Chipcon Products from Texas Instruments

Z-Stack 操作系统抽象层
应用程序编程接口

文件编号：F8W-2003-0002

德州仪器股份有限公司
美国加利福尼亚州圣迭戈
(619) 497-3845

版本	描述	日期
1.0	最初发行的ZigBee1.0版本。	04/08/2005
1.1	增加了个域网内存API说明中个域网初始化的注释。	07/22/2005
1.2	修改了事物管理API 的讨论研究。	08/25/2005
1.3	改变了标题页的标志, 改变了页脚的版权。	02/27/2006
1.4	修改了电源管理的 API。	11/27/2006
1.5	弃用了osal_self() 和 osalTaskAdd()。	12/18/2007

目录

1、引言

1.1 目地

文件的目地是规定操作系统抽象层的API。这个API允许Z-stack中的软件组件的要点或任务环境(包括操纵系统或连接到中断系统)被写入独立于特定的操作系统中。该OSAL是被执行的目标。

1.2 范围

文件中列举了通过OSAL提供所有调用函数。函数调用中细节给予充分的详述便于程序员去实现它们。

1.3 简称

API	应用程序编程接口
OSAL	操作系统抽象层
PC	个人电脑
SPI	串行端口接口

2、API 概述

2.1 概述

操作系统抽象层通过特定的操作环境常用于屏蔽软件组成中的（Z-stack）。在一定程序上它提供了与操作环境无关的下述功能。

1、任务记录，初始化，启动
2、任务之间的信息交换
3、任务同步
4、中断处理
5、定时器
6、内存分配

3、信息管理 API

3.1 介绍

信息管理 API 为任务和处理单元之间的信息交换提供了一种具有不同处理环境的机制（例如，在一个控制循环中调用中断服务常规程序或函数）。这个 API 中的函数可以使任务分配或回收信息缓冲区，给其它任务发送命令信息以及接收回复信息。

3.2 osal_msg_allocate ( )

3.2.1 函数描述

这个函数被一个任务调用去分配一个信息缓冲，这个任务/函数将填充这信息并且调用 osal_msg_send()发送信息到另一个任务中。假如缓冲器不能被分配，msg_ptr 将设置为空。

注意：不能把这个函数和 osal_mem_alloc()混淆，这个函数用来分配缓冲区以在任务之间发送信息 [（使用）osal_msg_send()]任务中，运用 osal_mem_alloc()分配块存储。

3.2.2 函数原形

```
byte   *osal_msg_allocate(   uint16   len   )
```

3.2.3 参数描述

len 是信息的长度。

3.2.4 返回值

这返回值是指向一个信息分配的缓冲区的指针。一个空值的返回标明了信息分配操作失败。

3.3 osal_msg_deallocate( )

3.3.1 函数描述

这个函数用来回收一个信息缓冲区。在完成处理一个接收信息后这个函数被一个任务(或处理机单元)调用。

3.3.2 函数原型

```
byte   osal_msg_deallocate(   byte   *msg_ptr   )
```

3.3.3 参数描述

msg_ptr 是指向必须被回收的信息缓冲的指针。

3.3.4 返回值

返回值指示了操作的结果。

返回值	描述
ZSUCCESS	分配成功
INVALID_MSG_POINTER	无效的信息指针
MSG_BUFFER_NOT_AVAIL	缓冲区队列

3.4 osal_msg_send()

3.4.1 函数描述

osal_msg_send( )函数被一个任务调用来给另一个任务或处理单元发送命令或数据信息。标识符的 destination_task 字段必须提到一个有效的系统任务。当调用 osal_create_task() 来启动一个任务时，任务标识符被分配。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 函数原型

```
byte osal_msg_send( byte destination_task, byte *msg_ptr )
```

3.4.3 参数描述

destination_task 是接收信息的任务的 ID。

msg_ptr 是指向包含信息的缓冲区的指针。msg_ptr 必须指向 osal_msg_allocate ()分配的一个有效缓冲区。

3.4.4 返回值

返回值是一个字节，表明操作结果。

返回值	描述
ZSUCCESS	信息发送成功
INVALID_MSG_POINTER	无效的信息指针
INVALID_TASK	Destination_task 是无效的

