RivieraWaves 系统内核

——疯壳·开发板系列

Wolverine-Team

2015/3/31

目录

官网地址：http://www.fengke.club
购买链接：http://shop115904315.taobao.com/
官方 QQ 群：193836402

术语与定义

RW RivieraWaves 系统
ID 身份（Identity）

一、系统概述

1.1 RW 简介

RW 是 RivieraWaves 系统，也是该公司的名称。RivieraWaves 是专业开发无线连接性平台的主要的先进 IP 供应商，其 IP 产品集成进大批量系统级芯片(SoC)中，最新的 Wi-Fi 802.11ac 和 Bluetooth Smart 4.1 IP 是完整的解决方案，包括了软件，硬件以及系统和 RF 支持，并且已经获得主要的半导体企业部署使用。RivieraWaves 的 Wi-Fi 和 Bluetooth IP 以备有高功效的基于硬件设计和基于 DSP 设计方式提供，为 LTE 和 Wi-Fi 带来了实现差异化的灵活性和统一的平台。2014 年，CEVA 管理层于美国东部时间 7 月 9 日上午 9:00 主持讨论 RivieraWaves 并购的电话会议。

1.2 特征

RW 内核是一个小而有效率的实时操作系统，有如下特征：

（1） 交换信息；
（2） 消息存储；
（3） 定时器功能；
（4） 内核也提供用于延迟执行的时间功能。

1.3 源文件

1.3.1 ke_config.h

文件中定义了所有用来裁剪内核而改变的常量。

1.3.2 ke_env.c/ke_env.h

文件包含了内核环境定义。

1.3.3 ke_event.c/ke_event.h

包含事件处理基元。

1.3.4 ke_mem.c/ke_mem.h

堆管理模块的实现。

1.3.5 ke_msg.c/ke_msg.h

文件中包含创建和删除一个任务的调度，也包含调度机制。

1.3.6 ke_queue.c/ke_queue.h

包含不同队列的处理函数。

1.3.7 ke_task.c/ke_task.h

包含内核任务管理的实现。

1.3.8 ke_timer.c/ke_timer.h

包含创建和删除一个定时任务的调度，也包括定时器调度机制。

1.4 头文件

为了使用内核提供的服务，需要包含以下两个头文件：

ke_task.h
ke_timer.h

1.5 内核环境

内核环境结构包含事件、定时器和消息管理的队列：

（1） evt_field：发送信息队列，但还没有递送到接收器；

（2） queue_sent：发送消息队列，但还没有递送到接收器；

（3） queue_saved：递送消息队列，但是没有被接收器处理；

（4） queue_timer：定时器队列；

（5） mblock_first：指向第一个连接列表元素的指针。

如果使能内核设定，则一下内容会被加入：

（1） max_heap_used：内核使用的最大堆内存；

（2） queue_timer：递送消息队列，但是没有被接收器处理。

二、消息机制

2.1 概述

消息队列提供向任务发送一个或多个消息的机制，两个队列被定义如下：

（1） queue_sent：发送消息队列，但是还没有递送给接收器；

（2） queue_saved：递送消息队列，但是还没有被接收器处理。

完成消息发送有三个步骤：

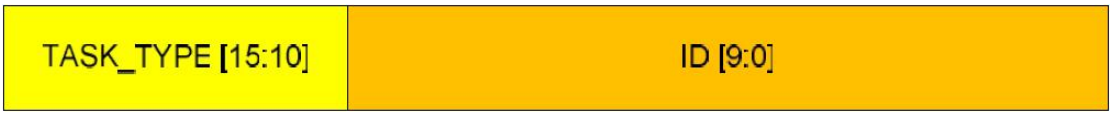
（1） 发送任务分配一个消息结构空间；

（2） 填充消息参数；

（3） 消息结构压栈至内核。

消息是通过一个唯一 ID 识别的，该 ID 包含任务类型和一个增长的数。以下宏定义构成一个任务的第一个消息的 ID：

```
#define KE_FIRST_MSG(task) ((ke_msg_id_t)((task) << 10))
```



