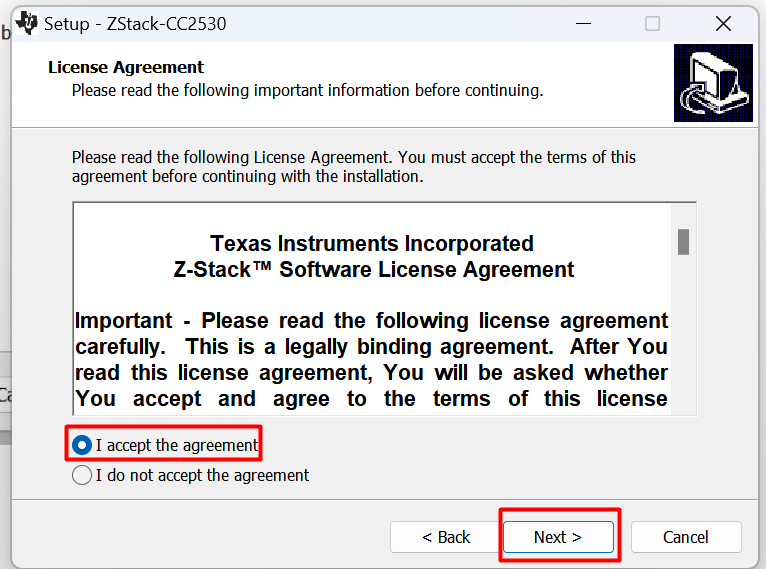
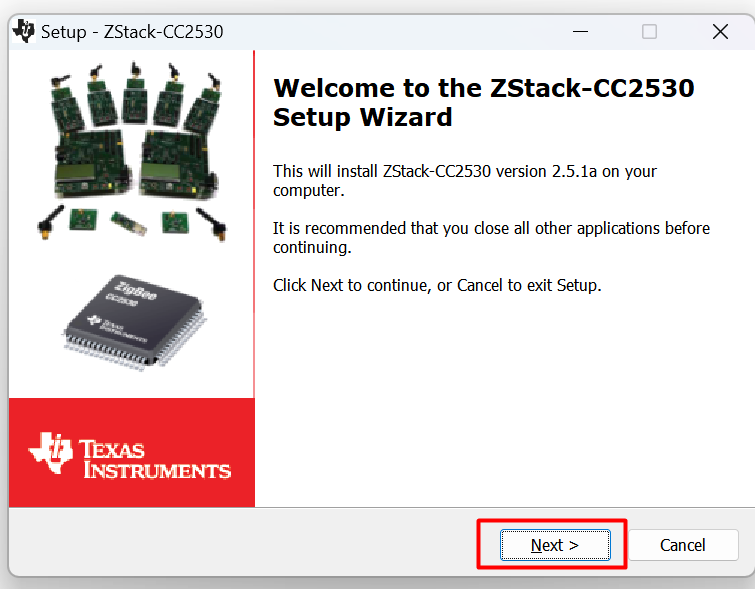
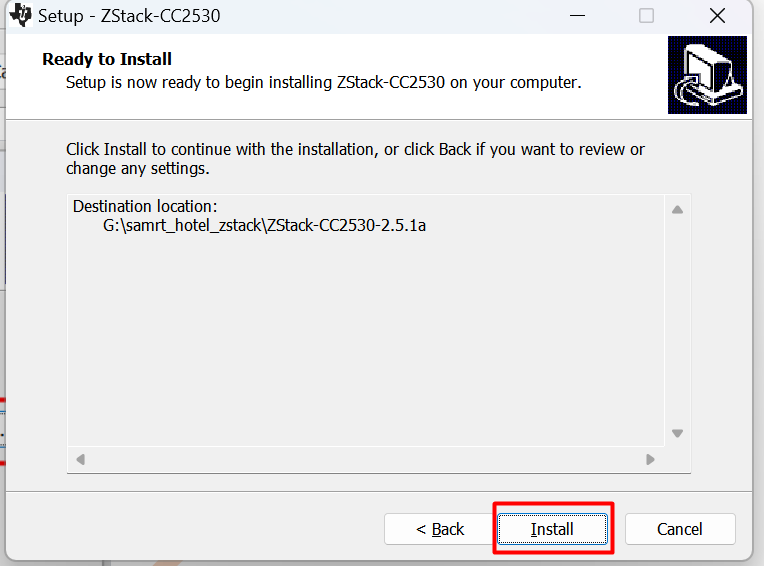
