Contents

第 1 章：概述

2021/4/29 •

ThreadX 模块组件为应用程序提供了一种基础结构，用于动态地加载独立于应用程序常驻部分构建的模块。在应用程序代码大小超过可用内存的情况下，此功能尤其有用。当部署核心映像后需要添加新功能时，它也可以提供帮助。此外，在需要部分固件更新时，可以使用动态加载模块。

根据模块报头中指定的属性，已加载模块的内存保护是可选的。指定内存保护时，会配置处理器的内存管理硬件，以便仅允许模块的所有线程访问模块的代码和数据内存。任何无关的内存访问或执行都将导致内存错误，并将终止有问题的模块线程。如果应用程序注册了内存错误通知回调，则还将调用此回调来通知应用程序出现内存错误。

ThreadX 模块组件依赖应用程序提供可在其中加载模块的内存区域。每个模块的指令区域可能会就地执行，或将其复制到 RAM 模块内存区域中进行执行。在所有情况下，都将从模块内存区域中分配模块数据内存要求。

不限制可同时加载的模块数量（除了可用的内存量以外），而常驻模块管理器代码只能有一个副本。图 1 说明了模块管理器与模块本身之间的关系。

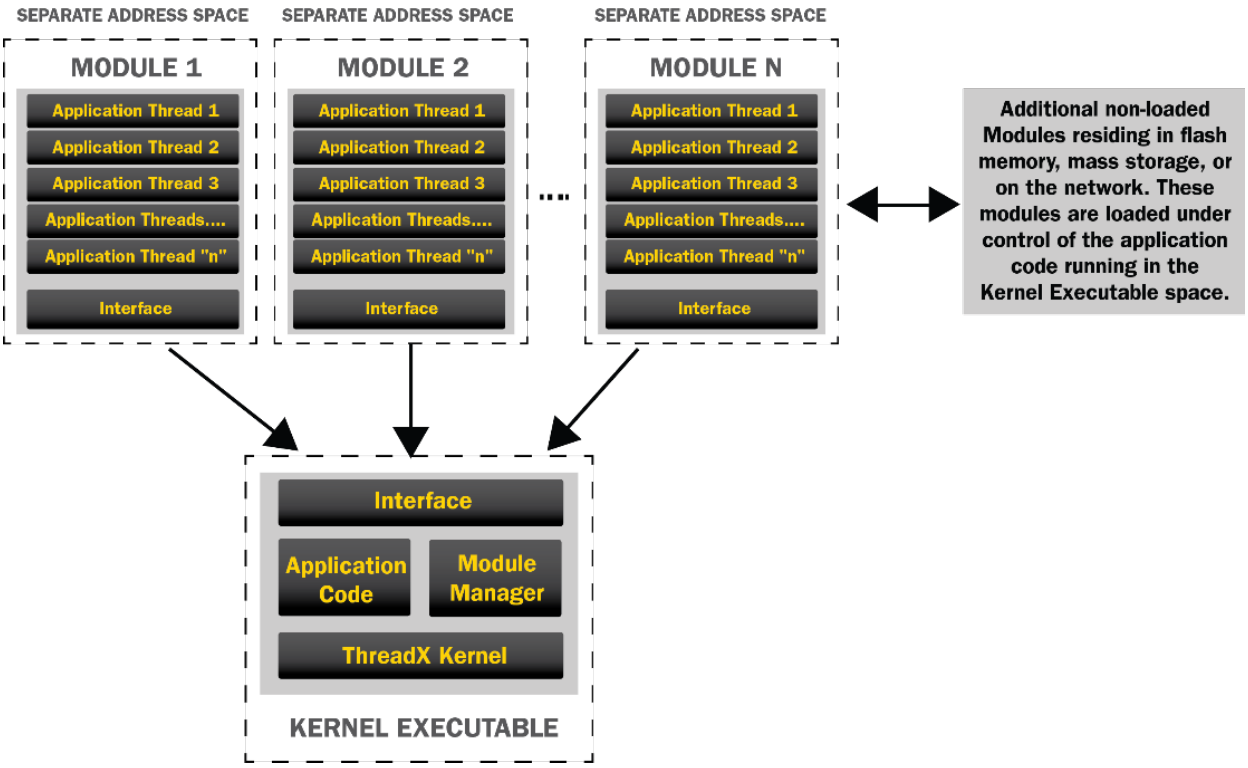

图 1 模块和模块管理器

每个模块都必须有其自己的内存区域，这是定义的应用程序的责任。模块和模块管理器利用相对于模块请求的 ThreadX 服务的预定义请求 ID 通过软件调度函数进行交互。此外，此模块还需要提供单个线程入口点以及所需的堆栈大小、优先级、模块 ID、回调线程堆栈大小/优先级等。此信息定义在每个模块的报头中。

模块管理器负责创建初始模块线程并开始其执行。模块的初始线程开始执行后，模块管理器负责处理模块发出的所有 ThreadX API 请求。模块对 ThreadX API 具有完全访问权限，包括可以在模块中创建其他线程。

模块源代码命名约定非常简单：所有模块管理器源文件都命名为 txm_module_manager_ *，所有与模块专门关联的文件将省略该名称的“manager”***部分。主要包含文件 _txm_module.h** 通过管理器和模块源代码分享。

第 2 章 - 模块要求

2021/4/29 •

ThreadX 模块包含报头，用于定义模块的基本特征。报头后跟模块的说明区域。模块可以就地执行，也可以在执行之前由模块管理器加载到模块内存区域中。唯一的要求是，报头始终位于模块的第一个地址。图 2 说明了一个基本模块布局。

[模块报头]
[模块说明区域]
[模块 RAM 区域]

图 2 - 模块布局

> **NOTE**
> 必须用适当的位置独立的代码和数据编译器/链接器选项生成模块。这样就可以在所有内存区域中执行模块。

创建模块线程时，将分配第二个堆栈空间，以便线程位于受内存保护的内核中时使用。线程内核堆栈空间的大小可由用户使用 txm_module_port.h 中 TXM_MODULE_KERNEL_STACK_SIZE 进行配置。这允许在创建模块线程时使用较小的堆栈大小，因为用户在调用 tx_thread_create 时指定的堆栈只在模块中使用 __。

> **NOTE**
> 模块线程堆栈的顶部包含线程入口信息结构 (TXM_MODULE_THREAD_ENTRY_INFO)，因此该结构的大小会减小可用的堆栈大小。在模块中创建线程时，请将它的堆栈大小至少增加到此结构的大小。

需要执行以下步骤来创建和生成 ThreadX 模块（下面更详细地介绍了每个步骤）。

1. 模块中的所有 C 文件必须在包含 txm_module.h 之前具有 #define TXM_MODULE 。这可以在正在编译的源文件中或作为项目设置的一部分来完成。这样做会将 ThreadX API 调用重新映射到特定于模块的 API 版本，该版本调用常驻模块管理器中的调度函数，以执行对实际 API 函数的调用。
2. 每个模块的第一个指令区域地址中必须有一个报头，该报头定义了模块的特性和资源需求。
3. 每个模块必须链接模块指令区域开头的报头。
4. 每个模块必须针对模块库 (txm.a) 进行链接，其中包含用于与 ThreadX 交互的模块特定函数。

模块源

ThreadX 模块具有其自己的一组源文件，旨在使用模块源代码直接进行链接和定位。这些文件联接单独模块和常驻模块管理器。模块文件如下所示。

txm_module.h	包含定义模块信息的文件。

ttt	tt
txm_module_port.h	包含定义特定于端口的模块信息的文件。
txm_module_user.h	定义用户可以自定义的值。
txm_module_initialize.s [1][3]	调用启动模块的内部函数。
txm_module_preamble.{s/S/68}	模块报头程序集文件。此文件定义了各种特定于模块的属性，并与模块应用程序代码链接。
txm_module_application_request.c [1]	模块应用程序请求函数将特定于应用程序的请求发送到常驻代码。
txm_module_callback_request_thread_entry.c [1]	负责处理模块所请求回叫的模块回叫线程，包括计时器和通知回叫。
txm_*.c [1][2]	标准 ThreadX API 服务，这些服务调用内核调度程序。
txm_module_object_allocate.c [1]	用于为管理器内存池中的模块对象分配内存的模块函数。
txm_module_object_deallocate.c [1]	用于为管理器内存池中的模块对象解除内存分配的模块函数。
txm_module_object_pointer_get.c [1]	用于检索指向系统对象的指针的模块函数。
txm_module_object_pointer_get_extended.c [1]	用于检索指向系统对象、名称长度安全的指针的模块函数。
txm_module_thread_shell_entry.c [1]	模块线程入口函数。
txm_module_thread_system_suspend.c [1]	挂起线程的模块函数。

[1] 位于库 txm.a 中。

[2] 这些文件与 ThreadX API 文件具有相同的名称，其中前缀为 txm_，而不是 tx_ 。

[3] txm_module_initialize.s 文件仅适用于使用 ARM 工具的端口。

模块报头

模块报头定义模块的特性和资源。在报头中定义了初始线程入口函数以及与线程关联的初始内存区域等信息。特定于端口的报头示例位于附录中。图 3 显示了通用目标的示例 ThreadX 模块报头（以 * 开头的行是应用程序通常修改的值）：

```
    AREA Init, CODE, READONLY

    /* Define public symbols. */
    EXPORT __txm_module_preamble

    /* Define application-specific start/stop entry points for the module. */
    IMPORT demo_module_start

    /* Define common external refrences. */
    IMPORT _txm_module_thread_shell_entry
    IMPORT _txm_module_callback_request_thread_entry
    IMPORT |Image$$ER_RO$$Length|
    IMPORT |Image$$ER_RW$$Length|

__txm_module_preamble
    DCD     0x4D4F4455                                  ; Module ID
    DCD     0x6                                         ; Module Major Version
    DCD     0x1                                         ; Module Minor Version
    DCD     32                                          ; Module Preamble Size in 32-bit words
*   DCD     0x12345678                                  ; Module ID (application defined)
*   DCD     0x01000001                                  ; Module Properties where:
                                                        ; Bits 31-24: Compiler ID
                                                        ;   0 -> IAR
                                                        ;   1 -> ARM
                                                        ;   2 -> GNU
                                                        ;   Bits 23-1: Reserved
                                                        ;   Bit 0: 0 -> Privileged mode execution (no MMU
protection)
                                                        ;          1 -> User mode execution (MMU protection)
    DCD     _txm_module_thread_shell_entry - . + .      ; Module Shell Entry Point
*   DCD     demo_module_start - . + .                   ; Module Start Thread Entry Point
    DCD     0                                           ; Module Stop Thread Entry Point
*   DCD     1                                           ; Module Start/Stop Thread Priority
*   DCD     2048                                        ; Module Start/Stop Thread Stack Size
    DCD     _txm_module_callback_request_thread_entry - . + . ; Module Callback Thread Entry
    DCD     1                                           ; Module Callback Thread Priority
    DCD     2048                                        ; Module Callback Thread Stack Size
    DCD     |Image$$ER_RO$$Length|                      ; Module Code Size
    DCD     |Image$$ER_RW$$Length|                      ; Module Data Size
    DCD     0                                           ; Reserved 0
    DCD     0                                           ; Reserved 1
    DCD     0                                           ; Reserved 2
    DCD     0                                           ; Reserved 3
    DCD     0                                           ; Reserved 4
    DCD     0                                           ; Reserved 5
    DCD     0                                           ; Reserved 6
    DCD     0                                           ; Reserved 7
    DCD     0                                           ; Reserved 8
    DCD     0                                           ; Reserved 9
    DCD     0                                           ; Reserved 10
    DCD     0                                           ; Reserved 11
    DCD     0                                           ; Reserved 12
    DCD     0                                           ; Reserved 13
    DCD     0                                           ; Reserved 14
    DCD     0                                           ; Reserved 15
    END
```

图 3

在大多数情况下，开发者只需定义模块的起始线程（偏移 0x1C）、模块 ID（偏移 0x10），启动/停止线程优先级（偏移 0x24），以及启动/停止线程堆栈大小（偏移 0x28）。上面的演示设置模块的起始线程为 demo_module_start，模块 ID 为 0x12345678，启动线程的优先级为 1，堆栈大小为 2048 字节 。

某些应用程序可能会选择性地定义停止线程，该操作在模块管理器停止模块时执行。此外，某些应用程序可能使用如下所示的“模块属性”字段。

模块属性位图

下表显示了属性位图的示例。特定于端口的属性位图位于附录中。

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x01 0x02 0x03	■ ID IAR ARM GNU

模块链接器控件文件

生成模块时，必须在所有其他代码节之前放置模块报头。模块必须用与位置无关的代码和数据部分生成。特定于端口的示例链接器文件位于附录中。

模块 ThreadX 库

每个模块必须针对以模块为中心的特殊 ThreadX 库进行链接。此库提供对常驻代码中 ThreadX 服务的访问权限。大多数访问权限通过 txm_*.c 文件来完成。下面是 ThreadX API 函数 tx_thread_relinquish 的模块访问调用示例（在 txm_thread_relinquish.c 中）* *_***。

```
(_txm_module_kernel_call_dispatcher)(TXM_THREAD_RELINQUISH_CALL, 0, 0, 0);
```

在此示例中，模块管理器提供的函数指针可用于调用模块管理器调度函数，该函数的 ID 与 tx_thread_relinquish 服务相关联。此服务不采用任何参数。

模块示例

下面是一个模块形式的标准 ThreadX 演示示例。标准 ThreadX 演示与模块演示之间的主要区别包括：

1. 用 txm_module.h 替换 tx_api.h*_
2. 在 txm_module.h 之前添加 #define TXM_MODULE
3. 用 demo_module_start 替换 main 和 tx_application_define
4. 声明指向 ThreadX 对象的指针，而不是对象本身。

```
/* Specify that this is a module! */
#define TXM_MODULE

/* Include the ThreadX module header. */
#include "txm_module.h"

/* Define constants. */
#define DEMO_STACK_SIZE         1024
#define DEMO_BYTE_POOL_SIZE     9120
```

```
#define DEMO_BYTE_POOL_SIZE     9120
#define DEMO_BLOCK_POOL_SIZE    100
#define DEMO_QUEUE_SIZE         100

/* Define the pool space in the bss section of the module. ULONG is used to
   get word alignment. */

ULONG                   demo_module_pool_space[DEMO_BYTE_POOL_SIZE / 4];

/* Define the ThreadX object control block pointers. */

TX_THREAD               *thread_0;
TX_THREAD               *thread_1;
TX_THREAD               *thread_2;
TX_THREAD               *thread_3;
TX_THREAD               *thread_4;
TX_THREAD               *thread_5;
TX_THREAD               *thread_6;
TX_THREAD               *thread_7;
TX_QUEUE                *queue_0;
TX_SEMAPHORE            *semaphore_0;
TX_MUTEX                *mutex_0;
TX_EVENT_FLAGS_GROUP    *event_flags_0;
TX_BYTE_POOL            *byte_pool_0;
TX_BLOCK_POOL           *block_pool_0;


/* Define the counters used in the demo application. */
ULONG       thread_0_counter;
ULONG       thread_1_counter;
ULONG       thread_1_messages_sent;
ULONG       thread_2_counter;
ULONG       thread_2_messages_received;
ULONG       thread_3_counter;
ULONG       thread_4_counter;
ULONG       thread_5_counter;
ULONG       thread_6_counter;
ULONG       thread_7_counter;
ULONG       semaphore_0_puts;
ULONG       event_0_sets;
ULONG       queue_0_sends;

/* 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);

/* Define notify functions. */

void semaphore_0_notify(TX_SEMAPHORE *semaphore_ptr)
{
    if (semaphore_ptr == semaphore_0)
        semaphore_0_puts++;
}


void event_0_notify(TX_EVENT_FLAGS_GROUP *event_flag_group_ptr)
{
    if (event_flag_group_ptr == event_flags_0)
        event_0_sets++;
}


void queue_0_notify(TX_QUEUE *queue_ptr)
{
```

```
    if (queue_ptr == queue_0)
        queue_0_sends++;
}

/* Define the module start function. */
void demo_module_start(ULONG id)
{
    CHAR *pointer;

    /* Allocate all the objects. In memory protection mode,
        modules cannot allocate control blocks within their
        own memory area so they cannot corrupt the resident
        portion of ThreadX by corrupting the control block(s). */
    txm_module_object_allocate(&thread_0, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_1, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_2, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_3, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_4, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_5, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_6, sizeof(TX_THREAD));
    txm_module_object_allocate(&thread_7, sizeof(TX_THREAD));
    txm_module_object_allocate(&queue_0, sizeof(TX_QUEUE));
    txm_module_object_allocate(&semaphore_0, sizeof(TX_SEMAPHORE));
    txm_module_object_allocate(&mutex_0, sizeof(TX_MUTEX));
    txm_module_object_allocate(&event_flags_0, sizeof(TX_EVENT_FLAGS_GROUP));
    txm_module_object_allocate(&byte_pool_0, sizeof(TX_BYTE_POOL));
    txm_module_object_allocate(&block_pool_0, sizeof(TX_BLOCK_POOL));

    /* Create a byte memory pool from which to allocate the thread stacks. */
    tx_byte_pool_create(byte_pool_0, "module byte pool 0",
        demo_module_pool_space, DEMO_BYTE_POOL_SIZE);

    /* Allocate the stack for thread 0. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);

    /* Create thread 0. */
    tx_thread_create(thread_0, "module thread 0", thread_0_entry, 0,
        pointer, DEMO_STACK_SIZE,
        1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the stack for thread 1. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);

    /* Create threads 1 and 2. These threads pass information through
        a ThreadX message queue. It is also interesting to note that
        these threads have a time slice. */
    tx_thread_create(thread_1, "module thread 1", thread_1_entry, 1,
        pointer, DEMO_STACK_SIZE,
        16, 16, 4, TX_AUTO_START);

    /* Allocate the stack and create thread 2. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);
    tx_thread_create(thread_2, "module thread 2", thread_2_entry, 2,
        pointer, DEMO_STACK_SIZE,
        16, 16, 4, TX_AUTO_START);

    /* Allocate the stack for thread 3. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);

    /* Create threads 3 and 4. These threads compete for a ThreadX
        counting semaphore. An interesting thing here is that both threads
        share the same instruction area. */
    tx_thread_create(thread_3, "module thread 3",
        thread_3_and_4_entry, 3,
        pointer, DEMO_STACK_SIZE,
```

```
        8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the stack and create thread 4. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);
    tx_thread_create(thread_4, "module thread 4",
        thread_3_and_4_entry, 4,
        pointer, DEMO_STACK_SIZE,
        8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the stack for thread 5. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);

    /* Create thread 5. This thread simply pends on an event flag which
        will be set by thread 0. */
    tx_thread_create(thread_5, "module thread 5", thread_5_entry, 5,
        pointer, DEMO_STACK_SIZE,
        4, 4, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the stack for thread 6. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);

    /* Create threads 6 and 7. These threads compete for a ThreadX mutex. */
    tx_thread_create(thread_6, "module thread 6",
        thread_6_and_7_entry, 6,
        pointer, DEMO_STACK_SIZE,
        8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the stack and create thread 7. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_STACK_SIZE, TX_NO_WAIT);
    tx_thread_create(thread_7, "module thread 7",
        thread_6_and_7_entry, 7,
        pointer, DEMO_STACK_SIZE,
        8, 8, TX_NO_TIME_SLICE, TX_AUTO_START);

    /* Allocate the message queue. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_QUEUE_SIZE*sizeof(ULONG), TX_NO_WAIT);

    /* Create the message queue shared by threads 1 and 2. */
    tx_queue_create(queue_0, "module queue 0", TX_1_ULONG, pointer,
        DEMO_QUEUE_SIZE*sizeof(ULONG));

    /* Register queue send callback. */
    tx_queue_send_notify(queue_0, queue_0_notify);

    /* Create the semaphore used by threads 3 and 4. */
    tx_semaphore_create(semaphore_0, "module semaphore 0", 1);

    /* Register semaphore put callback. */
    tx_semaphore_put_notify(semaphore_0, semaphore_0_notify);

    /* Create the event flags group used by threads 1 and 5. */
    tx_event_flags_create(event_flags_0, "module event flags 0");

    /* Register event flag set callback. */
    tx_event_flags_set_notify(event_flags_0, event_0_notify);

    /* Create the mutex used by thread 6 and 7 without priority
        inheritance. */
    tx_mutex_create(mutex_0, "module mutex 0", TX_NO_INHERIT);

    /* Allocate the memory for a small block pool. */
    tx_byte_allocate(byte_pool_0, (VOID **) &pointer,
        DEMO_BLOCK_POOL_SIZE, TX_NO_WAIT);
```

```
    /* Create a block memory pool. */
    tx_block_pool_create(block_pool_0, "module block pool 0",
        sizeof(ULONG), pointer, DEMO_BLOCK_POOL_SIZE);

    /* Allocate a block. */
    tx_block_allocate(block_pool_0, (VOID **) &pointer,
        TX_NO_WAIT);

    /* Release the block back to the pool. */
    tx_block_release(pointer);

}

/* Define all the threads. */

void thread_0_entry(ULONG thread_input)
{
    UINT status;

    /* This thread simply sits in while-forever-sleep loop. */
    while(1)
    {
        /* Increment the thread counter. */
        thread_0_counter++;

        /* Sleep for 10 ticks. */
        tx_thread_sleep(10);

        /* Set event flag 0 to wake up thread 5. */
        status = tx_event_flags_set(event_flags_0, 0x1, TX_OR);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;
    }
}

void thread_1_entry(ULONG thread_input)
{
    UINT status;

    /* This thread simply sends messages to a queue shared by
       thread 2. */
    while(1)
    {
        /* Increment the thread counter. */
        thread_1_counter++;

        /* Send message to queue 0. */
        status = tx_queue_send(queue_0, &thread_1_messages_sent,
            TX_WAIT_FOREVER);

        /* Check completion status. */
        if (status != TX_SUCCESS)
            break;

        /* Increment the message sent. */
        thread_1_messages_sent++;
    }
}

void thread_2_entry(ULONG thread_input)
{
    ULONG received_message;
    UINT status;

    /* This thread retrieves messages placed on the queue by thread 1. */
    while(1)
    {
```

```
        /* Increment the thread counter. */
        thread_2_counter++;

        /* Retrieve a message from the queue. */
        status = tx_queue_receive(queue_0, &received_message, TX_WAIT_FOREVER);

        /* Check completion status and make sure the message is what
           we expected. */
        if ((status != TX_SUCCESS) || (received_message != thread_2_messages_received))
            break;

        /* Otherwise, all is okay. Increment the received message count. */
        thread_2_messages_received++;
    }
}

void thread_3_and_4_entry(ULONG thread_input)
{
    UINT status;

    /* This function is executed from thread 3 and thread 4. As the loop
       below shows, these function compete for ownership of semaphore_0. */
    while(1)
    {
        /* Increment the thread counter. */
        if (thread_input == 3)
            thread_3_counter++;
        else
            thread_4_counter++;

        /* Get the semaphore with suspension. */
        status = tx_semaphore_get(semaphore_0, TX_WAIT_FOREVER);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;

        /* Sleep for 2 ticks to hold the semaphore. */
        tx_thread_sleep(2);

        /* Release the semaphore. */
        status = tx_semaphore_put(semaphore_0);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;
    }
}

void thread_5_entry(ULONG thread_input)
{
    UINT status;
    ULONG actual_flags;

    /* This thread simply waits for an event in a forever loop. */
    while(1)
    {
        /* Increment the thread counter. */
        thread_5_counter++;

        /* Wait for event flag 0. */
        status = tx_event_flags_get(event_flags_0, 0x1, TX_OR_CLEAR,
                                    &actual_flags, TX_WAIT_FOREVER);

        /* Check status. */
        if ((status != TX_SUCCESS) || (actual_flags != 0x1))
            break;
    }
}
```

```
}

void thread_6_and_7_entry(ULONG thread_input)
{
    UINT status;

    /* This function is executed from thread 6 and thread 7. As the loop
       below shows, these function compete for ownership of mutex_0. */
    while(1)
    {
        /* Increment the thread counter. */
        if (thread_input == 6)
            thread_6_counter++;
        else
            thread_7_counter++;

        /* Get the mutex with suspension. */
        status = tx_mutex_get(mutex_0, TX_WAIT_FOREVER);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;

        /* Get the mutex again with suspension. This shows that an
           owning thread may retrieve the mutex it owns multiple times. */
        status = tx_mutex_get(mutex_0, TX_WAIT_FOREVER);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;

        /* Sleep for 2 ticks to hold the mutex. */
        tx_thread_sleep(2);

        /* Release the mutex. */
        status = tx_mutex_put(mutex_0);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;

        /* Release the mutex again. This will actually release ownership
           since it was obtained twice. */
        status = tx_mutex_put(mutex_0);

        /* Check status. */
        if (status != TX_SUCCESS)
            break;
    }
}
```

