OS抽象层
应用程序接口
文件编号：SWRA194
德州仪器
圣迭戈，加利福尼亚州美国
目录

1. 简介

2. 1.1目的

本文件的目的是定义在OS抽象层（OSAL）的API。这个API允许的TI堆叠产品的软件组件，例如Z-
堆栈™，的RemoTI™和BLE，可以独立于操作系统的具体的书面，内核或任务环境（包括控制回路或连接-于—中断系统）

1.2适用范围

该文件列举了由OSAL提供的所有函数调用。足够详细地指定的函数调用，允许程序员来实现它们。

1.3缩略语

API应用程序接口 BLE蓝牙低能量 NV非易失
OSAL操作系统（OS）抽象层 RF4CE射频消费电子
的RemoTI德州仪器（TI）的RF4CE协议栈 的Z-Stack德州仪器ZigBee协议栈

2，API概述 2.1概述

操作系统抽象层被用来从处理环境的具体屏蔽的TI栈软件组件。它提供的方式，独立于处理环境的以下功能。
1.任务登记，初始化，出发 任务
2之间信息交流
3.任务同步
4.中断处理
5.计时器
6.内存分配

3.消息管理API

3.1简介

消息管理API提供了一种机制用于交换具有不同处理环境的任务或处理元件之间的消息（例如，中断称为控制环中服务程序或函数）.The功能在此API使任务分配和解除分配的消息缓冲区，发送命令信息到另一个任务，并收到回复信息。

3.2 osal_msg_allocate（）

3.2.1简介
这个函数被一个任务来分配消息缓冲区，任务/函数将填写的信息中并调用osal_msg_send（）将消息发送给另一个任务。如果缓冲器不能分配，msg_ptr将被设置为NULL。 注意：不要将此功能与osal_mem_alloc（），这个函数是用来分配一个缓冲区的任务之间发送消息[使用osal_msg_send（）。使用osal_mem_alloc（）分配的内存块。
3.2.2原型
UINT8 * osal_msg_allocate（UINT16 LEN）

3.2.3详细参数
len是这个消息的长度。
3.2.4返回
返回值是一个指向分配给消息的缓冲区。返回NULL表示信息分配操作失败。

3.3 osal_msg_deallocate（）

3.3.1简介
此功能用于解除分配的消息缓冲器。这个功能是由一个任务（或处理元件）称为后处理完

接收到的消息。
3.3.2原型
UINT8 osal_msg_deallocate（UINT8 * msg_ptr）
3.3.3详细参数
msg_ptr是指向需要被解除分配的消息缓冲器。
3.3.4返回
返回值指示该操作的结果。 返回值 说明
SUCCESS 德分配成功
INVALID_MSG_POINTER 无效的消息指针
MSG_BUFFER_NOT_AVAIL 缓冲区排队

3.4 osal_msg_send（）

3.4.1简介

所述osal_msg_send函数由一个任务发送一个命令或数据消息到另一任务或处理元件。该destination_task标识符字段必须指向一个有效的系统任务。该osal_msg_send（）函数也将设置目标任务事件列表中的SYS_EVENT_MSG事件。

3.4.2原型
UINT8 osal_msg_send（UINT8 destination_task，UINT8 * msg_ptr）
3.4.3详细参数
destination_task是任务的ID来接收该消息。
msg_ptr是指向包含消息的缓冲器。
Msg_ptr必须是一个指针（）通过osal_msg_allocate分配的一个有效的消息缓冲区。
3.4.4返回
返回值是1字节字段，指示该操作的结果。 返回值 说明
SUCCESS 邮件发送成功
INVALID_MSG_POINTER 无效的消息指针 INVALID_TASK Destination_task无效

3.5 osal_msg_receive（）

3.5.1简介
该功能称为由任务来检索接收的命令消息。使用osal_msg_deallocate（）调用处理消息后，调用任务必须取消分配的消息缓冲区。
3.5.2原型
UINT8 * osal_msg_receive（UINT8 TASK_ID）
3.5.3详细参数
TASK_ID是调用任务（该消息注定）的标识。
3.5.4返回
返回值是一个指针，指向包含该消息或NULL，如果没有接收到的消息的缓冲器。