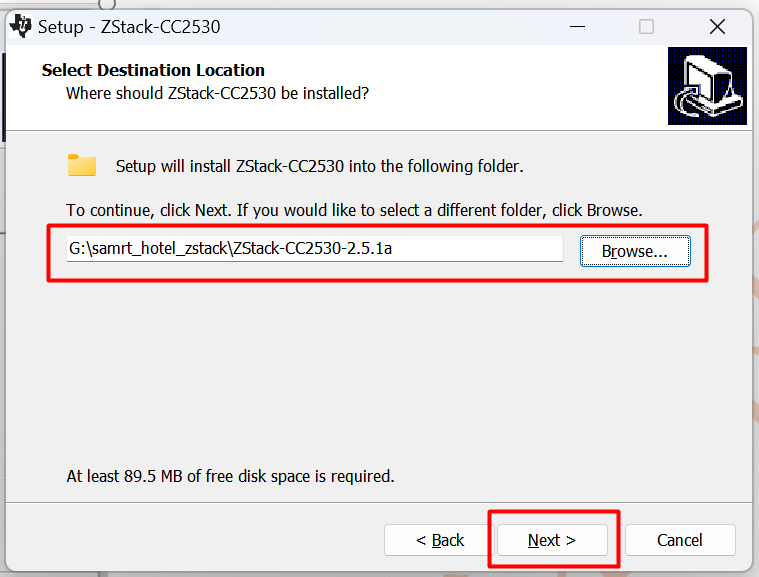
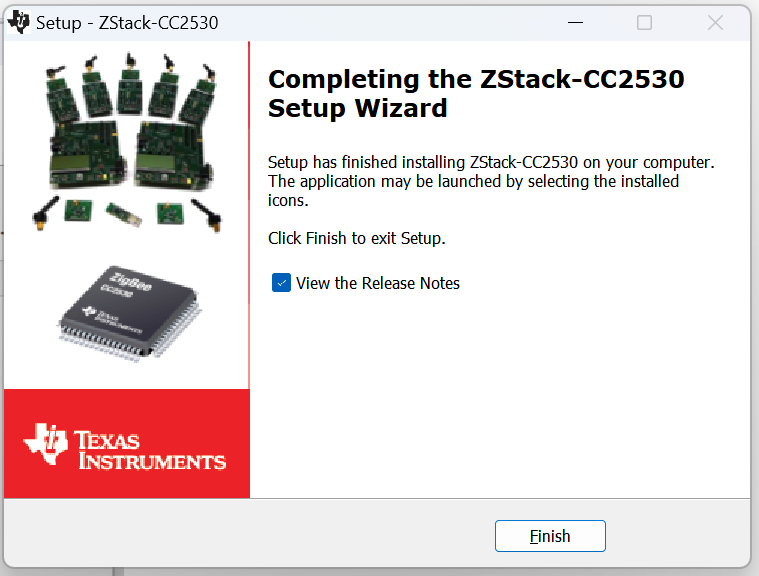
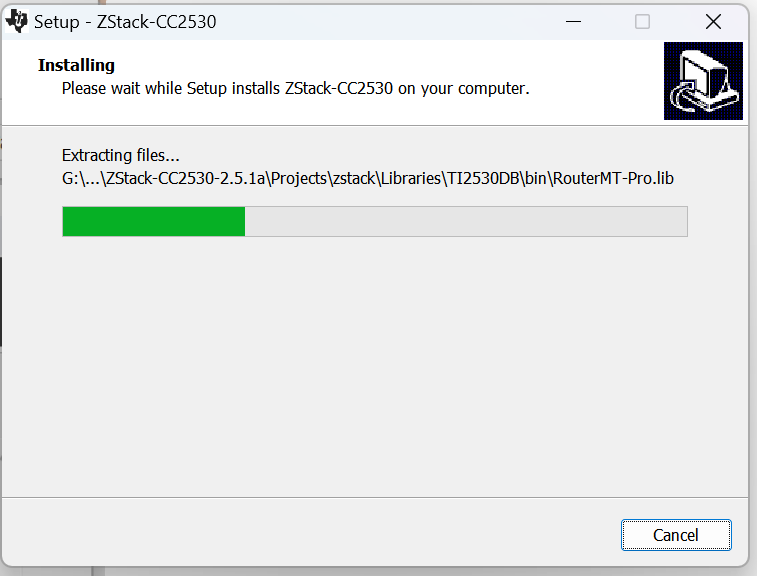
## 协议栈的安装和打开示例工程

