



EGM0017 (60H)

FLUXO E METODOLOGIAS DE PROJETO DE SISTEMAS EMBARCADOS

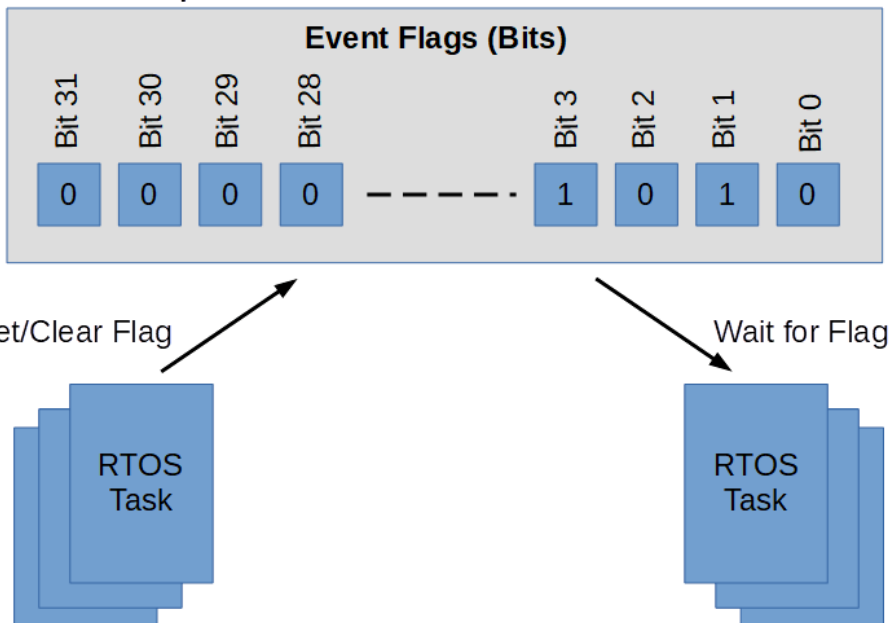
PROF. JOSENALDE OLIVEIRA

josenalde.oliveira@ufrn.br

FreeRTOS – mais sobre sincronismo de tarefas – grupo de eventos

- Task dispara ao receber dois ou mais FLAGS – ações em conjunto necessárias para disparar task
- https://github.com/josenalde/flux-embedded-design/blob/main/src/event_group.c
- **Notificações:** modo otimizado de sinalização entre TASKs (mais otimizado que semáforo)
<https://github.com/josenalde/flux-embedded-design/blob/main/src/notifications.c>

