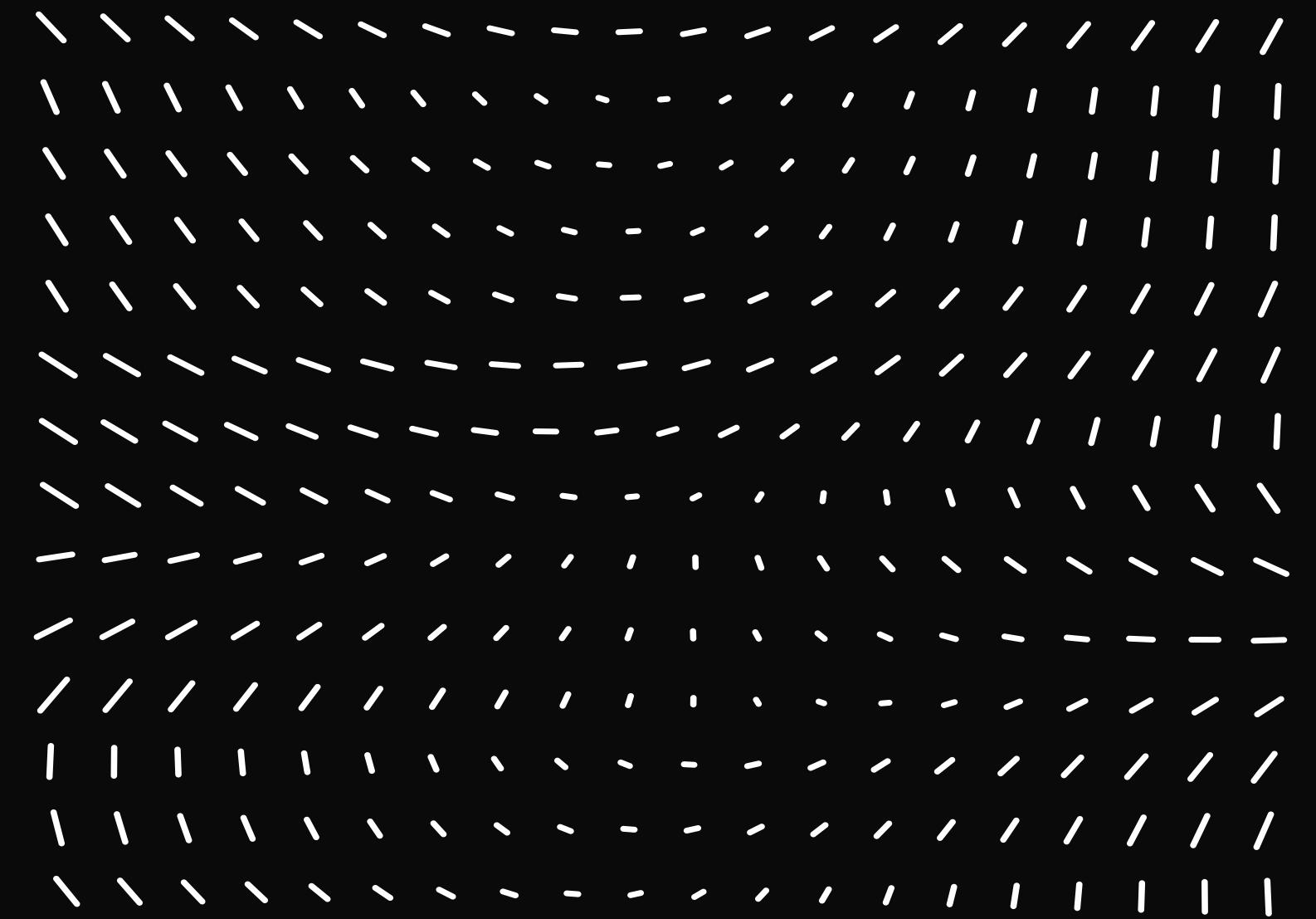


Logbook ASE



Aula 1:

→ Sistemas Embutidos vs Sistemas de Uso geral

↓
Produzidos com um determinado propósito

↳ Tem preocupações que os outros sistemas não têm

laptop, p.ex.

↓
Optimizam uso médio

→ Procurar soluções para mini-projeto

→ Prática:

- Kit Espressif
- Instalar firmware da placa:
 - ESP-IDF Command Line Tool → já faz os settings necessários
 - Plugin VSCode

~~> Exemplos:

C:// Espressif / frameworks / esp-idf-v5.0 / examples

Na command-line:

- build - idf.py build
- flash - idf.py -p [PORT] flash
- monitor - idf.py -p [PORT] monitor

_____ // _____

To do:

✓ Teste do HelloWorld (na pasta get started)

Aula 2:

→ Esp32 → Kit vs chip vs System-on-a chip (SoCs)

→ Projetos no IDF:

↳ Na command-line:

• idf.py help → aparecem todas as opções

– create-project

– docs → ajuda na página web

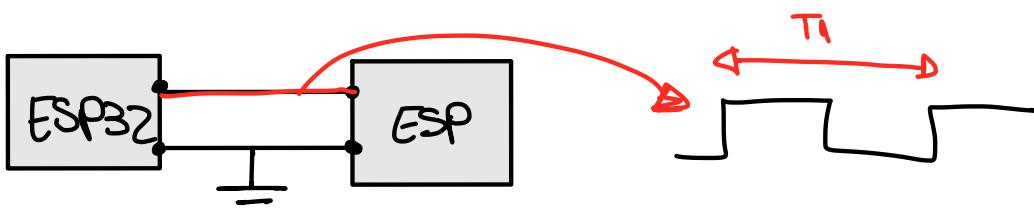
Teste do LED blink (na pasta get-started)

→ idf.py menuconfig

↳ Example configuration → para alterar o período do blink

Tarefa da próxima aula:

- 1 Geração de um sinal periódico (onda quadrada) num GPIO do kit ESP32, sendo a frequência controlada por terminal (duty-cycle de 50%).
- 2 Teste do sistema do ponto 1 com osciloscópio (trazer pontas de prova).
- 3 Aquisição de um sinal num GPIO do kit ESP32 (assumindo que é uma onda quadrada com duty-cycle de 50 %), determinação e apresentação da frequência no terminal.
- 4 Teste do sistema do ponto 3 em loopback (no mesmo ESP32), com o gerador criado no ponto 1 e testado no ponto 2.
- 5 Teste do sistema entre dois kits ESP32, em que um funciona como gerador (pontos 1 e 2) e o outro como analisador (pontos 3 e 4).
- 6 Realização de um jogo de determinação da frequência mistério entre dois kits ESP32.



