Sistemas Embarcados CSW41

Matheus Mattos

LAB 5

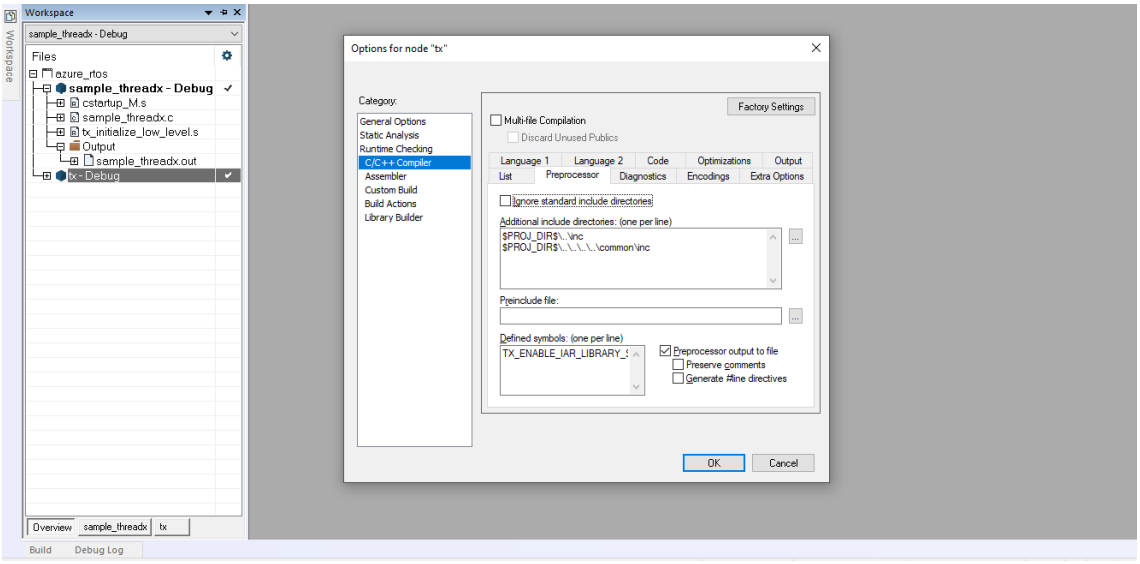
1. Objetivo

O objetivo deste laboratório é de se portar para a Tiva o projeto exemplo do ThreadX.

2. Configuração do projeto na IDE (IAR)

Para este laboratório foi necessário configurar novamente o ambiente de desenvolvimento visto que iremos utilizar uma nova plataforma de RTOS no caso a ThreadX.

