一图读懂 ZStack 控制核心 OSAL 的基本架构和工作机理

广东职业技术学院 欧浩源 ohy3686@foxmail.com

Z-Stack 协议栈 = OSAL 操作系统 + CC2530 硬件模块 + AF 无线网络应用

系统上电后首先进行底层硬件和程序环境的初始化,然后对系统所有的任务进行注册等处理,最后启动 OSAL 操作系统进入任务轮询,不断查询每个任务是否有事件产生。如果有任务事件发生,则执行相应的事件处理函数,如果没有,就查询下一个任务,如此往复。

```
main()
                                                  void osalInitTasks( void )
 osal_int_disable(INTS_ALL); //关闭所有中断
                                                    uint8 taskID = 0;
 HAL_BOARD_INIT();
                          //初始化系统时钟
 zmain_vdd_check();
                          //检查芯片电压
                                                    macTaskInit( taskID++ ); //初始化第0个任务, taskID = 0
 InitBoard( OB_COLD );
                          //初始化硬件I/0端口
                                                                        //初始化第1个任务, taskID = 1
                                                    nwk_init( taskID++ );
 HalDriverInit();
                          //初始化硬件驱动层
                                                                        //初始化第2个任务, taskID = 3
                                                    Hal_Init( taskID++ );
 osal_nv_init( NULL );
                          //初始化Flash存储器
                                                                        //taskID随任务数而逐一递增
                          //初始化MAC厚
 ZMacInit().
                                                    User_Init( taskID );
                                                                        //初始化第6个任务, taskID = 6
 zmain_ext_addr();
                          //确定64位网络地址
 .....
 afInit();
                          //初始化AF层
                                                                   任务号 taskID
 osalInitTasks();
                   //逐一初始化系统所有的任务
                                                                                              ¥
 osal_int_enable(INTS_ALL); //打开所有中断
                          //初始化按键、显示等
 InitBoard( OB_READY );
                                                  void osal_run_system( void )
 osal_start_system(); //启动OSAL并运行操作系统
                                                    uint8 idx = 0;
                                                                   //任务号索引
 return 0;
               void osal_start_system( void ) {
                                                    循环遍历系统所有任务的事件,不为0则有事件产生
                 for(;;)
                                                    do(if(tasksEvents[idx]) break;)while(++idx < tasksCnt);</pre>
                   osal_run_system();
                                                    如果有任务事件产生,则调用相应的事件处理进程。
                                                    events = (tasksArr[idx])( idx, events );
任务事件数组,或者叫任务事件列表。
                                                             任务事件处理函数的函数指针数组。
一个元素为一个任务事件,严格按任务号排列。
                                                              一个元素为一个任务事件处理函数的入口地址。
tasksEvents[]
                                                              tasksArr[]
                                       taskID为0的任务,
                                       事件为tasksEvents数组
                //任务号 taskID = 0
                                                                               //MAC层事件处理进程
 mac任务的事件,
                                                               macEventLoop,
                                       中索引为0的元素。
 nwk任务的事件,
                //任务号 taskID = 1
                                                               nwk_event_loop,
                                                                               //网络层事件处理进程
 Hal任务的事件,
                //任务号 taskID = 2
                                       事件处理函数为tasksArr
                                                               Hal ProcessEvent,
                                                                               //硬件层事件处理进程
                                       数组中索引为0的元素。
                //任务号 taskID = 3
                                                               MT ProcessEvent.
                                                                               //调试任务处理进程
 MT任务的事件,
                                       其他任务如此类推…
                //任务号 taskID = 4
 APS任务的事件,
                                                               APS_event_loop,
                                                                               //APS层事件处理进程
                //任务号 taskID = 5
                                                                               //ZD层事件处理进程
 ZD任务的事件,
                                                               ZDApp_event_loop,
 用户任务的事件,
                //任务号 taskID = 6
                                                               User_ProcessEvent,
                                                                               //用户事件处理进程
自定义任务事件处理函数的思路:
                                 uint16 User_ProcessEvent(uint8 task_id, uint16 events)
1. 定义一个afIncomingMSGPacket t类型
的变量,接收任务事件的消息。
2. 通过osal_msg_receive()函数从该任务
                                   afIncomingMSGPacket_t *MSGpkt;
                                                                  //指向接收消息结构体的指针MSGpkt
的消息队列中, 取出系统事件的消息。
                                   if ( events & SYS_EVENT_MSG )
                                                                  //接收到系统强制事件
3. 因为一个系统事件又包括几个消息, 所
以通过switch-case语句判断该系统事件
                                     MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( task_id );
的触发消息,从而调用相应函数。
                                     while( MSGpkt ) {
常用的有下面3个:
                                      switch( MSGpkt->hdr. event )
KEY_CHANGE: 按键触发。
AF_INCOMING_MSG_CMD: 收到无线数据。
                                        case KEY_CHANGE: ..... break;
ZDO_STATE_CHANGE: 网络状态发生变化。
                                        case AF INCOMING MSG CMD: .... break;
4. 通过while循环,把该任务系统事件的
                                        case ZDO_STATE_CHANGE: ..... break:
消息队列中所有的消息全部处理完毕。
5. 每处理完一个事件的消息, 都要通过
                                      osal_msg_deallocate( (uint8 *) MSGpkt );
osal_msg_deallocate()函数来释放消息
                                      MSGpkt = (afIncomingMSGPacket_t *)osal_msg_receive( task_id );
5. 每处理完一个任务事件,都要通过异或
                                    return (events ^ SYS_EVENT_MSG);
                                                                  //清除已处理和返回未处理的系统事件
的办法, 把已处理的事件清除, 同时返回
未处理的事件, 在下一个任务轮询中, 将
                                   if(events & USER01_EVENT_MSG)
                                                                   //接收到用户自定义事件01
继续处理
                                     .....}
6. 将系统强制事件处理完毕之后, 再去处
                                   if(events & USER02_EVENT_MSG)
                                                                   //接收到用户自定义事件02
理用户自定义的事件。
                                     .....}
                                   return 0:
该函数的设计思路必须掌握,很重要!
```