3.6 osal_msg_find（）

3.6.1简介

此功能搜索现有的OSAL信息匹配TASK_ID和事件的参数。
3.6.2原型
osal_event_hdr_t * osal_msg_find（UINT8 TASK_ID，UINT8 event）
3.6.3详细参数
TASK_ID是标识符的入队OSAL信息必须匹配。 事件是OSAL事件ID的入队OSAL信息必须匹配。
3.6.4返回
返回值是一个指向上的成功或NULL失败匹配的OSAL信息。

4，任务同步API

4.1简介

这个API使得任务等待事件的发生，控制返回在等待。该API中的函数可以被用来设置为任务的事件，并通知任务一旦任何事件被设置。

4.2 osal_set_event（）

4.2.1简介

这个函数被调用来设置事件标志的任务。
4.2.2原型
UINT8 osal_set_event（UINT8 TASK_ID，UINT16 event_flag）

4.2.3详细参数
TASK_ID是该事件是要设置的任务的标识符。
event_flag是一个2字节的位图，每个比特指定一个事件。只有一个系统事件（SYS_EVENT_MSG），事件的其余/位由接收任务定义。
4.2.4返回
返回值指示该操作的结果。 返回值 说明
SUCCESS 成功
INVALID_TASK 无效的任务

5.定时器管理API

5.1简介

该API允许通过内部（TI堆栈）的任务和外部的（应用层）任务使用定时器。该API提供的功能来启动和停止计时器。该定时器可在1millisecond的增量进行设置。

5.2 osal_start_timerEx（）

5.2.1
这个函数被调用，以启动一个定时器。当计时器到期时，给定的事件位将被设置。本次活动将针对TaskID定义任务设置。该定时器是一个单次定时器，这意味着当定时器期满它不重新加载。

5.2.2原型

UINT8 osal_start_timerEx（UINT8 TASKID，UINT16 event_id， UINT32 timeout_value）;
5.2.3详细参数
的TaskID的是，是当定时器到期时，以得到该事件的任务的任务ID。
EVENT_ID是用户定义的事件位。当定时器到期时，将调用任务将通知（事件）。 timeout _value是时间的量（以毫秒为单位）的计时器事件被设置之前。
5.2.4返回
返回值指示该操作的结果。 返回值 说明
SUCCESS
定时器启动成功
NO_TIMER_AVAILABLE 无法启动定时器

5.3 osal_start_reload_timer（）

5.3.1简介

调用此函数启动计时器，到期时，将设置事件位和自动重新加载的超时值。本次活动将针对TaskID定义任务设置。

5.3.2原型
UINT8 osal_start_reload_timer（UINT8 TASKID，UINT16 event_id， UINT32 timeout_value ）;

5.3.3详细参数
的TaskID的是，是当定时器到期时，以得到该事件的任务的任务ID。
EVENT_ID是用户定义的事件位。当定时器到期时，将调用任务将通知（事件）。
timeout_value是时间的量（以毫秒为单位）的计时器事件被设置之前。此值加载到定时器，当定时器超时。

5.3.4返回

返回值指示该操作的结果。
返回值
说明
SUCCESS
定时器启动成功
NO_TIMER_AVAILABLE

无法启动定时器

5.4 osal_stop_timerEx（）

5.4.1简介

这个函数被调用来停止已启动的计时器。如果成功，该函数将取消定时器，并防止与计时器相关联的事件。

5.4.2原型
UINT8 osal_stop_timerEx（UINT8 TASK_ID，UINT16 event_id）;
5.4.3详细参数
TASK_ID是要为其停止定时器的任务。
event_id的是要被停止的定时器的标识符。

5.4.4返回
返回值指示该操作的结果。
返回值
说明
SUCCESS
计时器成功停止
INVALID_EVENT_ID
无效的事件

5.5 osal_GetSystemClock（）

5.5.1简介

这个函数被调用，以读取系统时钟
5.5.2原型
UINT32 osal_GetSystemClock（void）;
5.5.3详细参数
无。
5.5.4返回
系统时钟以毫秒为单位。

6，中断管理API

6.1简介

这个API使得任务与外部中断接口。 API中的函数允许一个任务到一个特定的服务程序与每个中断相关联。中断可以启用或禁用。里面的服务程序，事件可能会被用于其他任务设置。

