Nucleus PLUS源码分析

Nucleus PLUS Internals

翻译：樊荣
Translate by cini
2003-6

Accelerated Technology, Inc.
720 Oak Circle Dr. E.
Mobile, AL 36609
(334) 661-5770

序

相关文档

Nucleus PLUS 参考手册，Accelerated Technology编著,描述如何操作和使用Nucleus PLUS 内核。

样式和符号约定

Program listings, program examples, filenames, menu items/buttons and interactive displays are each shown in a special font.
Program listings and program examples - Courier New
Filenames - COURIER NEW, ALL CAPS
Interactive Command Lines - **Courier New, Bold**
Menu Items/Buttons - *Times New Roman Italic*

商标

MS-DOS 是 Microsoft商标
UNIX 是 X/Open商标
IBM PC 是 IBM商标

联系

请按如下联系
Accelerated Technology
720 Oak Circle Drive, East
Mobile, AL 36609
800-468-6853
334-661-5770
334-661-5788 (fax)
support@atinucleus.com
http://www.atinucleus.com

Document Part Number : 0001027-001
Last Revised: June 16, 1999

译者：樊荣
coosty@163.com
If you got any Problem, Suggestion, Advice or Question ,
Please mail to: coosty@163.com
Any correction will be appreciated.

内容

第一章. 介绍

手册的目的

Nucleus PLUS是以源码形式分发的。 由于Nucleus PLUS 源码是非常庞大，一般用户往往要有一段艰难的理解它的时间。该手册就是为帮助Nucleus PLUS 用户理解源码而设计的。

关于Nucleus PLUS

Nucleus PLUS 是一个实时的、抢占的、多任务的为嵌入式实时设计的内核。大约95%的Nucleus PLUS 代码是用ANSI C编写的。所以，Nucleus PLUS非常适合移植，它现在能用于绝大多数微处理器。Nucleus PLUS 通常是以C语言库实现的。实时Nucleus PLUS 应用是被和Nucleus PLUS 库链接在一起。这个生成的代码能被下载到目标板或者放置在ROM中。在通常的目标环境中，假设所有的功能都被打开，Nucleus PLUS 的二进制执行映像大约需要 20 Kbytes 的内存。

Nucleus PLUS 结构

Accelerated Technology’s 软件公司的开发惯例有助于使代码清晰、模块化、可靠、可复用、易于维护。Nucleus PLUS 包含许多软件组件。每一个软件组件有特定的目的和提供给其他组件的特定接口。每一个Nucleus PLUS 软件组件在后来的章节进行更多的描述。

第二章．实现约定

Components
Component Composition
Naming Conventions
Indentation
Comments

组件 Components

Accelerated Technology (ATI) 使用软件组件方法学。一个组件有单一的清晰的目的。软件组件往往包含几个C和/或汇编程序。每一个软件组件提供定义非常好的外部接口。通过使用外部接口来利用组件。除了很少的例外，在组件外通过组件访问全局变量是不允许的。因为使用组件方法学，Nucleus PLUS 软件组件是非常容易被替换和复用的。

组件包含 Component Composition

为了定义数据类型和常量，一个软件组件通常包含一个include文件； 为了定义组件的外部接口，包含一个include文件。软件组件还包含一个或更多C和/或汇编程序。包含的文件的文件名满足如下约定：

文件	意义
XX_DEFS.H	组件的常量和数据类型
XX_EXTR.H	组件的外部接口定义在这个文件中，外部接口以函数原形的方式定义
XXD.C	组件的静态的和全局的变量定义在这个文件中，除了很少的例外，一个组件的数据结构仅仅只能被本组件访问
XXI.C	组件初始化定义在这个文件中
XXF.C	这个文件提供被组件管理的对象的信息
XXC.C	这个文件提供组件的核心函数
XXCE.C	这个文件提供组件的核心函数的错误处理外壳函数
XXS.C	补充的组件函数定义在这个文件中
XXSE.C	这个文件提供组件的补充的函数的错误处理外壳函数

格式 Format
所有的软件源码文件有相同的基本格式。文件的第一部分包含该文件的通用的信息，这一部分称为序。文

件的第二部分主要描述内部的数据结构和内部的函数原形。文件的剩下部分包括实际的函数。

序 Prologue

序的目的是描述文件的内容，标识ATI 为文件的所以者，提供关于文件的版本信息。下面是一个文件虚的实例：

```
/************************************************************************/
/* */
/*Copyright (c) 199x by Accelerated Technology, Inc. */
/* */
/* the subject matter of this material. All manufacturing, */
/* reproduction, use, and sales rights pertaining to this subject */
/* matter are governed by the license agreement. The recipient of */
/* this software implicitly accepts the terms of the license. */
/* */
/* */
/************************************************************************/
/************************************************************************/
/* FILE NAME VERSION */
/* */
/* [name of this file] n.n */
/* */
/* COMPONENT */
/* */
/* [identifies the component] */
/* */
/* DESCRIPTION */
/* */
/* [general description of this file] */
/* */
/* AUTHOR */
/* */
/* [author’ s name] */
/* */
/* DATA STRUCTURES */
/* */
/* [global component data structures defined in this file]*/
/* */
/* FUNCTIONS */
/* */
/* [functions defined in this file] */
/* */
/* DEPENDENCIES */
/* [other file dependencies] */
/* */
/* HISTORY */
/* */
/*NAME DATE REMARKS */
/* [information about revising and verifying changes to this file] */
/************************************************************************/
```

序之后 After the Prologue

序之后的内容是常量、全局数据结构和组件内部函数原形。当然也包括定义组件数据结构的或外部接口的include文件。

文件的剩余部分 Remainder of File
软件组件的剩余部分包括C或汇编语言的函数。在每一个函数之前有一个描述块。该函数描述块的格式如下：

```
/*************************************************************************/
/* FUNCTION */
/* */
/* [name of the function] */
/* */
/* DESCRIPTION */
/* */
/* [general description of function] */
/* */
/* AUTHOR */
/* */
/* [author's name] */
/* */
/* CALLED BY */
/* [functions that call this function] */
/* CALLS */
/* */
/* [函数调用 Functions Called by this function] */
/* */
/* INPUTS */
/* */
/* [inputs to the function] */
/* */
/* OUTPUTS */
/* */
/* [outputs of this function] */
/* */
/* HISTORY */
/* */
/* NAME DATE REMARKS */
/* */
/* [information about revising and verifying changes to this function] */
/* */
/*************************************************************************/
```

命名规则 Naming Conventions

命名规则有意设计成通过合并三个或四个文件的字符为全局变量和函数的名字来使检查ATI源码更容易些。当然，所以的命名符合它们自己的用途。详细的关于命名规则是在下面的子章节中进行描述。

组件名字 Component Names
组件名字通常限制在两个字符。通过在每一个文件名前两个字符为组件名字来表明该文件属于那一个组件。
例子：
动态内存管理组件（Dynamic Memory Management Component）名字： DM
DM包含的文件：
DM_DEFS.H
DM_EXTR.H
DMC.C
DMCE.C

DMI.C
DMF.C
DMD.C

宏定义名字 #define Names
宏定义可包含下划线、大写字母和数字。最大的宏定义长度为31个字符。另外，如宏定义“CC_”,这儿“CC”是该宏定义所在文件的文件名的头两个字母。
例子 (for file EX_DEFS.H) ：

```
#define EX_MY_CONSTANT 10
```

结构名字 Structure Names
结构名字可包含下划线、大写字母和数字。最大的结构名字长度为31个字符。另外，如结构名字“CC_”,这儿“CC”是该结构名字所在文件的文件名的头两个字母。
例子 (for file EX_DEFS.H):

```
struct EX_MY_STRUCT
{
int ex_member_a;
int ex_member_b;
int ex_member_c;
};
```

类型名字 Typedef Names
类型名字包含下划线、大写字母和数字。最大的类型名字长度为31个字符。另外，如类型名字“CC_”,这儿“CC”是该类型名字所在文件的文件名的头两个字母。
例子 (for file EX_DEFS.H):

```
typedef struct EX_MY_STRUCT
{
int ex_member_a;
int ex_member_b;
int ex_member_c;
} EX_MY_TYPEDEF;
```

结构成员名字 Structure Member Names
结构成员名字包括下划线、小写字母和数字。结构成员名字长度不超过31个字符。另外，如结构成员名字“cc_”,这儿“cc”是结构成员名字所在文件的文件名“CC_DEFS.H”的头两个字母。
例子 (for file EX_DEFS.H):

```
struct EX_MY_STRUCT
{
int ex_member_a;
int ex_member_b;
int ex_member_c;
};
```

全局变量名字Global Variable Names
Nucleus Plus全局变量名字包括小写字母和可能有的下划线和数字。全局变量名字长度不超过31个字符。另外，如全局变量名字“CCC”,这儿“CCC”是全局变量名字所在文件的文件名“CCC.c”的头三个字母。
例子 (for file EXD.C):

```
int EXD_Global_Integer;
```

局部变量名字 Local Variable Names
局部变量名字(在C函数内部定义的变量名字)包括小写字母和可能有的下划线和数字。局部变量名字长度不超过31个字符。另外，局部变量名的头三个字母不必和包含函数的文件的文件名头三个字母一样。
例子 (for file EXD.C):

```
/* Assume the following declaration is inside a function. */
```

```
int i;
```

函数名字 Function Names

Nucleus Plus 函数名字包括下划线、下划线后紧跟一个大写字母、一些小写字母和数子。函数名字长度不超过31个字符。另外，函数名的头三个字母和包含函数的文件的文件名头三个字母一样。

例子 (for file EXD.C):

```
void EXD_My_Function(unsigned int i)
{
.
.
.
}
```

缩排 Indentation

基本的缩排单位是四个空格，函数定义、变量定义和条件语句结构在第一列开始。实际的命令语句在第四列开始。注意：‘{’ 和 ‘}’各占一行。‘{’和上一行进行相同的缩排，而‘}’和前一个‘{’ 进行相同的缩排。

例子 (for file EXD.C):

```
void EXD_Example_Function(int i, int b)
{
unsigned int a;
char b;
/* Actual instructions start. */
i = 0;
while (i < 100)
{
/* Increment i. */
i = i + 1;
}
}
```

注释 Comments

注释是Nucleus PLUS 重要的部分。

例子:

```
/* This is the first type of meaningful comment. */
i = 10;
j++; /* This is the second type of comment. */
```

第三章. 软件概述

Basic Usage
Data Types
Service Call Mapping
Environment Dependencies
Version Control

基本用法 Basic Usage

Nucleus PLUS 通常是以C语言库实现的。实时Nucleus PLUS 应用是被和Nucleus PLUS 库链接在一起。这个生成的代码能被下载到目标板或者放置在ROM中。

NUCLEUS.LIB 是典型的Nucleus PLUS 库的文件名。它是用BUILD_LI.BAT批处理文件生成的。BUILD_LI.BAT 的内容依赖于所使用的开发工具。

- **运行模式 Operation Mode**

在处理器结构中有管理模式和用户模式之分，Nucleus PLUS 应用任务通常是在管理模式下运行。这是因为应用任务需要直接地调用需调用特权指令的系统服务程序。这样做减少了系统服务调用的开销而且也便于实现。当然这样做也就**允许任务访问任意资源**。

- **应用程序初始化 Application Initialization**

用户有责任提供他们自己的初始化函数，该函数称为 Application_Initialize。这个函数应创建任务、队列、和当系统开始时的其他系统对象。如果应用程序不在系统运行时动态创建和删除系统对象，那么所以的系统对象都应该在Application_Initialize被创建。在用户的Application_Initialize函数执行完毕返回后，多任务立刻开始执行。在一些目标环境下，底层的系统初始化文件，INT.S、INT.ASM、或者INT.SRC 可能需要修改。这些文件初始化系统定时中断、可用储存空间和其他处理器或者目标板的特定实体。

- **包含文件 Include File**

所以的用户调用的Nucleus PLUS 服务和/或数据类型必须被包含在**NUCLEUS.H**文件中。这个文件包含数据类型定义、常量定义和Nucleus PLUS 服务函数原形。不同的Nucleus PLUS 移植系统，这个文件是不同的。

数据类型 Data Types

Nucleus PLUS 在NUCLEUS.H定义了几个标准的数据类型。这些数据类型保证剩余的数据类型能被编译为适当的目标C编译器数据类型。所以，Nucleus PLUS 能在大量的目标环境下以相同的风格执行。

下面的数据类型是被Nucleus PLUS定义的：

数据类型	意义
UNSIGNED	它需要是一个32-位的无符号的整数，它通常被定义为一个无符号长整数（unsigned long）的C数据类型
SIGNED	它需要是一个32-位的有符号的整数，它通常被定义为一个有符号长整数（long）的C数据类型.
OPTION	最小的容易使用的数据类型，它通常被定义为一个无符号字符（unsigned char）的C数据类型.
DATA_ELEMENT	和OPTION相同的数据类型
UNSIGNED_CHAR	它需要是一个8-位的无符号的字符（unsigned char）
CHAR	它需要是一个8-位的的字符（char）
STATUS	和目标C编译器的有符号整数（int）等价
INT	整数数据类型，它具有word的大小
VOID	和目标C编译器的void（void）等价
UNSIGNED_PTR	该数据类型是一个指向UNSIGNED的指针
BYTE_PTR	该数据类型是一个指向UNSIGNED_PTR的指针

系统服务映射关系 Service Call Mapping

Nucleus PLUS主包含文件NUCLEUS.H,包含和那些定义在《*Nucleus PLUS Reference Manual*》函数相匹配的函数原形。然而，这些NU_* 的函数实际并不真实存在。对于绝大多数的Nucleus PLUS 函数，有一个真正执行操作的函数，也有一个在调用真正执行操作的函数前对用户调用进行错误检查的外壳函数。如果错误检查条件定义宏NU_NO_ERROR_CHECKING不被定义,通过宏替换， Nucleus PLUS 系统服务是被映射成下面的函数。这使得在不需要时可以很方便地去掉错误检查。

● 错误检查 Error Checking

如果NU_NO_ERROR_CHECKING 标识不被定义（缺省情况),定义在《*Nucleus PLUS Reference Manual* 》的系统服务函数NU_*被映射成下面的函数：

Nucleus PLUS Service Internal Function	Internal Function
NU_Activate_HISR	TCCE_Activate_HISR
NU_Allocate_Memory	DMCE_Allocate_Memory
NU_Allocate_Partition	PMCE_Allocate_Partition
NU_Broadcast_To_Mailbox	MBSE_Broadcast_To_Mailbox
NU_Broadcast_To_Pipe	PISE_Broadcast_To_Pipe
NU_Broadcast_To_Queue	QUSE_Broadcast_To_Queue
NU_Change_Preemption	TCSE_Change_Preemption
NU_Change_Priority	TCSE_Change_Priority
NU_Change_Time_Slice	TCSE_Change_Time_Slice
NU_Check_Stack	TCT_Check_Stack
NU_Control_Interrupts	TCT_Control_Interrupts
NU_Control_Signals	TCSE_Control_Signals

NU_Control_Timer	TMSE_Control_Timer
NU_Create_Driver	IOCE_Create_Driver
NU_Create_Event_Group	EVCE_Create_Event_Group
NU_Create_HISR	TCCE_Create_HISR
NU_Create_Mailbox	MBCE_Create_Mailbox
NU_Create_Memory_Pool	DMCE_Create_Memory_Pool
NU_Create_Partition_Pool	PMCE_Create_Partition_Pool
NU_Create_Pipe	PICE_Create_Pipe
NU_Create_Queue	QUCE_Create_Queue
NU_Create_Semaphore	SMCE_Create_Semaphore
NU_Create_Task	TCCE_Create_Task
NU_Create_Timer	TMSE_Create_Timer
NU_Current_HISR_Pointer	TCF_Current_HISR_Pointer
NU_Current_Task_Pointer	TCC_Current_Task_Pointer
NU_Deallocate_Memory	DMCE_Deallocate_Memory
NU_Deallocate_Partition	PMCE_Deallocate_Partition
NU_Delete_Driver	IOCE_Delete_Driver
NU_Delete_Event_Group	EVCE_Delete_Event_Group
NU_Delete_HISR	TCCE_Delete_HISR
NU_Delete_Mailbox	MBCE_Delete_Mailbox
NU_Delete_Memory_Pool	DMCE_Delete_Memory_Pool
NU_Delete_Partition_Pool	PMCE_Delete_Partition_Pool
NU_Delete_Pipe	PICE_Delete_Pipe
NU_Delete_Queue	QUCE_Delete_Queue
NU_Delete_Semaphore	SMCE_Delete_Semaphore
NU_Delete_Task	TCCE_Delete_Task
NU_Delete_Timer	TMSE_Delete_Timer
NU_Disable_History_Saving	HIC_Disable_History_Saving
NU_Driver_Pointers	IOF_Driver_Pointers
NU_Enable_History_Saving	HIC_Enable_History_Saving
NU_Established_Drivers	IOF_Established_Drivers
NU_Established_Event_Groups	EVF_Established_Event_Groups
NU_Established_HISRs	TCF_Established_HISRs
NU_Established_Mailboxes	MBF_Established_Mailboxes
NU_Established_Memory_Pools	DMF_Established_Memory_Pools
NU_Established_Partition_Pools	PMF_Established_Partition_Pools
NU_Established_Pipes	PIF_Established_Pipes
NU_Established_Queues	QUF_Established_Queues
NU_Established_Semaphores	SMF_Established_Semaphores
NU_Established_Tasks	TCF_Established_Tasks
NU_Established_Timers	TMF_Established_Timers
NU_Event_Group_Information	EVF_Event_Group_Information
NU_Event_Group_Pointers	EVF_Event_Group_Pointers
NU_HISR_Information	TCF_HISR_Information
NU_HISR_Pointers	TCF_HISR_Pointers
NU_License_Information	LIC_License_Information
NU_Local_Control_Interrupts	TCT_Local_Control_Interrupts
NU_Mailbox_Information	MBF_Mailbox_Information
NU_Mailbox_Pointers	MBF_Mailbox_Pointers
NU_Make_History_Entry	HIC_Make_History_Entry_Service
NU_Memory_Pool_Information	DMF_Memory_Pool_Information
NU_Memory_Pool_Pointers	DMF_Memory_Pool_Pointers
NU_Obtain_Semaphore	SMCE_Obtain_Semaphore

NU_Partition_Pool_Information	PMF_Partition_Pool_Information
NU_Partition_Pool_Pointers	PMF_Partition_Pool_Pointers
NU_Pipe_Information	PIF_Pipe_Information
NU_Pipe_Pointers	PIF_Pipe_Pointers
NU_Protect	TCT_Protect
NU_Queue_Information	QUF_Queue_Information
NU_Queue_Pointers	QUF_Queue_Pointers
NU_Receive_From_Mailbox	MBCE_Receive_From_Mailbox
NU_Receive_From_Pipe	PICE_Receive_From_Pipe
NU_Receive_From_Queue	QUCE_Receive_From_Queue
NU_Receive_Signals	TCSE_Receive_Signals
NU_Register_LISR	TCC_Register_LISR
NU_Register_Signal_Handler	TCSE_Register_Signal_Handler
NU_Release_Information	RLC_Release_Information
NU_Release_Semaphore	SMCE_Release_Semaphore
NU_Relinquish	TCCE_Relinquish
NU_Request_Driver	IOCE_Request_Driver
NU_Reset_Mailbox	MBSE_Reset_Mailbox
NU_Reset_Pipe	PISE_Reset_Pipe
NU_Reset_Queue	QUSE_Reset_Queue
NU_Reset_Semaphore	SMSE_Reset_Semaphore
NU_Reset_Task	TCCE_Reset_Task
NU_Reset_Timer	TMSE_Reset_Timer
NU_Restore_Interrupts	TCT_Restore_Interrupts
NU_Resume_Driver	IOCE_Resume_Driver
NU_Resume_Task	TCCE_Resume_Service
NU_Retrieve_Clock	TMT_Retrieve_Clock
NU_Retrieve_Events	EVCE_Retrieve_Events
NU_Retrieve_History_Entry	HIC_Retrieve_History_Entry
NU_Semaphore_Information	SMF_Semaphore_Information
NU_Semaphore_Pointers	SMF_Semaphore_Pointers
NU_Send_Signals	TCSE_Send_Signals
NU_Send_To_Front_Of_Pipe	PISE_Send_To_Front_Of_Pipe
NU_Send_To_Front_Of_Queue	QUSE_Send_To_Front_Of_Queue
NU_Send_To_Mailbox	MBCE_Send_To_Mailbox
NU_Send_To_Pipe	PICE_Send_To_Pipe
NU_Send_To_Queue	QUCE_Send_To_Queue
NU_Set_Clock	TMT_Set_Clock
NU_Set_Events	EVCE_Set_Events
NU_Setup_Vector	INT_Setup_Vector
NU_Sleep	TCCE_Task_Sleep
NU_Suspend_Driver	IOCE_Suspend_Driver
NU_Suspend_Task	TCCE_Suspend_Service
NU_Task_Information	TCF_Task_Information
NU_Task_Pointers	TCF_Task_Pointers
NU_Terminate_Task	TCCE_Terminate_Task
NU_Timer_Information	TMF_Timer_Information
NU_Timer_Pointers	TMF_Timer_Pointers
NU_Unprotect	TCT_Unprotect

- **无错误检查 No Error Checking**

  如果宏NU_NO_ERROR_CHECKING 标志被定义(通常使用-D 编译开关)，《*Nucleus PLUS Reference*

Manual》定义的系统服务NU_* 被映射到如下的内部函数：

Nucleus PLUS Service	Internal Function
NU_Activate_HISR	TCC_Activate_HISR
NU_Allocate_Memory	DMC_Allocate_Memory
NU_Allocate_Partition	PMC_Allocate_Partition
NU_Broadcast_To_Mailbox	MBS_Broadcast_To_Mailbox
NU_Broadcast_To_Pipe	PIS_Broadcast_To_Pipe
NU_Broadcast_To_Queue	QUS_Broadcast_To_Queue
NU_Change_Preemption	TCS_Change_Preemption
NU_Change_Priority	TCS_Change_Priority
NU_Change_Time_Slice	TCS_Change_Time_Slice
NU_Check_Stack	TCT_Check_Stack
NU_Control_Interrupts	TCT_Control_Interrupts
NU_Control_Signals	TCS_Control_Signals
NU_Control_Timer	TMS_Control_Timer
NU_Create_Driver	IOC_Create_Driver
NU_Create_Event_Group	EVC_Create_Event_Group
NU_Create_HISR	TCC_Create_HISR
NU_Create_Mailbox	MBC_Create_Mailbox
NU_Create_Memory_Pool	DMC_Create_Memory_Pool
NU_Create_Partition_Pool	PMC_Create_Partition_Pool
NU_Create_Pipe	PIC_Create_Pipe
NU_Create_Queue	QUC_Create_Queue
NU_Create_Semaphore	SMC_Create_Semaphore
NU_Create_Task	TCC_Create_Task
NU_Create_Timer	TMS_Create_Timer
NU_Current_HISR_Pointer	TCF_Current_HISR_Pointer
NU_Current_Task_Pointer	TCC_Current_Task_Pointer
NU_Deallocate_Memory	DMC_Deallocate_Memory
NU_Deallocate_Partition	PMC_Deallocate_Partition
NU_Delete_Driver	IOC_Delete_Driver
NU_Delete_Event_Group	EVC_Delete_Event_Group
NU_Delete_HISR	TCC_Delete_HISR
NU_Delete_Mailbox	MBC_Delete_Mailbox
NU_Delete_Memory_Pool	DMC_Delete_Memory_Pool
NU_Delete_Partition_Pool	PMC_Delete_Partition_Pool
NU_Delete_Pipe	PIC_Delete_Pipe
NU_Delete_Queue	QUC_Delete_Queue
NU_Delete_Semaphore	SMC_Delete_Semaphore
NU_Delete_Task	TCC_Delete_Task
NU_Delete_Timer	TMS_Delete_Timer
NU_Disable_History_Saving	HIC_Disable_History_Saving
NU_Driver_Pointers	IOF_Driver_Pointers
NU_Enable_History_Saving	HIC_Enable_History_Saving
NU_Established_Drivers	IOF_Established_Drivers
NU_Established_Event_Groups	EVF_Established_Event_Groups
NU_Established_HISRs	TCF_Established_HISRs
NU_Established_Mailboxes	MBF_Established_Mailboxes
NU_Established_Memory_Pools	DMF_Established_Memory_Pools
NU_Established_Partition_Pools	PMF_Established_Partition_Pools
NU_Established_Pipes	PIF_Established_Pipes
NU_Established_Queues	QUF_Established_Queues

NU_Established_Semaphores	SMF_Established_Semaphores
NU_Established_Tasks	TCF_Established_Tasks
NU_Established_Timers	TMF_Established_Timers
NU_Event_Group_Information	EVF_Event_Group_Information
NU_Event_Group_Pointers	EVF_Event_Group_Pointers
NU_HISR_Information	TCF_HISR_Information
NU_HISR_Pointers	TCF_HISR_Pointers
NU_License_Information	LIC_License_Information
NU_Local_Control_Interrupts	TCT_Local_Control_Interrupts
NU_Mailbox_Information	MBF_Mailbox_Information
NU_Mailbox_Pointers	MBF_Mailbox_Pointers
NU_Make_History_Entry	HIC_Make_History_Entry_Service
NU_Memory_Pool_Information	DMF_Memory_Pool_Information
NU_Memory_Pool_Pointers	DMF_Memory_Pool_Pointers
NU_Obtain_Semaphore	SMC_Obtain_Semaphore
NU_Partition_Pool_Information	PMF_Partition_Pool_Information
NU_Partition_Pool_Pointers	PMF_Partition_Pool_Pointers
NU_Pipe_Information	PIF_Pipe_Information
NU_Pipe_Pointers	PIF_Pipe_Pointers
NU_Protect	TCT_Protect
NU_Queue_Information	QUF_Queue_Information
NU_Queue_Pointers	QUF_Queue_Pointers
NU_Receive_From_Mailbox	MBC_Receive_From_Mailbox
NU_Receive_From_Pipe	PIC_Receive_From_Pipe
NU_Receive_From_Queue	QUC_Receive_From_Queue
NU_Receive_Signals	TCS_Receive_Signals
NU_Register_LISR	TCC_Register_LISR
NU_Register_Signal_Handler	TCS_Register_Signal_Handler
NU_Release_Information	RLC_Release_Information
NU_Release_Semaphore	SMC_Release_Semaphore
NU_Relinquish	TCC_Relinquish
NU_Request_Driver	IOC_Request_Driver
NU_Reset_Mailbox	MBS_Reset_Mailbox
NU_Reset_Pipe	PIS_Reset_Pipe
NU_Reset_Queue	QUS_Reset_Queue
NU_Reset_Semaphore	SMS_Reset_Semaphore
NU_Reset_Task	TCC_Reset_Task
NU_Reset_Timer	TMS_Reset_Timer
NU_Restore_Interrupts	TCT_Restore_Interrupts
NU_Resume_Driver	IOC_Resume_Driver
NU_Resume_Task	TCC_Resume_Service
NU_Retrieve_Clock	TMT_Retrieve_Clock
NU_Retrieve_Events	EVC_Retrieve_Events
NU_Retrieve_History_Entry	HIC_Retrieve_History_Entry
NU_Semaphore_Information	SMF_Semaphore_Information
NU_Semaphore_Pointers	SMF_Semaphore_Pointers
NU_Send_Signals	TCS_Send_Signals
NU_Send_To_Front_Of_Pipe	PIS_Send_To_Front_Of_Pipe
NU_Send_To_Front_Of_Queue	QUS_Send_To_Front_Of_Queue
NU_Send_To_Mailbox	MBC_Send_To_Mailbox
NU_Send_To_Pipe	PIC_Send_To_Pipe
NU_Send_To_Queue	QUC_Send_To_Queue
NU_Set_Clock	TMT_Set_Clock

NU_Set_Events	EVC_Set_Events
NU_Setup_Vector	INT_Setup_Vector
NU_Sleep	TCC_Task_Sleep
NU_Suspend_Driver	IOC_Suspend_Driver
NU_Suspend_Task	TCC_Suspend_Service
NU_Task_Information	TCF_Task_Information
NU_Task_Pointers	TCF_Task_Pointers
NU_Terminate_Task	TCC_Terminate_Task
NU_Timer_Information	TMF_Timer_Information
NU_Timer_Pointers	TMF_Timer_Pointers
NU_Unprotect	TCT_Unprotect

● 条件编译 Conditional Compilation

Nucleus PLUS 源码有一定的条件编译选项。有一些是在应用程序编译时可用，而绝大多数是在Nucleus PLUS系统库编译时可用。

● 系统库条件编译标识 Library Conditional Flags

Nucleus PLUS 系统库条件编译标识往往在一个编译批处理文件中使用。这些条件编译选项控制各种 Nucleus PLUS 系统库特征。这些条件编译选项如下：

条件编译选项	意义
NU_ENABLE_HISTORY	允许在特定的文件中进行历史记录记录。注意仅仅行如**C.C的文件是受该选项影响
NU_ENABLE_STACK_CHECK	允许在特定的文件中每个函数开始进行堆栈检查。注意仅仅行如**C.C的文件是受该选项影响
NU_ERROR_STRING	允许在一个致命的系统错误发生时生成一个ASCII的错误字符串。这个标记是适用于 ERD.C, ERI.C, 和 ERC.C这三个文件。
NU_NO_ERROR_CHECKING	禁止在文件TMI.C中创建定时器HISR时使用错误检查外壳函数
NU_DEBUG	映射定义在NUCLEUS.H的应用数据结构到Nucleus PLUS使用的实际的内部数据结构。这个选项允许用户直接检查所有的Nucleus PLUS的数据结构。所有的库文件和应用文件可以使用或不使用该选项。
NU_INLINE	通过in-line 代码替换一些linked-list的处理过程来改善执行效率。这个标记适用于所有的**C.C 或者 **S.C 文件。

● 系统库条件编译值 Library Conditional Values

在扩展的条件编译标识中，有几个定义在NUCLEUS.H中的条件编译值。应该非常小心地改变这些条件编译值(除开R1, R2, R3, R4)。这些条件编译值如下：

条件编译值	意义
NU_POINTER_ACCESS	This value specifies how many separate memory accesses are required to load and store a data pointer. A value of one allows an in-line optimization. Any value greater

	than one uses a function to load/store certain data pointers under protection from interrupts.
PAD_1	This value specifies how many bytes of padding should be added after a single character in a structure.
PAD_2	This value specifies how many bytes of padding should be added after two consecutive characters in a structure.
PAD_3	This value specifies how many bytes of padding should be added after three consecutive characters in a structure.
R1, R2, R3, R4	These values are used to place the “register” modifier in front of frequently used variables in Nucleus PLUS. R1 is used to modify the most frequently used variable. By defining any of these to “register” the corresponding variable in the source code is assigned register status.

