UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Documentação de Projeto – Parte 2
Design, Estudo da Plataforma

Projeto: Simulador de Elevadores

Autores: Raissa A N Higa

Versão: 13-Dez-2021

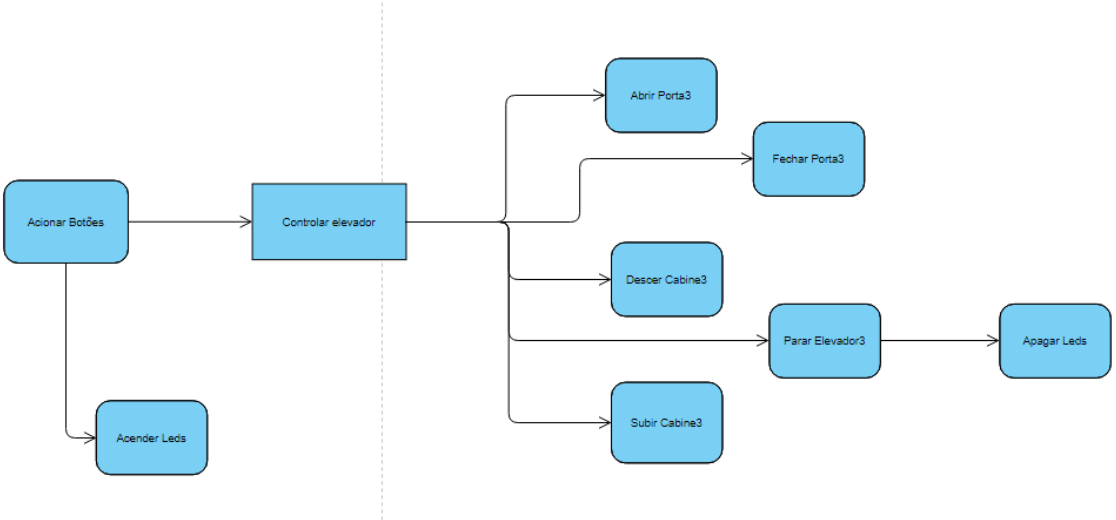
Parte 2a – Design

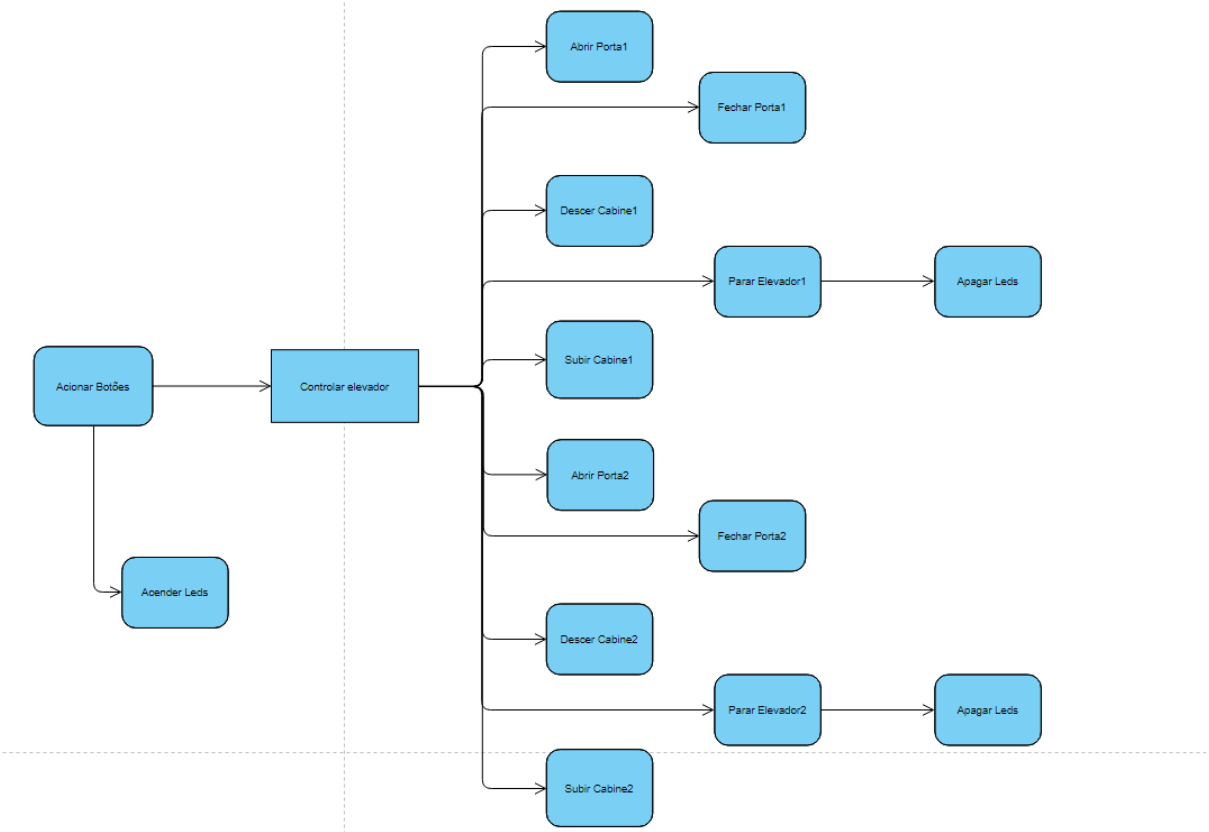
1 Introdução

Nessa segunda parte da documentação, são detalhados o design da solução escolhida, assim como o estudo das plataformas a serem utilizadas no projeto. O sistema em questão se trata de um simulador de um controlador de 3 elevadores, cada um podendo levar a 16 andares diferentes, em que é possível fazer requisições de chamadas de elevadores para o andar desejado e paradas nos andares destino pedidos. Conforme já especificado na parte 1 da Documentação sobre CONOPS, Domínio do Problema e Especificações.

