UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Documentação de Projeto – Parte 2
Design, Estudo da Plataforma

Projeto: Sistema de Elevadores

Autor: Mateus Vieira Freitas

Versão: 14-Dez-2021

Parte 2a – Design

1 Introdução

Este documento tem por objetivo elaborar a arquitetura do sistema descrito no documento anterior.

2 Arquitetura Funcional

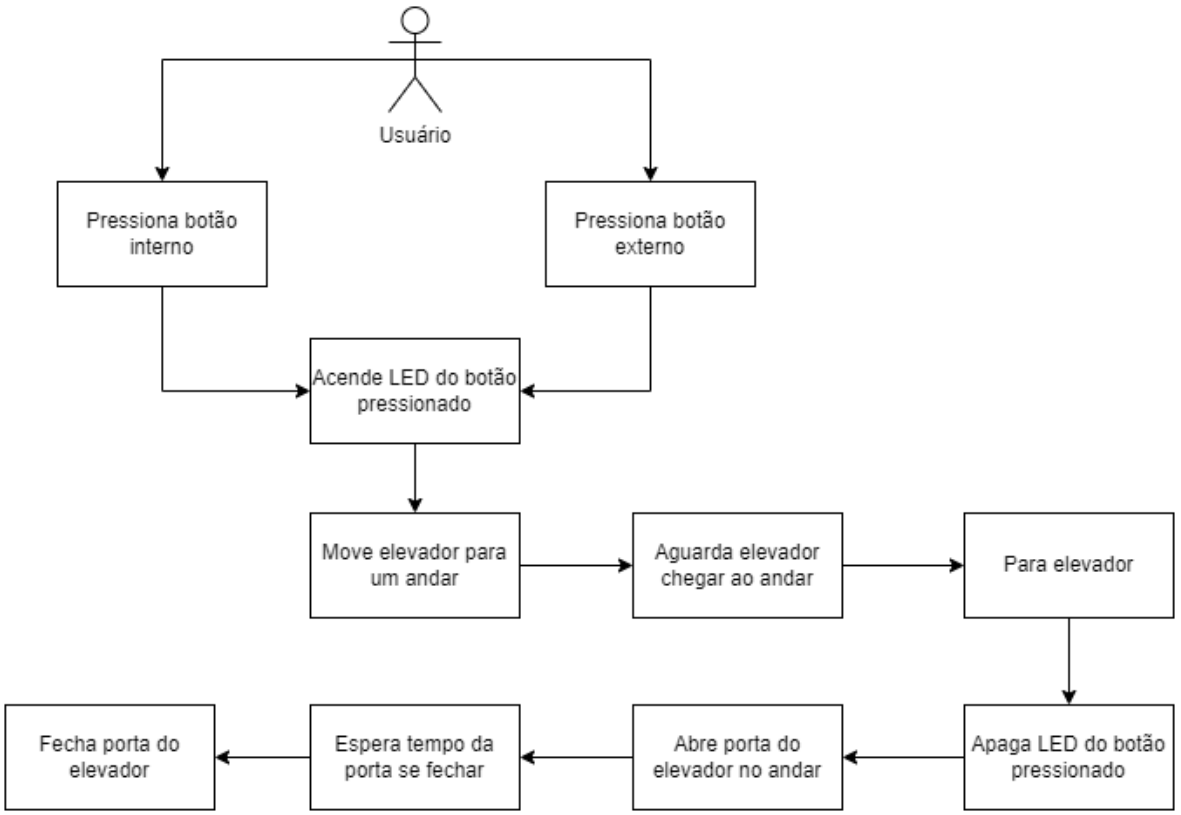


Figura 1: Diagrama da arquitetura funcional do sistema

O usuário do sistema de elevadores utiliza botões internos aos elevadores para direcionar o destino destes, requisitando que o elevador vá para algum andar específico; e botões externos, presentes nos andares, para requisitar um elevador (no caso, um par de botões externos por elevador). As rotas feitas pelos elevadores são tratadas, como já mencionado no documento anterior, e serão melhor explicadas ao longo deste documento.

3 Arquitetura Física

Na parte física, tem-se uma fonte de alimentação que fornece energia para o computador, onde o usuário utiliza o simulador, e para a placa, já que esta está conectada ao computador. No simulador, é possível o pressionamento dos botões dos elevadores (internos e externos), os quais gerarão as requisições à comunicação serial da placa. Com isso, o controlador do sistema consegue inserir as requisições em uma fila para ordenar os andares em que o elevador irá passar. Assim, a resposta é mostrada visualmente no simulador, com os elevadores se deslocando. O sistema contará com 3 threads, conforme requisitado, e a comunicação serial será pelo protocolo UART.