- **应用程序条件编译标识 Application Conditional Flags**

当编译一个应用程序时有几个条件编译标识可供选择。Nucleus PLUS 应用程序可以通过在编译行定义NU_NO_ERROR_CHECKING 宏来禁止对通用系统服务的参数进行错误检查。这可以减少运行时间和减少代码大小。通过在编译时定义NU_DEBUG 选项，定义在UCLEUS.H的应用程序数据结构能直接映射到Nucleus PLUS内部数据结构。这允许用户直接在源码调试环境下检查每一个Nucleus PLUS 内部数据结构。如果NU_DEBUG 选项被使用，那么最好重新用NU_DEBUG 选项编译所以的Nucleus PLUS源码。

目标环境依赖 Environment Dependencies

依赖于目标处理器和开发工具的Nucleus PLUS 文件仅有独立的四个文件。其中三个(INT.?, TCT.?, and TMT.?)是一般用汇编语言编写的。这些文件为目标环境提供底层的基本运行平台。另一个文件是NUCLEUS.H ，它直接或间接地被系统所以文件包含。该文件定义了大量地数据类型和与处理器和开发工具相关地信息。

- **初始化 Initialization**

文件INT.[S, ASM, or SRC] 负责底层的初始化和访问处理器中断向量表服务。该文件也包含一些缺省地中断处理服务(ISR) 函数。函数INT_Initialize 是和给定的目标板相关的。例如，如果处理器不能产生内部定时中断，则设置定时器就是和板级相关的。这意味着即使是相同的处理器结构，INT 可能需要修改对应不同的目标板。

- **线程控制 Thread Control**

文件TCT.[S, ASM, or SRC] 首要责任是在系统和线程中进行控制器权转移。线程被定义为一个Nucleus PLUS 任务或一个Nucleus PLUS HISR。这个文件包含在任务和HISR进行上下文切换的所有必需的代码。 另外该文件也包含处理竞争冲突和任务信号量的所有必需的代码。

- **定时管理 Timer Management**

文件TMT.[S, ASM, or SRC] 主要负责处理Nucleus PLUS的定时中断，包括定时器中断处理函数。在绝大多数移植系统中定时中断处理函数被设计为low-overhead当没有定时期满时。

- **Nucleus PLUS包含文件 Nucleus PLUS Include File**

所有的Nucleus PLUS 源文件直接或间接地包含NUCLEUS.H 文件。引用Nucleus PLUS 系统服务和/或数据结构 地应用程序文件也必须包含NUCLEUS.H文件。该文件定义了大量数据类型、中断禁止/使能的值、中断向量的个数、系统控制块的大小和其他和目标板相关的信息.

版本控制 Version Control

在Nucleus PLUS 系统中有几个不同的版本层。系统版本定义在文件RLD.C 中的ASCII 字符串RLD_Release_String。这个版本信息包括通用C源码版本和目标板特殊源码版本。例如，目标板特殊源码DOS/Borland v 1.1和通用C源码v 2.2 的版本字符是“Copyright (c) 199x ATI - Nucleus PLUS - DOS Borland C Version 1.1.G2.2”

Nucleus PLUS 每一个文件也有版本信息。这个在文件开头的版本信息指示该文件的版本标识。许多情况下，在文件开头的版本信息是和系统版本信息不同。在文件里的函数在函数开头也有版本信息。这个在函数开头的版本信息指示该函数做了那些改动和它可以和那些Nucleus PLUS文件版本匹配。

第四章．组件描述

通用服务组件 Common Service Component (CS)
初始化组件 Initialization Component (IN)
线程调度组件 Thread Control Component (TC)
定时组件 Timer Component (TM)
活动定时队列 Active Timers List
邮箱组件 Mailbox Component (MB)
队列组件 Queue Component (QM)
管道组件 Pipe Component (PI)
信号量组件 Semaphore Component (SM)
事件组件 Event Flag Component (EV)
存储分配组件 Partition Memory Component (PM)
动态存储组件 Dynamic Memory Component (DM)
输入输出组件 Input/Output DriverComponent (IO)
历史组件 History Component (HI)
错误处理组件 Error Component (ER)
许可证组件 License Component(LI)
版本控制组件 Release Component(RL)

本章描述了Nucleus PLUS的各种组件的源码文件、数据结构、函数功能。Nucleus PLUS 共有16个不同的组件.

通用服务组件(CS)

通用服务组件 (CS)负责提供给其他Nucleus PLUS 组件，使其具有链表功能。每一个通用服务组件节点数据结构都是被其他系统数据结构包含。

通用服务组件文件

通用服务组件(CS)包括三个文件。每一个通用服务组件文件详细说明如下：

文件	描述
CS_DEFS.H	这个文件包括CS用到的一些特殊的数据结构。

CS_EXTR.H	这个文件定义了所有的CS外部接口。
CSC.C	这个文件包括所有CS函数。这些函数完成基本的在链表中添加和删除表项的功能。

通用服务控制块

通用服务控制块CS_NODE 包括向前和向后的指针 （the previous and next pointers）和其他处理通用服务请求的必要的结构。

结构定义

```
struct CS_NODE_STRUCT *cs_previous
struct CS_NODE_STRUCT *cs_next
DATA_ELEMENT cs_priority
DATA_ELEMENT cs_padding[PAD_1]
```

结构概述

*cs_previous 指向通用服务链表的前一个表项。.它是通用服务链表的一部分，通用服务链表是一个双向的循环链表。

*cs_next 指向通用服务链表的后一个表项。.它是通用服务链表的一部分，通用服务链表是一个双向的循环链表。

cs_priority 指示任务或高级中断服务程序的优先级。

cs_padding 这被用于对齐通用服务组件(CS)This is used to align the Common Services structure on an even boundary. In some ports this field is not used.

通用服务函数

以下部分概要地描述了通用服务组件的函数。回顾实际的源码可以获得更多的信息。

- CSC_Place_On_List

这个函数将一个表项插入到一个双向的循环链表中。

语法

```
VOID CSC_Place_On_List(CS_NODE **head, CS_NODE *new_node)
```

函数调用

无

- CSC_Priority_Place_On_List

这个函数将一个表项按它的优先级高低放置在链表中。这个表项被放置在所以优先级比它高或和它相同的表项之后。注意数字越低优先级越高。

语法

```
VOID CSC_Priority_Place_On_List(CS_NODE **head, CS_NODE *new_node)
```

函数调用

无

- CSC_Remove_From_List

这个函数将一个表项从链表中删除list.

语法

```
VOID CSC_Remove_From_List(CS_NODE **head, CS_NODE *node)
```

函数调用

无

初始化组件（IN）

初始化组件负责初始化Nucleus PLUS系统。通常有两个初始化过程。首先初始化和目标板有关的部分，再初始化每一个Nuclues PLUS组件。最后的初始化过程再Application_Initialize被调用，它包括一些用户定义的初始化。初始化完成后控制权被转交给线程调度TCT_Schedule。请参考《Nucleus PLUS参考手册》第三章得到更多的初始化信息。

初始化组件文件

初始化组件(IN)包括三个文件，每一个初始化组件文件详细说明如下：

文件	描述
IN_EXTR.H	这个文件定义了所有的IN外部接口。
INC.C	这个文件定义了IN的核心函数。这些函数处理基本的系统初始化服务。
INT.[S,ASM,SRC]	这个文件包括所以的和目标板有关的IN函数。一个简单的初始化文件也为用户用途提供。

初始化组件函数

以下部分概要地描述了通用服务组件(IN)的函数。回顾实际的源码可以获得更多的信息。

- INC_Initialize

这是系统主初始化函数。所有的组件都被该函数初始化。再系统初始化完成后，Application_Initialize 程序被调用，在所以的初始化完成后，该函数调用TCT_Schedule开始进行任务调度。

语法

VOID INC_Initialize(VOID *first_available_memory)

函数调用

Application_Initialize
RLC_Release_Information
LIC_License_Information
ERI_Initialize
HII_Initialize
TCI_Initialize
MBI_Initialize
QUI_Initialize

- INT_Initialize

这是一个汇编函数，它处理底层的和目标板相关的初始化。当这个函数完成后，控制权转交给和目标板无关的初始化函数**INC_Initialize**。这个函数必须提供如下功能：

设置必须的处理器/系统控制寄存器
初始化向量表
设置系统堆栈指针
设置定时中断
计算（设置）定时高级中断服务程序堆栈和优先级
计算（设置）最先可用存储器开始地址

- 移交控制权给INC_Initialize完成初始化系统组件

语法

VOID INT_Initialize(void)

函数调用

INC_Initialize

- INT_Vectors_Loaded

这是一个汇编函数，它通过返回标记指示缺省向量表是否已被设置。如果标记是false，表示每一个 LISR 寄存器已经 设置ISR shell 为实际的向量表。

语法
INT INT_Vectors_Loaded(void)
函数调用
无

- INT_Setup_Vector

这是一个汇编函数，它用新的向量代替旧的向量，以前的向量被返回给调用者。
语法
VOID *INT_Setup_Vector(INT vector, VOID *new)
函数调用
无

线程控制组件（TC）

线程控制组件(TC)是负责管理相互竞争的执行实体，实时Nucleus PLUS任务和高级中断服务程序(HISRs)。每个Nucleus Plus 任务完成不同的目的，大多数应用有许多任务。为了控制这些执行的实体，任务通常被分配一个优先级。Nucleus PLUS 优先级范围从 0到 255，这儿 0 是最高的优先级，255 是最低的优先级。相比较低的优先级任务，优先级较高的任务优先执行。而且优先级较高的任务处于就学状态时，它可以抢占优先级低的任务，除非优先级低的任务不允许抢占。任务总是处与五个状态中的一个：执行、就绪、挂起、停止、完成。

Nucleus PLUS高级中断服务程序(HISRs)是ISR可被Nucleus PLUS服务中断的部分。高级中断服务程序(HISRs)有0到2的优先级，0是最高的优先级。请参考《Nucleus PLUS参考手册》第三章得到更多的线程控制组件信息。

线程控制组件文件

线程控制组件 (TC) 包括九个文件。每一个初始化组件文件详细说明如下：

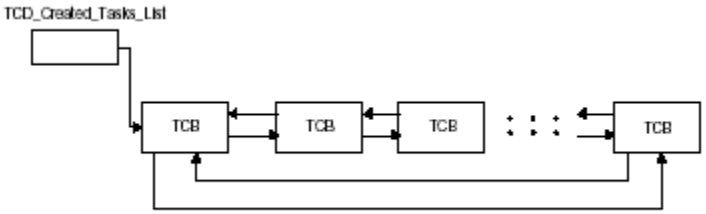
文件	描述
TC_DEFS.H	这个文件包括TC用到的一些特殊的数据结构。
TC_EXTR.H	这个文件定义了所有的TC外部接口。
TCD.C	这个文件包括TC用到的所有的全局数据结构。
TCI.C	这个文件包括TC的初始化函数。
TCF.C	这个文件包括TC
TCC.C	这个文件包括TC处理创建任务和删除任务的核心函数。
TCS.C	这个文件包括一些补充的TC函数，这些函数往往较少使用。
TCCE.C	这个文件包括包含在TCC.c文件定义的函数的错误检查函数
TCSE.C	这个文件包括包含在TCS.c文件定义的函数的错误检查函数
TCT.[S,ASM,SRC]	这个文件包括所以的和目标板有关的TC函数。

线程控制组件数据结构

已创建任务链表

Nucleus PLUS 能动态地创建和删除。每一个已创建任务地线程控制块(TCB)都被保存在一个双向地循环链表中。新近创建地任务被放置在链表地尾部，而删除一个任务就被完全地从链表中删除。这个链表地头指针是

TCD_Created_Tasks_List。



任务总数

当前已创建Nucleus PLUS任务是用一个变量TCD_Total_Tasks表明. 这个变量地内容相应表明在已创建链表中的TCBs的个数。Manipulation of this variable is also done under the protection of TCD_List_Protect.

已创建任务保护链表

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建任务链表的完整性。 这种保护归功于使用名为TCD_List_Protect的内在的保护结构。所以的任务的创建和删除都在TCD_List_Protect的保护下。

数据成员描述

TC_TCB *tc_tcb_pointer

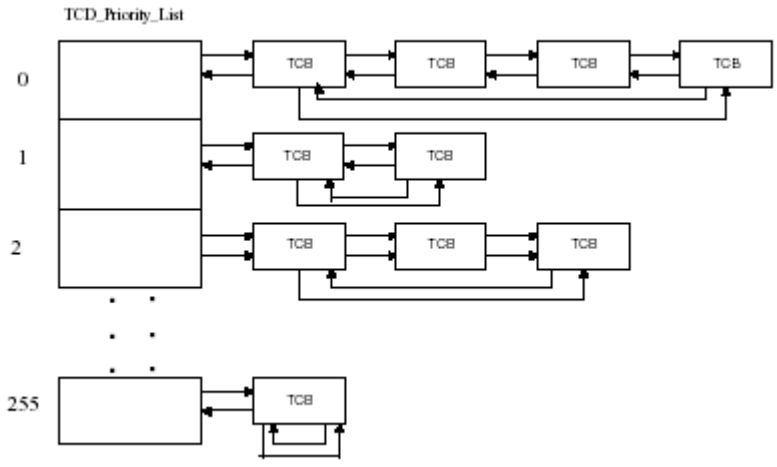
UNSIGNED tc_thread_waiting

数据成员概要

数据成员	描述
tc_tcb_pointer	当前被保护线程的标识符
tc_thread_waiting	指示一个或多个线程在等待的标记

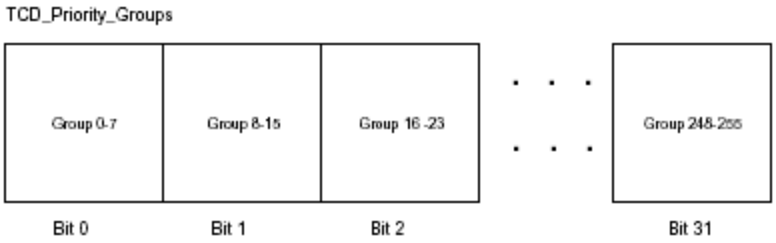
优先级链表

TCD_Priority_List 是一个TCP指针的数组。这个数组的每一个元素是在此优先级下准备执行任务的链表头指针。如果此优先级为空，侧在此优先级下无任务准备执行。这个数组是靠优先级检索的。



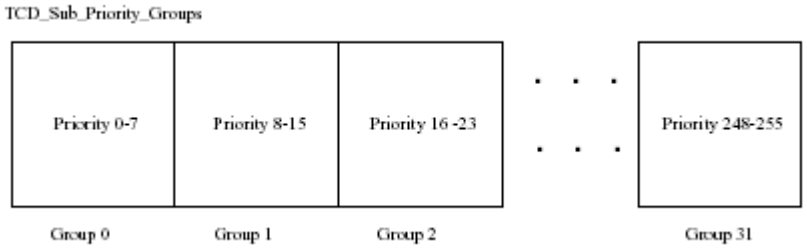
优先级组

TCD_Priority_Groups是一个32-位的无符号的整数。每一位对应一个8个优先级的组。例如，如果位0被设置，就表示优先级0到8的任务中至少有一个准备好执行。



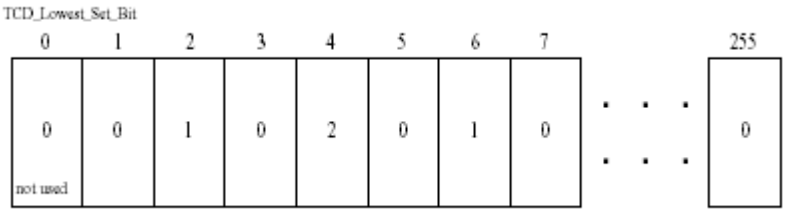
优先级子组

Nucleus PLUS 通过使用优先级子组精确的决定那一个优先级从优先级组。TCD_Sub_Priority_Groups是一个由优先级子组构成的数组。下标 0 对应优先级 0 到 8。每一个元素的第0位对应优先级0，而第7位对应优先级7。



Lowest Bit Set

TCD_Lowest_Set_Bit仅仅只是一个查照表。该表是由1 to 255的值进行索引的。 is nothing more than a standard look-up table. The table is
indexed by values ranging from 1 to 255. The value at any position in the table contains the lowest set bit for that value. This is used to determine the highest priority task represented in the previously defined bit maps. For example, the lowest bit set for the value of 7 is contained in index 7 of the TCD_Lowest_Set_Bit array. In the table below, the value of 7 is shown to have bit 0 set, which is correct.



执行的任务

Nucleus PLUS 维护一个指向要被执行的任务的指针。这个指针被叫做TCD_Execute_Task。注意TCD_Execute_Task并不是必然地指向当前正在执行任务的指针。There are several points in the system where this is true. Two common situations arise during task preemption and during task protection conflicts.

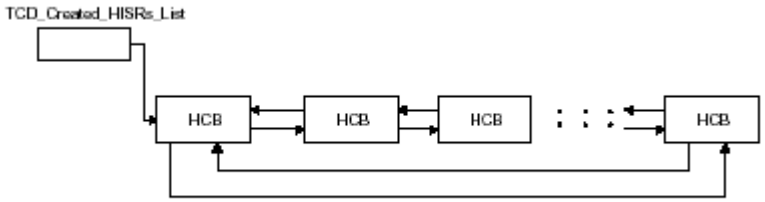
最高优先级

Nucleus PLUS 变量 TCD_Highest_Priority 指示准备执行任务的最高优先级。注意这对于描述当前正在执行任务的优先级并不是必须的。如果当前正在执行任务禁止优先级，则该变量是必须的。如果当前没有任务执

行，那么该变量的值设置为最大的优先级。

已创建 HISRs 链表

TCD_Created_HISRs_List 是已创建HISR链表的头指针。如果这个指针为空NU_NULL, 那么当前没有HISR被创建。



HISRs 总数

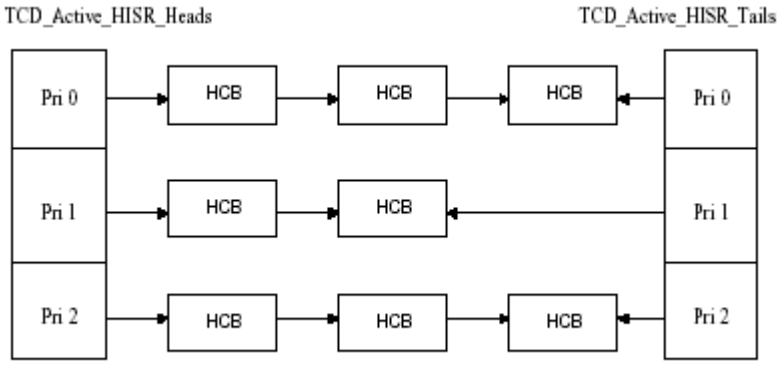
当前已创建的Nucleus PLUS HISRs 的总数是用变量**TCD_Total_HISRs**指示。这个变量指示对应在已创建链表的**HCBs**的个数。TCD_HISR_Protect 的保护也处理该变量的值。

激活HISR 头指针集合

Nucleus PLUS 维护一个记录激活所有HISR链表头指针的数组。这个数组叫做**TCD_Active_HISR_Heads**。有三种**HISR**的优先级被允许。HISR 的优先级为该数组的下标。优先级/下标 0 意味最高的优先级，而优先级/下标 2 意味最低的优先级。

激活HISR 尾指针集合

Nucleus PLUS 维护一个记录激活所有HISR链表尾指针的数组。这个数组叫做TCD_Active_HISR_Tails。有三种**HISR**的优先级被允许。HISR 的优先级为该数组的下标。优先级/下标 0 意味最高的优先级，而优先级/下标 2 意味最低的优先级。



执行 HISR

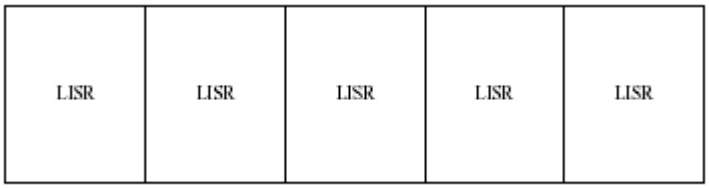
TCD_Execute_HISR 包含一个指向最高优先级的准备执行的HISR。如果这个指针为空，当前没有HISR被激活。注意当前线程总是和该指针不相等。

当前线程

当前执行线程控制块被保存在变量**TCD_Current_Thread**中。 所以，该变量指针指向**TC_TCB** 或者**TC_HCB** 结构。除开初始化，该变量仅被TC组件的和目标板相关部分设置或者清除。

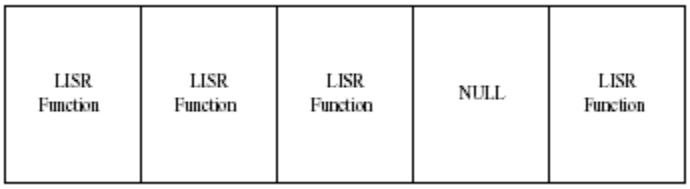
已注册LISR

Nucleus PLUS 维护一个叫做 TCD_Registered_LISRs 的链表，该链表指示LISR是否注册了一个给定的中断。该链表以不为零的向量为下标。该链表的值可以作为LISR 指针数组的下标。



LISR 指针 LISR Pointers

TCD_LISR_Pointers 是一个当中断产生时指向要调用的LISR函数的指针的链表。如果该函数入口为空，表示没有特定的LISR函数被调用，反之该函数入口可被使用。



中断个数 Interrupt Count

当前处理中断服务程序的个数是用变量 TCD_Interrupt_Count表示。如果该变量内容为零，表示没有中断被处理。如果该变量内容大于 1 ，表示嵌套中断被处理。

堆栈交换 Stack Switched

TCD_Stack_Switched 是一个指明系统堆栈在线程上下文被保存后是否切换的标识。有些移植系统并不使用该变量。

系统保护 System Protect

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护系统数据结构的完整性。 这种保护归功于使用名为TCD_System_Protect的内在的保护结构。所以的系统的创建和删除都在TCD_System_Protect的保护下。

数据成员描述

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

数据成员概要

数据成员	描述
tc_tcb_pointer	标识当前被保护的线程
tc_thread_waiting	标识一个或更多的正在等待的需要保护的线程。

系统堆栈 System Stack

```
VOID            *TCD_System_Stack;
```

TCD_System_Stack 指向系统堆栈的基地址。当系统是空闲时或中断处理中，系统堆栈被使用。这个变量一般在和目标板相关部分进行初始化。

LISR保护 LISR Protect

```
TC_PROTECT      TCD_LISR_Protect;
```

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护LISR数据结构的完整性。 这种保护归功于使用名为TCD_LISR_Protect的内在的保护结构。所以的LISR的创建和删除都在TCD_LISR_Protect的保护下。

数据成员描述
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting

数据成员概要

数据成员	描述
tc_tcb_pointer	标识当前被保护的线程
tc_thread_waiting	标识一个或更多的正在等待的需要保护的线程。

HISR保护 HISR Protect
TC_PROTECT TCD_HISR_Protect;

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护HISR数据结构的完整性。 这种保护归功于使用名为TCD_HISR_Protect的内在的保护结构。所以的LISR的创建和删除都在TCD_HISR_Protect的保护下。

数据成员描述
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting

数据成员概要

数据成员	描述
tc_tcb_pointer	标识当前被保护的线程
tc_thread_waiting	标识一个或更多的正在等待的需要保护的线程。

中断级别 Interrupt Level
INT TCD_Interrupt_Level;

TCD_Interrupt_Level 表示在系统中允许中断的状态。有些移植系统该变量为布尔型。

未处理中断 Unhandled Interrupts
INT TCD_Unhandled_Interrupt;

Nucleus PLUS 维护一个表明在系统错误状态下最后的未处理的中断向量数，该变量是TCD_Unhandled_Interrupts。

任务控制块 Task Control Block

```
typedef struct TC_TCB_STRUCT
{
    /* Standard thread information first.  This information is used by
       the target dependent portion of this component.  Changes made
       to this area of the structure can have undesirable side effects.  */

    CS_NODE             tc_created;           /* Node for linking to    */
                                              /*   created task list    */
    UNSIGNED            tc_id;                /* Internal TCB ID        */
    CHAR                tc_name[NU_MAX_NAME]; /* Task name              */
    DATA_ELEMENT        tc_status;            /* Task status            */
    DATA_ELEMENT        tc_delayed_suspend;   /* Delayed task suspension*/
    DATA_ELEMENT        tc_priority;          /* Task priority          */
    DATA_ELEMENT        tc_preemption;        /* Task preemption enable */
    UNSIGNED            tc_scheduled;         /* Task scheduled count   */
```

```
    UNSIGNED            tc_cur_time_slice;      /* Current time slice     */
    VOID                *tc_stack_start;        /* Stack starting address */
    VOID                *tc_stack_end;          /* Stack ending address   */
    VOID                *tc_stack_pointer;      /* Task stack pointer     */
    UNSIGNED            tc_stack_size;          /* Task stack's size      */
    UNSIGNED            tc_stack_minimum;       /* Minimum stack size     */
    struct TC_PROTECT_STRUCT
                        *tc_current_protect;    /* Current protection     */
    VOID                *tc_saved_stack_ptr;    /* Previous stack pointer */
    UNSIGNED            tc_time_slice;          /* Task time slice value  */


    /* Information after this point is not used in the target dependent
       portion of this component.  Hence, changes in the following section
       should not impact assembly language routines.  */
    struct TC_TCB_STRUCT
                        *tc_ready_previous,     /* Previously ready TCB   */
                        *tc_ready_next;         /* next and previous ptrs */

    /* Task control information follows.  */

    UNSIGNED            tc_priority_group;      /* Priority group mask bit*/
    struct TC_TCB_STRUCT
                       **tc_priority_head;      /* Pointer to list head   */
    DATA_ELEMENT        *tc_sub_priority_ptr;   /* Pointer to sub-group   */
    DATA_ELEMENT        tc_sub_priority;        /* Mask of sub-group bit  */
    DATA_ELEMENT        tc_saved_status;        /* Previous task status   */
    DATA_ELEMENT        tc_signal_active;       /* Signal active flag     */

#if     PAD_3
    DATA_ELEMENT        tc_padding[PAD_3];
#endif

                                                /* Task entry function    */
    VOID                (*tc_entry)(UNSIGNED, VOID *);
    UNSIGNED            tc_argc;                /* Optional task argument */
    VOID                *tc_argv;               /* Optional task argument */
    VOID                (*tc_cleanup) (VOID *);/* Clean-up routine       */
    VOID                *tc_cleanup_info;       /* Clean-up information   */
    struct TC_PROTECT_STRUCT
                        *tc_suspend_protect;    /* Protection at time of  */
                                                /*   task suspension      */

    /* Task timer information.  */
    INT                 tc_timer_active;        /* Active timer flag      */
    TM_TCB              tc_timer_control;       /* Timer control block    */

    /* Task signal control information.  */

    UNSIGNED            tc_signals;             /* Current signals        */
    UNSIGNED            tc_enabled_signals;     /* Enabled signals        */

    /* tc_saved_status and tc_signal_active are now defined above in an
       attempt to keep DATA_ELEMENT types together.  */
```

```
    /* Signal handling routine.  */
    VOID                (*tc_signal_handler) (UNSIGNED);

    /* Reserved words for the system and a single reserved word for the
       application.  */
    UNSIGNED            tc_system_reserved_1;  /* System reserved word   */
    UNSIGNED            tc_system_reserved_2;  /* System reserved word   */
    UNSIGNED            tc_system_reserved_3;  /* System reserved word   */
    UNSIGNED            tc_app_reserved_1;     /* Application reserved   */

#if      NU_FPU_ENABLE
    char _fp_regs [NU_FPU_SIZE_AREA];
#endif

} TC_TCB;
```

任务控制块 TC_TCB 包括任务的优先级和其他任务控制处理必须的数据成员。

数据成员描述

```
CS_NODE tc_created
UNSIGNED tc_id
CHAR tc_name[NU_MAX_NAME]
DATA_ELEMENT tc_status
DATA_ELEMENT tc_delayed_suspend
DATA_ELEMENT tc_priority
DATA_ELEMENT tc_preemption
UNSIGNED tc_scheduled
UNSIGNED tc_cur_time_slice
VOID *tc_stack_start
VOID *tc_stack_end
VOID *tc_stack_pointer
UNSIGNED tc_stack_size
UNSIGNED tc_stack_minimum
struct TC_PROTECT_STRUCT *tc_current_protect
VOID *tc_saved_stack_ptr
UNSIGNED tc_time_slice
struct TC_TCB_STRUCT *tc_ready_previous
struct TC_TCB_STRUCT *tc_ready_next
UNSIGNED tc_priority_group
TC_TCB_STRUCT **tc_priority_head
DATA_ELEMENT *tc_sub_priority_ptr
DATA_ELEMENT tc_sub_priority
DATA_ELEMENT tc_saved_status
DATA_ELEMENT tc_signal_active
DATA_ELEMENT tc_padding[PAD_3]
VOID (*tc_entry)(UNSIGNED, VOID *)
UNSIGNED tc_argc
VOID *tc_argv
VOID (*tc_cleanup)(VOID *)
VOID *tc_cleanup_info
struct TC_PROTECT_STRUCT *tc_suspend_protect
INT tc_timer_active
TM_TCB tm_timer_control
UNSIGNED tc_signals
UNSIGNED tc_enabled_signals
```

```
VOID (*tc_signal_handler)(UNSIGNED)
UNSIGNED tc_system_reserved_1
UNSIGNED tc_system_reserved_2
UNSIGNED tc_system_reserved_3
UNSIGNED tc_app_reserved_1
```