Aula 3:

General Purpose Input / Output (GPIO)

→ Definir um pin digital:

gpio-set-direction(x, Y);

x → Número do pino → GPIO-NUM-x #define

Y → GPIO-MODE-OUTPUT
GPIO-NODE-INPUT

→ Num pino Output, definir valor lógico:

gpio-set-level(GPIO-NUM-x, Y)
1/0

→ Num pino input, ler um valor:

gpio-get-level(GPIO-NUM-x)



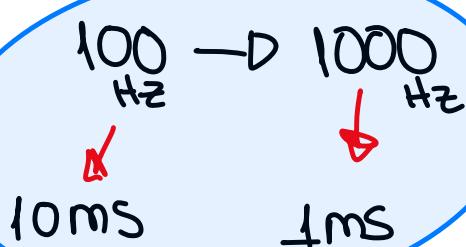
idf.py menuconfig

(por projeto)

↳ Component Config → FreeRTOS → Kernel

configTICK_RATE_HZ → inverso do pont TICK_PERIOD_MS

TICKTIME → ritmo a que o timer do OS avança



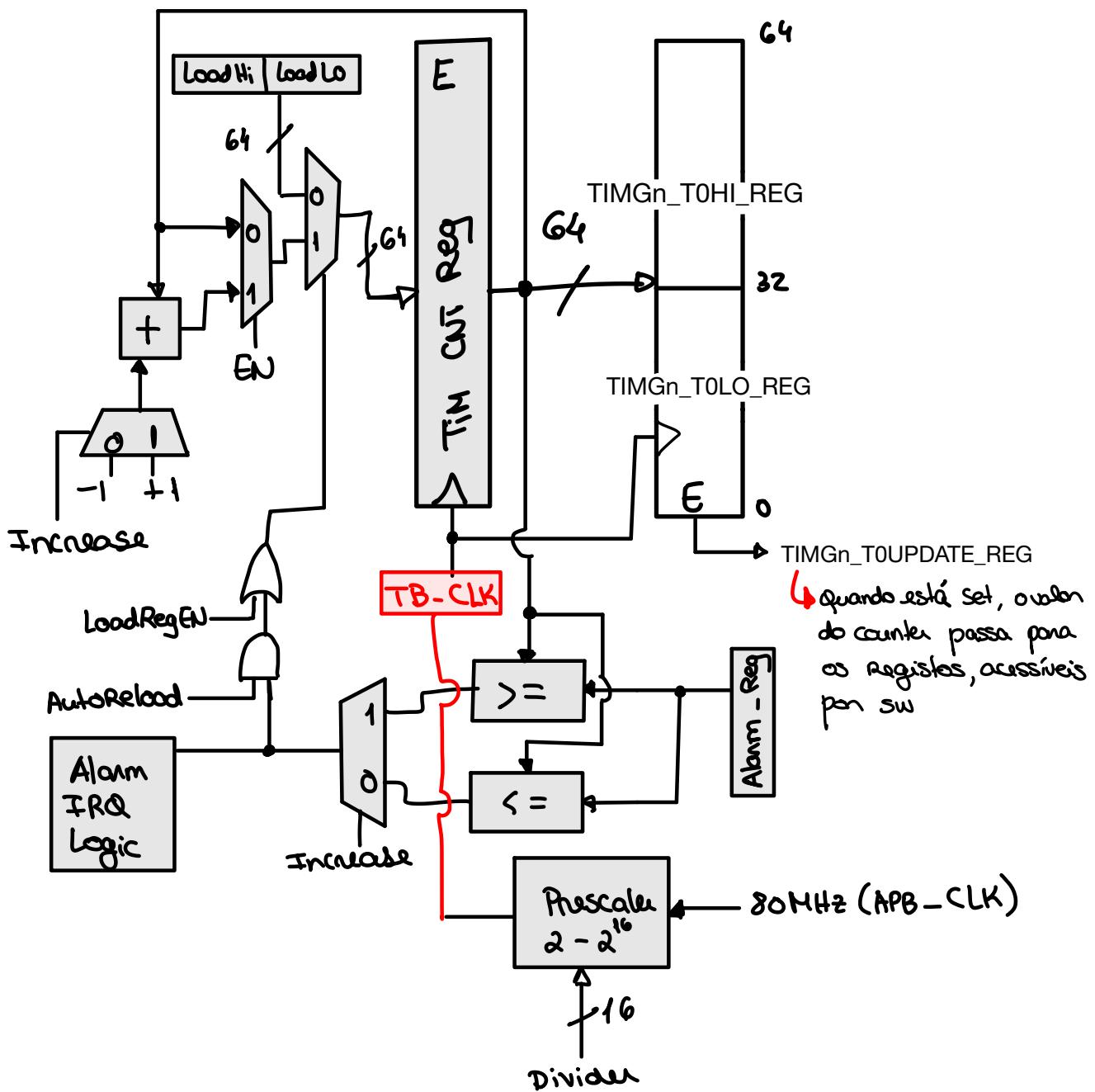
- melhora resolução temporal
- escala mais fina
- HAS...
- maior overhead
- maior consumo energético

FreeRTOS:

- Faz gestão de tarefas, e de escalonamento
- Têm noções do tempo-real, logo tem funções de tempo: vTaskDelay(...)

Aula 4:

Timers da ESP32:



Interruption Service Routine:

→ Dá apoio quando há um pedido de interrupção

Quando um periférico pede a atenção do processador

-> Que aspectos temos de ter atenção:

~> Uma ISR deve ser pequena (sucinta):

-> Salvo guardar o contexto (registos, stack, ...)

-> Desativar interrupções (algumas, dependendo da utilização, prioridade, etc.)

→ Chamar callback

→ Repor o contexto

Callback → pode ser chamada dentro da ISR para executar uma determinada tarefa

Nota: ligar os 3 pinos de GND da ESP

Aula 5:

I²C

→ Inter-Integrated Circuit

→ Protocolo de comunicação entre circuitos integrados

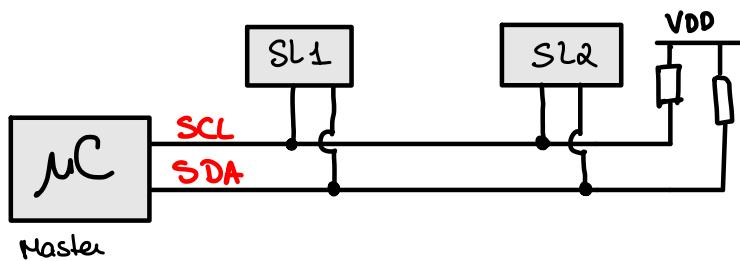
→ Série → Inicia as Comunicações

→ Master - Slave

→ Apenas 2 linhas (SDA e SCL) para haver comunicação

→ Endereçamento dos Slaves

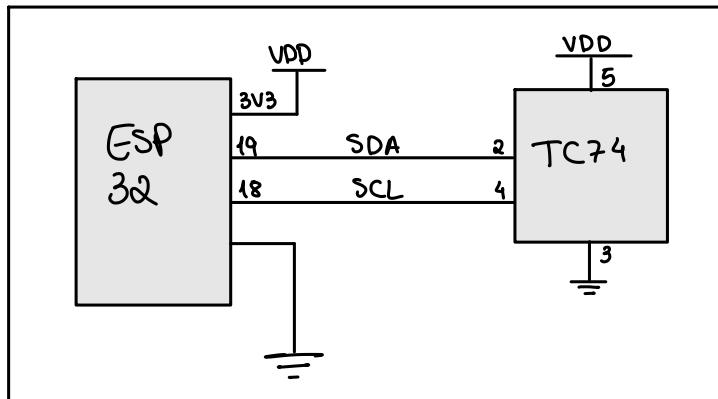
→ Simplicidade



→ Resistências de pullup:
- Serve para as linhasarem a V_{DD} quando estão no ar.