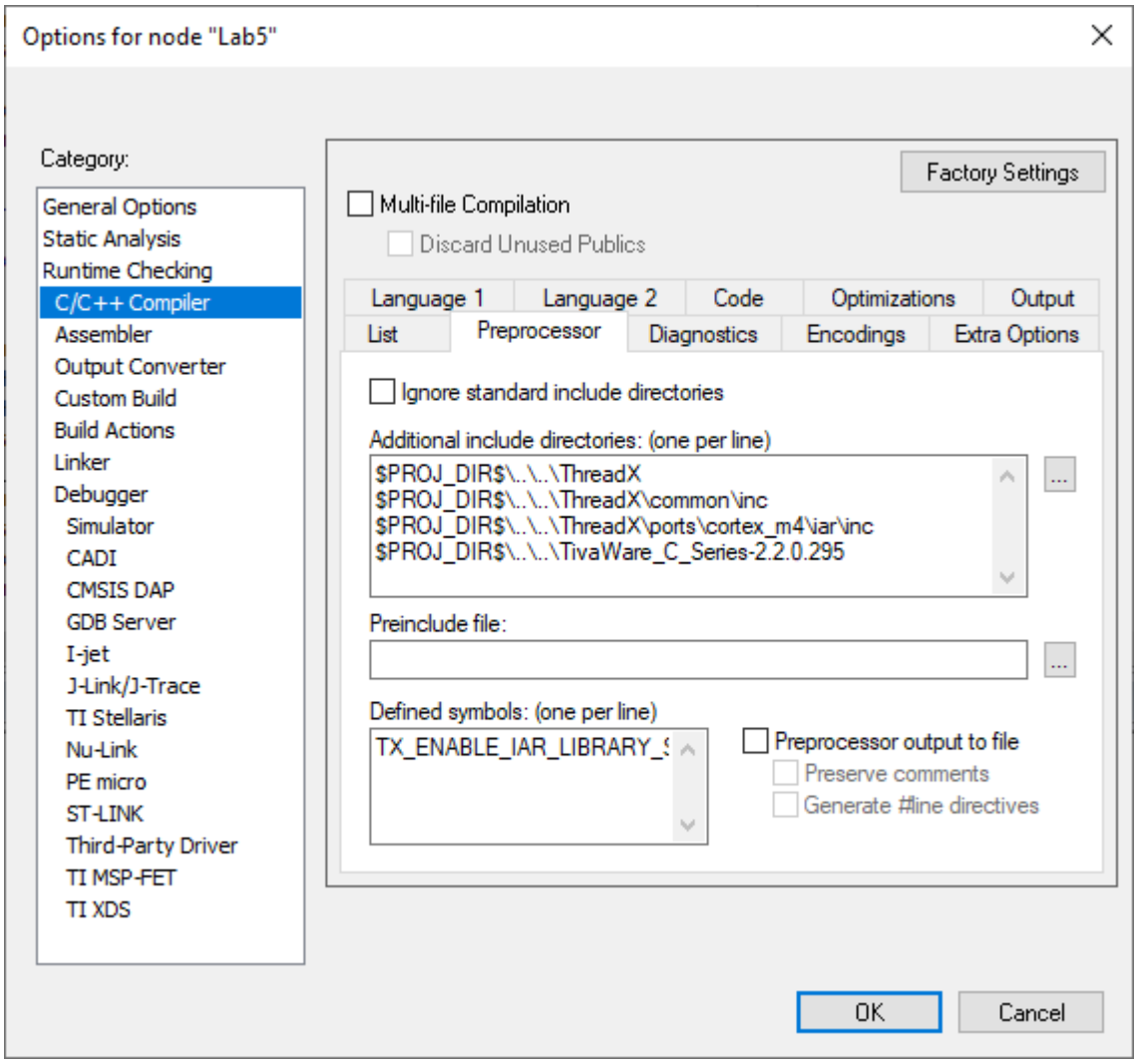
No primeiro passo foi necessário abrir o workspace do azure_rtos e mapear os arquivos tanto para o projeto tx quanto para o sample_threadx, conforme a imagem abaixo.