6.2 osal_int_enable（）

6.2.1简介

这个函数被调用，以允许中断。一旦启用，发生中断的原因与该中断相关的服务程序被调用。

6.2.2原型

UINT8 osal_int_enable（UINT8 interrupt_id）

6.2.3详细参数
interrupt_id
标识要启用中断。

6.2.4返回
返回值指示该操作的结果。
返回值
说明
SUCCESS
中断成功启用
INVALID_INTERRUPT_ID
无效中断

6.3 osal_int_disable（）

6.3.1简介

这个函数被调用来禁止中断。当禁用中断时，与该中断相关的服务例程不被调用。

6.3.2原型
UINT8 osal_int_disable（UINT8 interrupt_id）
6.3.3详细参数
interrupt_id
标识要禁用中断。

6.3.4返回
返回值
指示该操作的结果。
返回值
说明
SUCCESS
中断禁用成功
INVALID_INTERRUPT_ID

无效中断

7.任务管理

API 7.1简介

这个API是用来在OSAL系统中添加和管理任务。每个任务是由一个初始化函数和一个事件处理功能。 OSAL调用osalInitTasks（）提供的应用程序]初始化任务，OSAL使用任务表（const的pTaskEventHandlerFn tasksArr []），呼吁每个任务（也提供应用程序）的事件处理器。

的任务表的实施例：

常量pTaskEventHandlerFn tasksArr [] = {
macEventLoop，
nwk_event_loop，
Hal_ProcessEvent，
MT_ProcessEvent，
APS_event_loop，
ZDApp_event_loop，
};

常量UINT8 tasksCnt = sizeof的（tasksArr）/的sizeof（tasksArr [0]）; 的osalInitTasks（）实现的例子： void osalInitTasks（void） {
UINT8的TaskID = 0;
tasksEvents =（UINT16 *）osal_mem_alloc（的sizeof（UINT16）* tasksCnt）;
osal_memset（tasksEvents，0，（sizeof（UINT16）* tasksCnt））;
macTaskInit（TASKID ++）; nwk_init（TASKID ++）;
Hal_Init（TASKID ++）;
MT_TaskInit（TASKID ++）;
APS_Init（TASKID ++）;
ZDApp_Init（TASKID ++）;
}

7.2 osal_init_system（）

7.2.1简介

这个函数初始化OSAL系统。该功能必须在启动时使用的任何其它OSAL功能之前被调用。

7.2.2原型

UINT8 osal_init_system（void）

7.2.3详细参数

无。

7.2.4返回
返回值
指示该操作的结果。
返回值
说明
SUCCESS
成功

7.3 osal_start_system（）

7.3.1简介

此功能是任务系统的主循环功能，重复调用osal_run_system（），从一个无限循环。这个函数永远不会返回。当使用不同的调度程序，它应该直接调用osal_run_system（）和不使用此功能。
7.3.2原型
无效osal_start_system（void）
7.3.3详细参数
无。
7.3.4返回
无。

7.4 osal_run_system（）

7.4.1简介
这个功能将会使一次通过OSAL taskEvents表并调用task_event_processor函数即发现与至少一个事件挂起第一任务。后一个事件被服务，所有剩余的事件将返回到下一次围绕主循环。如果没有挂起的事件（所有任务），这个函数使处理器进入休眠模式。

7.4.2原型
无效osal_run_system（void）
7.4.3详细参数
无。
7.4.4返回 无。

7.5 osal_self（）

7.5.1简介

该函数返回当前活动OSAL任务的任务ID，对应于OSAL任务表的任务的索引。这个函数可用于由应用程序以确定它在其下运行的OSAL任务ID。如果在没有OSAL任务是积极的呼吁，该函数返回值TASK_NO_TASK。
.5.2原型

UINT8 osal_self（void）
7.5.3详细参数
无。
7.5.4返回
当前活动任务的OSAL任务ID。
返回值
说明
为0x00 - 0xFE的
积极OSAL任务ID
为0xFF（TASK_NO_TASK）
没有OSAL任务是积极的

8.内存管理

API 8.1简介

这个API代表了一个简单的内存分配系统。这些功能允许动态内存分配。

8.2 osal_mem_alloc（）

8.2.1简介
此功能是一个简单的内存分配函数返回一个指向缓冲区（如果成功的话）。

8.2.2原型
void *的osal_mem_alloc（UINT16 size）;
8.2.3详细参数
大小 - 字节数就想在缓冲器中。
8.2.4返回
一个空指针（应强制转换为预期的缓冲型）到新分配的缓冲区。如果没有足够的内存来分配则返回NULL指针。