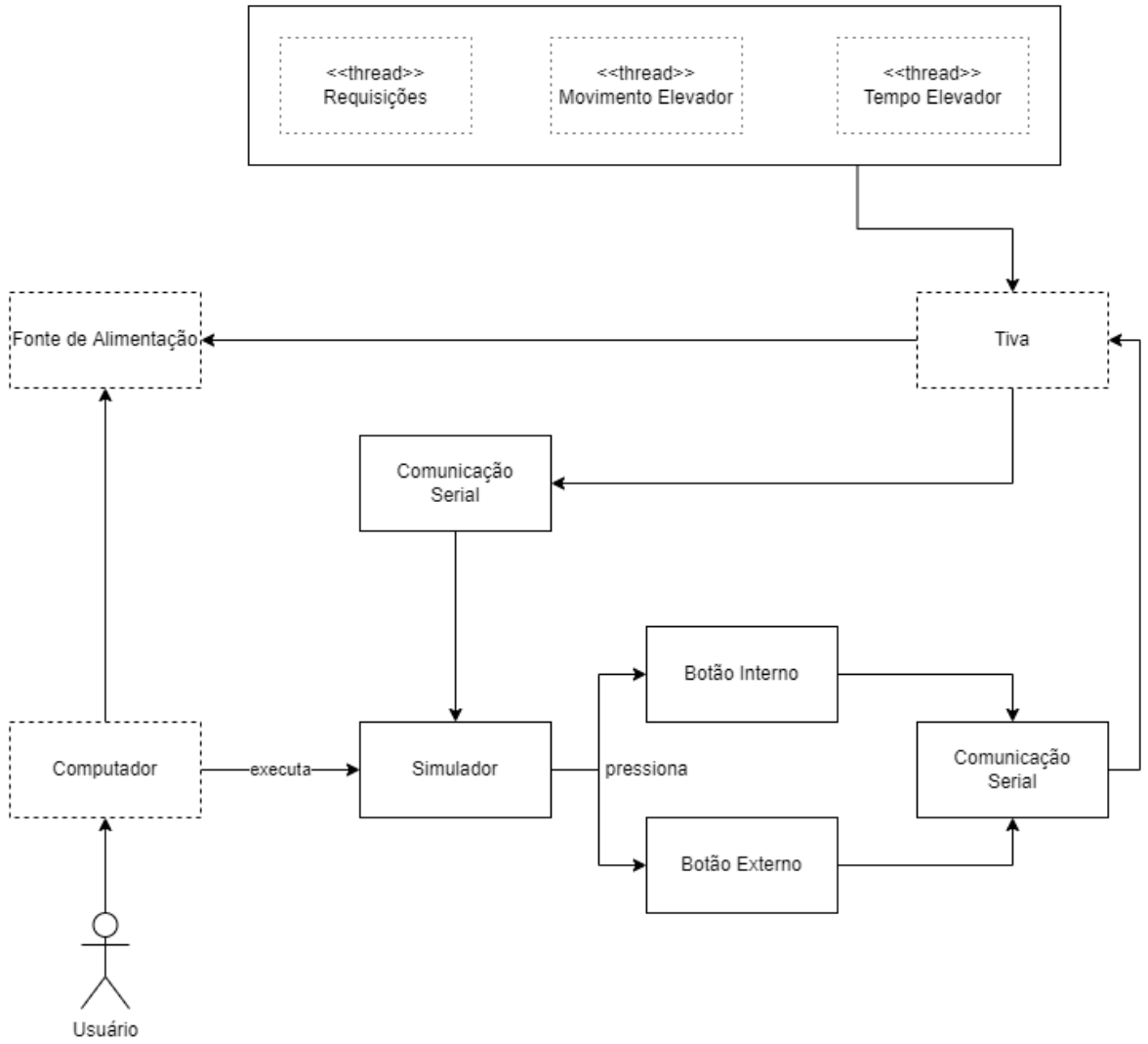


Figura 2: Diagrama do relacionamento entre hardware e software

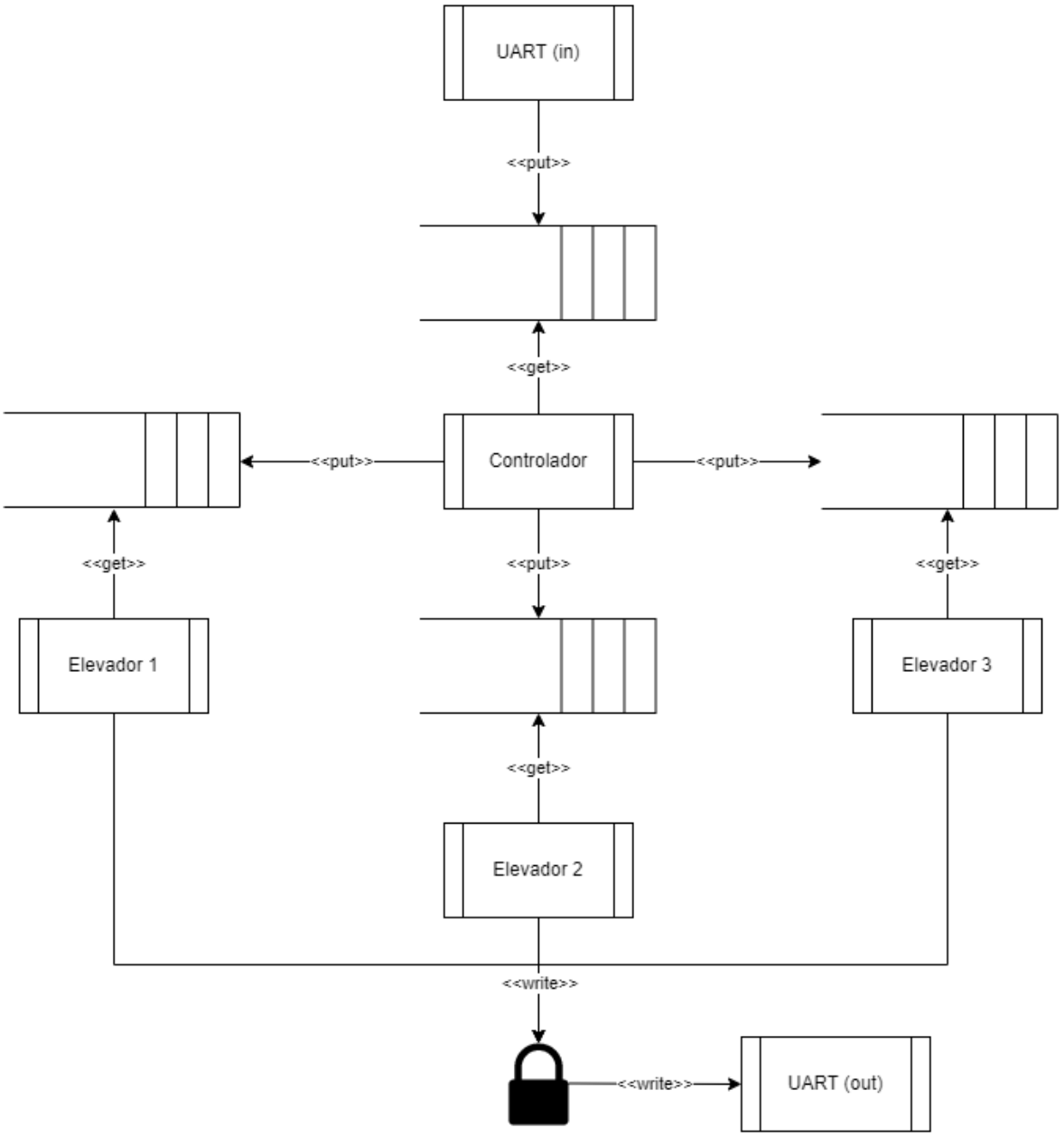


Figura 3: Diagrama da arquitetura do sistema

4 Interface com o Usuário

Como o sistema de elevadores será desenvolvido com o auxílio de um simulador (apresentado na Figura 3 abaixo), a interface com o usuário se consiste nos botões presentes na simulação, representando os botões externos e internos dos elevadores (Figuras 5 e 6).

