不错的参考文档

<https://wenku.baidu.com/view/0f1a9679c281e53a5802ffb5.html?sxts=1522217040574>

<https://wenku.baidu.com/view/57c833bf0722192e4436f67d.html>

1.文件结构：

Tx\_api.h C头文件，包含所有的系统equates, 数据结构，服务原型。

Tx\_port.h C头文件，开发工具的所有特殊的数据定义及结构。

Demo.c C文件，一个小的例程。

Build\_ap.bat MS-DOS批处理文件，讲述如何建立ThreadX实例。

Build\_tx.bat MS-DOS批处理文件，讲述如何建立ThreadX C库，包括在完全模式中。

Tx.lib 二进制的ThreadX C库，包括在标准模式中。

一个简单的例子：

#include "tx\_api.h"

unsigned long my\_thread\_counter = 0;

TX\_THREAD my\_thread;

main()

{

/\* Enter the ThreadX kernel. \*/

tx\_kernel\_enter();

}

void tx\_application\_define(void \*first\_unused\_memory)

{

/\* Create my\_thread! \*/

tx\_thread\_create(&my\_thread, "My Thread",

my\_thread\_entry, 0x1234, first\_unused\_memory, 1024,

3, 3, TX\_NO\_TIME\_SLICE, TX\_AUTO\_START);

}

void my\_thread\_entry(ULONG thread\_input)

{

/\* Enter into a forever loop. \*/

while(1)

{

my\_thread\_counter++;

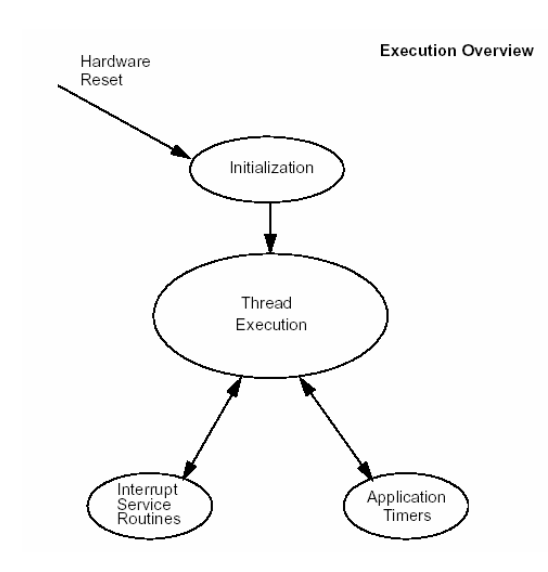
tx\_thread\_sleep(1);