Event Group



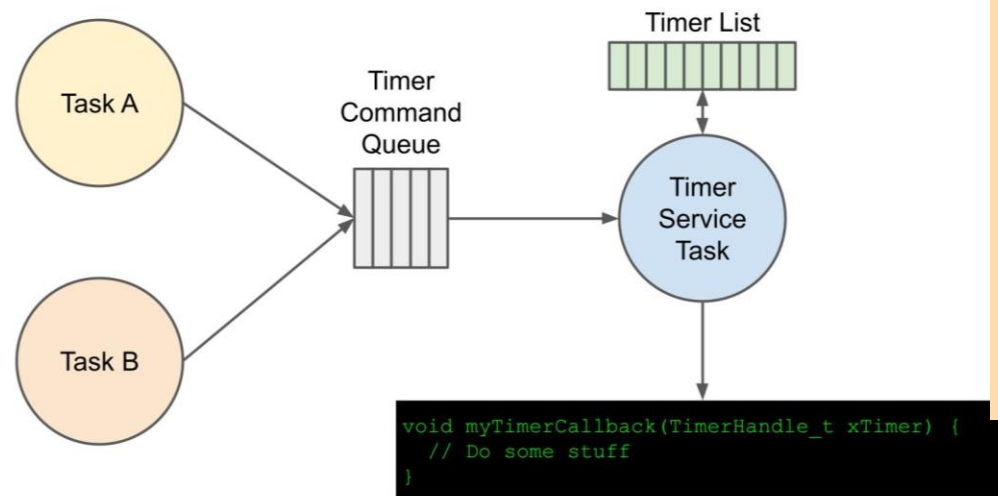
TIMERS - RTOS

- Temporizadores em sistemas embarcados são usados para atrasar a execução de funções ou executar tarefas periodicamente
- Temporizadores baseados em software são implementados com base no conceito de TICKs do RTOS
- No [FreeRTOS](#), podem ocorrer as situações:
 - `vTaskDelay` para bloquear a tarefa em execução (running) por um tempo dado em TICKS
 - Pode-se ter uma task não bloqueante usando `xTaskTickCount()` (similar ao `millis()`, `micros()` com bare metal)
 - Alguns uCs e placas possuem timers em hardware, que disparam uma ISR quando a contagem atinge o valor desejado
 - Timers em software não dependem diretamente de *hardware*, mas no FreeRTOS dependem do tick do RTOS que é função do hardware
 - No FreeRTOS existe uma [API](#) para gerenciar timers

TIMERS - RTOS

- Ao incluir a **API**, automaticamente é executada em *background* uma tarefa de serviço de temporização (TST) (*timer daemon*) separadamente de outras *tasks*. Esta *task* gerencia todos os *timers* e funções de *callback*.

Software Timers in FreeRTOS



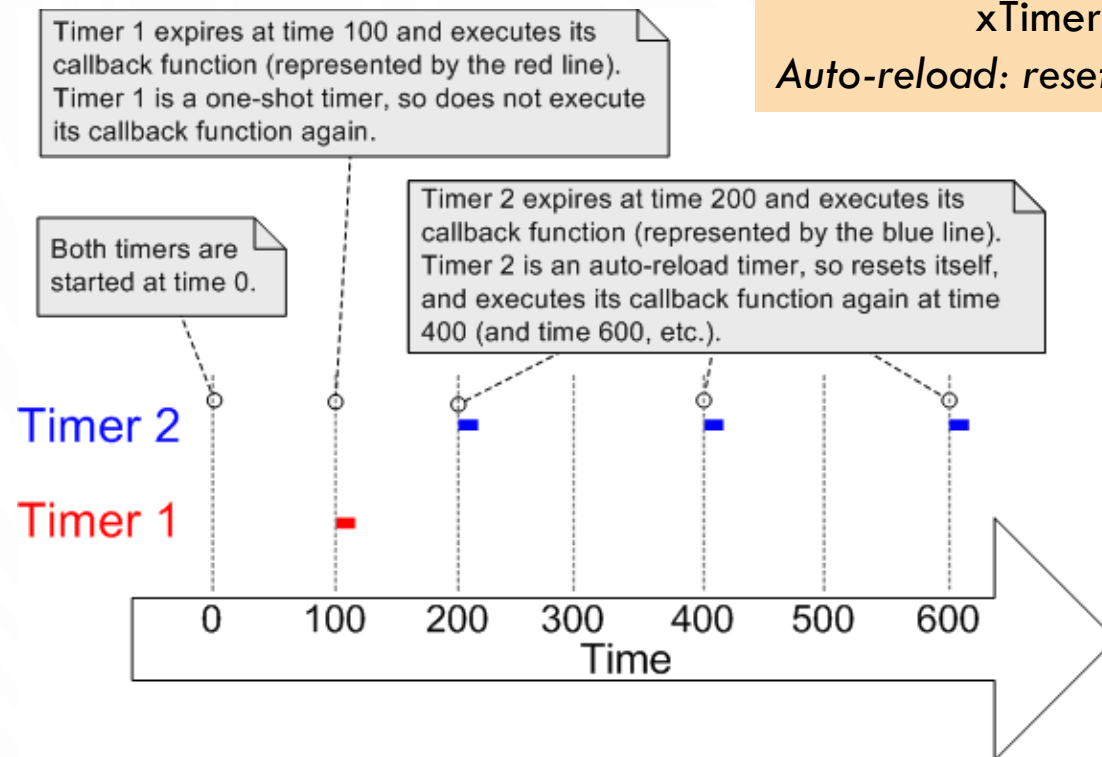
As chamadas da API se comunicam com o TST por meio de uma fila.