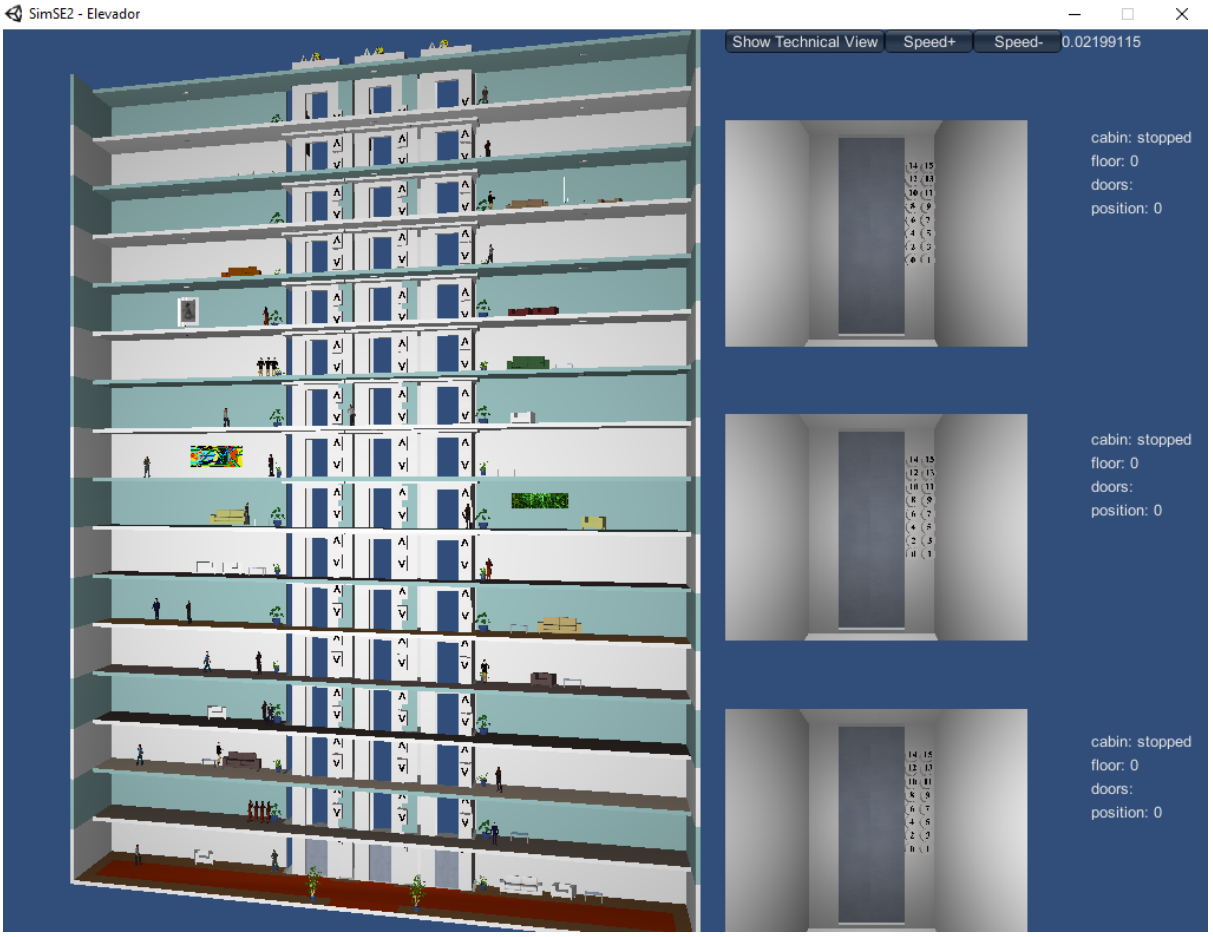
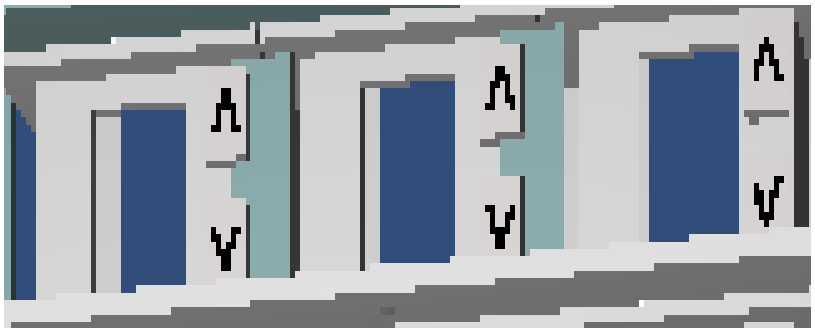


Figura 4: Tela do simulador



Figuras 5 e 6: Botões internos e externos dos elevadores no simulador

5 Mapeamento da Arquitetura Funcional à Arquitetura Física

As funcionalidades do sistema devem ser implementadas por componentes da arquitetura física, conforme o mapeamento da tabela abaixo.

	Pressiona botão externo	Pressiona botão interno	Acende LED do botão	Move elevador para andar	Aguarda elevador chegar ao andar	Para elevador	Apaga LED do botão	Abre porta do elevador no andar	Espera tempo da porta se fechar	Fecha porta do elevador
Simulador	x	x								
Elevador			x	x	x	x	x			
Tempo Elevador								x	x	x

6 Arquitetura do Hardware

A placa Tiva TM4C1294 da Texas Instruments inclui todos os elementos de hardware necessários para o projeto, isso porque o sistema de elevadores utilizará um simulador para seu funcionamento. A alimentação da placa se dará por meio da conexão desta com um computador, através da porta Micro USB.