2 Arquitetura Funcional

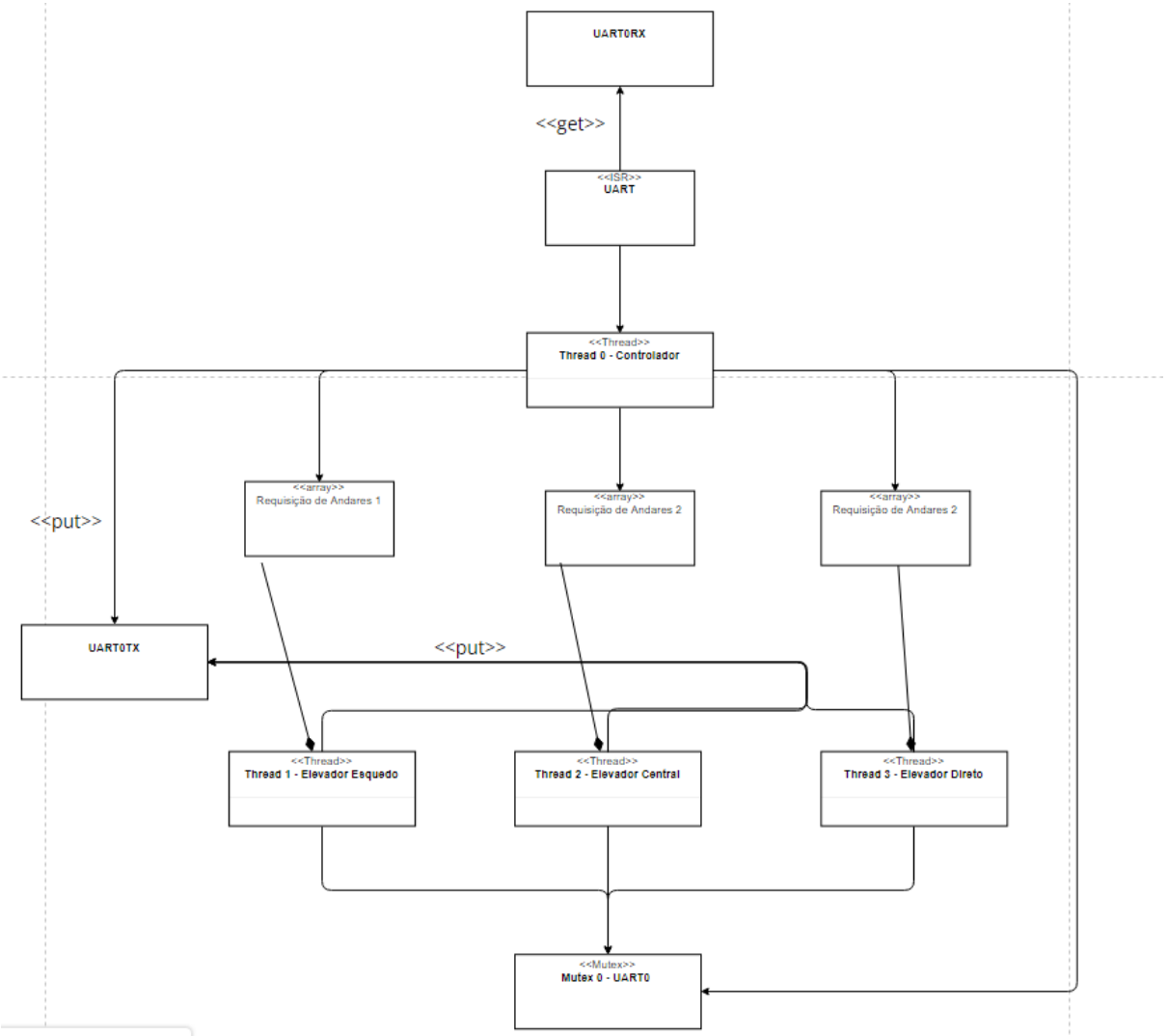
Através da interface do usuário, é possível acionar botões que são enviados como comandos que acendem os respectivos leds e controlam a subida e descida das cabines, a abertura e o fechamento das portas e a parada do elevador. Quando o elevador para em algum andar ele apaga o led do andar em questão.





