CMSIS-RTOS API v2
e RTOS Keil RTX 5 (continuação)

Prof. Hugo Vieira Neto

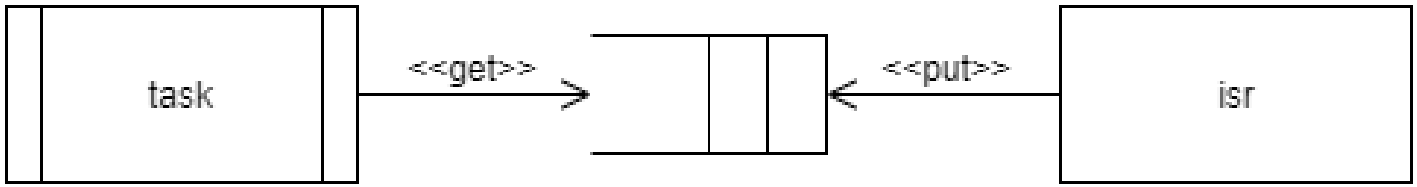
2020/2

Filas de Mensagens

- Troca de mensagens é um modelo básico de comunicação entre tarefas em que dados são enviados e recebidos explicitamente
  - Os dados são trocados por meio de uma estrutura do tipo FIFO (first-in-first-out)
  - A operação assemelha-se a acessos a um periférico de E/S, em vez de acessos diretos a memória compartilhada

Filas de Mensagens

- Funções de gerenciamento de mensagens do RTOS Keil RTX 5 permitem controlar, enviar, receber ou esperar mensagens
- O tipo de dados a ser passado pode ser um inteiro ou um ponteiro (32 bits)



Gerenciamento de Mensagens

```
osMessageQueueId_t osMessageQueueNew
(uint32_t msg_count, uint32_t msg_size,
const osMessageQueueAttr_t *attr)
```

- Cria e inicializa uma fila de mensagens

```
osStatus_t osMessageQueueDelete
(osMessageQueueId_t mq_id)
```

- Deleta uma fila de mensagens

Gerenciamento de Mensagens

```
osStatus_t osMessageQueuePut
(osMessageQueueId_t mq_id, const void
*msg_ptr, uint8_t msg_prio, uint32_t
timeout)
```

- Coloca uma mensagem em uma fila (se estiver cheia, aguarda por um determinado limite de tempo)*

```
osStatus_t osMessageQueueGet
(osMessageQueueId_t mq_id, void *msg_ptr,
uint8_t *msg_prio, uint32_t timeout)
```

- Retira uma mensagem de uma fila (se estiver vazia, aguarda por um determinado limite de tempo)*

Gerenciamento de Mensagens

```
uint32_t osMessageQueueGetCount
(osMessageQueueId_t mq_id)
```

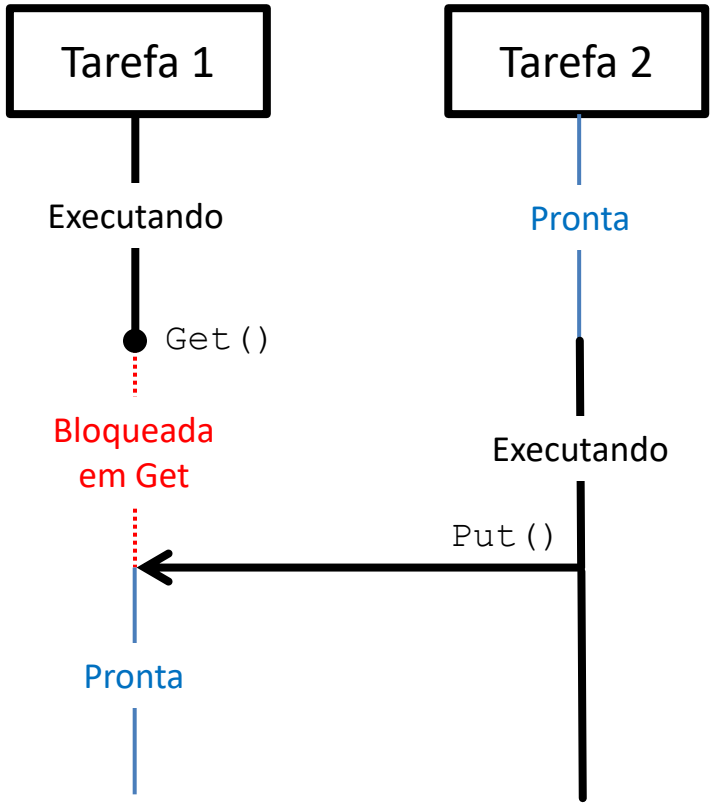
- Obtém o número de mensagens enfileiradas numa fila*

```
uint32_t osMessageQueueGetSpace
(osMessageQueueId_t mq_id)
```