生成模块

生成模块取决于所使用的工具链。有关特定于端口的示例，请参阅附录。所有端口的常见活动包括以下各项。

- 生成模块库
- 生成示例应用程序

每个模块都需要 txm_module_preamble（专用于模块的安装程序）和模块库（例如 txm.a）。

第 3 章 - 模块管理器要求

2021/4/30 •

ThreadX 模块管理器与 ThreadX RTOS 一起驻留在应用程序的常驻部分中。它负责启动模块，以及处理和调度 ThreadX API 服务的所有模块请求。

> **NOTE**
> ThreadX 模块管理器源文件（C 和程序集）应添加到 ThreadX 库项目“tx”中。

要生成 ThreadX 模块管理器，需要执行以下步骤（下面更详细介绍了每个步骤）。

1. 必须扩展 TX_THREAD 控制块，以包含模块信息。完成此操作的最简单方法是将 tx_port.h 文件中的 TX_THREAD_EXTENSION_2 的定义替换为 txm_module_port.h 中找到的 TX_THREAD_EXTENSION_2。_ _ 有关特定于端口的扩展，请参阅附录。

示例扩展：

```
#define TX_THREAD_EXTENSION_2                   \
    VOID      tx_thread_module_instance_ptr;    \
    VOID      tx_thread_module_entry_info_ptr;  \
    ULONG     tx_thread_module_current_user_mode; \
    ULONG     tx_thread_module_user_mode;       \
    VOID     *tx_thread_module_kernel_stack_start;\
    VOID     *tx_thread_module_kernel_stack_end;  \
    ULONG     tx_thread_module_kernel_stack_size; \
    VOID     *tx_thread_module_stack_ptr;       \
    VOID     *tx_thread_module_stack_start;     \
    VOID     *tx_thread_module_stack_end;       \
    ULONG     tx_thread_module_stack_size;      \
    VOID     *tx_thread_module_reserved;
```

必须在 tx_port.h 中定义以下扩展。

```
#define TX_EVENT_FLAGS_GROUP_EXTENSION  VOID    *tx_event_flags_group_module_instance; \
     VOID   (*tx_event_flags_group_set_module_notify)(struct TX_EVENT_FLAGS_GROUP_STRUCT *group_ptr);

#define TX_QUEUE_EXTENSION              VOID    *tx_queue_module_instance; \
     VOID   (*tx_queue_send_module_notify)(struct TX_QUEUE_STRUCT *queue_ptr);

#define TX_SEMAPHORE_EXTENSION          VOID    *tx_semaphore_module_instance; \
     VOID   (*tx_semaphore_put_module_notify)(struct TX_SEMAPHORE_STRUCT *semaphore_ptr);

#define TX_TIMER_EXTENSION              VOID    *tx_timer_module_instance; \
     VOID   (*tx_timer_module_expiration_function)(ULONG id);
```

2. 将所有 txm_module_manager_ C 文件和程序集文件添加到 ThreadX 库项目 tx 中。*
3. 重新生成所有库和可执行项目。如果需要 NetX Duo，则应定义 TXM_MODULE_ENABLE_NETX_DUO 来生成所有模块 C 代码和模块管理器 C 代码。

模块管理器源

ThreadX 模块管理器有一组源文件，它们设计为与常驻 ThreadX 代码直接链接并与其位于一起。这些文件提供了启动模块以及处理来自模块的所有后续 ThreadX API 请求的功能。模块管理器文件如下所示。

ttt	CONTENTS
txm_module.h	包含定义模块信息(这些信息也包含在模块源代码中)的文件。
txm_module_manager_dispatch.h	包含定义调度帮助程序函数的文件。
txm_module_manager_util.h	包含定义内部实用工具帮助程序宏和函数的文件。
txm_module_port.h	包含定义特定于端口的模块信息(这些信息也包含在模块源代码中)的文件。
tx_initialize_low_level.s,S,68* [] _	替换现有 ThreadX 库文件。更新了模块管理器和内存异常的向量表和其他向量处理程序。此文件仅存在于 Cortex-A7/ARM、Cortex-M7/ARM、Cortex-R4/ARM、Cortex-R4/IAR、MCF544xx/GHS、RX63/IAR 和 RX65N/IAR 中。_*
tx_thread_context_restore.s*_	替换现有 ThreadX 库文件。在中断处理后还原线程上下文。此文件仅存在于 Cortex-A7/ARM、Cortex-R4/ARM 和 Cortex-R4/IAR 中。_*
tx_thread_schedule.s,S,68 *[]*	替换现有 ThreadX 库文件。扩展了计划程序代码，在此示例中用于更新内存管理寄存器。
tx_thread_stack_build.s*_	替换现有 ThreadX 库文件。生成线程的堆栈帧。此文件仅存在于 Cortex-A7/ARM、Cortex-R4/ARM 和 Cortex-R4/IAR 中。_*
txm_module_manager_thread_stack_build.s,S,68 *[]*	生成所有模块初始堆栈，包含与位置无关的数据访问的设置。
txm_module_manager_user_mode_entry.s,S* [] _	允许模块进入内核模式。此文件仅存在于 Cortex-A7/ARM、Cortex-R4/ARM 和 Cortex-R4/IAR 中。_*
txm_module_manager_alignment_adjust.c	处理特定于端口的对齐要求。
txm_module_manager_application_request.c	处理对常驻代码的特定于应用程序的请求。
txm_module_manager_callback_request.c	向模块发送回调请求。
txm_module_manager_event_flags_notify_trampoline.c	处理来自 ThreadX 的事件标志设置通知调用。
txm_module_manager_external_memory_enable.c	在内存管理表中为模块可访问的共享内存空间创建一个条目。
txm_module_manager_file_load.c	分配一个二进制模块文件并将其加载到模块内存区域中，并为执行该文件做好准备。
txm_module_manager_in_place_load.c	分配模块数据区域，并为从提供的代码地址执行模块做好准备。
txm_module_manager_initialize.c	初始化模块管理器，包括可用于加载和运行模块的模块内存区域的规范。
txm_module_manager_initialize_mmu.c*_	初始化 MMU。用户可以根据其内存映射编辑此文件。此文件仅存在于 Cortex-A7/ARM 中。_*

ttt	CONTENTS
txm_module_manager_mm_initialize.c*_	初始化 MPU/MMU。用户可以根据其内存映射编辑此文件。此文件仅存在于 Cortex-A7/ARM 中。_*
txm_module_manager_kernel_dispatch.c	根据请求 ID 处理模块 API 请求。
txm_module_manager_maximum_module_priority_set.c	设置模块中允许的最高线程优先级。
txm_module_manager_memory_fault_handler.c	处理在正在执行的模块中检测到的内存错误。
txm_module_manager_memory_fault_notify.c	在每次出现内存错误时注册应用程序通知回调。
txm_module_manager_memory_load.c	分配并加载模块的代码和数据，并为执行该模块做好准备。
txm_module_manager_mm_register_setup.c	根据代码和数据加载到的位置为模块设置 MPU/MMU 寄存器。
txm_module_manager_object_allocate.c	为模块对象分配内存。
txm_module_manager_object_deallocate.c	解除分配模块对象的内存。
txm_module_manager_object_pointer_get.c	搜索提供的对象类型和名称，如果找到，则返回对象指针。
txm_module_manager_object_pointer_get_extended.c	搜索提供的对象类型和名称，如果找到，则返回对象指针。出于安全考虑而指定的名称长度。
txm_module_manager_object_pool_create.c	在模块的数据区域之外创建一个对象池，使应用程序可以从中分配对象。
txm_module_manager_properties_get.c	获取指定模块的属性。
txm_module_manager_queue_notify_trampoline.c	处理来自 ThreadX 的队列通知调用。
txm_module_manager_semaphore_notify_trampoline.c	处理来自 ThreadX 的信号灯放置通知调用。
txm_module_manager_start.c	启动模块执行。
txm_module_manager_stop.c	停止模块执行。
txm_module_manager_thread_create.c	创建所有模块线程。
txm_module_manager_thread_notify_trampoline.c	处理来自 ThreadX 的线程进入/退出通知调用。
txm_module_manager_thread_reset.c	重置模块线程。
txm_module_manager_timer_notify_trampoline.c	处理 ThreadX 中的计时器过期。
txm_module_manager_unload.c	从模块内存区域卸载模块。
txm_module_manager_util.c	管理器的内部帮助程序函数。

模块管理器初始化

应用程序的常驻部分负责调用模块管理器初始化函数 txm_module_manager_initialize。此函数可设置用于加载和卸载模块的内部结构，包括设置用于分配模块内存的内存区域。

模块管理器加载

模块管理器可以从二进制模块文件或已存在于常驻代码区域中的模块代码节，将模块动态加载到模块内存中。此外，模块管理器还可以就地执行代码，也就是说，仅在模块内存中分配模块数据，并且就地执行代码。以下模块管理器加载 API 函数可用。

- txm_module_manager_file_load
- txm_module_manager_in_place_load
- txm_module_manager_memory_load

模块管理器的内存保护版本还可以确保以适当的对齐方式加载模块，并为每个模块正确设置内存管理寄存器。通过模块报头选项启用内存保护时，模块内存访问仅限于模块代码和数据区域。

模块管理器启动

模块管理器通过 txm_module_manager_start API 函数启动先前加载的模块的执行。为了启动模块执行，此函数将创建一个线程，该线程将从模块报头中指定的起始位置处进入模块。模块报头中还指定了此线程的优先级和堆栈大小。

模块管理器停止

模块管理器通过 txm_module_manager_stop 函数终止先前加载并且正在执行的模块的执行。此 API 函数首先终止并删除初始启动线程。如果模块报头指定了停止线程，则会创建并执行此线程。模块管理器将等待一段固定的时间，等待停止线程完成。完成后，将删除该模块创建的所有系统资源，并将该模块置于休眠状态，在此状态下，可以重启或卸载该模块。

模块管理器卸载

模块管理器通过 txm_module_manager_unload 函数卸载先前加载但未在执行的模块。此 API 可释放与模块关联的所有内存，使这些内存在将来可用于其他模块。

模块管理器请求

模块对模块管理器发出的请求通过 txm_module.h 中的宏完成，这些宏映射所有 ThreadX 调用，以通过模块管理器提供给模块的函数指针调用模块管理器调度函数。

由模块通过调用 txm_module_application_request 进行的其他特定于应用程序的服务由用于 ThreadX API 的同一宏机制处理。默认情况下，模块管理器中的此处理函数为空，此设计可让应用程序添加所需的代码来处理特定于应用程序的请求。

如果请求未由模块管理器实现，则模块管理器将返回值为 TX_NOT_AVAILABLE 的错误状态。如果模块请求的操作超出其访问范围，也会返回此错误代码。例如，不允许模块使用其代码区域之外的计时器控制块或回调地址创建计时器。

模块管理器示例

下面是模块管理器代码的示例，此示例代码可启动之前在第 2 章中定义的示例模块。假定已在 ROM 地址 0x00800000 处加载了该模块（可能由调试器加载）。

```
#include "tx_api.h"
#include "txm_module.h"

#define DEMO_STACK_SIZE 1024

/* Define the ThreadX object control blocks. */
TX_THREAD   module_manager;

/* Define thread prototype. */
void        module_manager_entry(ULONG thread_input);

/* Define the module object pool area. */
UCHAR       object_memory[8192];

/* Define the module data pool area. */
#define MODULE_DATA_SIZE 65536
UCHAR       module_data_area[MODULE_DATA_SIZE];

/* Define module instances. */
TXM_MODULE_INSTANCE     my_module1;
TXM_MODULE_INSTANCE     my_module2;

/* Define the count of memory faults. */
ULONG memory_faults;

/* Define fault handler. */
VOID module_fault_handler(TX_THREAD *thread, TXM_MODULE_INSTANCE *module)
{
    /* Just increment the fault counter. */
    memory_faults++;
}

/* Define main entry point. */
int main()
{
    /* Enter the ThreadX kernel. */
    tx_kernel_enter();
}

/* Define what the initial system looks like. */
void tx_application_define(void *first_unused_memory)
{
    /* Create the module manager thread. */
    tx_thread_create(&module_manager, "Module Manager Thread", module_manager_entry, 0,
                     first_unused_memory, DEMO_STACK_SIZE,
                     1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
}

/* Define the test threads. */
void module_manager_entry(ULONG thread_input)
{
    /* Initialize the module manager. */
    txm_module_manager_initialize((VOID *) module_data_area, MODULE_DATA_SIZE);

    /* Create a pool for module objects. */
    txm_module_manager_object_pool_create(object_memory, sizeof(object_memory));

    /* Register a fault handler. */
    txm_module_manager_memory_fault_notify(module_fault_handler);

    /* Load the module that is already there,
        in this example it is placed at 0x00800000. */
    txm_module_manager_in_place_load(&my_module1, "my module1", (VOID *) 0x00800000);

    /* Load a second instance of the module. */
    txm_module_manager_in_place_load(&my_module2, "my module2", (VOID *) 0x00800000);

    /* Enable shared memory region for module2. */
    txm_module_manager_external_memory_enable(&my_module2, (void*)0x20600000, 0x10000, 0x3F);
```

```
    /* Start the modules. */
    txm_module_manager_start(&my_module1);
    txm_module_manager_start(&my_module2);

    /* Sleep for a while and let the modules run... */
    tx_thread_sleep(300);

    /* Stop the modules. */
    txm_module_manager_stop(&my_module1);
    txm_module_manager_stop(&my_module2);

    /* Unload the modules. */
    txm_module_manager_unload(&my_module1);
    txm_module_manager_unload(&my_module2);

    /* Reload the modules. */
    txm_module_manager_in_place_load(&my_module2, "my module2", (VOID *) 0x00800000);
    txm_module_manager_in_place_load(&my_module1, "my module1", (VOID *) 0x00800000);

    /* Give both modules shared memory. */
    txm_module_manager_external_memory_enable(&my_module2, (void*)0x20600000, 0x010000, 0x3F);
    txm_module_manager_external_memory_enable(&my_module1, (void*)0x20600000, 0x010000, 0x3F);

    /* Set maximum module1 priority to 5. */
    txm_module_manager_maximum_module_priority_set(&my_module1, 5);

    /* Start the modules again. */
    txm_module_manager_start(&my_module2);
    txm_module_manager_start(&my_module1);

    /* Now just spin... */
    while(1)
    {
        tx_thread_sleep(100);

        /* Threads 0 and 5 in module1 are not created because they violate the maximum priority. */
    }
}
```

模块管理器生成

必须将 txm_module_manager 源文件添加到 ThreadX 库中。_*

ThreadX 模块管理器应用程序实际上与标准 ThreadX 应用程序相同，后者是与 ThreadX 库 tx.a 链接在一起的一个或多个应用程序文件。如何生成模块管理器应用程序取决于所使用的工具链。有关特定于端口的示例，请参阅附录。

第 4 章 - 模块 API

2021/4/29 •

模块 API 摘要

下面还有一些可用于模块的其他 API：

- ****txm_module_application_request** - 对常驻代码发出特定于应用程序的请求
- ****txm_module_object_allocate** - 在模块外部为对象分配内存
- ****txm_module_object_deallocate** - 解除先前的对象内存分配
- ****txm_module_object_pointer_get** - 查找系统对象并检索对象指针
- ****txm_module_object_pointer_get_extended** - 查找系统对象并检索对象指针（名称长度安全）

返回值

对于某些 Azure RTOS API，会返回其他错误代码。其他的错误代码定义如下：

- **TXM_MODULE_INVALID_PROPERTIES** (0xF3)：指示模块没有正确的属性，无法进行 API 调用。例如，在用户模式下调用跟踪 API。
- **TXM_MODULE_INVALID_MEMORY** (0xF4)：指示模块提供的内存无效或位于无效的位置。例如在内存保护模块中，对象控制块不可以位于模块可访问的内存中。
- **TXM_MODULE_INVALID_CALLBACK** (0xF5)：API 中指定的回调超出了模块代码的范围，因此无效。

---

txm_module_application_request

对常驻代码发出特定于应用程序的请求。

原型

```
UINT txm_module_application_request(
    ULONG request,
    ULONG param_1,
    ULONG param_2,
    ULONG param_3);
```

说明

此服务向应用程序的常驻部分发出指定请求。假设在调用之前已准备好请求结构。请求的实际处理过程发生在函数 _txm_module_manager_application_request 的常驻代码中。默认情况下，此函数留空，专供常驻应用程序开发人员修改。

输入参数

- **request**：请求 ID（由应用程序定义）
- **param_1**：第一个参数
- **param_2**：第二个参数
- **param_3**：第三个参数

返回值

- **TX_SUCCESS** (0x00)：成功的请求。
- **TX_NOT_AVAILABLE** (0x1D)：请求不受常驻代码支持。

允许来自

模块线程

示例

```
/* Call application resident code with ID=77 and the
   parameters set to 1, 2, 3. */
status = txm_module_application_request(77, 1, 2, 3);

/* If status is TX_SUCCESS the request was successful. */
```

---

txm_module_object_allocate

在对象池(由常驻应用程序创建)中为模块对象控制块分配内存。

原型

```
UINT txm_module_object_allocate(
   VOID **object_ptr,
   ULONG object_size);
```

说明

此服务从模块外部的内存中为模块对象分配内存，这有助于防止模块代码破坏对象控制块。在内存保护系统中，必须先通过此 API 分配所有对象控制块，然后才能创建它们。

输入参数

- **object_ptr**：成功分配时对象指针的目标。
- **object_size**：要分配的对象的大小(以字节为单位)。

返回值

- **TX_SUCCESS** (0x00)：成功分配对象。
- **TX_NO_MEMORY** (0x10)：内存不足。
- **TX_NOT_AVAILABLE** (0x1D)：模块管理器尚未创建要从中进行分配的对象池

允许来自

模块线程

示例

```
TX_QUEUE *queue_pointer;

/* Allocate a control block for a module message queue. */
status = txm_module_object_allocate(&queue_pointer, sizeof(TX_QUEUE));

/* If status is TX_SUCCESS the queue_pointer points to
   memory allocated outside of the module and can be supplied
   to tx_queue_create to create a queue for the module. */
```

另请参阅

- txm_module_object_deallocate
- txm_module_object_pointer_get

---

txm_module_object_deallocate

解除先前的对象内存分配

原型

```
UINT txm_module_object_deallocate(VOID *object_ptr);
```

说明

此服务不再需要，已被弃用。

先前通过 txm_module_object_allocate 分配的内存在 tx_delete 服务中解除分配。

输入参数

- **object_ptr**：要解除分配的对象指针。

返回值

- **TX_SUCCESS** (0x00)：成功分配对象。

允许来自

模块线程

示例

```
TX_QUEUE *queue_pointer;

/* Deallocate control block for a module message queue. */
status = txm_module_object_deallocate(queue_pointer);

/* If status is TX_SUCCESS the object memory associated
   with queue_pointer is deallocated. */
```

另请参阅

- txm_module_object_allocate
- txm_module_object_pointer_get

---

txm_module_object_pointer_get

查找系统对象并检索对象指针

原型

```
UINT txm_module_object_pointer_get(
   UINT object_type, CHAR *name,
   VOID **object_ptr);
```

说明

此服务检索具有特定名称的特定类型的对象指针。如果找不到该对象，将返回错误。否则，如果找到该对象，则该对象的地址将位于“object_ptr”中。然后，可以使用此指针来进行系统服务调用，与系统中的常驻代码和/或其他已加载的模块交互。

输入参数

- **object_type**：请求的 ThreadX 对象的类型。有效类型如下：
  - TXM_BLOCK_POOL_OBJECT
  - TXM_BYTE_POOL_OBJECT
  - TXM_EVENT_FLAGS_OBJECT

- TXM_MUTEX_OBJECT
- TXM_QUEUE_OBJECT
- TXM_SEMAPHORE_OBJECT
- TXM_THREAD_OBJECT
- TXM_TIMER_OBJECT
- TXM_IP_OBJECT
- TXM_PACKET_POOL_OBJECT
- TXM_UDP_SOCKET_OBJECT
- TXM_TCP_SOCKET_OBJECT

- **name**：创建对象时定义的特定于应用程序的对象名称。
- **object_ptr**：对象指针的目标。

返回值

- **TX_SUCCESS** (0x00)：成功获取对象。
- **TX_OPTION_ERROR** (0x08)：对象类型无效。
- **TX_PTR_ERROR** (0x03)：目标无效。
- **TX_SIZE_ERROR** (0x05)：大小无效。
- **TX_NO_INSTANCE** (0x0D)：找不到对象。

允许来自

模块线程

示例

```
TX_QUEUE *queue_pointer;

/* Find the pointer for "fft_queue" in the resident part
   of the application. */
status = txm_module_object_pointer_get(TXM_QUEUE_OBJECT,
         "fft_queue", &queue_pointer);

/* If status is TX_SUCCESS the found queue pointer is in
   "queue_pointer". This queue pointer can then be used to
   send messages to the "fft_queue." */
```

另请参阅

- txm_module_manager_object_pointer_get_extended

---

txm_module_object_pointer_get_extended

查找系统对象并检索对象指针

原型

```
UINT txm_module_object_pointer_get_extended(UINT object_type,
                                            CHAR *name,
                                            UINT name_length,
                                            VOID **object_ptr);
```

说明

此服务检索具有特定名称的特定类型的对象指针。如果找不到该对象，将返回错误。否则，如果找到该对象，则该对象的地址将位于“object_ptr”中。然后，可以使用此指针来进行系统服务调用，与系统中的常驻代码和/或其他已加载的模块交互。

输入参数

- **object_type**：请求的 ThreadX 对象的类型。有效类型如下：
  - TXM_BLOCK_POOL_OBJECT
  - TXM_BYTE_POOL_OBJECT
  - TXM_EVENT_FLAGS_OBJECT
  - TXM_MUTEX_OBJECT
  - TXM_QUEUE_OBJECT
  - TXM_SEMAPHORE_OBJECT
  - TXM_THREAD_OBJECT
  - TXM_TIMER_OBJECT
  - TXM_IP_OBJECT
  - TXM_PACKET_POOL_OBJECT
  - TXM_UDP_SOCKET_OBJECT
  - TXM_TCP_SOCKET_OBJECT
- **name**：创建对象时定义的特定于应用程序的对象名称。
- **name_length**：名称的长度。
- **object_ptr**：对象指针的目标。

返回值

- **TX_SUCCESS** (0x00)：成功获取对象。
- **TX_OPTION_ERROR** (0x08)：对象类型无效。
- **TX_PTR_ERROR** (0x03)：目标无效。
- **TX_SIZE_ERROR** (0x05)：大小无效。
- **TX_NO_INSTANCE** (0x0D)：找不到对象。

允许来自

模块线程

示例

```
TX_QUEUE *queue_pointer;

/* Find the pointer for "fft_queue" in the resident part
   of the application. */
   status = txm_module_object_pointer_get_extended(TXM_QUEUE_OBJECT,
            "fft_queue", 9, &queue_pointer);

/* If status is TX_SUCCESS the found queue pointer is in
   "queue_pointer". This queue pointer can then be used to
   send messages to the "fft_queue." */
```

另请参阅

- txm_module_manager_object_pointer_get

第 5 章 - 模块管理器 API

2021/4/29 •

模块管理器 API 摘要

下面还有一些可用于应用程序常驻部分的其他 API。

- ****txm_module_manager_external_memory_enable** - 允许模块访问共享内存空间
- ****txm_module_manager_file_load** - 通过 FileX 从文件中加载模块
- ****txm_module_manager_in_place_load** - 加载模块数据，就地执行
- ****txm_module_manager_initialize** - 初始化模块管理器
- ****txm_module_manager_mm_initialize** - 初始化内存管理硬件
- ****txm_module_manager_maximum_module_priority_set** - 设置模块中允许的最高线程优先级
- ****txm_module_manager_memory_fault_notify** - 在出现内存错误时注册应用程序回调
- ****txm_module_manager_memory_load** - 从内存中加载模块
- ****txm_module_manager_object_pool_create** - 为模块创建对象池
- ****txm_module_manager_properties_get** - 获取模块属性
- ****txm_module_manager_start** - 开始执行指定模块
- ****txm_module_manager_stop** - 停止执行指定模块
- ****txm_module_manager_unload** - 卸载模块