3.5 osal_msg_receive()

3.5.1 函数描述

这个函数被一个任务调用来检索一条已收到的命令信息。调用 osal_msg_deallocate()处理信息之后，调用任务必须回收信息缓冲区）。

3.5.2 函数原型

```
byte   *osal_msg_receive(   byte   task_id   )
```

3.5.3 参数描述

task_id 是调用任务（信息指定的）的标识符。

3.5.4 返回值

返回值是一个指向包含该信息的缓冲区的指针，如果没有已接收的信息为空。

4、任务同步 API

4.1 介绍

这个 API 使得任务等待事件发生，并在等待期间返回控制。这个 API 中的函数可以用来为一个任务设置事件，并无论设置了什么事件都通知任务。

4.2 osal_set_event()

4.2.1 函数描述

通过这个函数调用为一个任务设置事件标志。

4.2.2 函数原型

byte osal_set_event(byte task_id, UINT16 event_flag)

4.2.3 参数描述

task_id 是要设置事件的任务的标识符。

event_flag 是两个字节的位图且每个位详述了一个事件。这仅有一个系统事件(SYS_EVENT_MSG)，其余的事件/位是通过接收任务来规定的。

4.2.4 返回值

返回值指示了操作的结果。

返回值	描述
ZSUCCESS	成功
INVALID_TASK	无效的事件

5、定时器管理 API

5.1 介绍

这个 API 使得内部的（Z-Stack）任务和外部的（应用层）任务都可以使用定时器。API 提供了启动和停止一个定时器的功能，这定时器可设定递增的一毫秒。

5.2 osal_start_timer()

5.2.1 函数描述

启动一个定时器时调用此函数。当定时器终止，这给定的事件的位将设置。这个事件通过 osal_start_timer 函数调用将在任务中设置。要明确指定这任务的标识符，运用 osal_start_timerEx()而不是 osal_start_timer()。

5.2.2 函数原型

```
byte  osal_start_timer(UINT16  event_id,  UINT16  timeout_value);
```

5.2.3 参数描述

event_id 是用户确定事件的位。当定时器终止时，调用任务将被告知（该事件）。

timeout_value 指定时器事件设置之前的时长（以毫秒为单位）。

5.2.4 返回值

返回值指示了操作的结果。

返回值	描述
ZSUCCESS	定时器启动成功
NO_TIMER_AVAILABLE	没有能够启动定时器

5.3 osal_start_timerEx()

5.3.1 函数描述

这类似于 osal_start_timer()，增加了 taskID 参数。它允许访问者调用程序为另一个任务设置定时器。当存在疑问时，运用 osal_start_timer()之上的函数。

5.3.2 函数原型

byte osal_start_timerEx(byte taskID, UINT16 event_id, UINT16 timeout_value);

5.3.3 参数描述

taskID 是当定时器终止时，获得该事件任务的 ID。
event_id 是用户确定事件的位。当定时器终止，调用任务将被告知（该事件）。
timeout_value 是定时器事件设置之前的时长（以毫秒为单位）。

5.3.4 返回值

返回值指出了操作的结果。

返回值	描述
ZSUCCESS	定时器启动成功
NO_TIMER_AVAILABLE	没有能够启动定时器

5.4 osal_stop_timer()

5.4.1 函数描述

调用函数用来停止一个已启动的定时器。假如成功的话，这函数将取消定时器并阻止设置调用程序中与定时器相关的事件。使用 osal_stop_timer()函数 意味着在调用 osal_stop_timer() 的任务中的计时器正在运行。为了在不同的任务中中止定时器，运用 osal_stop_timerEx() 而不是 osal_stop_timer()。

5.4.2 函数原型

byte osal_stop_timer(UINT16 event_id);

5.4.3 参数描述

event_id指要停止的计时器的标识符。

5.4.4 返回值

返回指示操作结果的值。

返回值	描述
ZSUCCESS	关闭定时器成功
INVALID_EVENT_ID	无效事件

5.5 osal_stop_timerEx()

