# 4、协议栈工作原理

**前言：**

前文已经有多次地方提及到协议栈，但是迟迟没有做一个介绍。呵呵，不是不讲，时候未到！我们需要在最适合的时候做最适合的事。今天，我们来讲述一下协议栈的工作原理，这个东西将是我们以后接触得最多的东西，从学习到项目开发，你不得不和他打交道。由于我们的学习平台是基于 **TI** 公司的，所以讲述的当然也是 **TI** 的 **Z-STACK**。

**内容讲解：**

相信大家已经知道 **CC2530** 集成了增强型的 **8051** 内核，在这个内核中进行组网通讯时候，如果再像以前基础实验的方法来写程序，相信大家都会望而止步，**ZigBee** 也不会在今天火起来了。所以 **ZigBee** 的生产商很聪明，比如 **TI** 公司，他们问你搭建一个小型的操作系统（本质也是大型的程序），名叫 **Z-stack**。他们帮你考虑底层和网络层的内容，将复杂部分屏蔽掉。让用户通过 **API** 函数就可以轻易用 **ZigBee**。这样大家使用他们的产品也理所当然了，确实高明。

也就是说，协议栈是一个小操作系统。大家不要听到是操作系统就感觉到很复杂。回想我们当初学习 **51** 单片机时候是不是会用到定时器的功能？嗯，我们会利用定时器计时，令 **LED** 一秒改变一次状态。好，现在进一步，我们利用同一个定时器计时，令 **LED1** 一秒闪烁一次，**LED2** 二秒闪烁一次。这样就有 **2** 个任务了。再进一步…有 **n** 个 **LED**,就有 **n** 个任务执行了。协议栈的最终工作原理也一样。从它工作开始，定时器周而复始地计时，有发送、接收…等任务要执行时就执行。这个方式称为任务轮询。

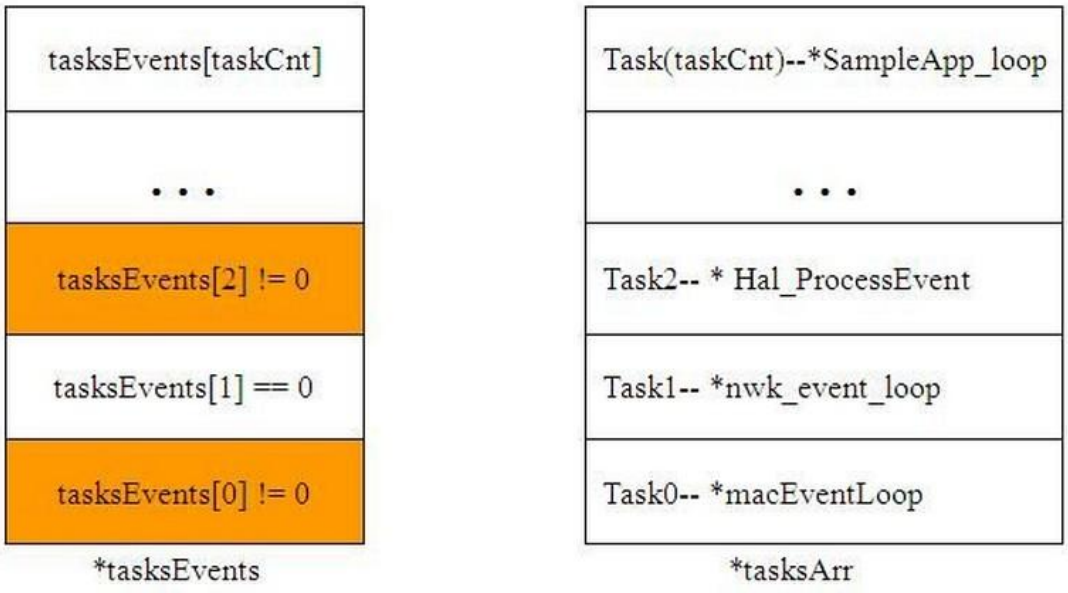



图 1 任务轮询

协议栈很久没打开了吧？没什么神秘的，我直接拿他们的东西来解剖！我们打开协议栈文件夹 **Texas Instruments\Projects\zstack** 。里面包含了 **TI** 公司的例程和工具。其中的功能我们会在用的的实验里讲解。再打开 **Samples** 文件夹：