8.3 osal_mem_free（）

8.3.1简介

此函数释放被再次使用所分配的存储器。如果内存已经分配osal_mem_alloc这种方法只适用（）。
8.3.2原型
void osal_mem_free（void * PTR）;
8.3.3详细参数
PTR - 指向缓冲区被“释放”。缓冲区必须事先与osal_mem_alloc分配的（）。
8.3.4返回
无。

9，电源管理API

9.1简介

本节描述OSAL的电源管理系统。该系统提供了一种为应用程序/任务通知OSAL何时可以安全关闭接收器和外部硬件，并使处理器在睡觉。
有两种功能，用来控制电源管理。
第一，osal_pwrmgr_device（）被调用来设置设备级别模式（省电或不省电）。然后，有任务的功率状态，每个任务可以容纳切断电源管理器从通过调用osal_pwrmgr_task_state（PWRMGR_HOLD）节省功率。如果一个任务“举行的”电源管理器，它需要调用osal_pwrmgr_task_state（PWRMGR_CONSERVE），允许电源管理器在电力保护模式继续。
默认情况下，当任务被初始化，每个任务的电源状态设置为PWRMGR_CONSERVE，所以如果一个任务不希望拖延节能（无变化），它并不需要调用osal_pwrmgr_task_state（）。
此外，在默认情况下，电池供电的设备将在PWRMGR_ALWAYS_ON状态，直到它加入了网络，那么这将改变其状态PWRMGR_BATTERY。这意味着，如果设备无法找到设备的加入也不会进入节电状态。如果你想改变这种行为，在应用程序的任务初始化函数中添加osal_pwrmgr_device（PWRMGR_BATTERY），或者当您的应用程序停止/暂停连接过程。
在电源管理器将着眼于设备模式和所有任务才去到电源养护状态集体电源状态。

9.2 osal_pwrmgr_init（）

9.2.1简介

此功能将初始化所使用的电源管理系统的变量。重要提示：不要调用这个函数，它已经称为osal_init_system（）。

9.2.2原型
无效osal_pwrmgr_init（void）;
9.2.3详细参数
无。
9.2.4返回
无。

9.3 osal_pwrmgr_powerconserve（）

9.3.1简介

这个函数被调用来进入掉电模式。重要提示：不要调用这个函数，它已经堪称OSAL主循环[osal_start_system（）。

9.3.2原型
无效osal_pwrmgr_powerconserve（void）;

9.3.3详细参数
无。
9.3.4返回
无。

9.4 osal_pwrmgr_device（）

9.4.1简介

该功能称为上电时或者当动力需求的变化（例如：电池的支持协调人）。这个函数设置该设备的电源管理的整体开/关状态。该功能被称为从中央控制实体（如ZDO）。
9.4.2原型
无效osal_pwrmgr_device（UINT8 pwrmgr_device）;
9.4.3详细参数
Pwrmgr_device - 修改或设置节能模式。
类型
描述
PWRMGR_ALWAYS_ON
有了这个选择，没有动力节约和设备是最有可能在主电源。
PWRMGR_BATTERY
打开省电。

9.4.4返回
无。

9.5 osal_pwrmgr_task_state（）

9.5.1简介

这个函数由每个任务陈述此任务是否想以节省电力。该任务将调用这个函数来投票决定是否它希望OSAL以降低功耗，或者希望在积蓄力量挡住。默认情况下，创建任务时，自己的电源状态设置为保存。如果任务总是希望交谈功率，它不需要调用该函数在所有。

9.5.2原型
UINT8 osal_pwrmgr_task_state（UINT8 TASK_ID，UINT8 state）;
9.5.3详细参数
state- 改变一个任务的电源state。
类型
描述
PWRMGR_CONSERVE
打开电源节约，所有任务必须同意。这是默认状态当任务被初始化。
PWRMGR_HOLD
打开省电关闭。
9.5.4返回