消息有一些列参数被定义在一个结构体中。

2.2 消息对象

消息结构包括：

hdr：链表表头；

hci_type：HCI 数据类型（仅仅被 HCI 使用）

hci_off：HCI 数据在信息中的便宜（仅仅被 HCI 使用）

hci_len：HCI 的通信长度（仅仅被 HCI 使用）

id：消息 ID

dest_id：目的内核 ID

src_id：源内核 ID

param_len：结构参数的长度

param：结构参数，必须是字对齐

2.3 消息 ID

消息 ID 的定义：typedef uint16_t ke_msg_id_t;

消息 ID 应该只在一个文件中定义避免重复定义：在 xx_task.h 文件中定义 xx 任务。

2.4 参数管理

在分配消息空间时，参数的大小已经被传递，内存在内核堆栈中已经分配。为了存储这个数据，参数的指针被返回。调度机制在完成传送后，会释放内存。

```
void *ke_msg_alloc(ke_msg_id_t const id,
                   ke_task_id_t const dest_id,
                   ke_task_id_t const src_id,
                   uint16_t const param_len)
{
  struct ke_msg *msg = (struct ke_msg*) ke_malloc(sizeof(struct ke_msg) +
                                                  param_len - sizeof (uint32_t));
  ...
  return param_ptr;
}
```

2.5 消息队列对象

消息队列被定义为一个由消息元素构成的关联列表：

*first：指向列表中第一个元素的指针；

*last：指向最后一个元素的指针。

如果使能内核配置，则下面的内容被添加：

cnt：列表中的元素数量；

maxcnt：列表中元素的最大数量；

mincnt：列表中元素的最少数量。

2.6 消息队列基元

2.6.1 消息空间分配

函数声明：void *ke_msg_alloc(ke_msg_id_t const id,
ke_task_id_t const dest_id,
ke_task_id_t const src_id,
uint16_t const param_len);

参数：id 为消息的 ID 号；dest_id 为目的任务的 ID 号；src_id 为源任务的 ID 号；param_len 为参数长度。

返回值：指向参数成员变量 ke_msg 的指针。如果参数结构为空，则指针指向消息的结尾并且不应该被使用（除非重新得到消息指针或者为了发送消息）。

描述：这个基本函数为必须发送的消息分配内存空间。这个内存是在堆栈中动态分配的，并且为了分配准确的内存大小，必须提供参数结构体变量的长度。

2.6.2 消息发送

函数声明：void ke_msg_send(void const *param_ptr);

参数：param_ptr 为指向需要被发送的消息成员变量参数的指针。

返回：无。

描述：发送一个预先被任何类似 ke_msg_alloc()的函数分配内存的消息。内核将释放消息内存。

注：一旦调用该函数，用户不可能获取到它的数据，因为内核可能复制这个信息，并且释放了原始的内存。

2.6.3 消息基本发送

函数声明：void ke_msg_send_basic (ke_msg_id_t const id,
ke_task_id_t const dest_id,
ke_task_id_t const src_id);

参数：id 为消息的 ID 号；dest_id 为目的任务的 ID 号；src_id 为源任务的 ID 号。

返回值：无。

描述：发送一个参数长度为 0 的消息。由于它在内部完成，所以不需要分配空间。

2.6.4 消息转发

函数声明：void ke_msg_forward (void const *param_ptr,
ke_task_id_t const dest_id,
ke_task_id_t const src_id);

参数：param_ptr 为指向需要被发送的消息成员变量参数的指针；dest_id 为目的任务的 ID 号；src_id 为源任务的 ID 号。

返回值：无。

描述：通过改变消息的目的任务 ID 号与源任务 ID 号，将一个消息转发给另一个任务。

2.6.5 消息释放

函数声明：void ke_msg_free (struct ke_msg const *msg);

参数：msg 为指向被释放的消息指针。

返回值：无。

描述：释放分配的消息空间。

三、调度机制

3.1 概述

调度机制在后台主循环中被调用；

在后台循环中，内核检查事件是否为空，并执行对应位被置位的事件处理函数。

3.2 优先级管理

调度机制检查事件并获取优先级最高的事件。

下面的函数负责找到优先级最高的事件：co_clz(field);

事件的优先级在文件 ke_event.h 中定义。0 的优先级最高，然后到 31 优先级依次递减，如下：

```
enum
{
  KE_HIGHEST_PRIO,
  …
  KE_LOWEST_PRIO,
  KE_WVT_MAX
};
```