新建一个文件夹保存安装的协议栈：

双击上述安装软件：ZStack-CC2530-2.5.1a.exe

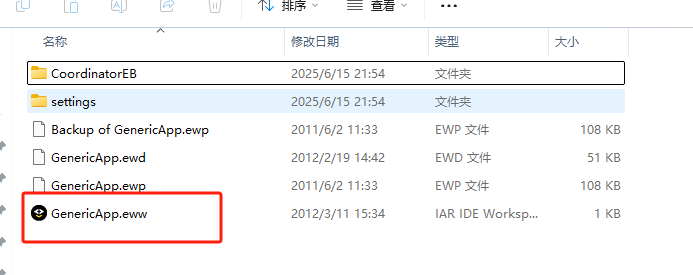




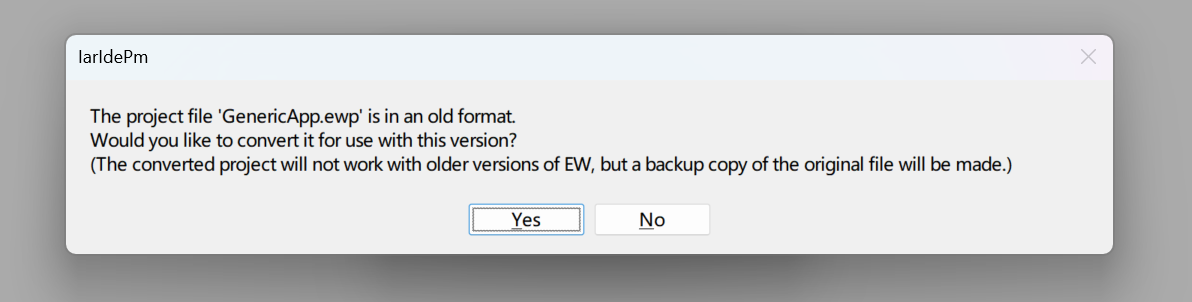


点击Finish，完成

打开协议栈例程，路径为ZStack-CC2530-2.5.1a\Projects\zstack\Samples\GenericApp\CC2530DB

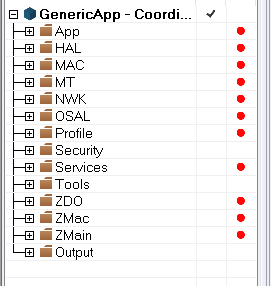


双击打开：



是否使用新版本的软件打开老的工程。 -- 点击Yes，老的IAR不能打开这个工程。

## 协议栈工程组成介绍



App（Application Programming）：应用层目录，这是用户创建各种不同工程的区域，这个目录包含了应用层的内容和这个项目的主要内容，在协议栈里面一般是以操作系统的任务实现的。

HAL（Hardware（H/W）Abstraction Layer）：硬件层目录，包含与硬件相关的配置和驱动，以及操作函数。（例如ADC、串口、LCD屏、LED灯、按键等）