返回值指示该操作的结果。
返回值
说明
SUCCESS
成功

INVALID_TASK
无效的任务

10.非挥发性内存API

10.1简介

本节介绍OSAL非易失性（NV）内存系统。该系统提供了一种方法为应用程序持久存储信息到设备的存储器中。它也可用于由堆栈对于由ZigBee规范所需的某些项目的持久存储。对NV函数设计的读写选自由任意数据类型，如结构或数组的用户定义的项目。用户可以读或写整个项目或项目的通过设置适当的偏移量和长度的元件。该API是独立的NV的存储介质，并可以为闪存或EEPROM来实现。
每个NV项目都有一个唯一的ID。有特定的ID值范围为应用程序，同时一些ID值预留或使用的栈或平台。如果你的应用程序创建自己的NV项目时，必须选择从应用程序的价值范围内的ID。请参阅下表。
VALUE
用户
为0x0000
保留的
0×0001 - 0×0020
OSAL
0x0021 - 0x0040
NWK
0x0041 - 0x0060
APS
0x0061 - 0x0080
安全
0x0081 - 0x00B0
ZDO
0x00B1 - 0x00E0
调试SAS
0x00E1 - 0100
保留的
0x0101 - 0x01FF

信任中心链路密钥
0x0201 - 0x0300
ZigBee的优点：APS链接键
ZigBee的RF4CE：网络层
0x0301 - 的0x0400
ZigBee的优点：主密钥
ZigBee的RF4CE：应用程序框架
0x0401 - 0x0FFF
应用
为0x1000 -0xFFFF
保留的
没有使用此API时，一些重要的注意事项：
1.这些被阻塞函数调用和操作可能需要若干毫秒才能完成。这是为NV的写入操作中尤其如此。另外，中断可能会为几毫秒禁用。最好是在时间执行这些功能时，他们不与其他时序关键的操作发生冲突。举例来说，一个好的时间来写NV项目将是当接收机处于关闭状态。
2.尝试执行NV写操作频繁。这需要时间和力量;也最闪光装置具有擦除周期的数量有限。

3.如果一个或多个NV项的变化的结构中，特别是从一台TI栈软件一个版本升级到另一个时，有必要擦除和重新初始化NV存储器。否则，读取和NV项改变将会失败或产生错误的结果写入操作。

10.2 osal_nv_item_init（）

10.2.1说明

初始化NV项目。此功能检查中的某个项目NV的存在。如果它不存在，则创建并用，传递给函数的数据，如果任何初始化。
该功能必须在调用osal_nv_read（）或osal_nv_write（）之前被调用为每个项目。

10.2.2原型
UINT8 osal_nv_item_init（UINT16 ID，UINT16 LEN，void * BUF）;
10.2.3参数详细信息
ID - 用户定义的项目编号。
LEN - 以字节为单位项目长度。
* BUF - 指向项目初始化数据。如果没有初始化数据，设置为NULL。

10.2.4返回
返回值指示该操作的结果。
返回值
说明
SUCCESS
成功
NV_ITEM_UNINIT
成功，但项目并不存在

NV_OPER_FAILED
操作失败

10.3 osal_nv_read（）

10.3.1说明

阅读来自NV的数据。这个函数可用于通过索引入项目具有偏移来读取从NV整个项目或项目的一个元素。读取数据复制到* buf中。
10.3.2原型
UINT8 osal_nv_read（UINT16 ID，UINT16 offset，UINT16 LEN，void * BUF）;

10.3.3参数详细信息
ID - 用户定义的项目编号。
偏移 - 内存偏移量项目以字节为单位。
LEN - 以字节为单位项目长度。
* BUF - 数据读入此缓冲区。

10.3.4返回
返回值
说明
SUCCESS
成功
NV_OPER_FAILED
操作失败

10.4 osal_nv_write（）

10.4.1说明

将数据写入NV。这个函数可用于通过索引入项目具有偏移来写整个项NV或一个项目的一个元素。
10.4.2原型
UINT8 osal_nv_write（UINT16 ID，UINT16 offset，UINT16 LEN，void * BUF）;
10.4.3参数详细信息
ID - 用户定义的项目编号。
偏移 - 内存偏移量项目以字节为单位。
LEN - 以字节为单位项目长度。
* BUF - 要写入的数据。
10.4.4返回
返回值指示该操作的结果。
返回值
说明
SUCCESS

成功
NV_ITEM_UNINIT
项目未初始化
NV_OPER_FAILED
操作失败

10.5 osal_nv_delete（）

10.5.1说明

删除从内华达州的一个项目。此功能检查的项目在NV的存在。如果该项目存在，并且其长度在函数调用提供的长度相匹配，该项目将被从NV中删除。

10.5.2原型

UINT8 osal_nv_delete（UINT16 ID，UINT16 LEN）;