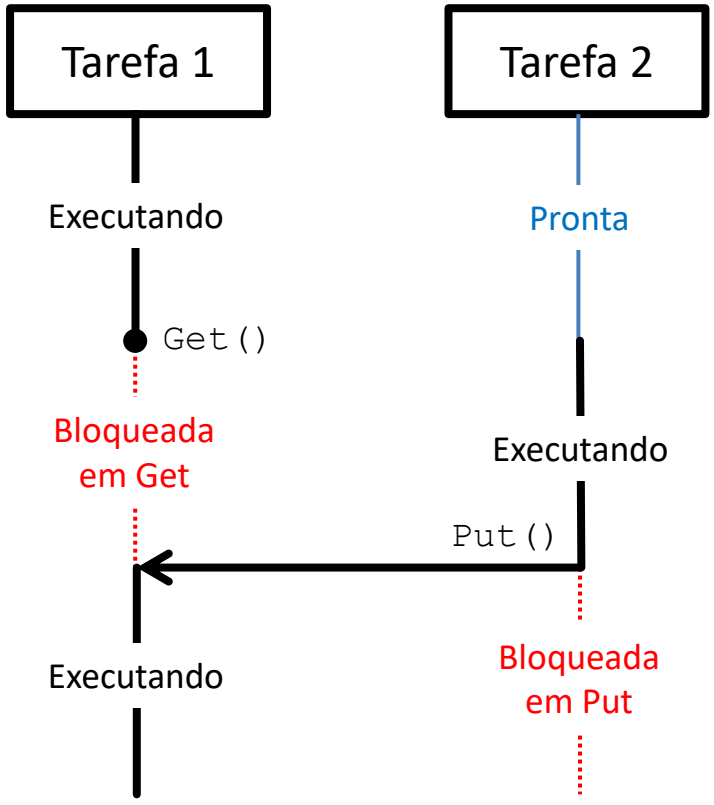
- Obtém o número de elementos vagos para mensagens em uma fila*

Sincronização por Mensagens

Método Assíncrono:



Método Síncrono:



Sincronização por Mensagens

- No RTOS Keil RTX 5:
  - Fila de mensagens com tamanho grande → método *quase* assíncrono (tarefa emissora tem grande probabilidade de nunca bloquear)
  - Fila de mensagens com tamanho igual a 1 → método *quase* síncrono (tarefa emissora bloqueia na segunda tentativa de colocar mensagem)

Exemplo de Uso (Mensagens)

- Ver código-exemplo no material de referência do CMSIS-RTOS v2 no website da Keil:
  - http://www.keil.com/pack/doc/cmsis/RTOS2/html/group__CMSIS__RTOS__Message.html

Exercício 7

- Altere o código do projeto “prodcons” do Exercício 5 (modelo produtor-consumidor) para fazer uso de uma fila de mensagens para a troca de informações entre as tarefas, mantendo a funcionalidade anteriormente especificada.
- Qual das duas formas de comunicação entre tarefas (semáforos ou fila de mensagens) é a mais prática de ser utilizada?

Sinalizadores de Tarefa

- Toda tarefa possui um conjunto próprio de 32 sinalizadores (thread flags), que podem ser utilizados para sincronização individual (tarefa-tarefa, tarefa-ISR)
- Para sincronização simultânea de múltiplas tarefas existem objetos sinalizadores de eventos (event flags)

Gerenciamento de Sinalizadores de Tarefa

`uint32_t` **`osThreadFlagsSet`**`(osThreadId_t thread_id, uint32_t flags)`

- Ativa sinalizador(es) da tarefa*

`uint32_t` **`osThreadFlagsWait`**`(uint32_t flags, uint32_t options, uint32_t timeout)`

- Aguarda a ativação de um ou mais sinalizadores da tarefa corrente dentro de um limite de tempo

- Opções de comportamento:
  - `osFlagsWaitAny` (padrão)
  - `osFlagsWaitAll`
  - `osFlagsNoClear`

Exemplo de Uso (Sinalizadores de Tarefa)

```
#include "system_tm4c1294.h" // CMSIS-Core
#include "driverleds.h" // device drivers
#include "cmsis_os2.h" // CMSIS-RTOS

osThreadId_t AcionaLED_id, Temporiza_id;

void AcionaLED(void *arg){
  uint8_t led = (uint32_t)arg;
  uint8_t state = 0;

  while(1){
    osThreadFlagsWait(0x0001, osFlagsWaitAny,
                              osWaitForever);
    state ^= led;
    LEDWrite(led, state);
  } // while
} // AcionaLED
```