---

txm_module_manager_external_memory_enable

允许模块访问共享内存空间。

原型

```
UINT txm_module_manager_external_memory_enable(
     TXM_MODULE_INSTANCE *module_instance,
     VOID *start_address,
     ULONG length,
     UINT attributes);
```

说明

此服务在内存管理硬件表中为该模块可访问的共享内存区域创建一个条目。

输入参数

- **module_instance**：指向模块实例的指针。
- **start_address**：共享内存区域的起始地址。
- **length**：共享内存区域的长度。
- **attributes**：内存区域的属性（缓存、读取、写入等）。属性特定于端口；请参阅附录了解属性格式。

返回值

- **TX_SUCCESS** (0x00)：成功创建条目。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化或功能不可用。
- **TX_PTR_ERROR** (0x03)：模块实例无效。
- **TX_START_ERROR** (0x10)：模块未处于已加载状态。
- **TXM_MODULE_ALIGNMENT_ERROR** (0xF0)：起始地址对齐方式无效。

- TXM_MODULE_INVALID_PROPERTIES (0xF3)：属性不兼容。

允许来自

初始化和线程

示例

```
TXM_MODULE_INSTANCE my_module;

/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Load the module that has its code area at address 0x080F0000. */
txm_module_manager_in_place_load(&my_module, "my module", (VOID *) 0x080F0000);

/* Create a shared memory space 256 bytes long at address 0x64005000
   with read & write, no execute, outer & inner write back cache
   attributes. Note that these attributes are port-specific. */
txm_module_manager_external_memory_enable(&my_module, (VOID*)0x64005000, 256, 0x3F);
```

---

txm_module_manager_file_load

通过 FileX 从文件中加载模块。

原型

```
UINT txm_module_manager_file_load(
    TXM_MODULE_INSTANCE *module_instance,
    CHAR *module_name,
    FX_MEDIA *media_ptr,
    CHAR *file_name);
```

说明

此服务将指定文件中包含的模块的二进制映像加载到模块内存区域中，并为执行该映像做好准备。假定提供的媒体已经打开。

> **NOTE**
> 请使用 FileX 系统加载文件。若要启用 FileX 访问权限，必须使用项目中定义的 FX_FILEX_PRESENT 生成模块、模块库、模块管理器和 ThreadX 库（带有模块管理器源）。

输入参数

- module_instance：指向模块实例的指针。
- module_name：模块的名称。
- media_ptr：指向已打开的 FileX 媒体的指针。
- file_name：模块的二进制文件的名称。

返回值

- TX_SUCCESS (0x00)：成功加载模块。
- TX_CALLER_ERROR (0x13)：调用方无效。
- TX_NOT_AVAILABLE (0x1D)：管理器未初始化。
- TX_NO_MEMORY (0x10)：内存不足，无法加载模块。
- TX_NOT_DONE (0x20)：媒体未打开，找不到文件或文件无效。
- TX_PTR_ERROR (0x03)：模块指针无效。

- **TXM_MODULE_ALIGNMENT_ERROR** (0xF0)：对齐方式无效。
- **TXM_MODULE_ALREADY_LOADED** (0xF1)：已加载模块。
- **TXM_MODULE_INVALID** (0xF2)：模块报头无效。
- **TXM_MODULE_INVALID_PROPERTIES** (0xF3)：属性不兼容。

允许来自

线程

示例

```
TXM_MODULE_INSTANCE my_module;

/* Initialize the module manager. */
status = txm_module_manager_initialize((VOID*)0x64010000,0x10000);

/* Load the module from a binary file. */
status = txm_module_manager_file_load(&my_module, "my module",
                                      &sdio_disk, "demo_thread_module.bin");

/* Start the module. */
status = txm_module_manager_start(&my_module);
```

另请参阅

- txm_module_manager_in_place_load
- txm_module_manager_memory_load
- txm_module_manager_unload

---

txm_module_manager_in_place_load

仅加载模块数据，在现有位置执行模块。

原型

```
UINT txm_module_manager_in_place_load(
    TXM_MODULE_INSTANCE *module_instance,
    CHAR *module_name,
    VOID *location);
```

说明

此服务仅将模块的数据区域加载到模块内存区域中，并为执行这些数据做好准备。模块代码将就地执行，即从模块报头在提供位置指定的地址偏移量开始执行。

输入参数

- **module_instance**：指向模块实例的指针。
- **module_name**：模块的名称。
- **location**：指向模块代码区域的指针，报头优先。

返回值

- **TX_SUCCESS** (0x00)：成功加载模块。
- **TX_CALLER_ERROR** (0x13)：调用方无效。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_NO_MEMORY** (0x10)：内存不足，无法加载模块。
- **TX_PTR_ERROR** (0x03)：指针、模块实例或模块报头无效。
- **TXM_MODULE_ALIGNMENT_ERROR** (0xF0)：对齐方式无效。

- **TXM_MODULE_ALREADY_LOADED** (0xF1)：已加载模块。
- **TXM_MODULE_INVALID** (0xF2)：模块报头无效。
- **TXM_MODULE_INVALID_PROPERTIES** (0xF3)：属性不兼容。

允许来自

初始化和线程

示例

```
TXM_MODULE_INSTANCE my_module;

/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Load the module that has its code area at address 0x080F0000. */
txm_module_manager_in_place_load(&my_module, "my module", (VOID *) 0x080F0000);

/* Start the module. */
txm_module_manager_start(&my_module);
```

另请参阅

- txm_module_manager_file_load
- txm_module_manager_memory_load
- txm_module_manager_unload

---

txm_module_manager_initialize

初始化模块管理器。

原型

```
UINT txm_module_manager_initialize(
    VOID *module_memory_start,
    ULONG module_memory_size);
```

说明

此服务初始化模块管理器的内部资源，包括用于加载模块的内存区域。

输入参数

- **module_memory_start**：指向模块内存开头的指针。
- **module_memory_size**：模块内存的大小（以字节为单位）。

返回值

- **TX_SUCCESS** (0x00)：初始化成功。
- **TX_CALLER_ERROR** (0x13)：调用方无效。

允许来自

初始化和线程

示例

```
/* Initialize the module manager with 64KB of RAM starting at
   address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);
```

另请参阅

- txm_module_manager_object_pool_create

---

txm_module_manager_initialize_mmu

初始化内存管理硬件。

原型

```
UINT txm_module_manager_initialize_mmu(VOID);
```

说明

此服务初始化 MMU。

输入参数

无

返回值

- **TX_SUCCESS** (0x00)：初始化成功。
- **TX_CALLER_ERROR** (0x13)：调用方无效。

允许来自

初始化和线程

示例

```
/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Initialize the MMU. */
txm_module_manager_initialize_mmu();
```

---

txm_module_manager_maximum_module_priority_set

设置模块中允许的最高线程优先级。

原型

```
UINT txm_module_manager_maximum_module_priority_set(
         TXM_MODULE_INSTANCE *module_instance,
         UINT priority);
```

说明

此服务设置在模块中允许的最高线程优先级。

输入参数

- **module_instance**：指向模块实例的指针。
- **priority**：最高线程优先级。

返回值

- **TX_SUCCESS** (0x00)：初始化成功。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_PTR_ERROR** (0x03)：模块实例无效。

- **TX_START_ERROR** (0x10)：模块未处于已加载状态。

允许来自

初始化和线程

示例

```
/* Load the module that has its code area at address 0x080F0000. */
txm_module_manager_in_place_load(&my_module, "my module", (VOID *) 0x080F0000);

/* Set the maximum thread priority in my_module to 5. */
txm_module_manager_maximum_module_priority_set(&my_module, 5);
```

---

txm_module_manager_memory_fault_notify

在出现内存错误时注册应用程序回调。

原型

```
UINT txm_module_manager_memory_fault_notify(
     VOID (*notify_function)(TX_THREAD *, MODULE_INSTANCE *));
```

说明

此服务向模块管理器注册指定的应用程序内存错误通知回调函数。如果出现内存错误，则会使用指向有问题的线程及其对应的模块实例的指针调用此函数。模块管理器处理过程会自动终止有问题的线程，但不会影响模块中的任何其他线程。由应用程序决定如何处理与内存错误相关的模块。

有关内存错误本身的特定信息，请参阅内部 _txm_module_manager_memory_fault_info 结构。

> **NOTE**
> 内存错误通知回调函数是直接通过内存错误异常执行的，所以只能调用中断服务例程中允许的 ThreadX API。因此，为了停止和卸载有问题的模块，应用程序通知回调必须将信号发送到应用程序任务，以便能够停止和卸载该模块。

输入参数

- **notify_function**：指向应用程序的内存错误通知回调的函数指针。提供 NULL 值将禁用内存错误通知。

返回值

- **TX_SUCCESS** (0x00)：成功注册通知函数。

允许来自

初始化和线程

示例

```
/* Register a memory fault callback. */
txm_module_manager_memory_fault_notify(my_memory_fault_handler);
```

---

txm_module_manager_memory_load

从内存中加载模块。

原型

```
UINT txm_module_manager_memory_load (
    TXM_MODULE_INSTANCE *module_instance,
    CHAR *module_name,
    VOID *location);
```

说明

此服务将模块的代码和数据区域加载到 txm_module_manager_initialize 设置的模块内存区域中，并为执行这些代码和数据做好准备。

输入参数

- **module_instance**：指向模块实例的指针。
- **module_name**：模块的名称。
- **location**：指向模块代码区域的指针，报头优先。

返回值

- **TX_SUCCESS** (0x00)：成功加载模块。
- **TX_CALLER_ERROR** (0x13)：调用方无效。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_NO_MEMORY** (0x10)：内存不足，无法加载模块。
- **TX_PTR_ERROR** (0x03)：指针、模块实例或模块报头无效。
- **TXM_MODULE_ALIGNMENT_ERROR** (0xF0)：对齐方式无效。
- **TXM_MODULE_ALREADY_LOADED** (0xF1)：已加载模块。
- **TXM_MODULE_INVALID** (0xF2)：模块报头无效。
- **TXM_MODULE_INVALID_PROPERTIES** (0xF3)：属性不兼容。

允许来自

初始化和线程

示例

```
TXM_MODULE_INSTANCE     my_module;

/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Load the module that has its code area at address 0x080F0000. */
 txm_module_manager_memory_load(&my_module, "my module", (VOID *) 0x080F0000);
```

另请参阅

- txm_module_manager_file_load
- txm_module_manager_in_place_load
- txm_module_manager_unload

---

txm_module_manager_object_pool_create

为模块创建对象池。

原型

```
UINT txm_module_manager_object_pool_create(
    VOID *pool_memory_start,
    ULONG pool_memory_size);
```

说明

此服务创建模块管理器对象内存池，模块可通过该池分配 ThreadX/NetX 对象，从而将系统对象保留在模块的内存区域之外。

输入参数

- **pool_memory_start**：指向对象内存开头的指针。
- **pool_memory_size**：对象内存池的大小（以字节为单位）。

返回值

- **TX_SUCCESS** (0x00)：初始化成功。
- **TX_CALLER_ERROR** (0x13)：调用方无效。

允许来自

初始化和线程

示例

```
TXM_MODULE_INSTANCE my_module;

/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Create an object memory pool in the next 64KB of memory. */
txm_module_manager_object_pool_create((VOID *) 0x64020000, 0x10000);
```

另请参阅

- txm_module_manager_initialize

---

txm_module_manager_properties_get

获取模块属性。

原型

```
UINT txm_module_manager_properties_get(
    TXM_MODULE_INSTANCE *module_instance,
    ULONG *module_properties_ptr);
```

说明

该服务返回模块的属性（在报头中指定）。

输入参数

- **module_instance**：指向模块实例的指针。
- **module_properties_ptr**：指向模块属性目标的指针。

返回值

- **TX_SUCCESS** (0x00)：初始化成功。
- **TX_PTR_ERROR** (0x03)：指针无效。
- **TX_CALLER_ERROR** (0x13)：调用方无效。

允许来自

线程

示例

```
TXM_MODULE_INSTANCE     my_module;
ULONG                   module_properties;

/* Initialize the module manager with 64KB of RAM starting at address 0x64010000. */
txm_module_manager_initialize((VOID *) 0x64010000, 0x10000);

/* Create an object memory pool in the next 64KB of memory. */
txm_module_manager_object_pool_create((VOID *) 0x64020000, 0x10000);

/* Load the module that has its code area at address 0x080F0000. */
txm_module_manager_in_place_load(&my_module, "my module", (VOID *) 0x080F0000);

/* Get module properties. */
txm_module_manager_properties_get(&my_module, &module_properties);
```

txm_module_manager_start

开始执行模块。

原型

```
UINT txm_module_manager_start(TXM_MODULE_INSTANCE *module_instance);
```

说明

此服务开始执行已加载的指定模块。

输入参数

- **module_instance**：指向以前加载的模块实例的指针。

返回值

- **TX_SUCCESS** (0x00)：成功启动模块。
- **TX_CALLER_ERROR** (0x13)：调用方无效。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_PTR_ERROR** (0x03)：指针或模块实例无效。
- **TX_START_ERROR** (0x10)：已启动模块。

允许来自

初始化和线程

示例

```
/* Start the module. */
txm_module_manager_start(&my_module);

/* Let the module run for a while. */
tx_thread_sleep(2000);

/* Stop the module. */
txm_module_manager_stop(&my_module);

/* Unload the module. */
txm_module_manager_unload(&my_module);
```

另请参阅

- txm_module_manager_stop

txm_module_manager_stop

停止执行模块。

原型

```
UINT txm_module_manager_stop(TXM_MODULE_INSTANCE *module_instance);
```

说明

此服务停止先前已加载并启动的模块。停止模块包括执行模块的可选停止线程、终止所有线程以及删除与模块关联的所有资源。

输入参数

- **module_instance**：指向模块实例的指针。

返回值

- **TX_SUCCESS** (0x00)：成功停止模块。
- **TX_CALLER_ERROR** (0x13)：调用方无效。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_PTR_ERROR** (0x03)：指针或模块实例无效。
- **TX_START_ERROR** (0x10)：未启动模块。

允许来自

线程

示例

```
/* Start the module. */
txm_module_manager_start(&my_module);

/* Let the module run for a while. */
tx_thread_sleep(20000);

/* Stop the module. */
txm_module_manager_stop(&my_module);

/* Unload the module. */
txm_module_manager_unload(&my_module);
```

另请参阅

- txm_module_manager_start

---

txm_module_manager_unload

卸载模块。

原型

```
UINT txm_module_manager_unload(TXM_MODULE_INSTANCE *module_instance);
```

说明

此服务卸载先前已加载并停止的模块，释放所有关联的模块内存资源。

输入参数

- **module_instance**：指向模块实例的指针。

返回值

- **TX_SUCCESS** (0x00)：成功卸载模块。
- **TX_CALLER_ERROR** (0x13)：调用方无效。
- **TX_NOT_AVAILABLE** (0x1D)：管理器未初始化。
- **TX_NOT_DONE** (0x20)：模块无效或模块未停止。
- **TX_PTR_ERROR** (0x03)：指针或模块实例无效。

允许来自

初始化和线程

示例

```
/* Stop the module. */
txm_module_manager_stop(&my_module);

/* Unload the module. */
status = txm_module_manager_unload(&my_module);
```

另请参阅

- txm_module_manager_file_load
- txm_module_manager_in_place_load
- txm_module_manager_memory_load
- txm_module_manager_stop

附录 - 特定于端口的示例

2021/5/1 •

ARM11 处理器

使用 GCC 的 ARM11

使用 GCC 的 ARM11 的模块报头

```
    .arm
    .section .preamble, "ax"

    /* Define the module preamble.  */

    .global _txm_module_preamble
_txm_module_preamble:
    .word       0x4D4F4455                                      @ Module ID
    .word       0x5                                             @ Module Major Version
    .word       0x6                                             @ Module Minor Version
    .word       32                                              @ Module Preamble Size in 32-bit words
    .word       0x12345678                                      @ Module ID (application defined)
    .word       0x02000000                                      @ Module Properties where:
                                                                @   Bits 31-24: Compiler ID
                                                                @           0 -> IAR
                                                                @           1 -> ARM
                                                                @           2 -> GNU
    .word       _txm_module_thread_shell_entry - . - 0          @ Module Shell Entry Point
    .word       demo_module_start - . - 0                       @ Module Start Thread Entry Point
    .word       0                                               @ Module Stop Thread Entry Point
    .word       1                                               @ Module Start/Stop Thread Priority
    .word       1024                                            @ Module Start/Stop Thread Stack Size
    .word       _txm_module_callback_request_thread_entry - . - 0   @ Module Callback Thread Entry
    .word       1                                               @ Module Callback Thread Priority
    .word       1024                                            @ Module Callback Thread Stack Size
    .word       __code_size__                                   @ Module Code Size
    .word       __data_size__                                   @ Module Data Size
    .word       0                                               @ Reserved 0
    .word       0                                               @ Reserved 1
    .word       0                                               @ Reserved 2
    .word       0                                               @ Reserved 3
    .word       0                                               @ Reserved 4
    .word       0                                               @ Reserved 5
    .word       0                                               @ Reserved 6
    .word       0                                               @ Reserved 7
    .word       0                                               @ Reserved 8
    .word       0                                               @ Reserved 9
    .word       0                                               @ Reserved 10
    .word       0                                               @ Reserved 11
    .word       0                                               @ Reserved 12
    .word       0                                               @ Reserved 13
    .word       0                                               @ Reserved 14
    .word       0                                               @ Reserved 15
```

使用 GCC 的 ARM11 的模块属性

BIT	“[”	[[
[23-0]	0	保留

BIT	“[”	[[
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 GCC 的 ARM11 的模块链接器

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x080F0000, LENGTH = 0x00010000
  RAM   (wx) : ORIGIN = 0x64001800, LENGTH = 0x00100000
}


SECTIONS
{
  __FLASH_segment_start__ = 0x080F0000;
  __FLASH_segment_end__   = 0x080FFFFF;
  __RAM_segment_start__   = 0x64001800;
  __RAM_segment_end__     = 0x64011800;

  __HEAPSIZE__ = 128;

  __preamble_load_start__ = __FLASH_segment_start__;
  .preamble __FLASH_segment_start__ : AT(__FLASH_segment_start__)
  {
    __preamble_start__ = .;
    *(.preamble .preamble.*)
  }
  __preamble_end__ = __preamble_start__ + SIZEOF(.preamble);

  __dynsym_load_start__ =  ALIGN(__preamble_end__ , 4);
  .dynsym ALIGN(__dynsym_load_start__ , 4) : AT(ALIGN(__dynsym_load_start__ , 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynsym))
    KEEP (*(.dynsym*))
    . = ALIGN(4);
  }
  __dynsym_end__ = __dynsym_load_start__ + SIZEOF(.dynsym);

  __dynstr_load_start__ =  ALIGN(__dynsym_end__ , 4);
  .dynstr ALIGN(__dynstr_load_start__ , 4) : AT(ALIGN(__dynstr_load_start__, 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynstr))
    KEEP (*(.dynstr*))
    . = ALIGN(4);
  }
  __dynstr_end__ = __dynstr_load_start__ + SIZEOF(.dynstr);

  __reldyn_load_start__ =  ALIGN(__dynstr_end__ , 4);
  .rel.dyn ALIGN(__reldyn_load_start__ , 4) : AT(ALIGN(__reldyn_load_start__ , 4))
  {
    . = ALIGN(4);
    KEEP (*(.rel.dyn))
    KEEP (*(.rel.dyn*))
    . = ALIGN(4);
  }
  __reldyn_end__ = __reldyn_load_start__ + SIZEOF(.rel.dyn);

  __relplt_load_start__ =  ALIGN(__reldyn_end__ , 4);
  .rel.plt ALIGN(__relplt_load_start__ , 4) : AT(ALIGN(__relplt_load_start__ , 4))
  {
    . = ALIGN(4);
```

```
  . = ALIGN(4);
  KEEP (*(.rel.plt))
  KEEP (*(.rel.plt*))
  . = ALIGN(4);
}
__relplt_end__ = __relplt_load_start__ + SIZEOF(.rel.plt);

__plt_load_start__ =  ALIGN(__relplt_end__ , 4);
.plt ALIGN(__plt_load_start__ , 4) : AT(ALIGN(__plt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.plt))
  KEEP (*(.plt*))
  . = ALIGN(4);
}
__plt_end__ = __plt_load_start__ + SIZEOF(.plt);

__interp_load_start__ =  ALIGN(__plt_end__ , 4);
.interp ALIGN(__interp_load_start__ , 4) : AT(ALIGN(__interp_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.interp))
  KEEP (*(.interp*))
  . = ALIGN(4);
}
__interp_end__ = __interp_load_start__ + SIZEOF(.interp);

__hash_load_start__ =  ALIGN(__interp_end__ , 4);
.hash ALIGN(__hash_load_start__ , 4) : AT(ALIGN(__hash_load_start__, 4))
{
  . = ALIGN(4);
  KEEP (*(.hash))
  KEEP (*(.hash*))
  . = ALIGN(4);
}
__hash_end__ = __hash_load_start__ + SIZEOF(.hash);

__text_load_start__ =  ALIGN(__hash_end__ , 4);
.text ALIGN(__text_load_start__ , 4) : AT(ALIGN(__text_load_start__, 4))
{
  __text_start__ = .;
  *(.text .text.* .glue_7t .glue_7 .gnu.linkonce.t.* .gcc_except_table  )
}
__text_end__ = __text_start__ + SIZEOF(.text);

__dtors_load_start__ = ALIGN(__text_end__ , 4);
.dtors ALIGN(__text_end__ , 4) : AT(ALIGN(__text_end__ , 4))
{
  __dtors_start__ = .;
  KEEP (*(SORT(.dtors.*))) KEEP (*(.dtors))
}
__dtors_end__ = __dtors_start__ + SIZEOF(.dtors);

__ctors_load_start__ = ALIGN(__dtors_end__ , 4);
.ctors ALIGN(__dtors_end__ , 4) : AT(ALIGN(__dtors_end__ , 4))
{
  __ctors_start__ = .;
  KEEP (*(SORT(.ctors.*))) KEEP (*(.ctors))
}
__ctors_end__ = __ctors_start__ + SIZEOF(.ctors);

__got_load_start__ = ALIGN(__ctors_end__ , 4);
.got ALIGN(__ctors_end__ , 4) : AT(ALIGN(__ctors_end__ , 4))
{
  . = ALIGN(4);
  _sgot = .;
  KEEP (*(.got))
  KEEP (*(.got*))
  . = ALIGN(4);
  _egot = .;
```

```
  _egot = .;
}
__got_end__ =  __got_load_start__ + SIZEOF(.got);

__rodata_load_start__ = ALIGN(__got_end__ , 4);
.rodata ALIGN(__got_end__ , 4) : AT(ALIGN(__got_end__ , 4))
{
  __rodata_start__ = .;
  *(.rodata .rodata.* .gnu.linkonce.r.*)
}
__rodata_end__ = __rodata_start__ + SIZEOF(.rodata);

__code_size__ =  __rodata_end__ - __FLASH_segment_start__;

__fast_load_start__ = ALIGN(__rodata_end__ , 4);

__fast_load_end__ = __fast_load_start__ + SIZEOF(.fast);

__new_got_start__ = ALIGN(__RAM_segment_start__ , 4);

__new_got_end__ =  __new_got_start__ + SIZEOF(.got);

.fast ALIGN(__new_got_end__ , 4) : AT(ALIGN(__rodata_end__ , 4))
{
  __fast_start__ = .;
  *(.fast .fast.*)
}
__fast_end__ = __fast_start__ + SIZEOF(.fast);

.fast_run ALIGN(__fast_end__ , 4) (NOLOAD) :
{
  __fast_run_start__ = .;
  . = MAX(__fast_run_start__ + SIZEOF(.fast), .);
}
__fast_run_end__ = __fast_run_start__ + SIZEOF(.fast_run);

__data_load_start__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4);
.data ALIGN(__fast_run_end__ , 4) : AT(ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4))
{
  __data_start__ = .;
  *(.data .data.* .gnu.linkonce.d.*)
}
__data_end__ = __data_start__ + SIZEOF(.data);