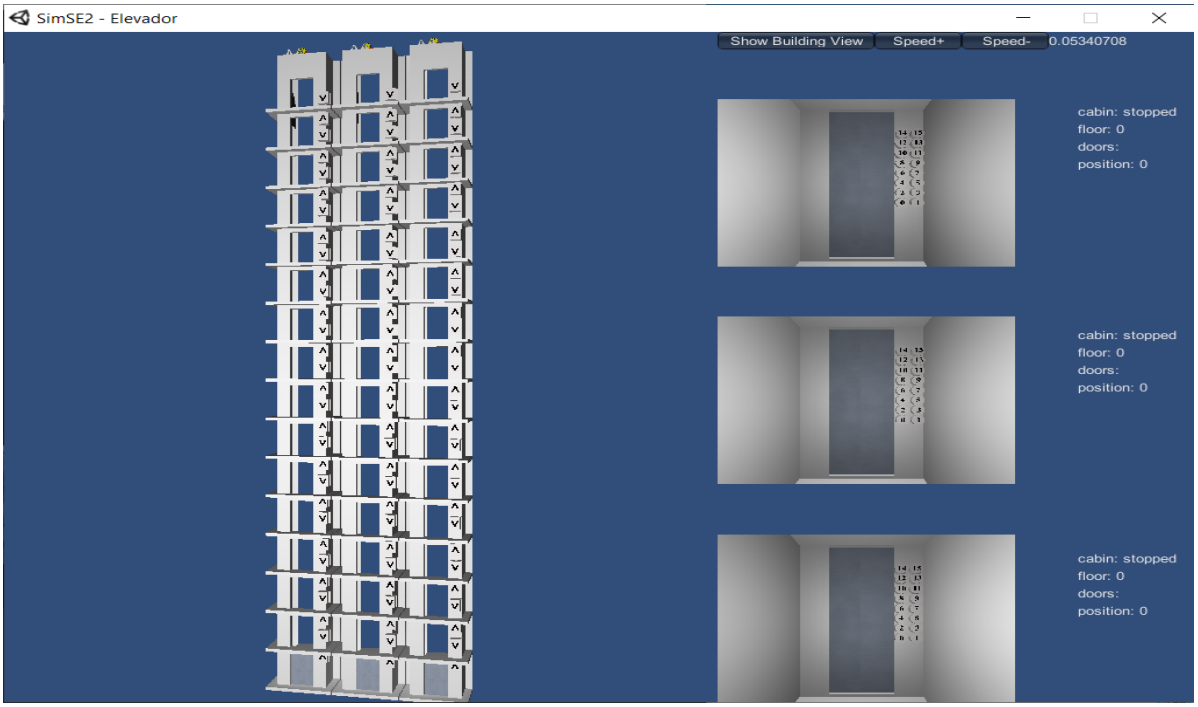
3 Arquitetura Física

O Diagrama a seguir descreve a arquitetura física da aplicação, com as relações entre as threads, interrupções e mutexes. Cada uma das threads de 1 a 3 controla um elevador, que compartilham do recurso UART0 para o qual podem enviar comandos para o simulador através do pino transmissor TX. Além disso, cada uma dessas threads possuem um vetor de requisições de andares que indicam em quais andares cada elevador deve parar. Ao receber uma interrupção devido a algum comando enviado pelo simulador através da UARTRX, a rotina de interrupção informa a thread 0, que atua como controladora de requisições e altera os valores dos vetores de requisições de andares para cada elevador de acordo com os comandos recebidos.



4 Interface com o Usuário

A interface com o usuário é feita através do já implementado simulador SE2 de elevadores disponibilizado para a disciplina. Nele é possível selecionar os botões internos e externos a serem pressionados, cada botão possui um Led associado em que é possível através de comandos de software acender e desligar sua emissão de luz. Graficamente, pode-se observar as portas abrindo e fechando, e as cabines se movimentando para cima e para baixo quando os comandos respectivos são recebidos, conforme descritos mais detalhadamente na Parte 2b Seção 1 Simulador SE2.



5 Mapeamento da Arquitetura Funcional à Arquitetura Física

A thread 0 faz o controle das requisições com o acendimento dos leds. A thread 1 controla o abrir e fechar da porta 1, o subir e descer da cabine 1 e a parada do elevador 1. Assim como as threads 2 e 3 para as suas respectivas portas, cabines e elevadores. As threads de 1 a 3 também controlam o apagar dos leds quando suas cabines param nos andares em questão. A UartRx controla o acionamento dos botões.

	Fechar Porta1	Fechar Porta2	Fechar Porta3	Abrir Porta1	Abrir Porta2	Abrir Porta3	Parar Elevador1	Parar Elevador2	Para Elevador3
Thread0									
Thread1	X			X			X		
Thread2		X			X			X	
Thread3			X			X			X
Uart Rx									

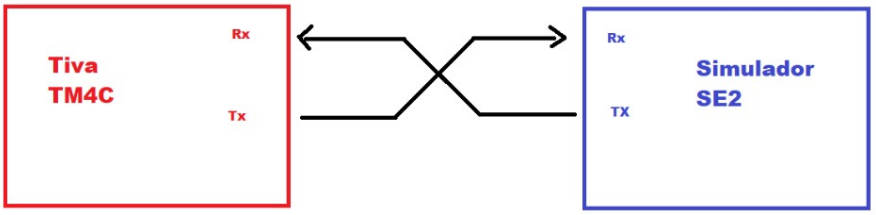
	Subir Cabine1	Subir Cabine2	Subir Cabine3	Descer Cabine1	Descer Cabine2	Descer Cabine3	Acender Led	Apagar Led	Acionar Botões
Thread0							X		
Thread1	X			X				X	
Thread2		X			X			X	
Thread3			X			X		X	
UartRx									X