→ Limitações: velocidade e arbitragem

Sensor de Temperatura TC74 com I₂C :



SDA → Data
SCL → Clock

Write Byte Format

Start Condition

SDA Recessivo ('1')

Dados a descer com CLKativo

Slave Address
0x4D
(Do datasheet)

Command Byte: selects which register you are writing to.

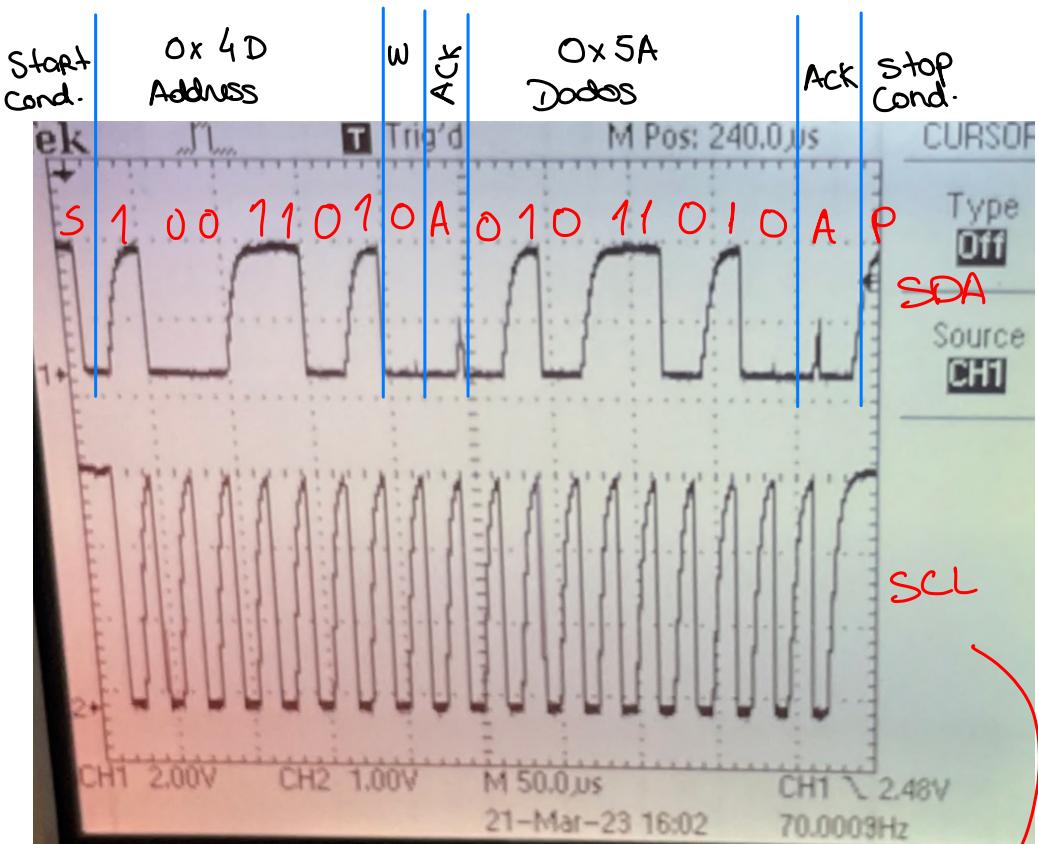
Data Byte: data goes into the register set by the command byte.
0x5A

Stop Condition

SDA sobe a Recessivo

Com CLK parado

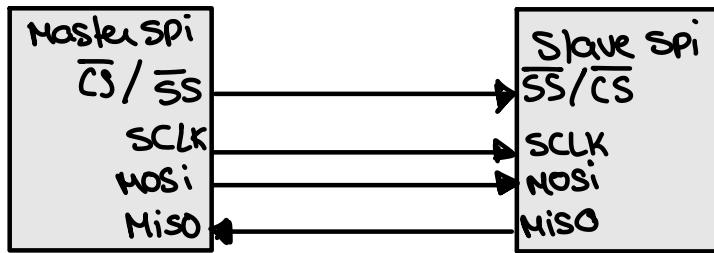
→ Ack: No Ack, o slave tenta colocar a 1 (recessivo), mas o slave responde logo com 0 dominante.
- Põe isso é que só vemos um "piquinho"



Devia estar a 50KHz, mas está a 40KHz, devido a arredondamentos internos provavelmente

Aula 6:

Serial Peripheral Interface (SPI)



SS → Chip/
Slave Select (Escolhe o slave com quem quer comunicar)

MOSI → Master Output Slave Input } ou SDI e SDO, dependendo

MISO → Master Input Slave Output } da ligação (SDI liga ao
SDO, mas
MOSI e MISO
não cruzam)

SPI vs I2C

→ SPI mais rápido pois não tem linhas bidimensionais, nem
arbitragem, etc.

- Quando tem mais que 1 slave:

Topologia em estrela:

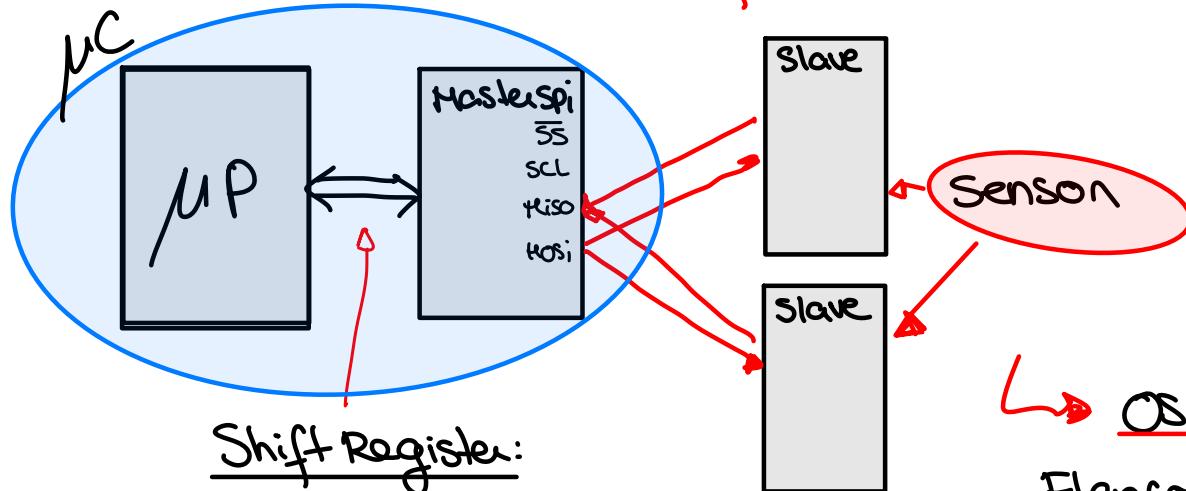
→ Tem que haver mais que um slave select, um para
cada slave

→ O slave que não está selecionado está em alta
impedância

Topologia em anel:

→ O MISO de um slave liga ao MOSI do outro
↳ mesmo SS

Usar SPI como interface com sensores:



μC coloca dados paralelos no master, que serializa para MOSI, e vice versa

Na ESP32:

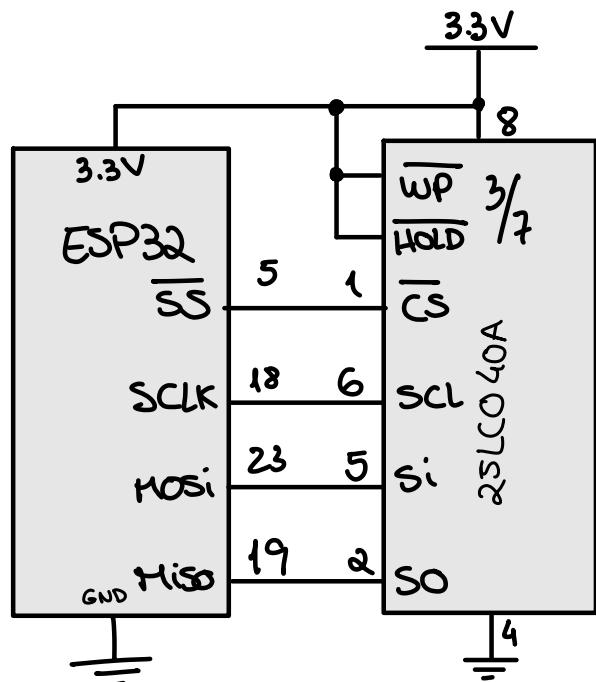
- VSPI
 - HSPI
- { Podem ser master / slave

→ Falling edge do CS

tx-data:

MOSI → temos dados 0x 5A e 0xC3, quando CS está ativo (ativo = '0')

Circuito de teste, memória EEPROM:



Tarefa integradora 1

Aula 7:

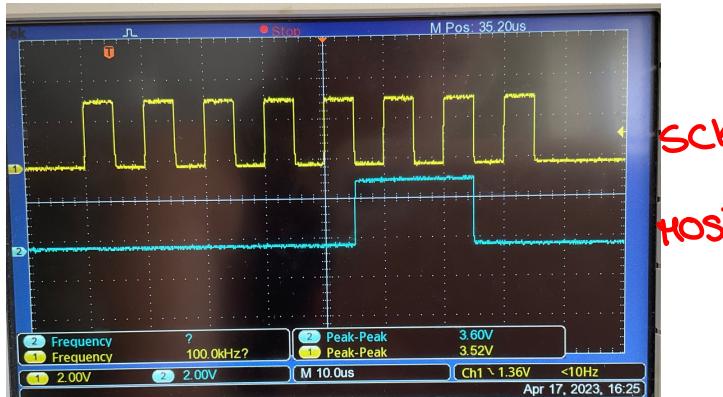
A tarefa integradora que defini na última aula consiste no seguinte:

- Aquisição, a cada 5 segundos, de 3 valores de temperatura intervalados de 100 ms (usando o sensor TC74);
- Após a aquisição dos 3 valores de temperatura, o sensor deve ser colocado em standby, só devendo ser acordado posteriormente quando for necessário adquirir novos valores;
- Cálculo da média entre os 3 valores lidos (sendo o resultado um número inteiro de 8 bits);
- Armazenamento de cada média calculada na memória EEPROM (25LC040A) até ao limite da sua capacidade;
- Quando a memória estiver completamente preenchida, deverá ser feito o seu “dump” para o terminal e construído um gráfico com os valores de temperatura.

uint8_t

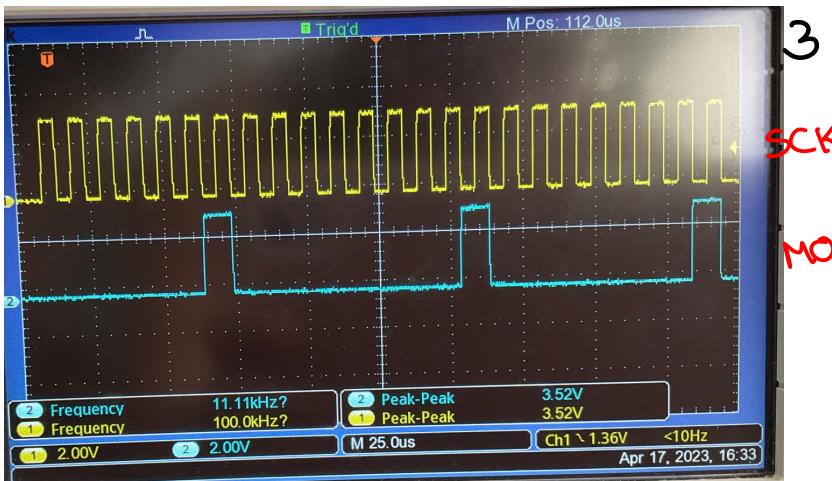
Teste do SPI:

Write Enable:



Escrita do byte
0x06 no MOSI

Write byte:



3 bytes: 0x02, 0x01, 0x01
SCK Instr Addr Datab

Write Page :



Read byte:



→ Tudo o que está fora da janela do chip Select não é relevante

Write Status:



0x05

Os bits escritos estão relacionados com o write protect (ver datasheet)

Read Status:



Só podemos escrever nos bits de Write Protect (3 e 2)

TABLE 3-2: STATUS REGISTER

7	6	5	4	3	2	1	0
-	-	-	-	W/R	W/R	R	R
X	X	X	X	BP1	BP0	WEL	WIP

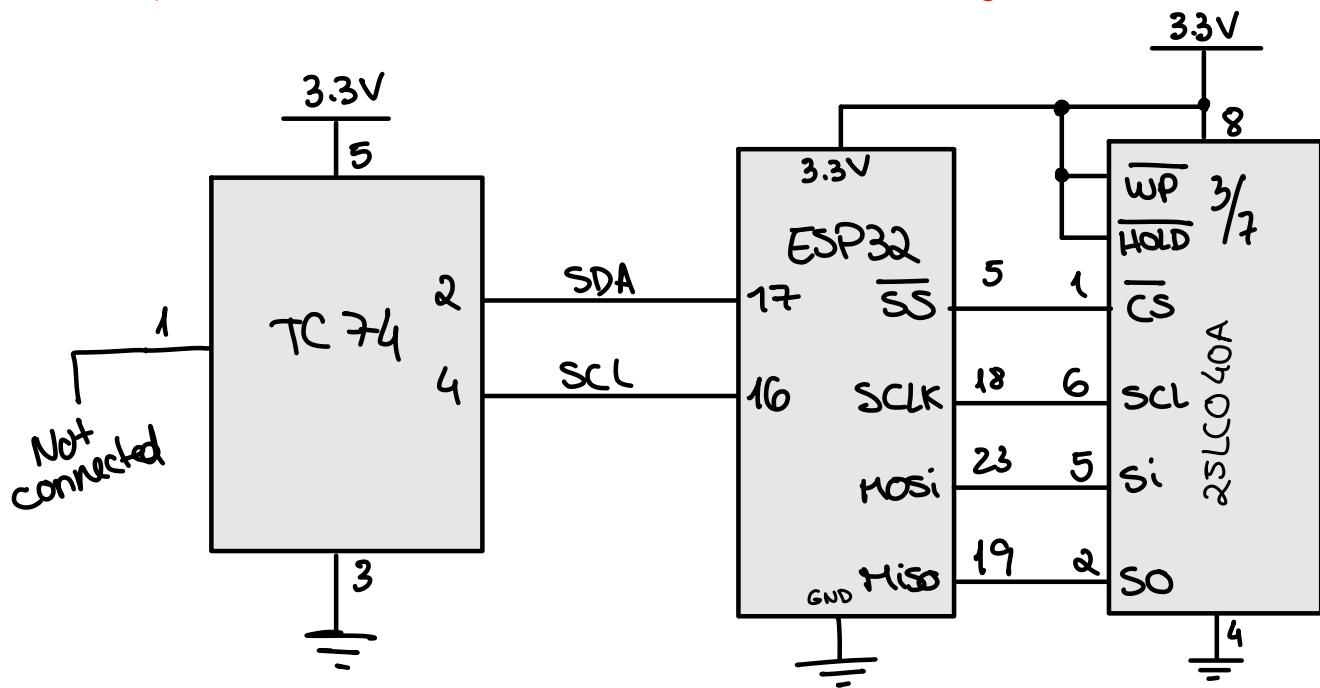
Note 1: W/R = writable/readable. R = read-only.