数据成员概要

数据成员	概要
tc_created	为连接到已创建任务双向循环链表的节点
tc_id	内部的任务标识
tc_name	用户定义的8个字符的任务名字
tc_status	任务的当前状态
tc_delayed_suspend	表明任务是否挂起的标识
tc_priority	任务的当前优先级
tc_preemption	表明任务是否允许抢占的标识
tc_scheduled	指明任务调度的个数
tc_cur_time_slice	指明当前任务的时间片
*tc_stack_start	指向任务堆栈的开始地址的指针
*tc_stack_end	指向任务堆栈的结束地址的指针
*tc_stack_pointer	任务堆栈指针
tc_stack_size	任务堆栈大小
tc_stack_minimum	任务最小允许堆栈大小
*tc_current_protect	指向任务当前保护结构的指针
*tc_saved_stack_ptr	任务上次堆栈指针
tc_time_slice	任务时间片
*tc_ready_previous	指向前一个就绪任务TCB 的指针
*tc_ready_next	指向在就绪链表后一个就绪任务TCB 的指针
tc_priority_group	优先级组的当前掩码
**tc_priority_head	指向优先级链表头指针的指针
*tc_sub_prioirty_pt	指向优先级子组的指针
tc_sub_priority	优先级子组的当前掩码
tc_saved_status	前一任务的状态
tc_signal_active	指明信号是否激活的标记
tc_padding	This is used to align the task structure on an even boundary. In some ports this field is not used.
(*tc_entry)(UNSIGNED,VOID *)	任务入口函数
tc_argc	传递给任务的参数
*tc_argv	传递给任务的参数
(*tc_cleanup)(VOID *)	任务清除函数
*tc_cleanup_info	指向任务清除信息的指针
*tc_suspend_protect	指向在任务挂起时保护结构的指针
tc_timer_active	指示定时是否激活的标识
tc_timer_control	定时控制块
tc_signals	指示当前信号
tc_enabled_signals	指示是否允许信号
(*tc_signal_handler) (UNSIGNED)	信号处理函数
tc_system_reserved_1	系统保留字
tc_system_reserved_2	系统保留字
tc_system_reserved_3	系统保留字
tc_app_reserved_1	应用程序保留字

HISR控制块 HISR Control Block

```
typedef struct TC_HCB_STRUCT
{
    /* Standard thread information first.  This information is used by
       the target dependent portion of this component.  Changes made
       to this area of the structure can have undesirable side effects.  */

    CS_NODE             tc_created;           /* Node for linking to    */
                                              /*   created task list    */
    UNSIGNED            tc_id;                /* Internal TCB ID        */
    CHAR                tc_name[NU_MAX_NAME]; /* HISR name              */
    DATA_ELEMENT        tc_not_used_1;        /* Not used field         */
    DATA_ELEMENT        tc_not_used_2;        /* Not used field         */
    DATA_ELEMENT        tc_priority;          /* HISR priority          */
    DATA_ELEMENT        tc_not_used_3;        /* Not used field         */
    UNSIGNED            tc_scheduled;         /* HISR scheduled count   */
    UNSIGNED            tc_cur_time_slice;    /* Not used in HISR       */
    VOID               *tc_stack_start;       /* Stack starting address */
    VOID               *tc_stack_end;         /* Stack ending address   */
    VOID               *tc_stack_pointer;     /* HISR stack pointer     */
    UNSIGNED            tc_stack_size;        /* HISR stack's size      */
    UNSIGNED            tc_stack_minimum;     /* Minimum stack size     */
    struct TC_PROTECT_STRUCT
                       *tc_current_protect;   /* Current protection     */
    struct TC_HCB_STRUCT
                       *tc_active_next;       /* Next activated HISR    */

    /* Information after this point is not used in the target dependent
       portion of this component.  Hence, changes in the following section
       should not impact assembly language routines.  */
    UNSIGNED            tc_activation_count;  /* Activation counter     */
    VOID                (*tc_entry)(VOID);    /* HISR entry function    */

#ifdef  MNT
    VOID                (*tc_actual_entry)();   /* HISR entry function  MNT 1.2 */
#endif

    /* Reserved words for the system and a single reserved word for the
       application.  */
    UNSIGNED            tc_system_reserved_1; /* System reserved word   */
    UNSIGNED            tc_system_reserved_2; /* System reserved word   */
    UNSIGNED            tc_system_reserved_3; /* System reserved word   */
    UNSIGNED            tc_app_reserved_1;    /* Application reserved   */

} TC_HCB;
```

HISR控制块 TC_HCB包括HISR的优先级和其他任务控制处理必须的数据成员。

数据成员描述

CS_NODE tc_created

```
UNSIGNED tc_id
CHAR tc_name[NU_MAX_NAME]
DATA_ELEMENT tc_not_used_1
DATA_ELEMENT tc_not_used_2
DATA_ELEMENT tc_priority
DATA_ELEMENT tc_not_used_3
UNSIGNED tc_scheduled
UNSIGNED tc_cur_time_slice
VOID *tc_stack_start
VOID *tc_stack_end
VOID *tc_stack_pointer
UNSIGNED tc_stack_size
UNSIGNED tc_stack_minimum
struct TC_PROTECT_STRUCT *tc_current_protect
struct TC_HCB_STRUCT *tc_active_next
UNSIGNED tc_activation_count
VOID (*tc_entry)(VOID)
UNSIGNED tc_system_reserved_1
UNSIGNED tc_system_reserved_2
UNSIGNED tc_system_reserved_3
UNSIGNED tc_app_reserved_1
```

数据成员概要

数据成员	概要
tc_created	为连接到已创建HISR双向循环链表的节点
tc_id	内部的任务标识
tc_name	用户定义的8个字符的任务名字
tc_not_used_1	该数据成员放置在此但不使用
tc_not_used_2	该数据成员放置在此但不使用
tc_priority	HISR优先级
tc_not_used_3	该数据成员放置在此但不使用
tc_scheduled	指明HISR调度的个数
tc_cur_time_slice	指明当前HISR的时间片
*tc_stack_start	指向HISR堆栈的开始地址的指针
*tc_stack_end	指向HISR堆栈的结束地址的指针
*tc_stack_pointer	HISR堆栈指针
tc_stack_size	HISR堆栈大小
tc_stack_minimum	HISR最小允许堆栈大小
*tc_current_protect	指向HISR当前保护结构的指针
*tc_active_next	HISR上次堆栈指针
tc_activation_count	HISR激活的次数
(*tc_entry)(VOID)	HISR入口函数
tc_system_reserved_1	系统保留字
tc_system_reserved_2	系统保留字
tc_system_reserved_3	系统保留字
tc_app_reserved_1	应用程序保留字

保护块 Protection Block

```
typedef struct TC_PROTECT_STRUCT
{
```

```
    TC_TCB              *tc_tcb_pointer;        /* Owner of the protection */
    UNSIGNED             tc_thread_waiting;     /* Waiting thread flag     */
} TC_PROTECT;
```

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护系统数据结构的完整性。 这种保护归功于使用名为TC_Protect t的内在的保护结构。所以的系统的数据结构创建和删除都在TC_Protect的保护下。

数据成员描述

TC_TCB *tc_tcb_pointer

UNSIGNED tc_thread_waiting

数据成员概要

数据成员	描述
*tc_tcb_pointer	当前被保护线程标识
tc_thread_waiting	指明一个或多个线程在等待保护

线程控制组件函数 Thread Control Functions

下面这部分提供概要的线程控制(TC)函数描述。回顾实际的源码可以获得更多的信息。

- **TCC_Create_Task**

该函数创建一个任务然后将其放入已创建任务链表中。所有的创建任务必需的资源需要提供给该函数。如果特殊设置，这个新创建的任务可以开始处于就绪状态，否则，它将处于挂起状态。

语法 Syntax

```
STATUS TCC_Create_Task (NU_TASK *task_ptr, CHAR *name, VOID
(*task_entry)(UNSIGNED, VOID *),
UNSIGNED argc, VOID *argv, VOID
*stack_address, UNSIGNED
stack_size, OPTION
priority,UNSIGNED time_slice,
OPTION preempt, OPTION auto_start)
```

函数调用 Functions Called

```
CSC_Place_On_List                    TCT TCT_Control_To_System
[HIC_Make_History_Entry]             TCT_Protect
TCC_Resume_Task                      TCT_Unprotect
TCT_Build_Task_Stack                 TMC_Init_Task_Timer
[TCT_Check_Stack]
```

- **TCC_Delete_Task**

该函数删除一个任务然后将其移出已创建任务链表中。该函数假定要删除的任务处于完成或终止状态。注意该函数并不释放任务控制块和堆栈所占内存，这是用户应用程序的职责。

语法 Syntax

```
STATUS TCC_Delete_Task(NU_TASK *task_ptr)
```

函数调用 Functions Called

```
CSC_Remove_From_List
```

[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Create_HISR**

该函数创建一个HISR 然后将其放入已创建HISR链表中。所有的创建HISR必需的资源需要提供给该函数。HISR 创建后总处于睡眠状态。

语法 Syntax

```
STATUS TCC_Create_HISR (NU_HISR *hisr_ptr, CHAR *name, VOID
(*hisr_entry)(VOID), OPTION priority,
VOID *stack_address,UNSIGNED
stack_size)
```

函数调用 Functions Called

CSC_Place_On_List
[HIC_Make_History_Entry]
TCT_Build_HISR_Stack
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Delete_HISR**

该函数删除一个HISR然后将其移出已创建HISR链表中。该函数假定要删除的任务处于非激活状态。注意该函数并不释放HISR控制块和堆栈所占内存，这是用户应用程序的职责。

语法 Syntax

```
STATUS TCC_Delete_HISR(NU_HISR *hisr_ptr)
```

函数调用 Functions Called

CSC_Remove_From_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Reset_Task**

该函数复位一个任务，注意任务复位必须在任务完成和终止状态才能被执行。此时该任务无条件处于挂起状态。

语法 Syntax

```
STATUS TCC_Reset_Task(NU_TASK *task_ptr, UNSIGNED argc,VOID *argv)
```

函数调用 Functions Called

[HIC_Make_History_Entry]
TCT_Build_Task_Stack
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Terminate_Task**

该函数终止一个任务。如果当前被挂起的任务被终止，该任务的清除函数被调用去清除挂起的数据结构。

语法 Syntax

```
STATUS TCC_Terminate_Task(NU_TASK *task_ptr)
函数调用 Functions Called
Cleanup routine
[HIC_Make_History_Entry]
TCC_Suspend_Task
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
TCT_Unprotect_Specific
TMC_Stop_Task_Timer
```

- **TCC_Resume_Task**

该函数继续先前的被挂起的任务。该任务必须处于这个函数要求的挂起状态。如果继续的任务的优先级比调用TCC_Resume_Task的任务优先级高并且当前任务是可抢占的，那么该函数返回值NU_TRUE。如果不允许抢占，那么返回值NU_FALSE。

```
语法 Syntax
STATUS TCC_Resume_Task(NU_TASK *task_ptr,
OPTION suspend_type)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Set_Current_Protect
TCT_Set_Execute_Task
TMC_Stop_Task_Timer
```

- **TCC_Resume_Service**

该函数提供应用程序调用的继续任务的系统服务。（译注：该函数调用TCC_Resume_Task）

```
语法 Syntax
STATUS TCC_Resume_Service(NU_TASK *task_ptr)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Unprotect
```

- **TCC_Suspend_Task**

该函数挂起特定的任务。如果这个特定的任务被自己调用挂起，则控制权交还给操作系统。

```
语法 Syntax
VOID TCC_Suspend_Task(NU_TASK *task_ptr, OPTION suspend_type,
VOID (*cleanup) (VOID*),VOID*information,
UNSIGNED timeout)
函数调用 Functions Called
HIC_Make_History_Entry
TCT_Control_To_System
TCT_Protect
TCT_Set_Execute_Task
TCT_Protect_Switch
TCT_Unprotect
```

TMC_Start_Task_Timer

- **TCC_Suspend_Service**

该函数提供应用程序调用的挂起任务的系统服务。（译注：该函数调用TCC_Suspend_Task）

语法 Syntax
STATUS TCC_Suspend_Service(NU_TASK *task_ptr)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Suspend_Task
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Task_Timeout**

该函数处理任务挂起超时的情况。注意任务sleep也被考虑为一种超时情况。

语法 Syntax
VOID TCC_Task_Timeout(NU_TASK *task_ptr)
函数调用 Functions Called
Caller’s cleanup function
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Protect
TCT_Set_Current_Protect
TCT_Unprotect
TCT_Unprotect_Specific

- **TCC_Task_Sleep**

该函数任务sleep挂起功能。它的注意用途是提供当前任务挂起的接口。

语法 Syntax
VOID TCC_Task_Sleep(UNSIGNED ticks)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Suspend_Task
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCC_Relinquish**

该函数将被调用的任务移到相同优先级任务最后。被调用的任务不会再次执行，除非其他相同优先级的任务都得到运行的机会后。

语法 Syntax
VOID TCC_Relinquish(VOID)
函数调用 Functions Called
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Control_To_System

TCT_Protect
TCT_Set_Execute_Task
TCT_Unprotect

- **TCC_Time_Slice**

该函数将特定的任务移到相同优先级任务最后。如果特定任务不处于就绪状态，该操作被忽略。

语法 Syntax
VOID TCC_Time_Slice(NU_TASK *task_ptr)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Set_Execute_Task
TCT_Unprotect

- **TCC_Current_Task_Pointer**

该函数返回当前正在执行任务的指针。如果当前执行的线程不是一个任务线程，那么返回值NU_NULL 。

语法 Syntax
NU_TASK *TCC_Current_Task_Pointer(VOID)
函数调用 Functions Called
None

- **TCC_Current_HISR_Pointer**

该函数返回当前正在执行HISR的指针。如果当前执行的线程不是一个HISR线程，那么返回值NU_NULL 。

语法 Syntax
NU_HISR *TCC_Current_HISR_Pointer(VOID)
函数调用 Functions Called
None

- **TCC_Task_Shell**

该函数是所有应用程序任务开始执行时的外壳函数。该外壳函数使控制权从应用程序任务返回时将任务终止（译注：挂起）。该外壳函数也传递参数argc 和 argv 给任务入口函数。

语法 Syntax
VOID TCC_Task_Shell(VOID)
函数调用 Functions Called
Task Entry Function
TCC_Suspend_Task
TCT_Protect

- **TCC_Signal_Shell**

该函数通过调用任务自己的信号处理函数进行信号处理，当信号处理完成后，该任务被放置在适当的状态。

语法 Syntax
VOID TCC_Signal_Shell(VOID)
函数调用 Functions Called
task’s signal handling routine

[TCT_Check_Stack]
TCT_Signal_Exit
TCT_Protect
TCT_Set_Execute_Task
TCT_Unprotect

- ## TCC_Dispatch_LISR

该函数通过中断向量表分发LISR。注意该函数是在中断期间调用。

语法 Syntax
VOID TCC_Dispatch_LISR(INT vector)
函数调用 Functions Called
application LISR
ERC_System_Error

- ## TCC_Register_LISR

该函数用参数提供的向量值注册LISR。如果提供的LISR值为**NU_NULL**， 则提供的向量为未注册的。先前的注册过的LISR被返回给调用者，并且返回完成状态。

语法 Syntax
STATUS TCC_Register_LISR(INT vector, VOID(*new_lisr)(INT),
VOID (**old_lisr)(INT))
函数调用 Functions Called
[HIC_Make_History_Entry]
INT_Retrieve_Shell
INT_Setup_Vector
INT_Vectors_Loaded
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- ## TCCE_Create_Task

该函数提供参数错误检查的创建任务功能。

语法 Syntax
STATUS TCCE_Create_Task(NU_TASK *task_ptr, CHAR *name, VOID
(*task_entry)(UNSIGNED,VOID*),
UNSIGNED argc, VOID *argv, VOID
*stack_address, UNSIGNED stack_size,
OPTION priority, UNSIGNED time_slice,
OPTION preempt, OPTION auto_start)
函数调用 Functions Called
TCC_Create_Task

- ## TCCE_Create_HISR

该函数提供参数错误检查的创建HISR功能。

语法 Syntax
STATUS TCCE_Create_HISR(NU_HISR *hisr_ptr, CHAR *name,
VOID (*hisr_entry)(VOID), OPTION priority,
VOID *stack_address, UNSIGNED stack_size)

函数调用 Functions Called
TCC_Create_HISR

- **TCCE_Delete_HISR**
  该函数提供参数错误检查的删除HISR功能。

  语法 Syntax
  STATUS TCCE_Delete_HISR (NU_HISR *hisr_ptr)
  函数调用 Functions Called
  TCC_Delete_HISR

- **TCCE_Delete_Task**
  该函数提供参数错误检查的删除任务功能。
  .
  语法 Syntax
  STATUS TCCE_Delete_Task(NU_TASK *task_ptr)
  函数调用 Functions Called
  TCC_Delete_Task

- **TCCE_Reset_Task**
  该函数提供参数错误检查的复位任务功能。
  .
  语法 Syntax
  STATUS TCCE_Reset_Task(NU_TASK* task_ptr,UNSIGNED argc,
  VOID *argv)
  函数调用 Functions Called
  TCC_Reset_Task

- **TCCE_Terminate_Task**
  该函数提供参数错误检查的终止任务功能。

  语法 Syntax
  STATUS TCCE_Terminate_Task(NU_TASK *task_ptr)
  函数调用 Functions Called
  TCC_Terminate_Task

- **TCCE_Resume_Service**
  该函数提供参数错误检查的继续任务功能。

  语法 Syntax
  STATUS TCCE_Resume_Service(NU_TASK *task_ptr)
  函数调用 Functions Called
  TCCE_Validate_Resume
  TCC_Resume_Service

- **TCCE_Suspend_Service**
  该函数提供参数错误检查的挂起任务功能。

  语法 Syntax
  STATUS TCCE_Suspend_Service(NU_TASK *task_ptr)
  函数调用 Functions Called

TCC_Suspend_Service

- **TCCE_Relinquish**

  该函数提供参数错误检查的放弃任务功能。如果当前线程不是一个任务，该操作被忽略。

  语法 Syntax
  VOID TCCE_Relinquish(VOID)
  函数调用 Functions Called
  TCC_Relinquish

- **TCCE_Task_Sleep**

  该函数提供参数错误检查的睡眠任务功能。如果当前线程不是一个任务，该操作被忽略。

  语法 Syntax
  VOID TCCE_Task_Sleep(UNSIGNED ticks)
  函数调用 Functions Called
  TCC_Task_Sleep

- **TCCE_Suspend_Error**

  该函数检查一个挂起操作错误。挂起操作仅仅被允许在任务线程中被执行，任何从其它线程发起的挂起请求都是错误的。

  语法 Syntax
  INT TCCE_Suspend_Error(VOID)
  函数调用 Functions Called
  None

- **TCCE_Activate_HISR**

  该函数提供参数错误检查的激活HISR功能。

  语法 Syntax
  STATUS TCCE_Activate_HISR(NU_HISR *hisr_ptr)
  函数调用 Functions Called
  TCT_Activate_HISR

- **TCCE_Validate_Resume**

  该函数证实挂起的服务和挂起的设备
  This function validates the resume service and resume driver calls with scheduling protection around the examination of the task status.
  语法 Syntax
  STATUS TCCE_Validate_Resume(OPTION resume_type,NU_TASK *task_ptr)
  函数调用 Functions Called
  TCT_Set_Current_Protect
  TCT_System_Protect
  TCT_System_Unprotect
  TCT_Unprotect

- **TCF_Established_Tasks**

返回当前确定任务的个数。先前被删除的任务不再被认为是确定的任务。

语法 Syntax
UNSIGNED TCF_Established_Tasks(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

- **TCF_Established_HISRs**

返回当前确定HISR的个数。先前被删除的HISR不再被认为是确定的任务。

语法 Syntax
UNSIGNED TCF_Established_HISRs(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

- **TCF_Task_Pointers**

创建一个任务指针链表，该链表从一个特定的位置开始。该链表中的任务指针个数等于所有的任务个数或者等于调用时传递来的一个参数（最大指针个数）。

语法 Syntax
UNSIGNED TCF_Task_Pointers(NU_TASK **pointer_list,
UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

- **TCF_HISR_Pointers**

创建一个HISR指针链表，该链表从一个特定的位置开始。该链表中的HISR指针个数等于所有的HISR个数或者等于调用时传递来的一个参数（最大指针个数）。

语法 Syntax
UNSIGNED TCF_HISR_Pointers(NU_HISR **pointer_list,
UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCF_Task_Information**

返回一个特定任务的信息。然而，如果提供的任务指针不是有效的，该函数简单地返回一个错误信息。

语法 Syntax
STATUS TCF_Task_Information(NU_TASK *task_ptr, CHAR *name,
DATA_ELEMENT *status, UNSIGNED
*scheduled_count, DATA_ELEMENT
*priority, OPTION *preempt,
UNSIGNED *time_slice, VOID
**stack_base, UNSIGNED
*stack_size, UNSIGNED
*minimum_stack)

函数调用 Functions Called
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

- **TCF_HISR_Information**

返回一个特定HISR的信息。然而，如果提供的HISR指针不是有效的，该函数简单地返回一个错误信息。

语法 Syntax
STATUS TCF_HISR_Information(NU_HISR *hisr_ptr, CHAR *name,
UNSIGNED *scheduled_count,
DATA_ELEMENT *priority, VOID
**stack_base, UNSIGNED
*stack_size, UNSIGNED
minimum_stack)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

- **TCI_Initialize**

该函数提供初始TC（线程控制）组件的功能。系统初始为空闲状态。

语法 Syntax
VOID TCI_Initialize(VOID)
函数调用 Functions Called
None

- **TCS_Change_Priority**

该函数改变特定任务的优先级。挂起或就绪状态的任务优先级可以被改变。如果新的优先级需要上下文切换，控制权被返还给操作系统。

语法 Syntax
OPTION TCS_Change_Priority(NU_TASK *task_ptr, OPTION
new_priority)
函数调用 Functions Called
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Set_Execute_Task
TCT_Unprotect

- **TCS_Change_Preemption**

该函数改变特定任务的是否可被抢占。如果不允许抢占，任务运行直到挂起或放弃。如果允许抢占，则导致上下文切换。

语法 Syntax
OPTION TCS_Change_Preemption(OPTION preempt)
函数调用 Functions Called
[HIC_Make_History_Entry]

[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Set_Execute_Task
TCT_Unprotect

- **TCS_Change_Time_Slice**

该函数改变特定任务的时间片，时间片为0意味着禁止时间片调度。

语法 Syntax
UNSIGNED TCS_Change_Time_Slice(NU_TASK *task_ptr,
UNSIGNED time_slice)
函数调用 Functions Called
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCS_Control_Signals**

该函数指定特定的允许信号量并且返回原来的允许信号量给调用者。如果新的允许信号量发生并且该信号量处理函数被注册，则信号量处理函数开始执行。

语法 Syntax
UNSIGNED TCS_Control_Signals(UNSIGNED enable_signal_mask)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Signal_Shell
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCS_Receive_Signals**

该函数返回信号给它的调用者。注意信号是自动被清除。

语法 Syntax
UNSIGNED TCS_Receive_Signals(VOID)
函数调用 Functions Called
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCS_Register_Signal_Handler**

该函数为调用它的任务注册一个信号处理函数。注意：如果允许信号当前发生，并且它是首次被注册，那么信号立刻被处理。

语法 Syntax
STATUS TCS_Register_Signal_Handler(VOID (*signal_handler)
(UNSIGNED))

函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Signal_Shell
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

- **TCS_Send_Signals**

该函数发送特定信号给特定任务。如果信号允许，特定任务会去处理该信号。

语法 Syntax
STATUS TCS_Send_Signals(NU_TASK *task_ptr, UNSIGNED signals)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Signal_Shell
TCT_Build_Signal_Frame
[TCT_Check_Stack]

- **TCSE_Change_Priority**

该函数提供参数错误检查的改变优先级功能。如果错误出现，该函数忽略错误，并返回被请求的优先级。

语法 Syntax
OPTION TCSE_Change_Priority(NU_TASK *task_ptr,OPTION new_priority)
函数调用 Functions Called
TCS_Change_Priority

- **TCSE_Change_Preemption**

该函数提供参数错误检查的改变抢占功能。如果当前线程不是一个任务线程，该操作被忽略。

语法 Syntax
OPTION TCSE_Change_Preemption(OPTION preempt)
函数调用 Functions Called
TCS_Change_Preemption

- **TCSE_Change_Time_Slice**

该函数提供参数错误检查的改变时间片功能。如果要改变时间片的任务不是一个有效的任务，该操作被忽略。

语法 Syntax
UNSIGNED TCSE_Change_Time_Slice(NU_TASK *task_ptr,UNSIGNED time_slice)
函数调用 Functions Called
TCS_Change_Time_Slice

- **TCSE_Control_Signals**

该函数检查调用者是否为一个无任务的线程生成，如果这样，那么该调用请求被简单地忽略。

语法 Syntax
UNSIGNED TCSE_Control_Signals(UNSIGNED enable_signal_mask)

函数调用 Functions Called
TCS_Control_Signals

- **TCSE_Receive_Signals**
  该函数决定调用者是否为一个任务的线程生成，如果不是这样，那么该调用请求被简单地忽略
  语法 Syntax
  UNSIGNED TCSE_Receive_Signals(VOID)
  函数调用 Functions Called
  TCS_Receive_Signals

- **TCSE_Register_Signal_Handler**
  该函数决定调用者是否为一个任务，如果调用者不是一个任务并且/或者挂起的信号处理指针为NULL,那么适当的错误状态被返回。
  语法 Syntax
  STATUS TCSE_Register_Signal_Handler(VOID (*signal_handler)(UNSIGNED))
  函数调用 Functions Called
  TCS_Register_Signal_Handler

- **TCSE_Send_Signals**
  该函数检查任务是否有效，如果任务不是有效的，则返回一个错误。

  语法 Syntax
  STATUS TCSE_Send_Signals(NU_TASK *task_ptr, UNSIGNED signals)
  函数调用 Functions Called
  TCS_Send_Signals

- **TCT_Control_Interrupts**
  这是一个汇编的函数，通过调用该函数允许或禁止特定的中断。中断如果被该函数禁止了，那么除非调用该函数允许该中断，否则该中断总处于禁止状态。

  语法 Syntax
  INT TCT_Control_Interrupts(new_level)
  函数调用 Functions Called
  None

- **TCT_Local_Control_Interrupts**
  这是一个汇编的函数，通过调用该函数允许或禁止特定的中断.

  语法 Syntax
  INT TCT_Local_Control_Interrupts(new_level)
  函数调用 Functions Called
  None

- **TCT_Restore_Interrupts**
  这是一个汇编的函数，通过调用该函数恢复特定的中断到全局变量TCD_Interrupt_Level。

  语法 Syntax
  VOID TCT_Restore_Interrupts(VOID)
  函数调用 Functions Called
  None

- **TCT_Build_Task_Stack**
  这是一个汇编的函数，该函数为一个特定任务创建初始划堆栈。这个初始化的堆栈包括初始化的寄存器的

值和任务入口指针。而且，这个初始化的堆栈和中断堆栈具有相同的形式。

语法 Syntax
VOID TCT_Build_Task_Stack(TC_TCB *task)
函数调用 Functions Called
None

- **TCT_Build_HISR_Stack**
  这是一个汇编的函数，该函数为一个特定HISR创建初始划堆栈。

语法 Syntax
VOID TCT_Build_HISR_Stack(TC_HCB *hisr)
函数调用 Functions Called
None

- **TCT_Build_Signal_Frame**
  这是一个汇编的函数，该函数在任务的堆栈的顶端创建一个帧。这导致在下一次任务被执行的时候任务的信号处理函数被执行。

语法 Syntax
VOID TCT_Build_Signal_Frame(TC_TCB *task)
函数调用 Functions Called
None

- **TCT_Check_Stack**
  这是一个汇编的函数，该函数检查当前堆栈的溢出状态。此外，该函数为调用线程和返回当前有效的堆栈空间保持堆栈空间最小数目的路径。

语法 Syntax
UNSIGNED TCT_Check_Stack(void)
函数调用 Functions Called
ERC_System_Error

- **TCT_Schedule**
  这是一个汇编的函数，该函数等待一个变成就绪的线程。当一个线程就绪时，该函数就开始将控制权转移给该线程。

语法 Syntax
VOID TCT_Schedule(void)
函数调用 Functions Called
TCT_Control_To_Thread

- **TCT_Control_To_Thread**
  这是一个汇编的函数，该函数传递控制权到特定的线程。每一个时间控制被传送给一个线程，线程的调度数目被递增。另外任务线程的time-slicing 也被该函数激活。该函数设置TCD_Current_Thread指针。

语法 Syntax
VOID TCT_Control_To_Thread(TC_TCB *thread)
函数调用 Functions Called
None

- **TCT_Control_To_System**
  这是一个汇编的函数，该函数从一个线程传递控制权到系统。

Note that this service is called in a solicited manner, i.e., it is not called from an interrupt thread. Registers required by the compiler to be preserved across function boundaries are saved by this routine. Note that this is usually a subset of the total number of available registers.

语法 Syntax

VOID TCT_Control_To_System(void)

函数调用 Functions Called

TCT_Schedule

- **TCT_Signal_Exit**

这是一个汇编的函数，该函数从一个信号处理函数中退出。该函数的主要目的是清除调度保护和交换堆栈指针到正常的任务的堆栈指针.

语法 Syntax

VOID TCT_Control_To_System(void)

函数调用 Functions Called

TCT_Schedule

- **TCT_Current_Thread**

这是一个汇编的函数，该函数返回当前线程的指针。

语法 Syntax

VOID *TCT_Current_Thread(void)

函数调用 Functions Called

None

- **TCT_Set_Execute_Task**

这是一个汇编的函数，该函数设置一个任务为保护下的执行任务。这个过程禁止中断。

语法 Syntax

VOID TCT_Set_Execute_Task(TC_TCB *task)

函数调用 Functions Called

None

- **TCT_Protect**

This assembly language function protects against multiple thread access. If another thread (TASK or HISR) owns the requested protection structure, then that thread will be scheduled to run until it releases the protection structure. At that time, the thread is suspended, and control is returned to the thread doing the `TCT_Protect` call. This prevents lower priority tasks from blocking higher priority threads trying to obtain a protection structure.