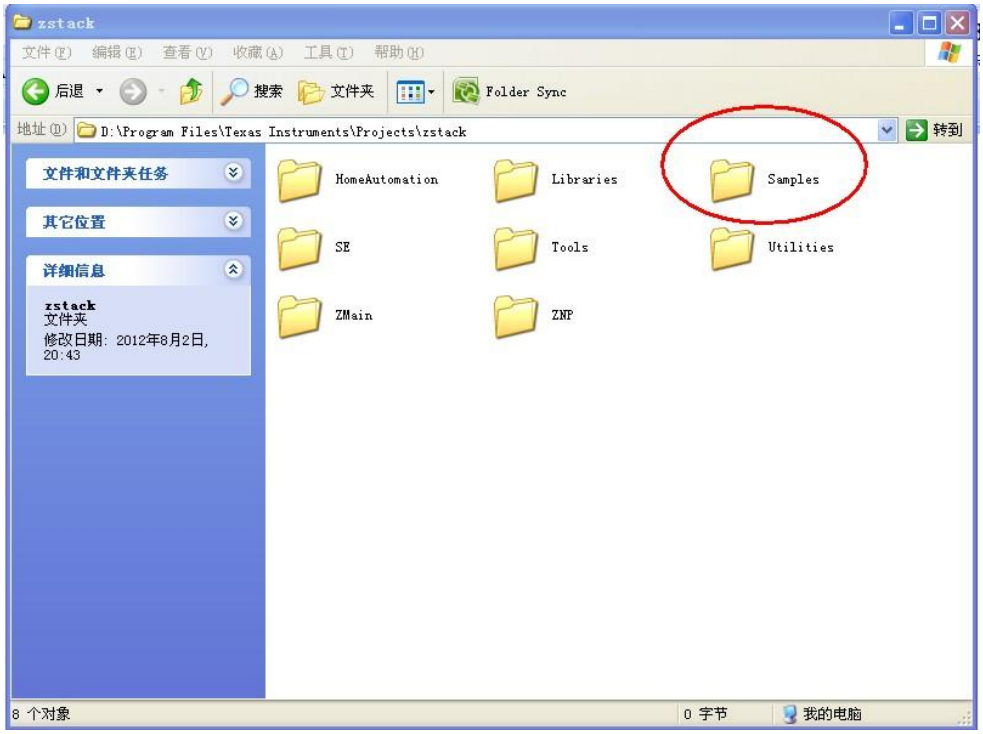



图 2

**Samples** 文件夹里面有三个例子: **GenericApp**、**SampleApp**、**SimpleApp** 在这里们选择 **SampleApp** 对协议栈的工作流程进行讲解。打开 **\SampleApp\CC2530DB** 下工程文件 **SampleApp.eww**。留意左边的工程目录，我们暂时只需要关注 Zmain 文件夹和 App 文件夹。

