PADO

Labs

Microcontroladores Aplicados a IoT

Dilson Liukiti Ito

04 – Watchdog, Timer e UART

Microcontroladores Aplicados a IoT

Watchdog

- Cão de guarda;
- um **Watchdog Timer** serve para seu sistema não permaneça travado caso ocorra alguma falha de software ou hardware;
- É reiniciado pela task "IDLE", de prioridade 0, disponível uma para cada núcleo;
- Caso uma task não entre no estado bloqueado fará com que as tasks de menor prioridade sofram de "starvation";

https://www.embarcados.com.br/esp32-watchdog-timer/

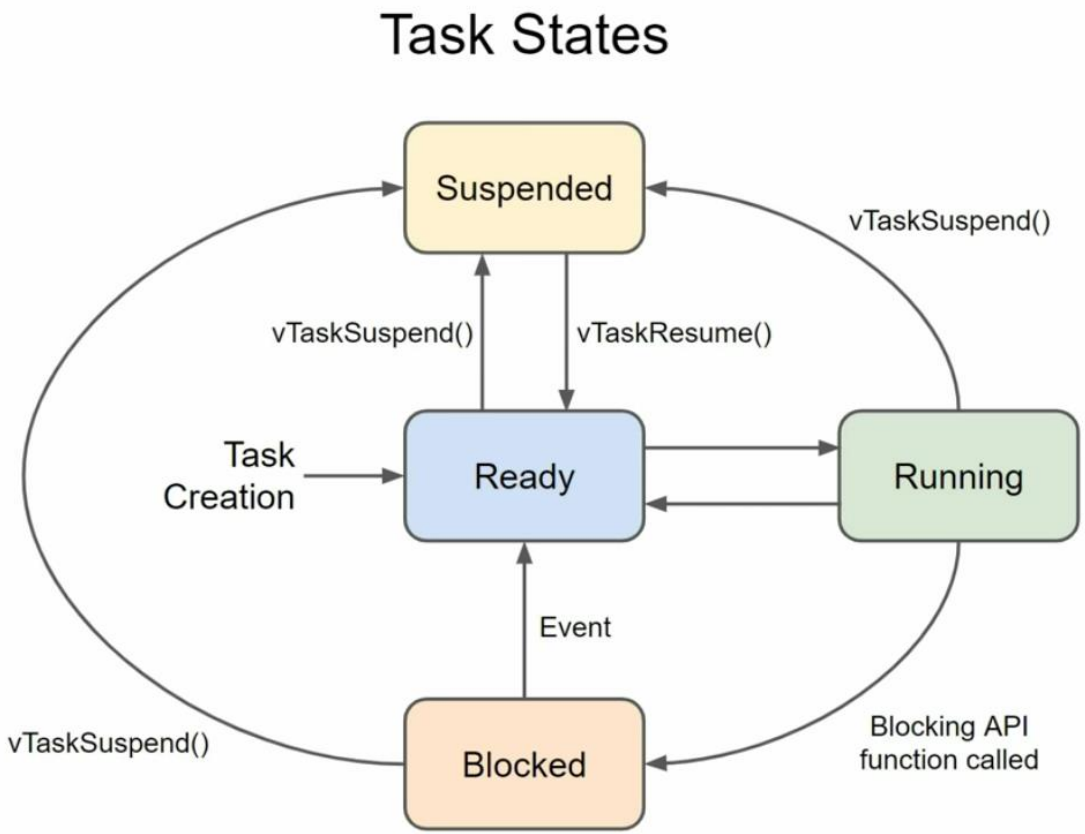
Exemplo 1

- Crie um arquivo watchdog.c

```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void task1 (void *param) {
    while(1) {

    }
}

void app_main(void) {
    xTaskCreate(&task1, "task1", 2048, NULL, 1, NULL);
}
```