语法 Syntax

VOID TCT_Protect(TC_PROTECT *protect)

函数调用 Functions Called

None

- **TCT_Unprotect**

这是一个汇编的函数，该函数释放当前活动的线程的保护。如果调用者不是一个活动的线程，那么该函数请求被忽略。

语法 Syntax

VOID TCT_Unprotect(void)

函数调用 Functions Called

None

- **TCT_Unprotect_Specific**

  这是一个汇编的函数，该函数释放一个特定的保护结构。

  语法 Syntax
  VOID TCT_Unprotect_Specific(TC_PROTECT *protect)
  函数调用 Functions Called
  None

- **TCT_Set_Current_Protect**

  这是汇编函数，它设置当前任务控制块的保护数据成员。
  .
  语法 Syntax
  VOID TCT_Set_Current_Protect(TC_PROTECT *protect)
  函数调用 Functions Called
  None

- **TCT_Protect_Switch**

  这是一个汇编的函数，该函数在一个特定任务不再占有任何和其相关的保护资源之前保存等待， 这对于任务不能被挂起或者终止除非任务释放所有的被保护的资源是必须。
  语法 Syntax
  VOID TCT_Protect_Switch(VOID *thread)
  函数调用 Functions Called
  None

- **TCT_Schedule_Protected**

  这是一个汇编的函数，该函数保存一个线程最小的上下文，然后大直接调度其他的线程，被调度的线程是在调用函数的线程的保护下。
  语法 Syntax
  VOID TCT_Schedule_Protected(VOID *thread)
  函数调用 Functions Called
  TCT_Control_To_Thread
  TCT_Interrupt_Context_Save
  这是一个汇编的函数，该函数保存中断线程的上下文。嵌套的中断也被支持。如果一个任务或HISR 被中断，在上下文被保存后，堆栈指针切换到系统堆栈指针。

  语法 Syntax
  VOID TCT_Interrupt_Context_Save(vector)
  函数调用 Functions Called
  None

- **TCT_Interrupt_Context_Restore**

  这是一个汇编的函数，如果是嵌套中断，该函数恢复中断的上下文。否则，该函数将控制权交给调度（scheduling）函数。

  语法 Syntax
  VOID TCT_Interrupt_Context_Restore(void)
  函数调用 Functions Called
  TCT_Schedule

- **TCT_Activate_HISR**

这是一个汇编的函数，该函数激活特定的HISR。如果这个HISR已经被激活，那么简单地将HISR的激活次数递增一个。否则，这个HISR将被放置在适当的HISR 优先级链表中准备被执行。

语法 Syntax
STATUS TCT_Activate_HISR(TC_HCB *hisr)
函数调用 Functions Called
None

- **TCT_HISR_Shell**

这是一个汇编的函数，该函数是每一个HISR的外壳函数。如果HISR已经完成处理，这个外壳函数将返回给系统。否则，它继续调用HISR函数直到HISR的激活次数为零。

语法 Syntax
VOID TCT_HISR_Shell(void)
函数调用 Functions Called
hisr -> tc_entry

- **TCT_Check_For_Preemption**

这是一个汇编的函数,该函数检查当一个最小的ISR被处理时是否有其他的中断发生。如果有其他中断发生，一个完整的上下文被保存并且恢复是被执行用来处理优先级。如果没有其他中断发生，那么控制权被返回给中断点。

语法 Syntax
VOID TCT_Check_For_Preemption(void)
函数调用 Functions Called
TCT_Interrupt_Context_Save
TCT_Interrupt_Context_Restore

定时管理组件 Timer Component (TM)

定时管理组件(TM)负责处理所有的Nucleus PLUS 定时设备。基本的定时时间是一个单一的硬件中断来计量的，名为滴答（译注：tick）。Nucleus PLUS的定时器可以被提供给应用级的程序在定时器到期去执行特定的函数。定时器也提供给任务用于任务睡眠和挂起超时。参考《*Nucleus PLUS Reference Manual*》第三章得到更多的信息。

定时管理组件文件 Timer Files

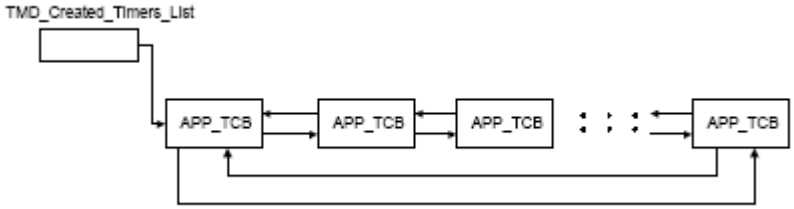
定时管理组件(TM)包括九个文件。每一个文如下描述：

文件 File	描述 Description
TM_DEFS.H	该文件定义TM组件特定的数据结构和常量。
TM_EXTR.H	该文件定义TM组件的外部接口。
TMD.C	该文件定义TM组件的全局数据结构。
TMI.C	该文件包括TM组件的初始化代码。
TMF.C	该文件包括提供TM组件的信息的函数。
TMC.C	该文件包括提供TM组件的核心函数，该文件定义的函数实现基本的开始和停止一个定时器。
TMS.C	该文件包括提供TM组件的附加函数，该文件定义的函数往往没有定义在TMC.C中的函数使用频繁。
TMSE.C	该文件包括有错误检查的定义在TMC.C中的函数接口。
TMT.[S,ASM,SRC]	该文件包括所有的和目标板相关的函数。

定时管理组件数据结构 Timer Data Structures

已创建定时器链表 Created Timers List

Nucleus PLUS 应用程序的定时器可被动态的创建和删除。每一个已创建定时器的定时器控制块是被保存在一个双向的循环链表中。最新创建的定时器是被放置在链表的尾部，而删除一个定时器是完整地从链表中删除。这个链表地头指针是TMD_Created_Timers_List. 这个已创建定时器链表是专用于应用程序地定时器的。



已创建定时器链表保护 Created Timer List Protection

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建定时器链表的完整性。 这种保护归功于使用名为TMD_Created_List_Protect的内在的保护结构。所以的定时器的创建和删除都在TMD_Created_List_Protect的保护下。

成员变量的声明

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

成员变量的概要

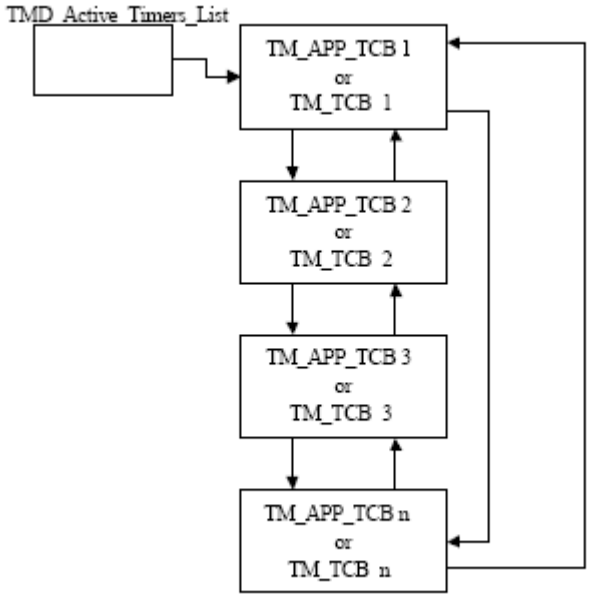
成员变量	描述
tc_tcb_pointer	Identifies the thread that currently has the protection.
tc_thread_waiting	Identifies the thread that currently has the protection.

定时器总数 Total Timers

当前已创建 Nucleus PLUS 定时器的个数是在变量TMD_Total_Timers中指示。这个变量的内容是已创建定时器链表中的TCBs的个数。对这个变量的操作都是在TMD_Created_List_Protect的保护下进行的。

激活定时器链表 Active Timers List

Nucleus PLUS 激活的定时器是在一个双向的循环链表中维护的。TMD_Active_Timers_List 是该链表的头指针。如果该指针为NULL，表示没有定时器被激活。这个定时器链表支持应用定时器和任务定时器。任务定时器结构在任务的TCB中反映。定时器链表被维护用来表示期满的时间。剩下的时间不是抽象的时间，而是距离期满还剩下的时间。这样做是为了避免在每一个定时器中断时都要调整全部的链表。一个剩余时间为0的定时器被认为是到期的。



激活定时器链表繁忙标记 Active List Busy

Nucleus PLUS 从完成的任务和/或者HISR保护激活定时器链表的完整性。这是通过一个叫做 `TMD_Active_List_Busy` 的标记完成的。所有激活定时器链表的添加和删除操作都在 `TMD_Active_List_Busy` 的保护下。

系统时钟 System Clock

Nucleus PLUS 维护一个不断递增的指示系统时钟的变量，该变量叫做`TMD_System_Clock`。每一个定时中断导致该变量增一。

开始定时器 Timer Start

Nucleus PLUS 存储最后的定时器请求值到一个变量中，该变量叫做TMD_Timer_Start。

定时器 Timer

变量TMD_Timer是一个递减的定时器，它用于描述系统中一个最小的激活定时器的值，当一个定时器到期时，该变量的值为0。

定时器状态 Timer State

TMD_Timer_State描述定时器变量的状态。如果该状态为激活，则定时器计数递减。如果状态为到期，则定时器HISR和定时器任务被初始化去处理定时器的到期。如果定时器指示的状态为未激活，则定时器的计数被忽略。

时间片 Time Slice

Nucleus PLUS 使用变量TMD_Time_Slice作为一个递减的值用于当前执行任务的时间片。当该值为0时，时间片处理开始。

时间片任务 Time Slice Task

TMD_Time_Slice_Task是一个在任务TCB的用于描述时间片的指针。该指针在时间片定时器期满、定时中断产生时被设置。

时间片状态 Time Slice State

Nucleus PLUS 通过变量TMD_Time_Slice_State描述时间片的状态。如果该状态为激活，则时间片计数递减。如果状态为到期，则定时器HISR被初始化去处理定时器的到期。如果时间片指示的状态为未激活，则时间片的计数被忽略。

高级中断服务 HISR

TMD_HISR是定时器HISR’s控制块。

HISR栈指针 HISR Stack Pointer

TMD_HISR_Stack_Ptr 指向保存定时器HISR的内存区域。注意该指针是被INT_Initialize设置。

HISR堆栈大小 Stack Size

Nucleus PLUS通过TMD_HISR_Stack_Size变量描述分配给定时器HISR栈空间的大小。注意该指针是被INT_Initialize设置。

HISR优先级 HISR Priority

TMD_HISR_Priority描述定时器HISR的优先级。优先级范围从0到2，而优先级0最高。注意该指针也是被INT_Initialize设置。

定时器控制块 Timer Control Block

定时器控制块TM_TCB包括包括剩余时间和其他处理定时器中断的数据成员。

数据成员描述

INT tm_timer_type
UNSIGNED tm_remaining_time
VOID *tm_information
struct TM_TCB_STRUCT *tm_next_timer
struct TM_TCB_STRUCT *tm_previous_timer
Field Summary

数据成员概要

Field	Description
tm_timer_type	Indicates if the timer is for an application or a task.
tm_remaining_time	This stores the amount of time remaining after expiration of the previous timer occurs. The true expiration is the sum of all previous timer’s remaining time on the active list
*tm_information	A pointer to general information about the timer.
*tm_next_timer	A pointer to the next timer in the list.
*tm_previous_timer	A pointer to the previous timer in the list.

应用程序定时器控制块 Application’s Timer Control Block

应用程序定时器控制块的包括一个指向定时器期满处理函数的指针和其他应用程序处理定时器请求必要的其他字段。

数据成员描述

CS_NODE tm_created
UNSIGNED tm_id
CHAR tm_name[NU_MAX_NAME]
VOID (*tm_expiration_routine)(UNSIGNED)

UNSIGNED tm_expiration_id
INT tm_enabled
UNSIGNED tm_expirations
UNSIGNED tm_initial_time
UNSIGNED tm_reschedule_time
TM_TCB tm_actual_timer
Field Summary
数据成员概要

Field	Descripton
tm_created	This is the link node structure for timers. It is linked into the created timers list, which is a doubly-linked, circular list.
tm_id	This holds the internal timer identification of 0x54494D45 which is an equivalent to ASCII TIME.
tm_name	This is the user-specified, 8 character name for the timer.
*tm_expiration_routine	A pointer to the timer expiration function.
tm_expiration_id	This is the name of the expiration.
tm_enabled	A flag that determines if the timer is enabled.
tm_expirations	This stores the number of timer expirations.
tm_initial_time	Stores the initial starting time for the timer.
tm_reschedule_time	Stores the reschedule time for the timer.
tm_actual_timer	The actual timer TCB.

定时管理组件函数 Timer Functions

以下部分概要地描述了定时管理组件(TM)的函数。回顾实际的源码可以获得更多的信息。

TMC_Init_Task_Timer
该函数负责初始化任务定时器。
语法 Syntax
VOID TMC_Init_Task_Timer(TM_TCB *timer, VOID *information)
函数调用 Functions Called
None

TMC_Start_Task_Timer
该函数负责开始一个任务定时器。注意任务控制组件调用该函数时有特定的保护考虑。
语法 Syntax
VOID TMC_Start_Task_Timer(TM_TCB *timer, UNSIGNED time)
函数调用 Functions Called
TMC_Start_Timer

TMC_Stop_Task_Timer
该函数负责停止一个任务定时器。注意任务控制组件调用该函数时有特定的保护考虑。
语法 Syntax
VOID TMC_Stop_Task_Timer(TM_TCB *timer)
函数调用 Functions Called
TMC_Stop_Timer

TMC_Start_Timer
该函数负责开始应用程序和任务的定时器。
语法 Syntax
VOID TMC_Start_Timer(TM_TCB *timer, UNSIGNED time)
函数调用 Functions Called
TMT_Read_Timer
TMT_Adjust_Timer
TMT_Enable_Timer

TMC_Stop_Timer
该函数负责停止应用程序和任务的定时器。
语法 Syntax
VOID TMC_Stop_Timer(TM_TCB *timer)
函数调用 Functions Called
TMT_Disable_Timer

TMC_Timer_HISR
该函数负责处理一个定时器期满时高级中断，如果一个应用定时器期满，该定时器期满函数被调用。如果时间片定时器期满，时间片处理被调用。
语法 Syntax
VOID TMC_Timer_HISR(VOID)
函数调用 Functions Called
TCC_Time_Slice
TMC_Timer_Expiration
TMT_Retrieve_TS_Task

TMC_Timer_Expiration
该函数是负责处理所有的任务期满函数。包括应用程序定时器和被用于sleeping and timeouts的基础任务定时器。
语法 Syntax
VOID TMC_Timer_Expiration(VOID)
函数调用 Functions Called
expiration_function
TCC_Task_Timeout
TCT_System_Protect
TCT_Unprotect

TMF_Established_Timers
该函数返回当前使能的定时器的个数。以前删除的定时器不再被认为使能的。
语法 Syntax
UNSIGNED TMF_Established_Timers(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

TMF_Timer_Pointers
Builds a list of timer pointers, starting at the specified location. The number of timer pointers placed in the list is equivalent to the total number of timers or the maximum number of pointers specified in the call.
语法 Syntax
UNSIGNED TMF_Timer_Pointers(NU_TIMER **pointer_list,
UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect

TCT_Unprotect

TMF_Timer_Information
该函数返回一个特定定时器信息。如果提供的定时器指针是非法的，该函数返回一个错误状态。
语法 Syntax
STATUS TMF_Timer_Information (NU_TIMER *timer_ptr, CHAR *name,
OPTION *enable, UNSIGNE*expirations,
UNSIGNED *id, UNSIGNED*initial_time,
UNSIGNED *reschedule_time)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

TMI_Initialize
该函数初始化控制定时器管理组件操作的数据结构，没有应用程序定时器被初始化。
语法 Syntax
VOID TMI_Initialize(VOID)
函数调用 Functions Called
ERC_System_Error
TCC_Create_HISR
TCCE_Create_HISR

TMS_Create_Timer
该函数创建一个定时器并且将其放置在已创建定时器链表。如果传递使能参数，则创建的定时器是激活的。
语法 Syntax
STATUS TMS_Create_Timer(NU_TIMER *timer_ptr, CHAR *name, VOID
(*expiration_routine)
(UNSIGNED), UNSIGNED id, UNSIGNED
initial_time, UNSIGNED
reschedule_time, OPTION enable)
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
TMS_Control_Timer

TMS_Delete_Timer
该函数删除一个应用程序定时器并且将其从已创建定时器链表中移出。
语法 Syntax
STATUS TMS_Delete_Timer(NU_TIMER *timer_ptr)
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_System_Protect
TCT_Unprotect

TMS_Reset_Timer
该函数复位特定应用程序定时器。注意在该函数调用前，定时器必须处于禁止状态。如果使能参数被使用，定时器在复位后自动处于激活状态。
语法 Syntax

```
STATUS TMS_Reset_Timer (NU_TIMER *timer_ptr, VOID
(*expiration_routine)(UNSIGNED),UNSIGNED
initial_time,UNSIGNED
reschedule_time, OPTION enable)
```

函数调用 Functions Called

```
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect
TMS_Control_Timer
```

TMS_Control_Timer

该函数使能和禁止特定定时器。如果定时器已经在要求的状态，则函数仅仅简单的离开。

语法 Syntax

```
STATUS TMS_Control_Timer(NU_TIMER *app_timer, OPTION enable)
```

函数调用 Functions Called

```
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect
TMC_Start_Timer
TMC_Stop_Timer
```

TMSE_Create_Timer

该函数提供参数错误检查的创建定时器函数。

语法 Syntax

```
STATUS TMSE_Create_Timer (NU_TIMER *timer_ptr, CHAR *name,
VOID (*expiration_routine)
(UNSIGNED), UNSIGNED id, UNSIGNED
initial_time, UNSIGNED
reschedule_time, OPTION enable)
```

函数调用 Functions Called

```
TMS_Create_Timer
```

TMSE_Delete_Timer

该函数提供参数错误检查的删除定时器函数。

语法 Syntax

```
STATUS TMSE_Delete_Timer(NU_TIMER *timer_ptr)
```

函数调用 Functions Called

```
TMS_Delete_Timer
```

TMSE_Reset_Timer

该函数提供参数错误检查的复位定时器函数。

语法 Syntax

```
STATUS TMSE_Reset_Timer (NU_TIMER *timer_ptr, VOID
(*expiration_routine)(UNSIGNED),
UNSIGNED initial_time, UNSIGNED
reschedule_time, OPTION enable)
```

函数调用 Functions Called

```
TMS_Reset_Timer
```

TMSE_Control_Timer

该函数提供参数错误检查的控制定时器函数。

语法 Syntax

```
STATUS TMSE_Control_Timer(NU_TIMER *timer_ptr, OPTION enable)
```

函数调用 Functions Called
TMS_Control_Timer

TMT_Set_Clock
该汇编函数用特定的值设置系统时钟。
语法 Syntax
VOID TMT_Set_Clock(UNSIGNED new_value)
函数调用 Functions Called
None

TMT_Retrieve_Clock
该汇编函数返回当前系统时钟。
语法 Syntax
UNSIGNED TMT_Retrieve_Clock(void)
函数调用 Functions Called
None

TMT_Read_Timer
该汇编函数返回当前倒数的定时器的值。
语法 Syntax
UNSIGNED TMT_Read_Timer(void)
函数调用 Functions Called
None

TMT_Enable_Timer
该汇编函数用特定的值使能当前倒数的定时器。
语法 Syntax
VOID TMT_Enable_Timer(UNSIGNED time)
函数调用 Functions Called
None

TMT_Adjust_Timer
这是一个汇编函数，如果新的值比当前的值小，该函数用新值调整倒数的定时器。
语法 Syntax
VOID TMT_Adjust_Timer(UNSIGNED time)
函数调用 Functions Called
None

TMT_Disable_Timer
这是一个汇编函数，该函数禁止倒数的定时器。
语法 Syntax
VOID TMT_Disable_Timer(void)
函数调用 Functions Called
None

TMT_Retrieve_TS_Task
这是一个汇编函数，该函数返回时间片任务指针。
语法 Syntax
NU_TASK *TMT_Retrieve_TS_Task(VOID)
函数调用 Functions Called
None

TMT_Timer_Interrupt
这是一个汇编函数，该函数处理实际的硬件中断。处理包括更新系统时钟、倒数的定时器和时间片定时器。如果定时器到期，则定时器HISR被激活。

语法 Syntax
VOID TMT_Timer_Interrupt(void)
函数调用 Functions Called
TCT_Activate_HISR
TCT_Interrupt_Context_Save
TCT_Interrupt_Context_Restore

邮箱组件 Mailbox Component (MB)

邮箱组件 (MB)负责处理所有的Nucleus PLUS邮箱设备。一个Nucleus PLUS邮箱是一个低级的任务间通讯机制。每一个邮箱能够持有一个消息。一个邮箱消息包括四个32-bit 的字。任务在空邮箱等待一个消息时可以被挂起。相反，任务当试图发送一个已经包含一个消息的邮箱时，任务可以挂起。邮箱可以被用户动态创建和删除。参考《*Nucleus PLUS Reference Manual*》第三章得到更多关于邮箱的信息。

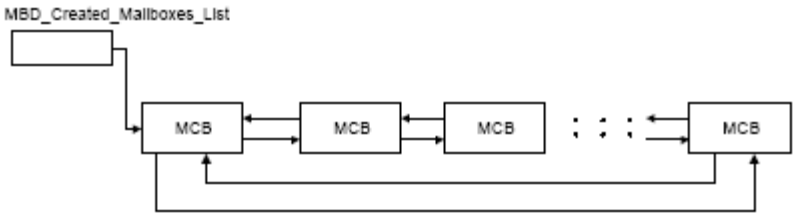
邮箱组件文件 Mailbox Files

邮箱组件包括九个文件，每一个邮箱组件源文件如下定义：

文件 File	描述 Description
MB_DEFS.H	该文件定义MB组件特定的数据结构和常量。
MB_EXTR.H	该文件定义MB组件的外部接口。
MBD.C	文件定义MB组件的全局数据结构。
MBI.C	该文件包括MB组件的初始化代码。
MBF.C	该文件包括提供MB组件的信息的函数。
MBC.C	该文件包括提供MB组件的核心函数，该文件定义的函数实现基本的发送和接收一个邮箱。
MBS.C	该文件包括提供MB组件的附加函数，该文件定义的函数往往没有定义在MBC.C中的函数使用频繁。
MBCE.C	该文件包括有错误检查的定义在MBC.C中的函数接口。
MBSE.C	该文件包括有错误检查的定义在MBS.C中的函数接口。

邮箱组件数据结构 Mailbox Data Structures

已创建邮箱链表 Created Mailbox List
Nucleus PLUS邮箱可以被动态创建和删除。每一个已创建任务地线程控制块(MCB)都被保存在一个双向地循环链表中。新近创建地任务被放置在链表地尾部，而删除一个任务就被完全地从链表中删除。这个链表地头指针是MBD_Created_Mailboxes_List。



已创建邮箱链表保护 Created Mailbox List Protection

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建邮箱链表的完整性。 这种保护归功于使用名为TMD_Created_List_Protect的内在的保护结构。所以的邮箱的创建和删除都在MBD_List_Protect的保护下。

成员变量的声明

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

成员变量的概要

成员变量 Field	描述 Description
tc_tcb_pointer	Identifies the thread that currently has the protection.
tc_thread_waiting	A flag indicating that one or more threads are waiting for the protection.

邮箱总数 Total Mailboxes

当前已创建 Nucleus PLUS 邮箱的个数是在变量MBD_Total_Mailboxes中指示。这个变量的内容是已创建定时器链表中的TCBs的个数。对这个变量的操作都是在MBD_List_Protect的保护下进行的。

邮箱控制块 Mailbox Control Block

邮箱控制块MB_MCB包括邮箱消息区域 (4 32-bit无符合字)和其他处理邮箱必须的成员变量。

数据成员描述

```
CS_NODE mb_created
UNSIGNED mb_id
CHAR mb_name[NU_MAX_NAME]
DATA_ELEMENT mb_message_present
DATA_ELEMENT mb_fifo_suspend
DATA_ELEMENT mb_padding[PAD_2]
UNSIGNED mb_tasks_waiting
UNSIGNED mb_message_area[MB_MESSAGE_SIZE]
struct MB_SUSPEND_STRUCT *mb_suspension_list
```

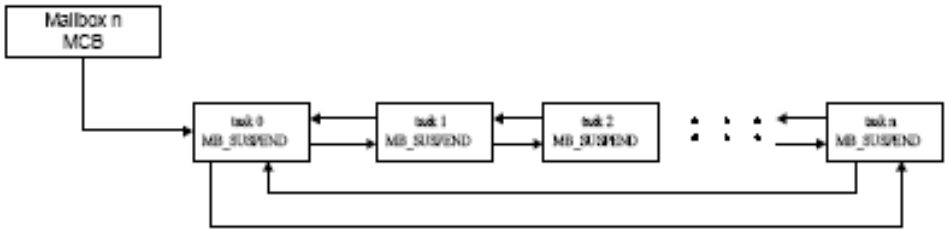
成员变量的概要

成员变量 Field	描述 Description
mb_created	This is the link node structure for mailboxes. It is linked into the created mailbox list, which is a doubly-linked, circular list.
mb_id	This holds the internal mailbox identification of 0x4D424F58 which is an equivalent to ASCII MBOX.
mb_name	This is the user-specified, 8 character name for the mailbox.
mb_message_present	A flag that indicates if a message is present in the mailbox.
mb_fifo_suspend	A flag that determines whether tasks suspend in fifo or priority order.
mb_padding	This is used to align the mailbox structure on an even boundary. In some ports this field is not used.
mb_tasks_waiting	Indicates the number of tasks that are currently suspended on the mailbox.
mb_message_area	The storage area for the message.

*mb_suspension_list	The head of the mailbox suspension list. If no tasks are suspended, this pointer isNULL.

邮箱挂起结构 Mailbox Suspension Structure

邮箱在空和满状态下，任务可以挂起。在挂起过程中，一个MB_SUSPEND 结构被创建。在挂起期间，该结构包括任务信息和任务的邮箱请求信息。该挂起结构被连接到MCB在一个双向的循环链表，该结构是在挂起的任务的堆栈中创建的。每一个挂起在邮箱中的任务都有一个挂起结构。放置在挂起结构链表中的挂起结构的顺序是在邮箱创建时就确定的。如果一个FIFO挂起结构被选择，则该结构被放置在链表的尾部。否则，如果高优先级的挂起结构被选择， 则该结构被放置在优先级更高或相等的任务挂起块后面。



数据成员描述

CS_NODE mb_suspend_link
MB_MCB *mb_mailbox
TC_TCB *mb_suspended_task
UNSIGNED *mb_message_area
STATUS mb_return_status

成员变量的概要

成员变量 Field	描述 Description
mb_suspend_link	A link node structure for linking with other suspended blocks. It is used in a doubly-linked circular suspension list.
*mb_mailbox	A pointer to the mailbox structure.
*mb_suspended_task	A pointer to the Task Control Block of the suspended task.
*mb_message_area	A pointer indicating where the suspended tasks's message buffer is.
mb_return_status	The completion status of the task suspended on the mailbox.

邮箱组件函数 Mailbox Functions

以下部分概要地描述了定时管理组件(MB)的函数。回顾实际的源码可以获得更多的信息。.