Ao criar um timer define-se uma *callback function* que é executada sempre que o tempo definido expira.
Visto que os *timers* dependem do tempo de *TICK*, não pode ser menor que 1 *TICK* (1 ms por padrão).

Existem 2 tipos: *one-hot-shot*: executa 1 vez após tempo expirar
auto-reload: executa periodicamente sempre que expira

TIMERS - RTOS

- One-shot versus Auto-reload



One-shot: pode ser apenas manualmente resetado

`xTimerReset()` ou `xTimerStart()` enquanto ainda está rodando...

Auto-reload: reset automático

TIMERS - RTOS

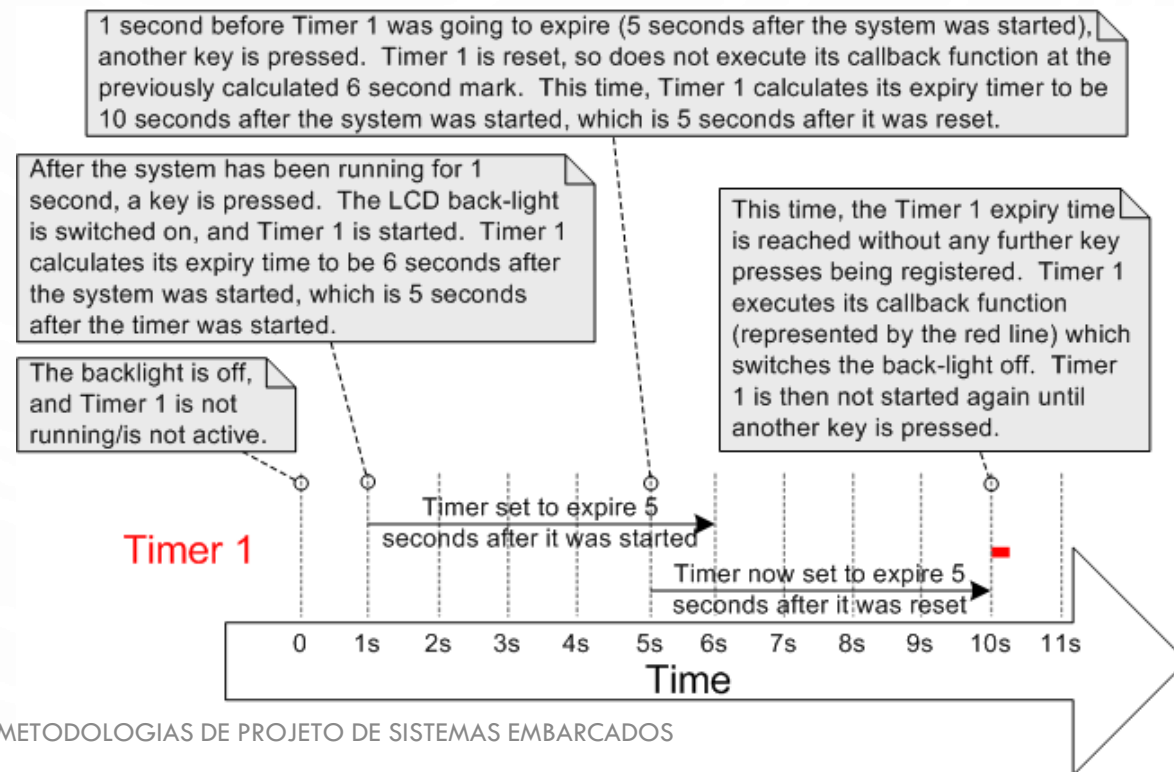
- Exemplo de projeto: sistema que lê entradas do usuário na interface Serial (teclado etc.) e exibe na tela do Serial Monitor (poderia ser um display LED, LCD etc.), mantendo o LED integrado ao ESP32 aceso (como se fosse o backlight do display). Se o usuário passar mais de 5 segundos (5000 ms) sem digitar na Serial, o LED apaga.