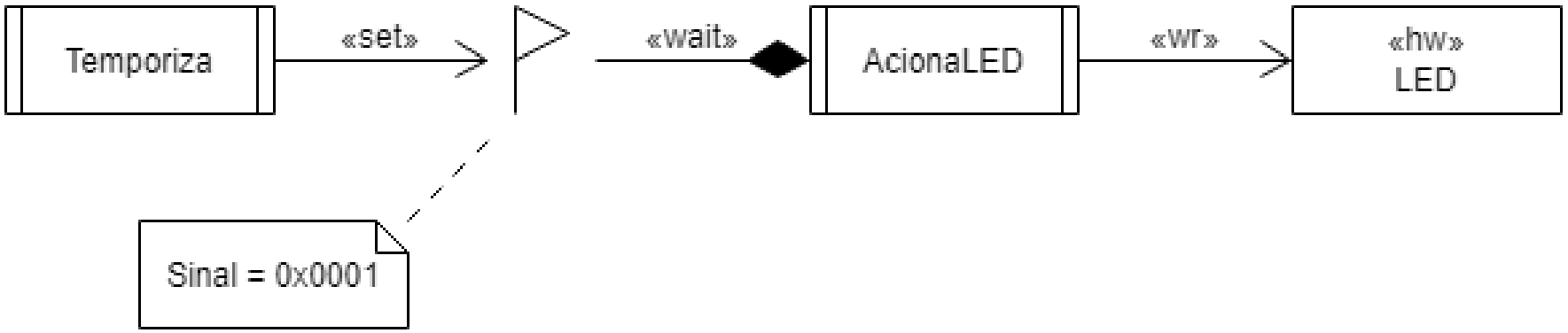
Exemplo de Uso (Sinalizadores de Tarefa)

```
void Temporiza(void *arg){
  uint32_t delay = (uint32_t)arg;
  uint32_t tick;

  while(1){
    tick = osKernelGetTickCount();

    osThreadFlagsSet(AcionaLED_id, 0x0001);

    osDelayUntil(tick + delay);
  } // while
} // Temporiza
```

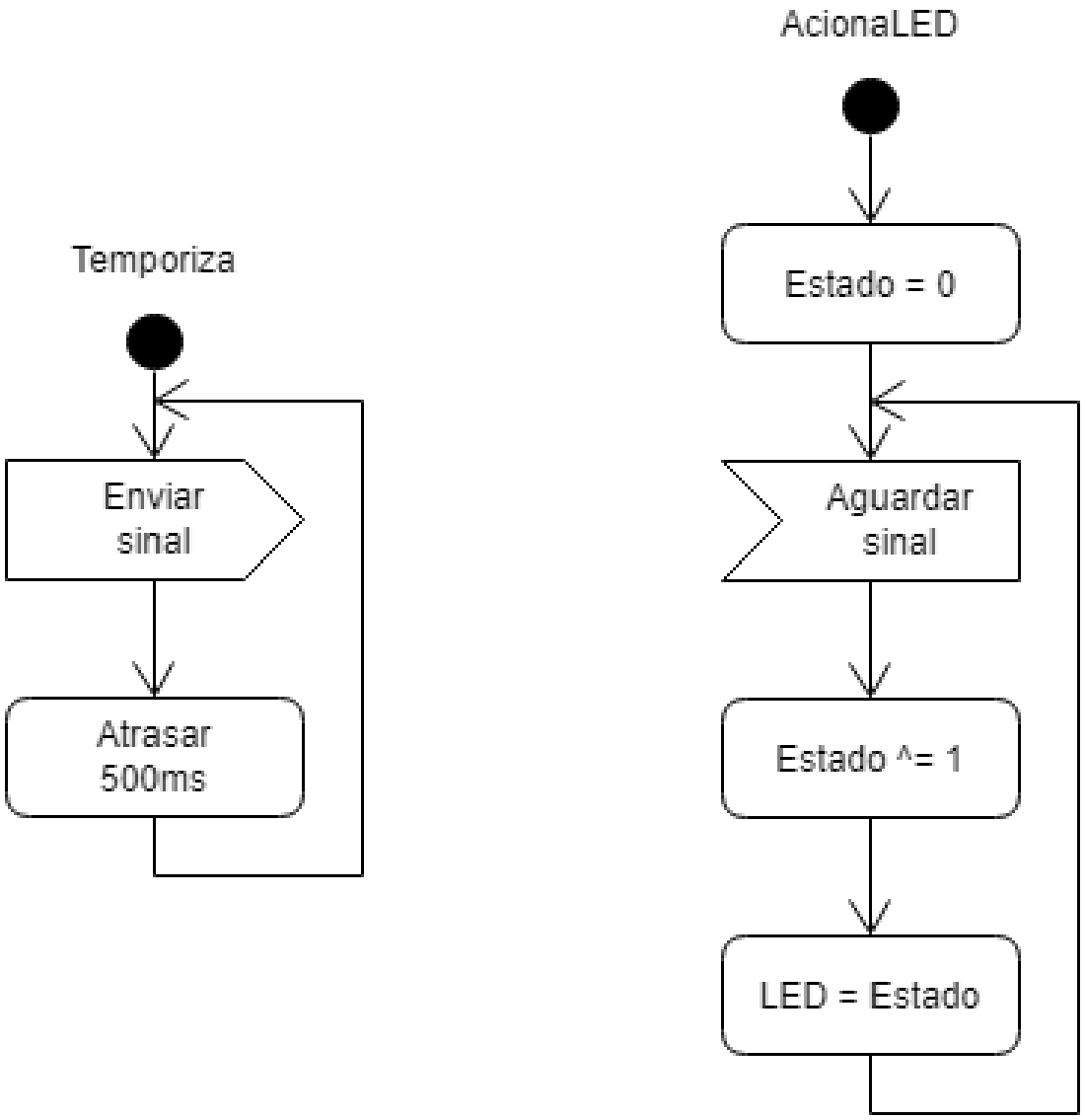
Exemplo de Uso (Sinalizadores de Tarefa)

```
void main(void){
  LEDInit(LED1);

  osKernelInitialize();

  AcionaLED_id = osThreadNew(AcionaLED, (void *)LED1,
                                                NULL);
  Temporiza_id = osThreadNew(Temporiza, (void *)500,
                                                NULL);
  if(osKernelGetState() == osKernelReady)
    osKernelStart();

  while(1);
} // main
```