7 Design Detalhado

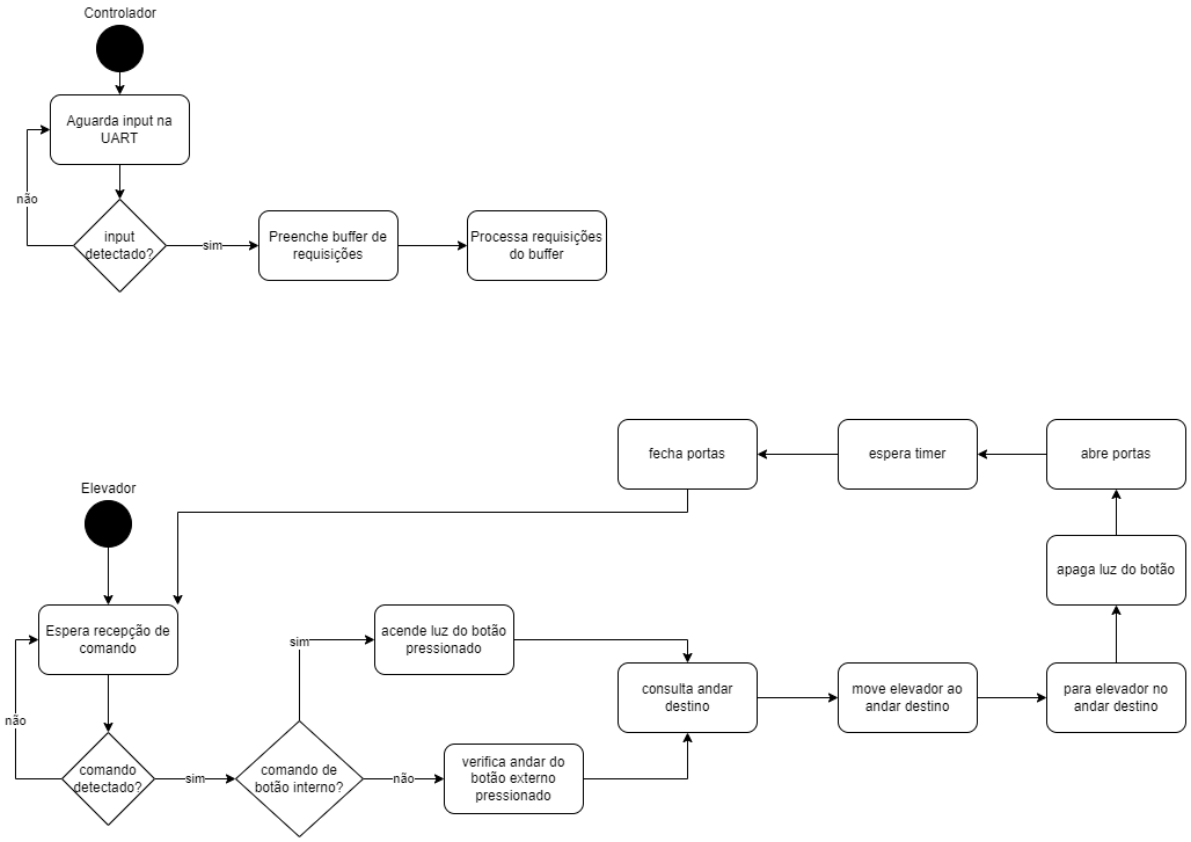


Figura 7 : Design do sistema

Parte 2b – Estudo da Plataforma

1 TivaWare SDK

O SDK da TivaWare fornece um conjunto de bibliotecas que incluem biblioteca de drivers para os periféricos da placa Tiva, biblioteca USB, além de diversas APIs de fácil uso para simplificar o desenvolvimento de software. Sendo assim, a TivaWare será de bastante utilidade para o sistema de elevadores a ser implementado, pois a biblioteca de drivers permitirá o uso dos módulos de GPIO e de controle de interrupções, para controle das interrupções dos botões, por exemplo. Além disso, a TivaWare também permitirá o controle e manipulação da interface serial da placa Tiva, visando a comunicação do simulador com o microcontrolador.