Notas:

① Após uma escrita, é necessário aguardar um tempo pré-definido (T_{WC}), por isso é melhor colocar um `vTaskDelay`

② É necessário colocar um `writeEnable` antes de cada escrita (não esquecer o T_{WC})

Esquemático da Tabela Integradora 1:



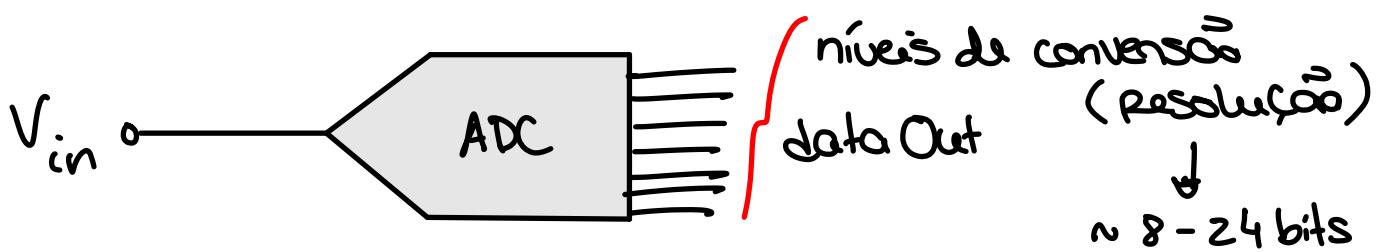
Aula 8 - ADC & DACs

Porquê usar ADC?

Porque o mundo real é **análogo**

Porquê usar DAC?

Para transmitir resultados de processamento para o mundo exterior



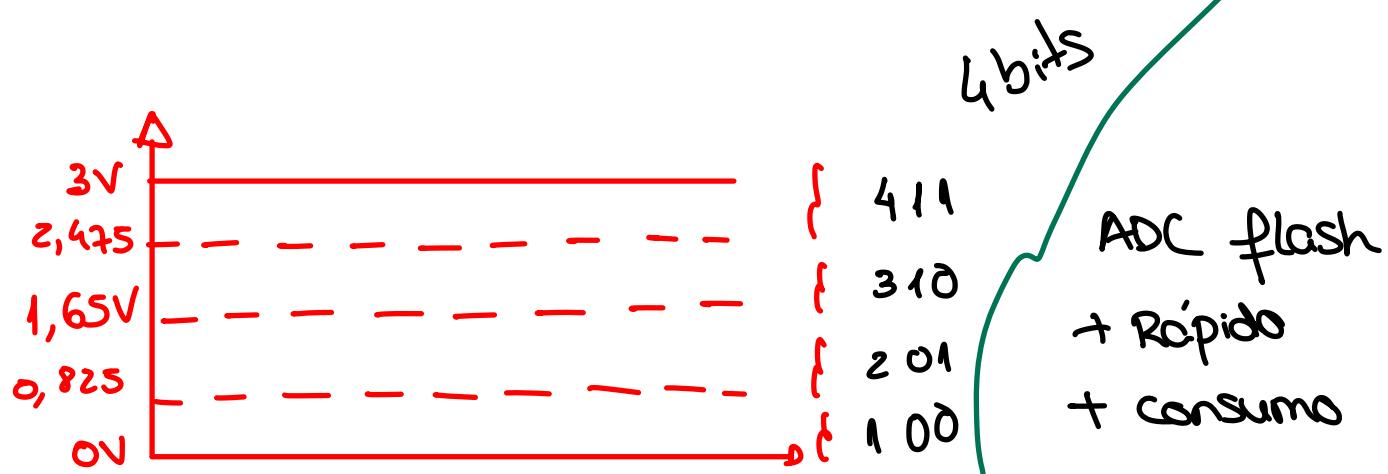
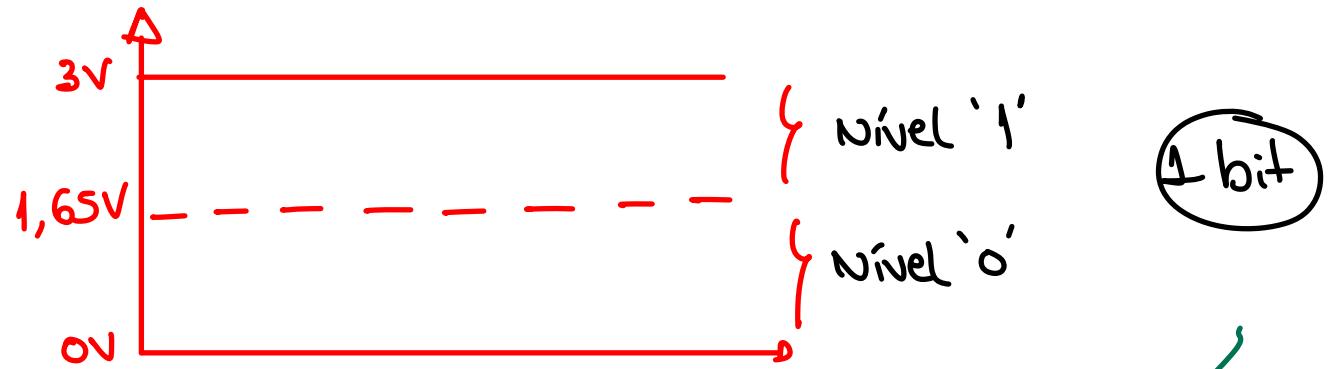
Características importantes:

- N.º de bits (resolução)

- Taxa de amostragem \rightarrow Ritmo ao qual o ADC transforma o sinal de entrada nos bits da saída

Teorema de Nyquist:

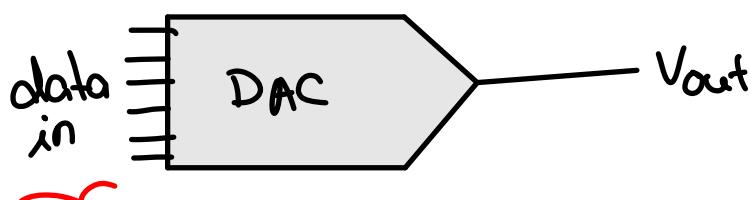
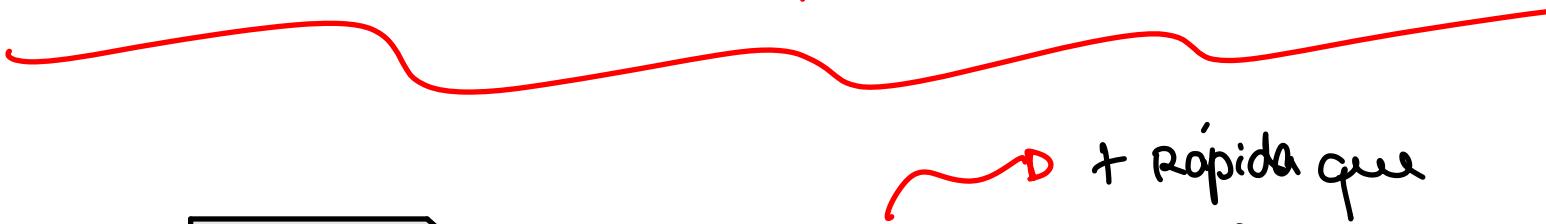
$$f_a > 2 \underbrace{f_{\max}}_{\text{do sinal}}$$



ADC do ESP32: Não é flash - é uma ADC SAR

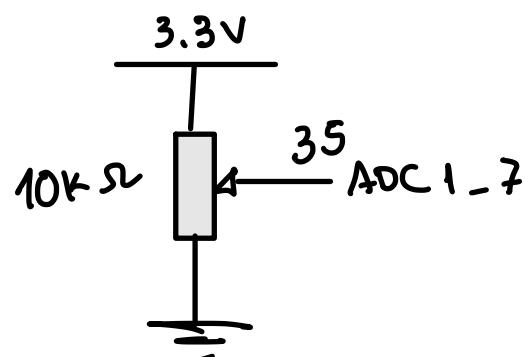
- consumo
+ Rápido

{ Successive Approximation -
Register



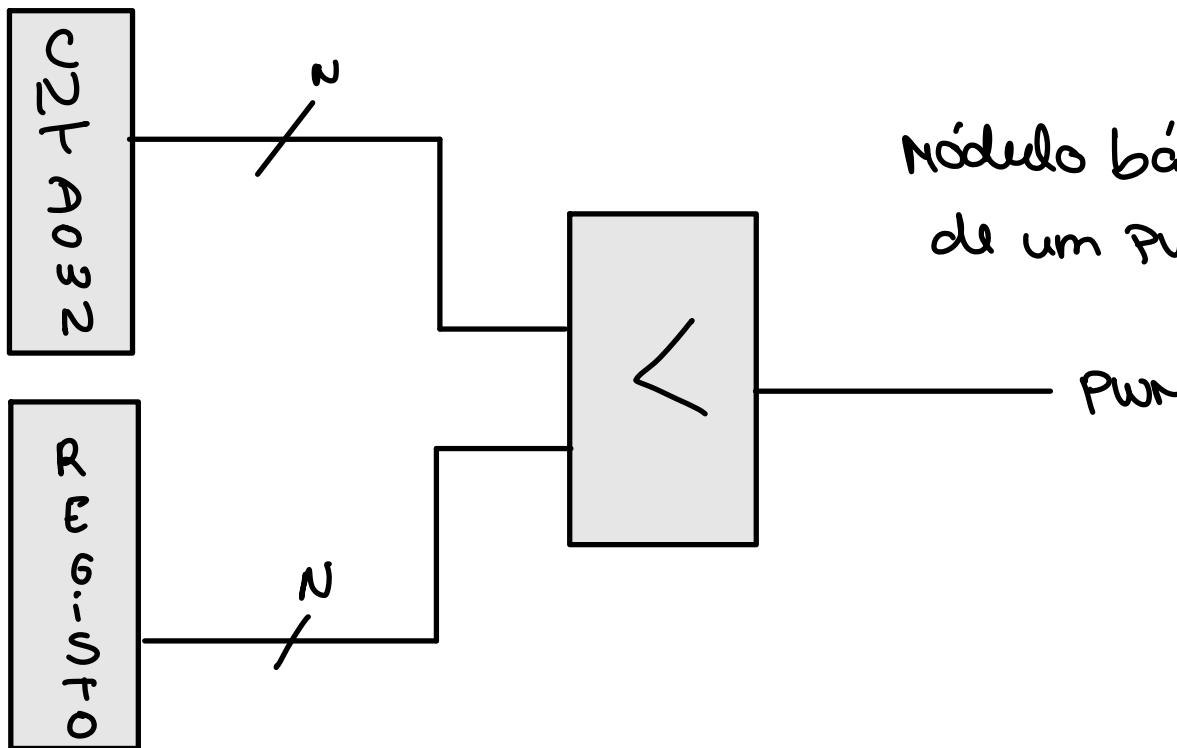
Digital

Teste ADC com potenciómetro:



Aula 9 : PWN & UART

Pulse Width Modulation



↳ Se Reg = 0 , então saída = 0 sempre ,
porque contador > 0

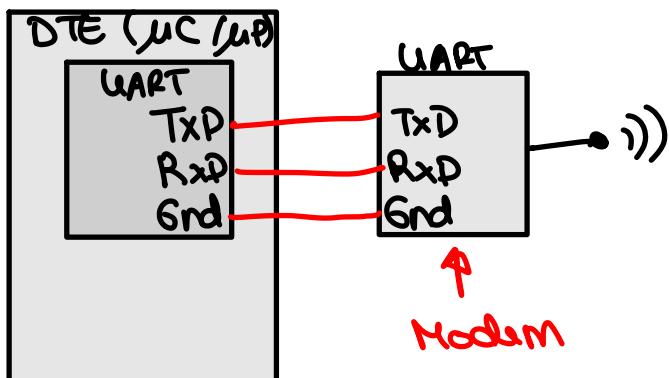
Exemplos PWM :

- peripherals / ledc / ledc-basic
- ↓ Ver também o PWM para
sist. mecânicos: MCPWM

UART (Universal Asynchronous Receiver / Transmitter)

Assíncrono → Não tem linhas de clock
Implementa o protocolo RS232:

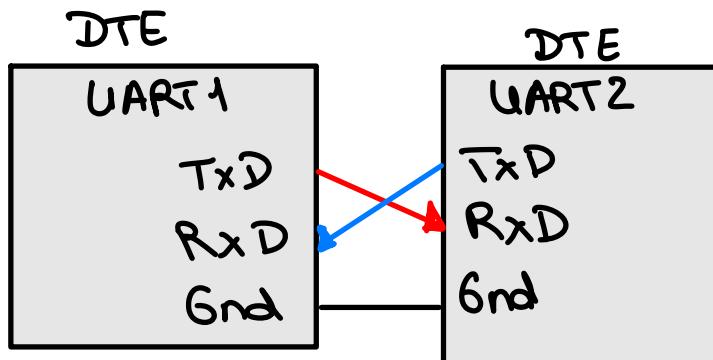
↳ Originalmente, servia para ligar dispositivos a modems



DTE: data Terminal equipment

Só que não era prático para ligar 2 DTE's

Atualmente:



- Bidirecional

Quando fazemos ligações da ESP32 com o PC, existem uma "Tradução" de RS232 para USB e vice-versa.