MAC：MAC层目录，包含MAC层的参数配置文件及其LIB库的函数接口文件。（只提供库，看不到源码）

MT（Monitor Test）：通过串口可控各层，与各层进行直接交互，同时可以将各层的数据通过串口连接到上位机，以便开发人员调试。

NWK（ZigBee Network Layer）：网络层目录，含网络层配置参数文件，以及网络层库的函数接口文件。

OSAL（Operating System（OS）Abstraction Layer）：协议栈的操作系统 -- 非阻塞轮询

Profile：AF（Application Framework）层（应用构架）目录，包含AF层处理函数文件。ZStack的AF层提供了开发人员建立一个设备描述所需的数据结构和辅助功能，是传入信息的终端多路复用器。

Security：安全层目录，包含安全层处理函数，如加密函数等。

Services：地址处理函数目录，包括地址模式的定义及地址处理函数。

Tools：工程配置目录，包括空间划分及ZStack相关配置信息。

ZDO：ZigBee设备对象层（ZigBee Device Objects,ZDO），提供了管理一个ZigBee设备的功能。ZDO层的API为应用程序的终端提供了管理ZigBee协调器、路由器或终端设备的接口，包括创建、查找和加入一个ZigBee网络、绑定应用程序终端，以及安全管理。

ZMac:MAC层目录，包括MAC层参数配置及MAC层LIB库函数回调处理函数。

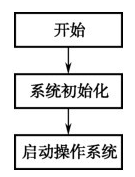
ZMain：主函数目录，包括入口函数及硬件配置文件。

Output：输出文件目录，这个是EW8051 IDE自动生成的。

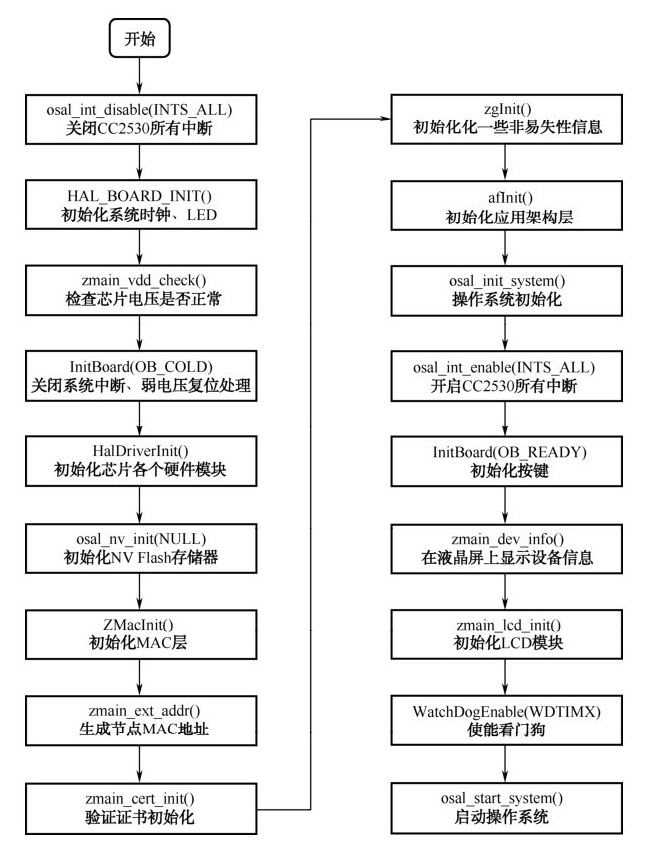
## 协议栈的启动过程

TI 公司的 ZStack 协议栈是一个基于轮转查询式的操作系统，它的main函数在ZMain目录下的ZMain.c中，总体上来说，该协议栈一共做了两件工作，一个是系统初始化，即由启动代码来初始化硬件系统和软件构架需要的各个模块；另外一个就是开始启动操作系统实体。