6 Arquitetura do Hardware

Para essa solução de implementação de um simulador controlador de elevadores, o principal atributo do hardware utilizado é o periférico de comunicação serial UART, o qual fará a transmissão e recepção dos comandos enviados pelo simulador ao controlador e do controlador para o simulador. A UART é um protocolo de comunicação full-duplex, em que o

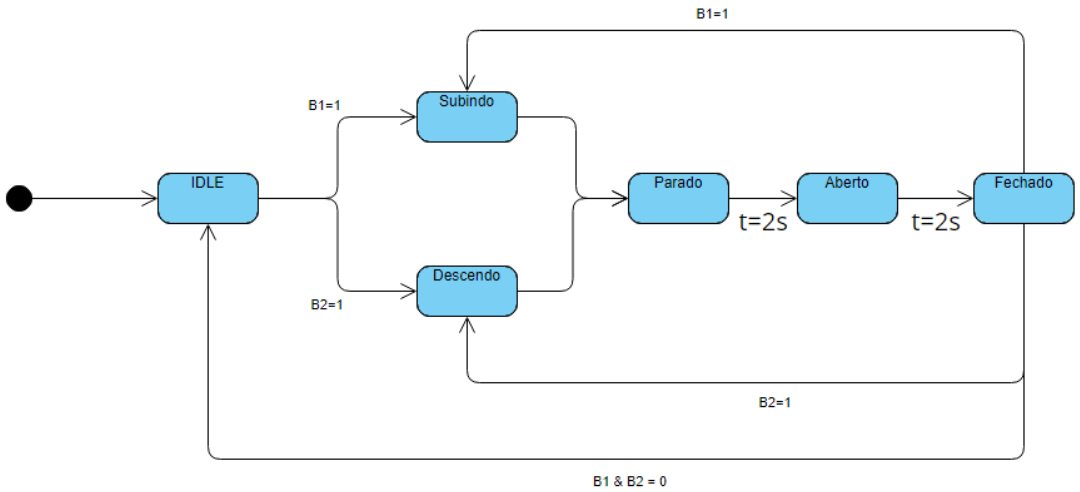
receptor Rx da placa Tiva receberá dados do transmissor Tx do Simulador SE2 e vice-versa, configurados com o mesmo *baudrate* (taxa de transmissão).

Será necessário além disso, a utilização dos recursos de interrupções, que serão ativos quando algum comando vindo do simulador for recebido, significando a requisição de algum elevador para um andar específico.

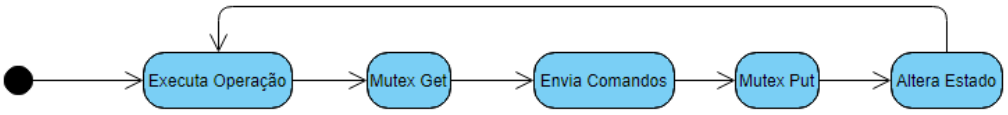


7 Design Detalhado

O Diagrama de Estados a seguir descreve o comportamento dos estados de um elevador. Ao iniciar, o elevador está em IDLE com a cabine parada e porta fechada. Quando algum botão de requisição é pressionado, se o andar referente ao botão é superior ao andar atual o elevador vai para o estado Subindo com a cabine se movimentando para cima, se o botão é para um andar inferior, o elevador vai para o estado Descendo com a cabine se movimentando para baixo. Ao chegar no andar de destino a cabine para e o elevador vai para o estado Parado, após 2 segundos a porta é aberta e o elevador vai para o estado Aberto, depois de mais 2 segundo a porta é fechada e vai para o estado Fechado. Se um botão para um andar acima é pressionado, vai para o estado Subindo, se for para um andar abaixo vai para o estado Descendo e se nenhum botão for pressionado o elevador volta para o estado IDLE.



O Diagrama de Atividades a seguir descreve o comportamento das threads 1 a 3. Primeiro se executa as operações referentes ao estado atual, após isso a thread tenta conseguir a posse do recurso compartilhado da UART de transmissão, quando bem sucedido envia os comandos para o simulador através da UART de acordo com o estado atual, para então devolver a posse do recurso compartilhado e alterar o estado se necessário.