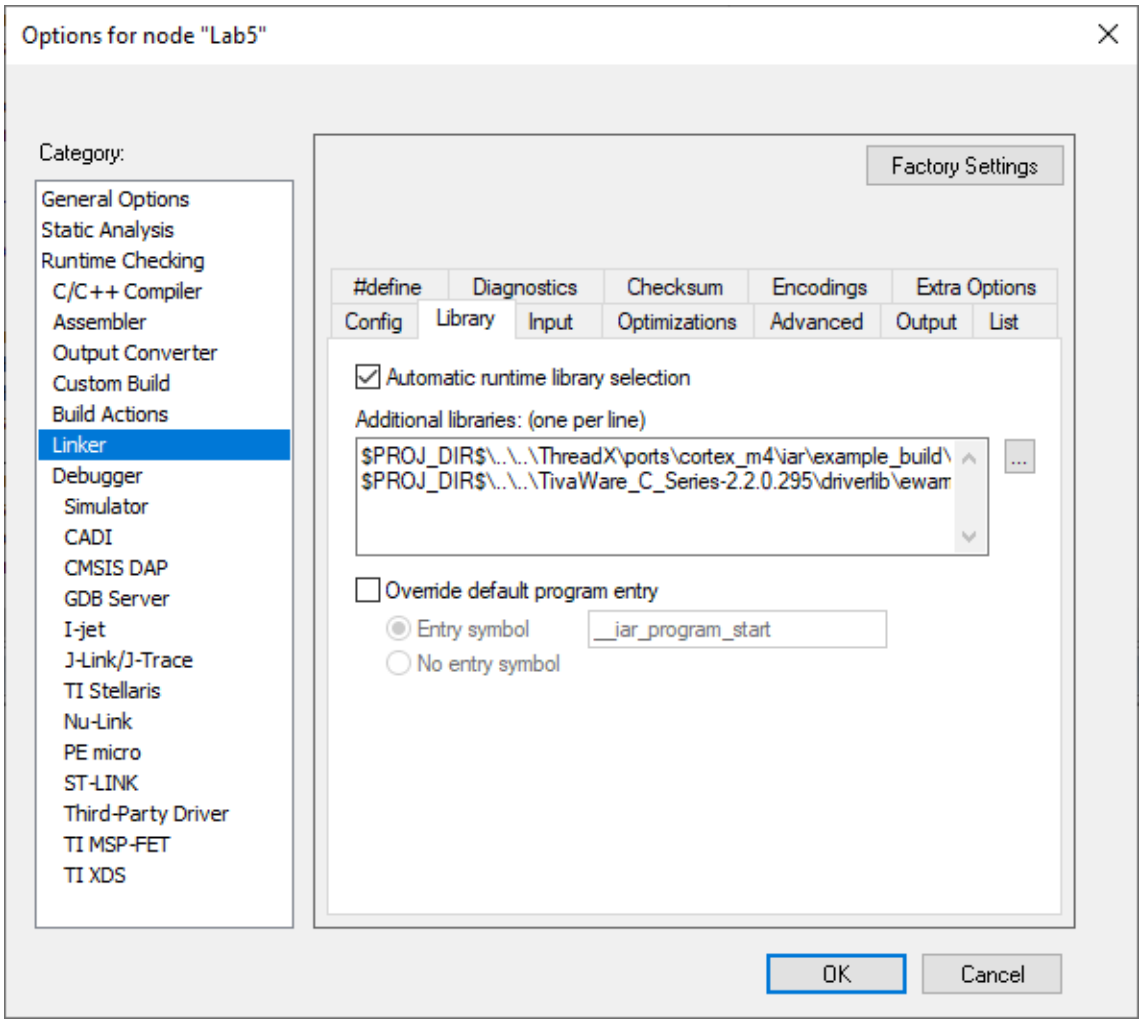
Feito isso, foi necessário compilar o ThreadX em uma biblioteca estática para que ele esteja disponível no nosso código, para compilar para arquitetura ARM Cortex M4 é só ir na pasta /ThreadX/ports/cortex_m4 e escolher em qual compilador você irá compilar. No nosso caso iremos utilizar a IDE da iar, para isso foi necessário abrir o projeto que se encontra na pasta \ThreadX\ports\cortex_m4\iar e compilar. Após feita a compilação com sucesso foi gerado o arquivo tx.a, que iremos utilizar mais para frente no laboratório 5.

Após a criação da biblioteca tx.a, foi criado o projeto Lab5 e copiado os arquivos sample_threadx.c, cstartup.c e tx_initialize_low_level.c para pasta src.