}

}

1. 执行概述

2.1执行概述



* 初始化
* 线程执行
* 中断服务例程
* 程序定时器

2.2内存的使用

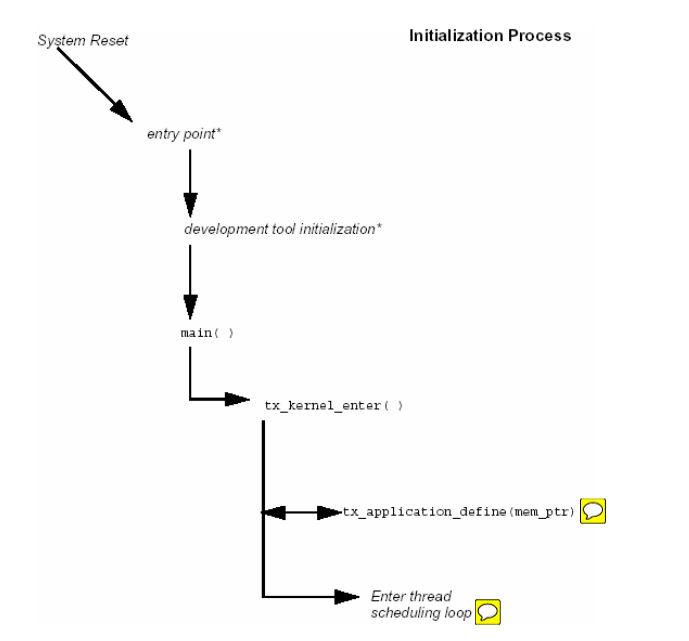
大多数的开发工具将程序映像分为五个基本区：

* 指令区
* 常量区
* 已初始化数据区
* 未初始化数据区机
* 系统堆栈区

2.3初始化

* 系统启动：所有的微处理器都有启动逻辑。当启动时（无论是硬件的还是软件的），特定的内存地址将重新分配应用程序的入口地址。程序的入口地址常用汇编语言写成
* 开发工具的初始化：严格的初始化过程是开发工具所特有的。初始化的全局和静态的C变量一般在这里建立
* Main主函数：对大多数的程序来说，主函数仅仅调用进入ThreadX的tx\_kernel\_enter。但是，应用程序能提前执行初步处理（常针对硬件初始化）来进入ThreadX中。
* （注意，tx\_kernel\_enter无返回，所以不能放置任何的操作于其后。）
* tx\_kernel\_enter ：此入口函数配合ThreadX内部不同数据结构的初始化，然后调用程序定义函数tx\_application\_define.当tx\_application\_define返回时，控制转移到线程调度循环，这标志着初始化的完成。
* 程序定义函数：tx\_application\_define函数定义了所有初始的应用线程、队列、消息、事件标志、存储池和定时器。它也可以在正常的应用程序执行过程中创建或者删除系统资源。但是，所有的初始应用资源在此处得到定义。值得一题的是tx\_application\_define函数仅有一个输入参数，就是RAM中的第一个可用的地址。一般它被用作线程堆栈、队列、内存池的初始运行内存的开始点。注意，初始化完成之后，只有一个执行线程可以建立和删除系统资源，包括其它的线程。因此，在初始化阶段至少要建立一个线程。
* 中断：在整个初始化的过程中中断是被禁止的。

初始化的总体结果：



2.4线程执行：

2.4.1线程执行状态

tx\_thread\_create来建立。在ThreadX中线程有五个状态，即就绪、挂起、执行、停止和完成。

有一点必须指出，在任意一个特定的时刻只有一个线程处于执行状态

线程处于停止状态是由于另一个线程或它自己调用了tx\_thread\_terminate服务。

（注意，如果需要重新开始一个以执行的或停止了的线程，程序首先需要删除该线程。然后才能重新建立并重新开始。）

线程的优先权 0-31

0优先级最高。

2.4.2线程调度：

* 就序状态的优先权最高的线程首先得到执行。如果相同优先权的多线程同时处于就序状态，则将以先进先出（FIFO）的方式执行
* 循环（Round-Robin）调度：tx\_thread\_relinquish。调用此服务给其它所有拥有相同优先权的就绪线程一个执行的机会
* 时间片：时间片是指线程完全占有处理器的最大的定时器计数的个数，可以修改。
* 抢占：抢占是指更高优先的线程临时中断正在运行线程的过程
* 抢占入口：抢占入口是一个什么样的概念呢？抢占入口允许线程定义一个禁止抢占的优先权限度。优先权高于这个限度的仍然允许抢占，优先权低的则不允许。
* 线程控制块：TX\_THREAD。tx\_api.h。调试的时候很好用。最好的方法是在创
* 调试线程：建线程之前用一个容易识别的数据格式如0xEFEF预设所有的堆栈区。在应用程序完全就绪之后，开始通过检查预设数据格式是否改变来判断堆栈区实际上利用了多少。调试者必须观察当前线程指针\_tx\_thread\_current\_ptr,看调用线程是否是正在调试的。

2.4.3 消息队列：

* 队列控制块Block TX\_QUEUE ：每一个消息队列的特征可以在其控制块中找到。控制块中包含着队列中的消息数目等信息。这种结构在tx\_api.h文件中被定义。确保接受消息接收站有足够大的空间以便容纳整个消息是很重要的。

2.4.4计数信号

* 信号控制块TX\_SEMAPHONE。用于控制线程的运行次序。其范围从0到4,294,967,295。

2.4.5 事件标志

* 事件标志组控制块TX\_EVENT\_FLAGS\_GROUP：事件标志可以由程序线程在初始化或者运行期间创建。在创建之初，事件标志组中的所有事件标志被设为0。程序中的事件标志组的数目没有限制。

2.4.6内存块池

* 内存块池控制块TX\_BLOCK\_POOL：在实时系统中以快速且确定的方式分配内存是一个挑战。基于此，ThreadX提供一种方法：创建内存块大小固定的多个池，并对其实行管理。

2.4.7 内存字节池

* 内存字节池控制块TX\_BYTE\_POOL：内存字节池的分配类似于传统的malloc调用，其中包括所需的内存大小（以字节计）。
* 严重问题：内存字节池中可分配的字节数比初始化所设定的要稍微少一点，因为对空闲内存区的管理导致了部分开支。每个空闲内存块需要两个C指针所占的空间。例如：一个有2000个可用字节的内存字节池可能满足不了1000个字节的分配需求。这是因为不能保证有多少个连续的空闲字节。即使存在1000个字节的空闲块，仍不能保证得多久才能发现这些空闲块。