GPIO

14.2 API Functions

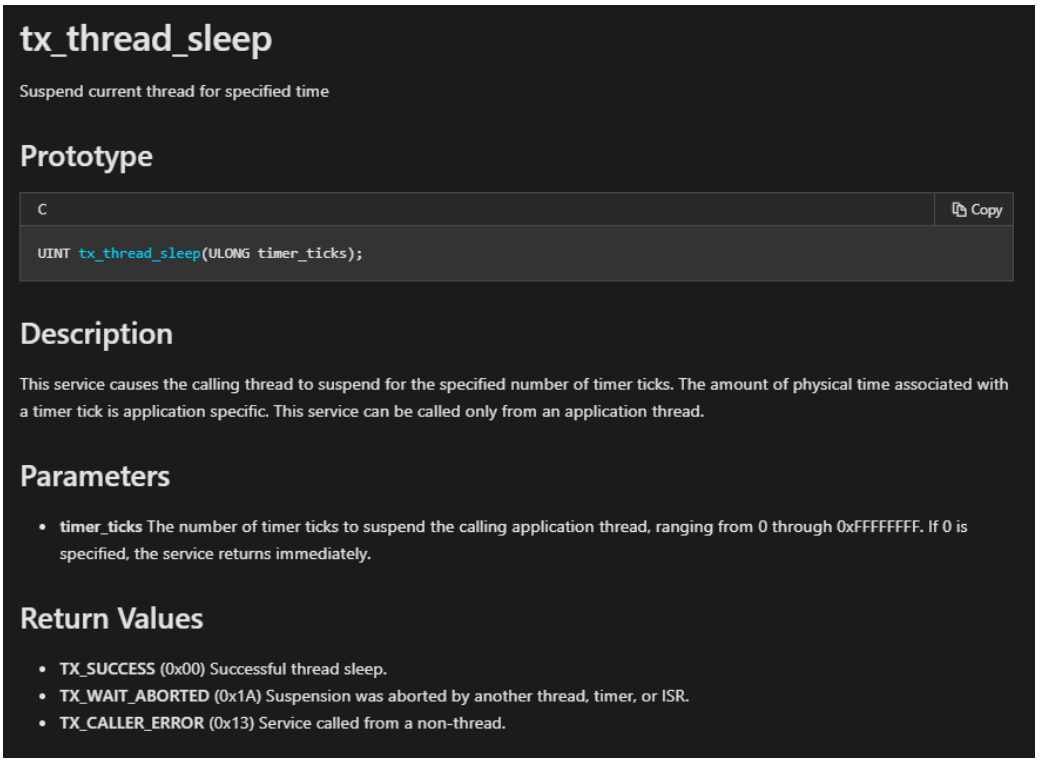
Functions

- void GPIOADCTriggerDisable (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOADCTriggerEnable (uint32_t ui32Port, uint8_t ui8Pins)
- uint32_t GPIODirModeGet (uint32_t ui32Port, uint8_t ui8Pin)
- void GPIODirModeSet (uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32PinIO)
- void GPIODMATriggerDisable (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIODMATriggerEnable (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOIntClear (uint32_t ui32Port, uint32_t ui32IntFlags)
- void GPIOIntDisable (uint32_t ui32Port, uint32_t ui32IntFlags)
- void GPIOIntEnable (uint32_t ui32Port, uint32_t ui32IntFlags)
- void GPIOIntRegister (uint32_t ui32Port, void (*pfnIntHandler)(void))
- void GPIOIntRegisterPin (uint32_t ui32Port, uint32_t ui32Pin, void (*pfnIntHandler)(void))
- uint32_t GPIOIntStatus (uint32_t ui32Port, bool bMasked)
- uint32_t GPIOIntTypeGet (uint32_t ui32Port, uint8_t ui8Pin)
- void GPIOIntTypeSet (uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32IntType)
- void GPIOIntUnregister (uint32_t ui32Port)
- void GPIOIntUnregisterPin (uint32_t ui32Port, uint32_t ui32Pin)
- void GPIOPadConfigGet (uint32_t ui32Port, uint8_t ui8Pin, uint32_t *pui32Strength, uint32_t *pui32PinType)
- void GPIOPadConfigSet (uint32_t ui32Port, uint8_t ui8Pins, uint32_t ui32Strength, uint32_t ui32PinType)
- void GPIOPinConfigure (uint32_t ui32PinConfig)
- int32_t GPIOPinRead (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeADC (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeCAN (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeComparator (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeComparatorOutput (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeDIVSCLK (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeEPI (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeEthernetLED (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeEthernetMII (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeGPIOInput (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeGPIOOutput (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeGPIOOutputOD (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeHibernateRTCCLK (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeI2C (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeI2CSCL (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeLCD (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypeOneWire (uint32_t ui32Port, uint8_t ui8Pins)
- void GPIOPinTypePWM (uint32_t ui32Port, uint8_t ui8Pins)

Figura 8 : Funções de GPIO no manual da TivaWare

2 ThreadX

O sistema utilizará uma thread relacionada ao tratamento de eventos dos botões pressionados no simulador e outra para a lógica do sistema de elevadores em si, incluindo atendimento aos chamados nos andares, movimentação dos elevadores, e todas as demais funcionalidades dos elevadores. A thread de eventos escreve em uma fila de mensagens se algum botão foi pressionado e a thread do elevador lê essas mensagens para calcular a próxima ação dos elevadores. A documentação do ThreadX se encontra no guia de usuário, disponível online.



tx_thread_sleep

Suspend current thread for specified time

Prototype

C Copy

```
UINT tx_thread_sleep(ULONG timer_ticks);
```

Description

This service causes the calling thread to suspend for the specified number of timer ticks. The amount of physical time associated with a timer tick is application specific. This service can be called only from an application thread.

Parameters

- **timer_ticks** The number of timer ticks to suspend the calling application thread, ranging from 0 through 0xFFFFFFFF. If 0 is specified, the service returns immediately.