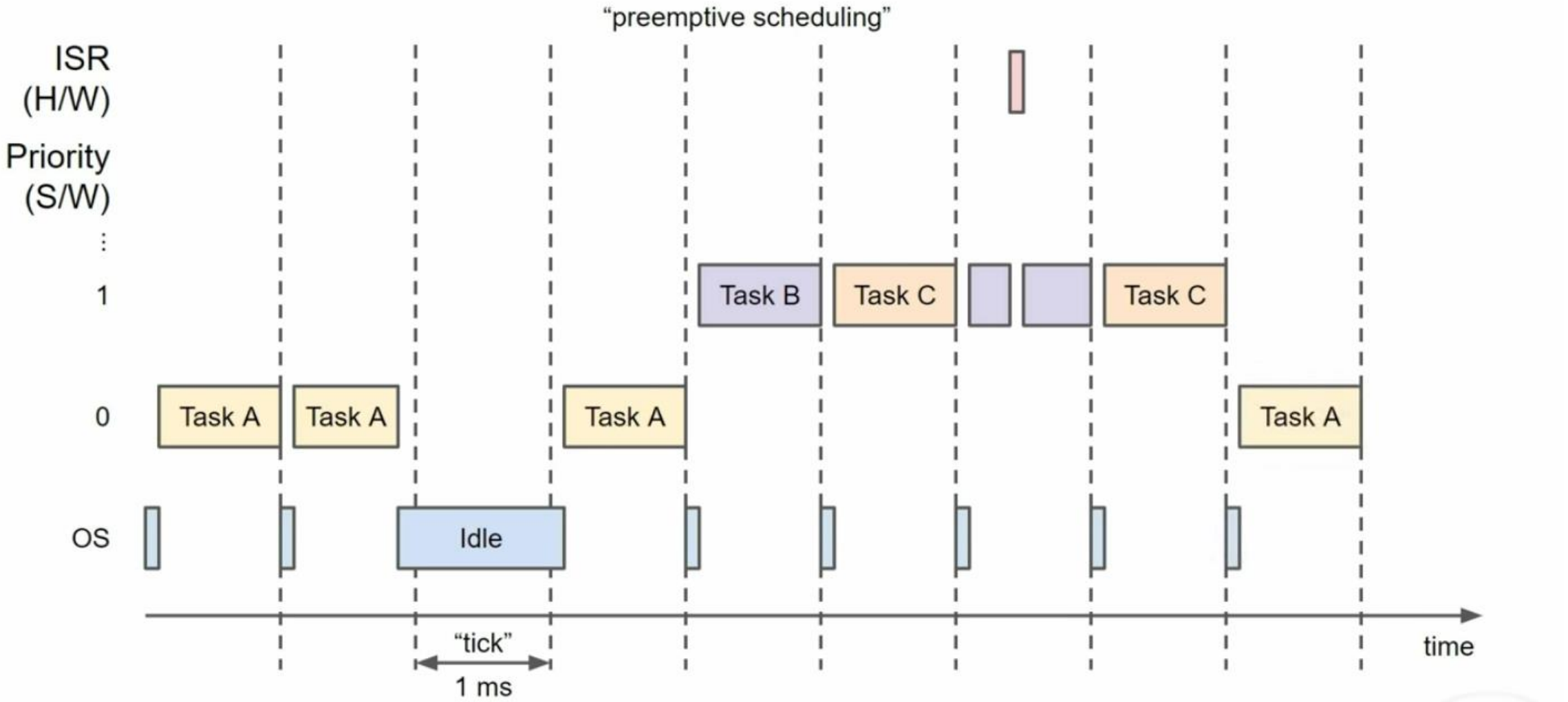
- Obs.: A task criada (task1) não se bloqueia. Isso fará com que tasks de menor prioridade não sejam executadas;