2.4.8 程序定时器

ThreadX程序定时器规定程序能够在特定的时间间隔执行C函数。程序定时器也可以只有一次截止(expire).这种类型的定时器称为单步定时器（one-shot timer）,而反复间隔的定时器称为周期定时器（periodic timer）。

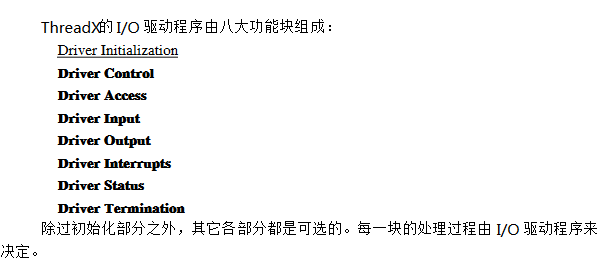
* 定时器间隔：每个定时器中断称为一个timer tick.介于两个timer tick之间的事件由程序来规定。但在多数应用中以10ms为标准。值得一提的是，隐藏的硬件要能够产生周期性的中断，这样程序定时器才能够工作.
* 定时器的精确度:增加定时器中断频率可以减小错误范围
* 定时器的执行:程序定时器按照被激活的顺序依次执行
* 程序定时器控制块:控制块中包含着32位的截止确认等信息。这种结构在tx\_api.h文件中被定义。

2.4.9中断

* tx\_interrupt\_control命令允许程序对中断进行使能或者禁止操作。先前的中断使能或禁止状态由此命令来返回。必须指出的是中断控制只对当前正在执行的程序段产生作用。

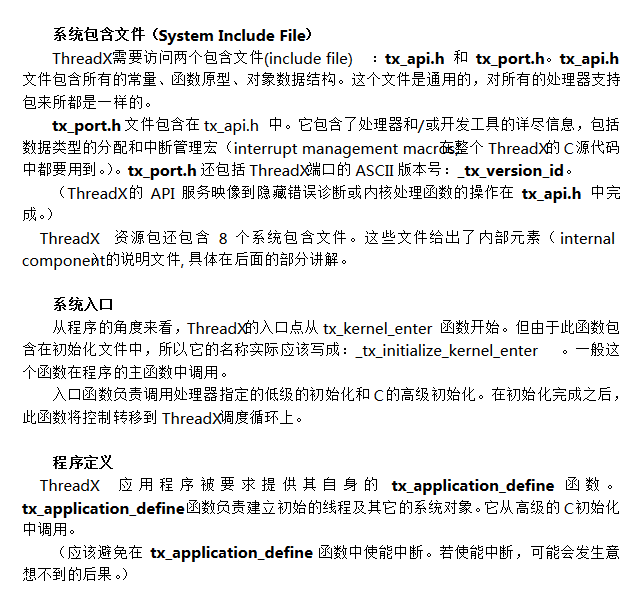
2.5 IO驱动

与外界进行信息交流是大多数嵌入式系统的重要特性。此种交流通过硬件设备和嵌入式应用软件的互相结合来实现。用来管理这些硬件设备的软件称之为I/O驱动程序。不可能靠提供一组驱动程序来达到满足每个应用程序要求的目的



* 驱动初始化:这一个功能块负责硬件设备的初始化以及驱动程序的内部数据结构的初始化.tx\_application\_define函数或调用线程来完成。
* 驱动程序的控制：驱动程序控制块负责运行的控制
* 驱动程序的存取：在程序中如果多个线程需要同时访问驱动程序，它们之间的作用必须通过在I/O驱动程序中增加分配/释放工具来控制，换句话说，程序可以用一个信号来控制驱动程序的存取，以此来避免额外的开销和驱动程序内的复杂化。
* 驱动程序的输入：此功能块负责所有的设备输入
* 驱动程序的输出：此功能块负责所有的设备输出
* 驱动程序的中断：大多数实时系统依靠硬件中断通知安装设备驱动程序输入、输出、控制及错误等事件。
* 驱动程序的状态：大多数实时系统依靠硬件中断通知安装设备驱动程序输入、输出、控制及错误等事件。Current device status.Input bytes.Output bytes.I/O reeor counts.
* 驱动程序的终止：此功能块为任意可选项。只有当驱动程序与/或硬件设备需要关闭时才要用到此项.