Diagrama de Objetos



Diagramas de Atividades



Exercício 8

- Abra o projeto “sinalizador” a partir da área de trabalho “EK-TM4C1294_RTOS_IAR8”.
- Analise a implementação e conclua:
  - Há desacoplamento entre as atividades de temporização e acionamento do LED?
  - Explique o que ocorre se as opções de espera pela ativação do sinalizador forem alteradas para `osFlagsWaitAny|osFlagsNoClear`.

Delegação de ISR a uma Tarefa

- Uma boa prática em sistemas embarcados operando em tempo real consiste em manter as rotinas de tratamento de interrupção tão breves quanto possível
- Quando o tratamento necessário é mais longo, costuma-se delegá-lo a uma tarefa de alta prioridade, desbloqueando-a por meio de um sinalizador de tarefa (ou algum outro mecanismo de sincronização)

Delegação de ISR a uma Tarefa

Rotina de tratamento (ISR)

```
void handler(){
  ClearInterruptFlag();
  ... // proc. mínimo
  osThreadFlagsSet(thread_id,
                   0x0001);
} // handler
```

Tarefa de alta prioridade

```
void thread(void *arg){
  while(1){
    osThreadFlagsWait(0x0001,
                      osFlagsWaitAny,
                      osWaitForever);
    ... // proc. adicional
  } // while
} // thread
```

Delegação de interrupção = *Interrupt deferral*

Exercício 9

- Abra o projeto “blinky” a partir da área de trabalho “EK-TM4C1294_RTOS_IAR8”.
- Analise o código-fonte no arquivo `blinky.c` quanto ao uso de mutexes e sinalizadores para sincronização entre tarefas.
- Elabore um diagrama de objetos que descreva a arquitetura desse projeto.

Exercício 9

- Classifique as tarefas do projeto “blinky” em:
  - Não periódicas
  - Puramente periódicas
  - Periódicas com sincronização
- Qual é o principal motivo da existência da tarefa `app_main`?
- Quem inicia a sequência de troca de sinais entre phaseA, phaseB, phaseC e phaseD?

Exercício 9

- Tendo como base o projeto “blinky”, implemente um sistema de quatro tarefas sincronizadas por sinalizadores que apresente uma contagem binária crescente nos quatro LEDs (LED D1 → LSB) do kit EK-TM4C1294XL a cada 500ms.
- Elabore um diagrama de objetos que descreva a arquitetura desse sistema.