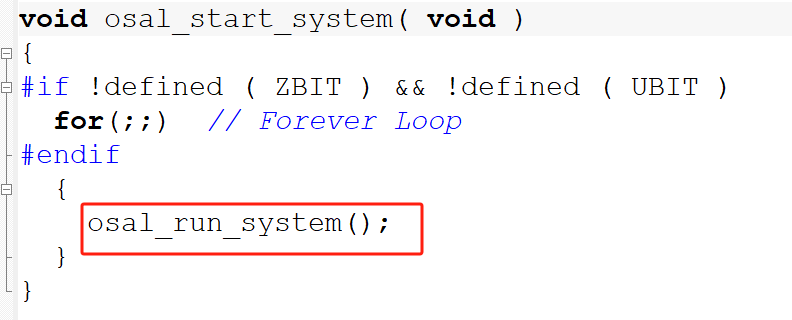




系统启动代码需要完成初始化硬件平台和软件架构所需要的各个模块，为操作系统的运行做好准备工作，主要分为初始化系统时钟、检测芯片工作电压、初始化堆栈、初始化各个硬件模块、初始化 Flash、形成芯片 MAC地址、初始化非易失变量、初始化MAC层协议、初始化应用帧层协议、初始化操作系统等部分，其具体的流程和对应的函数，



最后操作系统开始运行

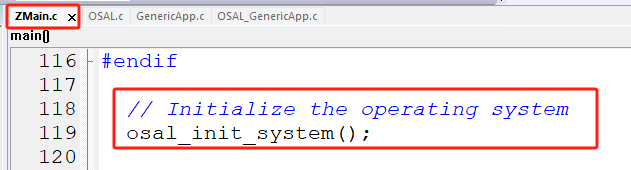


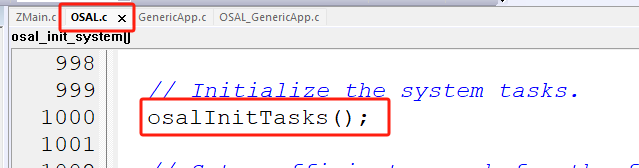
## OSAL调度管理

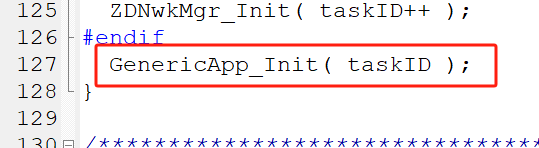
为了方便任务管理，ZStack协议栈定义了OSAL层（Operation System Abstraction Layer，操作系统抽象层）。OSAL 完全构建在应用层上，主要采用轮询的概念，并引入优先级，它的主要作用是隔离ZStack协议栈和特定硬件系统，用户无须过多了解具体平台的底层，就可以利用操作系统抽象层提供的丰富工具实现各种功能，包括任务注册、初始化和启动、同步任务、多任务间的消息传递、中断处理、定时器控制、内存定位等。