MBC_Create_Mailbox

该函数创建一个邮箱然后将其放入已创建邮箱链表中。

语法 Syntax

STATUS MBC_Create_Mailbox (NU_MAILBOX *mailbox_ptr, CHAR *name, OPTION suspend_type)

函数调用 Functions Called

CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]

TCT_Protect
TCT_Unprotect

MBC_Delete_Mailbox
该函数删除一个邮箱然后将其移出已创建邮箱链表中。所有被邮箱挂起的任务将继续。注意该函数并不释放邮箱控制块和堆栈所占内存，这是用户应用程序的职责。
语法 Syntax
STATUS MBC_Delete_Mailbox(NU_MAILBOX *mailbox_ptr)
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System

MBC_Send_To_Mailbox
该函数发送一个4-word的消息到特定的邮箱。如果有一个或更多任务在邮箱中等待一个消息，消息被拷贝到第一个等待任务消息区域并且任务将继续。如果邮箱是满的，调用的任务是可以挂起。
语法 Syntax
STATUS MBC_Send_To_Mailbox(NU_MAILBOX *mailbox_ptr, VOID
*message, UNSIGNED suspend)
函数调用 Functions Called
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

MBC_Receive_From_Mailbox
该函数从特定邮箱接收一个消息。如果当前在邮箱有一个消息，消息被从邮箱中移出并且放置到调用者的参数中。否则，如果在邮箱没有消息准备好，则调用的任务有可能被挂起。
语法 Syntax
STATUS MBC_Receive_From_Mailbox(NU_MAILBOX *mailbox_ptr,
VOID *message, UNSIGNED suspend)
函数调用 Functions Called
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

MBC_Cleanup
该函数负责从一个邮箱移出一个挂起的块。除非任务期满或任务终止，该函数则被调用。注意保护（如同挂起的时间）是有效的。
语法 Syntax
VOID MBC_Cleanup(VOID *information)
函数调用 Functions Called
CSC_Remove_From_List

MBCE_Create_Mailbox
该函数提供参数错误检查的创建邮箱函数。
语法 Syntax
STATUS MBCE_Create_Mailbox (NU_MAILBOX *mailbox_ptr,

CHAR *name, OPTION suspend_type)
函数调用 Functions Called
MBC_Create_Mailbox

MBCE_Delete_Mailbox
该函数提供参数错误检查的删除邮箱函数。.
语法 Syntax
STATUS MBCE_Delete_Mailbox(NU_MAILBOX *mailbox_ptr)
函数调用 Functions Called
MBC_Delete_Mailbox

MBCE_Send_To_Mailbox
该函数提供参数错误检查的发送邮箱函数。
语法 Syntax
STATUS MBCE_Send_To_Mailbox(NU_MAILBOX *mailbox_ptr,VOID *message,UNSIGNED suspend)
函数调用 Functions Called
MBC_Sent_To_Mailbox
TCCE_Suspend_Error

MBCE_Receive_From_Mailbox
该函数提供参数错误检查的接收邮箱函数。
语法 Syntax
STATUS MBCE_Receive_From_Mailbox (NU_MAILBOX *mailbox_ptr,VOID *message,UNSIGNED suspend)
函数调用 Functions Called
MBC_Receive_From_Mailbox
TCCE_Suspend_Error

MBF_Established_Mailboxes
返回当前的已确定邮箱个数。先前删除的邮箱不再被包括在确定邮箱中。
语法 Syntax
UNSIGNED MBF_Established_Mailboxes(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

MBF_Mailbox_Pointers
创建一个邮箱指针链表，该链表从特定未知开始。Builds a list of mailbox pointers, starting at the specified location. The number of mailbox
pointers placed in the list is equivalent to the total number of mailboxes or the maximum number of pointers specified in the call.
语法 Syntax
UNSIGNED MBF_Mailbox_Pointers(NU_MAILBOX **pointer_list,UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

MBF_Mailbox_Information
返回特定邮箱信息。如果提供的邮箱指针是非法的，该函数简单地返回一个错误地状态。
语法 Syntax
STATUS MBF_Mailbox_Information(NU_MAILBOX *mailbox_ptr,
CHAR *name, OPTION *suspend_type,
DATA_ELEMENT *message_present,
UNSIGNED *tasks_waiting, NU_TASK
**first_task)
函数调用 Functions Called

[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

MBI_Initialize
该函数初始化邮箱组件操作控制数据结构，没有邮箱被初始化。
语法 Syntax
VOID MBI_Initialize(VOID)
函数调用 Functions Called
None

MBS_Reset_Mailbox
该函数复位一个邮箱到初始化状态。任何在邮箱的消息被丢弃。随着邮箱复位，并且所有的因邮箱挂起的任务将继续。
语法 Syntax
STATUS MBS_Reset_Mailbox(NU_MAILBOX *mailbox_ptr)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect

MBS_Broadcast_To_Mailbox
该函数广播一个消息到所有的等待邮箱消息的任务。如果没有任务在等待，则该函数就如同一个普通发送消息函数。
语法 Syntax
STATUS MBS_Broadcast_To_Mailbox(NU_MAILBOX *mailbox_ptr, VOID
*message, UNSIGNED suspend)
函数调用 Functions Called
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

MBSE_Reset_Mailbox
该函数提供参数错误检查的复位邮箱函数。
语法 Syntax
STATUS MBSE_Reset_Mailbox(NU_MAILBOX *mailbox_ptr)
函数调用 Functions Called
MBS_Reset_Mailbox

MBSE_Broadcast_To_ Mailbox
该函数提供参数错误检查的广播邮箱函数。
语法 Syntax
STATUS MBSE_Broadcast_To_Mailbox(NU_MAILBOX *mailbox_ptr,
VOID *message,
UNSIGNED suspend)
函数调用 Functions Called
MBS_Broadcast_To_Mailbox
TCCE_Suspend_Error

队列组件 Queue Component (QU)

队列组件 (QU)负责处理所有的Nucleus PLUS队列设备。一个Nucleus PLUS队列是一种在任务间传递消息的机制 。每一个邮箱能够持有多个消息。一个队列消息包括一个或多个32-bit 的字（word）。任务在等待一个空队列时可以被挂起。相反的，任务当试图发送一个消息到满队列中时也可被挂起。邮箱可以被用户动态创建和删除。参考《*Nucleus PLUS Reference Manual*》第三章得到更多关于队列的信息。

队列组件文件 Queue Files

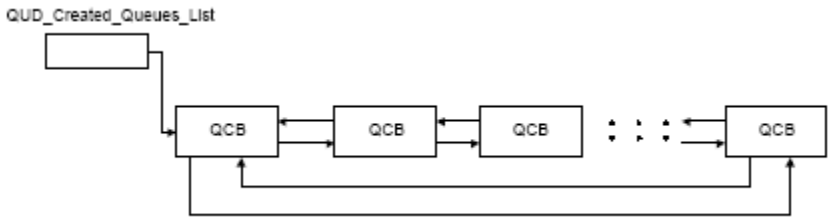
邮箱组件包括九个文件，每一个邮箱组件源文件如下定义：

文件 File	描述 Description
QU_DEFS.H	该文件定义QU组件特定的数据结构和常量。
QU_EXTR.H	该文件定义QU组件的外部接口。
QUD.C	文件定义QU组件的全局数据结构。
QUI.C	该文件包括QU组件的初始化代码。
QUF.C	该文件包括提供QU组件的信息的函数。
QUC.C	该文件包括提供QU组件的核心函数，该文件定义的函数实现基本的发送和接收一个队列。
QUS.C	该文件包括提供QU组件的附加函数，该文件定义的函数往往没有定义在QUC.C中的函数使用频繁。
QUCE.C	该文件包括有错误检查的定义在QUC.C中的函数接口。
QUSE.C	该文件包括有错误检查的定义在QUS.C中的函数接口。

队列组件数据结构 Queue Data Structures

已创建队列链表 Created Queue List

Nucleus PLUS 队列可以被动态地创建和删除。对于每一个创建的队列的队列控制块是被维护在一个双向的循环链表。最新创建的队列是被放置在链表的末端，而删除的队列是完整的从链表中移出。该链表的头指针是QUD_Created_Queues_List。



已创建队列链表保护 Created Queue List Protection

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建队列链表的完整性。 这种

保护归功于使用名为QUD_List_Protect的内在的保护结构。所以的队列的创建和删除都在TCD_List_Protect的保护下。

成员变量的声明

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

成员变量的概要

成员变量 Field	描述 Description
tc_tcb_pointer	Identifies the thread that currently has the protection.
tc_thread_waiting	A flag indicating that one or more threads are waiting for the protection.

队列总数 Total Queues

当前Nucleus PLUS 创建的队列总数是被变量QUD_Total_Queues指示。该变量的内容是在已创建链表的QCBs的个数。处理该变量是被QUD_List_Protect保护。

队列控制块 Queue Control Block

队列控制块QU_QCB包括队列消息区域（一个或更多的32-bit无符号字（word））和其他处理队列请求必需的数据成员。

成员变量的声明

```
CS_NODE qu_created
UNSIGNED qu_id
CHAR qu_name[NU_MAX_NAME]
DATA_ELEMENT qu_fixed_size
DATA_ELEMENT qu_fifo_suspend
DATA_ELEMENT qu_padding
UNSIGNED qu_queue_size
UNSIGNED qu_messages
UNSIGNED qu_message_size
UNSIGNED qu_available
UNSIGNED_PTR qu_start
UNSIGNED_PTR qu_end
UNSIGNED_PTR qu_read
UNSIGNED_PTR qu_write
UNSIGNED qu_tasks_waiting
struct QU_SUSPEND_STRUCT *qu_urgent_list
struct QU_SUSPEND_STRUCT *qu_suspension_list
```

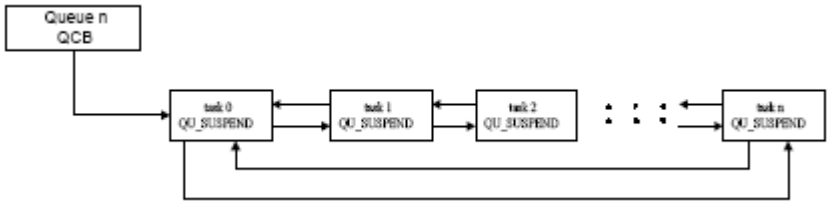
成员变量的概要

成员变量 Field	描述 Description
qu_created	This is the link node structure for queues. It is linked into the created queues list, which is a doubly-linked, circular list.
qu_id	This holds the internal queue identification of 0x51554555, which is equivalent to ASCII QUEU.
qu_name	This is the user-specified, 8 character name for the queue.
qu_fixed_size	A flag that indicates if the size of the queue is fixed or variable.

qu_fifo_suspend	A flag that determines whether tasks suspend in fifo or priority order.
qu_padding	This is used to align the queue structure on an even boundary. In some ports this field is not used.
qu_queue_size	This is the total size of the queue.
qu_messages	A flag that indicates if there is a message present in the queue.
qu_message_size	Holds the size of the queue message.
qu_available	Tells how many bytes are available in the queue.
qu_start	Stores the beginning of the queue.
qu_end	Stores the end of the queue.
qu_read	This is the read pointer.
qu_write	This is the write pointer.
qu_tasks_waiting	Indicates the number of tasks that are currently suspended on the queue.
*qu_urgent_list	A pointer to the suspension list for urgent messages.
*qu_suspension_list	The head pointer of the queue suspension list. If no tasks are suspended, this pointer isNULL.

挂起队列结构 Queue Suspension Structure

队列在空和满状态下，任务可以挂起。在挂起过程中，一个QU_SUSPEND结构被创建。在挂起期间，该结构包括任务信息和任务的队列请求信息。该挂起结构被连接到QCB在一个双向的循环链表，该结构是在挂起的任务的堆栈中创建的。每一个挂起在队列中的任务都有一个挂起结构。放置在挂起结构链表中的挂起结构的顺序是在队列创建时就确定的。如果一个FIFO挂起结构被选择，则该结构被放置在链表的尾部。否则，如果高优先级的挂起结构被选择， 则该结构被放置在优先级更高或相等的任务挂起块后面。



成员变量的声明

```
QU_SUSPEND_STRUCT
CS_NODE qu_suspend_link
QU_QCB *qu_queue
TC_TCB *qu_suspended_task
UNSIGNED_PTR qu_message_area
UNSIGNED qu_message_size
UNSIGNED qu_actual_size
STATUS qu_return_status
```

成员变量的概要

成员变量 Field	描述 Description
qu_suspend_link	A link node structure for linking with other suspended blocks. It is used in a doubly-linked, circular suspension list.
*qu_queue	A pointer to the queue structure.
*qu_suspended_task	A pointer to the Task Control Block of the suspended task.
qu_message_area	A pointer indicating where the suspended task's message

	buffer is.
qu_message_size	Stores the size of the requested message
qu_actual_size	Stores the actual size of the message.
qu_return_status	The completion status of the task suspended on the queue.

队列组件函数 Queue Functions

以下部分概要地描述了定时管理组件(QU)的函数。回顾实际的源码可以获得更多的信息。

QUC_Create_Queue
该函数创建一个队列并且放置它到已创建队列链表中。
语法 Syntax
```
STATUS QUC_Create_Queue(NU_QUEUE *queue_ptr, CHAR *name,
VOID *start_address, UNSIGNED
queue_size,OPTION message_type,
UNSIGNED message_size,
OPTION suspend_type)
```
函数调用 Functions Called
```
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
```

QUC_Delete_Queue
该函数删除一个队列并且将它从已创建队列链表中移出。所有在队列中挂起的任务将继续。注意该函数并不会释放队列所占有的内存。释放队列所占有的内存是应用程序的责任。
语法 Syntax
```
STATUS QUC_Delete_Queue(NU_QUEUE *queue_ptr)
```
函数调用 Functions Called
```
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
```

QUC_Send_To_Queue
该函数发送一个消息到特定的队列中。消息的长度是被调用者确定的。如果存在一个或更多在队列中等待消息而挂起的任务，该消息将被拷贝到第一个等待的任务区域。如果任务的请求得到满足，它将继续。否则，如果队列不能容纳消息，suspension of the calling task is an option of the caller。
语法 Syntax
```
STATUS QUC_Send_To_Queue(NU_QUEUE *queue_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)
```
函数调用 Functions Called
```
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
```

TCC_Suspend_Task

QUC_Receive_From_Queue

该函数从一个特定的队列中接收数据。消息的大小是被调用者指定。如果当前在队列中有一个消息，消息将被从队列中移出并放置到调用者的区域。如果该请求不能被满足，则调用者有可能被挂起。

语法 Syntax

```
STATUS QUC_Receive_From_Queue(NU_QUEUE *queue_ptr, VOID
*message, UNSIGNED size,
UNSIGNED *actual_size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task
```

QUC_Cleanup

该函数负责从一个队列移出一个挂起的块。它不会被调用除非超时或一个任务终止。注意：保护是一直起作用的（如同挂起的时间）。

语法 Syntax

```
VOID QUC_Cleanup(VOID *information)
```

函数调用 Functions Called

```
CSC_Remove_From_List
```

QUCE_Create_Queue

该函数提供参数错误检查的创建一个队列功能。

语法 Syntax

```
STATUS QUCE_Create_Queue(NU_QUEUE *queue_ptr, CHAR *name,
VOID *start_address, UNSIGNED
queue_size, OPTION message_type,
UNSIGNED message_size,
OPTION suspend_type)
```

函数调用 Functions Called

```
QUC_Create_Queue
```

QUCE_Delete_Queue

该函数提供参数错误检查的删除一个队列功能。

语法 Syntax

```
STATUS QUCE_Delete_Queue(NU_QUEUE *queue_ptr)
```

函数调用 Functions Called

```
QUC_Delete_Queue
```

QUCE_Send_To_Queue

该函数提供参数错误检查的发送消息到一个队列功能。

语法 Syntax

```
STATUS QUCE_Send_To_Queue(NU_QUEUE *queue_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
QUC_Send_To_Queue
TCCE_Suspend_Error
```

QUCE_Receive_From_Queue
该函数提供参数错误检查的从一个队列接收数据功能。
语法 Syntax
```
STATUS QUCE_Receive_From_Queue(NU_QUEUE *queue_ptr,
VOID *message, UNSIGNED
size, UNSIGNED*actual_size,
UNSIGNED suspend)
```
函数调用 Functions Called
QUC_Receive_From_Queue
TCCE_Suspend_Error

QUF_Established_Queues
该函数返回确定的队列个数。先前被删除的队列不再被包括在确定的队列中。
语法 Syntax
```
UNSIGNED QUF_Established_Queues(VOID)
```
函数调用 Functions Called
[TCT_Check_Stack]

QUF_Queue_Information
该函数返回特定的队列信息。然而，如果提供的队列指针是非法的，该函数简单的返回一个错误状态。
语法 Syntax
```
STATUS QUF_Queue_Information(NU_QUEUE*queue_ptr, CHAR *name, VOID
**start_address, UNSIGNED*queue_size,
UNSIGNED *available,
UNSIGNED *messages,
OPTION *message_type,
UNSIGNED *message_size,
OPTION *suspend_type,
UNSIGNED *tasks_waiting,
NU_TASK **first_task)
```
函数调用 Functions Called
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

QUF_Queue_Pointers
创建一个在特定位置开始的队列指针链表，队列指针个数 The number of queue pointers placed in the list is equivalent to the total number of queues or the maximum number of pointers specified in the call.
语法 Syntax
```
UNSIGNED QUF_Queue_Pointers(NU_QUEUE **pointer_list,
UNSIGNED maximum_pointers)
```
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

QUI_Initialize
该函数初始化队列组件操作控制数据结构，没有队列被初始化。
语法 Syntax
```
VOID QUI_Initialize(VOID)
```
函数调用 Functions Called
None

QUS_Reset_Queue

该函数复位特定队列到初始化状态。任何在队列的消息被丢弃。随着队列复位，并且所有的因队列挂起的任务将继续。

语法 Syntax

```
STATUS QUS_Reset_Queue(NU_QUEUE *queue_ptr)
```

函数调用 Functions Called

```
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect
```

QUS_Send_To_Front_Of_Queue

该函数发送一个消息到一个特定消息队列的开头。消息的长度是被调用者确定。如果存在任何在队列中等待消息而挂起的任务。消息被拷贝到第一个等待任务区域并且该任务将继续。如果在队列中有足够的空间，消息被拷贝在所有其他消息前面。如果队列中有足够的空间，调用者是有可能被挂起。

语法 Syntax

```
STATUS QUS_Send_To_Front_Of_Queue(NU_QUEUE *queue_ptr,
VOID *message,
UNSIGNED size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
CSC_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task
[TCT_Check_Stack]
```

QUS_Broadcast_To_Queue

该函数广播一个消息到所有的等待特定队列消息的任务。如果没有任务在等待，则该函数就如同一个普通发送消息函数。

语法 Syntax

```
STATUS QUS_Broadcast_To_Queue(NU_QUEUE *queue_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task
```

QUSE_Reset_Queue

该函数提供参数错误检查的复位一个队列功能。

语法 Syntax

```
STATUS QUSE_Reset_Queue(NU_QUEUE *queue_ptr)
```

函数调用 Functions Called

```
QUS_Reset_Queue
```

QUSE_Send_To_Front_Of_Queue

该函数执行带参数错误检查的发送消息到队列头的功能。

语法 Syntax
STATUS QUSE_Send_To_Front_Of_Queue(NU_QUEUE *queue_ptr,
UNSIGNED size,
UNSIGNED suspend)
函数调用 Functions Called
QUS_Send_To_Front_Of_Queue
TCCE_Suspend_Error

QUSE_Broadcast_To_Queue
该函数提供参数错误检查的广播一个消息队列功能。
语法 Syntax
STATUS QUSE_Broadcast_To_Queue(NU_QUEUE *queue_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)
函数调用 Functions Called
QUS_Broadcast_To_Queue
TCCE_Suspend_Error

管道组件 Pipe Component (PI)

管道组件 (PI)负责处理所有的Nucleus PLUS管道设备。一个Nucleus PLUS管道是一种在任务间传递消息的机制 。每一个管道能够持有多个消息。一个管道消息包括一个或多个字节（byte）。任务在等待一个空管道时可以被挂起。相反的，任务当试图发送一个消息到满管道中时也可被挂起。管道可以被用户动态创建和删除。参考《*Nucleus PLUS Reference Manual*》第三章得到更多关于队列的信息

管道组件文件 Pipe Files

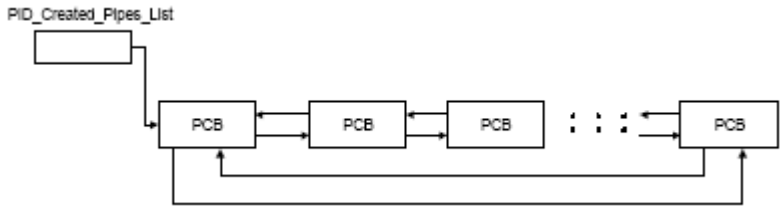
管道组件(PI)包括九个文件，每一个管道组件源文件如下定义：

文件 File	描述 Description
PI_DEFS.H	该文件定义PI组件特定的数据结构和常量。
PI_EXTR.H	该文件定义PI组件的外部接口。
PID.C	文件定义PI组件的全局数据结构。
PII.C	该文件包括PI组件的初始化代码。
PIF.C	该文件包括提供PI组件的信息的函数。
PIC.C	该文件包括提供PI组件的核心函数，该文件定义的函数实现基本的发送和接收一个邮箱。
PIS.C	该文件包括提供PI组件的附加函数，该文件定义的函数往往没有定义在MBC.C中的函数使用频繁。
PICE.C	该文件包括有错误检查的定义在PIC.C中的函数接口。
PISE.C	该文件包括有错误检查的定义在PIS.C中的函数接口。

管道组件数据结构 Pipe Data Structures

已创建管道链表 Created Pipe List

Nucleus PLUS 管道可以被动态地创建和删除。对于每一个创建的管道的管道控制块(PCB)是被维护在一个双向的循环链表。最新创建的管道是被放置在链表的末端，而删除的队列是完整的从链表中移出。该链表的头指针是**PID_Created_Pipes_List**。



已创建管道链表保护 Created Pipe List Protection

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建管道链表的完整性。 这种保护归功于使用名为**PID_List_Protect**的内在的保护结构。所以的队列的创建和删除都在**PID_List_Protect**的保护下。

成员变量的声明

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

成员变量的概要

文件 File	描述 Description
tc_tcb_pointer	Identifies the thread that currently has the protection.
tc_thread_waiting	A flag indicating that one or more threads are waiting for the protection.

管道总数Total Pipes

当前Nucleus PLUS 创建的管道总数是被变量**PID_Total_Queues**指示。该变量的内容是在已创建链表的QCBs的个数。处理该变量是被PID_List_Protect保护。

管道控制块 Pipe Control Block

管道控制块**PI_PCB**包括管道消息区域（一个或更多的字节（byte））和其他处理队列请求必需的数据成员。

Pipe Control Block

The Pipes Control Block **PI_PCB** contains the pipe message area (1 or more bytes) and other fields necessary for processing pipe requests.

成员变量的声明

```
CS_NODE pi_created
UNSIGNED pi_id
CHAR pi_name[NU_MAX_NAME]
DATA_ELEMENT pi_fixed_size
DATA_ELEMENT pi_fifo_suspend
DATA_ELEMENT pi_padding[PAD_2]
UNSIGNED pi_pipe_size
UNSIGNED pi_message_size
UNSIGNED pi_available
```

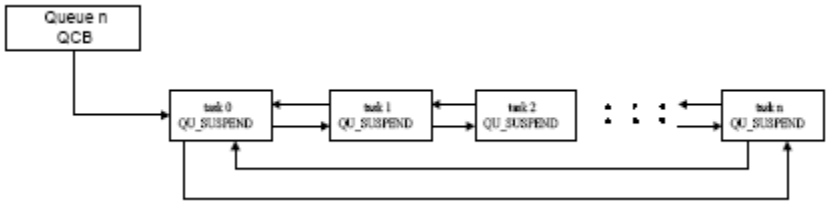
```
BYTE_PTR pi_start
BYTE_PTR pi_end
BYTE_PTR pi_read
BYTE_PTR pi_write
UNSIGNED pi_tasks_waiting
UNSIGNED pi_messages
struct PI_SUSPEND_STRUCT *pi_urgent_list
struct PI_SUSPEND_STRUCT *pi_suspension_list
```

成员变量的概要

文件 File	描述 Description
qu_created	This is the link node structure for queues. It is linked into the created queues list, which is a doubly-linked, circular list.
qu_id	This holds the internal queue identification of 0x51554555, which is equivalent to ASCII QUEU.
qu_name	This is the user-specified, 8 character name for the queue.
qu_fixed_size	A flag that indicates if the size of the queue is fixed or variable.
qu_fifo_suspend	A flag that determines whether tasks suspend in fifo or priority order.
qu_padding	This is used to align the queue structure on an even boundary. In some ports this field is not used.
qu_queue_size	This is the total size of the queue.
qu_messages	A flag that indicates if there is a message present in the queue.
qu_message_size	Holds the size of the queue message.
qu_available	Tells how many bytes are available in the queue.
qu_start	Stores the beginning of the queue.
qu_end	Stores the end of the queue.
qu_read	This is the read pointer.
qu_write	This is the write pointer.
qu_tasks_waiting	Indicates the number of tasks that are currently suspended on the queue.
*qu_urgent_list	A pointer to the suspension list for urgent messages.
*qu_suspension_list	The head pointer of the queue suspension list. If no tasks are suspended, this pointer is NULL.

挂起管道结构 Pipe Suspension Structure

管道在空和满状态下，任务可以挂起。在挂起过程中，一个PI_SUSPEND结构被创建。在挂起期间，该结构包括任务信息和任务的管道请求信息。该挂起结构被连接到PCB在一个双向的循环链表，该结构是在挂起的任务的堆栈中创建的。每一个挂起在管道中的任务都有一个挂起结构。放置在挂起结构链表中的挂起结构的顺序是在管道创建时就确定的。如果一个FIFO挂起结构被选择，则该结构被放置在链表的尾部。否则，如果高优先级的挂起结构被选择， 则该结构被放置在优先级更高或相等的任务挂起块后面。



成员变量的声明

```
CS_NODE pi_suspend_link
PI_PCB *pi_pipe
TC_TCB *pi_suspended_task
BYTE_PTR pi_message_area
UNSIGNED pi_message_size
UNSIGNED pi_actual_size
STATUS pi_return_status
```

成员变量的概要

文件 File	描述 Description
qu_suspend_link	A link node structure for linking with other suspended blocks. It is used in a doubly-linked, circular suspension list.
*qu_queue	A pointer to the queue structure.
*qu_suspended_task	A pointer to the Task Control Block of the suspended task.
qu_message_area	A pointer indicating where the suspended task's message buffer is.
qu_message_size	Stores the size of the requested message
qu_actual_size	Stores the actual size of the message.
qu_return_status	The completion status of the task suspended on the queue.

管道组件函数 Pipe Functions

以下部分概要地描述了管道管理组件(QU)的函数。回顾实际的源码可以获得更多的信息。

PIC_Create_Pipe

该函数创建一个管道并且放置它到已创建管道链表中。

语法 Syntax

```
STATUS PIC_Create_Pipe(NU_PIPE *pipe_ptr, CHAR *name, VOID
*start_address, UNSIGNED pipe_size,
OPTION message_type, UNSIGNED
message_size, OPTION suspend_type)
```

函数调用 Functions Called

```
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
```

PIC_Delete_Pipe

该函数删除一个管道并且将它从已创建管道链表中移出。所有在管道中挂起的任务将继续。注意该函数并不会释放管道所占有的内存。释放管道所占有的内存是应用程序的责任。

语法 Syntax

STATUS PIC_Delete_Pipe(NU_PIPE *pipe_ptr)

函数调用 Functions Called

CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System

PIC_Send_To_Pipe

This function sends a message to the specified pipe. The message length is determined by the caller. If there are one or more tasks suspended on the pipe for a message, the message is copied into the message area of the first waiting task. If the task's request is satisfied, it is resumed. Otherwise, if the pipe cannot hold the message, suspension of the calling task is an option of the caller.

语法 Syntax

STATUS PIC_Send_To_Pipe(NU_PIPE *pipe_ptr, VOID *message,
UNSIGNED size, UNSIGNED suspend)

函数调用 Functions Called

CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

PIC_Receive_From_Pipe

该函数从一个特定的管道中接收数据。消息的大小是被调用者指定。如果当前在管道中有一个消息，消息将被从管道中移出并放置到调用者的区域。如果该请求不能被满足，则调用者有可能被挂起。

语法 Syntax

STATUS PIC_Receive_From_Pipe(NU_PIPE *pipe_ptr, VOID
*message, UNSIGNED size,
UNSIGNED *actual_size,
UNSIGNED suspend)

函数调用 Functions Called

CSC_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task
TCC_Task_Priority
TCT_Check_Stack]
TCT_Control_To_System
TCT_Current_Thread
TCT_System_Protect
TCT_Unprotect

PIC_Cleanup

该函数负责从一个管道移出一个挂起的块。它不会被调用除非超时或一个任务终止。注意：保护是一直起

作用的（如同挂起的时间）。

语法 Syntax

VOID PIC_Cleanup(VOID *information)

函数调用 Functions Called

CSC_Remove_From_List

PICE_Create_Pipe

该函数提供参数错误检查的创建一个管道功能。

语法 Syntax

STATUS PICE_Create_Pipe(NU_PIPE *pipe_ptr, CHAR *name,
VOID *start_address, UNSIGNED
pipe_size, OPTION message_type,
UNSIGNED message_size,
OPTION suspend_type)

函数调用 Functions Called

PIC_Create_Pipe

PICE_Delete_Pipe

该函数提供参数错误检查的删除一个管道功能。

语法 Syntax

STATUS PICE_Delete_Pipe(NU_PIPE *pipe_ptr)

函数调用 Functions Called

PIC_Delete_Pipe

PICE_Send_To_Pipe

This function performs error checking on the parameters supplied to the send message to pipe function.

语法 Syntax

STATUS PICE_Send_To_Pipe(NU_PIPE *pipe_ptr, VOID *message,
UNSIGNED size, UNSIGNED suspend)

函数调用 Functions Called

PIC_Send_To_Pipe
TCCE_Suspend_Error

PICE_Receive_From_Pipe

该函数提供参数错误检查的从一个管道接收数据功能。

语法 Syntax

STATUS PICE_Receive_From_Pipe(NU_PIPE *pipe_ptr, VOID
*message, UNSIGNED size,
UNSIGNED *actual_size,
UNSIGNED suspend)

函数调用 Functions Called

PIC_Receive_From_Pipe
TCCE_Suspend_Error

PIF_Established_Pipes

该函数返回确定的管道个数。先前被删除的管道不再被包括在确定的管道中。

语法 Syntax

UNSIGNED PIF_Established_Pipes(VOID)

函数调用 Functions Called

[TCT_Check_Stack]

PIF_Pipe_Information
该函数返回特定的管道信息。然而，如果提供的管道指针是非法的，该函数简单的返回一个错误状态。
语法 Syntax
```
STATUS PIF_Pipe_Information(NU_PIPE *pipe_ptr, CHAR *name,
VOID **start_address,
UNSIGNED *pipe_size,
UNSIGNED *available,
UNSIGNED *messages,
OPTION *message_type,
UNSIGNED *message_size,
OPTION *suspend_type,
UNSIGNED *tasks_waiting,
NU_TASK **first_task)
```
函数调用 Functions Called
```
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect
```

PIF_Pipe_Pointers
Builds a list of pipe pointers, starting at the specified location. The number of pipe pointers placed in the list is equivalent to the total number of pipes or the maximum number of pointers specified in the call.
语法 Syntax
```
UNSIGNED PIF_Pipe_Pointers(NU_PIPE **pointer_list,
UNSIGNED maximum_pointers)
```
函数调用 Functions Called
```
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
```

PII_Initialize
该函数初始化管道组件操作控制数据结构，没有管道被初始化。
语法 Syntax
```
VOID PII_Initialize(VOID)
```
函数调用 Functions Called
None

PIS_Reset_Pipe
This function resets the specified pipe back to the original state. Any messages in the pipe are discarded. Also, any tasks currently suspended on the pipe are resumed with the reset status.
语法 Syntax
```
STATUS PIS_Reset_Pipe(NU_PIPE *pipe_ptr)
```
函数调用 Functions Called
```
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect
```

PIS_Send_To_Front_Of_Pipe

This function sends a message to the front of the specified message pipe. The message length is determined by the caller. If there are any tasks suspended on the pipe for a message, the message is copied into the message area of the first waiting task and that task is resumed. If there is enough room in the pipe, the message is copied in front of all other messages. If there is not enough room in the pipe, suspension of the caller is possible.

语法 Syntax

STATUS PIS_Send_To_Front_Of_Pipe(NU_PIPE *pipe_ptr,
VOID *message, UNSIGNED
size, UNSIGNED suspend)

函数调用 Functions Called

CSC_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

PIS_Broadcast_To_Pipe

该函数广播一个消息到所有的等待特定管道消息的任务。如果没有任务在等待，则该函数就如同一个普通发送消息函数。

语法 Syntax

STATUS PIS_Broadcast_To_Pipe(NU_PIPE *pipe_ptr,
VOID *message, UNSIGNED size,
UNSIGNED suspend)

函数调用 Functions Called

CSC_Place_On_List
CSC_Priority_Place_On_List
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
TCC_Suspend_Task

PISE_Reset_Pipe

该函数提供参数错误检查的复位一个管道功能。

语法 Syntax

STATUS PISE_Reset_Pipe(NU_PIPE *pipe_ptr)

函数调用 Functions Called

PIS_Reset_Pipe

PISE_Send_To_Front_Of_Pipe

This function performs error checking on the parameters supplied to the send message to front of pipe function.