Exercício 9

- Cada tarefa deverá ser responsável pelo acionamento de um único LED do kit.
- O que deve ser alterado na implementação do sistema para se obter uma contagem binária decrescente nos quatro LEDs do kit?
- Seria possível executar uma contagem binária utilizando quatro instâncias de uma mesma tarefa nesse sistema?

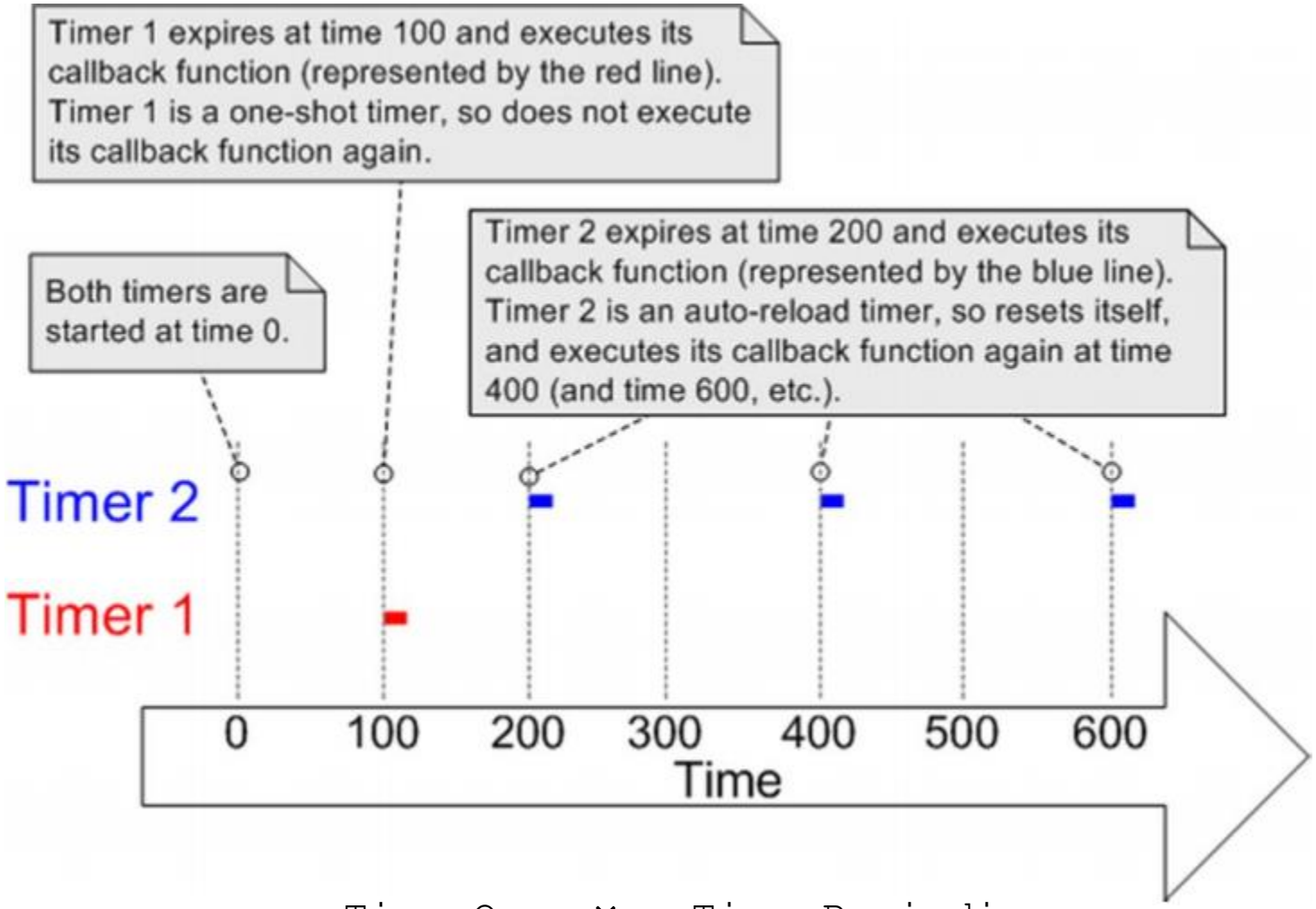
Temporizadores de Software

- Temporizadores de software utilizam o tique do RTOS para chamar uma função depois de decorrido um determinado período de tempo
- Não consomem temporizadores de hardware mas não possuem resolução de tempo tão elevada quanto aqueles
- São gerenciados por uma tarefa específica (`osRtxTimerThread`) com prioridade alta (reconfigurável)