Return Values

- **TX_SUCCESS** (0x00) Successful thread sleep.
- **TX_WAIT_ABORTED** (0x1A) Suspension was aborted by another thread, timer, or ISR.
- **TX_CALLER_ERROR** (0x13) Service called from a non-thread.

tx_queue_send

Send message to message queue

Prototype

C Copy

```
UINT tx_queue_send(
    TX_QUEUE *queue_ptr,
    VOID *source_ptr,
    ULONG wait_option);
```

Description

This service sends a message to the specified message queue. The sent message is **copied** to the queue from the memory area specified by the source pointer.

Parameters

- **queue_ptr**
  Pointer to a previously created message queue.
- **source_ptr**
  Pointer to the message.
- **wait_option**
  Defines how the service behaves if the message queue is full. The wait options are defined as follows:
  - **TX_NO_WAIT** (0x00000000) - Selecting TX_NO_WAIT results in an immediate return from this service regardless of whether or not it was successful. *This is the only valid option if the service is called from a non-thread; e.g., Initialization, timer, or ISR.*
  - **TX_WAIT_FOREVER** (0xFFFFFFFF) - Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until there is room in the queue.
  - timeout value (0x00000001 through 0xFFFFFFFE) - Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for room in the queue.

tx_queue_receive

Get message from message queue

Prototype

C Copy

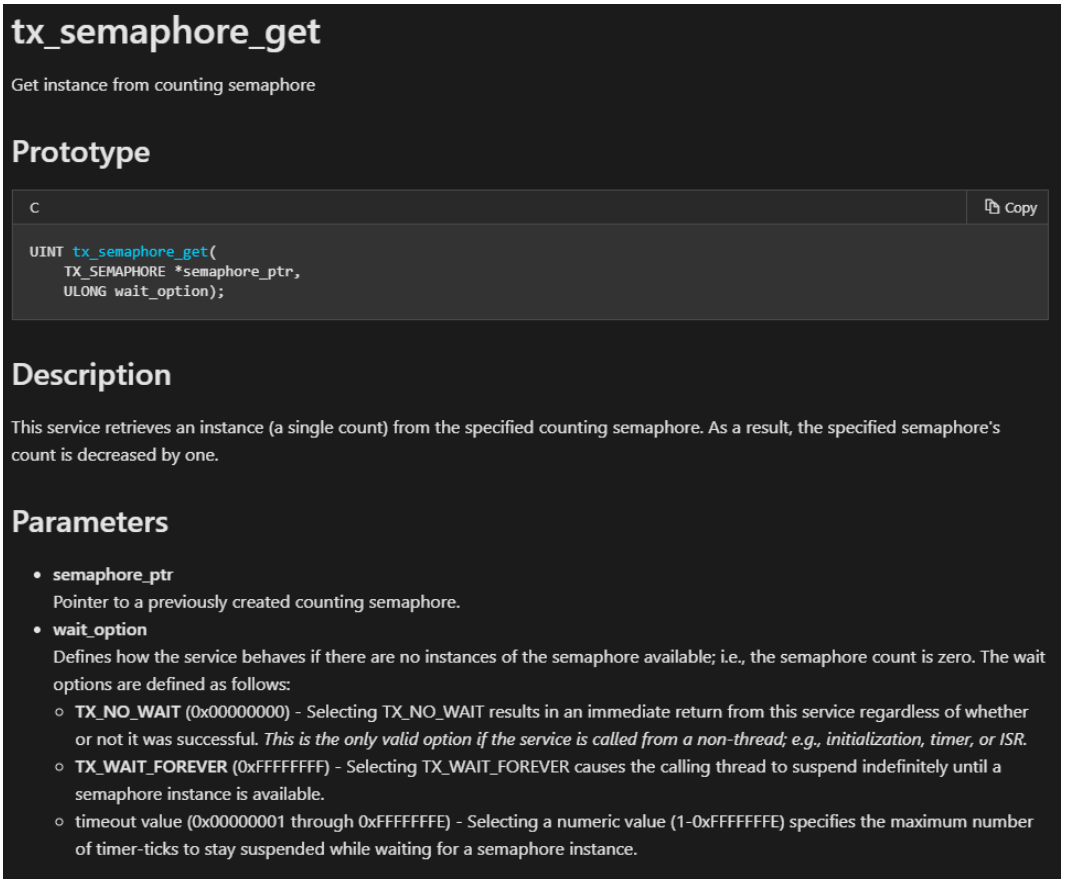
```
UINT tx_queue_receive(
    TX_QUEUE *queue_ptr,
    VOID *destination_ptr,
    ULONG wait_option);
```

Description

This service retrieves a message from the specified message queue. The retrieved message is **copied** from the queue into the memory area specified by the destination pointer. That message is then removed from the queue.