Exemplo 1: Estados da task



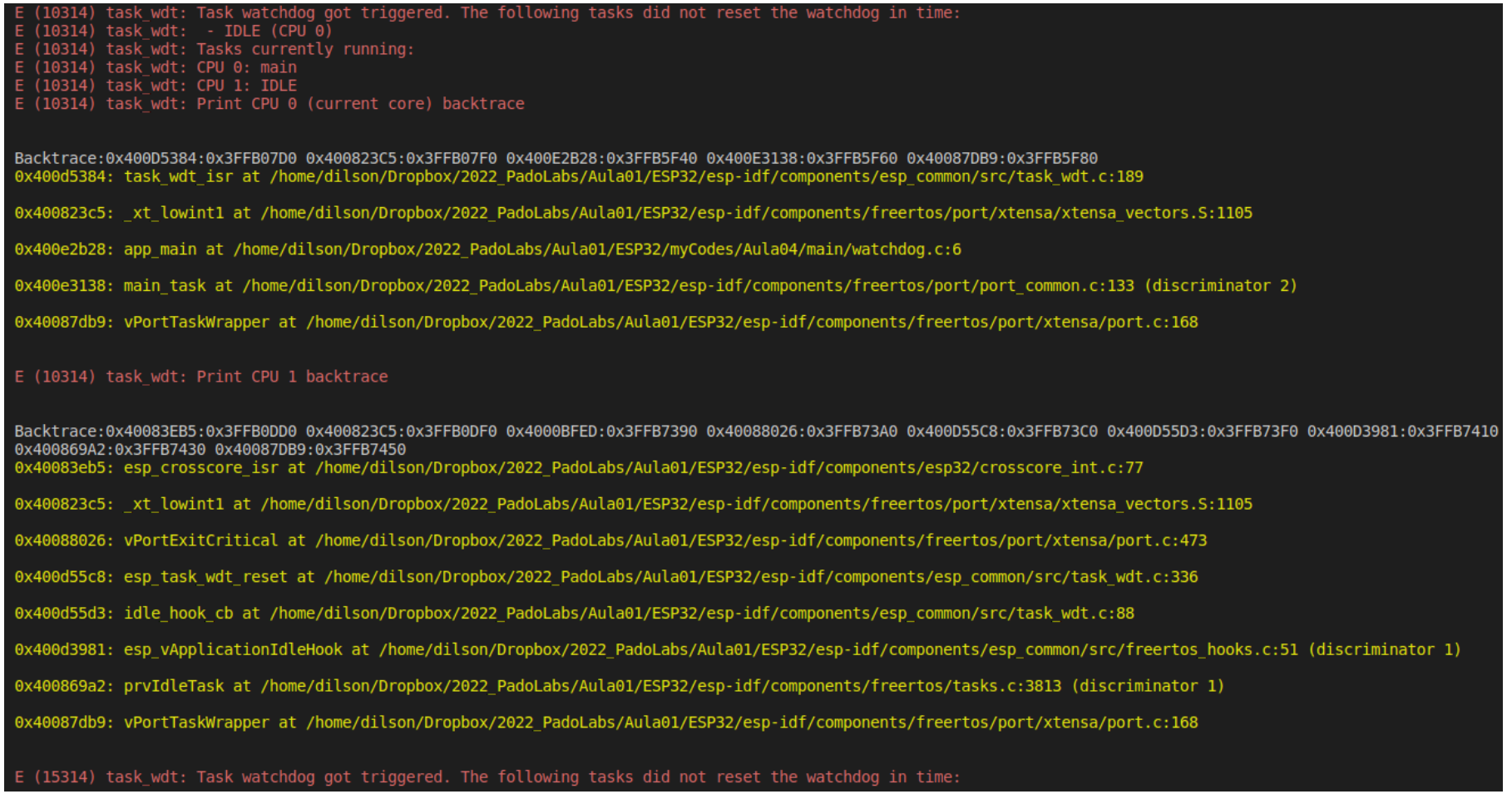
https://www.youtube.com/watch?v=95yUbClyf3E&list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz&index=4

Exemplo 1: Scheduler and Tasks



https://www.youtube.com/watch?v=95yUbClyf3E&list=PLEBQazB0HUyQ4hAPU1cJED6t3DU0h34bz&index=6

Exemplo 1: Acionamento do Watchdog



- As mensagens irão aparecer de 5 em 5 segundos;

Watchdog (cont.)