__data_load_end__ = __data_load_start__ + SIZEOF(.data);

__FLASH_segment_used_end__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4) + SIZEOF(.data);

.data_run ALIGN(__fast_run_end__ , 4) (NOLOAD) :
{
  __data_run_start__ = .;
  . = MAX(__data_run_start__ + SIZEOF(.data), .);
}
__data_run_end__ = __data_run_start__ + SIZEOF(.data_run);

__bss_load_start__ = ALIGN(__data_run_end__ , 4);
.bss ALIGN(__data_run_end__ , 4) (NOLOAD) : AT(ALIGN(__data_run_end__ , 4))
{
  __bss_start__ = .;
  *(.bss .bss.* .gnu.linkonce.b.*) *(COMMON)
}
__bss_end__ = __bss_start__ + SIZEOF(.bss);

__non_init_load_start__ = ALIGN(__bss_end__ , 4);
.non_init ALIGN(__bss_end__ , 4) (NOLOAD) : AT(ALIGN(__bss_end__ , 4))
{
  __non_init_start__ = .;
  *(.non_init .non_init.*)
}
```

```
  __non_init_end__ = __non_init_start__ + SIZEOF(.non_init);

  __heap_load_start__ = ALIGN(__non_init_end__ , 4);
  .heap ALIGN(__non_init_end__ , 4) (NOLOAD) : AT(ALIGN(__non_init_end__ , 4))
  {
    __heap_start__ = .;
    *(.heap)
    . = ALIGN(MAX(__heap_start__ + __HEAPSIZE__ , .), 4);
  }
  __heap_end__ = __heap_start__ + SIZEOF(.heap);

  __data_size__ =  __heap_end__ - __RAM_segment_start__;

}
```

为使用 GCC 的 ARM11 生成模块

生成使用 GCC 的 ARM11 模块的简单命令行示例：

```
arm-none-eabi-gcc -c -g -mcpu=arm1136j-s -msingle-pic-base -fPIC -mpic-register=r9 txm_module_preamble.S
arm-none-eabi-gcc -c -g -mcpu=arm1136j-s -msingle-pic-base -fPIC -mpic-register=r9 gcc_setup.S
arm-none-eabi-gcc -c -g -mcpu=arm1136j-s -msingle-pic-base -fPIC -mpic-register=r9 demo_threadx_module.c
arm-none-eabi-ld -A arm1136j-s -T demo_threadx_module.ld txm_module_preamble.o gcc_setup.o
demo_threadx_module.o txm.a txm.a -o demo_threadx_module.out -M > demo_threadx_module.map
```

使用 GCC 的 ARM11 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;  \
                                VOID    *tx_thread_module_entry_info_ptr;
```

为使用 GCC 的 ARM11 生成模块管理器

未提供示例。

外部内存属性为使用 GCC 的 ARM11 启用 API

此端口未启用此功能。

使用 AC5 的 ARM11

使用 AC5 的 ARM11 的模块报头

```
    AREA  Init, CODE, READONLY

;    /* Define public symbols.  */

    EXPORT __txm_module_preamble

;    /* Define application-specific start/stop entry points for the module.  */

    IMPORT demo_module_start

;    /* Define common external references.  */

    IMPORT  _txm_module_thread_shell_entry
    IMPORT  _txm_module_callback_request_thread_entry
    IMPORT  |Image$$ER_RO$$Length|

__txm_module_preamble
        DCD       0x4D4F4455                                    ; Module ID
        DCD       0x5                                           ; Module Major Version
        DCD       0x3                                           ; Module Minor Version
        DCD       32                                            ; Module Preamble Size in 32-bit words
        DCD       0x12345678                                    ; Module ID (application defined)
        DCD       0x01000000                                    ; Module Properties where:
                                                                ;   Bits 31-24: Compiler ID
                                                                ;           0 -> IAR
                                                                ;           1 -> ARM
                                                                ;           2 -> GNU
                                                                ;   Bits 23-0:  Reserved
        DCD       _txm_module_thread_shell_entry - . + .        ; Module Shell Entry Point
        DCD       demo_module_start - . + .                     ; Module Start Thread Entry Point
        DCD       0                                             ; Module Stop Thread Entry Point
        DCD       1                                             ; Module Start/Stop Thread Priority
        DCD       1024                                          ; Module Start/Stop Thread Stack Size
        DCD       _txm_module_callback_request_thread_entry - . + . ; Module Callback Thread Entry
        DCD       1                                             ; Module Callback Thread Priority
        DCD       1024                                          ; Module Callback Thread Stack Size
        DCD       |Image$$ER_RO$$Length|                        ; Module Code Size
        DCD       0x4000                                        ; Module Data Size - default to 16K
(need to make sure this is large enough for module's data needs!)
        DCD       0                                             ; Reserved 0
        DCD       0                                             ; Reserved 1
        DCD       0                                             ; Reserved 2
        DCD       0                                             ; Reserved 3
        DCD       0                                             ; Reserved 4
        DCD       0                                             ; Reserved 5
        DCD       0                                             ; Reserved 6
        DCD       0                                             ; Reserved 7
        DCD       0                                             ; Reserved 8
        DCD       0                                             ; Reserved 9
        DCD       0                                             ; Reserved 10
        DCD       0                                             ; Reserved 11
        DCD       0                                             ; Reserved 12
        DCD       0                                             ; Reserved 13
        DCD       0                                             ; Reserved 14
        DCD       0                                             ; Reserved 15

        END
```

使用 AC5 的 ARM11 的模块属性

BIT	“[”	[[
[23-0]	0	保留

BIT	“[”	[[
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC5 的 ARM11 的模块链接器

基于命令行生成，没有链接器文件示例。

为使用 AC5 的 ARM11 生成模块

生成使用 AC5 的 ARM11 模块的简单命令行示例：

```
armasm -g --cpu ARM1136J-S --apcs /interwork --apcs /ropi --apcs /rwpi txm_module_preamble.s
armcc -g -c -O0 --cpu ARM1136J-S --apcs /interwork --apcs /ropi --apcs /rwpi demo_threadx_module.c
armlink -d -o demo_threadx_module.axf --elf --ro 0 --first txm_module_preamble.o(Init) --
entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --list demo_threadx_module.map
txm_module_preamble.o demo_threadx_module.o txm.a
```

使用 AC5 的 ARM11 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;  \
                                VOID    *tx_thread_module_entry_info_ptr;
```

为使用 AC5 的 ARM11 生成模块管理器

生成使用 AC5 的 ARM11 模块管理器的简单命令行示例：

```
armasm -g --cpu ARM1136J-S --apcs /interwork tx_initialize_low_level.s
armcc -g -c -O2 --cpu ARM1136J-S --apcs /interwork demo_threadx_module_manager.c
armcc -g -c -O2 --cpu ARM1136J-S --apcs /interwork module_code.c
armlink -d -o demo_threadx_module_manager.axf --elf --ro 0 --first tx_initialize_low_level.o(Init) --remove
--map --symbols --list demo_threadx_module_manager.map tx_initialize_low_level.o
demo_threadx_module_manager.o module_code.o tx.a
```

外部内存属性为使用 AC5 的 ARM11 启用 API

此端口未启用此功能。

Cortex-A7 处理器

使用 AC5 的 Cortex-A7

使用 AC5 的 Cortex-A7 的模块报头

```
    AREA  Init, CODE, READONLY

;    /* Define public symbols.  */

    EXPORT __txm_module_preamble

;    /* Define application-specific start/stop entry points for the module.  */

    IMPORT demo_module_start

;    /* Define common external references.  */

    IMPORT  _txm_module_thread_shell_entry
    IMPORT  _txm_module_callback_request_thread_entry
    IMPORT  |Image$$ER_RO$$Length|
    IMPORT  |Image$$ER_RW$$Length|

__txm_module_preamble
    DCD       0x4D4F4455                                    ; Module ID
    DCD       0x5                                           ; Module Major Version
    DCD       0x3                                           ; Module Minor Version
    DCD       32                                            ; Module Preamble Size in 32-bit words
    DCD       0x12345678                                    ; Module ID (application defined)
    DCD       0x01000001                                    ; Module Properties where:
                                                            ;   Bits 31-24: Compiler ID
                                                            ;           0 -> IAR
                                                            ;           1 -> ARM
                                                            ;           2 -> GNU
                                                            ;   Bits 23-1:  Reserved
                                                            ;   Bit 0:  0 -> Privileged mode execution
(no MMU protection)
                                                            ;           1 -> User mode execution (MMU
protection)
    DCD       _txm_module_thread_shell_entry - . + .        ; Module Shell Entry Point
    DCD       demo_module_start - . + .                     ; Module Start Thread Entry Point
    DCD       0                                             ; Module Stop Thread Entry Point
    DCD       1                                             ; Module Start/Stop Thread Priority
    DCD       1024                                          ; Module Start/Stop Thread Stack Size
    DCD       _txm_module_callback_request_thread_entry - . + . ; Module Callback Thread Entry
    DCD       1                                             ; Module Callback Thread Priority
    DCD       1024                                          ; Module Callback Thread Stack Size
    DCD       |Image$$ER_RO$$Length| + |Image$$ER_RW$$Length|   ; Module Code Size
    DCD       0x4000                                        ; Module Data Size - default to 16K (need to
make sure this is large enough for module's data needs!)
    DCD       0                                             ; Reserved 0
    DCD       0                                             ; Reserved 1
    DCD       0                                             ; Reserved 2
    DCD       0                                             ; Reserved 3
    DCD       0                                             ; Reserved 4
    DCD       0                                             ; Reserved 5
    DCD       0                                             ; Reserved 6
    DCD       0                                             ; Reserved 7
    DCD       0                                             ; Reserved 8
    DCD       0                                             ; Reserved 9
    DCD       0                                             ; Reserved 10
    DCD       0                                             ; Reserved 11
    DCD       0                                             ; Reserved 12
    DCD       0                                             ; Reserved 13
    DCD       0                                             ; Reserved 14
    DCD       0                                             ; Reserved 15

    END
```

使用 AC5 的 Cortex-A7 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
[23-1]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC5 的 Cortex-A7 的模块链接器

基于命令行生成，没有链接器文件示例。

为使用 AC5 的 Cortex-A7 生成模块

生成使用 AC5 的 Cortex-A7 模块的简单命令行示例：

```
armasm -g --cpu=cortex-a7.no_neon --fpu=softvfp --apcs=/interwork/ropi/rwpi txm_module_preamble.s
armcc  -g --cpu=cortex-a7.no_neon --fpu=softvfp -c --apcs=/interwork/ropi/rwpi --lower_ropi
demo_threadx_module.c
armlink -d -o demo_threadx_module.axf --elf --ro 0 --first txm_module_preamble.o(Init) --
entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --list demo_threadx_module.map
txm_module_preamble.o demo_threadx_module.o txm.a
```

使用 AC5 的 Cortex-A7 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   ULONG   tx_thread_vfp_enable;                   \
                                VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;
```

为使用 AC5 的 Cortex-A7 生成模块管理器

生成使用 AC5 的 Cortex-A7 模块管理器的简单命令行示例：

```
armasm -g --cpu=cortex-a7.no_neon --fpu=softvfp --apcs=interwork tx_initialize_low_level.s
armcc -g --cpu=cortex-a7.no_neon --fpu=softvfp -c demo_threadx_module_manager.c
armcc -g --cpu=cortex-a7.no_neon --fpu=softvfp -c module_code.c
armlink -d -o demo_threadx_module_manager.axf --elf --ro 0x80000000 --first
tx_initialize_low_level.o(VECTORS) --remove --map --symbols --list demo_threadx_module_manager.map
tx_initialize_low_level.o demo_threadx_module_manager.o module_code.o tx.a
```

外部内存属性为使用 AC5 的 Cortex-A7 启用 API

以下属性可用于设置共享内存设置：| 属性参数 | 含义 | |---|---| | TXM_MMU_ATTRIBUTE_XN | 从不执行 | | TXM_MMU_ATTRIBUTE_B | B 设置 | | TXM_MMU_ATTRIBUTE_C | C 设置 | | TXM_MMU_ATTRIBUTE_AP | AP 设置 | | TXM_MMU_ATTRIBUTE_TEX | TEX 设置 |

请参阅 ARM 文档了解如何配置这些设置。

Cortex-M3 处理器

使用 AC5 的 Cortex-M3

使用 AC5 的 Cortex-M3 的模块报头

```
    AREA Init, CODE, READONLY

    PRESERVE8

    ; Define public symbols

    EXPORT __txm_module_preamble

    ; Define application-specific start/stop entry points for the module

    EXTERN demo_module_start

    ; Define common external references

    IMPORT  _txm_module_thread_shell_entry
    IMPORT  _txm_module_callback_request_thread_entry
    IMPORT  |Image$$ER_RO$$Length|
    IMPORT  |Image$$ER_RW$$Length|
    IMPORT  |Image$$ER_RW$$RW$$Length|
    IMPORT  |Image$$ER_RW$$ZI$$Length|
    IMPORT  |Image$$ER_ZI$$ZI$$Length|

__txm_module_preamble
    DCD     0x4D4F4455                                          ; Module ID
    DCD     0x6                                                 ; Module Major Version
    DCD     0x1                                                 ; Module Minor Version
    DCD     32                                                  ; Module Preamble Size in 32-bit words
    DCD     0x12345678                                          ; Module ID (application defined)
    DCD     0x01000007                                          ; Module Properties where:
                                                                ;   Bits 31-24: Compiler ID
                                                                ;           0 -> IAR
                                                                ;           1 -> ARM
                                                                ;           2 -> GNU
                                                                ;   Bit 0:  0 -> Privileged mode execution
                                                                ;           1 -> User mode execution
                                                                ;   Bit 1:  0 -> No MPU protection
                                                                ;           1 -> MPU protection (must have
user mode selected)
                                                                ;   Bit 2:  0 -> Disable shared/external
memory access
                                                                ;           1 -> Enable shared/external
memory access
    DCD     _txm_module_thread_shell_entry - __txm_module_preamble              ; Module Shell Entry Point
    DCD     demo_module_start - __txm_module_preamble           ; Module Start Thread Entry Point
    DCD     0                                                   ; Module Stop Thread Entry Point
    DCD     1                                                   ; Module Start/Stop Thread Priority
    DCD     1024                                                ; Module Start/Stop Thread Stack Size
    DCD     _txm_module_callback_request_thread_entry - __txm_module_preamble   ; Module Callback Thread
Entry
    DCD     1                                                   ; Module Callback Thread Priority
    DCD     1024                                                ; Module Callback Thread Stack Size
    DCD     |Image$$ER_RO$$Length| + |Image$$ER_RW$$Length|         ; Module Code Size
    DCD     |Image$$ER_RW$$Length| + |Image$$ER_ZI$$ZI$$Length| ; Module Data Size
    DCD     0                                                   ; Reserved 0
    DCD     0                                                   ; Reserved 1
    DCD     0                                                   ; Reserved 2
    DCD     0                                                   ; Reserved 3
    DCD     0                                                   ; Reserved 4
    DCD     0                                                   ; Reserved 5
    DCD     0                                                   ; Reserved 6
    DCD     0                                                   ; Reserved 7
    DCD     0                                                   ; Reserved 8
```

```
    DCD     0                                       ; Reserved 9
    DCD     0                                       ; Reserved 10
    DCD     0                                       ; Reserved 11
    DCD     0                                       ; Reserved 12
    DCD     0                                       ; Reserved 13
    DCD     0                                       ; Reserved 14
    DCD     0                                       ; Reserved 15

    END
```

使用 AC5 的 Cortex-M3 的模块属性

BIT	"["	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC5 的 Cortex-M3 的模块链接器

未提供示例链接器文件;链接在命令行上完成。请参阅下一节。

为使用 AC5 的 Cortex-M3 生成模块

下面提供了一个生成脚本示例:

```
armasm -g --cpu=cortex-m3 --apcs=/interwork/ropi/rwpi txm_module_preamble.S
armcc  -g --cpu=cortex-m3 -c --apcs=/interwork/ropi/rwpi --lower_ropi -I../inc -
I../../../../common_modules/inc -I../../../../common_modules/module_manager/inc -I../../../../common/inc
sample_threadx_module.c
armlink -d -o sample_threadx_module.axf --elf --ro=0x30000 --rw=0x40000 --first txm_module_preamble.o(Init)
--entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --list
sample_threadx_module.map txm_module_preamble.o sample_threadx_module.o txm.a
```

使用 AC5 的 Cortex-M3 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 AC5 的 Cortex-M3 生成模块管理器

请参阅示例 build_threadx_module_manager_demo.bat:

```
armasm -g --cpu=cortex-m3 --apcs=interwork tx_initialize_low_level.S
armcc -g --cpu=cortex-m3 -c -I../inc -I../../../../common_modules/inc -
I../../../../common_modules/module_manager/inc -I../../../../common/inc sample_threadx_module_manager.c
armlink -d -o sample_threadx_module_manager.axf --elf --ro 0x00000000 --first
tx_initialize_low_level.o(RESET) --remove --map --symbols --list sample_threadx_module_manager.map
tx_initialize_low_level.o sample_threadx_module_manager.o tx.a
```

外部内存属性为使用 AC5 的 Cortex-M3 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 AC6 的 Cortex-M3

使用 AC6 的 Cortex-M3 的模块报头

```
    .text
    .align 4
    .syntax unified
    .section Init

    // Define public symbols
    .global __txm_module_preamble

    // Define application-specific start/stop entry points for the module
    .global demo_module_start

    // Define common external references
    .global  _txm_module_thread_shell_entry
    .global  _txm_module_callback_request_thread_entry

    .eabi_attribute Tag_ABI_PCS_RO_data, 1
    .eabi_attribute Tag_ABI_PCS_R9_use,  1
    .eabi_attribute Tag_ABI_PCS_RW_data, 2

__txm_module_preamble:
    .dc.l   0x4D4F4455                                          // Module ID
    .dc.l   0x6                                                 // Module Major Version
    .dc.l   0x1                                                 // Module Minor Version
    .dc.l   32                                                  // Module Preamble Size in 32-bit words
    .dc.l   0x12345678                                          // Module ID (application defined)
    .dc.l   0x01000007                                          // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected)
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
    .dc.l   _txm_module_thread_shell_entry - __txm_module_preamble // Module Shell Entry Point
    .dc.l   demo_module_start - __txm_module_preamble           // Module Start Thread Entry Point
    .dc.l   0                                                   // Module Stop Thread Entry Point
    .dc.l   1                                                   // Module Start/Stop Thread Priority
    .dc.l   1024                                                // Module Start/Stop Thread Stack Size
    .dc.l   _txm_module_callback_request_thread_entry - __txm_module_preamble // Module Callback Thread
Entry
    .dc.l   1                                                   // Module Callback Thread Priority
    .dc.l   1024                                                // Module Callback Thread Stack Size
    .dc.l   0x10000                                             // Module Code Size
    .dc.l   0x10000                                             // Module Data Size
    .dc.l   0                                                   // Reserved 0
    .dc.l   0                                                   // Reserved 1
    .dc.l   0                                                   // Reserved 2
    .dc.l   0                                                   // Reserved 3
    .dc.l   0                                                   // Reserved 4
    .dc.l   0                                                   // Reserved 5
    .dc.l   0                                                   // Reserved 6
    .dc.l   0                                                   // Reserved 7
    .dc.l   0                                                   // Reserved 8
    .dc.l   0                                                   // Reserved 9
    .dc.l   0                                                   // Reserved 10
    .dc.l   0                                                   // Reserved 11
    .dc.l   0                                                   // Reserved 12
    .dc.l   0                                                   // Reserved 13
    .dc.l   0                                                   // Reserved 14
    .dc.l   0                                                   // Reserved 15
```

使用 AC6 的 Cortex-M3 的模块属性

BIT	"["	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC6 的 Cortex-M3 的模块链接器

未使用链接器文件。请参阅项目设置。

为使用 AC6 的 Cortex-M3 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 AC6 的 Cortex-M3 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 AC6 的 Cortex-M3 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 AC6 的 Cortex-M3 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| |
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 GNU 的 Cortex-M3

使用 GNU 的 Cortex-M3 的模块报头

```
    .text
    .align 4
    .syntax unified

    /* Define public symbols.  */
    .global __txm_module_preamble

    /* Define application-specific start/stop entry points for the module.  */
    .global demo_module_start

    /* Define common external refrences.  */
    .global _txm_module_thread_shell_entry
    .global _txm_module_callback_request_thread_entry

__txm_module_preamble:
    .dc.l      0x4D4F4455                                       // Module ID
    .dc.l      0x6                                              // Module Major Version
    .dc.l      0x1                                              // Module Minor Version
    .dc.l      32                                               // Module Preamble Size in 32-bit words
    .dc.l      0x12345678                                       // Module ID (application defined)
    .dc.l      0x02000007                                       // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bits 23-3: Reserved
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
    .dc.l      _txm_module_thread_shell_entry - . - 0           // Module Shell Entry Point
    .dc.l      demo_module_start - . - 0                        // Module Start Thread Entry Point
    .dc.l      0                                                // Module Stop Thread Entry Point
    .dc.l      1                                                // Module Start/Stop Thread Priority
    .dc.l      1024                                             // Module Start/Stop Thread Stack Size
    .dc.l      _txm_module_callback_request_thread_entry - . - 0 // Module Callback Thread Entry
    .dc.l      1                                                // Module Callback Thread Priority
    .dc.l      1024                                             // Module Callback Thread Stack Size
    .dc.l      __code_size__                                    // Module Code Size
    .dc.l      __data_size__                                    // Module Data Size
    .dc.l      0                                                // Reserved 0
    .dc.l      0                                                // Reserved 1
    .dc.l      0                                                // Reserved 2
    .dc.l      0                                                // Reserved 3
    .dc.l      0                                                // Reserved 4
    .dc.l      0                                                // Reserved 5
    .dc.l      0                                                // Reserved 6
    .dc.l      0                                                // Reserved 7
    .dc.l      0                                                // Reserved 8
    .dc.l      0                                                // Reserved 9
    .dc.l      0                                                // Reserved 10
    .dc.l      0                                                // Reserved 11
    .dc.l      0                                                // Reserved 12
    .dc.l      0                                                // Reserved 13
    .dc.l      0                                                // Reserved 14
    .dc.l      0                                                // Reserved 15
```

使用 GNU 的 Cortex-M3 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 GNU 的 Cortex-M3 的模块链接器

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x00030000, LENGTH = 0x00010000
  RAM   (wx) : ORIGIN = 0, LENGTH = 0x00100000
}


SECTIONS
{
  __FLASH_segment_start__ = 0x00030000;
  __FLASH_segment_end__   = 0x00040000;
  __RAM_segment_start__   = 0;
  __RAM_segment_end__     = 0x8000;

  __HEAPSIZE__ = 128;

  __preamble_load_start__ = __FLASH_segment_start__;
  .preamble __FLASH_segment_start__ : AT(__FLASH_segment_start__)
  {
    __preamble_start__ = .;
    *(.preamble .preamble.*)
  }
  __preamble_end__ = __preamble_start__ + SIZEOF(.preamble);

  __dynsym_load_start__ =  ALIGN(__preamble_end__ , 4);
  .dynsym ALIGN(__dynsym_load_start__ , 4) : AT(ALIGN(__dynsym_load_start__ , 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynsym))
    KEEP (*(.dynsym*))
    . = ALIGN(4);
  }
  __dynsym_end__ = __dynsym_load_start__ + SIZEOF(.dynsym);

  __dynstr_load_start__ =  ALIGN(__dynsym_end__ , 4);
  .dynstr ALIGN(__dynstr_load_start__ , 4) : AT(ALIGN(__dynstr_load_start__, 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynstr))
    KEEP (*(.dynstr*))
    . = ALIGN(4);
  }
  __dynstr_end__ = __dynstr_load_start__ + SIZEOF(.dynstr);
```

```
__reldyn_load_start__ =  ALIGN(__dynstr_end__ , 4);
.rel.dyn ALIGN(__reldyn_load_start__ , 4) : AT(ALIGN(__reldyn_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.dyn))
  KEEP (*(.rel.dyn*))
  . = ALIGN(4);
}
__reldyn_end__ = __reldyn_load_start__ + SIZEOF(.rel.dyn);

__relplt_load_start__ =  ALIGN(__reldyn_end__ , 4);
.rel.plt ALIGN(__relplt_load_start__ , 4) : AT(ALIGN(__relplt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.plt))
  KEEP (*(.rel.plt*))
  . = ALIGN(4);
}
__relplt_end__ = __relplt_load_start__ + SIZEOF(.rel.plt);