10.5.3参数详细信息

ID - 用户定义的项目编号。

LEN - 以字节为单位项目长度。

10.5.4返回

返回值指示该操作的结果。

返回值

说明

SUCCESS

成功

NV_ITEM_UNINIT

项目未初始化

NV_BAD_ITEM_LEN

不正确长度参数

NV_OPER_FAILED

操作失败

10.6 osal_nv_item_len（）

10.6.1说明

获取NV项目的长度。这个函数返回一个NV项目的长度，如果发现了，否则为零。

10.6.2原型

UINT16 osal_nv_item_len（UINT16 ID）;

10.6.3参数详细信息

ID - 用户定义的项目编号。

10.6.4返回

返回值指示该操作的结果。

返回值

说明

0

NV项未找到
1 - N
NV项的长度

10.7 osal_offsetof（）

10.7.1说明

该宏计算存储器中的结构中的一个元素的字节偏移。它是用于计算所使用的NV API函数的偏移参数是有用的。
10.7.2原型
osal_offsetof（类型，成员）
10.7.3参数详细信息
类型 - 结构型。
成员 - 结构成员。

11.简单的非挥发性内存API

11.1简介

本节介绍OSAL简单的非挥发性内存系统。像OSAL NV存储器系统，简单NV存储系统提供了一种方法，应用到持久存储信息到设备内存。另一方面，在OSAL NV存储器系统不同，简单NV存储器系统提供更简单的API来驱动应用程序代码大小和堆栈代码大小向下以及OSAL简单的NV系统实现的代码大小。用户可以读或写整个项目，但它不能部分读取或写入的项目。
就像NV存储器系统，每个NV项具有一个唯一的ID。有特定的ID值范围为应用程序，同时一些ID值预留或使用的栈或平台。如果你的应用程序创建自己的NV项目，它必须选择从应用程序的价值范围内的ID。请参阅下表。
请注意，表可能并不适用于一定的自定义堆栈的构建。
VALUE
用户
为0x00
保留的
为0x01 - 0x6F
预留的ZigBee RF4CE网络层
0x70 - 0x7F的
预留的ZigBee RF4CE应用程序框架（RTI）
0x80的 - 0xFE的
应用
为0xFF
保留的
没有使用此API时，一些重要的注意事项：
1.这些被阻塞函数调用和操作可能需要几百毫秒才能完成。这是为NV的写入操作中尤其如此。另外，中断可能会为几毫秒禁用。最好是在时间执行这些功能时，他们不与其他时

序关键的操作发生冲突。举例来说，一个好的时间来写NV项目将是当接收机处于关闭状态。

2.此外，这些功能不能由一个中断服务例程中调用，除非由一个单独的应用说明另有规定或释放的实现的音符。

3.尝试执行NV写操作频繁。这需要时间和力量;也最闪光装置具有擦除周期的数量有限。 4.如果一个或多个NV项的变化的结构中，特别是从一台TI栈软件一个版本升级到另一个时，有必要擦除和重新初始化NV存储器。否则，读取和NV项改变将会失败或产生错误的结果写入操作。

11.2 osal_snv_read（）

11.2.1说明

阅读来自NV的数据。此功能可被用于读取来自NV整个项目。读取数据复制到* PBUF。

11.2.2原型

UINT8 osal_snv_read（osalSnvId_t ID，osalSnvLen_t LEN，void * PBUF）;

11.2.3参数详细信息

ID - 用户定义的项目编号。

LEN - 以字节为单位项目长度。

* PBUF - 数据读入此缓冲区。

11.2.4返回

返回值指示该操作的结果。需要注意的是试图读取的项目，这是从未写入之前，将导致进入NV_OPER_FAILED返回代码。

返回值	说明
SUCCESS	成功
NV_OPER_FAILED	操作失败

11.3 osal_snv_write（）

11.3.1说明

将数据写入NV。该功能可用于编写整个项目NV。

11.3.2原型

UINT8 osal_snv_write（osalSnvId_t ID，osalSnvLen_t LEN，void * PBUF）;