$ idf.py menuconfig

- Component config / Commom ESP-related

```
(Top) → Component config → Common ESP-related
                    Espressif IoT Development Framework Configuration
[*] Enable lookup of error code strings
(32) System event queue size
(2304) Event loop task stack size
(3584) Main task stack size
(1024) Inter-Processor Call (IPC) task stack size
[*] IPC runs at caller's priority
(2048) Minimal allowed size for shared stack
    Channel for console output (Default: UART0)  --->
[*] Interrupt watchdog
(300)   Interrupt watchdog timeout (ms)
[*]     Also watch CPU1 tick interrupt
[*] Initialize Task Watchdog Timer on startup
[ ]     Invoke panic handler on Task Watchdog timeout
(5)     Task Watchdog timeout period (seconds)
[*]     Watch CPU0 Idle Task
[*]     Watch CPU1 Idle Task
[ ] Place panic handler code in IRAM


[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info            [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Exemplo 1: Altere a prioridade para 0


```c
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

void task1 (void *param) {
    while(1) {

    }
}

void app_main(void) {
    xTaskCreate(&task1, "task1", 2048, NULL, 0, NULL);
}
```

- O que acontece? E por quê?

Evitar estouro de watchdog

- Adicionar delays ou funções bloqueantes (esperar elemento de fila, semáforo);

```
void task1(void)
{
    while(1) {
        vTaskDelay( pdMS_TO_TICKS( 100 ) ); // delay de 100 milissegundos
        vTaskDelay( 100 / portTICK_PERIOD_MS ); // delay de 100 milissegundos
    }
}
```

Download watchdog.c

Timer

- Temporizador: contadores com intervalos programáveis;
- Permite gerar interrupções em intervalos regulares de tempo
- Documentação Oficial:
  - https://docs.espressif.com/projects/esp-idf/en/v4.3.2/esp32/api-reference/system/esp_timer.html

```
#include "esp_timer.h"
esp_timer_handle_t timerHandler;
```

Timer (cont.)

- A função de callback é chamada quando o tempo se esgota
- Importante! Realizar o mínimo de tarefas nas funções de callback: não utilizar loop infinito, não esperar por eventos ou utilizar vTaskDelay();
- Limite mínimo de 20us para o temporizador de uso único
- Limite mínimo de 50us para o temporizador periódico

https://docs.espressif.com/projects/esp-idf/en/v4.3.2/esp32/api-reference/system/esp_timer.html

Timer (cont.)


```
esp_timer_handle_t timerHandler;
```

```
esp_timer_create_args_t timer_args = {
    .callback = timerCallback,
    .arg = NULL,
    .name = "testTimer",
    .dispatch_method = ESP_TIMER_TASK,
};
```