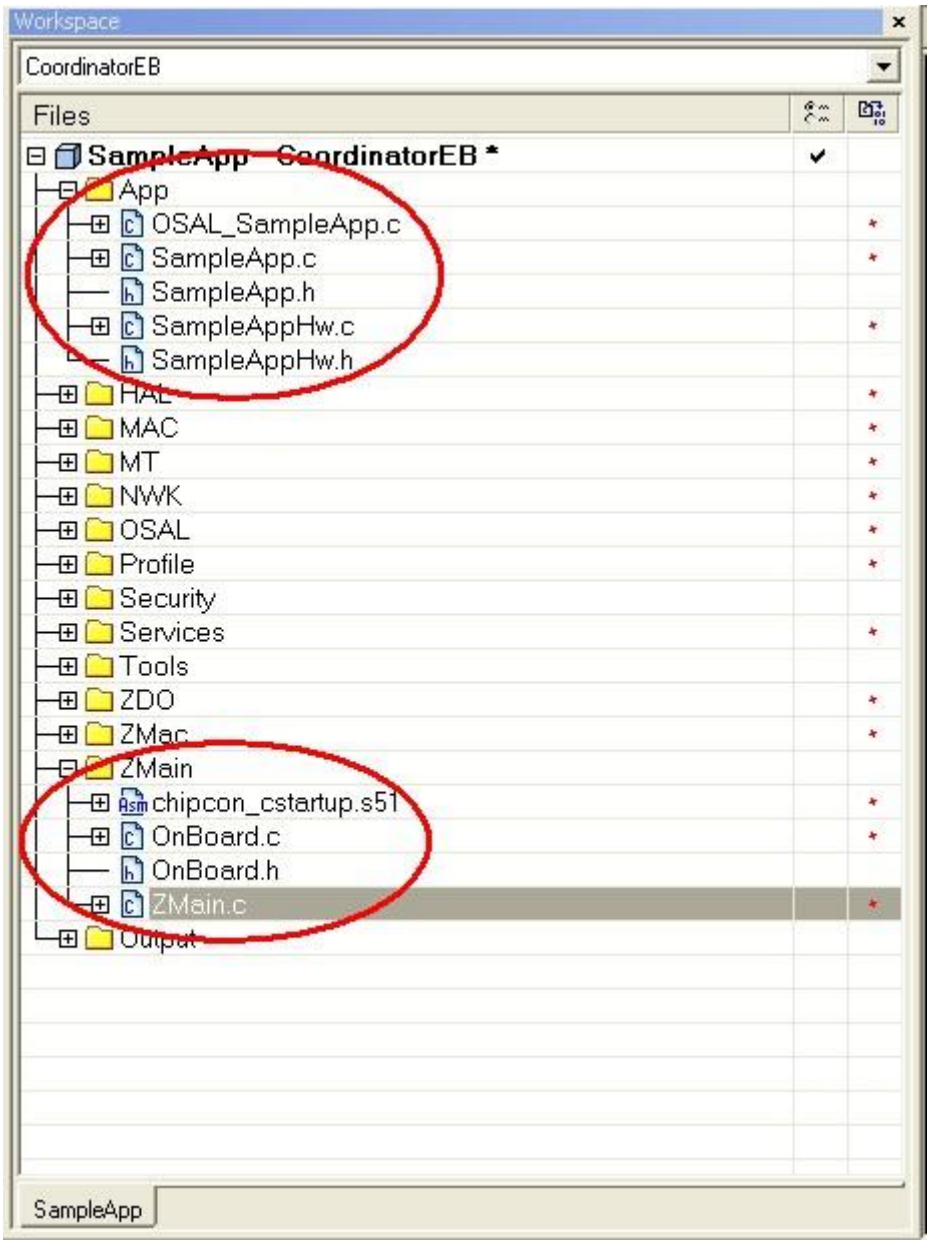



图 3

任何程序都在 **main** 函数开始运行，**Z-STACK** 也不例外。打开 **Zmain.C,** 找到 **int main( void )** 函数。我们大概浏览一下 **main** 函数代码：