Sinais de controlo de fluxo da UART:

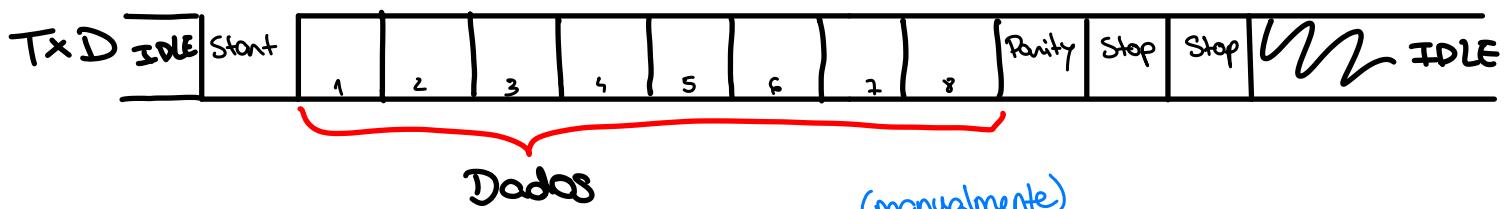
RTS : Request To Send

CTS : Clear To Send

Nota: Não é necessário o handshake, o RTS e o CTS estão ligados

Trama RS232:

- Linha inativa \rightarrow TxD a '1'
- Start bit \rightarrow TxD é colocado a zero
- Bit de paridade \rightarrow (?)



\rightarrow O programador tem de configurar nos 2 módulos RS232:

- N. de Stop bits
- N. de bits de dados (Até 8 bits)
- Baudrate
- Bit de paridade (0 ou 1)

O RxD lê sempre a meio do sinal de um bit

Possíveis erros que podem ocorrer na transmissão:

Erro de paridade: a paridade não corresponde ao calculado

Framing error: Mal formação (receber 1 stopbit quando estava à espera de 2)

Data overrun/overflow: quando o n.º de bits de dados recebidos é diferente do configurado

Componentes da UART:

- FIFOs de Rx e Tx
- Shift Registers
- Máquina de estados

13.3.2 UART Architecture

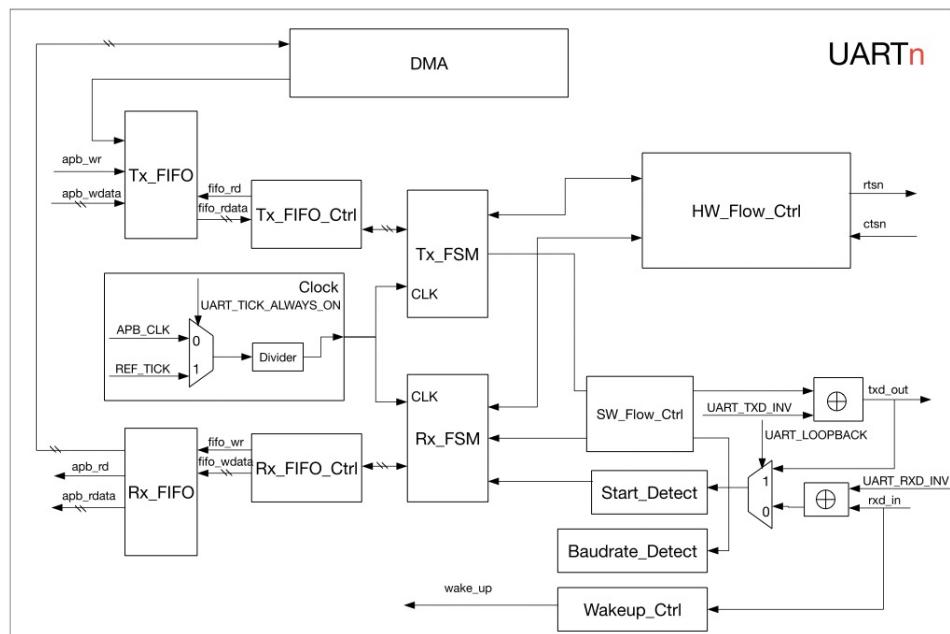


Figure 13-1. UART Basic Structure

page 340 - Technical Reference Manual

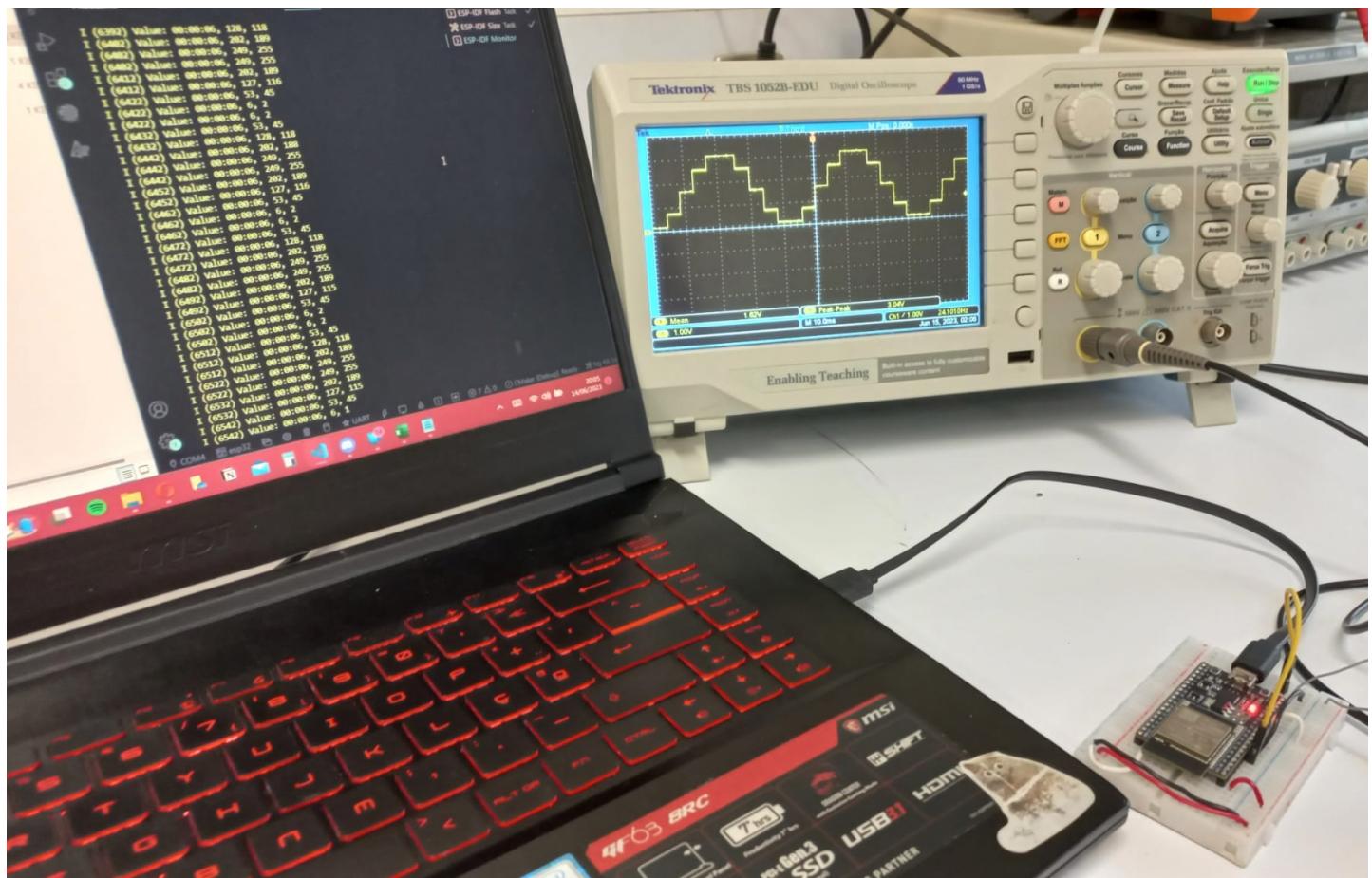
Exemplos da UART:

- peripherals /uart /uart_async_rx_tx_tasks

Aula 10 - Tarefa Integradora 2

✓ A segunda e última tarefa integradora a realizar na próxima semana consiste no seguinte:

- Determinação das amostras de um sinal sinusoidal, com uma frequência de 50 Hz, $T=0,02s$ definido com 10 amostras por período, cada uma resolução de 16 bits (c/sinal) – pode ser feito com o Matlab.
DAC tem 8 bits
 $T_a = \frac{0,02}{10} = 0,002$
 $f_a = 500\text{Hz}$
- Armazenamento das amostras num array com a dimensão adequada.
- Geração na saída de uma DAC, de forma cíclica, do sinal cujas amostras estão armazenadas no array definido no ponto anterior.
- Amostragem com uma ADC do sinal gerado pela DAC (loopback analógico da DAC para a ADC) (as taxas de amostragem da DAC e da ADC devem ser iguais).
- Dump para uma porta série das amostras aplicadas na DAC e das recolhidas da ADC na forma: <Timestamp>, <DACsample>, <ADCsample>, sendo todos os campos representados como strings contendo dígitos decimais.
- Geração de um sinal PWM, com uma taxa de amostragem 100 vezes superior à da DAC e da ADC, e cujo duty-cycle deve ser diretamente proporcional ao valor da amostra (considerando a excursão/gama da DAC/ADC). (ledc para gerar PWM)
- Visualização de todos os sinais relevantes com o osciloscópio.



Tipos de variáveis e Respetivo armazenamento:

```
int f (...) {
```

{ int n; } Variável local automática. É criada quando a função é chamada e destruída quando a função termina. É armazenada na stack.

```
int f (...) {
```

{ static int; } estática, não automática. Não pode ser criada fora de função, mas mantém sempre o mesmo valor em diferentes chamadas à função. Normalmente é inicializada, mas só tem esse valor na primeira chamada. É armazenada no .data (sig. dados estáticos)

int m; → global e estática, nunca é destruída e reside no .data.

```
int f (...) {
```

{ Se não tiver static, pode ser visível em todos os módulos. Se tiver, só pode ser visível no ficheiro em que está ficheiros

int f (...) { → preciso fazer o cast, pois o malloc retorna (void *)

int * p = (int *) malloc (10);

{ 4 + 10 ≈ 14 bytes
 stack heap

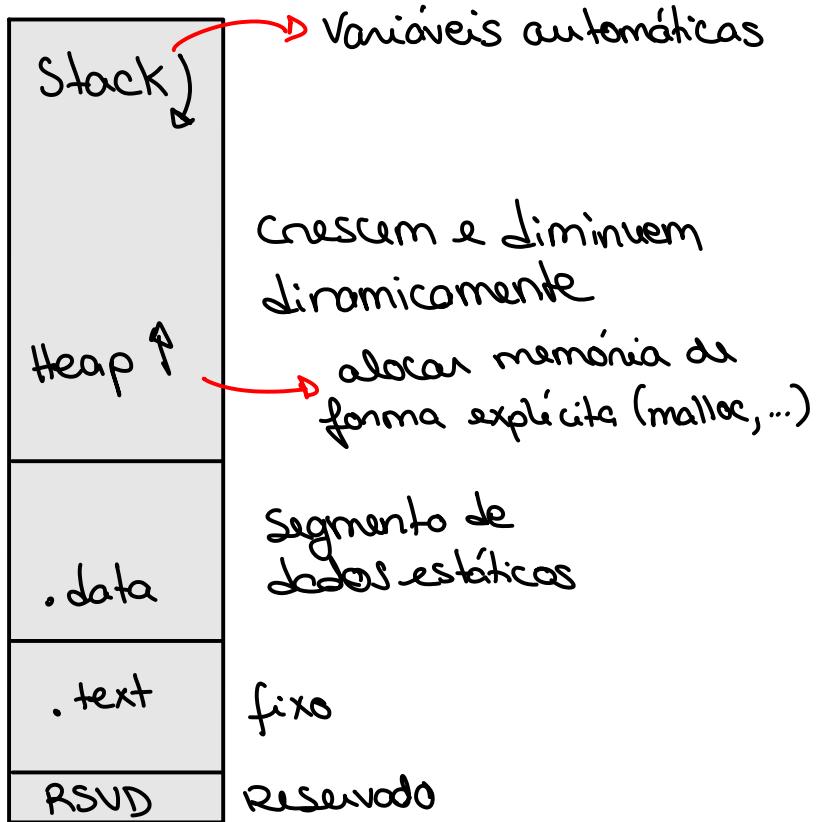
malloc → devolve a posição de memória do 1º elemento

não esquecer → ou null se der erro de memória

char *
int *
float *
...

{ endereço de
memória

/ em larg. de 32 bits,
cada 1 destes variáveis
ocupa 32 bits



Toolchain de compilação:

- Pré-processador (junta os ficheiros .h e .c e devolve define, compilação condicional, ... (#))
- Compilador (sai em .o / .obj)
- Linker (pega em todos os .o → junta num .elf / .exe)
 - ↳ o linker junta também em bibliotecas pré compiladas (libm.a)
 - ↳ windows
 - ↳ archive

Projeto:

Os requisitos obrigatórios do mini-projeto são os seguintes:

- ✓ - Utilizar o ESP32DevKitC como base do embedded system;
- ✓ - A aplicação a executar no kit ESP32DevKitC deve ser desenvolvida em C/C++ e tirar partido do FreeRTOS;
- ✓ - Devem ser explorados os periféricos do ESP32 que fizerem sentido no contexto do projeto, incluindo aspetos de interrupções e DMA;
- ✓ - Os dados recolhidos do sensor e processados no ESP32 devem ser apresentados num dashboard remoto, sendo para tal necessária conectividade de rede (WiFi / BT);
- ✓ - Deve ser disponibilizada uma ligação por Terminal; independente do dashboard remoto;
- ✗ - Devem ser exploradas as várias funcionalidades das ferramentas de desenvolvimento, incluindo debug.

Os aspetos opcionais a incluir no mini-projeto são os seguintes (não sendo uma lista fechada):

- ✗ - Podem ser explorados os modos de baixo consumo energético do ESP32;
- ✗ - Podem ser suportadas atualizações remotas (Over-the-Air) do sistema;
- ✓ - Pode ser incluído algum tipo de atuador cuja utilização faça sentido com o sensor usado (de forma a criar um loop de controlo; ou que seja controlado através do dashboard);
- ✗ - Pode ser suportado um sistema de ficheiros para armazenar dados localmente.

Ventoinha

Devido ao PWM
não é possível