```
void timerCallback (void* param)
{

}
```

```
esp_timer_create(&timer_args, &timerHandler);
```

```
esp_err_t err = esp_timer_start_once(timerHandler, timeout_us);
```

```
esp_err_t err = esp_timer_start_periodic(timerHandler, timeout_us);
```

```
esp_err_t err = esp_timer_stop(timerHandler);
```

Exemplo: timer

- Vendo a diferença entre o start_once e o start_periodic;

Download timer.c

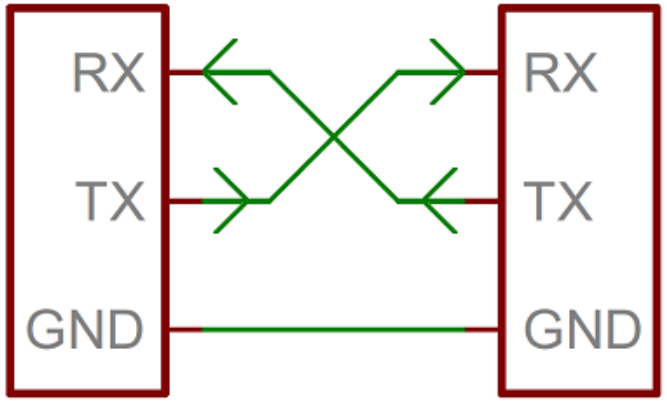
Exemplo: timerBlink

- Vamos utilizar o timer para fazer o acionamento de um LED;
- Monte um circuito com um LED;
- Execute o programa timerBlink.c

Download timerBlink.c

UART

- Universal asynchronous receiver/transmitter
- protocolo, ou seja, um conjunto de regras para a troca de dados seriais entre dois dispositivos
- utiliza somente dois fios entre o transmissor e o receptor para transmitir e receber em ambas as direções



https://www.robocore.net/upload/tutoriais/50_img_9_H.png?939
https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

UART (cont.)

- A velocidade de transmissão deve ser predefinida;
- As taxas de baud mais comuns utilizadas em UART atualmente são 4800, 9600, 19200, 57600 e 115200;
- Além de ter a mesma taxa de bauds, ambos os lados de uma conexão UART também têm que usar a mesma estrutura de frames e parâmetros;

https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

UART (cont.)

- o transmissor precisa sinalizar que os bits de dados estão chegando. Isso é possível ao utilizar o bit inicial. O bit inicial é uma transição do estado inativo para um estado baixo
- o bit final indica o fim dos dados do usuário. O bit de parada é uma transição de volta para o estado alto ou inativo. Podem haver dois bits de parada