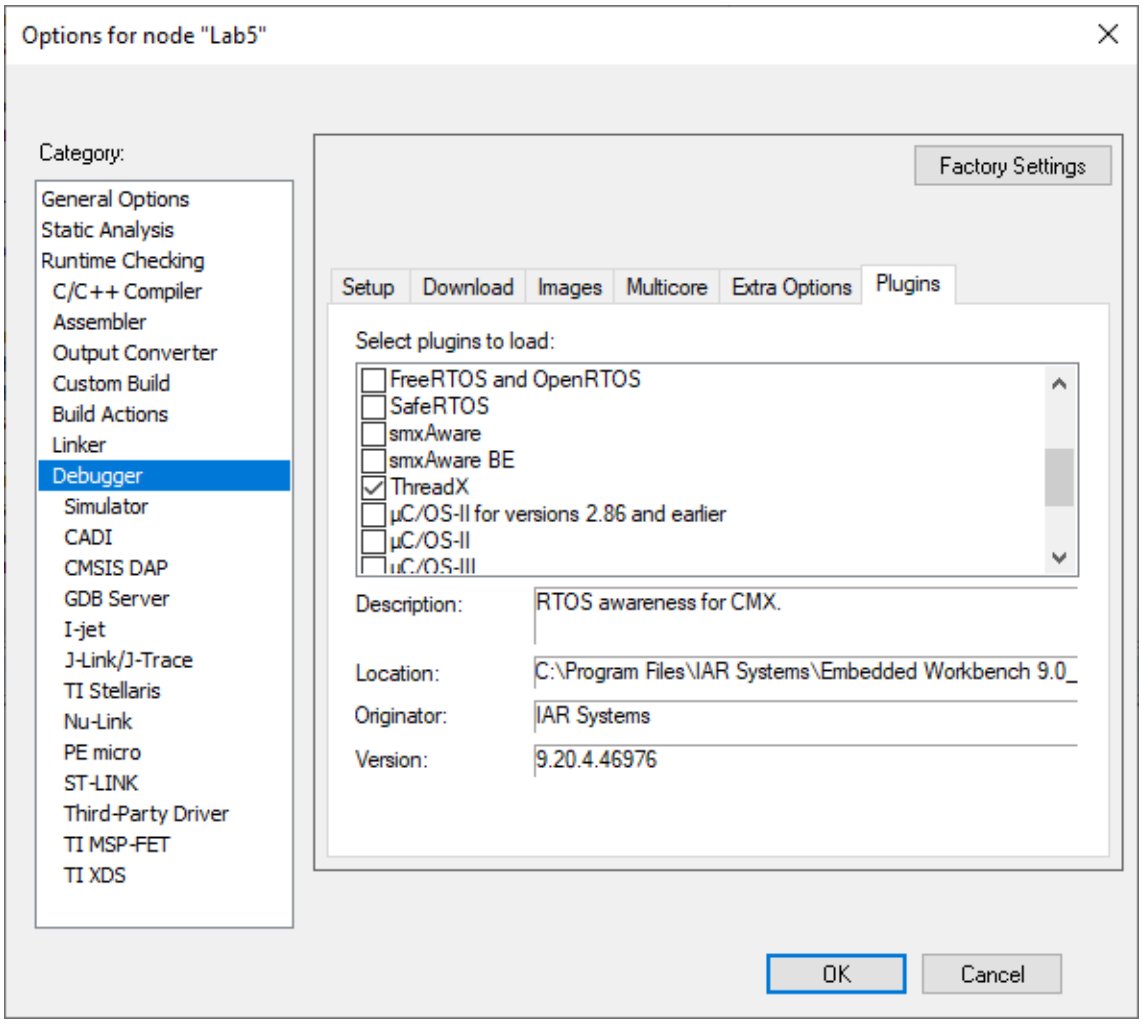
Com o laboratório 5 criado no IAR dentro da nossa workspace, foi necessário fazer o mapeamento do Threadx e da TivaWare no projeto conforme as imagens abaixo.



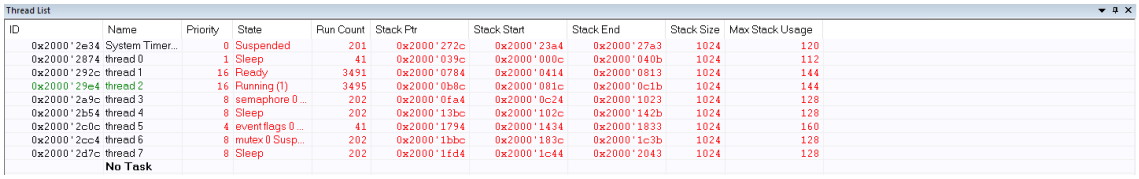
No Linker foi necessário mapear a biblioteca do ThreadX, o arquivo que foi compilado nos primeiros passos deste projeto e mapear a biblioteca do Tivaware, conforme a imabem abaixo.



Em Debugger foi necessário habilitar o plugin do ThreadX, para que na hora do debug as funções do ThreadX estaja disponível para acesso.



O plugin da ThreadX é muito importante pois ele dá informações sobre as threads durante o debug, basta apenas apertar o botão pause que é possível ver as informações de cada thread, existem várias funções que podem ser exploradas, mas para este lab iremos utilizar a Thread List que é necessária para preencher a tabela.

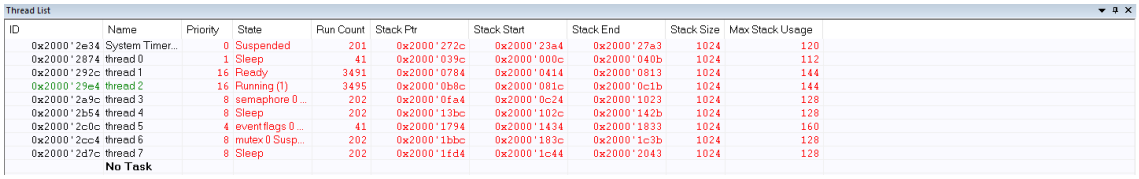


Todos os nomes de Threads estão definidos logo no começo do código, assim é possível preencher todos os campos "entry function " da tabela.

```
/* Define thread prototypes.  */

void    thread_0_entry(ULONG thread_input);
void    thread_1_entry(ULONG thread_input);
void    thread_2_entry(ULONG thread_input);
void    thread_3_and_4_entry(ULONG thread_input);
void    thread_5_entry(ULONG thread_input);
void    thread_6_and_7_entry(ULONG thread_input);
```

Para preencher o stack size e priority utilizamos o plugin Thread List.