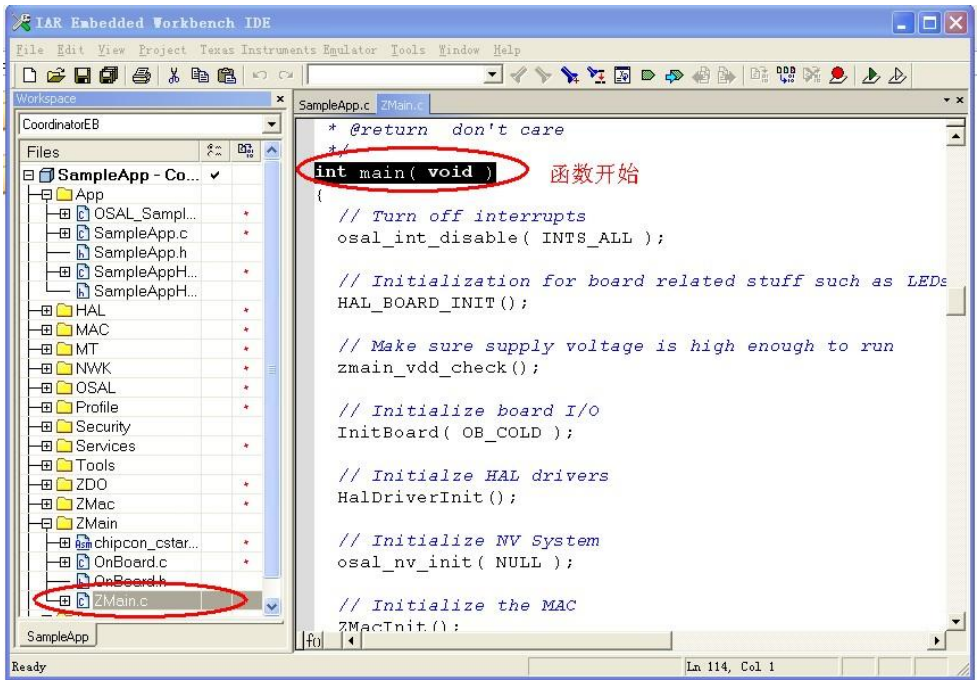



图 4

```
/*************************************************************************
*
 * @fn          main
 * @brief       First function called after startup.
 * @return      don't care
 */
int main( void )
{
  // Turn off interrupts
  osal_int_disable( INTS_ALL );     //关闭所有中断

  // Initialization for board related stuff such as LEDs
  HAL_BOARD_INIT();                 //初始化系统时钟

  // Make sure supply voltage is high enough to run
  zmain_vdd_check();                //检查芯片电压是否正常

  // Initialize board I/O
  InitBoard( OB_COLD );             //初始化 I/O ， LED 、Timer  等

  // Initialze HAL drivers
  HalDriverInit();                  //初始化芯片各硬件模块

  // Initialize NV System
  osal_nv_init( NULL );             // 初始化 Flash  存储器
```

```
  // Initialize the MAC
  ZMacInit();                 //初始化 MAC 层

  // Determine the extended address
  zmain_ext_addr();       //确定 IEEE   64 位地址

  // Initialize basic NV items
  zgInit();                    // 初始化非易失变量

#ifndef   NONWK
  // Since the AF isn't a task, call it's initialization routine
  afInit();
#endif

  // Initialize the operating system
  osal_init_system();       // 初始化操作系统

  // Allow interrupts
  osal_int_enable( INTS_ALL );     // 使能全部中断

  // Final board initialization
  InitBoard( OB_READY );       // 初始化按键

  // Display information about this device
  zmain_dev_info();           //显示设备信息

  /* Display the device info on the LCD */
#ifdef LCD_SUPPORTED
  zmain_lcd_init();
#endif

#ifdef   WDT_IN_PM1
  /* If WDT is used, this is a good place to enable it. */
  WatchDogEnable( WDTIMX );
#endif
  osal_start_system(); // No Return from here  执行操作系统，进去后不会返回
  return 0;                // Shouldn't get here.
}
```

我们大概看了上面的代码后，可能感觉很多函数不认识。没关系，代码很有条理性，开始先执行初始化工作。包括硬件、网络层、任务等的初始化。然后执行 osal_start_system();操作系统。进去后可不会回来了。在这里，我们重点了解 2 个函数：

1、初始化操作系统

**osal_init_system();**

2、运行操作系统

**osal_start_system();**

*怎么看？在函数名上单击右键——**go to definition of…**，便可以进入函数。

1、 我们先来看 **osal_init_system();**系统初始化函数，进入函数。发现里面有 6 个初始化函数，没事，我们需要做的是掐住咽喉。这里我们只关心 **osalInitTasks();**任务初始化函数。继续由该函数进入。