https://github.com/josenalde/parallel_programming_rtos/blob/main/src/timer2/timer2.ino

Funções de callback executam no contexto da TST. Portanto é essencial que a callback não tenha comandos de bloqueio. Por exemplo, não deve chamar `vTaskDelay()`, `vTaskDelayUntil()`, ou especificar um tempo não nulo para acesso a uma fila ou semáforo.

TIMERS - RTOS

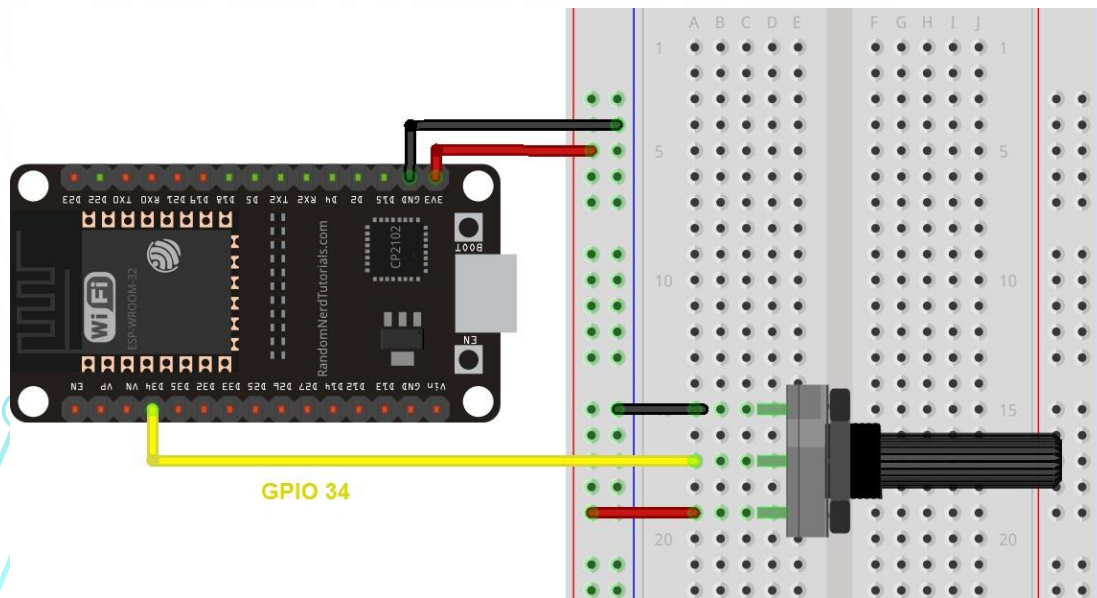
- Exemplo de projeto: sistema que lê entradas do usuário na interface Serial (teclado etc.) e exibe na tela do Serial Monitor (poderia ser um display LED, LCD etc.), mantendo o LED integrado ao ESP32 aceso (como se fosse o backlight do display). Se o usuário passar mais de 5 segundos (5000 ms) sem digitar na Serial, o LED apaga.