Thread List

ID	Name	Priority	State	Run Count	Stack Ptr	Stack Start	Stack End	Stack Size	Max Stack Usage
0x2000'2e34	System Timer...	0	Suspended	201	0x2000'272c	0x2000'23a4	0x2000'27a3	1024	120
0x2000'2874	thread 0	1	Sleep	41	0x2000'039c	0x2000'000c	0x2000'040b	1024	112
0x2000'292c	thread 1	16	Ready	3491	0x2000'0784	0x2000'0414	0x2000'0813	1024	144
0x2000'29e4	thread 2	16	Running (1)	3495	0x2000'0b8c	0x2000'081c	0x2000'0c1b	1024	144
0x2000'2a9c	thread 3	8	semaphore 0 ...	202	0x2000'0fa4	0x2000'0c24	0x2000'1023	1024	128
0x2000'2b54	thread 4	8	Sleep	202	0x2000'13bc	0x2000'102c	0x2000'142b	1024	128
0x2000'2c0c	thread 5	4	event flags 0 ...	41	0x2000'1794	0x2000'1434	0x2000'1833	1024	160
0x2000'2cc4	thread 6	8	mutex 0 Susp...	202	0x2000'1bbc	0x2000'183c	0x2000'1c3b	1024	128
0x2000'2d7c	thread 7	8	Sleep	202	0x2000'1fd4	0x2000'1c44	0x2000'2043	1024	128
	No Task								

Para preencher o auto start e o time slicing, foi necessário consultar a documentação do ThreadX no qual diz que o quarto parâmetro da função tx_thread_create() é o time-slice (Tempo que a tarefa executa sem ser interrompida) e o quinto parâmetro é o auto start.

