

3. Modulação por Largura de Pulso (PWM)

A modulação PWM é utilizada para controle de potência, variação de brilho de LEDs e acionamento de motores. O ESP32 pode gerar sinais PWM em quase todos os seus pinos.

Exemplo: Controle de Brilho de um LED com PWM

O código abaixo permite ajustar o brilho de um LED inserindo um valor de **ciclo de trabalho (0-1023)** .

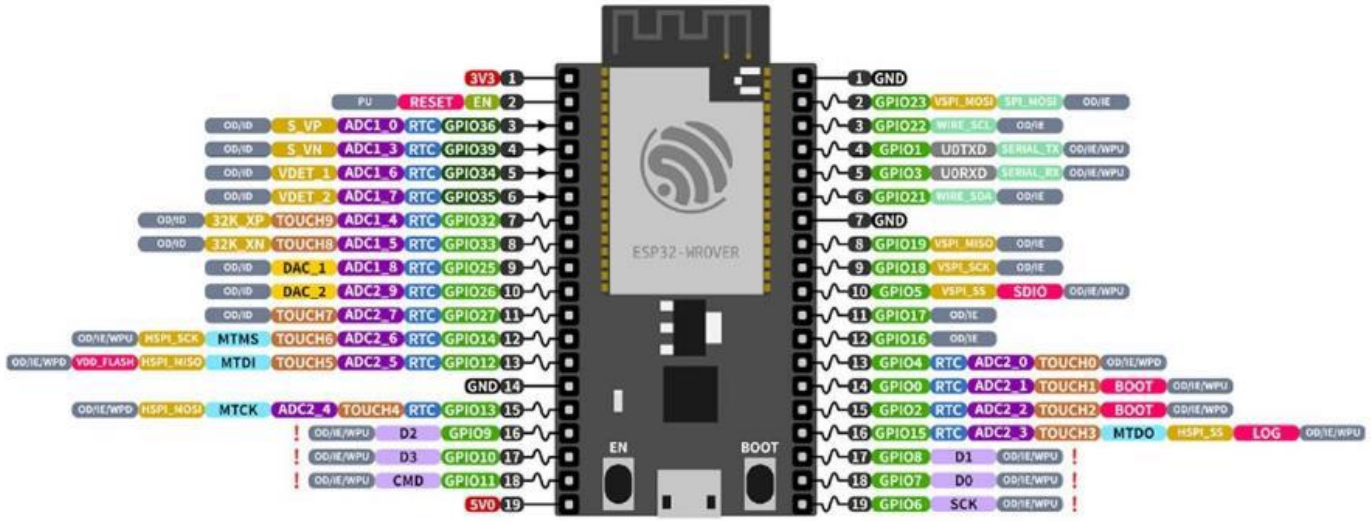
```
from machine import Pin, PWM
import time

# Configuração do pino PWM (GPIO5)
led = PWM(Pin(5), freq=1000)  # Frequência de 1 kHz

while True:
    duty = int(input("Digite o valor do duty cycle (0-1023): "))
    if 0 <= duty <= 1023:
        led.duty(duty)
        print(f"Duty Cycle ajustado para {duty}/1023")
    else:
        print("Digite um valor entre 0 e 1023!")
```

Explicação:

- O PWM tem valores de **ciclo de trabalho** entre **0 (0%) e 1023 (100%)** .
- Quanto maior o ciclo de trabalho, maior será o brilho do LED.

4. Exercícios Propostos

1. Explique o motivo pelo qual o LM35 deve ser alimentado com **5V** e não com **3.3V** para garantir leituras mais precisas no ESP32.
2. No código do DAC, se um usuário inserir o valor **128** , qual será a tensão de saída gerada? Justifique seu cálculo.
3. Qual seria o efeito prático em um LED se alterarmos a frequência do PWM de **100 Hz** para **10 kHz** ?
4. Modifique o código de leitura do **LM35** para que a temperatura lida seja convertida diretamente para um **sinal PWM** que controla o brilho de um LED. Quanto maior a temperatura, maior o brilho do LED.



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

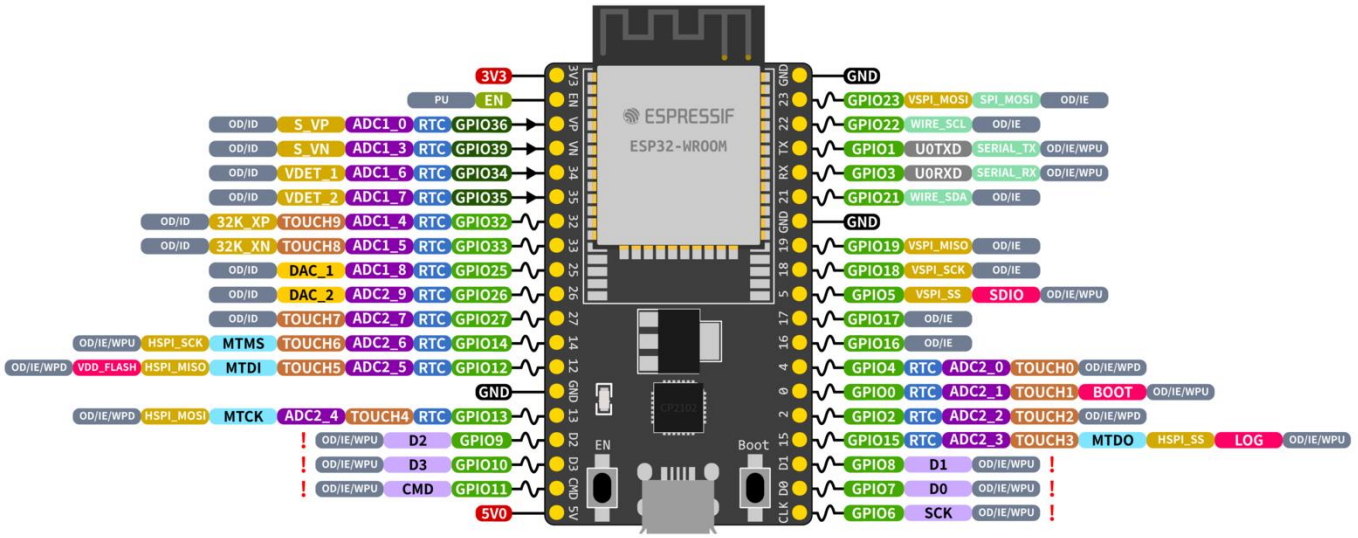
PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
JTAG/USB JTAG for Debugging and USB
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE
WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*

ESP32-DevKitC

ESPRESSIF



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions

RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
Can't be used as regular GPIO

GPIO STATE
WPU: *Weak Pull-up (Internal)*
WPD: *Weak Pull-down (Internal)*
PU: *Pull-up (External)*
IE: *Input Enable (After Reset)*
ID: *Input Disabled (After Reset)*
OE: *Output Enable (After Reset)*
OD: *Output Disabled (After Reset)*