2.6 ThreadX的内部结构



2.6.1 Threadx元素

ThreadX 元素有8个。每个元素都有相同的基本结构。其处理方法和数据结构很容易从其他的元素中区分出来。Threadx的软件元素：

* Initialize
* Thread
* Timer
* Queue
* Semaphore
* Event Flag
* Block Memory
* Byte Memory

ThreadX软件元素都有一个说明文件。说明文件为标准的C包含文件，其中包括了所有的常量、数据结构、外部和内部元素函数原型（external and internal component function prototypes）和元素的全局数据定义

2.6.2 元素体函数

不同数目的元素体或操作函数组成了ThreadX的软件元素。

作为一个基本原则，元素体函数是唯一允许访问元素全局数据的函数。和其它元素的相互作用必须使用在其它元素说明文件中定义的访问函数。

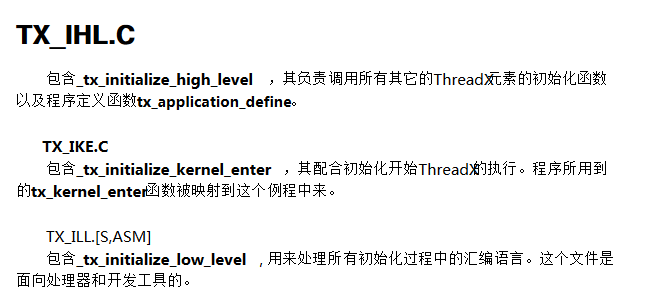
2.6.3 初始化元素

初始化元素负责所有ThreadX的初始化工作。这个过程包括建立处理器资源以及其它元素的初始化函数。一旦基本的ThreadX初始化完成之后，程序的tx\_application\_define函数被调用去执行指定的初始化。初始化完成之后线程调度循环开始。

2.6.3.1 TX\_INI.H

这是ThreadX初始化元素的说明文件。所有的元素常量、外部接口、数据结构都在这个文件中被定义。这是ThreadX初始化元素的说明文件。所有的元素常量、外部接口、数据结构都在这个文件中被定义。

\_tx\_initialize\_unused\_memory：此空指针包含ThreadX初始化之后程序的第一个可用的内存地址。此变量的内容传给tx\_application\_define函数。



2.6.3.2 线程元素

此元素负责所有的线程管理活动，包括线程的建立、调度、中断管理等。线程元素是所有的ThreadX元素中和处理器及编译器联系最为紧密的，因此，其大部分文件是汇编语言写的。

TX\_THR.H:这是ThreadX线程元素的说明文件。所有的元素常量、外部接口、数据结构都在这个文件中定义。

总要的参数：

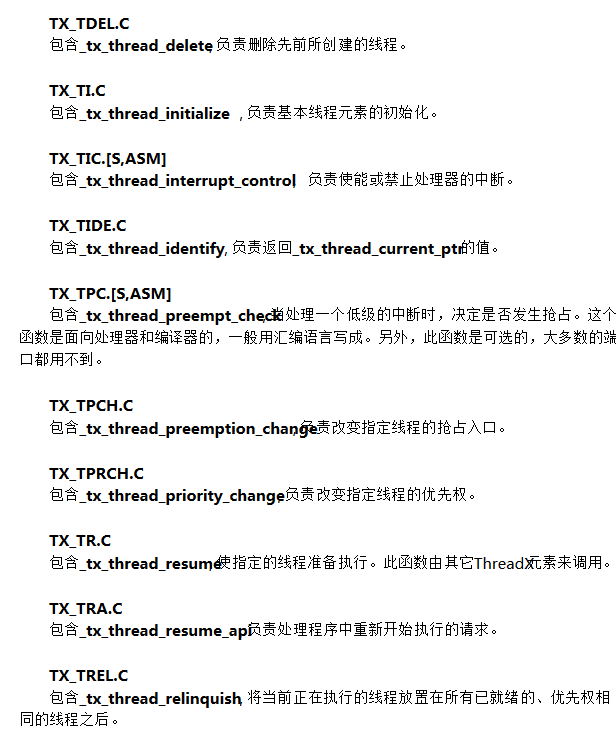
* \_tx\_thread\_system\_stack\_ptr:此空指针包含系统堆栈指针的地址。系统堆栈用于ThreadX调度循环中和中断处理中。
* \_tx\_thread\_current\_ptr:此TX\_THREAD指针包含当前运行的线程的控制块的地址。如果此指针为NULL,系统处于空闲状态。
* \_tx\_thread\_execute\_ptr:此TX\_THREAD指针包含下一个要执行的线程的地址。其由调度循环用来决定哪个线程在下一步将要执行。
* \_tx\_thread\_created\_ptr:此TX\_THREAD指针是已创建线程列表的头指针。此列表是一个双向循环的链表，其成员为所有已创建线程的控制块。
* \_tx\_thread\_created\_count:系统中当前已经创建的线程的数目
* \_tx\_thread\_system\_state:当前的系统状态
* \_tx\_thread\_preempted\_map:每一位代表32级优先权的每一级。当某一位置位时，表示相应优先权的线程被抢占了（当其有有效的抢占入口时）
* \_tx\_thread\_priority\_map:每一位代表32级优先权的每一级。其用于当一个高优先权的线程挂起的时候，找出下一个优先权较低的就绪的线程
* \_tx\_thread\_lowest\_bit:一张查询表，用来快速找出一个字节中最低位的设
* 置。其用于检查\_tx\_thread\_priority\_map，找出下一个要就绪的优先级组。
* \_tx\_thread\_priority\_list :列表数组直接以线程的优先权为下标。若有其中一项不为NULL，则该项所代表的优先权至少有一个线程将处于就绪状态。每个优先权列表中的线程以双向循环链表的形式来组织，其成员为线程控制块。
* \_tx\_thread\_preempt\_disable:一种使ThreadX服务进入内部临界区处理的机制,这减少了在ThreadX服务中禁止时间中断的需要.
* \_tx\_thread\_special\_string:最初开发ThreadX的相关人员和机构.