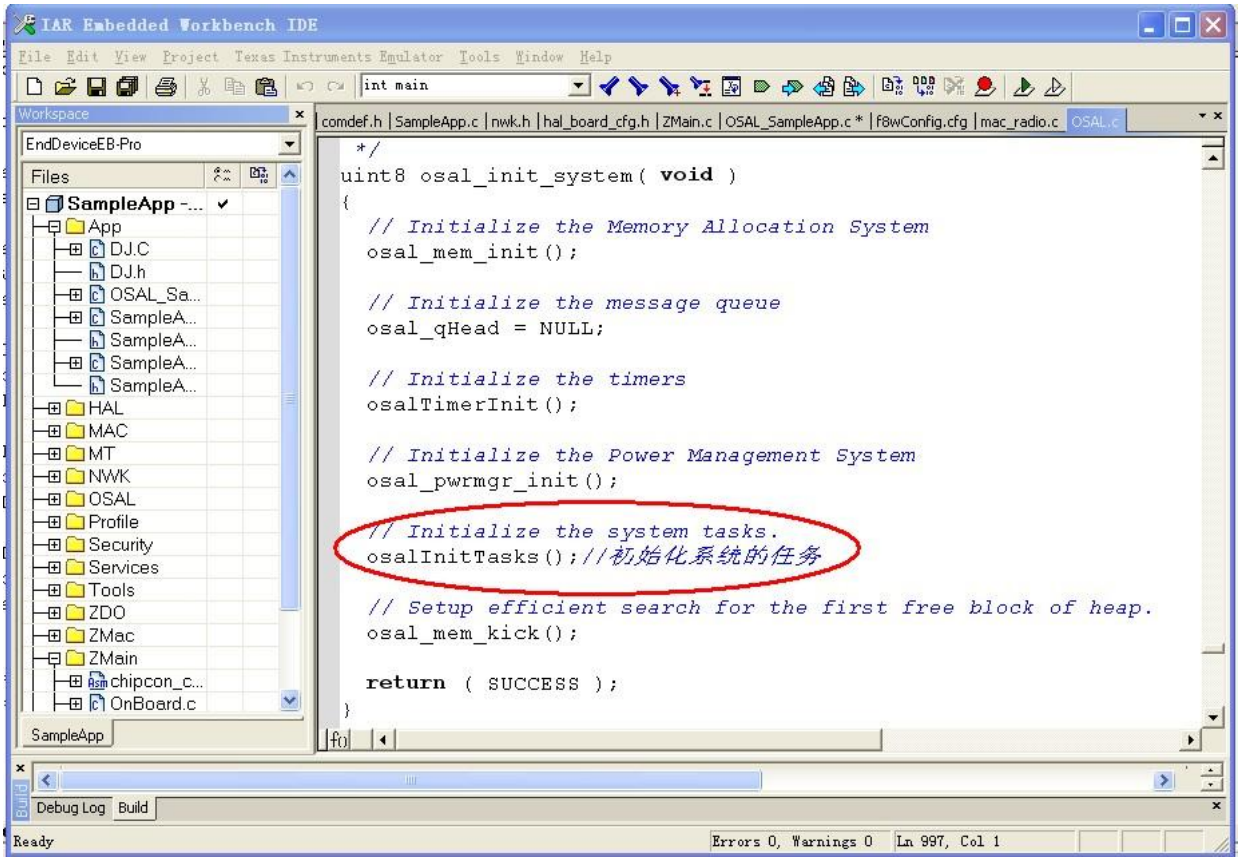



图 5

终于到尽头了。这一下子代码更不熟悉了。不过我们可以发现，函数好像能在 **taskID** 这个变量上找到一定的规律。请看下面程序注释。

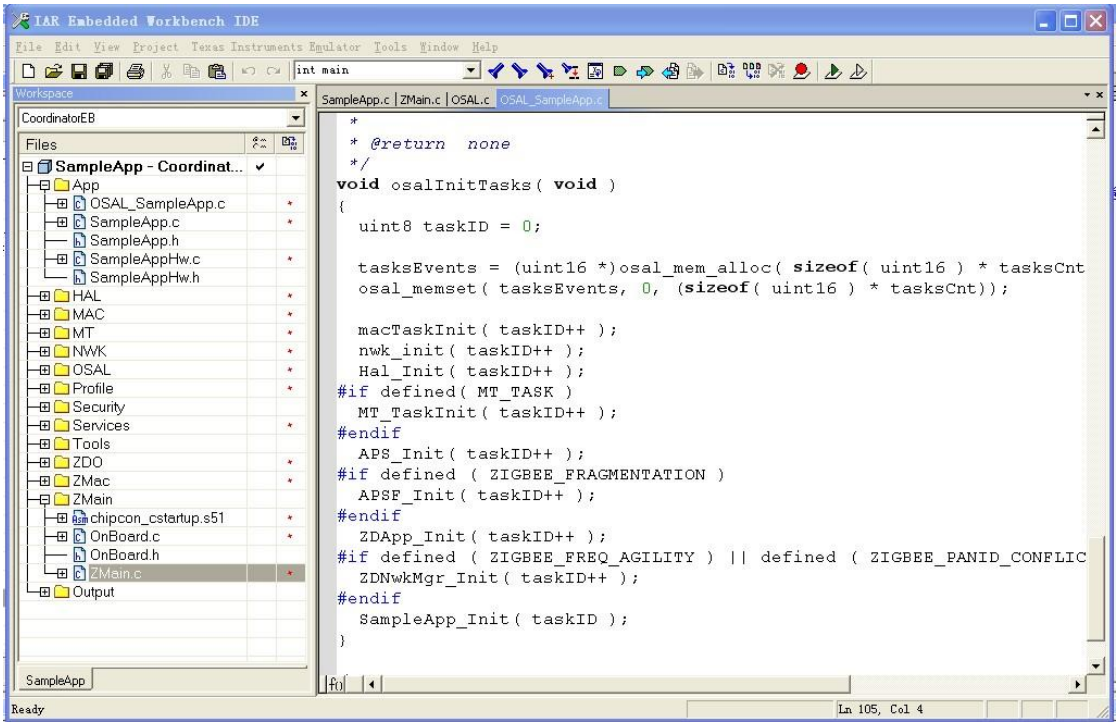



图 6

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

   // 分配内存，返回指向缓冲区的指针
   tasksEvents = (uint16 *)osal_mem_alloc( sizeof( uint16 ) * tasksCnt);