简单来说就是实现多个事件同时进行

OSAL启动并创建任务







OSAL中的任务就是不同应用功能编写的程序，其中包含几个重要的内容任务ID、事件、事件发生回调函数

任务ID：区分不同的任务。

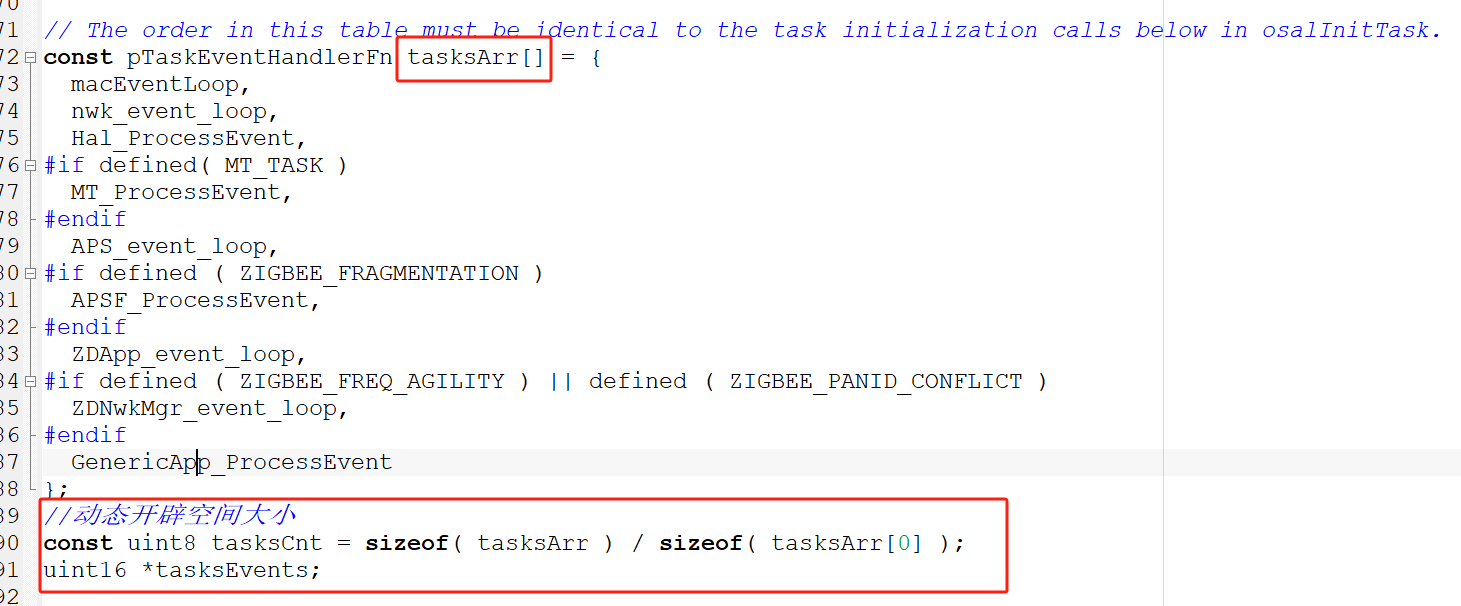
事件：在OSAL操作系下，操作系统提供了事件的功能，保证各个层的通信。例如 -- RF射频接收到数据，可以通过事件通知相应的任务执行相应的内容。

在OSAL最开始的任务初始化中系统已经创建了几个必要的任务



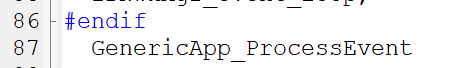
值得注意的是在OSAL的任务初始化中有一个动态开辟空间那么这个空间的大小是多少，为什么要开辟空间，开辟的空间是用来干什么呢？

1. 动态开辟空间的大小（就是taskArr的大小）



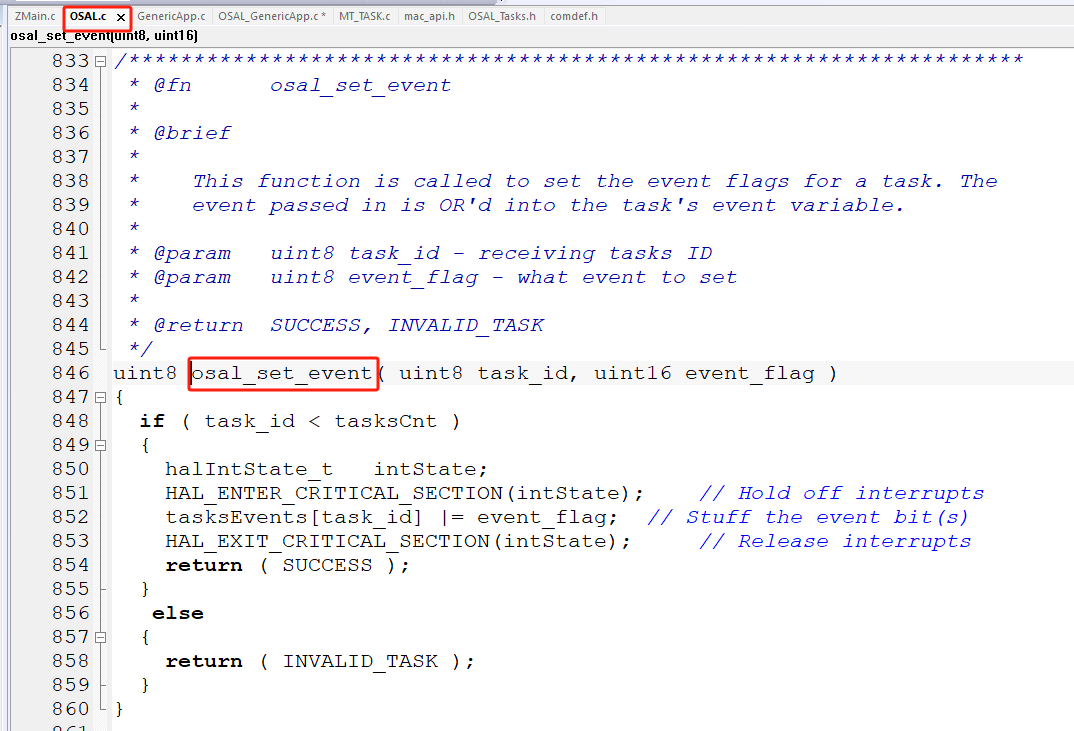
1. taskArr数组中都是函数，一旦某个任务发生了事件，OSAL如何处理这个任务的事件？里面保存的就是所有任务，一旦接收到事件后，要执行的函数（处理该事件） -- 类比STM32中的中断过程。所以数组的类型是函数指针类型。