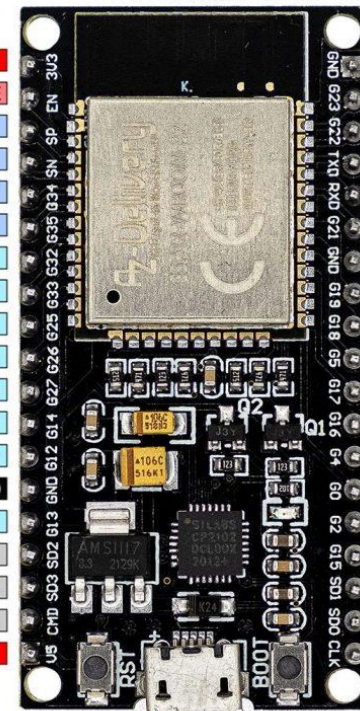
TIMERS - RTOS

- Exercício em equipe (grupo dos projetos finais):
 - Criar 3 tarefas (*taskIntReadings*, *taskIntToVoltage*, *taskPlotVoltage*) – definir núcleo e prioridades
 - Requisitos funcionais (*taskIntReadings*)
 - No ESP32 ler sinal analógico de um potenciômetro no pino 34 a cada 0.3s (300 ms). Controlar o tempo usando software timer do ESP32. Este valor inteiro entre 0 e 4095 deve ser inserido numa fila chamada **queueIntReadings**
 - Requisitos funcionais (*taskIntToVoltage*)
 - O valor inteiro da leitura do potenciômetro é consumido da fila **queueIntReadings** e é realizada a conversão para double, com a fórmula: $(\text{valor inteiro}) * (3.3 / 4096)$. Este valor double já convertido é inserido na fila **queueVoltage**
 - Requisitos funcionais (*taskPlotVoltage*)
 - O valor double da voltagem é consumido de **queueVoltage** e é exibido na interface Serial (pode ser exibido o gráfico no Serial Plotter)
 - Uma quarta tarefa chamada *taskControlled* lê esta mesma fila **queueVoltage** e, caso o valor de voltagem seja > 1.5 Volts um LED acende, caso contrário o LED apaga, simulando um controle LIGA-DESLIGA. Controlar acesso à fila **queueVoltage** com mutex

TIMERS - RTOS



RTC_IO 0		ADC1_CH0	SENS_VP	GPIO36
RTC_IO 3		ADC1_CH3	SENS_VN	GPIO39
RTC_IO 4		ADC1_CH6	VDET_1	GPIO34
RTC_IO 5		ADC1_CH7	VDET_2	GPIO35
RTC_IO 9	TOUCH_9	ADC1_CH4	XTAL_P	GPIO32
RTC_IO 8	TOUCH_8	ADC1_CH5	XTAL_N	GPIO33
RTC_IO 6		ADC2_CH8	DAC_1	GPIO25
RTC_IO 7		ADC2_CH9	DAC_2	GPIO26
RTC_IO 17	TOUCH_7	ADC2_CH7		GPIO27
RTC_IO 16	TOUCH_6	ADC2_CH6	HSPI_CLK	GPIO14
RTC_IO 15	TOUCH_5	ADC2_CH5	HSPI_MISO	GPIO12
RTC_IO 14	TOUCH_4	ADC2_CH4	HSPI_MOSI	GPIO13
		SPI_HD	SD_2	GPIO9
		SPI_WP	SD_3	GPIO10
		CMD		GPIO11



GND	GPIO23	VSPI_MOSI	
	GPIO22	VSPI_WP	SCL
	GPIO1	TX_0	
	GPIO3	RX_0	
	GPIO21	VSPI_HD	SDA
GND	GPIO19	VSPI_MISO	
	GPIO18	VSPI_CLK	
	GPIO5	VSPI_CS	
	GPIO17	TX_2	
	GPIO16	RX_2	
	GPIO4	HSPI_HD	ADC2_CH0 TOUCH_0 RTC_IO 10
	GPIO0		ADC2_CH1 TOUCH_1 RTC_IO 12
	GPIO2	HSPI_WD	ADC2_CH2 TOUCH_2 RTC_IO 13
	GPIO15	HSPI_CS	ADC2_CH3
	GPIO8	SD_1	SPI_MOSI
	GPIO7	SD_0	SPI_MISO
	GPIO6	CLK	SPI_CLK