5.5.1 函数描述

这个函数和 osal_stop_timer 相似，例外的是在调用 osal_stop_timerEx 时 task_id 被指定。

5.5.2 函数原型

```
byte  osal_stop_timerEx(  byte  task_id,  UINT16  event_id  );
```

5.5.3 参数描述

task_id 是停止计时器的任务。
event_id 被停止的定时器的标识符。

5.5.4 返回值

返回值表明操作结果。

返回值	描述
ZSUCCESS	关闭定时器成功
INVALID_EVENT_ID	无效事件

5.6 osal_GetSystemClock()

5.6.1 函数描述

调用此函数读取系统时钟。

5.6.2 函数原型

```
uint32  osal_GetSystemClock(  void  );
```

5.6.3 参数描述

无。

5.6.4 返回值

系统时钟，以毫秒为单位。

6、中断管理 API

6.1 介绍

此 API 使得一个任务可以与外部中断相互交流。API 中的函数允许和每个中断去联络一个具体的服务例程。中断可以启用或禁用。在服务例程内部，可以为其它任务设置事件。

6.2 osal_int_enable()

6.2.1 函数描述

调用此函数启用一个中断。一旦启用，中断发生将引起与该中断相联系的服务例程的调用。

6.2.2 函数原型

byte osal_int_enable(byte interrupt_id)

6.2.3 参数描述

interrupt_id 识别有效的中断字符。指明要启用的中断。

6.2.4 返回值

返回值指示操作结果。

返回值	描述
ZSUCCESS	开启中断成功
INVALID_INTERRUPT_ID	无效中断

6.3 osal_int_disable()

6.3.1 函数描述

调用此函数禁用一个中断。当禁用一个中断时，与该中断相联系的服务例程不被调用。

6.3.2 函数原型

byte osal_int_disable(byte interrupt_id)

6.3.3 参数描述

interrupt_id 识别无效的中断字符。指明要禁用的中断。

6.3.4 返回值

返回值指明操作结果。

返回值	描述
ZSUCCESS	关闭中断成功
INVALID_INTERRUPT_ID	无效中断

7、任务管理 API

7.1 介绍

在 OSAL 系统中，API 常用于添加和管理任务。每个任务由初始化函数和事件处理函数组成。OSAL 调用 osalInitTasks()(应用程序提供)去初始化这任务且 OSAL 运用一个任务列表(const pTaskEventHandlerFn tasksArr[])去为每个任务(也是应用程序提供)调用事件处理程序。

执行任务列表的例子：

```
const pTaskEventHandlerFn tasksArr[] =
    {
      macEventLoop,
      nwk_event_loop,
      Hal_ProcessEvent,
      MT_ProcessEvent,
      APS_event_loop,
      ZDApp_event_loo
      p,
    };

const uint8 tasksCnt = sizeof( tasksArr ) / sizeof( tasksArr[0] );
```

执行osalInitTasks()的例子：

```
void osalInitTasks( void )
    {
      uint8 taskID = 0;

      tasksEvents = (uint16 *)osal_mem_alloc( 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 函数描述

这函数初始化 OSAL 系统。在使用任何其它 OSAL 函数之前必须先调用此函数启动 OSAL 系统。

7.2.2 函数原型

```
byte osal_init_system( void )
```

7.2.3 参数描述

无。

7.2.4 返回值

指明操作结果。

返回值	描述
ZSUCCESS	成功

7.3 osal_start_system()

7.3.1 函数描述

这个函数是任务系统中的主循环函数。它将仔细检查所有的任务事件且且为含有该事件的任务调用任务事件处理函数。假如这有特定任务的事件，这个函数将为该任务调用事件处理例程来处理事件。相应任务的事件处理例程一次处理一个事件。一个事件被服务后，剩余的事件将等待下一次循环。如果这没有事件(服务与所有任务)，这个函数使处理器程序处于睡眼模式。

7.3.2 函数原型