语法 Syntax

STATUS PISE_Send_To_Front_Of_Pipe(NU_PIPE *pipe_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)

函数调用 Functions Called

PIS_Send_To_Front_Of_Pipe
TCCE_Suspend_Error

PISE_Broadcast_To_Pipe

该函数提供参数错误检查的广播一个消息管道功能。

语法 Syntax

```
STATUS PISE_Broadcast_To_Pipe(NU_PIPE *pipe_ptr, VOID
*message, UNSIGNED size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
PIS_Broadcast_To_Pipe
TCCE_Suspend_Error
```

信号量组件 Semaphore Component (SM)

队列组件 (SM)负责处理所有的Nucleus PLUS信号量设备。一个Nucleus PLUS信号量是一种在一个应用中许多任务间同步机制 。Nucleus PLUS提供的信号量计数范围为0到4,294,967,294。任务在等待非0的信号量时可以被挂起。信号量可以被用户动态创建和删除。参考《*Nucleus PLUS Reference Manual*》第三章得到更多关于队列的信息

信号量组件文件 Semaphore Files

队列组件(SM)包括九个文件，每一个管道组件源文件如下定义：

文件 File	描述 Description
SM_DEFS.H	该文件定义SM组件特定的数据结构和常量。
SM_EXTR.H	该文件定义SM组件的外部接口。
SMD.C	文件定义SM组件的全局数据结构。
SMI.C	该文件包括SM组件的初始化代码。
SMF.C	该文件包括提供SM组件的信息的函数。
SMC.C	该文件包括提供SM组件的核心函数，该文件定义的函数实现基本的发送和接收一个邮箱。
SMS.C	该文件包括提供SM组件的附加函数，该文件定义的函数往往没有定义在MBC.C中的函数使用频繁。
SMCE.C	该文件包括有错误检查的定义在SMC.C中的函数接口。
SMSE.C	该文件包括有错误检查的定义在SMS.C中的函数接口。

信号量组件数据结构 Semaphore Data Structures

已创建信号量链表 Created Semaphore List

Nucleus PLUS 信号可以被动态地创建和删除。对于每一个创建的信号的信号控制块(SCB)是被维护在一个双向的循环链表。最新创建的管道是被放置在链表的末端，而删除的队列是完整的从链表中移出。该链表的头指针是SMD_Created_Semaphores_List。

已创建信号量链表保护 Created Semaphore List Protection

Nucleus PLUS 在任务和任务、HISR和HISR、任务和HISR之间竞争时保护已创建信号量链表的完整性。 这

种保护归功于使用名为SMD_List_Protect的内在的保护结构。所以的信号量的创建和删除都在SMD_List_Protect的保护下。

成员变量的声明

TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting

成员变量的概要

文件 File	描述 Description
tc_tcb_pointer	Identifies the thread that currently has the protection.
tc_thread_waiting	A flag indicating that one or more threads are waiting for the protection.

信号量总数 Total Semaphore

当前Nucleus PLUS 创建的信号量总数是被变量SMD_Total_Semaphores指示。该变量的内容是在已创建链表的SCBs的个数。处理该变量是被SMD_List_Protect保护。

管道控制块 Semaphore Control Block

The Semaphores Control Block **SM_SCB** contains the semaphore count and other fields necessary for processing semaphore requests.

Field Declarations
CS_NODE sm_created
UNSIGNED sm_id
CHAR sm_name[NU_MAX_NAME]
UNSIGNED sm_semaphore_count
DATA_ELEMENT sm_fifo_suspend
DATA_ELEMENT sm_padding[PAD_1]
UNSIGNED sm_tasks_waiting
struct SM_SUSPEND_STRUCT *sm_suspension_list
Field Summary
Field Description

sm_created This is the link node structure for semaphores. It is linked into the created semaphores list, which is a doubly-linked, circular list.

sm_id This holds the internal semaphore identification of 0x53454D41 which is equivalent to ASCII SEMA.

sm_name This is the user-specified, 8 character name for the semaphore.

sm_semaphore_count Stores the current count of the semaphore.

sm_fifo_suspend A flag that determines whether tasks suspend in fifo or priority order.

sm_padding This is used to align the semaphore structure on an even boundary.
In some ports this field is not used.

sm_tasks_waiting Indicates the number of tasks that are currently suspended on the semaphore.

***sm_suspension_list** The head pointer of the semaphore suspension list. If no tasks are suspended, this pointer is **NULL**.

Semaphore Suspension Structure

Tasks can suspend on a semaphore whose current count is zero. During the suspension process a **SM_SUSPEND_STRUCT** structure is built. This structure contains information about the task and the task's semaphore request at the time of suspension. This suspension structure is linked onto the SCB in a doubly-linked, circular list and is

allocated from the suspending task's stack. There is one suspension block for every task suspended on the semaphore.
The order of the suspension block placement on the suspend list is determined at semaphore creation. If a FIFO suspension was selected, the suspension block is added to the end of the list. Otherwise, if priority suspension was selected, the suspension block is placed after suspension blocks for tasks of equal or higher priority.
Field Declarations
CS_NODE sm_suspend_link
SM_SCB *sm_semaphore
TC_TCB *sm_suspended_task
STATUS sm_return_status
Field Summary
Field Description
sm_suspend_link A link node structure for linking with other suspended blocks. It is used in a doubly-linked, circular suspension list.
***sm_semaphore** A pointer to the semaphore structure.
***sm_suspended_task** A pointer to the Task Control Block of the suspended task.
sm_return_status The completion status of the task suspended on the semaphore
Semaphore n
SCB
task 0
SM_SUSPEND
task 1
SM_SUSPEND
task 2
SM_SUSPEND
task n
SM_SUSPEND

信号量组件函数 Semaphore Functions

以下部分概要地描述了信号量组件(SM)的函数。回顾实际的源码可以获得更多的信息。

SMC_Create_Semaphore
This function creates a semaphore and places it on the list of created semaphores.
语法 Syntax
STATUS SMC_Create_Semaphore(NU_SEMAPHORE *semaphore_ptr,
CHAR *name,
UNSIGNED initial_count,
OPTION suspend_type)
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
SMC_Delete_Semaphore
This function deletes a semaphore and removes it from the list of created semaphores. All

tasks suspended on the semaphore are resumed. Note that this function does not free the memory associated with the semaphore control block. That is the responsibility of the application.

语法 Syntax

STATUS SMC_Delete_Semaphore(NU_SEMAPHORE *semaphore_ptr)

函数调用 Functions Called

CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System

SMC_Obtain_Semaphore

This function obtains an instance of the semaphore. An instance corresponds to decrementing the counter by one. If the counter is greater than zero at the time of this call, this function can be completed immediately. Otherwise, suspension is possible.

语法 Syntax

STATUS SMC_Obtain_Semaphore(NU_SEMAPHORE *semaphore_ptr,
UNSIGNED suspend)

函数调用 Functions Called

CSC_Place_On_List
CSC_Priority_Place_On_List
[HIC_Make_History_Entry]
TCC_Suspend_Task
TCC_Task_Priority
[TCT_Check_Stack]
TCT_Current_Thread
TCT_System_Protect
TCT_Unprotect

SMC_Release_Semaphore

This function releases a previously obtained semaphore. If one or more tasks are waiting, the first task is given the released instance of the semaphore. Otherwise, the semaphore instance counter is simply incremented.

语法 Syntax

STATUS SMC_Release_Semaphore(NU_SEMAPHORE *semaphore_ptr)

函数调用 Functions Called

CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect

SMC_Cleanup

This function is responsible for removing a suspension block from a semaphore. It is not called unless a timeout or a task terminate is in progress. Note that protection (the same as at suspension time) is already in effect.

语法 Syntax

VOID SMC_Cleanup(VOID *information)

函数调用 Functions Called

CSC_Remove_From_List
SMCE_Create_Semaphore

This function performs error checking on the parameters supplied to the create semaphore function.

语法 Syntax

```
STATUS SMCE_Create_Semaphore(NU_SEMAPHORE *semaphore_ptr,
CHAR *name, UNSIGNED
initial_count,
OPTION suspend_type)
```

函数调用 Functions Called

SMC_Create_Semaphore

SMCE_Delete_Semaphore

This function performs error checking on the parameters supplied to the delete semaphore function.

语法 Syntax

```
STATUS SMCE_Delete_Semaphore(NU_SEMAPHORE *semaphore_ptr)
```

函数调用 Functions Called

SMC_Delete_Semaphore

SMCE_Obtain_Semaphore

This function performs error checking on the parameters supplied to the obtain semaphore function.

语法 Syntax

```
STATUS SMCE_Obtain_Semaphore(NU_SEMAPHORE *semaphore_ptr,
UNSIGNED suspend)
```

函数调用 Functions Called

SMC_Obtain_Semaphore

TCCE_Suspend_Error

SMCE_Release_Semaphore

This function performs error checking on the parameters supplied to the release semaphore function.

语法 Syntax

```
STATUS SMCE_Release_Semaphore(NU_SEMAPHORE *semaphore_ptr)
```

函数调用 Functions Called

SMC_Release_Semaphore

SMF_Established_Semaphores

This function returns the current number of established semaphores. Semaphores previously deleted are no longer considered established.

语法 Syntax

```
UNSIGNED SMF_Established_Semaphores(VOID)
```

函数调用 Functions Called

[TCT_Check_Stack]

SMF_Semaphore_Pointers

Builds a list of semaphore pointers, starting at the specified location. The number of semaphore pointers placed in the list is equivalent to the total number of semaphores or the maximum number of pointers specified in the call.

语法 Syntax

```
UNSIGNED SMF_Semaphore_Pointers(NU_SEMAPHORE
**pointer_list,
UNSIGNED maximum_pointers)
```

函数调用 Functions Called

[TCT_Check_Stack]

TCT_Protect

TCT_Unprotect

SMF_Semaphore_Information
This function returns information about the specified semaphore. However, if the supplied semaphore pointer is invalid, the function simply returns an error status.
语法 Syntax
STATUS SMF_Semaphore_Information (NU_SEMAPHORE
*semaphore_ptr,
CHAR *name,
UNSIGNED *current_count,
OPTION *suspend_type,
UNSIGNED tasks_waiting,
NU_TASK **first_task)
Functions Calle
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect

SMI_Initialize
This function initializes the data structures that control the operation of the Semaphore Component. There are no semaphores initially.
语法 Syntax
VOID SMI_Initialize(VOID)
函数调用 Functions Called
None
SMS_Reset_Semaphore
This function resets a semaphore back to the initial state. All tasks suspended on the semaphore are resumed with the reset completion status.
语法 Syntax
STATUS SMS_Reset_Semaphore(NU_SEMAPHORE *semaphore_ptr,
UNSIGNED initial_count)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect

SMSE_Reset_Semaphore
This function performs error checking on the parameters supplied to the reset semaphore function.
语法 Syntax
STATUS SMSE_Reset_Semaphore(NU_SEMAPHORE *semaphore_ptr,
UNSIGNED initial_count)
函数调用 Functions Called
SMS_Reset_Semaphore

Event Group Component (EV)

The Event Group Component (EV) is responsible for processing all Nucleus PLUS event group facilities. A Nucleus PLUS event is a mechanism to indicate that a certain system event has occurred. An event is represented by a single bit in an event group. This bit is

called an event flag. There are 32 event flags in each event group. Tasks may suspend while waiting for a particular set of event flags. Event groups are dynamically created and deleted by the user. Please see Chapter 3 of the *Nucleus PLUS Reference Manual* for more detailed information about events.

Event Group Files

The Event Group Component (EV) consists of seven files. Each source file of the Event Group Component is defined below.

File Description

EV_DEFS.H This file contains constants and data structure definitions specific to the EV.

EV_EXTR.H All external interfaces to the EV are defined in this file.

EVD.C Global data structures for the EV are defined in this file.

EVI.C This file contains the initialization function for the EV.

EVF.C This file contains the information gathering functions for the EV.

EVC.C This file contains all of the core functions of the EV. Functions that handle basic set-event and retrieve-event services are defined in this file.

EVCE.C This file contains the error checking function interfaces for the core functions defined in **EVC.C.**

Event Group Data Structures

Created Event Group List

Nucleus PLUS events may be created and deleted dynamically. The Event Group Control Block (GCB) for each created event group is kept on a doubly-linked, circular list. Newly created event groups are placed at the end of the list, while deleted event groups are completely removed from the list. The head pointer of this list is EVD_Created_Events_Group_List.

Created Event Group List Protection

Nucleus PLUS protects the integrity of the Created Events Group List from competing tasks and/or HISRs. This is done by using an internal protection structure called **EVD_List_Protect.** All event group creation and deletion is done under the protection of EVD_List_Protect.

语法 Syntax

```
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
```

函数调用 Functions Called

Field Description

tc_tcb_pointer Identifies the thread that currently has the protection.

tc_thread_waiting A flag indicating that one or more threads are waiting for protection

GCB GCB GCB GCB

EVD_Created_Events_Group_List

Total Event Groups

The total number of currently created Nucleus PLUS event groups is contained in the variable **EVD_Total_Event_Groups.** The contents of this variable corresponds to the

number of GCBs on the created list. Manipulation of this variable is also done under the protection of EVD_List_Protect.

Event Group Control Block

The Event Group Control Block **EV_GCB** contains the current event flags and other fields necessary for processing event requests.

Field Declarations

CS_NODE ev_created
UNSIGNED ev_id
CHAR ev_name[NU_MAX_NAME]
UNSIGNED ev_current_events
UNSIGNED ev_tasks_waiting
struct EV_SUSPEND_STRUCT *ev_suspension_list

Field Summary

Field Description

ev_created This is the link node structure for events. It is linked into the created events group list, which is a doubly-linked, circular list.

ev_id This holds the internal event group identification of 0x45564E54 which is equivalent to ASCII EVNT.

ev_name This is the user-specified, 8 character name for the event group.

ev_current_events Contains the current event flags.

ev_tasks_waiting Indicates the number of tasks that are currently suspended on an event group.

***ev_suspension_list** The head pointer of the event group suspension list. If no tasks are suspended, this pointer is **NULL**.

Event Group Suspension Structure

Tasks can suspend when an event group does not match the user specified combination of event flags. During the suspension process the **EV_SUSPEND_STRUCT** structure is built. This structure contains information about the task and the task's event group request at the time of suspension. This suspension structure is linked onto the GCB in a doubly-linked, circular list and is allocated off of the suspending task's stack. There is one suspension block for every task suspended on the event group .

The order of the suspension block placement on the suspend list is determined at event group creation. If a FIFO suspension was selected, the suspension block is added to the end of the list. Otherwise, if priority suspension was selected, the suspension block is placed after suspension blocks for tasks of equal or higher priority.

Field Declarations

CS_NODE ev_suspend_link
EV_GCB *ev_event_group
UNSIGNED ev_requested_events
DATA_ELEMENT ev_operation
DATA_ELEMENT ev_padding[PAD_1]
TC_TCB *ev_suspended_task
STATUS ev_return_status
UNSIGNED ev_actual_events

Event Group n
GCB
task 0
EV_SUSPEND
task 1

EV_SUSPEND
task 2
EV_SUSPEND
task n
EV_SUSPEND

Field Summary
Field Description
ev_suspend_link A link node structure for linking with other suspended blocks. It is used in a doubly-linked circular suspension list
***em_event_group** A pointer to the event group structure.
ev_requested_events The event group that has been requested.
ev_operation The type of operation that is requested on the event group. This is typically some sort of AND/OR combination.
ev_padding This is used to align the suspend event group structure on an even boundary. In some ports this field is not used.
***ev_suspended_task** A pointer to the Task Control Block of the suspended task.
ev_return_status The completion status of the task suspended on the event group.
ev_actual_events The set of actual event flags returned by the request.

Event Group Functions

The following sections provide a brief description of the functions in the Event Group Component (EV). Review of the actual source code is recommended for further information.
EVC_Create_Event_Group
Creates an event group and then places it on the list of created event groups.
语法 Syntax
STATUS EVC_Create_Event_Group(NU_EVENT_GROUP
*event_group_ptr,
CHAR *name)
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

EVC_Delete_Event_Group
Deletes an event group and removes it from the list of created event groups. All tasks suspended on the event group are resumed. Note that this function does not free the memory associated with the event group control block. That is the responsibility of the application.
语法 Syntax
STATUS EVC_Delete_Event_Group(NU_EVENT_GROUP *event_group_ptr)

函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Unprotect
EVC_Set_Events
Sets event flags within the specified event flag group. Event flags may be **AND**ed or **OR**ed against the current events of the group. Tasks suspended on a group are resumed when the requested event is satisfied.
语法 Syntax
STATUS EVC_Set_Events(NU_EVENT_GROUP *event_group_ptr,
UNSIGNED events, OPTION operation)
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect

EVC_Retrieve_Events
Retrieves various combinations of event flags from the specified event group. If the group does not contain the necessary flags, suspension of the calling task is possible.
语法 Syntax
STATUS EVC_Retrieve_Events (NU_EVENT_GROUP
*event_group_ptr,
UNSIGNED requested_events,
OPTION operation,
UNSIGNED *retrieved_events,
UNSIGNED suspend)
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
TCC_Suspend_Task
[TCT_Check_Stack]
TCT_Current_Thread
TCT_System_Protect
TCT_Unprotect
EVC_Cleanup
This function is responsible for removing a suspension block from an event group. It is not called unless a timeout or a task terminate is in progress. Note that protection (the same as at suspension time) is already in effect.
语法 Syntax
VOID EVC_Cleanup(VOID *information)
函数调用 Functions Called
CSC_Remove_From_List

EVCE_Create_Event_Group
This function performs error checking on the parameters supplied to the create event group function.
语法 Syntax
STATUS EVCE_Create_Event_Group(NU_EVENT_GROUP
*event_group_ptr,
CHAR *name)
函数调用 Functions Called
EVC_Create_Event_Group
EVCE_Delete_Event_Group
This function performs error checking on the parameters supplied to the delete event group function.
语法 Syntax
STATUS EVCE_Delete_Event_Group(NU_EVENT_GROUP *event_group_ptr)
函数调用 Functions Called
EVC_Delete_Event_Group
EVCE_Set_Events
This function performs error checking on the parameters supplied to the set events group function.
语法 Syntax
STATUS EVCE_Set_Events(NU_EVENT_GROUP *event_group_ptr,
UNSIGNED events, OPTION operation)
函数调用 Functions Called
EVC_Set_Events
EVCE_Retrieve_Events

This function performs error checking on the parameter supplied to the retrieve events function.
语法 Syntax
STATUS EVCE_Retrieve_Events(NU_EVENT_GROUP *event_group_ptr,
UNSIGNED requested_events,
OPTION operation, UNSIGNED
*retrieved_events, UNSIGNED suspend)
函数调用 Functions Called
EVC_Retrieve_Events
TCCE_Suspend_Error
EVF_Established_Event_Groups
Returns the current number of established event groups. Event groups previously deleted are no longer considered established.
语法 Syntax
UNSIGNED EVF_Established_Event_Groups(VOID)
函数调用 Functions Called
[TCT_Check_Stack]
EVF_Event_Group_Pointers
Builds a list of event group pointers, starting at the specified location. The number of event group pointers placed in the list is equivalent to the total number of event groups or the maximum number of pointers specified in the call.
语法 Syntax
UNSIGNED EVF_Event_Group_Pointers(NU_EVENT_GROUP **pointer_list,
UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect

TCT_Unprotect
EVF_Event_Group_Information

Returns information about the specified event group. However, if the supplied event group pointer is invalid, the function simply returns an error status.

语法 Syntax

```
STATUS EVF_Event_Group_Information(NU_EVENT_GROUP
*event_group_ptr,
CHAR *name,
UNSIGNED *event_flags,
UNSIGNED *tasks_waiting,
NU_TASK **first_task)
```

函数调用 Functions Called

[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect
EVI_Initialize

This function initializes the data structures that control the operation of the Event Group Component. There are no event groups initially.

语法 Syntax

```
VOID EVI_Initialize(VOID)
```

函数调用 Functions Called

None

内存分配组件 Partition Memory Component (PM)

内存分配组件(PM) 负责处理所以的Nucleus PLUS内存分配设备。一个Nucleus PLUS分配内存池包括用户指定数目的固定大小的内存分配。当从一个空内存池等待足够的动态内存时，任务有可能挂起。分配内存池可以被用户动态的创建和删除。请参考《*Nucleus PLUS Reference Manual* 》获得更多的关于内存分配组件的信息。

内存分配组件文件 Partition Memory Files

内存分配组件(PM)包括七个文件。每一个I/O设备组件源码是如下定义的：

文件 File	描述 Description
PM_DEFS.H	该文件定义PM组件特定的数据结构和常量。
PM_EXTR.H	该文件定义PM组件的外部接口。
PMD.C	文件定义PM组件的全局数据结构。
PMI.C	该文件包括PM组件的初始化代码。
PMF.C	该文件包括提供PM组件的信息的函数。
PMC.C	该文件包括提供PM组件的核心函数，该文件定义的函数实现基本的分配和释放内存。

PMCE.C	该文件包括有错误检查的定义在PM C.C中的函数接口。

内存分配组件数据结构 Partition Memory Data Structures

Created Partition Memory List
Nucleus PLUS partition pools may be created and deleted dynamically. The Partition Memory Control Block (PCB) for each created partition memory pool is kept on a doublylinked, circular list. Newly created partition memory pools are placed at the end of the list, while deleted partition memory pools are completely removed from the list. The head pointer of this list is **PMD_Created_Pools_List.**
Created Partition Memory List Protection
Nucleus PLUS protects the integrity of the Created Partition Memory List from competing tasks and/or HISRs. This is done by using an internal protection structure called **PMD_List_Protect.** All partition memory creation and deletion is done under the protection of **PMD_List_Protect.**
Field Declarations
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
Field Summary
Field Description
tc_tcb_pointer Identifies the thread that currently has the protection.
tc_thread_waiting A flag indicating that one or more threads are waiting for the protection.
PCB PCB PCB PCB
PMD_Created_Pools_List

Total Partition Pools
The total number of currently created Nucleus PLUS partition memory pools is contained in the variable **PMD_Total_Pools.** The contents of this variable corresponds to the number of PCBs on the created list. Manipulation of this variable is also done under the protection of **PMD_List_Protect.**
Available Partitions List
The Available Partitions List is a singly-linked **NULL** terminated list, which contains the available partitions. The PCB contains pointers to the starting address of the list as well as the next available partition in the list. Allocated partitions are removed from the front of the list and deallocated partitions are place at the front of the list. Each partition has a
header block that links the partitions together.ports this field is not used.
Partition Pool Control Block
The Partition Memory Pool Control Block **PM_PCB** contains the starting address of the current memory pool and other fields necessary for processing partition pool requests.
Field Declarations
CS_NODE pm_created
UNSIGNED pm_id
CHAR pm_name[NU_MAX_NAME]
VOID *pm_start_address
UNSIGNED pm_pool_size
UNSIGNED pm_partition_size
UNSIGNED pm_available
UNSIGNED pm_allocated

struct PM_HEADER_STRUCT *pm_available_list
DATA_ELEMENT pm_fifo_suspend
DATA_ELEMENT pm_padding[PAD_1]
UNSIGNED pm_tasks_waiting
struct PM_SUSPEND_STRUCT *pm_suspension_list
PCB PM_HEADER 0
PM_HEADER 1
PM_HEADER n
NULL
pm_start_address
pm_available_list

Field Summary
Field Description
pm_created This is the link node structure for partition memory pools. It is linked into the created partition pools list, which is a doubly-linked, circular list.
pm_id This holds the internal partition memory pool identification of 0x50415254 which is equivalent to ASCII PART.
pm_name This is the user-specified, 8 character name for the partition memory pool.
***pm_start_address** This is the starting address of the current partition memory pool.
pm_pool_size Holds the size of the partition memory pool.
pm_partition_size This is the size of the current memory pool partition.
pm_available This is the number of partitions available for use in the current memory pool.
pm_allocated Holds the number of allocated partitions.
***pm_available_list** This is the list of available partitions of the current memory pool.
pm_fifo_suspend A flag that determines whether tasks suspend in fifo or priority order.
pm_padding This is used to align the partition memory pool structure on an even boundary. In some ports this field is not used.
pm_tasks_waiting Indicates the number of tasks that are currently suspended on a partition memory pool.
***pm_suspension_list** The head pointer of the partition memory pool suspension list. If no tasks are suspended, this pointer is **NULL**.

Partition Memory Pool Header Structure
The partition header structure **PM_HEADER** is placed at the beginning of each available partition. Each header contains a pointer to the next available partition, except for the last partition, which points to a null terminator. Each partition header also contains a pointer to its PCB (Partition Memory Pool Control Block).
Field Declarations
struct PM_HEADER_STRUCT *pm_next_available
PM_PCB *pm_partition_pool
Field Summary
Field Description
***pm_next_available** A pointer to the next partition in the available list.

pm_partition_pool A pointer to this partition's PCB.

Partition Memory Pool Suspension Structure

Tasks can suspend on empty and full partition memory pool conditions. During the suspension process a **PM_SUSPEND** structure is built. This structure contains information about the task and the task's partition pool request at the time of suspension. This suspension structure is linked to the PCB in a doubly-linked, circular list and is allocated from the suspending task's stack. There is one suspension block for every task suspended on the partition memory pool.

The suspension block's position on the suspend list is determined at partition pool creation. If a FIFO suspension was selected, the suspension block is added to the end of the list. Otherwise, if priority suspension was selected, the suspension block is placed after suspension blocks with tasks of equal or higher priority.

Partition Memory Pool Header Structure

The partition header structure **PM_HEADER** is placed at the beginning of each available partition. Each header contains a pointer to the next available partition, except for the last partition, which points to a null terminator. Each partition header also contains a pointer to its PCB (Partition Memory Pool Control Block).

Field Declarations

struct PM_HEADER_STRUCT *pm_next_available

PM_PCB *pm_partition_pool

Field Summary

Field Description

pm_next_available A pointer to the next partition in the available list.

pm_partition_pool A pointer to this partition's PCB.

Partition Memory Pool Suspension Structure

Tasks can suspend on empty and full partition memory pool conditions. During the suspension process a **PM_SUSPEND** structure is built. This structure contains information about the task and the task's partition pool request at the time of suspension. This suspension structure is linked to the PCB in a doubly-linked, circular list and is allocated from the suspending task's stack. There is one suspension block for every task suspended on the partition memory pool.

The suspension block's position on the suspend list is determined at partition pool creation. If a FIFO suspension was selected, the suspension block is added to the end of the list. Otherwise, if priority suspension was selected, the suspension block is placed after suspension blocks with tasks of equal or higher priority.

Field Declarations

CS_NODE pm_suspend_link

PM_PCB *pm_partiton_pool

TC_TCB *pm_suspended_task

VOID *pm_return_status

Partition Pool n

PCB

task 0

PM_SUSPEND

task 1

PM_SUSPEND

task 2

PM_SUSPEND

task n

PM_SUSPEND

Field Summary

Field Description
pm_suspend_link A link node structure for linking with other suspended blocks. It is used in a doubly-linked, circular suspension list.
*pm_partiton_pool A pointer to the partition memory pool structure.
*pm_suspended_task A pointer to the Task Control Block of the suspended task.
*pm_return_pointer The return memory address that has been requested.
pm_return_status The completion status of the task suspended on the partition pool.

内存分配组件函数 Partition Memory Functions

The following sections provide a brief description of the functions in the Partition Memory Component (PM). Review of the actual source code is recommended for further information.
PMC_Create_Partition_Pool
Creates a memory partition pool and then places it on the list of created partition pools.
语法 Syntax
```
STATUS PMC_Create_Partition_Pool(NU_PARTITION_POOL
*pool_ptr, CHAR *name,
VOID *start_address,
UNSIGNED pool_size,
UNSIGNED partition_size,
OPTION suspend_type)
```
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

PMC_Delete_Partition_Pool
This function deletes a memory partition pool and removes it from the list of created partition pools. All tasks suspended on the partition pool are resumed with the appropriate error status. Note that this function does not free any memory associated with either the pool area or the pool control block. That is the responsibility of the application.
语法 Syntax
```
STATUS PMC_Delete_Partition_Pool(NU_PARTITION_POOL *pool_ptr)
```
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Unprotect
PMC_Allocate_Partition

This function allocates a memory partition from the specified memory partition pool. If a memory partition is currently available, this function is completed immediately. Otherwise, if there are no partitions currently available, suspension is possible.

语法 Syntax

```
STATUS PMC_Allocate_Partition(NU_PARTITION_POOL *pool_ptr,
VOID **return_pointer,
UNSIGNED suspend)
```

函数调用 Functions Called

```
CSC_Place_On_List
CSC_Priority_Place_On_List
[HIC_Make_History_Entry]
TCC_Suspend_Task
TCC_Task_Priority
[TCT_Check_Stack]
TCT_Current_Thread
TCT_System_Protect
TCT_Unprotect
```

PMC_Deallocate_Partition

This function deallocates a previously allocated partition. If there is a task waiting for a partition, the partition is simply given to the waiting task and the waiting task is resumed. Otherwise, the partition is returned to the partition pool.

语法 Syntax

```
STATUS PMC_Deallocate_Partition(VOID *partition)
```

函数调用 Functions Called

```
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_System_Protect
TCT_Unprotect
```

PMC_Cleanup

This function is responsible for removing a suspension block from a partition pool. It is not called unless a timeout or a task terminate is in progress. Note that protection(the same as at suspension time) is already in effect.

语法 Syntax

```
VOID PMC_Cleanup(VOID *information)
```

函数调用 Functions Called

```
CSC_Remove_From_List
```

PMCE_Create_Partition_Pool

This function performs error checking on the parameters supplied to the create partition pool function.