Parte 2b – Estudo da Plataforma

1 Simulador SE2

No manual do simulador disponibilizado para a matéria, encontra-se toda a documentação de comando para o controle dos elevadores. É possível enviar comandos para cada elevador individualmente utilizando o formato: <elevador><comando><parâmetro>,

devendo ser terminado pelo código CR. As tabelas a seguir, retiradas do manual, detalham as funções de cada caracter.

Caractere	Detalhes
e	Envia comando para o elevador esquerdo
c	Envia comando para o elevador central
d	Envia comando para o elevador da direita

Caractere de elevador

Caractere	Função	Detalhes
r	Inicializa	Manda elevador para andar 0 com as portas abertas
a	Abre Portas	Abre as portas do elevador
f	Fecha Portas	Fecha as portas do elevador
s	Sobe	Manda elevador subir até receber comando de parada
d	Desce	Manda elevador descer até receber comando de parada
p	Para	Manda elevador parar o movimento
x	Consulta	Pergunta ao simulador posição do elevador
L<a..p>	Ligar botão	Liga luz do botão indicado
D<a..p>	Desligar botão	Desliga luz do botão indicado

Comandos para os elevadores

Caractere	Botão
a	Botão do térreo
b	Botão do 1º andar
c	Botão do 2º andar
d	Botão do 3º andar
e	Botão do 4º andar
f	Botão do 5º andar
g	Botão do 6º andar
h	Botão do 7º andar
i	Botão do 8º andar
j	Botão do 9º andar
k	Botão do 10º andar
l	Botão do 11º andar
m	Botão do 12º andar
n	Botão do 13º andar
o	Botão do 14º andar
p	Botão do 15º andar

Botões Internos

Para a utilização do simulador, também é necessário entender as respostas retornadas pelo simulador, ao requisitar ações ou ocorrência de eventos. Ao requisitar a posição do elevador é retornado um valor entre 0 e 75000. A ocorrência de eventos retorna um sinal como especificado na tabela a seguir:

Sinal	Significado	Detalhes
0	Elevador no térreo	Posição do elevador igual a 0
1	Elevador no 1º andar	Posição do elevador igual a posição do 1º andar
2	Elevador no 2º andar	Posição do elevador igual a posição do 2º andar
3	Elevador no 3º andar	Posição do elevador igual a posição do 3º andar
A	Portas completamente abertas	Porta direita e esquerda atingiram limite na movimentação de abertura
F	Portas completamente fechadas	Porta direita e esquerda atingiram limite na movimentação de fechamento
<elevador><a..j>	Botão a,b,c,d,e,f,g,h,i, ou j pressionado	Botão de movimentação do elevador (sobe e desce para cada andar, ir até andar) pressionado.

Ao pressionados, os botões internos de dentro da cabine enviam um código no formato: <eleavdor>I<botão>. E os botões externos para requisição de subida ou descida de um elevador até um determinado andar ao serem pressionados retornam: <elevador>E<andar_dezena><andar_unidade><direção>

Código	Botão	Código	Botão
a	Botão do térreo	i	Botão do 8º andar
b	Botão do 1º andar	j	Botão do 9º andar
c	Botão do 2º andar	k	Botão do 10º andar
d	Botão do 3º andar	l	Botão do 11º andar
e	Botão do 4º andar	m	Botão do 12º andar
f	Botão do 5º andar	n	Botão do 13º andar
g	Botão do 6º andar	o	Botão do 14º andar
h	Botão do 7º andar	p	Botão do 15º andar

Código	Elevador
e	Esquerdo
c	Central
d	Direito

Código	Elevador	Código	Andar	Código	Andar
e	Esquerdo	00	Térreo	08	8
c	Central	01	1	09	9
d	Direito	02	2	10	10
Código	Direção	03	3	11	11
s	Sobe	04	4	12	12
d	Desce	05	5	13	13
		06	6	14	14
		07	7	15	15

2 TIVA TM4C

Para fazer a comunicação serial entre a placa da Texas Instruments e a interface do simulador, é necessário de uma porta UART que opera através da porta USB a qual é conectada ao PC. Analisando os manuais a UART0 pode ser utilizada com o USB e o GPIO portA pinos 0 e 1 funcionam como receptor e transmissor da UART0 respectivamente.