Temporizadores de Software

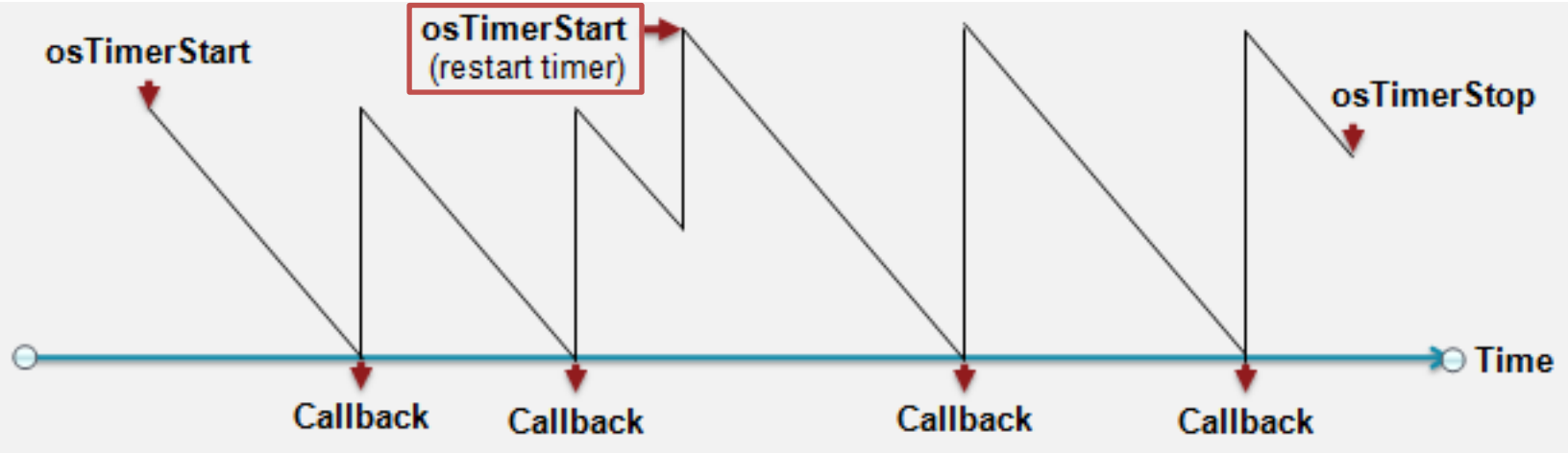
- One-shot (`osTimerOnce`):
  - Conta um período de tempo e para a contagem
  - Ao final da contagem provoca a chamada da função de callback
- Auto-reload (`osTimerPeriodic`):
  - Conta um período de tempo e reinicia a contagem
  - Ao final de cada contagem provoca a chamada da função de callback

One-shot × Auto-reload



`osTimerOnce` × `osTimerPeriodic`

Temporizador Periódico



Behavior of a Periodic Timer

- `osTimerStart` define o valor inicial e inicia uma contagem regressiva
- Quando a contagem atinge zero, é chamada a função de *callback*
- `osTimerStop` para a contagem regressiva

Gerenciamento de Temporizadores

```
osTimerId_t osTimerNew(osTimerFunc_t func,
osTimerType_t type, void *argument, const
osTimerAttr_t *attr)
```

- Cria e inicializa um temporizador

```
osStatus_t osTimerDelete(osTimerId_t
timer_id)
```

- Deleta um temporizador

Gerenciamento de Temporizadores

```
osStatus_t osTimerStart(osTimerId_t
timer_id, uint32_t ticks)
```

- Inicia ou reinicia a contagem de tempo

```
osStatus_t osTimerStop(osTimerId_t timer_id)
```

- Para a contagem de tempo

```
uint32_t osTimerIsRunning(osTimerId_t
timer_id)
```

- Verifica se a contagem de tempo está ativa

Estrutura de uma Função de Callback

```
void callback(void *arg){
  ... // atividades da função de callback
} // callback
```

- É possível passar um único argumento por referência à função (`osTimerNew`)
  - Útil para definir comportamentos diferentes para uma mesma função para vários temporizadores
- Deve poder ser chamada a partir de uma ISR (não pode causar bloqueio)