- É comum haver um bit de paridade para checar por eventuais erros na transmissão

https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

UART (cont.)

- Os bits de dados são dados de usuário ou bits "úteis" e vêm imediatamente depois do bit inicial. Pode haver de 5 a 9 bits de dados de usuários, apesar de ser mais comum haver 7 ou 8 bits.
- Esses bits de dados geralmente são transmitidos com o bit menos significativo primeiro.

https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

UART (cont.)

- letra maiúscula "S" em um ASCII de 7 bits, a sequência de bits é 1 0 1 0 0 1 1. Primeiro invertemos a ordem dos bits para colocá-los na ordem menos significativa, ou seja 1 1 0 0 1 0 1
- 7 bits ASCII 'S' (0x52) = 1 0 1 0 0 1 1
- Ordem LSB = 1 1 0 0 1 0 1

https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

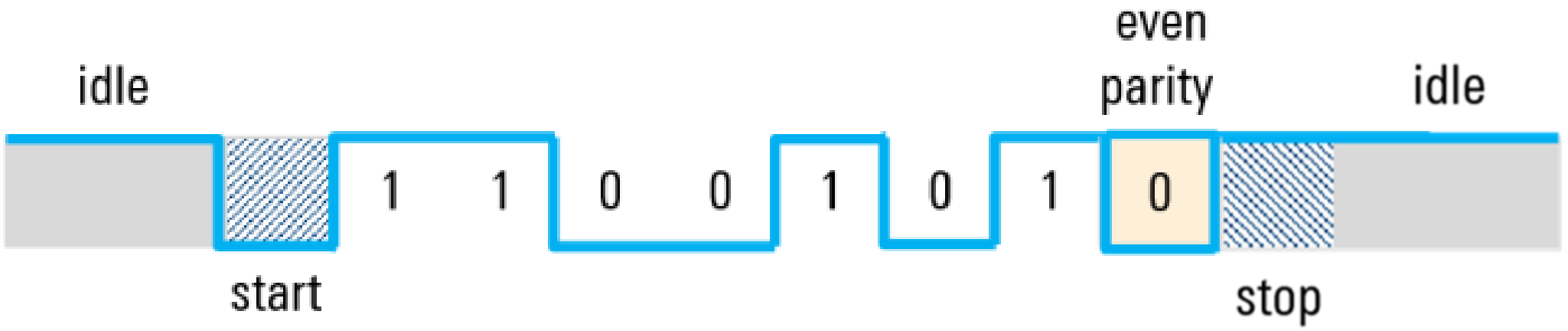
UART (cont.) / Paridade

- bit opcional de paridade que pode ser utilizado para detecção de erros
- O valor do bit de paridade depende do tipo de paridade sendo utilizado (par ou ímpar):
- Na **paridade par**, esse bit é definido de modo que o número total de 1s no frame será par.
- Na **paridade ímpar**, esse bit é definido de modo que o número total de 1s no frame seja ímpar.

https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

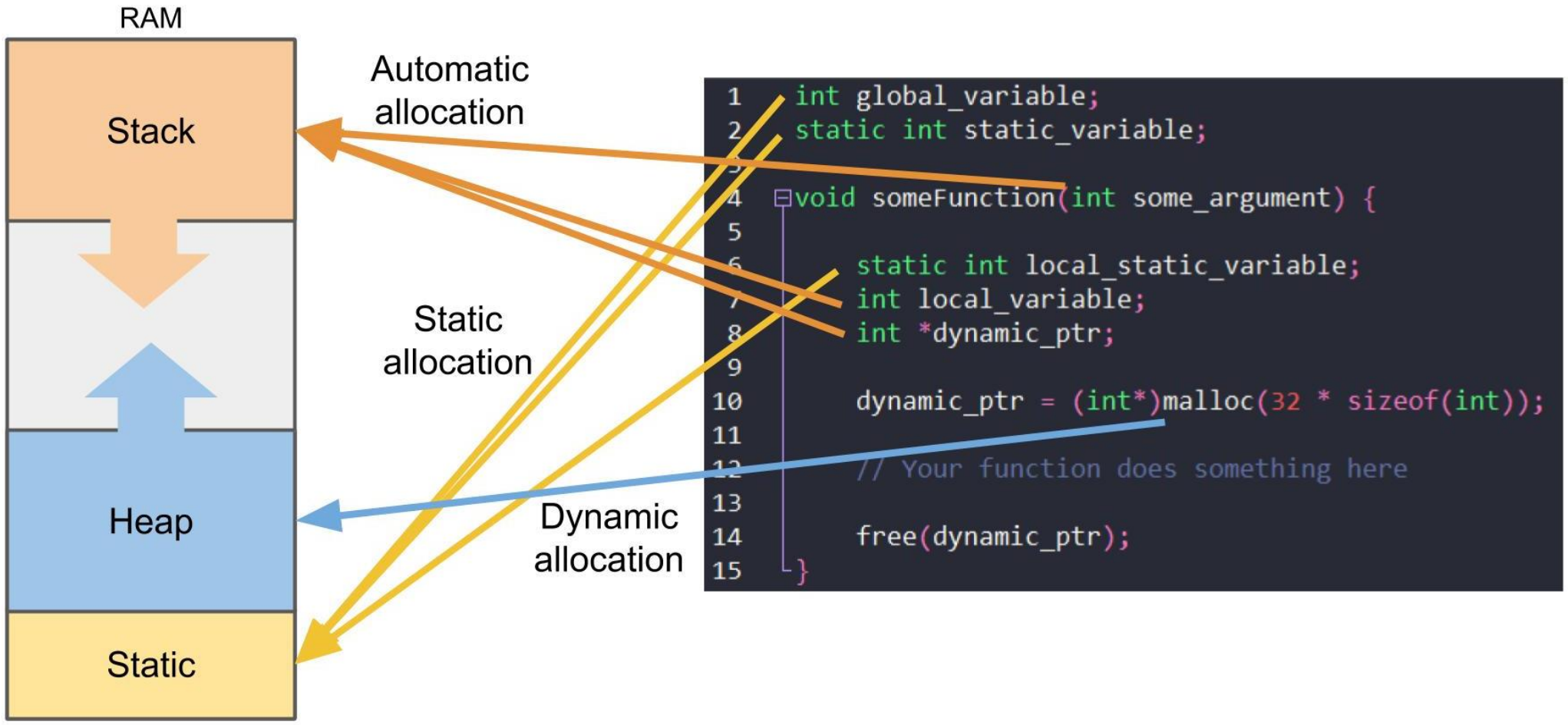
UART (cont.) / Paridade

- O "S" maiúsculo (1 0 1 0 0 1 1) contêm um total de três zeros e quatro uns. Ao utilizar a paridade par, o bit de paridade é zero porque já existe um número par de uns. Ao utilizar a paridade ímpar, o bit de paridade tem que ser um para fazer o frame ter um número ímpar de 1s.