IO	Pin	Analog or Special Function[a]	1
PA0	33	-	U0Rx
PA1	34	-	U0Tx

3 TivaWare

A solução necessita da utilização de algum meio de comunicação serial entre o simulador e a placa, para isso temos as funcionalidades da UART implementadas pela TivaWare, algumas das funções que podem vir a ser úteis estão descritas a seguir, baseado no manual TivaWare Peripheral Driver Library.

UARTConfigSetExpClk

void UARTConfigSetExpClk(uint32_t ui32Base,

uint32_t ui32UARTClk,

uint32_t ui32Baud,

uint32_t ui32Config)

Descrição: Configura a UART para operar em um formato específico.

Parâmetros:

ui32Base - endereço base da porta UART

ui32UARTClk - clock rate do módulo UART

ui32Baud - Baud Rate desejado

ui32Config - formado de dados do port

UARTIntEnable

void UARTIntEnable(uint32_t ui32Base,

uint32_t ui32IntFlags)

Descrição: Habilita as interrupções originadas por uma UART

Parâmetros:

ui32Base - endereço base da porta UART

ui32IntFlags - bit mask de interrupção a ser habilitado

UARTIntStatus

UARTIntStatus(uint32_t ui32Base,

bool bMasked)

Descrição: Retorna o status de interrupção de determinado UART

Parâmetros:

ui32Base - endereço base da porta UART

bMasked - verdadeiro se uma interrupção mascarada é requerida.

UARTIntClear

void UARTIntClear(uint32_t ui32Base,

uint32_t ui32IntFlags)

Descrição:

Parâmetros:

ui32Base - endereço base da porta UART

ui32IntFlags - bit mask de interrupção a ser limpado

UARTCharsAvail

bool UARTCharsAvail(uint32_t ui32Base)

Descrição: Determina se há caracter na FIFO de recebidos.

Parâmetros:

ui32Base - endereço base da porta UART

UARTCharPutNonBlocking

bool UARTCharPutNonBlocking(uint32_t ui32Base,

unsigned char ucData)

Descrição: Envia um character para uma FIFO do port específico.

Parâmetros:

ui32Base - endereço base da porta UART

ucData - caracter trasmitido

UARTCharGetNonBlocking

int32_t UARTCharGetNonBlocking(uint32_t ui32Base)

Descrição: Recebe um character de uma FIFO de um port específico.

Parâmetros:

ui32Base - endereço base da porta UART

Como a solução utiliza de UARTs, é necessário configurar os GPIOs para operar nesse modo. Alguns funções podem auxiliar nesse processo:

GPIOPinConfigure

void GPIOPinConfigure(uint32_t ui32PinConfig)

Descrição: Configura a função alternativa de um pino GPIO

Parâmetros:

ui32PinConfig - valor do pin a ser configurado

GPIOPinTypeUART

void GPIOPinTypeUART(uint32_t ui32Port,

uint8_t ui8Pins)

Descrição: Configura um pino para uso do periférico UART

Parâmetros:

ui32Port - endereço base da porta GPIO

ui8Pins - conjunto de bits que representam o pino.

4 ThreadX

Como a solução do controle dos 3 elevadores será baseada em threads, é necessário entender como a biblioteca utiliza os serviços de threads. Todas as informações estão disponíveis na documentação Azure RTOS ThreadX disponibilizada pela microsoft.

Entre as funções relacionadas à threads possíveis de serem utilizadas no projeto estão:

tx_thread_create

```
UINT tx_thread_create(
    TX_THREAD *thread_ptr,
    CHAR *name_ptr,
    VOID (*entry_function)(ULONG),
    ULONG entry_input,
    VOID *stack_start,
    ULONG stack_size,
    UINT priority,
    UINT preempt_threshold,
    ULONG time_slice,
    UINT auto_start);
```

Descrição: Cria uma Thread que começa a execução em uma função de entrada específica. Stack, prioridade, preempção e time-slice são atributos a serem especificados como parâmetros de entrada.