语法 Syntax

```
STATUS PMCE_Create_Partition_Pool(NU_PARTITION_POOL
*pool_ptr, CHAR *name,
VOID *start_address,
UNSIGNED pool_size,
UNSIGNED partition_size,
OPTION suspend_type)
```

函数调用 Functions Called

```
PMC_Create_Partition_Pool
```

PMCE_Delete_Partition_Pool
This function performs error checking on the parameters supplied to the delete partition pool function.
语法 Syntax
STATUS PMCE_Delete_Partition_Pool(NU_PARTITION_POOL
*pool_ptr)
函数调用 Functions Called
PMC_Delete_Partition_Pool

PMCE_Allocate_Partition
This function performs error checking on the parameters supplied to the allocate partition function.
语法 Syntax
STATUS PMCE_Allocate_Partition(NU_PARTITION_POOL *pool_ptr,
VOID **return_pointer,
UNSIGNED suspend)
函数调用 Functions Called
PMC_Allocate_Partition
TCCE_Suspend_Error
PMCE_Deallocate_Partition
This function performs error checking on the parameters supplied to the deallocate partition function.
语法 Syntax
STATUS PMCE_Deallocate_Partition(VOID *partition)
函数调用 Functions Called
PMC_Deallocate_Partition
PMF_Established_Partition_Pools
This function returns the current number of established partition pools. Pools previously deleted are no longer considered established.
语法 Syntax
UNSIGNED PMF_Established_Partition_Pools(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

PMF_Partition_Pool_Pointers
Builds a list of pool pointers, starting at the specified location. The number of pool pointers placed in the list is equivalent to the total number of pools or the maximum number of pointers specified in the call.
语法 Syntax
UNSIGNED PMF_Partition_Pool_Pointers(NU_PARTITION_POOL
*pointer_list, UNSIGNED
maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
PMF_Partition_Pool_Information
This function returns information about the specified partition pool. However, if the supplied partition pool pointer is invalid, the function simply returns an error status.
语法 Syntax
STATUSPMF_Partition_Pool_Information(NU_PARTITION_POOL
*pool_ptr,
CHAR *name,

```
VOID **start_address,
UNSIGNED *pool_size,
UNSIGNED *partition_size,
UNSIGNED *available,
UNSIGNED *allocated,
OPTION *suspend_type,
UNSIGNED *tasks_waiting,
NU_TASK **first_task)
```

函数调用 Functions Called

```
[TCT_Check_Stack]
TCT_System_Protect
TCT_Unprotect
```

```
PMI_Initialize
This function initializes the data structures that control the operation of the Partition
Memory component. There are no partition pools initially.
```

语法 Syntax

```
VOID PMI_Initialize(VOID)
```

函数调用 Functions Called

None

动态内存管理组件 Dynamic Memory Component (DM)

动态内存管理组件(DM) 负责处理所以的Nucleus PLUS动态内存设备。一个Nucleus PLUS动态内存池包括用户指定数目的字节（byte）。内存池的位置是由应用程序指定的。当等待足够的动态内存时，任务有可能挂起。动态内存池可以被用户动态的创建和删除。请参考《*Nucleus PLUS Reference Manual*》获得更多的关于动态内存管理组件的信息。

动态内存管理组件文件 Dynamic Memory Files

动态内存管理组件(DM)包括七个文件。每一个I/O设备组件源码是如下定义的：

文件 File	描述 Description
DM_DEFS.H	该文件定义DM组件特定的数据结构和常量。
DM_EXTR.H	该文件定义DM组件的外部接口。
DMD.C	文件定义DM组件的全局数据结构。
DMI.C	该文件包括DM组件的初始化代码。
DMF.C	该文件包括提供DM组件的信息的函数。
DMC.C	该文件包括提供DM组件的核心函数，该文件定义的函数实现基本的分配和释放内存。
DMCE.C	该文件包括有错误检查的定义在DMC.C中的函数接口。

态内存管理组件数据结构 Dynamic Memory Data Structures

Created Dynamic Memory List

Nucleus PLUS dynamic memory pools may be created and deleted dynamically. The Dynamic Memory Control Block (PCB) for each created dynamic memory pool is kept on a doubly-linked, circular list. Newly created dynamic memory pools are placed at the end of the list, while deleted dynamic memory pools are completely removed from the list. The head pointer of this list is **DMD_Created_Pools_List.**

Created Dynamic Memory List Protection

Nucleus PLUS protects the integrity of the Created Dynamic Memory List from competing tasks and/or HISRs. This is done by using an internal protection structure called **DMD_List_Protect.** All dynamic memory creation and deletion is done under the protection of **DMD_List_Protect.**

Field Declarations

TC_TCB *tc_tcb_pointer

UNSIGNED tc_thread_waiting

Field Summary

Field Description

tc_tcb_pointer Identifies the thread that currently has the protection.

tc_thread_waiting A flag indicating that one or more threads are waiting for the protection.

PCB PCB PCB PCB

DMD_Created_Pools_List

Total Dynamic Pools

The total number of currently created Nucleus PLUS dynamic memory pools is contained in the variable **DMD_Total_Pools.** The contents of this variable corresponds to the number of PCBs on the created list. Manipulation of this variable is also done under the protection of **DMD_List_Protect.**

Available Memory List

The Available Memory List is a doubly-linked, **NULL** terminated, circular list, which contains the available dynamic memory blocks. The PCB contains pointers to the starting address of the list as well as the next available block in the list. A search pointer is also contained in the PCB. It linearly searches for and accumulates available memory blocks in order to fill memory requests. Allocated blocks are removed from the front of the list and deallocated blocks are placed back in the list at the point where they came from. Each block includes a header that links the various blocks together.

PCB DM_HEADER 0

DM_HEADER 1

DM_HEADER n

dm_start_address

dm_search_ptr

dm_memory_list

End of Pool

Dynamic Pool Control Block

The Dynamic Memory Pool Control Block **DM_PCB** contains the starting address of the current memory pool and other fields necessary for processing dynamic memory pool

requests.
Field Declarations
CS_NODE dm_created
TC_PROTECT dm_protect
UNSIGNED dm_id
CHAR dm_name[NU_MAX_NAME]
VOID *dm_start_address
UNSIGNED dm_pool_size
UNSIGNED dm_min_allocation
UNSIGNED dm_available
struct DM_HEADER_STRUCT *dm_memory_list struct
DM_HEADER_STRUCT *dm_search_ptr
DATA_ELEMENT dm_fifo_suspend
DATA_ELEMENT dm_padding[PAD_1]
UNSIGNED dm_tasks_waiting
struct DM_SUSPEND_STRUCT *dm_suspension_list

Field Summary
Field Description
dm_created This is the link node structure for dynamic memory pools. It is linked into the created dynamic pools list, which is a doubly-linked, circular list.
dm_protect A pointer to the protection structure for the dynamic memory pool.
dm_id This holds the internal dynamic memory pool identification of 0x44594E41, which is equivalent to ASCII DYNA.
dm_name This is the user-specified, 8 character name for the dynamic memory pool.
***dm_start_address** This is the starting address of the current dynamic memory pool.
dm_pool_size Holds the size of the dynamic memory pool.
dm_min_allocation The minimum number of bytes to be allocated in a block.
dm_available This is the total number of bytes available for use in the current memory pool.
***dm_memory_list** A list of the memory blocks in the current memory pool.
***dm_search_ptr** The search pointer used for locating a dynamic memory pool header.
dm_fifo_suspend A flag that determines whether tasks suspend in fifo or priority order.
dm_padding This is used to align the dynamic memory pool structure on an even boundary. In some ports this field is not used.
dm_tasks_waiting Indicates the number of tasks that are currently suspended on a dynamic memory pool.
***dm_suspension_list** The head pointer of the dynamic memory pool suspension list. If no tasks are suspended, this pointer is NULL.

Dynamic Memory Pool Header Structure

The dynamic header structure **DM_HEADER** is placed at the beginning of each available memory block. Each header contains pointers to both the next available

memory block and the previous available memory block. The last block's next pointer points to a null terminator. Each dynamic memory header also contains a pointer to its PCB.

Field Declarations

struct DM_HEADER_STRUCT *dm_next_memory
struct DM_HEADER_STRUCT *dm_previous_memory
DATA_ELEMENT dm_memory_free
DM_PCB *dm_memory_pool

Field Summary

Field Description

*dm_next_memory A pointer to the next memory block in the available list.
*dm_previous_memory A pointer to the previous memory block in the available list.
dm_memory_free A flag that indicates if the current memory block is free.
dm_memory_pool A pointer to the PCB which this memory block belongs to.

Dynamic Memory Pool Suspension Structure

Tasks can suspend on empty and full dynamic memory pool conditions. During the suspension process a DM_SUSPEND_STRUCT structure is built. This structure contains information about the task and the task' s dynamic pool request at the time of suspension. This suspension structure is linked onto the PCB in a doubly-linked, circular list and is allocated off of the suspending task' s stack. There is one suspension block for every task suspended on the dynamic memory pool.

The order of the suspension block placement on the suspend list is determined at dynamic pool creation. If a FIFO suspension was selected, the suspension block is added to the end of the list. Otherwise, if priority suspension was selected, the suspension block is placed after suspension blocks for tasks of equal or higher priority.

Field Declarations

CS_NODE dm_suspend_link
DM_PCB *dm_memory_pool
UNSIGNED dm_request_size
TC_TCB *dm_suspended_task
VOID *dm_return_pointer
STATUS dm_return_status

Field Summary

Field Description

dm_suspend_link A link node structure for linking with other suspended blocks. It is used in a doubly-linked, circular suspension list.
*dm_memory_pool A pointer to the dynamic memory pool structure.
dm_request_size Contains the size of the requested memory block.
*dm_suspended_task A pointer to the Task Control Block of the suspended task.
*dm_return_pointer The return memory address that has been requested.
dm_return_status The completion status of the task suspended on the dynamic pool.

Dynamic Pool n

PCB
task 0
DM_SUSPEND
task 1
DM_SUSPEND
task 2
DM_SUSPEND

task n
DM_SUSPEND

动态内存管理组件函数 Dynamic Memory Functions

以下的部分提供动态内存管理组件(DM)函数的主要信息。

DMC_Create_Memory_Pool
Creates a dynamic memory pool and then places it on the list of created dynamic memory pools. If the list does not exist, then this pool becomes the first item in the dynamic memory pools list.
STATUS DMC_Create_Memory_Pool(NU_MEMORY_POOL *pool_ptr,
CHAR *name,
VOID *start_address,
UNSIGNED pool_size, UNSIGNED
min_allocation,
OPTION suspend_type)
函数调用 Functions Called
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

DMC_Delete_Memory_Pool
This function deletes a dynamic memory pool and removes it from the list of created memory pools. All tasks suspended on the memory pool are resumed with the appropriate error status. Note that this function does not free any memory associated with either the pool area or the pool control block. That is the responsibility of the application.
语法 Syntax
STATUS DMC_Delete_Memory_Pool(NU_MEMORY_POOL *pool_ptr)
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Protect
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Unprotect

DMC_Allocate_Memory
This function allocates memory from the specified dynamic memory pool. If enough dynamic memory is currently available, this function is completed immediately. Otherwise, task suspension is possible.
语法 Syntax

```
STATUS DMC_Allocate_Memory(NU_MEMORY_POOL *pool_ptr, VOID
**return_pointer, UNSIGNED size,
UNSIGNED suspend)
```

函数调用 Functions Called

```
CSC_Place_On_List
[HIC_Make_History_Entry]
TCC_Suspend_Task
TCC_Task_Priority
[TCT_Check_Stack]
TCT_Current_Thread
TCT_Protect
TCT_Set_Suspend_Protect
TCT_System_Protect
TCT_Unprotect
TCT_Unprotect_Specific
```

DMC_Deallocate_Memory

This function deallocates a previously allocated dynamic memory block. The deallocated dynamic memory block is merged with any adjacent neighbors. This insures that there are no consecutive blocks of free memory in the pool, which makes the search easier. If there is a task waiting for dynamic memory, a determination of whether or not the request can now be satisfied is made after the deallocation is complete.

语法 Syntax

```
STATUS DMC_Deallocate_Memory(VOID *memory)
```

函数调用 Functions Called

```
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Protect
TCT_Unprotect
DMC_Cleanup
```

This function is responsible for removing a suspension block from a memory pool. It is not called unless a timeout or a task terminate is in progress. Note that protection (the same as at suspension time) is already in effect.

语法 Syntax

```
VOID DMC_Cleanup(VOID *information)
```

函数调用 Functions Called

```
CSC_Remove_From_List
```

DMCE_Create_Memory_Pool

This function performs error checking on the parameters supplied to the create dynamic memory pool function.

语法 Syntax

```
STATUS DMCE_Create_Memory_Pool(NU_MEMORY_POOL *pool_ptr,CHAR
*name, VOID
*start_address, UNSIGNED
pool_size, UNSIGNED
min_allocation, OPTION
```

suspend_type)
函数调用 Functions Called
DMC_Create_Memory_Pool

DMCE_Delete_Memory_Pool
This function performs error checking on the parameters supplied to the delete dynamic memory pool function.
语法 Syntax
STATUS DMCE_Delete_Memory_Pool(NU_MEMORY_POOL *pool_ptr)
函数调用 Functions Called
DMC_Delete_Memory_Pool

DMC_Allocate_Memory
This function allocates memory from the specified dynamic memory pool. If enough dynamic memory is currently available, this function is completed immediately. Otherwise, task suspension is possible.
语法 Syntax
STATUS DMC_Allocate_Memory(NU_MEMORY_POOL *pool_ptr, VOID
**return_pointer,
UNSIGNED size,
UNSIGNED suspend)
函数调用 Functions Called
CSC_Place_on_list
[HIC_Make_History_Entry]
TCC_Suspend_Task
TCC_Task_Priority
[TCT_Check_Stack]
TCT_Currtne_Thread
TCT_Protect
TCT_Set_Suspend_Protect
TCT_System_Protect
TCT_Unprotect
TCT_Unprotect_Specific

DMC_Deallocate_Memory
This function deallocates a previously allocated dynamic memory block. The deallocated dynamic memory block is merged with any adjacent neighbors. This insures that there are no consecutive blocks of free memory in the pool, which makes the search easier. If there is a task waiting for dynamic memory, a determination of whether or not the request can now be satisfied is made after the deallocation is complete.
语法 Syntax
STATUS DMC_Deallocate_Memory(VOID *memory)
函数调用 Functions Called
CSC_Remove_From_List
[HIC_Make_History_Entry]
TCC_Resume_Task
[TCT_Check_Stack]
TCT_Control_To_System
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Protect

TCT_Unprotect

DMC_Cleanup
This function is responsible for removing a suspension block from a memory pool. It is not called unless a timeout or a task terminate is in progress. Note that protection (the same as at suspension time) is already in effect.
语法 Syntax
VOID DMC_Cleanup(VOID *information)
函数调用 Functions Called
CSC_Remove_From_List

DMCE_Create_Memory_Pool
This function performs error checking on the parameters supplied to the create dynamic memory pool function.
语法 Syntax
STATUS DMCE_Create_Memory_Pool(NU_MEMORY_POOL *pool_ptr,
CHAR *name, VOID
*start_address, UNSIGNED
pool_size, UNSIGNED
min_allocation, OPTION
suspend_type)
函数调用 Functions Called
DMC_Create_Memory_Pool
DMCE_Delete_Memory_Pool
This function performs error checking on the parameters supplied to the delete dynamic memory pool function.
语法 Syntax
STATUS DMCE_Delete_Memory_Pool(NU_MEMORY_POOL *pool_ptr)
函数调用 Functions Called
DMC_Delete_Memory_Pool

DMCE_Allocate_Memory
This function performs error checking on the parameters supplied to the allocate memory function.
语法 Syntax
STATUS DMCE_Allocate_Memory(NU_MEMORY_POOL *pool_ptr, VOID
**return_pointer,
UNSIGNED size,
UNSIGNED suspend)
函数调用 Functions Called
DMC_Allocate_Memory
TCCE_Suspend_Error

DMCE_Deallocate_Memory
This function performs error checking on the parameters supplied to the deallocate memory function.
语法 Syntax
STATUS DMCE_Deallocate_Memory(VOID *memory)
函数调用 Functions Called
DMC_Deallocate_Memory
DMF_Established_Memory_Pools

Returns the current number of established memory pools. Pools previously deleted are no longer considered established.

语法 Syntax

```
UNSIGNED DMF_Established_Memory_Pools(VOID)
```

函数调用 Functions Called

[TCT_Check_Stack]

DMF_Memory_Pool_Pointers

Builds a list of pool pointers, starting at the specified location. The number of pool pointers placed in the list is equivalent to the total number of pools or the maximum number of pointers specified in the call.

语法 Syntax

```
UNSIGNED DMF_Memory_Pool_Pointers(NU_MEMORY_POOL
**pointer_list, UNSIGNED
maximum_pointers)
```

函数调用 Functions Called

[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

DMF_Memory_Pool_Information

Returns information about the specified memory pool. However, if the supplied memory pool pointer is invalid, the function simply returns an error status.

语法 Syntax

```
STATUS DMF_Memory_Pool_Information(NU_MEMORY_POOL
*pool_ptr, CHAR *name,
VOID **start_address,
UNSIGNED *pool_size,
UNSIGNED*min_allocation,
UNSIGNED *available,
OPTION *suspend_type,
UNSIGNED
*tasks_waiting,
NU_TASK **first_task)
```

函数调用 Functions Called

[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

DMI_Initialize

This function initializes the data structures that control the operation of the Dynamic Memory component. There are no dynamic memory pools initially.

语法 Syntax

```
VOID DMI_Initialize(VOID)
```

函数调用 Functions Called

None

I/O设备组件 Input/Output Driver Component (IO)

I/O设备组件(IO) 负责处理所以的Nucleus PLUS I/O设备。Nucleus PLUS I/O设备组件提供标准的I/O设备

接口：初始化、分配、释放、输入、输出、状态和停止。该接口是通过一个通用的控制结构。它允许应用程序以相似甚至是一样的方式处理多样的外部设备。任务在等待一个外部设备时可以被挂起。I/O设备可以被用户动态地创建和删除。请参考《*Nucleus PLUS Reference Manual*》获得更多的关于历史组件的信息。

I/O设备组件文件 Input/Output Driver Files

I/O设备组件(IO)包括七个文件。每一个I/O设备组件源码是如下定义的：

文件 File	描述 Description
IO_DEFS.H	该文件定义IO组件特定的数据结构和常量。
IO_EXTR.H	该文件定义IO组件的外部接口。
IOD.C	文件定义IO组件的全局数据结构。
IOI.C	该文件包括IO组件的初始化代码。
IOF.C	该文件包括提供IO组件的信息的函数。
IOC.C	该文件包括提供IO组件的核心函数，该文件定义的函数实现基本的输入和输出。
IOCE.C	该文件包括有错误检查的定义在IOC.C中的函数接口。

I/O设备组件数据结构 Input/Output Data Structures

Created Input/Output List
Nucleus PLUS input/output drivers may be created and deleted dynamically. The Input/Output Control Block **(NU_DRIVER)** for each created input/output driver is kept on a doubly-linked, circular list. Newly created input/output drivers are placed at the end of the list, while deleted input/output drivers are completely removed from the list. The head pointer of this list is **IOD_Created_Drivers_List.**
Input/Output Driver Control Block
The Input/Output Driver Control Block **(NU_DRIVER)** contains the entry function of the current driver and other fields necessary for processing input/output driver requests.
Field Declarations
UNSIGNED words [NU_DRIVER_SIZE]
CHAR nu_driver_name[NU_MAX_NAME]
VOID *nu_info_ptr
UNSIGNED nu_driver_id
VOID (*nu_driver_entry)(struct NU_DRIVER_STRUCT*,
NU_DRIVER_REQUEST *)
Field Summary
Field Description
words This is the link node structure for I/O drivers. It is linked into the created I/O drivers list, which is a doubly-linked, circular list.
nu_driver_name This is the user-specified, 8-character name for the I/O driver.
***nu_info_ptr** A pointer to the users structure.

nu_driver_id This holds the internal I/O driver identification of **0x494F4452,** which is an equivalent to ASCII IODR.
(*nu_driver_entry) This is the I/O drivers entry function.
NU_DRIVER NU_DRIVER NU_DRIVER NU_DRIVER
IOD_Created_Drivers_List

Created Input/Output List Protection

Nucleus PLUS protects the integrity of the Created Input/Output List from competing tasks and/or HISRs. This is done by using an internal protection structure called **IOD_List_Protect.** All input/output creation and deletion is done under the protection of IOD_List_Protect.
Field Declarations
TC_TCB *tc_tcb_pointer
UNSIGNED tc_thread_waiting
Field Summary
Field Description
tc_tcb_pointer Identifies the thread that currently has the protection.
tc_thread_waiting A flag indicating that one or more threads are waiting for the protection.

Total Input/Output Drivers

The total number of currently created Nucleus PLUS input/output drivers is contained in the variable **IOD_Total_Drivers.** The contents of this variable corresponds to the number of **NU_DRIVER**s on the created list. Manipulation of this variable is also done under the protection of **IOD_List_Protect.**
Input/Output Driver Request Structure
The input/output driver request structure **NU_DRIVER_REQUEST** is responsible for passing necessary information to and from a created I/O driver. The type of information in the request is specified by the **nu_function** field in the request structure. Of course, the exact interpretation of this structure depends on the specific driver.
Field Declarations
INT nu_function
UNSIGNED nu_timeout
STATUS nu_status
UNSIGNED nu_supplemental
VOID nu_supplemental_ptr
union NU_REQUEST_INFO_UNION nu_request_info
Field Summary
Field Description
nu_function This is the I/O request function code. It can have one of 7 values depending on which request is desired:
NU_INITIALIZE 1
NU_ASSIGN 2
NU_RELEASE 3
NU_INPUT 4
NU_OUTPUT 5
NU_STATUS 6
NU_TERMINATE 7
nu_timeout Holds the timeout on request.
nu_status Contains the status of the request.
nu_supplemental Contains user supplied supplemental information.
***nu_supplemental_ptr** A pointer to the driver specific supplemental

information [optional].
nu_request_info A union of the structures that are used for driver requests. These requests include initialization, assign, release, input, output, status and terminate.

Input/Output Driver Initialization Requests

I/O drivers initialization requests are made using the initialization structure **NU_INITIALIZE_STRUCT.** This structure contains information about the driver's base address and the driver's interrupt vector. This request is designated with an **NU_INITIALIZE** value in the nu_function field of the **NU_DRIVER_REQUEST** structure.

Field Declarations

VOID *nu_io_address
UNSIGNED nu_logical_units
VOID *nu_memory
INT nu_vector

Field Summary

Field Description

***nu_io_address** A pointer to the base I/O address of the driver.
nu_logical_units Contains the number of logical units in the driver.
***nu_memory** A generic memory pointer.
nu_vector Contains the interrupt vector number of the driver.

Input/Output Driver Assignment Requests

I/O driver assignment requests are made using the **NU_ASSIGN_STRUCT** structure. This request is designated with an **NU_ASSIGN** value in the **nu_function** field of the NU_DRIVER_REQUEST structure.

Field Declarations

UNSIGNED nu_logical_unit
INT nu_assign_info

Field Summary

Field Description

nu_logical_unit Contains the I/O driver's logical unit number.
nu_assign_info This variable is used for additional I/O driver assign information.

Input/Output Driver Release Requests

I/O driver release requests are made using the **NU_RELEASE_STRUCT** structure. This request is designated with an **NU_RELEASE** value in the **nu_function** field of the NU_DRIVER_REQUEST structure.

Field Declarations

UNSIGNED nu_logical_unit
INT nu_release_info

Field Summary

Field Description

nu_logical_unit Contains the I/O driver's logical unit number.
nu_assign_info This variable is used for additional I/O driver release information.

Input/Output Driver Input Requests

I/O driver inputs are made using the **NU_INPUT_STRUCT** structure. This structure contains information about data sent to the driver for processing. This request is designated with an **NU_INPUT** value in the nu_function field of the NU_DRIVER_REQUEST structure.

Field Declarations

UNSIGNED nu_logical_unit

UNSIGNED nu_offset
UNSIGNED nu_request_size
UNSIGNED nu_actual_size
Void *nu_buffer_ptr
Field Summary
Field Description
nu_logical_unit Contains the I/O driver's logical unit number.
nu_offset An I/O offset used as an offset from the device base I/O address or an offset into the input buffer.
nu_request_size The requested size of the I/O driver input data.
nu_actual_size The actual size of the I/O driver input data.
***nu_buffer_ptr** A pointer to the I/O driver input data buffer.

Input/Output Driver Output Requests

I/O driver output requests are made using the **NU_OUTPUT_STRUCT** structure. This structure contains information about data received from the driver. This request is designated with an **NU_OUTPUT** value in the **nu_function** field of the NU_DRIVER_REQUEST structure.
Field Declarations
UNSIGNED nu_logical_unit
UNSIGNED nu_offset
UNSIGNED nu_request_size
UNSIGNED nu_actual_size
VOID *nu_buffer_ptr
Field Summary
Field Description
nu_logical_unit Contains the I/O driver's logical unit number.
nu_offset An I/O offset used as an offset from the device base I/O address or an offset into the output buffer.
nu_request_size The requested size of the I/O driver output data.
nu_actual_size The actual size of the I/O driver output data.
***nu_buffer_ptr** A pointer to the I/O driver output buffer.
Input/Output Driver Status Requests
I/O driver status requests are made using the **NU_STATUS_STRUCT** structure. This request is designated with an **NU_STATUS** value in the **nu_function** field of the NU_DRIVER_REQUEST structure.
Field Declarations
UNSIGNED nu_logical_unit
VOID *nu_extra_status
Field Summary
Field Description
nu_logical_unit Contains the I/O driver's logical unit number.
***nu_extra_status** A pointer to additional status information.

Input/Output Driver Terminate Requests

I/O driver terminate requests are made using the **NU_TERMINATE_STRUCT** structure. This request is designated with an **NU_TERMINATE** value in the nu_function field of the **NU_DRIVER_REQUEST** structure.
Field Declarations
UNSIGNED nu_logical_unit
Field Summary
Field Description
nu_logical_unit Contains the I/O driver's logical unit number.

I/O设备组件函数 Input/Output Driver Functions

以下的部分提供I/O设备组件(IO)函数的主要信息。回顾实际的源代码可以获得更多的信息。

IOC_Create_Driver

该函数创建一个I/O设备并且将其放置在已创建I/O设备链表。注意该函数不会实际地调用该设备。

语法 Syntax

```
STATUS IOC_Create_Driver (NU_DRIVER *driver, CHAR *name,
VOID (*driver_entry)
(NU_DRIVER *, NU_DRIVER_REQUEST *))
```

函数调用 Functions Called

```
CSC_Place_On_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
```

IOC_Delete_Driver

该函数删除一个I/O设备并且将其移出已创建I/O设备链表。注意该函数不会实际地调用该设备。

语法 Syntax

```
STATUS IOC_Delete_Driver(NU_DRIVER *driver)
```

函数调用 Functions Called

```
CSC_Remove_From_List
[HIC_Make_History_Entry]
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect
```

IOC_Request_Driver

该函数发送一个用户请求到特定的I/O设备。

语法 Syntax

```
STATUS IOC_Request_Driver(NU_DRIVER *driver,NU_DRIVER_REQUEST *request)
```

函数调用 Functions Called

```
[HIC_Make_History_Entry]
[TCT_Check_Stack]
```

IOC_Resume_Driver

继续先前在I/O设备中挂起的任务。通常该函数是被一个I/O设备调用。

语法 Syntax

```
STATUS IOC_Resume_Driver(NU_TASK *task)
```

函数调用 Functions Called

```
[HIC_Make_History_Entry]
TCC_Resume_Task
TCT_Control_To_System
[TCT_Check_Stack]
TCT_Get_Current_Protect
TCT_Set_Current_Protect
TCT_System_Protect
TCT_System_Unprotect
TCT_Unprotect
TCT_Unprotect_Specific
```

IOC_Suspend_Driver
该函数挂起一个在I/O设备的任务。It is the responsibility of the I/O driver to keep track of tasks waiting inside an I/O driver.
语法 Syntax
STATUS IOC_Suspend_Driver(VOID (*terminate_routine)
(VOID *), VOID *information,
UNSIGNED timeout)
函数调用 Functions Called
[HIC_Make_History_Entry]
TCC_Suspend_Task
TCT_Current_Thread
[TCT_Check_Stack]
TCT_Get_Current_Protect
TCT_Set_Suspend_Protect
TCT_System_Protect
TCT_Unprotect_Specific

IOCE_Create_Driver
该函数提供参数错误检查的创建一个I/O设备函数。
语法 Syntax
STATUS IOCE_Create_Driver(NU_DRIVER *driver, CHAR *name,
VOID (*driver_entry)
(NU_DRIVER*,NU_DRIVER_REQUEST*))
函数调用 Functions Called
IOC_Create_Driver

IOCE_Delete_Driver
该函数提供参数错误检查的删除一个I/O设备函数。
语法 Syntax
STATUS IOCE_Delete_Driver(NU_DRIVER *driver)
函数调用 Functions Called
IOC_Delete_Driver

IOCE_Request_Driver
This function performs error checking on the parameters supplied to the I/O driver request function.
语法 Syntax
STATUS IOCE_Request_Driver(NU_DRIVER *driver,
NU_DRIVER_REQUEST *request)
函数调用 Functions Called
IOC_Request_Driver
IOCE_Resume_Driver
This function performs error checking on the parameters supplied to the I/O driver resume function.
语法 Syntax
STATUS IOCE_Resume_Driver(NU_TASK *task)
函数调用 Functions Called
IOC_Resume_Driver
TCCE_Validate_Resume

IOCE_Suspend_Driver

This function performs error checking on the parameters supplied to the I/O driver suspend function.
语法 Syntax
STATUS IOCE_Suspend_Driver(VOID (*terminate_routine)
(VOID*), VOID *information,
UNSIGNED timeout)
函数调用 Functions Called
IOC_Suspend_Driver
TCCE_Suspend_Error
IOF_Established_Drivers
Returns the current number of established I/O drivers. I/O drivers previously deleted are no longer considered established.
语法 Syntax
UNSIGNED IOF_Established_Drivers(VOID)
函数调用 Functions Called
[TCT_Check_Stack]

IOF_Driver_Pointers
Builds a list of driver pointers, starting at the specified location. The number of driver pointers placed in the list is equivalent to the total number of drivers or the maximum number of pointers specified in the call.
语法 Syntax
UNSIGNED IOF_Driver_Pointers(NU_DRIVER **pointer_list,
UNSIGNED maximum_pointers)
函数调用 Functions Called
[TCT_Check_Stack]
TCT_Protect
TCT_Unprotect

IOI_Initialize
该函数初始化控制I/O设备组件操作的数据结构，没有I/O设备被初始化。
语法 Syntax
VOID IOI_Initialize(VOID)
函数调用 Functions Called
None

历史组件 History Component (HI)

历史组件(HI) 负责处理所以的Nucleus PLUS 历史信息。Nucleus PLUS历史组件维护一个循环的关于大量系统活动信息的记录。应用任务和HISRs能产生历史记录的调目。每一个在历史记录的调目包括特定的特定的Nucleus PLUS服务调用和被调用的信息。请参考《*Nucleus PLUS Reference Manual*》获得更多的关于历史组件的信息。

历史组件文件 History Files

错误处理组件(ER)包括五个文件。每一个错误处理组件源码是如下定义的：

文件 File	描述 Description
HI_DEFS.H	该文件定义HI组件特定的数据结构和常量。
HI_EXTR.H	该文件定义HI组件的外部接口。
HIC.C	该文件包括提供HI组件的核心函数，该文件定义的函数实现基本的允许和禁止一个历史记录。
HID.C	这个文件包括HI用到的全局数据结构。
HII.C	该文件包括HI组件的初始化代码。

历史组件数据结构 History Data Structures

历史使能 History Enable

Nucleus PLUS 历史条目可以被动态地创建。历史使能标记指示历史记录是否可用。如果该值为NU_FALSE, 历史记录是禁止的。否则，历史记录是使能的，而且一个适当的条目将在历史记录中被创建。

写索引 Write Index

The index of the next entry into the Nucleus PLUS History table is contained in the variable **HID_Write_Index.**该变量的内容 The contents of this variable corresponds to the location of the index of the next available entry in the History table. Manipulation of this variable is also done under the protection of **HID_History_Protect.**

读索引 Read Index

The index of the oldest entry into the Nucleus PLUS History table is contained in the variable **HID_Read_Index.** The contents of this variable corresponds to the location of the index of the oldest entry in the History table. Manipulation of this variable is also done under the protection of **HID_History_Protect.**

历史表保护 History Table Protection

Nucleus PLUS protects the integrity of the History Table from competing tasks and/or HISRs. This is done by using an internal protection structure called **HID_History_Protect.** All History enabling and disabling is done under the protection of HID_History_Protect.