https://www.rohde-schwarz.com/br/produtos/teste-e-medicao/osciloscopios/educational-content/compreender-uart_254524.html

Alocação de memória



https://www.digikey.com/en/maker/projects/introduction-to-rtos-solution-to-part-4-memory-management/6d4dfcaa1ff84f57a2098da8e6401d9c

Exemplo: Entrada de teclado

- Vamos utilizar os conceitos vistos no módulo 1 sobre C e vamos montar um programa que receba dados do teclado e os mostre no terminal

Download teclado.c

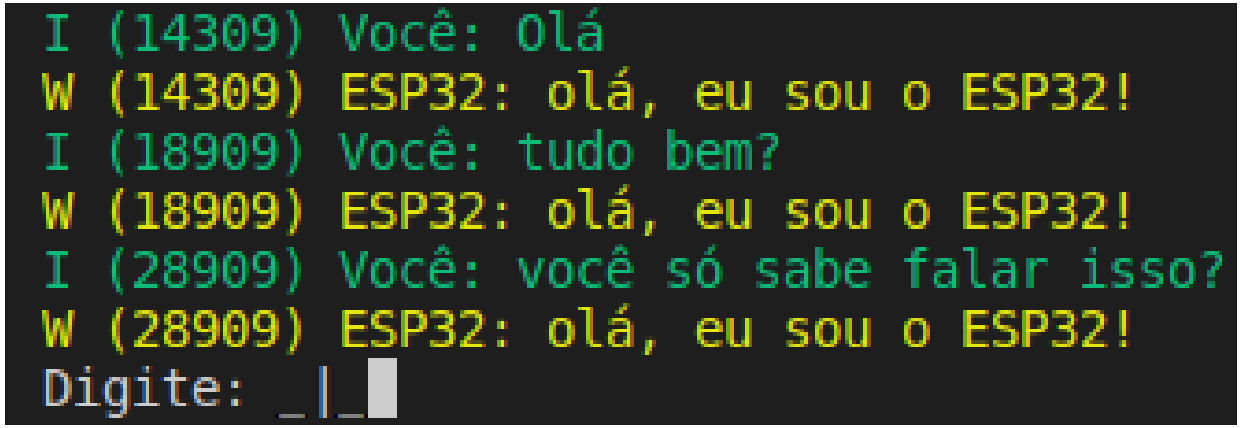
Exemplo: UART / Events

- Este exemplo ilustra o gerenciamento de eventos na UART do ESP32 por meio de tasks e filas

  Download uart_events.c

Exercício

- Faça um programa que simule um chat pela serial:
  - As mensagens deverão aparecer na tela como uma conversa de WhatsApp;
  - Utilize o programa "Entrada de teclado" como base para criar as mensagens a serem enviadas;
  - Você deverá conectar seu ESP32 com o ESP32 de um colega por meio da UART para vocês trocarem mensagens e status pela serial



Fonte: Comando para limpar uma linha do terminal

Exercício (cont.)

- Cada um deverá montar um sistema de status, que consiste em um circuito com 3 LEDs que serão acionados nas seguintes situações:
  - LED Vermelho deverá ligar quando seu colega não digitar nada em seu terminal por mais de 30 segundos;
  - LED Amarelo deverá permanecer piscando em 10Hz enquanto seu colega estiver digitando uma mensagem;
  - LED Verde deverá notificar o recebimento de novas mensagens enviadas pelo seu colega, devendo desligar no momento em que você começa a digitar uma nova mensagem.

PADO

Labs