Parâmetros:
thread_ptr - ponteiro para um block de controle de threads
name_ptr - ponteiro para o nome da thread
entry_function - especifica a função inicial de execução
entry_input - valor de 32-bits passado para a função de entrada quando executada
stack_start - endereço inicial da área de memória do stack
stack_size - número de bytes na área de memória do stack
priority - prioridade da thread
preempt_threshold - preempção só pode ocorrer como prioridades maiores que esse valor
time_slice - número de timer-ticks em que a thread pode executar antes de perder o processador para outras threads de mesma prioridade.

tx_thread_sleep

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

Descrição: Causa a thread que a chamou a suspensão pelo determinado número de ticks especificado pela entrada.

Parâmetros:
timer_ticks - número de ticks em que a thread deve permanecer suspensa.

tx_thread_suspend

UINT tx_thread_suspend(TX_THREAD *thread_ptr);

Descrição: Suspende a thread especificada. Para voltar a executar tx_thread_resume deve ser chamada.

Parâmetros:
thread_ptr - ponteiro para a thread a ser suspensa.

tx_thread_resume

UINT tx_thread_resume(TX_THREAD *thread_ptr);

Descrição:

Parâmetros: Retoma a execução de uma thread suspensa
thread_ptr - ponteiro para uma thread suspendida.

Operações utilizando de recursos compartilhados também podem ser necessárias para a solução, para isso, a seguinte série de funções lidam com a criação e gerenciamento de mutuxes:

tx_mutex_create

```
UINT tx_mutex_create(
   TX_MUTEX *mutex_ptr,
   CHAR *name_ptr,
   UINT priority_inherit);
```

Descrição: Cria um mutex para a exclusão mútua de um recurso compartilhado.

Parâmetros:
mutex_ptr - ponteiro para um bloco de controle de mutex.
name_ptr - nome do mutex

priority_inherit - especifica se o mutex suporta herança de prioridade.TX_INHERIT se suporta e TX_NO_INHERIT se não suporta.

tx_mutex_get

```
UINT tx_mutex_get(
   TX_MUTEX *mutex_ptr,
   ULONG wait_option);
```

Descrição: Tenta reservar o uso de um recurso compartilhado por um período.

Parâmetros:
mutex_ptr - ponteiro para um mutex
wait_option - tempo máximo de espera pelo recurso compartilhado quando esse está em posse de outra thread.TX_NO_WAIT não espera. TX_WAIT_FOREVER, espera indefinidamentte.

tx_mutex_put

```
UINT tx_mutex_put(TX_MUTEX *mutex_ptr);
```

Descrição: Libera o uso de um recurso compartilhado.

Parâmetros:
mutex_ptr - ponteiro para um mutex

Operações que envolvem a criação e alocação das áreas de memórias também podem ser úteis para o sistema.

tx_byte_pool_create

```
UINT tx_byte_pool_create(
   TX_BYTE_POOL *pool_ptr,
   CHAR *name_ptr,
   VOID *pool_start,
   ULONG pool_size);
```

Descrição: Cria uma área de memória

Parâmetros:
pool_ptr - ponteiro para o bloco de controle de memória
name_ptr - ponteiro para o nome da memory pool
pool_start - endereço inicial da memory pool
pool_size - número total de bytes disponíveis para a memory pool.

tx_byte_allocate

```
UINT tx_byte_allocate(
    TX_BYTE_POOL *pool_ptr,
    VOID **memory_ptr,
    ULONG memory_size,
    ULONG wait_option);
```

Descrição: Aloca um número específico de bytes de uma área de memória.

Parâmetros:
pool_ptr - ponteiro para uma memory block pool
memory_ptr - ponteiro para um ponteiro de memória destino.
memory_size - número de bytes requisitados
wait_option - comportamento se não houver espaço suficiente disponível.