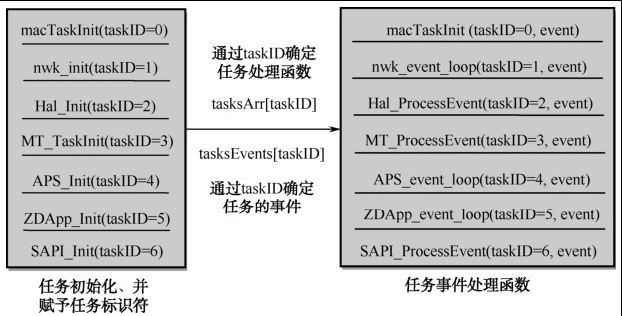
对于我们来说经常用的就是GenericApp\_ProcessEvent自己编写的任务程序



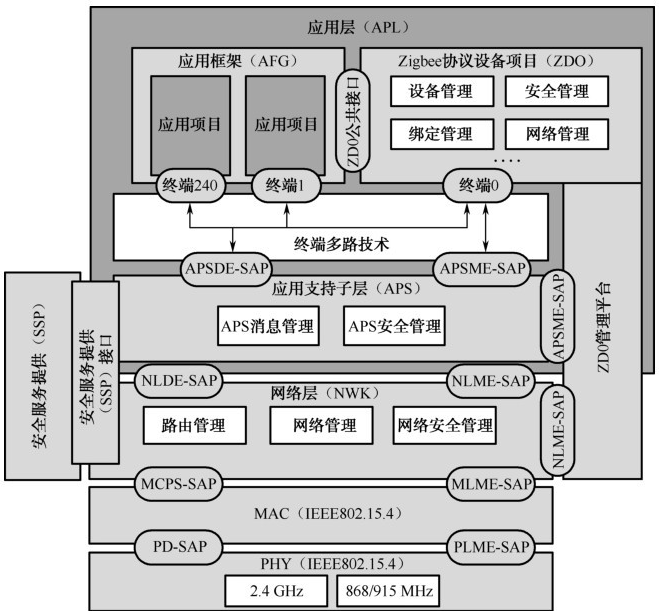
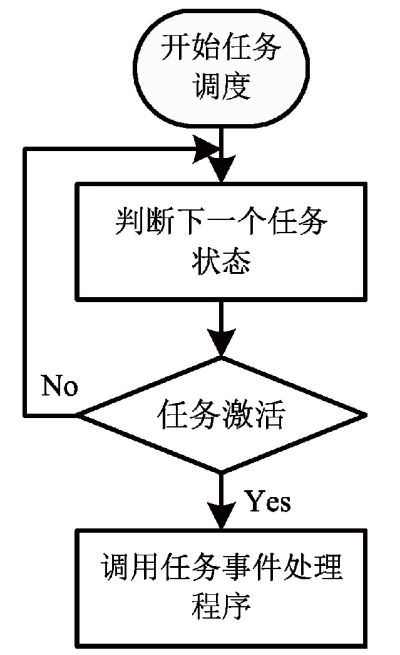
当该任务的事件发生后就会进入到这个函数中进行处理