3.3 调度算法

（1）在后台循环中，内核将第一个消息从队列中弹出。

（2）根据消息的结构（消息，目的任务），内核决定调用哪一个消息处理函数。

（3）根据消息处理函数的返回状态，内核执行相应的操作。

3.4 保存服务

保存服务可以保存一个消息，例如保存一个消息到内存中，而不被执行。如果在接收到一个消息之后任务状态改变，则调度机制将试着在调度其它信号之前处理保存的消息。

四、任务

4.1 任务类型

被内核定义的一个常量，对每个任务唯一。

4.2 任务描述

任务描述为一个包含关于任务所有信息的结构体：

消息可以在它的任何状态下被接收；

消息可以在默认状态下被接收；

任务实例的数目；

任务的状态数目；

每个任务实例的当前状态。

内核使用一个指针指向每个任务描述，用于处理从一个任务发送给另一个任务的消息调度。

五、内核定时器

5.1 概述

RW 内核提供一个参考时间；

RW 内核提供定时器服务：启动、停止定时器；

定时器通过一个保留队列的延时消息实现；

定时器消息没有参数。

5.2 时隙定义

时间通过定义的时隙来计量，最小步长为 10ms。

5.3 定时器对象

定时器消息结构体中包括：

*next：指向下一个定时器的指针；

Id：消息 ID；

task：目的任务的 ID；

time：定时时间。

5.4 定时器设置

（1）当一个任务请求以后有一个定时器时，内核从空循环中分配一个定时器结构体。内核填充通过任务传递的新定时器参数（定时时间、定时器 ID、任务 ID）。

（2）内核将新定时器任务放进顺序定时器列表中的正确位置。

（3）如果新定时器是列表中的第一个，则内核编辑硬件所有目标时间来匹配新定时器的目标时间。

5.5 定时器基元

5.5.1 定时器设置

启动或重新启动一个定时器。

函数声明：void ke_timer_set(ke_msg_id_t const timer_id, ke_task_id_t const task, uint16_t const delay);

参数：timer_id 为定时器 ID 号；task_id 为任务 ID 号；delay 为定时时间值，单位为 10ms。

返回值：无。

描述：如果定时已经存在，则首先取消该定时，然后创建一个新定时。定时器可以是一次性的或者是间断性的，例如：可以再每次触发之后自动设置一次。

5.5.2 定时器清除

删除一个已经注册的定时器。

函数声明：void ke_timer_clear(ke_msg_id_t const timer_id, ke_task_id_t const task);

参数：timer_id 为定时器 ID 号；task_id 为任务 ID 号。

返回值：无。

描述：该函数搜索对应的定时器 ID 与任务 ID，如果找到则停止并释放，不然返回一个错误消息。

5.5.3 定时器活动

检测指定的定时器是否活动。

函数声明：bool ke_timer_active(ke_msg_id_t const timer_id, ke_task_id_t const task);

参数：timer_id 为定时器 ID 号；task_id 为任务 ID 号。

返回值：如果指定的定时器活动则返回真，否则返回假。

描述：该函数将第一个定时器从定时队列中弹出并通过发送一个内核消息来通知对应的任务。如果这个定时器是间断性的，则再设置一次。如果是一次性的，则释放定时器。该函数也检查下一个定时器，如果他们已经期满或者即将期满则执行它们。

5.5.4 定时器期满

（1）一旦目标定时器期满，后台调度一个内核事件。在事件调度中，内核将第一个定时器从队列中弹出。

（2）通过定时器结构体（定时器 ID、目标任务），内核发送对应的消息给定时器对应的任务。

（3）内核释放执行完的定时器。

（4）内核调整硬件整体目标时间来匹配下一个定时器的时间。

六、宏定义

（1）通过任务的类型和索引构建任务 ID 号：

```
#define KE_BUILD_ID(type, index) ( (ke_task_id_t)(((index) << 8)|(type)) )
```

（2）从任务 ID 中重新获取任务类型：

```
#define KE_TYPE_GET(ke_task_id) ((ke_task_id) & 0xFF)
```

（3）从任务 ID 中重新获取任务索引：

```
#define KE_IDX_GET(ke_task_id) (((ke_task_id) >> 8) & 0xFF)
```