__plt_load_start__ =  ALIGN(__relplt_end__ , 4);
.plt ALIGN(__plt_load_start__ , 4) : AT(ALIGN(__plt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.plt))
  KEEP (*(.plt*))
  . = ALIGN(4);
}
__plt_end__ = __plt_load_start__ + SIZEOF(.plt);

__interp_load_start__ =  ALIGN(__plt_end__ , 4);
.interp ALIGN(__interp_load_start__ , 4) : AT(ALIGN(__interp_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.interp))
  KEEP (*(.interp*))
  . = ALIGN(4);
}
__interp_end__ = __interp_load_start__ + SIZEOF(.interp);

__hash_load_start__ =  ALIGN(__interp_end__ , 4);
.hash ALIGN(__hash_load_start__ , 4) : AT(ALIGN(__hash_load_start__, 4))
{
  . = ALIGN(4);
  KEEP (*(.hash))
  KEEP (*(.hash*))
  . = ALIGN(4);
}
__hash_end__ = __hash_load_start__ + SIZEOF(.hash);

__text_load_start__ =  ALIGN(__hash_end__ , 4);
.text ALIGN(__text_load_start__ , 4) : AT(ALIGN(__text_load_start__, 4))
{
  __text_start__ = .;
  *(.text .text.* .glue_7t .glue_7 .gnu.linkonce.t.* .gcc_except_table  )
}
__text_end__ = __text_start__ + SIZEOF(.text);

__dtors_load_start__ = ALIGN(__text_end__ , 4);
.dtors ALIGN(__text_end__ , 4) : AT(ALIGN(__text_end__ , 4))
{
  __dtors_start__ = .;
  KEEP (*(SORT(.dtors.*))) KEEP (*(.dtors))
}
__dtors_end__ = __dtors_start__ + SIZEOF(.dtors);

__ctors_load_start__ = ALIGN(__dtors_end__ , 4);
.ctors ALIGN(__dtors_end__ , 4) : AT(ALIGN(__dtors_end__ , 4))
{
```

```
  __ctors_start__ = .;
  KEEP (*(SORT(.ctors.*))) KEEP (*(.ctors))
}
__ctors_end__ = __ctors_start__ + SIZEOF(.ctors);

__got_load_start__ = ALIGN(__ctors_end__ , 4);
.got ALIGN(__ctors_end__ , 4) : AT(ALIGN(__ctors_end__ , 4))
{
  . = ALIGN(4);
  _sgot = .;
  KEEP (*(.got))
  KEEP (*(.got*))
  . = ALIGN(4);
  _egot = .;
}
__got_end__ =  __got_load_start__ + SIZEOF(.got);

__rodata_load_start__ = ALIGN(__got_end__ , 4);
.rodata ALIGN(__got_end__ , 4) : AT(ALIGN(__got_end__ , 4))
{
  __rodata_start__ = .;
  *(.rodata .rodata.* .gnu.linkonce.r.*)
}
__rodata_end__ = __rodata_start__ + SIZEOF(.rodata);

__code_size__ =  __rodata_end__ - __FLASH_segment_start__;

__fast_load_start__ = ALIGN(__rodata_end__ , 4);

__fast_load_end__ = __fast_load_start__ + SIZEOF(.fast);

__new_got_start__ = ALIGN(__RAM_segment_start__ , 4);

__new_got_end__ =  __new_got_start__ + SIZEOF(.got);

.fast ALIGN(__new_got_end__ , 4) : AT(ALIGN(__rodata_end__ , 4))
{
  __fast_start__ = .;
  *(.fast .fast.*)
}
__fast_end__ = __fast_start__ + SIZEOF(.fast);

.fast_run ALIGN(__fast_end__ , 4) (NOLOAD) :
{
  __fast_run_start__ = .;
  . = MAX(__fast_run_start__ + SIZEOF(.fast), .);
}
__fast_run_end__ = __fast_run_start__ + SIZEOF(.fast_run);

__data_load_start__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4);
.data ALIGN(__fast_run_end__ , 4) : AT(ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4))
{
  __data_start__ = .;
  *(.data .data.* .gnu.linkonce.d.*)
}
__data_end__ = __data_start__ + SIZEOF(.data);

__data_load_end__ = __data_load_start__ + SIZEOF(.data);

__FLASH_segment_used_end__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4) + SIZEOF(.data);

.data_run ALIGN(__fast_run_end__ , 4) (NOLOAD) :
{
  __data_run_start__ = .;
  . = MAX(__data_run_start__ + SIZEOF(.data), .);
}
__data_run_end__ = __data_run_start__ + SIZEOF(.data_run);

__bss_load_start__ = ALIGN(__data_run_end__ , 4);
```

```
  .bss ALIGN(__data_run_end__ , 4) (NOLOAD) : AT(ALIGN(__data_run_end__ , 4))
  {
    __bss_start__ = .;
    *(.bss .bss.* .gnu.linkonce.b.*) *(COMMON)
  }
  __bss_end__ = __bss_start__ + SIZEOF(.bss);

  __non_init_load_start__ = ALIGN(__bss_end__ , 4);
  .non_init ALIGN(__bss_end__ , 4) (NOLOAD) : AT(ALIGN(__bss_end__ , 4))
  {
    __non_init_start__ = .;
    *(.non_init .non_init.*)
  }
  __non_init_end__ = __non_init_start__ + SIZEOF(.non_init);

  __heap_load_start__ = ALIGN(__non_init_end__ , 4);
  .heap ALIGN(__non_init_end__ , 4) (NOLOAD) : AT(ALIGN(__non_init_end__ , 4))
  {
    __heap_start__ = .;
    *(.heap)
    . = ALIGN(MAX(__heap_start__ + __HEAPSIZE__ , .), 4);
  }
  __heap_end__ = __heap_start__ + SIZEOF(.heap);

  __data_size__ =  __heap_end__ - __RAM_segment_start__;

}
```

为使用 GNU 的 Cortex-M3 生成模块

请参阅 build_threadx_module_sample.bat:

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -fpie -fno-plt -mpic-data-is-text-relative -msingle-pic-base
txm_module_preamble.s
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -fpie -fno-plt -mpic-data-is-text-relative -msingle-pic-base
gcc_setup.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -fpie -fno-plt -mpic-data-is-text-relative -msingle-pic-base -
I..\inc -I..\..\..\..\common\inc -I..\..\..\..\common_modules\inc sample_threadx_module.c
arm-none-eabi-ld -A cortex-m3 -T sample_threadx_module.ld txm_module_preamble.o gcc_setup.o
sample_threadx_module.o -e _txm_module_thread_shell_entry txm.a -o sample_threadx_module.axf -M >
sample_threadx_module.map
```

使用 GNU 的 Cortex-M3 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 GNU 的 Cortex-M3 生成模块管理器

请参阅 build_threadx_module_manager_sample.bat:

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -mthumb -I..\inc -I..\..\..\..\common\inc -
I..\..\..\..\common_modules\inc sample_threadx_module_manager.c
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -mthumb tx_simulator_startup.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m3 -mthumb cortexm_crt0.S
arm-none-eabi-ld -A cortex-m3 -ereset_handler -T sample_threadx.ld tx_simulator_startup.o cortexm_crt0.o
sample_threadx_module_manager.o tx.a  libc.a -o sample_threadx_module_manager.axf -M >
sample_threadx_module_manager.map
```

外部内存属性为使用 GNU 的 Cortex-M3 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 IAR 的 Cortex-M3

使用 IAR 的 Cortex-M3 的模块报头

```
    SECTION .text:CODE

    AAPCS INTERWORK, ROPI, RWPI_COMPATIBLE,  VFP_COMPATIBLE
    PRESERVE8

    /* Define public symbols.  */

    PUBLIC __txm_module_preamble


    /* Define application-specific start/stop entry points for the module.  */

    EXTERN demo_module_start


    /* Define common external references.  */

    EXTERN  _txm_module_thread_shell_entry
    EXTERN  _txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                    ; Module ID
    DC32      0x5                                           ; Module Major Version
    DC32      0x6                                           ; Module Minor Version
    DC32      32                                            ; Module Preamble Size in 32-bit words
    DC32      0x12345678                                    ; Module ID (application defined)
    DC32      0x00000007                                    ; Module Properties where:
                                                            ;   Bits 31-24: Compiler ID
                                                            ;           0 -> IAR
                                                            ;           1 -> ARM
                                                            ;           2 -> GNU
                                                            ;   Bits 23-3: Reserved
                                                            ;   Bit 2:  0 -> Disable shared/external
memory access
                                                            ;           1 -> Enable shared/external
memory access
                                                            ;   Bit 1:  0 -> No MPU protection
                                                            ;           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                            ;   Bit 0:  0 -> Privileged mode execution
                                                            ;           1 -> User mode execution


    DC32      _txm_module_thread_shell_entry - . - 0        ; Module Shell Entry Point
    DC32      demo_module_start - . - 0                     ; Module Start Thread Entry Point
    DC32      0                                             ; Module Stop Thread Entry Point
    DC32      1                                             ; Module Start/Stop Thread Priority
    DC32      1024                                          ; Module Start/Stop Thread Stack Size
```

```
    DC32    1024                                                ; Module Start/Stop Thread Stack Size
    DC32    _txm_module_callback_request_thread_entry - . - 0 ; Module Callback Thread Entry
    DC32    1                                                   ; Module Callback Thread Priority
    DC32    1024                                                ; Module Callback Thread Stack Size
    DC32    ROPI$$Length                                        ; Module Code Size
    DC32    RWPI$$Length                                        ; Module Data Size
    DC32    0                                                   ; Reserved 0
    DC32    0                                                   ; Reserved 1
    DC32    0                                                   ; Reserved 2
    DC32    0                                                   ; Reserved 3
    DC32    0                                                   ; Reserved 4
    DC32    0                                                   ; Reserved 5
    DC32    0                                                   ; Reserved 6
    DC32    0                                                   ; Reserved 7
    DC32    0                                                   ; Reserved 8
    DC32    0                                                   ; Reserved 9
    DC32    0                                                   ; Reserved 10
    DC32    0                                                   ; Reserved 11
    DC32    0                                                   ; Reserved 12
    DC32    0                                                   ; Reserved 13
    DC32    0                                                   ; Reserved 14
    DC32    0                                                   ; Reserved 15

    END
```

使用 IAR 的 Cortex-M3 的模块属性

BIT	“ι”	ιι
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 IAR 的 Cortex-M3 的模块链接器

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\a_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x080f0000;
define symbol __ICFEDIT_region_ROM_end__   = 0x080fffff;
define symbol __ICFEDIT_region_RAM_start__ = 0x64002800;
define symbol __ICFEDIT_region_RAM_end__   = 0x64100000;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0;
define symbol __ICFEDIT_size_svcstack__ = 0;
define symbol __ICFEDIT_size_irqstack__ = 0;
define symbol __ICFEDIT_size_fiqstack__ = 0;
define symbol __ICFEDIT_size_undstack__ = 0;
define symbol __ICFEDIT_size_abtstack__ = 0;
define symbol __ICFEDIT_size_heap__     = 0x1000;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

//define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
//define block SVC_STACK with alignment = 8, size = __ICFEDIT_size_svcstack__ { };
//define block IRQ_STACK with alignment = 8, size = __ICFEDIT_size_irqstack__ { };
//define block FIQ_STACK with alignment = 8, size = __ICFEDIT_size_fiqstack__ { };
//define block UND_STACK with alignment = 8, size = __ICFEDIT_size_undstack__ { };
//define block ABT_STACK with alignment = 8, size = __ICFEDIT_size_abtstack__ { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

//place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

define movable block ROPI with alignment = 4, fixed order
{
  ro object txm_module_preamble_stm32f2xx.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base
{
  rw,
  block HEAP
};

place in ROM_region   { block ROPI };
place in RAM_region   { block RWPI };
```

为使用 IAR 的 Cortex-M3 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 Cortex-M3 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                ULONG   tx_thread_module_saved_lr;              \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;             \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 Cortex-M3 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 Cortex-M3 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

Cortex-M33 处理器

使用 AC6 的 Cortex-M33

使用 AC6 的 Cortex-M33 的模块报头

```
    .text
    .align 4
    .syntax unified
    .section RESET

    // Define public symbols
    .global __txm_module_preamble

    // Define application-specific start/stop entry points for the module
    .global demo_module_start

    // Define common external references
    .global  _txm_module_thread_shell_entry
    .global  _txm_module_callback_request_thread_entry

    .eabi_attribute Tag_ABI_PCS_RO_data, 1
    .eabi_attribute Tag_ABI_PCS_R9_use,  1
    .eabi_attribute Tag_ABI_PCS_RW_data, 2

__txm_module_preamble:
    .dc.l   0x4D4F4455                                      // Module ID
    .dc.l   0x6                                             // Module Major Version
    .dc.l   0x1                                             // Module Minor Version
    .dc.l   32                                              // Module Preamble Size in 32-bit words
    .dc.l   0x12345678                                      // Module ID (application defined)
    .dc.l   0x01000007                                      // Module Properties where:
                                                            //   Bits 31-24: Compiler ID
                                                            //           0 -> IAR
                                                            //           1 -> ARM
                                                            //           2 -> GNU
                                                            //   Bit 0:  0 -> Privileged mode execution
                                                            //           1 -> User mode execution
                                                            //   Bit 1:  0 -> No MPU protection
                                                            //           1 -> MPU protection (must have
user mode selected)
                                                            //   Bit 2:  0 -> Disable shared/external
memory access
```

```
                                                        //          1 -> Enable shared/external
memory access
    .dc.l   _txm_module_thread_shell_entry - __txm_module_preamble // Module Shell Entry Point
    .dc.l   demo_module_start - __txm_module_preamble           // Module Start Thread Entry Point
    .dc.l   0                                                   // Module Stop Thread Entry Point
    .dc.l   1                                                   // Module Start/Stop Thread Priority
    .dc.l   1024                                                // Module Start/Stop Thread Stack Size
    .dc.l   _txm_module_callback_request_thread_entry - __txm_module_preamble // Module Callback Thread
Entry
    .dc.l   1                                                   // Module Callback Thread Priority
    .dc.l   1024                                                // Module Callback Thread Stack Size
    //the tools can't add two symbols together, but it should look like this:
    //.dc.l   Image$$ER_RO$$Length + Image$$ER_RW$$Length         // Module Code Size
    //.dc.l   Image$$ER_RW$$Length + Image$$ER_ZI$$ZI$$Length     // Module Data Size
    //so instead we'll define hard values:
    .dc.l   0x4000                                              // Module Code Size
    .dc.l   0x4000                                              // Module Data Size
    .dc.l   0                                                   // Reserved 0
    .dc.l   0                                                   // Reserved 1
    .dc.l   0                                                   // Reserved 2
    .dc.l   0                                                   // Reserved 3
    .dc.l   0                                                   // Reserved 4
    .dc.l   0                                                   // Reserved 5
    .dc.l   0                                                   // Reserved 6
    .dc.l   0                                                   // Reserved 7
    .dc.l   0                                                   // Reserved 8
    .dc.l   0                                                   // Reserved 9
    .dc.l   0                                                   // Reserved 10
    .dc.l   0                                                   // Reserved 11
    .dc.l   0                                                   // Reserved 12
    .dc.l   0                                                   // Reserved 13
    .dc.l   0                                                   // Reserved 14
    .dc.l   0                                                   // Reserved 15
```

使用 AC6 的 Cortex-M33 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC6 的 Cortex-M33 的模块链接器

Keil 工具链不需要链接器文件。请参阅示例项目中的生成设置。下面列出了重要的链接器选项:

```
--entry demo_module_start --first __txm_module_preamble
```

为使用 AC6 的 Cortex-M33 生成模块

编译器设置:

```
-xc -std=c99 --target=arm-arm-none-eabi -mcpu=cortex-m33 -mfpu=fpv5-sp-d16 -mfloat-abi=hard -c
-fno-rtti -funsigned-char -fshort-enums -fshort-wchar
-mlittle-endian -gdwarf-3 -fropi -frwpi -O1 -ffunction-sections -Wno-packed -Wno-missing-variable-
declarations -Wno-missing-prototypes -Wno-missing-noreturn -Wno-sign-conversion -Wno-nonportable-include-
path -Wno-reserved-id-macro -Wno-unused-macros -Wno-documentation-unknown-command -Wno-documentation -Wno-
license-management -Wno-parentheses-equality -I ../../../../../common_modules/inc -I
../../../../../common/inc -I ../../../../../ports_module/cortex_m33/ac6/inc -I ../demo_secure_zone
-I./RTE/_FVP_Simulation_Model
-IC:/Users/your_path/AppData/Local/Arm/Packs/ARM/CMSIS/5.5.1/CMSIS/Core/Include
-IC:/Users/your_path/AppData/Local/Arm/Packs/ARM/CMSIS/5.5.1/Device/ARM/ARMCM33/Include
-D__UVISION_VERSION="531" -D_RTE_ -DARMCM33_DSP_FP_TZ -D_RTE_
-o ./Objects/*.o -MD
```

使用 AC6 的 Cortex-M33 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2                   VOID    *tx_thread_module_instance_ptr;         \
                                                VOID    *tx_thread_module_entry_info_ptr;       \
                                                ULONG   tx_thread_module_current_user_mode;     \
                                                ULONG   tx_thread_module_user_mode;             \
                                                ULONG   tx_thread_module_saved_lr;              \
                                                VOID    *tx_thread_module_kernel_stack_start;   \
                                                VOID    *tx_thread_module_kernel_stack_end;     \
                                                ULONG   tx_thread_module_kernel_stack_size;     \
                                                VOID    *tx_thread_module_stack_ptr;            \
                                                VOID    *tx_thread_module_stack_start;          \
                                                VOID    *tx_thread_module_stack_end;            \
                                                ULONG   tx_thread_module_stack_size;            \
                                                VOID    *tx_thread_module_reserved;
```

为使用 AC6 的 Cortex-M33 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、sample_threadx_module 项目和 demo_threadx_non-secure_zone 项目。

外部内存属性为使用 AC6 的 Cortex-M33 启用 API

IIII
TXM_MODULE_ATTRIBUTE_NON_SHAREABLE
TXM_MODULE_ATTRIBUTE_OUTER_SHAREABLE
TXM_MODULE_ATTRIBUTE_INNER_SHAREABLE
TXM_MODULE_ATTRIBUTE_READ_WRITE
TXM_MODULE_ATTRIBUTE_READ_ONLY

使用 GNU 的 Cortex-M33

使用 GNU 的 Cortex-M33 的模块报头

使用 GNU 的 Cortex-M33 的模块属性

BIT	"I"	II
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）

BIT	“[”	[[
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 GNU 的 Cortex-M33 的模块链接器
为使用 GNU 的 Cortex-M33 生成模块
使用 GNU 的 Cortex-M33 的线程扩展定义
为使用 GNU 的 Cortex-M33 生成模块管理器
外部内存属性为使用 GNU 的 Cortex-M33 启用 API

[[[[
TXM_MODULE_ATTRIBUTE_NON_SHAREABLE
TXM_MODULE_ATTRIBUTE_OUTER_SHAREABLE
TXM_MODULE_ATTRIBUTE_INNER_SHAREABLE
TXM_MODULE_ATTRIBUTE_READ_WRITE
TXM_MODULE_ATTRIBUTE_READ_ONLY

使用 IAR 的 Cortex-M33

使用 IAR 的 Cortex-M33 的模块报头

```
    SECTION .text:CODE

    AAPCS INTERWORK, ROPI, RWPI_COMPATIBLE,  VFP_COMPATIBLE
    PRESERVE8

    /* Define public symbols.  */

    PUBLIC __txm_module_preamble


    /* Define application-specific start/stop entry points for the module.  */

    EXTERN demo_module_start


    /* Define common external refrences.  */

    EXTERN  _txm_module_thread_shell_entry
    EXTERN  _txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                    // Module ID
    DC32      0x6                                           // Module Major Version
    DC32      0x1                                           // Module Minor Version
    DC32      32                                            // Module Preamble Size in 32-bit words
```

```
    DC32      52                                                // Module Preamble Size in 32-bit words
    DC32      0x12345678                                        // Module ID (application defined)
    DC32      0x00000007                                        // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bits 23-3: Reserved
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
    DC32      _txm_module_thread_shell_entry - . - 0            // Module Shell Entry Point
    DC32      demo_module_start - . - 0                         // Module Start Thread Entry Point
    DC32      0                                                 // Module Stop Thread Entry Point
    DC32      1                                                 // Module Start/Stop Thread Priority
    DC32      1024                                              // Module Start/Stop Thread Stack Size
    DC32      _txm_module_callback_request_thread_entry - . - 0 // Module Callback Thread Entry
    DC32      1                                                 // Module Callback Thread Priority
    DC32      1024                                              // Module Callback Thread Stack Size
    DC32      ROPI$$Length                                      // Module Code Size
    DC32      RWPI$$Length                                      // Module Data Size
    DC32      0                                                 // Reserved 0
    DC32      0                                                 // Reserved 1
    DC32      0                                                 // Reserved 2
    DC32      0                                                 // Reserved 3
    DC32      0                                                 // Reserved 4
    DC32      0                                                 // Reserved 5
    DC32      0                                                 // Reserved 6
    DC32      0                                                 // Reserved 7
    DC32      0                                                 // Reserved 8
    DC32      0                                                 // Reserved 9
    DC32      0                                                 // Reserved 10
    DC32      0                                                 // Reserved 11
    DC32      0                                                 // Reserved 12
    DC32      0                                                 // Reserved 13
    DC32      0                                                 // Reserved 14
    DC32      0                                                 // Reserved 15

    END
```

使用 IAR 的 Cortex-M33 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留

BIT	“[”	[[
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 IAR 的 Cortex-M33 的模块链接器

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\a_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x080f0000;
define symbol __ICFEDIT_region_ROM_end__   = 0x080fffff;
define symbol __ICFEDIT_region_RAM_start__ = 0x64002800;
define symbol __ICFEDIT_region_RAM_end__   = 0x64100000;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0;
define symbol __ICFEDIT_size_svcstack__ = 0;
define symbol __ICFEDIT_size_irqstack__ = 0;
define symbol __ICFEDIT_size_fiqstack__ = 0;
define symbol __ICFEDIT_size_undstack__ = 0;
define symbol __ICFEDIT_size_abtstack__ = 0;
define symbol __ICFEDIT_size_heap__     = 0x1000;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

//define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
//define block SVC_STACK with alignment = 8, size = __ICFEDIT_size_svcstack__ { };
//define block IRQ_STACK with alignment = 8, size = __ICFEDIT_size_irqstack__ { };
//define block FIQ_STACK with alignment = 8, size = __ICFEDIT_size_fiqstack__ { };
//define block UND_STACK with alignment = 8, size = __ICFEDIT_size_undstack__ { };
//define block ABT_STACK with alignment = 8, size = __ICFEDIT_size_abtstack__ { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

//place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

define movable block ROPI with alignment = 4, fixed order
{
  ro object txm_module_preamble.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base
{
  rw,
  block HEAP
};

place in ROM_region   { block ROPI };
place in RAM_region   { block RWPI };
```

为使用 IAR 的 Cortex-M33 生成模块

使用 IAR 的 Cortex-M33 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2                   VOID    *tx_thread_module_instance_ptr;         \
                                                VOID    *tx_thread_module_entry_info_ptr;       \
                                                ULONG   tx_thread_module_current_user_mode;     \
                                                ULONG   tx_thread_module_user_mode;             \
                                                ULONG   tx_thread_module_saved_lr;              \
                                                VOID    *tx_thread_module_kernel_stack_start;   \
                                                VOID    *tx_thread_module_kernel_stack_end;     \
                                                ULONG   tx_thread_module_kernel_stack_size;     \
                                                VOID    *tx_thread_module_stack_ptr;            \
                                                VOID    *tx_thread_module_stack_start;          \
                                                VOID    *tx_thread_module_stack_end;            \
                                                ULONG   tx_thread_module_stack_size;            \
                                                VOID    *tx_thread_module_reserved;             \
                                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 Cortex-M33 生成模块管理器

尚未提供示例工作区。即将推出。

外部内存属性为使用 IAR 的 Cortex-M33 启用 API

tttt
TXM_MODULE_ATTRIBUTE_NON_SHAREABLE
TXM_MODULE_ATTRIBUTE_OUTER_SHAREABLE
TXM_MODULE_ATTRIBUTE_INNER_SHAREABLE
TXM_MODULE_ATTRIBUTE_READ_WRITE
TXM_MODULE_ATTRIBUTE_READ_ONLY