   // 设置所分配的内存空间单元值为 0
   osal_memset( tasksEvents, 0, (sizeof( uint16 ) * tasksCnt));

   // 任务优先级由高向低依次排列，高优先级对应 taskID  的值反而小
   macTaskInit( taskID++ );          //macTaskInit(0) ，用户不需考虑
   nwk_init( taskID++ );             //nwk_init(1)，用户不需考虑
   Hal_Init( taskID++ );             //Hal_Init(2) ，用户需考虑
#if defined( MT_TASK )
   MT_TaskInit( taskID++ );
#endif
   APS_Init( taskID++ );             //APS_Init(3) ，用户不需考虑
#if defined ( ZIGBEE_FRAGMENTATION )
   APSF_Init( taskID++ );
#endif
   ZDApp_Init( taskID++ );                 //ZDApp_Init(4) ，用户需考虑
#if defined ( ZIGBEE_FREQ_AGILITY ) || defined
( ZIGBEE_PANID_CONFLICT )
   ZDNwkMgr_Init( taskID++ );
```

```
#endif
   SampleApp_Init( taskID );              // SampleApp_Init _Init(5) ，用户需考虑
}
```

我们可以这样理解，函数对 taskID 个东西进行初始化，每初始化一个，taskID++。大家看到了注释后面有些写着用户需要考虑，有些则写着用户不需考虑。没错，需要考虑的用户可以根据自己的硬件平台或者其他设置，而写着不需考虑的也是不能修改的。TI 公司出品协议栈已完成的东西。这里先提前卖个关子 SampleApp_Init( taskID );很重要，也是我们应用协议栈例程的必需要函数，用户通常在这里初始化自己的东西。

至此，osal_init_system();大概了解完毕。

2、 我们再来看第二个函数 osal_start_system();运行操作系统。同样用 go to definition 的方法进入该函数。呵呵，结果发现很不理想。甚至很多函数形式没见过。看代码吧：

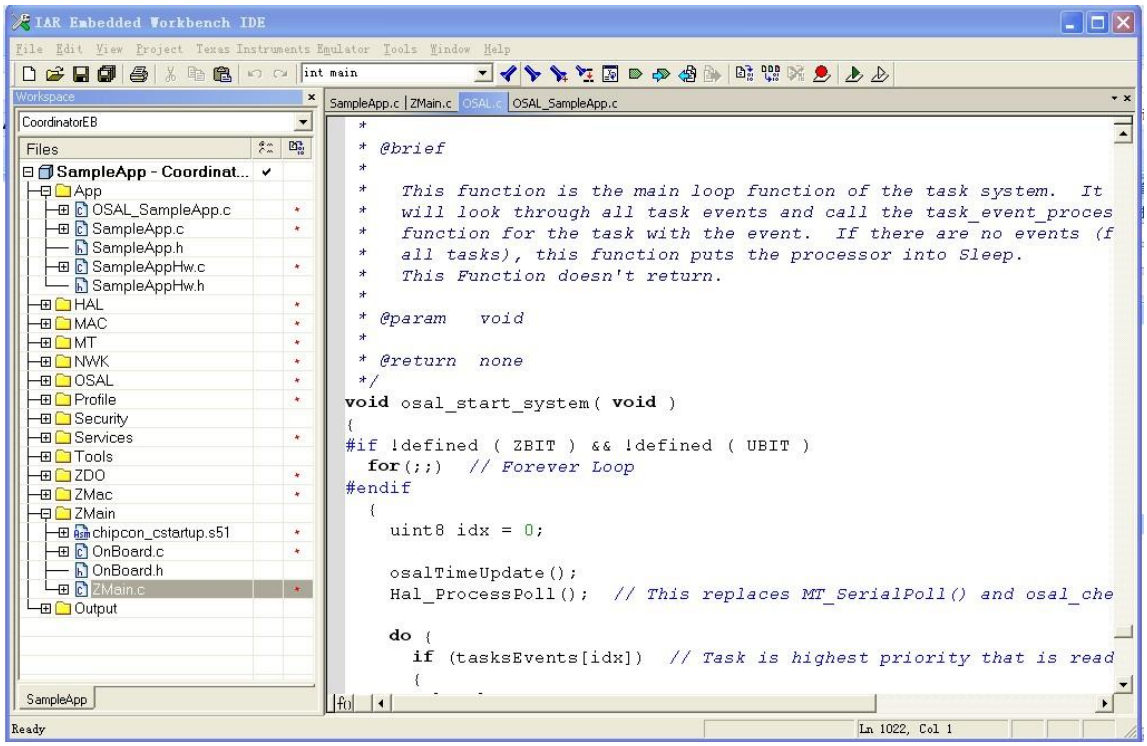



图 7