```
byte    osal_init_system(    void    )
```

7.3.3 参数描述

无。

7.3.4 返回值

返回指示操作结果的值。

7.4 osal_self()

7.4.1 函数描述

这一函数已被淘汰并不再支持。

7.5 osalTaskAdd ()

7.5.1 函数描述

这个函数已被淘汰且不被支持。关于OSAL任务初始化和事件处理参考7.1节。

8、内存管理 API

8.1 介绍

该API 代表一个简单的内存分配系统。这些函数允许动态存储内存分配。

8.2 osal_mem_alloc()

8.2.1 函数描述

这个函数是一个简单的内存分配函数，返回一个指向缓冲区的指针（如果成功）。

8.2.2 函数原型

```
void  *osal_mem_alloc(  uint16  size  );
```

8.2.3 函数描述

size—要求的缓冲区的字节数值。

8.2.4 返回值

指向新分配的缓冲区的空指针（应指向目的缓冲类型）。如果没有足够的内存可分配，一个返回NULL指针。

8.3 osal_mem_free()

8.3.1 函数描述

此函数释放存储空间便于再次运用。仅在内存已经通过调用osal_mem_alloc()被分配才有效。

8.3.2 函数原型

```
void  osal_mem_free(  void  *ptr  );
```

8.3.3 参数描述

ptr—指向要释放的缓冲区的指针。这个缓冲区必须之前通过调用osal_mem_alloc()已被分配。以前分配过的空间。

8.3.4 返回值

无。

9、电源管理 API

9.1 介绍

这章节描述OSAL的电源管理系统。当它安全关闭接收器和外部硬件时，这个系统为应用程序或任务提供了一种告知OSAL的方法。接着使处理器转入睡眠。

有2个函数控制电源管理。第一个，调用osal_pwrmgr_device()去设置设备的级别模式(省电或不省电)。接着，有一个任务电源状态，通过调用osal_pwrmgr_task_state(PWRMGR_HOLD)，每个任务可以在节电模式中推迟电源管理。假如一个任务保持电源管理，它将必须调用osal_pwrmgr_task_state(PWRMGR_CONSERVE)，允许在节电模式中继续使用电源管理。

默认情况下，当任务已初始化，每个任务的能源状态设置成PWRMGR_CONSERVE,保护电池的管理软件。因此这个任务不想推迟电源保护，它就不必调用osal_pwrmgr_task_state()。

在进入电源保护状态之前，电源管理将查看设备模式，以及所有任务的集体电源状态。

9.2 osal_pwrmgr_device()

9.2.1 函数描述

当升高电源或需要改变电源时（例如，电池支持的协调器）这个函数应由中心控制实体（比如ZDO）调用。

9.2.2 函数原型

```
void osal_pwrmgr_state( byte pwrmgr_device );
```

9.2.3 参数描述

pwrmgr_device—改变和设置电源的节省模式。

类型	描述
PWRMGR_ALWAYS_ON	这一选项没有省电模式，设备很可能使用主电源供电。
PWRMGR_BATTERY	打开省电模式

9.2.4 返回值

无。

9.3 osal_pwrmgr_task_state()

9.3.1 函数描述

无论这个任务是否想要保护电源，每个任务都将调用此函数。任务将调用此函数来表决是否需要OSAL保护电源或推迟电源保护。默认情况下，当一个任务被创建时，它自己的电源状态设置为保护模式。如果该任务一直想要保护电源，就根本不必调用此函数。

9.3.2 函数原型