Cortex-M4 处理器

使用 AC5 的 Cortex-M4

使用 AC5 的 Cortex-M4 的模块报头

```
    AREA Init, CODE, READONLY

    PRESERVE8

    /* Define public symbols.  */

    EXPORT __txm_module_preamble

    ; Define application-specific start/stop entry points for the module

    EXTERN demo_module_start

    /* Define common external references.  */

    IMPORT  _txm_module_thread_shell_entry
    IMPORT  _txm_module_callback_request_thread_entry
    IMPORT  |Image$$ER_RO$$Length|
    IMPORT  |Image$$ER_RW$$Length|
    IMPORT  |Image$$ER_RW$$RW$$Length|
    IMPORT  |Image$$ER_RW$$ZI$$Length|
    IMPORT  |Image$$ER_ZI$$ZI$$Length|

__txm_module_preamble
    DCD     0x4D4F4455                                          // Module ID
```

```
    DCD     0x6                                                 // Module Major Version
    DCD     0x1                                                 // Module Minor Version
    DCD     32                                                  // Module Preamble Size in 32-bit words
    DCD     0x12345678                                          // Module ID (application defined)
    DCD     0x01000007                                          // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bits 23-3: Reserved
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected)
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
    DCD     _txm_module_thread_shell_entry - __txm_module_preamble              // Module Shell Entry Point
    DCD     demo_module_start - __txm_module_preamble           // Module Start Thread Entry Point
    DCD     0                                                   // Module Stop Thread Entry Point
    DCD     1                                                   // Module Start/Stop Thread Priority
    DCD     1024                                                // Module Start/Stop Thread Stack Size
    DCD     _txm_module_callback_request_thread_entry - __txm_module_preamble   // Module Callback Thread
Entry
    DCD     1                                                   // Module Callback Thread Priority
    DCD     1024                                                // Module Callback Thread Stack Size
    DCD     |Image$$ER_RO$$Length| + |Image$$ER_RW$$Length|     // Module Code Size
    DCD     |Image$$ER_RW$$Length| + |Image$$ER_ZI$$ZI$$Length| // Module Data Size
    DCD     0                                                   // Reserved 0
    DCD     0                                                   // Reserved 1
    DCD     0                                                   // Reserved 2
    DCD     0                                                   // Reserved 3
    DCD     0                                                   // Reserved 4
    DCD     0                                                   // Reserved 5
    DCD     0                                                   // Reserved 6
    DCD     0                                                   // Reserved 7
    DCD     0                                                   // Reserved 8
    DCD     0                                                   // Reserved 9
    DCD     0                                                   // Reserved 10
    DCD     0                                                   // Reserved 11
    DCD     0                                                   // Reserved 12
    DCD     0                                                   // Reserved 13
    DCD     0                                                   // Reserved 14
    DCD     0                                                   // Reserved 15

    END
```

使用 AC5 的 Cortex-M4 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留

BIT	“[”	[[
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC5 的 Cortex-M4 的模块链接器

未提供示例链接器文件;链接在命令行上完成。请参阅下一节。

为使用 AC5 的 Cortex-M4 生成模块

请参阅示例 build_threadx_module_demo.bat:

```
armasm -g --cpreproc --cpu=cortex-m4 --fpu=vfpv4 --apcs=/interwork/ropi/rwpi txm_module_preamble.S
armcc  -g --cpu=cortex-m4 --fpu=vfpv4 -c --apcs=/interwork/ropi/rwpi --lower_ropi -I../inc -
I../../../../common_modules/inc -I../../../../common_modules/module_manager/inc -I../../../../common/inc
sample_threadx_module.c
armlink -d -o sample_threadx_module.axf --elf --ro=0x30000 --rw=0x40000 --first txm_module_preamble.o(Init)
--entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --list
sample_threadx_module.map txm_module_preamble.o sample_threadx_module.o txm.a
```

使用 AC5 的 Cortex-M4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 AC5 的 Cortex-M4 生成模块管理器

请参阅示例 build_threadx_module_manager_demo.bat:

```
armasm -g --cpreproc --cpu=cortex-m4 --fpu=vfpv4 --apcs=/interwork tx_initialize_low_level.S
armcc -g --cpu=cortex-m4 --fpu=vfpv4 -c -I../inc -I../../../../common_modules/inc -
I../../../../common_modules/module_manager/inc -I../../../../common/inc sample_threadx_module_manager.c
armlink -d -o sample_threadx_module_manager.axf --elf --ro 0x00000000 --first
tx_initialize_low_level.o(RESET) --remove --map --symbols --list sample_threadx_module_manager.map
tx_initialize_low_level.o sample_threadx_module_manager.o tx.a
```

外部内存属性为使用 AC5 的 Cortex-M4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 AC6 的 Cortex-M4

使用 AC6 的 Cortex-M4 的模块报头

```
    .text
    .align 4
    .syntax unified
    .section Init

    // Define public symbols
    .global __txm_module_preamble

    // Define application-specific start/stop entry points for the module
    .global demo_module_start

    // Define common external references
    .global  _txm_module_thread_shell_entry
    .global  _txm_module_callback_request_thread_entry

    .eabi_attribute Tag_ABI_PCS_RO_data, 1
    .eabi_attribute Tag_ABI_PCS_R9_use,  1
    .eabi_attribute Tag_ABI_PCS_RW_data, 2

__txm_module_preamble:
    .dc.l   0x4D4F4455                                          // Module ID
    .dc.l   0x6                                                 // Module Major Version
    .dc.l   0x1                                                 // Module Minor Version
    .dc.l   32                                                  // Module Preamble Size in 32-bit words
    .dc.l   0x12345678                                          // Module ID (application defined)
    .dc.l   0x01000007                                          // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected)
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
    .dc.l   _txm_module_thread_shell_entry - __txm_module_preamble // Module Shell Entry Point
    .dc.l   demo_module_start - __txm_module_preamble           // Module Start Thread Entry Point
    .dc.l   0                                                   // Module Stop Thread Entry Point
    .dc.l   1                                                   // Module Start/Stop Thread Priority
    .dc.l   1024                                                // Module Start/Stop Thread Stack Size
    .dc.l   _txm_module_callback_request_thread_entry - __txm_module_preamble // Module Callback Thread
Entry
    .dc.l   1                                                   // Module Callback Thread Priority
    .dc.l   1024                                                // Module Callback Thread Stack Size
    .dc.l   0x10000                                             // Module Code Size
    .dc.l   0x10000                                             // Module Data Size
    .dc.l   0                                                   // Reserved 0
    .dc.l   0                                                   // Reserved 1
    .dc.l   0                                                   // Reserved 2
    .dc.l   0                                                   // Reserved 3
    .dc.l   0                                                   // Reserved 4
    .dc.l   0                                                   // Reserved 5
    .dc.l   0                                                   // Reserved 6
    .dc.l   0                                                   // Reserved 7
    .dc.l   0                                                   // Reserved 8
    .dc.l   0                                                   // Reserved 9
    .dc.l   0                                                   // Reserved 10
    .dc.l   0                                                   // Reserved 11
    .dc.l   0                                                   // Reserved 12
    .dc.l   0                                                   // Reserved 13
    .dc.l   0                                                   // Reserved 14
    .dc.l   0                                                   // Reserved 15
```

使用 AC6 的 Cortex-M4 的模块属性

BIT	"["	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC6 的 Cortex-M4 的模块链接器

未使用链接器文件。请参阅项目设置。

为使用 AC6 的 Cortex-M4 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 AC6 的 Cortex-M4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 AC6 的 Cortex-M4 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 AC6 的 Cortex-M4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| |
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 GNU 的 Cortex-M4

使用 GNU 的 Cortex-M4 的模块报头

```
    .text
    .align 4
    .syntax unified

    /* Define public symbols.  */
    .global __txm_module_preamble

    /* Define application-specific start/stop entry points for the module.  */
    .global demo_module_start

    /* Define common external refrences.  */
    .global _txm_module_thread_shell_entry
    .global _txm_module_callback_request_thread_entry

__txm_module_preamble:
    .dc.l      0x4D4F4455                                       // Module ID
    .dc.l      0x6                                              // Module Major Version
    .dc.l      0x1                                              // Module Minor Version
    .dc.l      32                                               // Module Preamble Size in 32-bit words
    .dc.l      0x12345678                                       // Module ID (application defined)
    .dc.l      0x02000007                                       // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bits 23-3: Reserved
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
    .dc.l      _txm_module_thread_shell_entry - . - 0           // Module Shell Entry Point
    .dc.l      demo_module_start - . - 0                        // Module Start Thread Entry Point
    .dc.l      0                                                // Module Stop Thread Entry Point
    .dc.l      1                                                // Module Start/Stop Thread Priority
    .dc.l      1024                                             // Module Start/Stop Thread Stack Size
    .dc.l      _txm_module_callback_request_thread_entry - . - 0 // Module Callback Thread Entry
    .dc.l      1                                                // Module Callback Thread Priority
    .dc.l      1024                                             // Module Callback Thread Stack Size
    .dc.l      __code_size__                                    // Module Code Size
    .dc.l      __data_size__                                    // Module Data Size
    .dc.l      0                                                // Reserved 0
    .dc.l      0                                                // Reserved 1
    .dc.l      0                                                // Reserved 2
    .dc.l      0                                                // Reserved 3
    .dc.l      0                                                // Reserved 4
    .dc.l      0                                                // Reserved 5
    .dc.l      0                                                // Reserved 6
    .dc.l      0                                                // Reserved 7
    .dc.l      0                                                // Reserved 8
    .dc.l      0                                                // Reserved 9
    .dc.l      0                                                // Reserved 10
    .dc.l      0                                                // Reserved 11
    .dc.l      0                                                // Reserved 12
    .dc.l      0                                                // Reserved 13
    .dc.l      0                                                // Reserved 14
    .dc.l      0                                                // Reserved 15
```

使用 GNU 的 Cortex-M4 的模块属性

BIT	"["	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 GNU 的 Cortex-M4 的模块链接器

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x00030000, LENGTH = 0x00010000
  RAM   (wx) : ORIGIN = 0, LENGTH = 0x00100000
}


SECTIONS
{
  __FLASH_segment_start__ = 0x00030000;
  __FLASH_segment_end__   = 0x00040000;
  __RAM_segment_start__   = 0;
  __RAM_segment_end__     = 0x8000;

  __HEAPSIZE__ = 128;

  __preamble_load_start__ = __FLASH_segment_start__;
  .preamble __FLASH_segment_start__ : AT(__FLASH_segment_start__)
  {
    __preamble_start__ = .;
    *(.preamble .preamble.*)
  }
  __preamble_end__ = __preamble_start__ + SIZEOF(.preamble);

  __dynsym_load_start__ =  ALIGN(__preamble_end__ , 4);
  .dynsym ALIGN(__dynsym_load_start__ , 4) : AT(ALIGN(__dynsym_load_start__ , 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynsym))
    KEEP (*(.dynsym*))
    . = ALIGN(4);
  }
  __dynsym_end__ = __dynsym_load_start__ + SIZEOF(.dynsym);

  __dynstr_load_start__ =  ALIGN(__dynsym_end__ , 4);
  .dynstr ALIGN(__dynstr_load_start__ , 4) : AT(ALIGN(__dynstr_load_start__, 4))
  {
    . = ALIGN(4);
    KEEP (*(.dynstr))
    KEEP (*(.dynstr*))
    . = ALIGN(4);
  }
  __dynstr_end__ = __dynstr_load_start__ + SIZEOF(.dynstr);
```

```
__reldyn_load_start__ =  ALIGN(__dynstr_end__ , 4);
.rel.dyn ALIGN(__reldyn_load_start__ , 4) : AT(ALIGN(__reldyn_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.dyn))
  KEEP (*(.rel.dyn*))
  . = ALIGN(4);
}
__reldyn_end__ = __reldyn_load_start__ + SIZEOF(.rel.dyn);

__relplt_load_start__ =  ALIGN(__reldyn_end__ , 4);
.rel.plt ALIGN(__relplt_load_start__ , 4) : AT(ALIGN(__relplt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.plt))
  KEEP (*(.rel.plt*))
  . = ALIGN(4);
}
__relplt_end__ = __relplt_load_start__ + SIZEOF(.rel.plt);

__plt_load_start__ =  ALIGN(__relplt_end__ , 4);
.plt ALIGN(__plt_load_start__ , 4) : AT(ALIGN(__plt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.plt))
  KEEP (*(.plt*))
  . = ALIGN(4);
}
__plt_end__ = __plt_load_start__ + SIZEOF(.plt);

__interp_load_start__ =  ALIGN(__plt_end__ , 4);
.interp ALIGN(__interp_load_start__ , 4) : AT(ALIGN(__interp_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.interp))
  KEEP (*(.interp*))
  . = ALIGN(4);
}
__interp_end__ = __interp_load_start__ + SIZEOF(.interp);

__hash_load_start__ =  ALIGN(__interp_end__ , 4);
.hash ALIGN(__hash_load_start__ , 4) : AT(ALIGN(__hash_load_start__, 4))
{
  . = ALIGN(4);
  KEEP (*(.hash))
  KEEP (*(.hash*))
  . = ALIGN(4);
}
__hash_end__ = __hash_load_start__ + SIZEOF(.hash);

__text_load_start__ =  ALIGN(__hash_end__ , 4);
.text ALIGN(__text_load_start__ , 4) : AT(ALIGN(__text_load_start__, 4))
{
  __text_start__ = .;
  *(.text .text.* .glue_7t .glue_7 .gnu.linkonce.t.* .gcc_except_table  )
}
__text_end__ = __text_start__ + SIZEOF(.text);

__dtors_load_start__ = ALIGN(__text_end__ , 4);
.dtors ALIGN(__text_end__ , 4) : AT(ALIGN(__text_end__ , 4))
{
  __dtors_start__ = .;
  KEEP (*(SORT(.dtors.*))) KEEP (*(.dtors))
}
__dtors_end__ = __dtors_start__ + SIZEOF(.dtors);

__ctors_load_start__ = ALIGN(__dtors_end__ , 4);
.ctors ALIGN(__dtors_end__ , 4) : AT(ALIGN(__dtors_end__ , 4))
{
```

```
  __ctors_start__ = .;
  KEEP (*(SORT(.ctors.*))) KEEP (*(.ctors))
}
__ctors_end__ = __ctors_start__ + SIZEOF(.ctors);

__got_load_start__ = ALIGN(__ctors_end__ , 4);
.got ALIGN(__ctors_end__ , 4) : AT(ALIGN(__ctors_end__ , 4))
{
  . = ALIGN(4);
  _sgot = .;
  KEEP (*(.got))
  KEEP (*(.got*))
  . = ALIGN(4);
  _egot = .;
}
__got_end__ =  __got_load_start__ + SIZEOF(.got);

__rodata_load_start__ = ALIGN(__got_end__ , 4);
.rodata ALIGN(__got_end__ , 4) : AT(ALIGN(__got_end__ , 4))
{
  __rodata_start__ = .;
  *(.rodata .rodata.* .gnu.linkonce.r.*)
}
__rodata_end__ = __rodata_start__ + SIZEOF(.rodata);

__code_size__ =  __rodata_end__ - __FLASH_segment_start__;

__fast_load_start__ = ALIGN(__rodata_end__ , 4);

__fast_load_end__ = __fast_load_start__ + SIZEOF(.fast);

__new_got_start__ = ALIGN(__RAM_segment_start__ , 4);

__new_got_end__ =  __new_got_start__ + SIZEOF(.got);

.fast ALIGN(__new_got_end__ , 4) : AT(ALIGN(__rodata_end__ , 4))
{
  __fast_start__ = .;
  *(.fast .fast.*)
}
__fast_end__ = __fast_start__ + SIZEOF(.fast);

.fast_run ALIGN(__fast_end__ , 4) (NOLOAD) :
{
  __fast_run_start__ = .;
  . = MAX(__fast_run_start__ + SIZEOF(.fast), .);
}
__fast_run_end__ = __fast_run_start__ + SIZEOF(.fast_run);

__data_load_start__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4);
.data ALIGN(__fast_run_end__ , 4) : AT(ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4))
{
  __data_start__ = .;
  *(.data .data.* .gnu.linkonce.d.*)
}
__data_end__ = __data_start__ + SIZEOF(.data);

__data_load_end__ = __data_load_start__ + SIZEOF(.data);

__FLASH_segment_used_end__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4) + SIZEOF(.data);

.data_run ALIGN(__fast_run_end__ , 4) (NOLOAD) :
{
  __data_run_start__ = .;
  . = MAX(__data_run_start__ + SIZEOF(.data), .);
}
__data_run_end__ = __data_run_start__ + SIZEOF(.data_run);

__bss_load_start__ = ALIGN(__data_run_end__ , 4);
```

```
  .bss ALIGN(__data_run_end__ , 4) (NOLOAD) : AT(ALIGN(__data_run_end__ , 4))
  {
    __bss_start__ = .;
    *(.bss .bss.* .gnu.linkonce.b.*) *(COMMON)
  }
  __bss_end__ = __bss_start__ + SIZEOF(.bss);

  __non_init_load_start__ = ALIGN(__bss_end__ , 4);
  .non_init ALIGN(__bss_end__ , 4) (NOLOAD) : AT(ALIGN(__bss_end__ , 4))
  {
    __non_init_start__ = .;
    *(.non_init .non_init.*)
  }
  __non_init_end__ = __non_init_start__ + SIZEOF(.non_init);

  __heap_load_start__ = ALIGN(__non_init_end__ , 4);
  .heap ALIGN(__non_init_end__ , 4) (NOLOAD) : AT(ALIGN(__non_init_end__ , 4))
  {
    __heap_start__ = .;
    *(.heap)
    . = ALIGN(MAX(__heap_start__ + __HEAPSIZE__ , .), 4);
  }
  __heap_end__ = __heap_start__ + SIZEOF(.heap);

  __data_size__ =  __heap_end__ - __RAM_segment_start__;

}
```

为使用 GNU 的 Cortex-M4 生成模块

请参阅 build_threadx_module_sample.bat：

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base txm_module_preamble.s
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base gcc_setup.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base -I..\inc -I..\..\..\..\common\inc -I..\..\..\..\common_modules\inc
sample_threadx_module.c
arm-none-eabi-ld -A cortex-m4 -T sample_threadx_module.ld txm_module_preamble.o gcc_setup.o
sample_threadx_module.o -e _txm_module_thread_shell_entry txm.a -o sample_threadx_module.axf -M >
sample_threadx_module.map
```

使用 GNU 的 Cortex-M4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 GNU 的 Cortex-M4 生成模块管理器

请参阅 build_threadx_module_manager_sample.bat：

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -mthumb cortexm_vectors.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -mthumb cortexm_crt0.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -mthumb tx_initialize_low_level.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -mthumb -I..\inc -
I..\..\..\..\common\inc -I..\..\..\..\common_modules\inc sample_threadx_module_manager.c
arm-none-eabi-gcc -g -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=vfpv4 -mthumb -T sample_threadx.ld -
ereset_handler -nostartfiles -o sample_threadx_module_manager.out -Wl,-Map=sample_threadx_module_manager.map
cortexm_vectors.o cortexm_crt0.o tx_initialize_low_level.o sample_threadx_module_manager.o tx.a
```

外部内存属性为使用 GNU 的 Cortex-M4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 IAR 的 Cortex-M4

使用 IAR 的 Cortex-M4 的模块报头

```
    SECTION .text:CODE

    AAPCS INTERWORK, ROPI, RWPI_COMPATIBLE,  VFP_COMPATIBLE
    PRESERVE8

    /* Define public symbols.  */

    PUBLIC __txm_module_preamble


    /* Define application-specific start/stop entry points for the module.  */

    EXTERN demo_module_start


    /* Define common external references.  */

    EXTERN  _txm_module_thread_shell_entry
    EXTERN  _txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                    ; Module ID
    DC32      0x5                                           ; Module Major Version
    DC32      0x6                                           ; Module Minor Version
    DC32      32                                            ; Module Preamble Size in 32-bit words
    DC32      0x12345678                                    ; Module ID (application defined)
    DC32      0x00000007                                    ; Module Properties where:
                                                            ;   Bits 31-24: Compiler ID
                                                            ;           0 -> IAR
                                                            ;           1 -> ARM
                                                            ;           2 -> GNU
                                                            ;   Bits 23-3: Reserved
                                                            ;   Bit 2:  0 -> Disable shared/external
memory access
                                                            ;           1 -> Enable shared/external
memory access
                                                            ;   Bit 1:  0 -> No MPU protection
                                                            ;           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                            ;   Bit 0:  0 -> Privileged mode execution
                                                            ;           1 -> User mode execution


    DC32      _txm_module_thread_shell_entry - . - 0        ; Module Shell Entry Point
    DC32      demo_module_start - . - 0                     ; Module Start Thread Entry Point
    DC32      0                                             ; Module Stop Thread Entry Point
    DC32      1                                             ; Module Start/Stop Thread Priority
```

```
    DC32    1                                                   ; Module Start/Stop Thread Priority
    DC32    1024                                                ; Module Start/Stop Thread Stack Size
    DC32    _txm_module_callback_request_thread_entry - . - 0   ; Module Callback Thread Entry
    DC32    1                                                   ; Module Callback Thread Priority
    DC32    1024                                                ; Module Callback Thread Stack Size
    DC32    ROPI$$Length                                        ; Module Code Size
    DC32    RWPI$$Length                                        ; Module Data Size
    DC32    0                                                   ; Reserved 0
    DC32    0                                                   ; Reserved 1
    DC32    0                                                   ; Reserved 2
    DC32    0                                                   ; Reserved 3
    DC32    0                                                   ; Reserved 4
    DC32    0                                                   ; Reserved 5
    DC32    0                                                   ; Reserved 6
    DC32    0                                                   ; Reserved 7
    DC32    0                                                   ; Reserved 8
    DC32    0                                                   ; Reserved 9
    DC32    0                                                   ; Reserved 10
    DC32    0                                                   ; Reserved 11
    DC32    0                                                   ; Reserved 12
    DC32    0                                                   ; Reserved 13
    DC32    0                                                   ; Reserved 14
    DC32    0                                                   ; Reserved 15

    END
```

使用 IAR 的 Cortex-M4 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 IAR 的 Cortex-M4 的模块链接器

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\a_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x080f0000;
define symbol __ICFEDIT_region_ROM_end__   = 0x080fffff;
define symbol __ICFEDIT_region_RAM_start__ = 0x64002800;
define symbol __ICFEDIT_region_RAM_end__   = 0x64100000;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0;
define symbol __ICFEDIT_size_svcstack__ = 0;
define symbol __ICFEDIT_size_irqstack__ = 0;
define symbol __ICFEDIT_size_fiqstack__ = 0;
define symbol __ICFEDIT_size_undstack__ = 0;
define symbol __ICFEDIT_size_abtstack__ = 0;
define symbol __ICFEDIT_size_heap__     = 0x1000;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

//define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
//define block SVC_STACK with alignment = 8, size = __ICFEDIT_size_svcstack__ { };
//define block IRQ_STACK with alignment = 8, size = __ICFEDIT_size_irqstack__ { };
//define block FIQ_STACK with alignment = 8, size = __ICFEDIT_size_fiqstack__ { };
//define block UND_STACK with alignment = 8, size = __ICFEDIT_size_undstack__ { };
//define block ABT_STACK with alignment = 8, size = __ICFEDIT_size_abtstack__ { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

//place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

define movable block ROPI with alignment = 4, fixed order
{
  ro object txm_module_preamble_stm32f4xx.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base
{
  rw,
  block HEAP
};

place in ROM_region   { block ROPI };
place in RAM_region   { block RWPI };
```

为使用 IAR 的 Cortex-M4 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 Cortex-M4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                ULONG   tx_thread_module_saved_lr;              \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;             \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 Cortex-M4 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 Cortex-M4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