Parameters

- **queue_ptr**
  Pointer to a previously created message queue.
- **destination_ptr**
  Location of where to copy the message.
- **wait_option**
  Defines how the service behaves if the message queue is empty. The wait options are defined as follows:
  - **TX_NO_WAIT** (0x00000000) - Selecting TX_NO_WAIT results in an immediate return from this service regardless of whether or not it was successful. This is the only valid option if the service is called from a non-thread; e.g., Initialization, timer, or ISR.
  - **TX_WAIT_FOREVER** (0xFFFFFFFF) - Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a message is available.
  - timeout value (0x00000001 through 0xFFFFFFFE) - Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for a message.



tx_semaphore_get

Get instance from counting semaphore

Prototype

C Copy

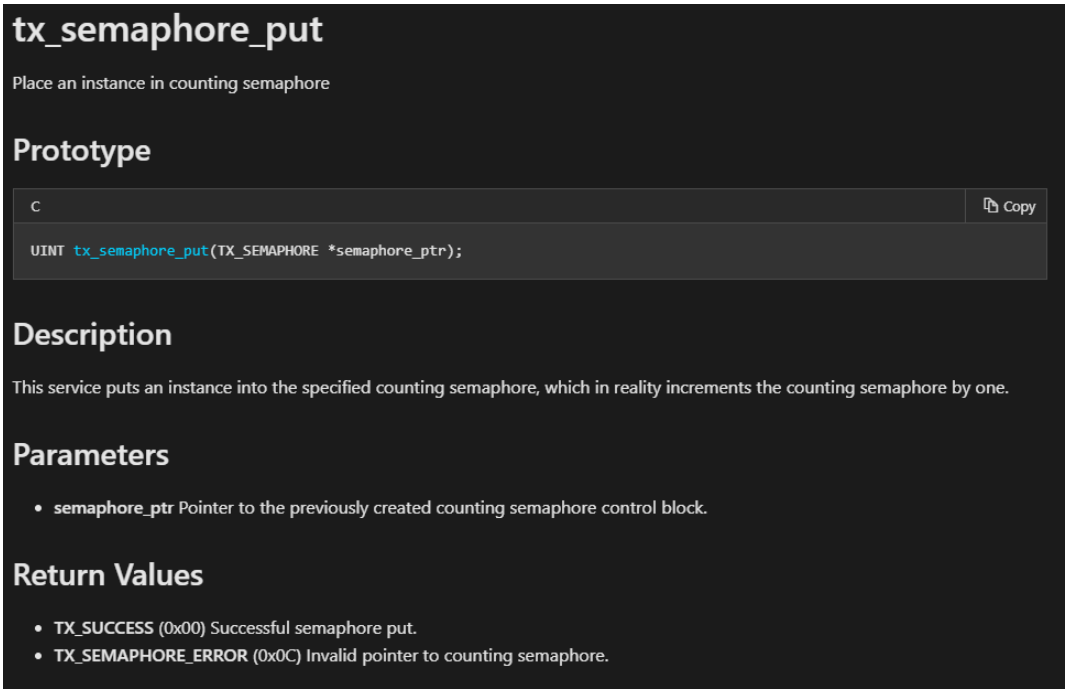
```
UINT tx_semaphore_get(
    TX_SEMAPHORE *semaphore_ptr,
    ULONG wait_option);
```

Description

This service retrieves an instance (a single count) from the specified counting semaphore. As a result, the specified semaphore's count is decreased by one.

Parameters

- **semaphore_ptr**
  Pointer to a previously created counting semaphore.
- **wait_option**
  Defines how the service behaves if there are no instances of the semaphore available; i.e., the semaphore count is zero. The wait options are defined as follows:
  - **TX_NO_WAIT** (0x00000000) - Selecting TX_NO_WAIT results in an immediate return from this service regardless of whether or not it was successful. *This is the only valid option if the service is called from a non-thread; e.g., initialization, timer, or ISR.*
  - **TX_WAIT_FOREVER** (0xFFFFFFFF) - Selecting TX_WAIT_FOREVER causes the calling thread to suspend indefinitely until a semaphore instance is available.
  - timeout value (0x00000001 through 0xFFFFFFFE) - Selecting a numeric value (1-0xFFFFFFFE) specifies the maximum number of timer-ticks to stay suspended while waiting for a semaphore instance.



tx_semaphore_put

Place an instance in counting semaphore

Prototype

C Copy

```
UINT tx_semaphore_put(TX_SEMAPHORE *semaphore_ptr);
```

Description

This service puts an instance into the specified counting semaphore, which in reality increments the counting semaphore by one.

Parameters

- **semaphore_ptr** Pointer to the previously created counting semaphore control block.

Return Values

- **TX_SUCCESS** (0x00) Successful semaphore put.
- **TX_SEMAPHORE_ERROR** (0x0C) Invalid pointer to counting semaphore.

Figuras 9, 10, 11, 12 e 13: Funções relevantes para manipulação de threads, filas e semáforos disponíveis no ThreadX