```
byte osal_pwrmgr_task_state( byte task_id, byte state );
```

9.3.3 参数描述

state—改变一个任务的电源状态。

标志	描述
PWRMGR_CONSERVE	打开省电模式，所有事件必须允许。事件初始化时这是默认状态。
PWRMGR_HOLD	关闭省电模式

9.3.4 返回值

返回标示操作状态的值。

返回值	描述
ZSUCCESS	成功
INVALID_TASK	无效事件

10、非易失性存储器的 API

10.1 介绍

这一节描述了OSAL非易失性存储器系统。该系统为应用程序提供了一种把信息永久保存到设备内存的方法。它还能用于ZigBee规范要求的把某些项目永久保存到协议栈。NV函数的职能是读写任意数据类型的用户自定义项目，比如结构体和数组。用户能通过设置适当的偏移和长度来读和写一个整体的项目或元素。API独立于NV存储介质，并且能用于实现闪存或EEPROM。

每个易失性的项目都仅有一个ID，当一些ID值由栈或平台保留或运用时，应用程序中有特定一系列的ID值。假如应用程序创建自己的易失性项目，它必须从应用范围的值内选择一个标识符。参考下面的列表：

值	使用者
0x0000	保留
0x0001 – 0x0020	操作系统抽象层
0x0021 – 0x0040	网络层
0x0041 – 0x0060	应用支持子层
0x0061 – 0x0080	安全
0x0081 – 0x00A0	Zigbee设备对象
0x00A1 – 0x0200	保留
0x0201 – 0x0FFF	应用程序
0x1000 -0xFFFF	保留

运用API时一些重要的注意事项：

1、 有一些模块化调用函数，操作系统可能需要几毫秒来完成。对于NV写入操作尤其是这样。另外，中断可能禁用几个毫秒。当他们不与时序要求较严格的操作发生冲突时，最好间或执行这些函数。例如，当接受器被关掉时正是写NV 项目的好时间。
2、 尽量不要太频繁的执行NV写入操作。它需要时间和能源，而且闪存设备有一个有限的擦除周期数。
3、 假如结构体中一个多个NV项目改变，尤其是从一个Z-Stack版本升级到另一个时，必须擦除和重新初始化NV内存。否则，修改NV项目的读写操作将失败，或产生错误结果。

10.2 osal_nv_item_init()

10.2.1 函数描述

初始化NV项目，这个函数检查存在NV的项目，假如不存在，它将通过这个函数去创建或初始化。假如真存在的话，在调用osal_nv_read() 或osal_nv_write()之前，每个项目都应调用此函数。

10.2.2 函数原型

byte osal_nv_item_init(uint16 id, uint16 len, void *buf);

10.2.3 参数描述

id—用户自定义项的ID。
len—项目字节长度。
*buf—初始化数据的指针。如没初始化数据，则设置为空。

10.2.4 返回值

返回指示操作结果的值。

返回值	描述
ZSUCCESS	成功
NV_ITEM_UNINIT	成功但对象不存在
NV_OPER_FAILED	操作失败

10.3 osal_nv_read()

10.3.1 函数描述

从NV中读出数据。这个函数能用来从NV 中带有偏移的索引指向的项目读出整个项目或一个元素。读出的数据复制到*buf中。

10.3.2 函数原型

byte osal_nv_read(uint16 id, uint16 offset, uint16 len, void *buf);

10.3.3 参数描述

id—用户自定义项的ID
offset—以字节为单位到项目的存储偏移量。
len—项目长度以字节为单位。
*buf—数据读取到缓冲区

10.3.4 返回值

指示返回操作结果的值。

返回值	描述
ZSUCCESS	成功
NV_ITEM_UNINIT	对象没有初始化
NV_OPER_FAILED	操作失败

10.4 osal_nv_write()

10.4.1 函数描述

写入数据到NV，这个函数用来通过带有偏移的索引指向项目的偏移量来写入整个NV项目。项目的元素。

10.4.2 函数原型

byte osal_nv_write(uint16 id, uint16 offset, uint16 len, void *buf);

10.4.3 参数描述

id—用户自定义ID。
offset—以字节为单位到项目的存储偏移量。
len—项目长度以字节为单位。
*buf—数据写入到缓冲区。

10.4.4 返回值

返回指定操作结果的值。

返回值	描述
ZSUCCESS	成功
NV_ITEM_UNINIT	对象未初始化
NV_OPER_FAILED	操作失败

10.5 osal_offsetof()

10.5.1 函数描述

这个宏计算一个结构体内元素的内存偏移量，以字节为单位。用它来计算NV API函数使用的参数的偏移量是有用的。

10.5.2 函数原型

osal_offsetof(type, member)

10.5.3 参数描述

type—结构体类型。
member—结构体成员。

版权声明