Cortex-M7 处理器

使用 AC5 的 Cortex-M7

使用 AC5 的 Cortex-M7 的模块报头

```
    AREA Init, CODE, READONLY

    PRESERVE8

    ; Define public symbols

    EXPORT __txm_module_preamble


    ; Define application-specific start/stop entry points for the module

    EXTERN demo_module_start


    ; Define common external references

    IMPORT  _txm_module_thread_shell_entry
    IMPORT  _txm_module_callback_request_thread_entry
    IMPORT  |Image$$ER_RO$$Length|
    IMPORT  |Image$$ER_RW$$Length|
    IMPORT  |Image$$ER_RW$$RW$$Length|
    IMPORT  |Image$$ER_RW$$ZI$$Length|
    IMPORT  |Image$$ER_ZI$$ZI$$Length|

__txm_module_preamble
    DCD     0x4D4F4455                                          ; Module ID
    DCD     0x5                                                 ; Module Major Version
    DCD     0x6                                                 ; Module Minor Version
    DCD     32                                                  ; Module Preamble Size in 32-bit words
    DCD     0x12345678                                          ; Module ID (application defined)
    DCD     0x01000007                                          ; Module Properties where:
                                                                ;   Bits 31-24: Compiler ID
                                                                ;           0 -> IAR
                                                                ;           1 -> ARM
                                                                ;           2 -> GNU
                                                                ;   Bit 0:  0 -> Privileged mode execution
                                                                ;           1 -> User mode execution
```

```
                                                        ;   Bit 1:  0 -> No MPU protection
                                                        ;           1 -> MPU protection (must have
user mode selected)
                                                        ;   Bit 2:  0 -> Disable shared/external
memory access
                                                        ;           1 -> Enable shared/external
memory access
    DCD     _txm_module_thread_shell_entry - __txm_module_preamble              ; Module Shell Entry Point
    DCD     demo_module_start - __txm_module_preamble           ; Module Start Thread Entry Point
    DCD     0                                                   ; Module Stop Thread Entry Point
    DCD     1                                                   ; Module Start/Stop Thread Priority
    DCD     1024                                                ; Module Start/Stop Thread Stack Size
    DCD     _txm_module_callback_request_thread_entry - __txm_module_preamble   ; Module Callback Thread
Entry
    DCD     1                                                   ; Module Callback Thread Priority
    DCD     1024                                                ; Module Callback Thread Stack Size
    DCD     |Image$$ER_RO$$Length| + |Image$$ER_RW$$Length|         ; Module Code Size
    DCD     |Image$$ER_RW$$Length| + |Image$$ER_ZI$$ZI$$Length| ; Module Data Size
    DCD     0                                                   ; Reserved 0
    DCD     0                                                   ; Reserved 1
    DCD     0                                                   ; Reserved 2
    DCD     0                                                   ; Reserved 3
    DCD     0                                                   ; Reserved 4
    DCD     0                                                   ; Reserved 5
    DCD     0                                                   ; Reserved 6
    DCD     0                                                   ; Reserved 7
    DCD     0                                                   ; Reserved 8
    DCD     0                                                   ; Reserved 9
    DCD     0                                                   ; Reserved 10
    DCD     0                                                   ; Reserved 11
    DCD     0                                                   ; Reserved 12
    DCD     0                                                   ; Reserved 13
    DCD     0                                                   ; Reserved 14
    DCD     0                                                   ; Reserved 15

    END
```

使用 AC5 的 Cortex-M7 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC5 的 Cortex-M7 的模块链接器

基于命令行生成，没有链接器文件示例。

为使用 AC5 的 Cortex-M7 生成模块

生成使用 AC5 的 Cortex-M7 模块的简单命令行示例：

```
armasm -g --cpu=cortex-m7 --fpu=softvfp --apcs=/interwork/ropi/rwpi txm_module_preamble.s
armcc  -g --cpu=cortex-m7 --fpu=softvfp -c --apcs=/interwork/ropi/rwpi --lower_ropi -I../inc -
I../../../../common_modules/inc -I../../../../common_modules/module_manager/inc -I../../../../common/inc
sample_threadx_module.c
armlink -d -o sample_threadx_module.axf --elf --ro 0 --first txm_module_preamble.o(Init) --
entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --list sample_threadx_module.map
txm_module_preamble.o sample_threadx_module.o txm.a
```

使用 AC5 的 Cortex-M7 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                ULONG   tx_thread_module_saved_lr;              \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;
```

为使用 AC5 的 Cortex-M7 生成模块管理器

生成使用 AC5 的 Cortex-M7 模块管理器的简单命令行示例:

```
armasm -g --cpu=cortex-m7 --fpu=softvfp --apcs=interwork tx_initialize_low_level.s
armcc -g --cpu=cortex-m7 --fpu=softvfp -c demo_threadx_module_manager.c
armcc -g --cpu=cortex-m7 --fpu=softvfp -c module_code.c
armlink -d -o demo_threadx_module_manager.axf --elf --ro 0x00000000 --first tx_initialize_low_level.o(RESET)
--remove --map --symbols --list demo_threadx_module_manager.map tx_initialize_low_level.o
demo_threadx_module_manager.o module_code.o tx.a
```

外部内存属性为使用 AC5 的 Cortex-M7 启用 API

使用 AC6 的 Cortex-M7

使用 AC6 的 Cortex-M7 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC6 的 Cortex-M7 的模块链接器

未使用链接器文件。请参阅项目设置。

为使用 AC6 的 Cortex-M7 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 AC6 的 Cortex-M7 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 AC6 的 Cortex-M7 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 AC6 的 Cortex-M7 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| |
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 GNU 的 Cortex-M7

使用 GNU 的 Cortex-M7 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 GNU 的 Cortex-M7 的模块链接器

```
MEMORY
{
  FLASH (rx) : ORIGIN = 0x00030000, LENGTH = 0x00010000
  RAM   (wx) : ORIGIN = 0, LENGTH = 0x00100000
}


SECTIONS
{
  __FLASH_segment_start__ = 0x00030000;
  __FLASH_segment_end__   = 0x00040000;
  __RAM_segment_start__   = 0;
  __RAM_segment_end__     = 0x8000;
```

```
__HEAPSIZE__ = 128;

__preamble_load_start__ = __FLASH_segment_start__;
.preamble __FLASH_segment_start__ : AT(__FLASH_segment_start__)
{
  __preamble_start__ = .;
  *(.preamble .preamble.*)
}
__preamble_end__ = __preamble_start__ + SIZEOF(.preamble);

__dynsym_load_start__ =  ALIGN(__preamble_end__ , 4);
.dynsym ALIGN(__dynsym_load_start__ , 4) : AT(ALIGN(__dynsym_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.dynsym))
  KEEP (*(.dynsym*))
  . = ALIGN(4);
}
__dynsym_end__ = __dynsym_load_start__ + SIZEOF(.dynsym);

__dynstr_load_start__ =  ALIGN(__dynsym_end__ , 4);
.dynstr ALIGN(__dynstr_load_start__ , 4) : AT(ALIGN(__dynstr_load_start__, 4))
{
  . = ALIGN(4);
  KEEP (*(.dynstr))
  KEEP (*(.dynstr*))
  . = ALIGN(4);
}
__dynstr_end__ = __dynstr_load_start__ + SIZEOF(.dynstr);

__reldyn_load_start__ =  ALIGN(__dynstr_end__ , 4);
.rel.dyn ALIGN(__reldyn_load_start__ , 4) : AT(ALIGN(__reldyn_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.dyn))
  KEEP (*(.rel.dyn*))
  . = ALIGN(4);
}
__reldyn_end__ = __reldyn_load_start__ + SIZEOF(.rel.dyn);

__relplt_load_start__ =  ALIGN(__reldyn_end__ , 4);
.rel.plt ALIGN(__relplt_load_start__ , 4) : AT(ALIGN(__relplt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.rel.plt))
  KEEP (*(.rel.plt*))
  . = ALIGN(4);
}
__relplt_end__ = __relplt_load_start__ + SIZEOF(.rel.plt);

__plt_load_start__ =  ALIGN(__relplt_end__ , 4);
.plt ALIGN(__plt_load_start__ , 4) : AT(ALIGN(__plt_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.plt))
  KEEP (*(.plt*))
  . = ALIGN(4);
}
__plt_end__ = __plt_load_start__ + SIZEOF(.plt);

__interp_load_start__ =  ALIGN(__plt_end__ , 4);
.interp ALIGN(__interp_load_start__ , 4) : AT(ALIGN(__interp_load_start__ , 4))
{
  . = ALIGN(4);
  KEEP (*(.interp))
  KEEP (*(.interp*))
  . = ALIGN(4);
}
```

```
__interp_end__ = __interp_load_start__ + SIZEOF(.interp);

__hash_load_start__ =  ALIGN(__interp_end__ , 4);
.hash ALIGN(__hash_load_start__ , 4) : AT(ALIGN(__hash_load_start__, 4))
{
  . = ALIGN(4);
  KEEP (*(.hash))
  KEEP (*(.hash*))
  . = ALIGN(4);
}
__hash_end__ = __hash_load_start__ + SIZEOF(.hash);

__text_load_start__ =  ALIGN(__hash_end__ , 4);
.text ALIGN(__text_load_start__ , 4) : AT(ALIGN(__text_load_start__, 4))
{
  __text_start__ = .;
  *(.text .text.* .glue_7t .glue_7 .gnu.linkonce.t.* .gcc_except_table  )
}
__text_end__ = __text_start__ + SIZEOF(.text);

__dtors_load_start__ = ALIGN(__text_end__ , 4);
.dtors ALIGN(__text_end__ , 4) : AT(ALIGN(__text_end__ , 4))
{
  __dtors_start__ = .;
  KEEP (*(SORT(.dtors.*))) KEEP (*(.dtors))
}
__dtors_end__ = __dtors_start__ + SIZEOF(.dtors);

__ctors_load_start__ = ALIGN(__dtors_end__ , 4);
.ctors ALIGN(__dtors_end__ , 4) : AT(ALIGN(__dtors_end__ , 4))
{
  __ctors_start__ = .;
  KEEP (*(SORT(.ctors.*))) KEEP (*(.ctors))
}
__ctors_end__ = __ctors_start__ + SIZEOF(.ctors);

__got_load_start__ = ALIGN(__ctors_end__ , 4);
.got ALIGN(__ctors_end__ , 4) : AT(ALIGN(__ctors_end__ , 4))
{
  . = ALIGN(4);
  _sgot = .;
  KEEP (*(.got))
  KEEP (*(.got*))
  . = ALIGN(4);
  _egot = .;
}
__got_end__ =  __got_load_start__ + SIZEOF(.got);

__rodata_load_start__ = ALIGN(__got_end__ , 4);
.rodata ALIGN(__got_end__ , 4) : AT(ALIGN(__got_end__ , 4))
{
  __rodata_start__ = .;
  *(.rodata .rodata.* .gnu.linkonce.r.*)
}
__rodata_end__ = __rodata_start__ + SIZEOF(.rodata);

__code_size__ =  __rodata_end__ - __FLASH_segment_start__;

__fast_load_start__ = ALIGN(__rodata_end__ , 4);

__fast_load_end__ = __fast_load_start__ + SIZEOF(.fast);

__new_got_start__ = ALIGN(__RAM_segment_start__ , 4);

__new_got_end__ =  __new_got_start__ + SIZEOF(.got);

.fast ALIGN(__new_got_end__ , 4) : AT(ALIGN(__rodata_end__ , 4))
{
    fast_start__ = .;
```

```
    __fast_start__ = .;
    *(.fast .fast.*)
  }
  __fast_end__ = __fast_start__ + SIZEOF(.fast);

  .fast_run ALIGN(__fast_end__ , 4) (NOLOAD) :
  {
    __fast_run_start__ = .;
    . = MAX(__fast_run_start__ + SIZEOF(.fast), .);
  }
  __fast_run_end__ = __fast_run_start__ + SIZEOF(.fast_run);

  __data_load_start__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4);
  .data ALIGN(__fast_run_end__ , 4) : AT(ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4))
  {
    __data_start__ = .;
    *(.data .data.* .gnu.linkonce.d.*)
  }
  __data_end__ = __data_start__ + SIZEOF(.data);

  __data_load_end__ = __data_load_start__ + SIZEOF(.data);

  __FLASH_segment_used_end__ = ALIGN(__fast_load_start__ + SIZEOF(.fast) , 4) + SIZEOF(.data);

  .data_run ALIGN(__fast_run_end__ , 4) (NOLOAD) :
  {
    __data_run_start__ = .;
    . = MAX(__data_run_start__ + SIZEOF(.data), .);
  }
  __data_run_end__ = __data_run_start__ + SIZEOF(.data_run);

  __bss_load_start__ = ALIGN(__data_run_end__ , 4);
  .bss ALIGN(__data_run_end__ , 4) (NOLOAD) : AT(ALIGN(__data_run_end__ , 4))
  {
    __bss_start__ = .;
    *(.bss .bss.* .gnu.linkonce.b.*) *(COMMON)
  }
  __bss_end__ = __bss_start__ + SIZEOF(.bss);

  __non_init_load_start__ = ALIGN(__bss_end__ , 4);
  .non_init ALIGN(__bss_end__ , 4) (NOLOAD) : AT(ALIGN(__bss_end__ , 4))
  {
    __non_init_start__ = .;
    *(.non_init .non_init.*)
  }
  __non_init_end__ = __non_init_start__ + SIZEOF(.non_init);

  __heap_load_start__ = ALIGN(__non_init_end__ , 4);
  .heap ALIGN(__non_init_end__ , 4) (NOLOAD) : AT(ALIGN(__non_init_end__ , 4))
  {
    __heap_start__ = .;
    *(.heap)
    . = ALIGN(MAX(__heap_start__ + __HEAPSIZE__ , .), 4);
  }
  __heap_end__ = __heap_start__ + SIZEOF(.heap);

  __data_size__ =  __heap_end__ - __RAM_segment_start__;

}
```

为使用 GNU 的 Cortex-M7 生成模块

请参阅 build_threadx_module_sample.bat:

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base txm_module_preamble.s
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base gcc_setup.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -fpie -fno-plt -mpic-data-is-text-
relative -msingle-pic-base -I..\inc -I..\..\..\..\common\inc -I..\..\..\..\common_modules\inc
sample_threadx_module.c
arm-none-eabi-ld -A cortex-m7 -T sample_threadx_module.ld txm_module_preamble.o gcc_setup.o
sample_threadx_module.o -e _txm_module_thread_shell_entry txm.a -o sample_threadx_module.axf -M >
sample_threadx_module.map
```

使用 GNU 的 Cortex-M7 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2               VOID    *tx_thread_module_instance_ptr;         \
                                            VOID    *tx_thread_module_entry_info_ptr;       \
                                            ULONG   tx_thread_module_current_user_mode;     \
                                            ULONG   tx_thread_module_user_mode;             \
                                            ULONG   tx_thread_module_saved_lr;              \
                                            VOID    *tx_thread_module_kernel_stack_start;   \
                                            VOID    *tx_thread_module_kernel_stack_end;     \
                                            ULONG   tx_thread_module_kernel_stack_size;     \
                                            VOID    *tx_thread_module_stack_ptr;            \
                                            VOID    *tx_thread_module_stack_start;          \
                                            VOID    *tx_thread_module_stack_end;            \
                                            ULONG   tx_thread_module_stack_size;            \
                                            VOID    *tx_thread_module_reserved;
```

为使用 GNU 的 Cortex-M7 生成模块管理器

请参阅 build_threadx_module_manager_sample.bat:

```
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -mthumb -I..\inc -
I..\..\..\..\common\inc -I..\..\..\..\common_modules\inc sample_threadx_module_manager.c
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -mthumb tx_simulator_startup.S
arm-none-eabi-gcc -c -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -mthumb cortexm_crt0.S
arm-none-eabi-gcc -g -mcpu=cortex-m7 -mfloat-abi=hard -mfpu=fpv5-d16 -mthumb -nostartfiles -ereset_handler -
T sample_threadx.ld tx_simulator_startup.o cortexm_crt0.o sample_threadx_module_manager.o tx.a -o
sample_threadx_module_manager.axf -Wl,-Map=sample_threadx_module_manager.map
```

外部内存属性为使用 GNU 的 Cortex-M7 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 IAR 的 Cortex-M7

使用 IAR 的 Cortex-M7 的模块报头

```
    SECTION .text:CODE

    AAPCS INTERWORK, ROPI, RWPI_COMPATIBLE,  VFP_COMPATIBLE
    PRESERVE8

    /* Define public symbols.  */

    PUBLIC __txm_module_preamble


    /* Define application-specific start/stop entry points for the module.  */

    EXTERN demo_module_start


    /* Define common external references.  */
```

```
    EXTERN  _txm_module_thread_shell_entry
    EXTERN  _txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                    ; Module ID
    DC32      0x5                                           ; Module Major Version
    DC32      0x6                                           ; Module Minor Version
    DC32      32                                            ; Module Preamble Size in 32-bit words
    DC32      0x12345678                                    ; Module ID (application defined)
    DC32      0x00000007                                    ; Module Properties where:
                                                            ;   Bits 31-24: Compiler ID
                                                            ;           0 -> IAR
                                                            ;           1 -> ARM
                                                            ;           2 -> GNU
                                                            ;   Bits 23-3: Reserved
                                                            ;   Bit 2:  0 -> Disable shared/external
memory access
                                                            ;           1 -> Enable shared/external
memory access
                                                            ;   Bit 1:  0 -> No MPU protection
                                                            ;           1 -> MPU protection (must have
user mode selected - bit 0 set)
                                                            ;   Bit 0:  0 -> Privileged mode execution
                                                            ;           1 -> User mode execution


    DC32      _txm_module_thread_shell_entry - . - 0        ; Module Shell Entry Point
    DC32      demo_module_start - . - 0                     ; Module Start Thread Entry Point
    DC32      0                                             ; Module Stop Thread Entry Point
    DC32      1                                             ; Module Start/Stop Thread Priority
    DC32      1024                                          ; Module Start/Stop Thread Stack Size
    DC32      _txm_module_callback_request_thread_entry - . - 0 ; Module Callback Thread Entry
    DC32      1                                             ; Module Callback Thread Priority
    DC32      1024                                          ; Module Callback Thread Stack Size
    DC32      ROPI$$Length                                  ; Module Code Size
    DC32      RWPI$$Length                                  ; Module Data Size
    DC32      0                                             ; Reserved 0
    DC32      0                                             ; Reserved 1
    DC32      0                                             ; Reserved 2
    DC32      0                                             ; Reserved 3
    DC32      0                                             ; Reserved 4
    DC32      0                                             ; Reserved 5
    DC32      0                                             ; Reserved 6
    DC32      0                                             ; Reserved 7
    DC32      0                                             ; Reserved 8
    DC32      0                                             ; Reserved 9
    DC32      0                                             ; Reserved 10
    DC32      0                                             ; Reserved 11
    DC32      0                                             ; Reserved 12
    DC32      0                                             ; Reserved 13
    DC32      0                                             ; Reserved 14
    DC32      0                                             ; Reserved 15

    END
```

使用 IAR 的 Cortex-M7 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行

BIT	“[”	[[
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 IAR 的 Cortex-M7 的模块链接器

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__     = 0x00400000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_RAM_start__ = 0x20450000;
define symbol __ICFEDIT_region_RAM_end__   = 0x2045f000 -1;
define symbol __ICFEDIT_region_RAM_NOCACHE_start__ = 0x2045f000;
define symbol __ICFEDIT_region_RAM_NOCACHE_end__   = 0x20460000 -1;
define symbol __ICFEDIT_region_ROM_start__ = 0x00500000;
define symbol __ICFEDIT_region_ROM_end__   = 0x00600000 -1;
define symbol __ICFEDIT_region_SDRAM_start__ = 0x70000000;
define symbol __ICFEDIT_region_SDRAM_end__   = 0x70200000 -1;

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__      = 0x2000;
define symbol __ICFEDIT_size_heap__        = 0x1000;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region RAM_region    = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__];
define region RAM_NOCACHE_region = mem:[from __ICFEDIT_region_RAM_NOCACHE_start__ to
__ICFEDIT_region_RAM_NOCACHE_end__];
define region ROM_region    = mem:[from __ICFEDIT_region_ROM_start__ to __ICFEDIT_region_ROM_end__];
define region SDRAM_region  = mem:[from __ICFEDIT_region_SDRAM_start__ to __ICFEDIT_region_SDRAM_end__];

define block HEAP   with alignment = 8, size = __ICFEDIT_size_heap__   { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

define movable block ROPI with alignment = 4, fixed order
{
  ro object txm_module_preamble.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base
{
  rw,
  block HEAP
};

place in ROM_region   { block ROPI };
place in RAM_region   { block RWPI };
```

为使用 IAR 的 Cortex-M7 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 Cortex-M7 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                ULONG   tx_thread_module_saved_lr;              \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;             \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 Cortex-M7 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 Cortex-M7 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

Cortex-R4 处理器

使用 AC6 的 Cortex-R4

使用 AC6 的 Cortex-R4 的模块报头

```
    .text

/* Define common external references.  */

.global     _txm_module_thread_shell_entry
.global     demo_module_start
.global     _txm_module_callback_request_thread_entry
.global     Image$$ER_RO$$Length
.global     Image$$ER_RW$$Length
.global     Image$$ER_ZI$$ZI$$Length

/* Stack aligned, ROPI and RWPI, R9 used as data offset register.  */
.eabi_attribute Tag_ABI_align_preserved, 1
.eabi_attribute Tag_ABI_PCS_RO_data, 1
.eabi_attribute Tag_ABI_PCS_R9_use,  1
.eabi_attribute Tag_ABI_PCS_RW_data, 2

__txm_module_preamble:
.word   0x4D4F4455                                      /* Module ID  */
.word   0x5                                             /* Module Major Version  */
.word   0x3                                             /* Module Minor Version  */
.word   32                                              /* Module Preamble Size in 32-bit words  */
.word   0x12345678                                      /* Module ID (application defined)  */
.word   0x01000001                                      /* Module Properties where:
                                                           Bits 31-24: Compiler ID
                                                                   0 -> IAR
                                                                   1 -> RVDS/ARM
                                                                   2 -> GNU
                                                           Bits 23-1:  Reserved
                                                           Bit 0:  0 -> Privileged mode execution (no
MMU protection)
                                                                   1 -> User mode execution (MMU
protection)  */
.word   _txm_module_thread_shell_entry - __txm_module_preamble  /* Module Shell Entry Point  */
.word   demo_module_start - __txm_module_preamble           /* Module Start Thread Entry Point  */
.word   0                                                   /* Module Stop Thread Entry Point   */
.word   1                                                   /* Module Start/Stop Thread Priority  */
.word   1024                                                /* Module Start/Stop Thread Stack Size  */
.word   _txm_module_callback_request_thread_entry - __txm_module_preamble   /* Module Callback Thread Entry
*/
.word   1                                                   /* Module Callback Thread Priority  */
.word   1024                                                /* Module Callback Thread Stack Size  */
.word   9000                                                /* Module Code Size */
.word   11000                                               /* Module Data Size */
.word   0                                                   /* Reserved 0  */
.word   0                                                   /* Reserved 1  */
.word   0                                                   /* Reserved 2  */
.word   0                                                   /* Reserved 3  */
.word   0                                                   /* Reserved 4  */
.word   0                                                   /* Reserved 5  */
.word   0                                                   /* Reserved 6  */
.word   0                                                   /* Reserved 7  */
.word   0                                                   /* Reserved 8  */
.word   0                                                   /* Reserved 9  */
.word   0                                                   /* Reserved 10  */
.word   0                                                   /* Reserved 11  */
.word   0                                                   /* Reserved 12  */
.word   0                                                   /* Reserved 13  */
.word   0                                                   /* Reserved 14  */
.word   0                                                   /* Reserved 15  */
```

使用 AC6 的 Cortex-R4 的模块属性

BIT	"["	[[
0	0 1	特权模式执行 用户模式执行
[23-1]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 AC6 的 Cortex-R4 的模块链接器

基于命令行生成，没有链接器文件示例。

为使用 AC6 的 Cortex-R4 生成模块

生成使用 AC6 的 Cortex-R4 模块的简单命令行示例：

```
armclang -c -g -fropi -frwpi --target=arm-arm-none-eabi -mcpu=cortex-r4 demo_threadx_module.c
armclang -c -g -fropi -frwpi --target=arm-arm-none-eabi -mcpu=cortex-r4 txm_module_preamble.S
armclang -c -g -fropi -frwpi --target=arm-arm-none-eabi -mcpu=cortex-r4 semihosting.c
armlink -d -o demo_threadx_module.axf --elf --ro 0x00100000 --first txm_module_preamble.o --
entry=_txm_module_thread_shell_entry --ropi --rwpi --remove --map --symbols --datacompressor=off --list
demo_threadx_module.map txm_module_preamble.o demo_threadx_module.o semihosting.o txm.a
```