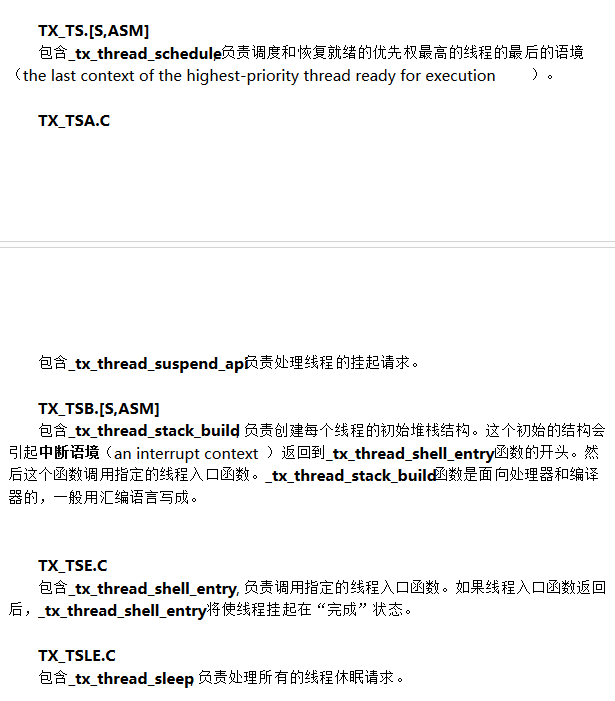
2.6.3.3 TX\_TC.C

此文件包含\_tx\_thread\_create函数，负责线程的创建

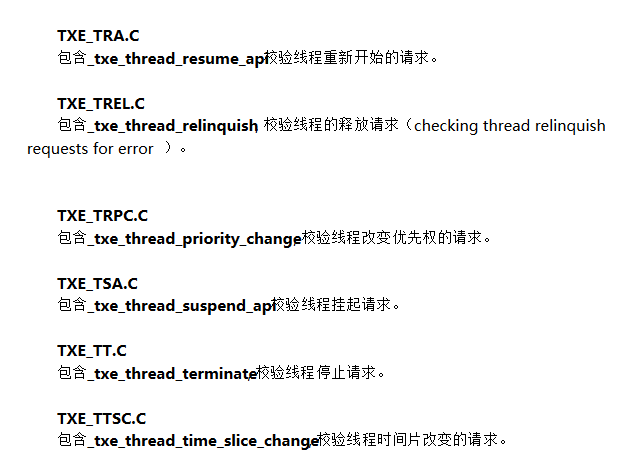
2.6.3.4 TX\_TCR.[S,ASM]

包含\_tx\_thread\_context\_restore函数,负责在可控制的ISR后部的处理。这个函数是面向处理器和编译器的，一般用汇编语言写成









还有一些相关功能的元素。