```
/**************************************************************************
 * @fn          osal_start_system

 * @brief*
 *      This function is the main loop function of the task system.    It
 *      will look through all task events and call the task_event_processor()
 *      function for the task with the event.    If there are no events (for
 *      all tasks), this function puts the processor into Sleep.
 *      This Function doesn't return.

 翻译：这个是任务系统轮询的主要函数。他会查找发生的事件然后调用相应的
        事件执行函数。如果没有事件登记要发生，那么就进入睡眠模式。这个
        函数是永远不会返回的。

  ---是不是看完官方的介绍清晰了一点？我的英语水平都可以，相信你用心也
        可以的。---

 * @param      void
 * @return    none
 *************************************************/
void osal_start_system( void )
{

#if !defined ( ZBIT ) && !defined ( UBIT )
   for(;;)    // Forever Loop
#endif
   {
       uint8 idx = 0;

    osalTimeUpdate();//这里是在扫描哪个事件被触发了，然后置相应的标志位
    Hal_ProcessPoll();    // This replaces MT_SerialPoll() and osal_check_timer().
    do {
        if (tasksEvents[idx])    // Task is highest priority that is ready.
        {
           break;    //       得到待处理的最高优先级任务索引号  idx
        }
     } while (++idx < tasksCnt);

     if (idx < tasksCnt)
     {
        uint16 events;
        halIntState_t intState;

        HAL_ENTER_CRITICAL_SECTION(intState);    // 进入临界区,保护
```

```
        events = tasksEvents[idx];              //提取需要处理的任务中的事件
        tasksEvents[idx] = 0;   // Clear the Events for this task.清除本次任务的事
件
        HAL_EXIT_CRITICAL_SECTION(intState); // 退出临界区