Exemplo de Uso (Temporizador)

```
#include "system_tm4c1294.h" // CMSIS-Core
#include "driverleds.h" // device drivers
#include "cmsis_os2.h" // CMSIS-RTOS

uint8_t state = 0;
osThreadId_t app_main_id;
osTimerId_t timer1_id;

void callback(void *arg){
  state ^= LED1;
  LEDWrite(LED1, state);
} // callback

void app_main(void *arg){
  osTimerStart(timer1_id, 100);
  osDelay(osWaitForever);
} // app_main
```

Exemplo de Uso (Temporizador)

```
void main(void){
  LEDInit(LED1);

  osKernelInitialize();

  app_main_id = osThreadNew(app_main, NULL, NULL);
  timer1_id = osTimerNew(callback, osTimerPeriodic, NULL,
                                                     NULL);
  if(osKernelGetState() == osKernelReady)
    osKernelStart();

  while(1);
} // main
```

Exercício 10

- Abra o projeto “temporizador” a partir da área de trabalho “EK-TM4C1294_RTOS_IAR8”.
- Localize o arquivo `RTX_Config.h` na lista de dependências do arquivo `rtx_lib.c` e analise-o. Quais são as configurações para:
  - `OS_TIMER_NUM`?
  - `OS_TIMER_THREAD_PRIO`?

Exercício 10

- Modifique o código-fonte no arquivo `temporizador.c` para que seja possível informar o LED a ser acionado à função de callback.
- Crie quatro temporizadores, um para cada LED do kit, com períodos diferentes, mas compartilhando uma mesma função de callback: LED D1 → 100ms, LED D2 → 150ms, LED D3 → 250ms, LED D4 → 350ms.