tx_thread_create

Create application thread

Prototype

C Copy

```
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 dos parâmetros da tx_thread_create().

Parameters

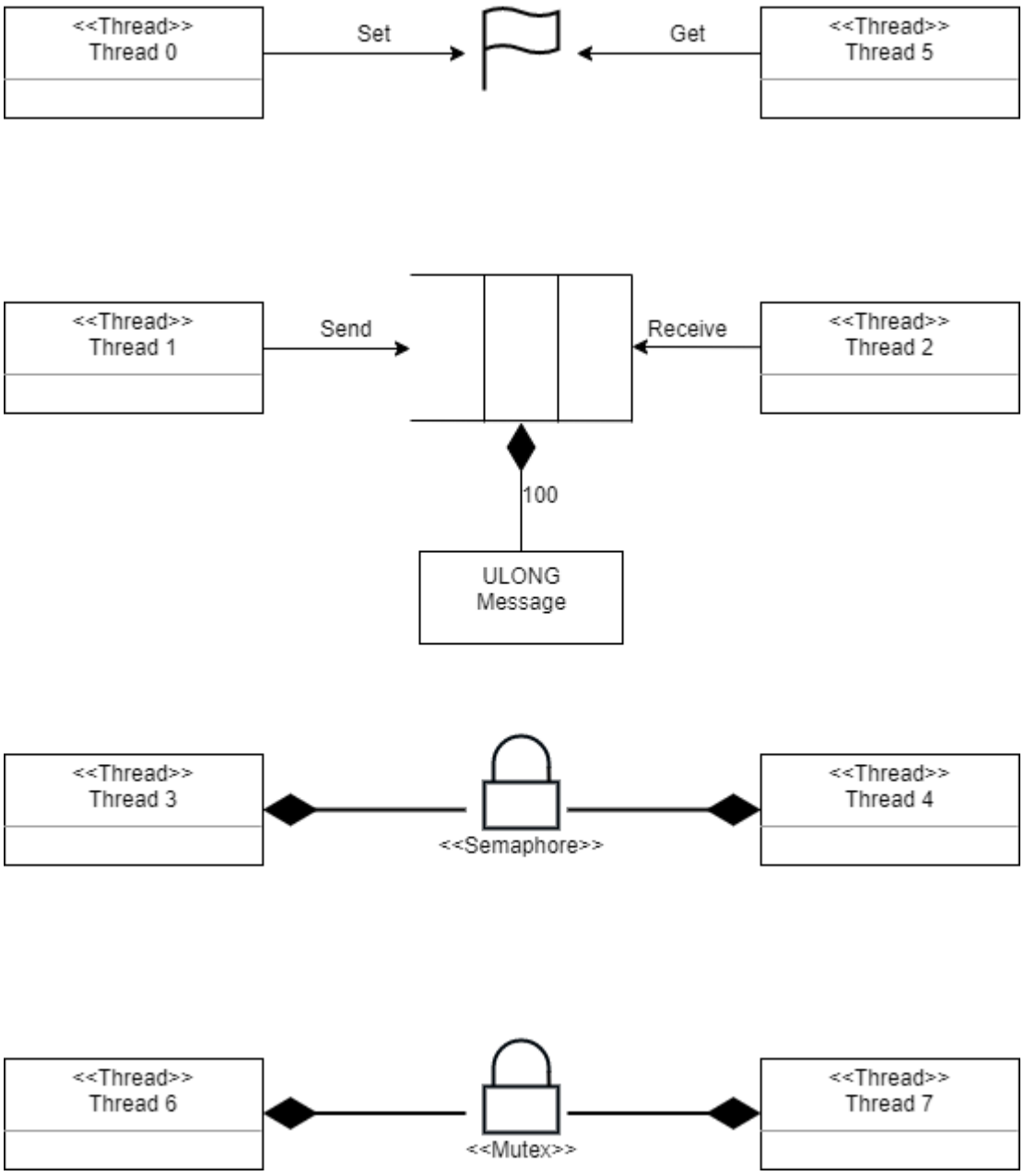
- *thread_ptr*: Pointer to a thread control block.
- *name_ptr*: Pointer to the name of the thread.
- *entry_function*: Specifies the initial C function for thread execution. When a thread returns from this entry function, it is placed in a *completed* state and suspended indefinitely.
- *entry_input*: A 32-bit value that is passed to the thread's entry function when it first executes. The use for this input is determined exclusively by the application.
- *stack_start*: Starting address of the stack's memory area.
- *stack_size*: Number bytes in the stack memory area. The thread's stack area must be large enough to handle its worst-case function call nesting and local variable usage.
- *priority*: Numerical priority of thread. Legal values range from 0 through (TX_MAX_PRIORITES-1), where a value of 0 represents the highest priority.
- *preempt_threshold*: Highest priority level (0 through (TX_MAX_PRIORITIES-1)) of disabled preemption. Only priorities higher than this level are allowed to preempt this thread. This value must be less than or equal to the specified priority. A value equal to the thread priority disables preemption-threshold.
- *time_slice*: Number of timer-ticks this thread is allowed to run before other ready threads of the same priority are given a chance to run. Note that using preemption-threshold disables time-slicing. Legal time-slice values range from 1 to 0xFFFFFFFF (inclusive). A value of **TX_NO_TIME_SLICE** (a value of 0) disables time-slicing of this thread.

Explorando todos os campos, as tabelas ficaram da seguinte maneira:

Thread Name	Entry function	Stack size	Priority	Auto Start	Time Slicing
thread 0	thread_0_entry	1024	1	yes	no
thread 1	thread_1_entry	1024	16	yes	4
thread 2	thread_2_entry	1024	16	yes	4
thread 3	thread_3_and_4_entry	1024	8	yes	no
thread 4	thread_3_and_4_entry	1024	8	yes	no
thread 5	thread_5_entry	1024	4	yes	no
thread 6	thread_6_and_7_entry	1024	8	yes	no
thread 7	thread_6_and_7_entry	1024	8	yes	no

Name	Control Structure	Size	Location
byte pool 0	byte_pool_0	9120	byte_pool_memory
queue 0	queue_0	100*sizeof(ULONG)	pointer
semaphore 0	semaphore_0	sizeof(TX_SEMAPHORE)	&semaphore_0
mutex 0	mutex_0	sizeof(TX_MUTEX)	&event_flags_0
event flags 0	event_flags_0	sizeof(TX_EVENT_FLAGS_GROUP)	&mutex_0
block pool 0	block_pool_0	100	pointer

O diagrama ficou da seguinte maneira:



Na criação do código foi inserido uma função antes da execução do tx_kernel_enter(). Essa função foi inserida para setar o clock.

Na thread_0, foi inserido a função piscaled(), essa função foi definida para ficar acendendo e apagando o led até 10x, após isso ele executa a próxima thread.