        events = (tasksArr[idx])( idx, events );//通过指针调用任务处理函数，关键

        HAL_ENTER_CRITICAL_SECTION(intState);//进入临界区
        tasksEvents[idx] |= events;   // Add back unprocessed events to the
                                          current task.保存未处理的事件
        HAL_EXIT_CRITICAL_SECTION(intState); // 退出临界区
      }
#if defined( POWER_SAVING )
      else    // Complete pass through all task events with no activity?
      {
        osal_pwrmgr_powerconserve(); // Put the processor/system into sleep
      }
#endif
    }
}
```

我们来关注一下 events = tasksEvents[idx]; 进入 tasksEvents[idx]数组定义，发现恰好在刚刚 osalInitTasks( void )函数上面。而且 taskID 一一对应。这就是初始化与调用的关系。taskID 把任务联系起来了。

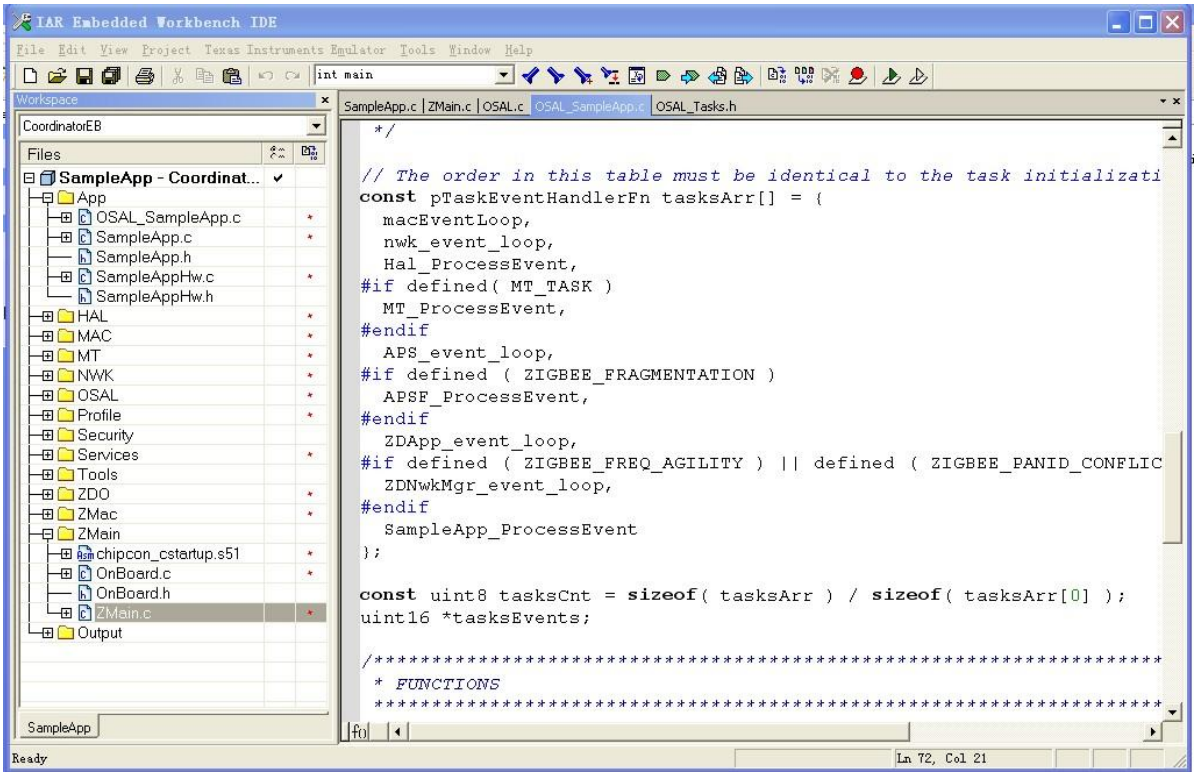
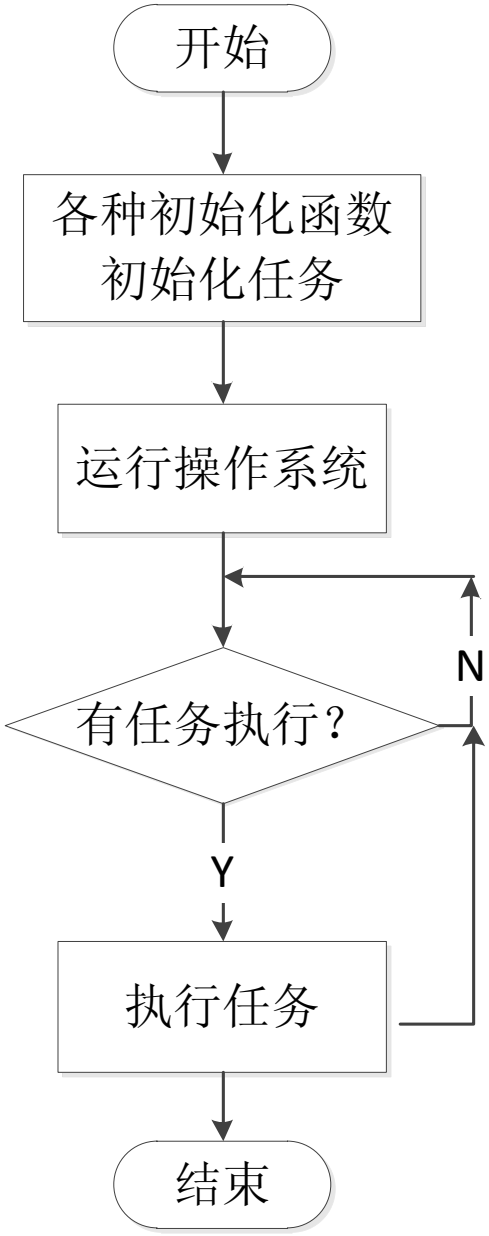



图 7

关于协议栈的介绍先到这里，其他会在以后的实例中结合程序来介绍，这样会更直观。大家可以根据需要再熟悉一下函数里面的内容。游一下这个代码的海洋。我们可以总结出一个协议栈简单的工作流程。




协议栈简要流程