使用 AC6 的 Cortex-R4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   ULONG   tx_thread_vfp_enable;                   \
                                VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;
```

为使用 AC6 的 Cortex-R4 生成模块管理器

生成使用 AC6 的 Cortex-R4 模块管理器的简单命令行示例：

```
armclang -c -g --target=arm-arm-none-eabi -mcpu=cortex-r4 demo_threadx_module_manager.c
armclang -c -g --target=arm-arm-none-eabi -mcpu=cortex-r4 timer.c
armclang -c -g --target=arm-arm-none-eabi -mcpu=cortex-r4 gic.c
armclang -c -g --target=arm-arm-none-eabi -mcpu=cortex-r4 tx_initialize_low_level.S
armclang -c -g --target=arm-arm-none-eabi -mcpu=cortex-r4 startup.S
armlink -d -o demo_threadx_module_manager.axf --elf --scatter=demo_threadx.scat --remove --map --symbols --
list demo_threadx_module_manager.map startup.o timer.o gic.o demo_threadx_module_manager.o
tx_initialize_low_level.o tx.a
```

外部内存属性为使用 AC6 的 Cortex-R4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

使用 IAR 的 Cortex-R4

使用 IAR 的 Cortex-R4 的模块报头

```
    SECTION .text:CODE

    AAPCS INTERWORK, ROPI, RWPI_COMPATIBLE,  VFP_COMPATIBLE
    PRESERVE8

    /* Define public symbols.  */
    PUBLIC __txm_module_preamble

    /* Define application-specific start/stop entry points for the module.  */
    EXTERN demo_module_start

    /* Define common external references.  */
    EXTERN  _txm_module_thread_shell_entry
    EXTERN  _txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                    ; Module ID
    DC32      0x5                                           ; Module Major Version
    DC32      0x6                                           ; Module Minor Version
    DC32      32                                            ; Module Preamble Size in 32-bit words
    DC32      0x12345678                                    ; Module ID (application defined)
    DC32      0x00000001                                    ; Module Properties where:
                                                            ;   Bits 31-24: Compiler ID
                                                            ;           0 -> IAR
                                                            ;           1 -> ARM
                                                            ;           2 -> GNU
                                                            ;   Bits 23-1: Reserved
                                                            ;   Bit 0:  0 -> Privileged mode execution
(no MPU protection)
                                                            ;           1 -> User mode execution (MPU
protection)
    DC32      _txm_module_thread_shell_entry - . - 0        ; Module Shell Entry Point
    DC32      demo_module_start - . - 0                     ; Module Start Thread Entry Point
    DC32      0                                             ; Module Stop Thread Entry Point
    DC32      1                                             ; Module Start/Stop Thread Priority
    DC32      1024                                          ; Module Start/Stop Thread Stack Size
    DC32      _txm_module_callback_request_thread_entry - . - 0 ; Module Callback Thread Entry
    DC32      1                                             ; Module Callback Thread Priority
    DC32      1024                                          ; Module Callback Thread Stack Size
    DC32      ROPI$$Length                                  ; Module Code Size
    DC32      RWPI$$Length                                  ; Module Data Size
    DC32      0                                             ; Reserved 0
    DC32      0                                             ; Reserved 1
    DC32      0                                             ; Reserved 2
    DC32      0                                             ; Reserved 3
    DC32      0                                             ; Reserved 4
    DC32      0                                             ; Reserved 5
    DC32      0                                             ; Reserved 6
    DC32      0                                             ; Reserved 7
    DC32      0                                             ; Reserved 8
    DC32      0                                             ; Reserved 9
    DC32      0                                             ; Reserved 10
    DC32      0                                             ; Reserved 11
    DC32      0                                             ; Reserved 12
    DC32      0                                             ; Reserved 13
    DC32      0                                             ; Reserved 14
    DC32      0                                             ; Reserved 15

    END
```

使用 IAR 的 Cortex-R4 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
[23-1]	0	保留
[31-24]	 0x00 0x01 0x02	■ ID IAR ARM GNU

使用 IAR 的 Cortex-R4 的模块链接器

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\a_v1_0.xml" */
/*-Specials-*/
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x00100000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0013FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x08000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x0800FFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0;
define symbol __ICFEDIT_size_svcstack__ = 0;
define symbol __ICFEDIT_size_irqstack__ = 0;
define symbol __ICFEDIT_size_fiqstack__ = 0;
define symbol __ICFEDIT_size_undstack__ = 0;
define symbol __ICFEDIT_size_abtstack__ = 0;
define symbol __ICFEDIT_size_heap__     = 0x100;
/**** End of ICF editor section. ###ICF###*/

define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

initialize by copy { readwrite };
do not initialize  { section .noinit };

define movable block ROPI with alignment = 4, fixed order
{
  ro object txm_module_preamble.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base
{
  rw,
  block HEAP
};

place in ROM_region   { block ROPI };
place in RAM_region   { block RWPI };
```

为使用 IAR 的 Cortex-R4 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 Cortex-R4 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   ULONG   tx_thread_vfp_enable;                  \
                                VOID    *tx_thread_module_instance_ptr;        \
                                VOID    *tx_thread_module_entry_info_ptr;      \
                                ULONG   tx_thread_module_current_user_mode;    \
                                ULONG   tx_thread_module_user_mode;            \
                                VOID    *tx_thread_module_kernel_stack_start;  \
                                VOID    *tx_thread_module_kernel_stack_end;    \
                                ULONG   tx_thread_module_kernel_stack_size;    \
                                VOID    *tx_thread_module_stack_ptr;           \
                                VOID    *tx_thread_module_stack_start;         \
                                VOID    *tx_thread_module_stack_end;           \
                                ULONG   tx_thread_module_stack_size;           \
                                VOID    *tx_thread_module_reserved;            \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 Cortex-R4 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 Cortex-R4 启用 API

模块始终具有读取共享内存的权限。| 属性参数 | 含义 | |---|---| | TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE | 写入权限 |

MCF544xx 处理器

使用 GHS 的 MCF544xx

使用 GHS 的 MCF544xx 的模块报头

```
    SECT    .preamble, x

    /* Define public symbols.  */
    XDEF __txm_module_preamble

__txm_module_preamble:
    DC.L    0x4D4F4455                              /* Module ID                              */
    DC.L    0x5                                     /* Module Major Version                   */
    DC.L    0x3                                     /* Module Minor Version                   */
    DC.L    32                                      /* Module Preamble Size in 32-bit words   */
    DC.L    0x12345678                              /* Module ID (application defined)        */
    DC.L    0x03000003                              /* Module Properties where:               */
                                                    /* Bits 31-24: Compiler ID                */
                                                    /*           3 -> GHS                     */
                                                    /* Bits 23-2: Reserved                    */
                                                    /* Bit 1:  0 -> No MMU protection         */
                                                    /*         1 -> MMU protection (must have */
                                                    /*              user mode selected)       */
                                                    /* Bit 0:  0 -> Supervisor mode execution */
                                                    /*         1 -> User mode execution       */
    DC.L    _txm_module_thread_shell_entry          /* Module Shell Entry Point               */
    DC.L    demo_module_start                       /* Module Start Thread Entry Point        */
    DC.L    0                                       /* Module Stop Thread Entry Point         */
    DC.L    1                                       /* Module Start/Stop Thread Priority      */
    DC.L    2048                                    /* Module Start/Stop Thread Stack Size    */
    DC.L    _txm_module_callback_request_thread_entry /* Module Callback Thread Entry         */
    DC.L    1                                       /* Module Callback Thread Priority        */
    DC.L    2048                                    /* Module Callback Thread Stack Size      */
    DC.L    module_code_size                        /* Module Code Size                       */
    DC.L    module_data_size                        /* Module Data Size                       */
    DC.L    0                                       /* Reserved 0                             */
    DC.L    0                                       /* Reserved 1                             */
    DC.L    0                                       /* Reserved 2                             */
    DC.L    0                                       /* Reserved 3                             */
    DC.L    0                                       /* Reserved 4                             */
    DC.L    0                                       /* Reserved 5                             */
    DC.L    0                                       /* Reserved 6                             */
    DC.L    0                                       /* Reserved 7                             */
    DC.L    0                                       /* Reserved 8                             */
    DC.L    0                                       /* Reserved 9                             */
    DC.L    0                                       /* Reserved 10                            */
    DC.L    0                                       /* Reserved 11                            */
    DC.L    0                                       /* Reserved 12                            */
    DC.L    0                                       /* Reserved 13                            */
    DC.L    0                                       /* Reserved 14                            */
    DC.L    0                                       /* Reserved 15                            */

    END
```

使用 GHS 的 MCF544xx 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MMU 保护 MMU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留

BIT	"["	[[
[31-24]	0x03	■ ID GHS

使用 GHS 的 MCF544xx 的模块链接器

```
DEFAULTS {

    heap_reserve =  256
    stack_reserve = 256
}

//
// Program layout for PIC and PID built programs.
//

-sec
{
//
// The data segment
//
    .pidbase                0x00000000 :
    .sdabase                           :
    .sbss                   NOCLEAR    :
    .sdata                             :
    .data                   NOLOAD     :
    .bss                    NOCLEAR    :
    .heap                   ALIGN(16) PAD( heap_reserve +
        // Add space for call-graph profiling if used:
        (isdefined(__ghs_indgcount)?(2000+(sizeof(.text)/2)):0) +
        // Add estimated space for call-count profiling if used:
        (isdefined(__ghs_indmcount)?10000:0) )
                                           :
    .stack      ALIGN(16) PAD(stack_reserve) :

    module_data_size = sizeof(.pidbase) + sizeof(.sdabase) + sizeof(.sbss) + sizeof(.sdata) + sizeof(.data)
+ sizeof(.bss) + sizeof(.heap) + sizeof(.stack);

//
// The code segment
//

    .picbase                0x00000000 :
    .preamble                          :
    .text                              :
    .syscall                           :
    .fixaddr                           :
    .fixtype                           :
    .rodata                            :
    .ROM.data               ROM(.data) :
    .secinfo                           :

    module_code_size =  endaddr(.secinfo);
}
```

为**使用 GHS 的 MCF544xx 生成模块**

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 GHS 的 MCF544xx 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   int     Errno;                                  \
                                VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                ULONG   tx_thread_module_mmu_protection;        \
                                VOID *  tx_thread_module_dispatch_return;       \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;
```

为使用 GHS 的 MCF544xx 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 GHS 的 MCF544xx 启用 API

MCF544xx 未启用此功能。

RX63 处理器

使用 IAR 的 RX63

使用 IAR 的 RX63 的模块报头

```
    /* Alignment of 4 (16-byte) */
    SECTION .text:CODE (4)

    /* Define public symbols.  */
    PUBLIC __txm_module_preamble

    /* Define application-specific start/stop entry points for the module.  */
    EXTERN _demo_module_start

    /* Define common external references.  */
    EXTERN  __txm_module_thread_shell_entry
    EXTERN  __txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                        // Module ID
    DC32      0x5                                               // Module Major Version
    DC32      0x6                                               // Module Minor Version
    DC32      32                                                // Module Preamble Size in 32-bit words
    DC32      0x12345678                                        // Module ID (application defined)
    DC32      0x00000007                                        // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected)
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
    DC32      __txm_module_thread_shell_entry - $               // Module Shell Entry Point
    DC32      _demo_module_start - $                            // Module Start Thread Entry Point
    DC32      0                                                 // Module Stop Thread Entry Point
    DC32      1                                                 // Module Start/Stop Thread Priority
    DC32      1024                                              // Module Start/Stop Thread Stack Size
    DC32      __txm_module_callback_request_thread_entry - $    // Module Callback Thread Entry
    DC32      1                                                 // Module Callback Thread Priority
    DC32      1024                                              // Module Callback Thread Stack Size
    DC32      ROPI$$Length                                      // Module Code Size
    DC32      RWPI$$Length                                      // Module Data Size
    DC32      0                                                 // Reserved 0
    DC32      0                                                 // Reserved 1
    DC32      0                                                 // Reserved 2
    DC32      0                                                 // Reserved 3
    DC32      0                                                 // Reserved 4
    DC32      0                                                 // Reserved 5
    DC32      0                                                 // Reserved 6
    DC32      0                                                 // Reserved 7
    DC32      0                                                 // Reserved 8
    DC32      0                                                 // Reserved 9
    DC32      0                                                 // Reserved 10
    DC32      0                                                 // Reserved 11
    DC32      0                                                 // Reserved 12
    DC32      0                                                 // Reserved 13
    DC32      0                                                 // Reserved 14
    DC32      0                                                 // Reserved 15

    END
```

使用 IAR 的 RX63 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护(必须选择用户模式)
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x02	■ ID IAR GNU

使用 IAR 的 RX63 的模块链接器

```
//-------------------------------------------------------------------------
// ILINK command file template for the Renesas RX microcontroller R5F563NB
//-------------------------------------------------------------------------
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

// ID Code Protection
define exported symbol __ID_BYTES_1_4   = 0xFFFFFFFF;
define exported symbol __ID_BYTES_5_8   = 0xFFFFFFFF;
define exported symbol __ID_BYTES_9_12  = 0xFFFFFFFF;
define exported symbol __ID_BYTES_13_16 = 0xFFFFFFFF;

// Endian Select Register (MDE)
// Choose between Little endian=0xFFFFFFFF or Big endian=0xFFFFFFF8
define exported symbol __MDES           = 0xFFFFFFFF;

// Option Function Select Register 0 (OFS0)
define exported symbol __OFS0           = 0xFFFFFFFF;

// Option Function Select Register 1 (OFS1)
define exported symbol __OFS1           = 0xFFFFFFFF;

//define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region RAM_region16 = mem:[from 0x00010000 to 0x00017FFF];
//define region ROM_region24 = mem:[from 0xFFF00000 to 0xFFFFFFFF];
//define region RAM_region24 = mem:[from 0x00000004 to 0x0001FFFF];
define region ROM_region32 = mem:[from 0xFFF10400 to 0xFFFFFFFF];
//define region RAM_region32 = mem:[from 0x00000004 to 0x0001FFFF];
//define region DATA_FLASH_region = mem:[from 0x00100000 to 0x00107FFF];

initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
do not initialize  { section .*.noinit };

define block HEAP     with alignment = 4, size = _HEAP_SIZE { };
//define block USTACK   with alignment = 4, size = _USTACK_SIZE { };
//define block ISTACK   with alignment = 4, size = _ISTACK_SIZE { };

//define block STACKS with fixed order { block ISTACK,
//                                       block USTACK };

//place at address mem:0xFFFFFF80 { ro section .nmivec };
//"ROM16":place in ROM_region16        { ro section .code16*,
//                                       ro section .data16* };
//"RAM16":place in RAM_region16        { rw section .data16*,
```

```
//                                      rw section __DLIB_PERTHREAD };
//"ROM24":place in ROM_region24        { ro section .code24*,
//                                      ro section .data24* };
//"RAM24":place in RAM_region24        { rw section .data24* };
//"ROM32":place in ROM_region32        { ro };
//"RAM32":place in RAM_region32        { rw,
//                                      ro section D,
//                                      ro section D_1,
//                                      ro section D_2,
//                                      block HEAP,
//                                      block STACKS,
//                                      last section FREEMEM };

//"DATAFLASH":place in DATA_FLASH_region
//                                    { ro section .dataflash* };

define movable block ROPI with alignment = 4, fixed order, static base CB
{
  ro object txm_module_preamble_rx63n.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base SB
{
  rw section .sbdata*,
  rw section .sbrel*,
  rw section __DLIB_PERTHREAD,
  block HEAP
};

place in ROM_region32   { block ROPI };
place in RAM_region16   { block RWPI };
```

为使用 IAR 的 RX63 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 RXRX635N 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;       \
                                VOID    *tx_thread_module_entry_info_ptr;     \
                                ULONG   tx_thread_module_current_user_mode;   \
                                ULONG   tx_thread_module_user_mode;           \
                                VOID    *tx_thread_module_kernel_stack_start; \
                                VOID    *tx_thread_module_kernel_stack_end;   \
                                ULONG   tx_thread_module_kernel_stack_size;   \
                                VOID    *tx_thread_module_stack_ptr;          \
                                VOID    *tx_thread_module_stack_start;        \
                                VOID    *tx_thread_module_stack_end;          \
                                ULONG   tx_thread_module_stack_size;          \
                                VOID    *tx_thread_module_reserved;           \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 RX63 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 RX63 启用 API

tttt	tt
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_EXECUTE	执行代码
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE	写入权限

????	??
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_READ	读取权限

RX65N 处理器

使用 IAR 的 RX65N

使用 IAR 的 RX65N 的模块报头

```
    /* Alignment of 4 (16-byte) */
    SECTION .text:CODE (4)

    /* Define public symbols.  */
    PUBLIC __txm_module_preamble

    /* Define application-specific start/stop entry points for the module.  */
    EXTERN _demo_module_start

    /* Define common external references.  */
    EXTERN  __txm_module_thread_shell_entry
    EXTERN  __txm_module_callback_request_thread_entry
    EXTERN  ROPI$$Length
    EXTERN  RWPI$$Length

    DATA
__txm_module_preamble:
    DC32      0x4D4F4455                                        // Module ID
    DC32      0x5                                               // Module Major Version
    DC32      0x6                                               // Module Minor Version
    DC32      32                                                // Module Preamble Size in 32-bit words
    DC32      0x12345678                                        // Module ID (application defined)
    DC32      0x00000007                                        // Module Properties where:
                                                                //   Bits 31-24: Compiler ID
                                                                //           0 -> IAR
                                                                //           1 -> ARM
                                                                //           2 -> GNU
                                                                //   Bit 0:  0 -> Privileged mode execution
                                                                //           1 -> User mode execution
                                                                //   Bit 1:  0 -> No MPU protection
                                                                //           1 -> MPU protection (must have
user mode selected)
                                                                //   Bit 2:  0 -> Disable shared/external
memory access
                                                                //           1 -> Enable shared/external
memory access
    DC32      __txm_module_thread_shell_entry - $               // Module Shell Entry Point
    DC32      _demo_module_start - $                            // Module Start Thread Entry Point
    DC32      0                                                 // Module Stop Thread Entry Point
    DC32      1                                                 // Module Start/Stop Thread Priority
    DC32      1024                                              // Module Start/Stop Thread Stack Size
    DC32      __txm_module_callback_request_thread_entry - $    // Module Callback Thread Entry
    DC32      1                                                 // Module Callback Thread Priority
    DC32      1024                                              // Module Callback Thread Stack Size
    DC32      ROPI$$Length                                      // Module Code Size
    DC32      RWPI$$Length                                      // Module Data Size
    DC32      0                                                 // Reserved 0
    DC32      0                                                 // Reserved 1
    DC32      0                                                 // Reserved 2
    DC32      0                                                 // Reserved 3
    DC32      0                                                 // Reserved 4
    DC32      0                                                 // Reserved 5
    DC32      0                                                 // Reserved 6
    DC32      0                                                 // Reserved 7
    DC32      0                                                 // Reserved 8
    DC32      0                                                 // Reserved 9
    DC32      0                                                 // Reserved 10
    DC32      0                                                 // Reserved 11
    DC32      0                                                 // Reserved 12
    DC32      0                                                 // Reserved 13
    DC32      0                                                 // Reserved 14
    DC32      0                                                 // Reserved 15

    END
```

使用 IAR 的 RX65N 的模块属性

BIT	“[”	[[
0	0 1	特权模式执行 用户模式执行
1	0 1	无 MPU 保护 MPU 保护（必须选择用户模式）
2	0 1	禁用共享/外部内存访问 启用共享/外部内存访问
[23-3]	0	保留
[31-24]	 0x00 0x02	■ ID IAR GNU

使用 IAR 的 RX65N 的模块链接器

```
//-------------------------------------------------------------------------
// ILINK command file template for the Renesas RX microcontroller R5F565N
//-------------------------------------------------------------------------
define exported symbol __link_file_version_4 = 1;

define memory mem with size = 4G;

// ID Code Protection
define exported symbol __ID_BYTES_1_4   = 0xFFFFFFFF;
define exported symbol __ID_BYTES_5_8   = 0xFFFFFFFF;
define exported symbol __ID_BYTES_9_12  = 0xFFFFFFFF;
define exported symbol __ID_BYTES_13_16 = 0xFFFFFFFF;

// Endian Select Register (MDE)
// Choose between Little endian=0xFFFFFFFF or Big endian=0xFFFFFFF8
define exported symbol __MDES           = 0xFFFFFFFF;

// Option Function Select Register 0 (OFS0)
define exported symbol __OFS0           = 0xFFFFFFFF;

// Option Function Select Register 1 (OFS1)
define exported symbol __OFS1           = 0xFFFFFFFF;

//define region ROM_region16 = mem:[from 0xFFFF8000 to 0xFFFFFFFF];
define region RAM_region16 = mem:[from 0x00010000 to 0x00017FFF];
//define region ROM_region24 = mem:[from 0xFFF00000 to 0xFFFFFFFF];
//define region RAM_region24 = mem:[from 0x00000004 to 0x0001FFFF];
define region ROM_region32 = mem:[from 0xFFF10400 to 0xFFFFFFFF];
//define region RAM_region32 = mem:[from 0x00000004 to 0x0001FFFF];
//define region DATA_FLASH_region = mem:[from 0x00100000 to 0x00107FFF];

initialize by copy { rw, ro section D, ro section D_1, ro section D_2 };
do not initialize  { section .*.noinit };

define block HEAP     with alignment = 4, size = _HEAP_SIZE { };
//define block USTACK   with alignment = 4, size = _USTACK_SIZE { };
//define block ISTACK   with alignment = 4, size = _ISTACK_SIZE { };

//define block STACKS with fixed order { block ISTACK,
//                                       block USTACK };

//place at address mem:0xFFFFFF80 { ro section .nmivec };
//"ROM16":place in ROM_region16        { ro section .code16*,
//                                       ro section .data16* };
//"RAM16":place in RAM_region16        { rw section .data16*,
```

```
//                                      rw section __DLIB_PERTHREAD };
//"ROM24":place in ROM_region24        { ro section .code24*,
//                                      ro section .data24* };
//"RAM24":place in RAM_region24        { rw section .data24* };
//"ROM32":place in ROM_region32        { ro };
//"RAM32":place in RAM_region32        { rw,
//                                      ro section D,
//                                      ro section D_1,
//                                      ro section D_2,
//                                      block HEAP,
//                                      block STACKS,
//                                      last section FREEMEM };

//"DATAFLASH":place in DATA_FLASH_region
//                                    { ro section .dataflash* };

define movable block ROPI with alignment = 4, fixed order, static base CB
{
  ro object txm_module_preamble_rx65n.o,
  ro,
  ro data
};

define movable block RWPI with alignment = 8, fixed order, static base SB
{
  rw section .sbdata*,
  rw section .sbrel*,
  rw section __DLIB_PERTHREAD,
  block HEAP
};

place in ROM_region32   { block ROPI };
place in RAM_region16   { block RWPI };
```

为使用 IAR 的 RX65N 生成模块

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

使用 IAR 的 RX65N 的线程扩展定义

```
#define TX_THREAD_EXTENSION_2   VOID    *tx_thread_module_instance_ptr;         \
                                VOID    *tx_thread_module_entry_info_ptr;       \
                                ULONG   tx_thread_module_current_user_mode;     \
                                ULONG   tx_thread_module_user_mode;             \
                                VOID    *tx_thread_module_kernel_stack_start;   \
                                VOID    *tx_thread_module_kernel_stack_end;     \
                                ULONG   tx_thread_module_kernel_stack_size;     \
                                VOID    *tx_thread_module_stack_ptr;            \
                                VOID    *tx_thread_module_stack_start;          \
                                VOID    *tx_thread_module_stack_end;            \
                                ULONG   tx_thread_module_stack_size;            \
                                VOID    *tx_thread_module_reserved;             \
                                VOID    *tx_thread_iar_tls_pointer;
```

为使用 IAR 的 RX65N 生成模块管理器

提供了示例工作区。生成 ThreadX 库、ThreadX 模块库、示例模块项目和示例模块管理器项目。

外部内存属性为使用 IAR 的 RX65N 启用 API

tttt	tt
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_EXECUTE	执行代码
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_WRITE	写入权限

☐☐☐☐	☐☐
TXM_MODULE_MANAGER_SHARED_ATTRIBUTE_READ	读取权限