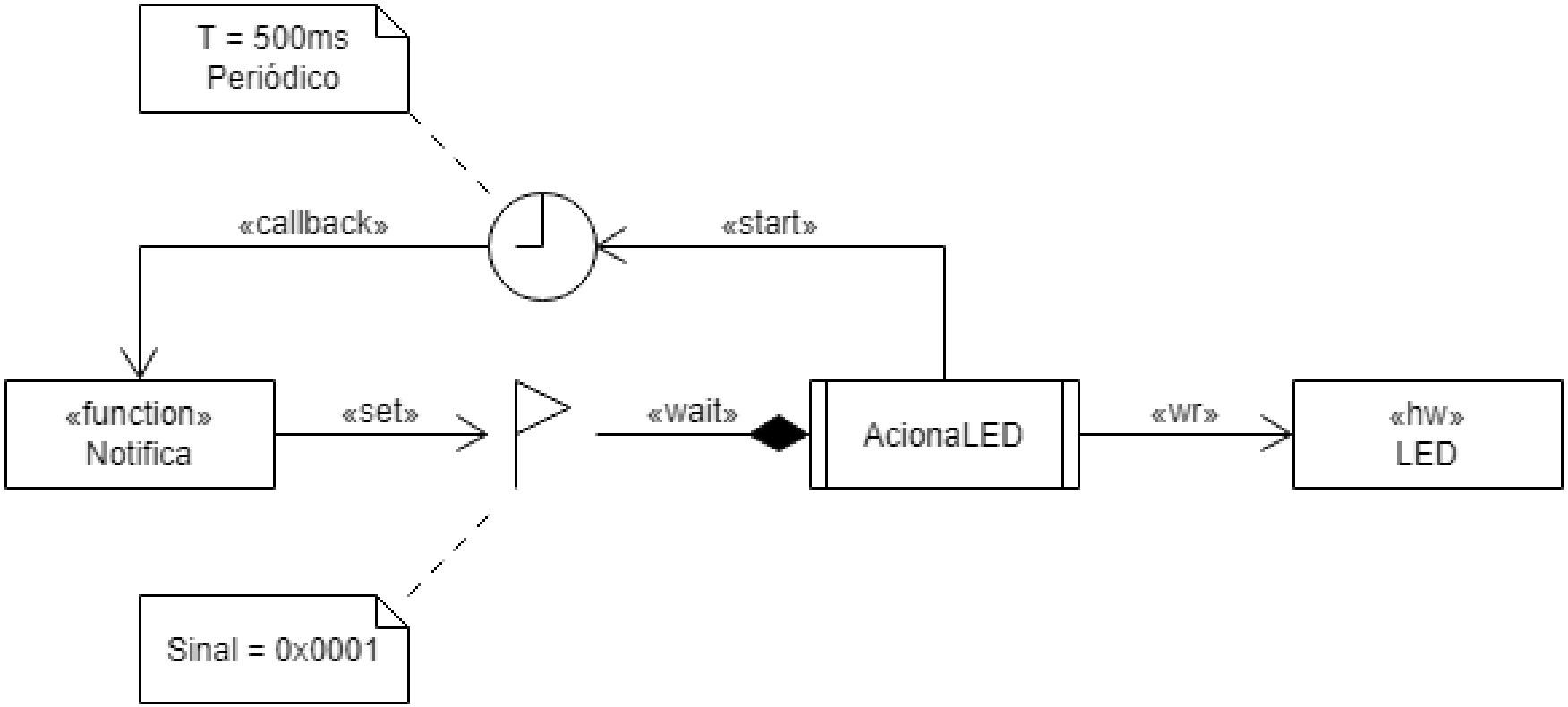
Exercício 10

- Por que é necessário criar uma tarefa com o propósito único de disparar o temporizador (`app_main`)? Não seria possível dispará-lo na própria função `main`?
- A tarefa `app_main` pode ser considerada periódica? Por quê?
- Quais são as diferenças entre a tarefa que aciona o LED no Exercício 5 e a função de callback do Exercício 10?

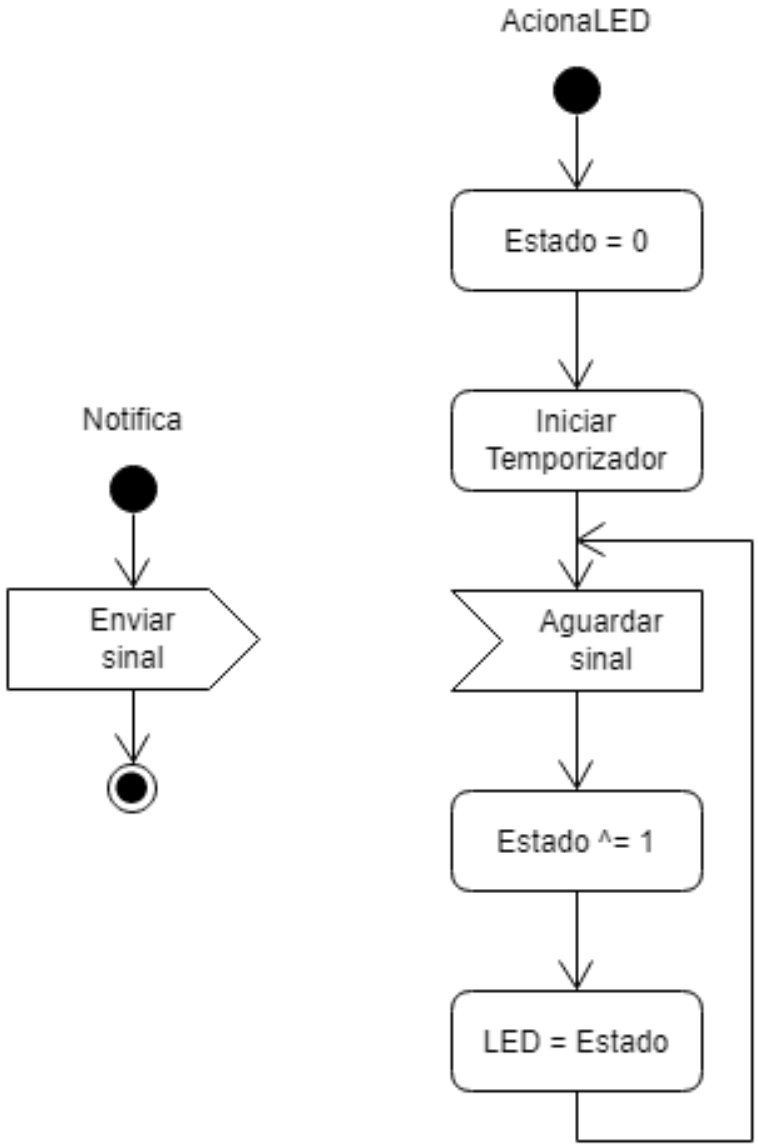
Exercício 11

- Tendo como base os projetos “tarefas”, “temporizador” e “sinalizador”, implemente uma tarefa periódica sincronizada com um temporizador de software por meio de um sinalizador de tarefa. A tarefa deverá acionar o LED D4 do kit EK-TM4C1294XL em intervalos de 500ms, conforme diagramas de objetos e de atividades apresentados a seguir.

Diagrama de Objetos



Diagramas de Atividades



Recursos Avançados do RTX 5

- Pesquisar na documentação:
  - Sinalizadores de eventos (event flags)
  - Pools de memória para compartilhamento
  - Tarefa `osRtxIdleThread`
  - Função `osRtxErrorNotify`
  - Estruturas de atributos dos diferentes tipos de objeto do RTOS