Field Declarations

TC_TCB *tc_tcb_pointer

UNSIGNED tc_thread_waiting

Field Summary

Field Declarations

tc_tcb_pointer Identifies the thread that currently has the protection.

tc_thread_waiting A flag indicating that one or more threads are waiting for the protection.

Total Entries

The total number of entries in the Nucleus PLUS History Table is contained in the variable **HID_Entry_Count.** The contents of this variable corresponds to the number of valid entries in the History table. Manipulation of this variable is also done under the protection of HID_History_Protect.

历史表结构 History Table Structure

历史表结构HI_HISTORY_ENTRY包括当前历史条目和其他处理历史请求必需字段的开始索引。

成员变量的声明
DATA_ELEMENT hi_id
DATA_ELEMENT hi_caller
UNSIGNED hi_param1
UNSIGNED hi_param2
UNSIGNED hi_param3
UNSIGNED hi_time
VOID *hi_thread
成员变量的概要

Field	Description
hi_id	This is the index in the table for History entries. It is a simple array consisting only of HI_HISTORY_ENTRY structures
hi_caller	The entity that made the entry into the History log. This can be a task, a HISR or the initialization process.
hi_param1	The first parameter for storing logged history information
hi_param2	The second parameter for storing logged history information.
hi_param3	The third parameter for storing logged history information.
hi_time	The current system time in clock ticks.
*hi_thread	A pointer to the calling thread.

历史组件函数 History Functions

以下的部分提供历史组件(HI)函数的主要信息。回顾实际的源代码可以获得更多的信息。
HIC_Disable_History_Saving
This function disables the history saving function.
语法 Syntax
VOID HIC_Disable_History_Saving(VOID)
函数调用 Functions Called
TCT_Protect
TCT_Unprotect

HIC_Enable_History_Saving
该函数禁止历史记录功能。
语法 Syntax
VOID HIC_Enable_History_Saving(VOID)
函数调用 Functions Called
TCT_Protect
TCT_Unprotect

HIC_Make_History_Entry_Service
该函数在历史表中创建一个应用程序条目。
语法 Syntax
VOID HIC_Make_History_Entry_Service(UNSIGNED param1, UNSIGNED param2, UNSIGNED param3)
函数调用 Functions Called
HIC_Make_History_Entry

HIC_Make_History_Entry

该函数在历史表的下一个可用的位置中创建一个条目(历史记录是使能的)。
语法 Syntax
VOID HIC_Make_History_Entry(DATA_ELEMENT id, UNSIGNED param1,UNSIGNED param2, UNSIGNED param3)
函数调用 Functions Called
TCC_Current_HISR_Pointer
TCC_Current_Task_Pointer
TCT_Get_Current_Protect
TCT_Protect
TCT_Set_Current_Protect
TCT_Unprotect
TCT_Unprotect_Specific
TMT_Retrieve_Clock

HIC_Retrieve_History_Entry
该函数在历史表的下一个最早的条目。如果条目是可用的，一个错误状态被返回。
STATUS HIC_Retrieve_History_Entry(DATA_ELEMENT*id,
UNSIGNED *param1, UNSIGNED
*param2, UNSIGNED *param3,
UNSIGNED *time, NU_TASK
**task, NU_HISR **hisr)
函数调用 Functions Called
TCT_Protect
TCT_Unprotect

HII_Initialize
该函数初始化历史组件操作的数据结构。
语法 Syntax
VOID HII_Initialize(VOID)
函数调用 Functions Called
None

错误处理组件 Error Component (ER)

许可证控制组件 (ER) 负责处理所以的Nucleus PLUS 系统错误。Nucleus PLUS版本错误处理组件是专注于处理系统致命错误的组件。当系统出现一个致命错误时，系统控制权被传送给错误处理组件。然后创建一个特定的ASCII错误消息。该消息帮助用户识别错误类型。最后系统陷入一个死循环。请参考《*Nucleus PLUS Reference Manual* 》获得更多的关于版本控制的信息。

错误处理组件文件 Error Files

错误处理组件(ER)包括四个文件。每一个错误处理组件源码是如下定义的：

文件 File	描述 Description
ER_EXTR.H	该文件定义ER组件的外部接口。

ERC.C	该文件包括提供ER组件的核心函数，该文件定义的函数实现系统基本的处理错误。
ERD.C	文件定义ER组件的全局数据结构。
ERI.C	该文件包括ER组件的初始化代码。

错误处理组件数据结构 Error Data Structures

错误代号 Error Codes

Nucleus PLUS错误是通过使用错误代号来察觉。当系统确定到一个错误发生，系统使用一个错误代号来确定错误类型。错误代号被放置到变量ERD_Error_Code 中。Nucleus PLUS错误代号被列在下表。

Code	Constant	Description
1	NU_ERROR_CREATING_TIMER_HISR	创建定时器HISR发生错误
2	NU_ERROR_CREATING_TIMER_TASK	创建定时器任务发生错误
3	NU_STACK_OVERFLOW	一个任务或HISR堆栈溢出错误
4	NU_UNHANDLED_INTERRUPT	在LISR注册之前，中断发生的错误

Error String

Nucleus PLUS 以一个ASCII字符串模式报告错误信息。这个字符串被保存在变量ERD_Error_String中。当一个错误发生时，系统报告的该错误代号的ASCII字符串解释包括在这个变量中。串仅仅在条件编译标记NU_ERROR_STRING被ERD.C、ERI.C和ERC.C包括时，该字符才被生成。

错误处理组件函数 Error Functions

以下的部分提供版本控制组件(ER)函数的主要信息。回顾实际的源代码可以获得更多的信息。

ERC_System_Error

该函数处理被大量系统组件确定的系统错误。通常这种错误类型被认为是致命的。

语法 Syntax

```
VOID ERC_System_Error(INT error_code)
```

函数调用 Functions Called

None

ERI_Initialize

该函数初始化错误处理组件数据结构。

语法 Syntax

```
VOID ERI_Initialize(VOID)
```

函数调用 Functions Called

None

许可证控制组件 Component (LI)

许可证控制组件 (LI) 负责处理所以的Nucleus PLUS 许可证信息。Nucleus PLUS版本控制组件是专注于存储和报告版本信息的组件。这些信息包括当前Nucleus PLUS软件版本和发表号。请参考《*Nucleus PLUS Reference Manual* 》获得更多的关于版本控制的信息。

许可证组件文件 License Files

许可证组件(LI)包括两个文件。每一个版本组件源码是如下定义的：

文件 File	描述 Description
LIC.C	该文件包括提供MB组件的核心函数，该文件定义的函数实现基本的系统许可证报道服务。
LID.C	文件定义LI组件的全局数据结构。

许可证组件数据结构 License Data Structures

License String

Nucleus PLUS 以一个ASCII字符串模式报告许可证信息。这个字符串被保存在变量`LID_License_String`中。这个变量包括用户许可证信息和用户序列号。.

许可证组件函数 License Functions

以下的部分提供版本控制组件(LI)函数的主要信息。回顾实际的源代码可以获得更多的信息。

LIC_License_Information

该函数返回指向许可证信息字符串的指针。字符串的信息包含Nucleus PLUS的用户和产品许可证信息。

语法 Syntax

```
CHAR *LIC_License_Information(VOID)
```

函数调用 Functions Called

None

版本控制组件 Release Component (RL)

版本控制组件 (RL) 负责处理所以的Nucleus PLUS 版本信息。Nucleus PLUS版本控制组件是专注于存储和报告版本信息的组件。这些信息包括当前Nucleus PLUS软件版本和发表号。请参考《*Nucleus PLUS Reference Manual* 》获得更多的关于版本控制的信息。

版本控制文件 Release Files

版本控制组件(RL)包括两个文件。每一个版本组件源码是如下定义的：

文件 File	描述 Description
RLC.C	该文件包括提供MB组件的核心函数，该文件定义的函数实现基本的系统版本报告服务。
RLD.C	文件定义RL组件的全局数据结构。

版本控制数据结构 Release Data Structures

版本字符串 Release String
Nucleus PLUS 以一个ASCII 字符串的形式报告版本信息。这个字符串被保存在变量RLD_Release_String中。这个变量包括Nucleus PLUS软件当前信息的描述。

特殊字符串 Special String
Nucleus PLUS 在一个ASCII 字符串中报告多种信息。这个字符串被保存在变量RLD_Special_String中，这个字符串包括Nucleus PLUS 系统的起源信息。

版本控制函数 Release Functions

以下的部分提供版本控制组件(RL)函数的主要信息。回顾实际的源代码可以获得更多的信息。

RLC_Release_Information
该函数返回指向版本信息字符串的指针。字符串的信息标识当前Nucleus PLUS的版本。

语法 Syntax
CHAR *RLC_Release_Information(VOID)
函数调用 Functions Called
None

附录A Nucleus PLUS 常数

该附录包括所有的在本书第四章(Nucleus PLUS Services)定义的Nucleus PLUS 常数。这些常数是先按字母顺序和按值的大小依次排列的。
Nucleus PLUS Constants (Alphabetical)

Name Decimal Value Hex Value
NU_ALLOCATE_MEMORY_ID 47 2F
NU_ALLOCATE_PARTITION_ID 43 2B
NU_AND 2 2
NU_AND_CONSUME 3 3
NU_BROADCAST_TO_MAILBOX_ID 16 10
NU_BROADCAST_TO_PIPE_ID 30 1E
NU_BROADCAST_TO_QUEUE_ID 23 17
NU_CHANGE_PREEMPTION_ID 11 B
NU_CHANGE_PRIORITY_ID 10 A
NU_CHANGE_TIME_SLICE_ID 65 41
NU_CONTROL_SIGNALS_ID 49 31
NU_CONTROL_TIMER_ID 58 3A
NU_CREATE_DRIVER_ID 60 3C
NU_CREATE_EVENT_GROUP_ID 37 25
NU_CREATE_HISR_ID 54 36
NU_CREATE_MAILBOX_ID 12 C
NU_CREATE_MEMORY_POOL_ID 45 2D
NU_CREATE_PARTITION_POOL_ID 41 29
NU_CREATE_PIPE_ID 25 19
NU_CREATE_QUEUE_ID 18 12
NU_CREATE_SEMAPHORE_ID 32 20
NU_CREATE_TASK_ID 2 2
NU_CREATE_TIMER_ID 56 38
NU_DEALLOCATE_MEMORY_ID 48 30
NU_DEALLOCATE_PARTITION_ID 44 2C
NU_DELETE_DRIVER_ID 61 3D
NU_DELETE_EVENT_GROUP_ID 38 26
NU_DELETE_HISR_ID 55 37
NU_DELETE_MAILBOX_ID 13 D
NU_DELETE_MEMORY_POOL_ID 46 2E
NU_DELETE_PARTITION_POOL_ID 42 2A
NU_DELETE_PIPE_ID 26 1A
NU_DELETE_QUEUE_ID 19 13
NU_DELETE_SEMAPHORE_ID 33 21
NU_DELETE_TASK_ID 3 3
NU_DELETE_TIMER_ID 57 39
NU_DISABLE_INTERRUPTS [Port Specific]
NU_DISABLE_TIMER 4 4
NU_DRIVER_SUSPEND 10 A
NU_ENABLE_INTERRUPTS [Port Specific]
NU_ENABLE_TIMER 5 5
NU_END_OF_LOG -1 FFFFFFFF
NU_EVENT_SUSPEND 7 7
NU_FALSE 0 0
NU_FIFO 6 6
NU_FINISHED 11 B
NU_FIXED_SIZE 7 7
NU_GROUP_DELETED -2 FFFFFFFE
NU_INVALID_DELETE -3 FFFFFFFD
NU_INVALID_DRIVER -4 FFFFFFFC
NU_INVALID_ENABLE -5 FFFFFFFB
NU_INVALID_ENTRY -6 FFFFFFFA
NU_INVALID_FUNCTION -7 FFFFFFF9
NU_INVALID_GROUP -8 FFFFFFF8

```
NU_INVALID_HISR -9 FFFFFFF7
NU_INVALID_MAILBOX -10 FFFFFFF6
NU_INVALID_MEMORY -11 FFFFFFF5
NU_INVALID_MESSAGE -12 FFFFFFF4
NU_INVALID_OPERATION -13 FFFFFFF3
NU_INVALID_PIPE -14 FFFFFFF2
NU_INVALID_POINTER -15 FFFFFFF1
NU_INVALID_POOL -16 FFFFFFF0
NU_INVALID_PREEMPT -17 FFFFFFEF
NU_INVALID_PRIORITY -18 FFFFFFEE
NU_INVALID_QUEUE -19 FFFFFFED
NU_INVALID_RESUME -20 FFFFFFEC
NU_INVALID_SEMAPHORE -21 FFFFFFEB
NU_INVALID_SIZE -22 FFFFFFEA
NU_INVALID_START -23 FFFFFFE9
NU_INVALID_SUSPEND -24 FFFFFFE8
NU_INVALID_TASK -25 FFFFFFE7
NU_INVALID_TIMER 3 FFFFFFE6
NU_INVALID_VECTOR -27 FFFFFFE5
NU_MAILBOX_DELETED -28 FFFFFFE4
NU_MAILBOX_EMPTY -29 FFFFFFE3
NU_MAILBOX_FULL -30 FFFFFFE2
NU_MAILBOX_RESET -31 FFFFFFE1
NU_MAILBOX_SUSPEND 3 3
NU_MEMORY_SUSPEND 9 9
NU_NO_MEMORY -32 FFFFFFE0
NU_NO_MORE_LISRS -33 FFFFFFDF
NU_NO_PARTITION -34 FFFFFFDE
NU_NO_PREEMPT 8 8
NU_NO_START 9 9
NU_NO_SUSPEND 0 0
NU_NOT_DISABLED -35 FFFFFFDD
NU_NOT_PRESENT -36 FFFFFFDC
NU_NOT_REGISTERED -37 FFFFFFDB
NU_NOT_TERMINATED -38 FFFFFFDA
NU_NULL 0 0
NU_OBTAIN_SEMAPHORE_ID 35 23
NU_OR 0 0
NU_OR_CONSUME 1 1
NU_PARTITION_SUSPEND 8 8
NU_PIPE_DELETED -39 FFFFFFD9
NU_PIPE_EMPTY -40 FFFFFFD8
NU_PIPE_FULL -41 FFFFFFD7
NU_PIPE_RESET -42 FFFFFFD6
NU_PIPE_SUSPEND 5 5
NU_POOL_DELETED -43 FFFFFFD5
NU_PREEMPT 10 A
NU_PRIORITY 11 B
NU_PURE_SUSPEND 1 1
NU_QUEUE_DELETED -44 FFFFFFD4
NU_QUEUE_EMPTY -45 FFFFFFD3
NU_QUEUE_FULL -46 FFFFFFD2
NU_QUEUE_RESET -47 FFFFFFD1
NU_QUEUE_SUSPEND 4 4
NU_READY 0 0
```

```
NU_RECEIVE_FROM_MAILBOX_ID 17 11
NU_RECEIVE_FROM_PIPE_ID 31 1F
NU_RECEIVE_FROM_QUEUE_ID 24 18
NU_RECEIVE_SIGNALS_ID 50 32
NU_REGISTER_LISR_ID 53 35
NU_REGISTER_SIGNAL_HANDLER_ID 51 33
NU_RELEASE_SEMAPHORE_ID 36 24
NU_RELINQUISH_ID 8 8
NU_REQUEST_DRIVER_ID 62 3E
NU_RESET_MAILBOX_ID 14 E
NU_RESET_PIPE_ID 27 1B
NU_RESET_QUEUE_ID 20 14
NU_RESET_SEMAPHORE_ID 34 22
NU_RESET_TASK_ID 4 4
NU_RESET_TIMER_ID 59 3B
NU_RESUME_DRIVER_ID 63 3F
NU_RESUME_TASK_ID 6 6
NU_RETRIEVE_EVENTS_ID 40 28
NU_SEMAPHORE_DELETED -48 FFFFFFD0
NU_SEMAPHORE_RESET -49 FFFFFFCF
NU_SEMAPHORE_SUSPEND 6 6
NU_SEND_SIGNALS_ID 52 34
NU_SEND_TO_FRONT_OF_QUEUE_ID 21 15
NU_SEND_TO_FRONT_OF_PIPE_ID 28 1C
NU_SEND_TO_MAILBOX_ID 15 F
NU_SEND_TO_PIPE_ID 29 1D
NU_SEND_TO_QUEUE_ID 22 16
NU_SET_EVENTS_ID 39 27
NU_SLEEP_ID 9 9
NU_SLEEP_SUSPEND 2 2
NU_START 12 C
NU_SUCCESS 0 0
NU_SUSPEND 0xFFFFFFFFUL FFFFFFFF
NU_SUSPEND_DRIVER_ID 64 40
NU_SUSPEND_TASK_ID 7 7
NU_TERMINATE_TASK_ID 5 5
NU_TERMINATED 12 C
NU_TIMEOUT -50 FFFFFFCE
NU_TRUE 1 1
NU_UNAVAILABLE -51 FFFFFFCD
NU_USER_ID 1 1
NU_VARIABLE_SIZE 13 D

Nucleus PLUS Constants (Value)
Name Decimal Value Hex Value
NU_ENABLE_INTERRUPTS [Port Specific]
NU_DISABLE_INTERRUPTS [Port Specific]
NU_FALSE 0 0
NU_NO_SUSPEND 0 0
NU_NULL 0 0
NU_OR 0
0
NU_READY 0 0
NU_SUCCESS 0 0
NU_OR_CONSUME 1 1
```

```
NU_PURE_SUSPEND 1 1
NU_TRUE 1 1
NU_USER_ID 1 1
NU_AND 2 2
NU_CREATE_TASK_ID 2 2
NU_SLEEP_SUSPEND 2 2
NU_AND_CONSUME 3 3
NU_DELETE_TASK_ID 3 3
NU_MAILBOX_SUSPEND 3 3
NU_DISABLE_TIMER 4 4
NU_QUEUE_SUSPEND 4 4
NU_RESET_TASK_ID 4 4
NU_ENABLE_TIMER 5 5
NU_PIPE_SUSPEND 5 5
NU_TERMINATE_TASK_ID 5 5
NU_FIFO 6 6
NU_RESUME_TASK_ID 6 6
NU_SEMAPHORE_SUSPEND 6 6
NU_EVENT_SUSPEND 7 7
NU_FIXED_SIZE 7 7
NU_SUSPEND_TASK_ID 7 7
NU_NO_PREEMPT 8 8
NU_PARTITION_SUSPEND 8 8
NU_RELINQUISH_ID 8 8
NU_MEMORY_SUSPEND 9 9
NU_NO_START 9 9
NU_SLEEP_ID 9 9
NU_CHANGE_PRIORITY_ID 10 A
NU_DRIVER_SUSPEND 10 A
NU_PREEMPT 10 A
NU_CHANGE_PREEMPTION_ID 11 B
NU_FINISHED 11 B
NU_PRIORITY 11 B
NU_CREATE_MAILBOX_ID 12 C
NU_START 12 C
NU_TERMINATED 12 C
NU_DELETE_MAILBOX_ID 13 D
NU_VARIABLE_SIZE 13 D
NU_RESET_MAILBOX_ID 14 E
NU_SEND_TO_MAILBOX_ID 15 F
NU_BROADCAST_TO_MAILBOX_ID 16 10
NU_RECEIVE_FROM_MAILBOX_ID 17 11
NU_CREATE_QUEUE_ID 18 12
NU_DELETE_QUEUE_ID 19 13
NU_RESET_QUEUE_ID 20 14
NU_SEND_TO_FRONT_OF_QUEUE_ID 21 15
NU_SEND_TO_QUEUE_ID 22 16
NU_BROADCAST_TO_QUEUE_ID 23 17
NU_RECEIVE_FROM_QUEUE_ID 24 18
NU_CREATE_PIPE_ID 25 19
NU_DELETE_PIPE_ID 26 1A
NU_RESET_PIPE_ID 27 1B
NU_SEND_TO_FRONT_OF_PIPE_ID 28 1C
NU_SEND_TO_PIPE_ID 29 1D
NU_BROADCAST_TO_PIPE_ID 30 1E
```

```
NU_RECEIVE_FROM_PIPE_ID 31 1F
NU_CREATE_SEMAPHORE_ID 32 20
NU_DELETE_SEMAPHORE_ID 33 21
NU_RESET_SEMAPHORE_ID 34 22
NU_OBTAIN_SEMAPHORE_ID 35 23
NU_RELEASE_SEMAPHORE_ID 36 24
NU_CREATE_EVENT_GROUP_ID 37 25
NU_DELETE_EVENT_GROUP_ID 38 26
NU_SET_EVENTS_ID 39 27
NU_RETRIEVE_EVENTS_ID 40 28
NU_CREATE_PARTITION_POOL_ID 41 29
NU_DELETE_PARTITION_POOL_ID 42 2A
NU_ALLOCATE_PARTITION_ID 43 2B
NU_DEALLOCATE_PARTITION_ID 44 2C
NU_CREATE_MEMORY_POOL_ID 45 2D
NU_DELETE_MEMORY_POOL_ID 46 2E
NU_ALLOCATE_MEMORY_ID 47 2F
NU_DEALLOCATE_MEMORY_ID 48 30
NU_CONTROL_SIGNALS_ID 49 31
NU_RECEIVE_SIGNALS_ID 50 32
NU_REGISTER_SIGNAL_HANDLER_ID 51 33
NU_SEND_SIGNALS_ID 52 34
NU_REGISTER_LISR_ID 53 35
NU_CREATE_HISR_ID 54 36
NU_DELETE_HISR_ID 55 37
NU_CREATE_TIMER_ID 56 38
NU_DELETE_TIMER_ID 57 39
NU_CONTROL_TIMER_ID 58 3A
NU_RESET_TIMER_ID 59 3B
NU_CREATE_DRIVER_ID 60 3C
NU_DELETE_DRIVER_ID 61 3D
NU_REQUEST_DRIVER_ID 62 3E
NU_RESUME_DRIVER_ID 63 3F
NU_SUSPEND_DRIVER_ID 64 40
NU_CHANGE_TIME_SLICE 65 41
NU_SUSPEND 0xFFFFFFFFUL FFFFFFFF
NU_END_OF_LOG -1 FFFFFFFF
NU_GROUP_DELETED -2 FFFFFFFE
NU_INVALID_DELETE -3 FFFFFFFD
NU_INVALID_DRIVER -4 FFFFFFFC
NU_INVALID_ENABLE -5 FFFFFFFB
NU_INVALID_ENTRY -6 FFFFFFFA
NU_INVALID_FUNCTION -7 FFFFFFF9
NU_INVALID_GROUP -8 FFFFFFF8
NU_INVALID_HISR -9 FFFFFFF7
NU_INVALID_MAILBOX -10 FFFFFFF6
NU_INVALID_MEMORY -11 FFFFFFF5
NU_INVALID_MESSAGE -12 FFFFFFF4
NU_INVALID_OPERATION -13 FFFFFFF3
NU_INVALID_PIPE -14 FFFFFFF2
NU_INVALID_POINTER -15 FFFFFFF1
NU_INVALID_POOL -16 FFFFFFF0
NU_INVALID_PREEMPT -17 FFFFFFEF
NU_INVALID_PRIORITY -18 FFFFFFEE
NU_INVALID_QUEUE -19 FFFFFFED
```

```
NU_INVALID_RESUME -20 FFFFFFEC
NU_INVALID_SEMAPHORE -21 FFFFFFEB
NU_INVALID_SIZE -22 FFFFFFEA
NU_INVALID_START -23 FFFFFFE9
NU_INVALID_SUSPEND -24 FFFFFFE8
NU_INVALID_TASK -25 FFFFFFE7
NU_INVALID_TIMER -26 FFFFFFE6
NU_INVALID_VECTOR -27 FFFFFFE5
NU_MAILBOX_DELETED -28 FFFFFFE4
NU_MAILBOX_EMPTY -29 FFFFFFE3
NU_MAILBOX_FULL -30 FFFFFFE2
NU_MAILBOX_RESET -31 FFFFFFE1
NU_NO_MEMORY -32 FFFFFFE0
NU_NO_MORE_LISRS -33 FFFFFFDF
NU_NO_PARTITION -34 FFFFFFDE
NU_NOT_DISABLED -35 FFFFFFDD
NU_NOT_PRESENT -36 FFFFFFDC
NU_NOT_REGISTERED -37 FFFFFFDB
NU_NOT_TERMINATED -38 FFFFFFDA
NU_PIPE_DELETED -39 FFFFFFD9
NU_PIPE_EMPTY -40 FFFFFFD8
NU_PIPE_FULL -41 FFFFFFD7
NU_PIPE_RESET -42 FFFFFFD6
NU_POOL_DELETED -43 FFFFFFD5
NU_QUEUE_DELETED -44 FFFFFFD4
NU_QUEUE_EMPTY -45 FFFFFFD3
NU_QUEUE_FULL -46 FFFFFFD2
NU_QUEUE_RESET -47 FFFFFFD1
NU_SEMAPHORE_DELETED -48 FFFFFFD0
NU_SEMAPHORE_RESET -49 FFFFFFCF
NU_TIMEOUT -50 FFFFFFCE
NU_UNAVAILABLE -51 FFFFFFCD
```

附录B 致命系统错误

该附录包含所有的标准的Nucleus PLUS致命错误常数。如果一个致命的系统错误发生，这些常数中的一个被传送到致命错误处理函数ERC_System_Error。

如果系统错误是NU_STACK_OVERFLOW，那么表示当前执行的线程的堆栈太小了。当前的线程被全局变量TCD_Current_Thread标识。它包含指向当前线程控制块的指针。

如果系统错误是NU_UNHANDLED_INTERRUPT，那么表示收到的中断没有相匹配的LISR。这个导致系统错误的中断向量是储存在全局变量TCD_Unhandled_Interrupt中。

Nucleus PLUS 致命系统错误

名字	十进制数值	十六进制数值
NU_ERROR_CREATING_TIMER_HISR	1	1

NU_ERROR_CREATING_TIMER_TASK	2	2
NU_STACK_OVERFLOW	3	3
NU_UNHANDLED_INTERRUPT	4	4

附录C I/O 设备结构请求

该附录包含所有的标准的Nucleus PLUS I/O 设备常数和结构。《*Nucleus PLUS Reference Manual*》第三章和第五章描述了如何使用I/O 设备。

Nucleus PLUS I/O 设备常数：

名字	十进制数值	十六进制数值
NU_IO_ERROR	-1	FFFFFFFF
NU_INITIALIZE	1	1
NU_ASSIGN	2	2
NU_RELEASE	3	3
NU_INPUT	4	4
NU_OUTPUT	5	5
NU_STATUS	6	6
NU_TERMINATE	7	7

Nucleus PLUS I/O设备C语言结构：

```
/* Define I/O driver request structures. */
struct NU_INITIALIZE_STRUCT
{
VOID *nu_io_address; /* Base IO address */
UNSIGNED nu_logical_units; /* Number of logical units */
VOID *nu_memory; /* Generic memory pointer */
INT nu_vector; /* Interrupt vector number */
};
struct NU_ASSIGN_STRUCT
{
UNSIGNED nu_logical_unit; /* Logical unit number */
INT nu_assign_info; /* Additional assign info */
};
struct NU_RELEASE_STRUCT
{
UNSIGNED nu_logical_unit; /* Logical unit number */
INT nu_release_info; /* Additional release info */
};
struct NU_INPUT_STRUCT
{
UNSIGNED nu_logical_unit; /* Logical unit number */
UNSIGNED nu_offset; /* Offset of input */
UNSIGNED nu_request_size; /* Requested input size */
UNSIGNED nu_actual_size; /* Actual input size */
VOID *nu_buffer_ptr; /* Input buffer pointer */
};
struct NU_OUTPUT_STRUCT
```

```
{
UNSIGNED nu_logical_unit; /* Logical unit number */
UNSIGNED nu_offset; /* Offset of output */
UNSIGNED nu_request_size; /* Requested output size */
UNSIGNED nu_actual_size; /* Actual output size */
VOID *nu_buffer_ptr; /* Output buffer pointer */
};
struct NU_STATUS_STRUCT
{
UNSIGNED nu_logical_unit; /* Logical unit number */
VOID *nu_extra_status; /* Additional status ptr */
};
struct NU_TERMINATE_STRUCT
{
UNSIGNED nu_logical_unit; /* Logical unit number */
};
typedef struct NU_DRIVER_REQUEST_STRUCT
{
INT nu_function; /* I/O request function */
UNSIGNED nu_timeout; /* Timeout on request */
STATUS nu_status; /* Status of request */
UNSIGNED nu_supplemental; /* Supplemental information */
VOID *nu_supplemental_ptr; /* Supplemental info pointer*/
/* Define a union of all the different types of request
structures.*/
union NU_REQUEST_INFO_UNION
{
struct NU_INITIALIZE_STRUCT nu_initialize;
struct NU_ASSIGN_STRUCT nu_assign;
struct NU_RELEASE_STRUCT nu_release;
struct NU_INPUT_STRUCT nu_input;
struct NU_OUTPUT_STRUCT nu_output;
struct NU_STATUS_STRUCT nu_status;
struct NU_TERMINATE_STRUCT nu_terminate;
} nu_request_info;
} NU_DRIVER_REQUEST;
```