11.3.3参数详细信息

ID - 用户定义的项目编号。

LEN - 以字节为单位项目长度。 * PBUF - 要写入的数据。

11.3.4返回

返回值指示该操作的结果。需要注意的是它是允许写入人们从未初始化为所述NV系统通过

其他手段的项目。 返回值 说明
SUCCESS 成功
NV_OPER_FAILED 操作失败

12. OSAL时钟系统

12.1简介

本节介绍了OSAL时钟系统。该系统提供了一种方法，以保持日期和时间的装置。该系统将保留的秒数，因为0小时，0分，0秒2000年1月1日UTC。以下两种数据类型/结构被用于在该系统中（在OSAL_Clock.h定义）： //秒自0小时，0分，0秒，对数 // 2000年1月1日UTC 类型定义UINT32 UTCTime; //与所用 typedef结构 {
UINT8秒; // 0-59 UINT8分钟; // 0-59 UINT8小时; // 0-23 UINT8天; // 0-30 UINT8月; // 0-11 UINT16年; // 2000+ }UTCTimeStruct;
您必须启用OSAL_CLOCK编译器标志使用此功能。另外，此功能不维护时间睡觉设备。

12.2 osalTimeUpdate（）

12.2.1说明

从osal_run_system（）调用，以更新的时间。该函数读取320usec蜱保持OSAL时钟时间的MAC的数量。不要调用这个函数别处。
12.2.2原型
无效osalTimeUpdate（void）;
12.2.3参数详细信息
无。
12.2.4返回
无。

12.3 osal_setClock（）

12.3.1说明
调用此函数初始化设备的时间。
12.3.2原型
无效osal_setClock（UTCTime NEWTIME）;
12.3.3参数详细信息
NEWTIME - 从0小时0分0秒，于2000年1月1日UTC以秒为单位的新时间。
12.3.4返回
无。

12.4 osal_getClock（）

12.4.1说明
调用此函数来获取设备的当前时间。
12.4.2原型

UTCTime osal_getClock（void）;
12.4.3参数详细信息
无。
12.4.4返回
当前时间，以秒为单位自零小时，0分0秒，于2000年1月1日UTC。

12.5 osal_ConvertUTCTime（）

12.5.1说明
调用此函数UTCTime转换为UTCTimeStruct。
12.5.2原型
无效osal_ConvertUTCTime（UTCTimeStruct * TM，UTCTime secTime）;
12.5.3参数详细信息
secTime - 从0小时0分0秒，于2000年1月1日UTC时间以秒为单位。 TM - 指针的时间结构。
12.5.4返回
无。

12.6 osal_ConvertUTCSecs（）

12.6.1说明

调用此函数UTCTimeStruct转换为UTCTime。
12.6.2原型
UTCTime osal_ConvertUTCSecs（UTCTimeStruct * TM）;
12.6.3参数详细信息
TM - 指针的时间结构。
12.6.4返回
因为0小时，0分，0秒转换时间，以秒为单位，于2000年1月1日UTC。

13. OSAL杂项

13.1简介

本节介绍不适合以前的OSAL分类杂项OSAL功能。

13.2 osal_rand（）

13.2.1说明
此函数返回一个16位的随机数。
13.2.2原型
UINT16 osal_rand（void ）;
13.2.3参数详细信息
无。

13.2.4返回
返回一个随机数。

13.3 osal_memcmp（）

13.3.1说明
比较内存部分。
13.3.2原型
UINT8 osal_memcmp（const void GENERIC * SRC1，const void GENERIC * SRC2，unsigned int len）；
13.3.3参数详细信息
SRC1 - 内存比较位置1。
SRC2 - 内存比较位置2。
LEN - 比较长。
13.3.4返回
TRUE - 同样的，假 - 不同。

13.4 osal_memset（）

13.4.1说明

设置一个缓冲器为特定值。
13.4.2原型
void *的osal_memset（void * dest，UINT8 value，int len）；

13.4.3参数详细信息
dest - 内存缓冲区“value”设置为。
价值 - 什么设置“dest”的每一个字节。
len - 长度设置“value”中的“dest”。

13.4.4返回
指针，其中在缓冲此功能停止。

13.5 osal_memcpy（）

13.5.1说明

复制一个缓冲到另一个缓冲区。
13.5.2原型
void *的osal_memcpy（void *dst，const void GENERIC * SRC,unsigned int len）；
13.5.3参数详细信息
DST - 目标缓冲区。

SRC - 源缓冲区。
LEN - 副本长度。
13.5.4返回
指针最终的目标缓冲区。