OSAL中判断事件发生是通过tasksEvents[idx]任务事件数组来进行的。在OSAL初始化时，tasksEvents[]数组被初始化为零，一旦系统中有事件发生，就用 osal\_set\_event（）函数把tasksEvents[taskID]赋值为对应的事件。不同的任务有不同的 taskID，这样任务事件数组tasksEvents[]中就表示了系统中哪些任务存在没有处理的事件，然后就会调用各任务处理对应的事件。任务是 OSAL 中很重要的概念，它通过函数指针来调用，参数有两个：任务标识符（taskID）和对应的事件（event）。ZStack中有7种默认的任务，它们存储在taskArr 这个函数指针数组中，定义如下。



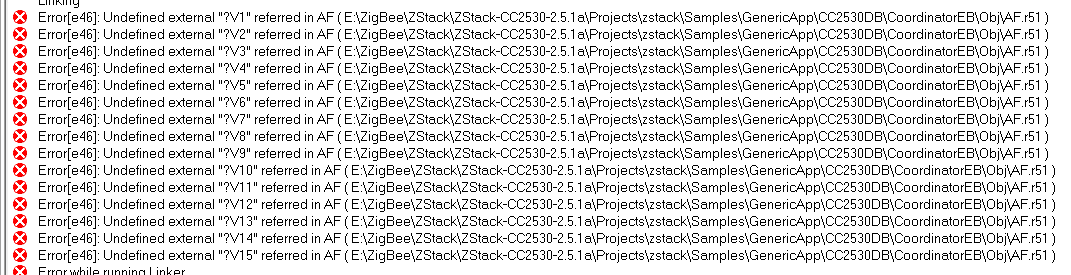


根据上述的解析过程可知，系统是按照死循环的形式工作的，模拟了通常的多任务操作系统，把CPU分成N个时间片，在高速的频率下感觉就是同时运行多个任务。

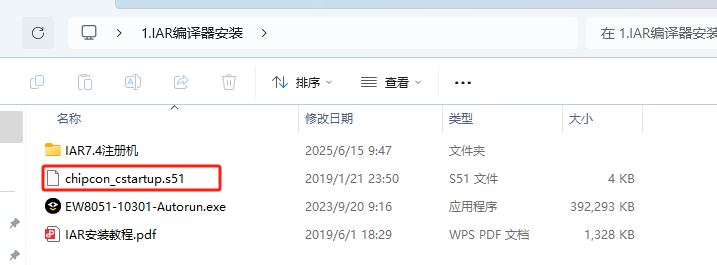


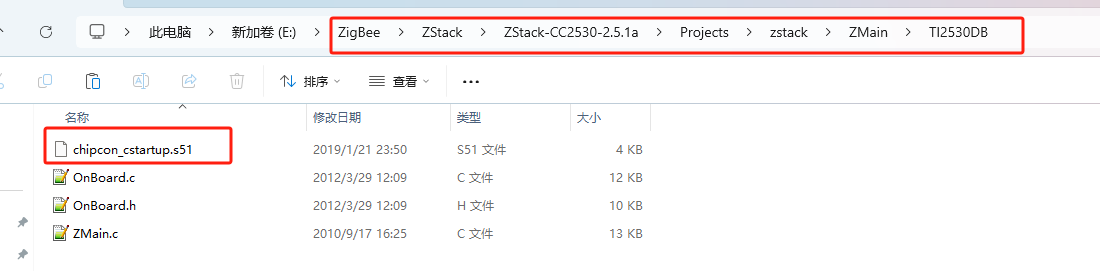
## 协议栈的实现LED灯闪烁

解决编译报错问题

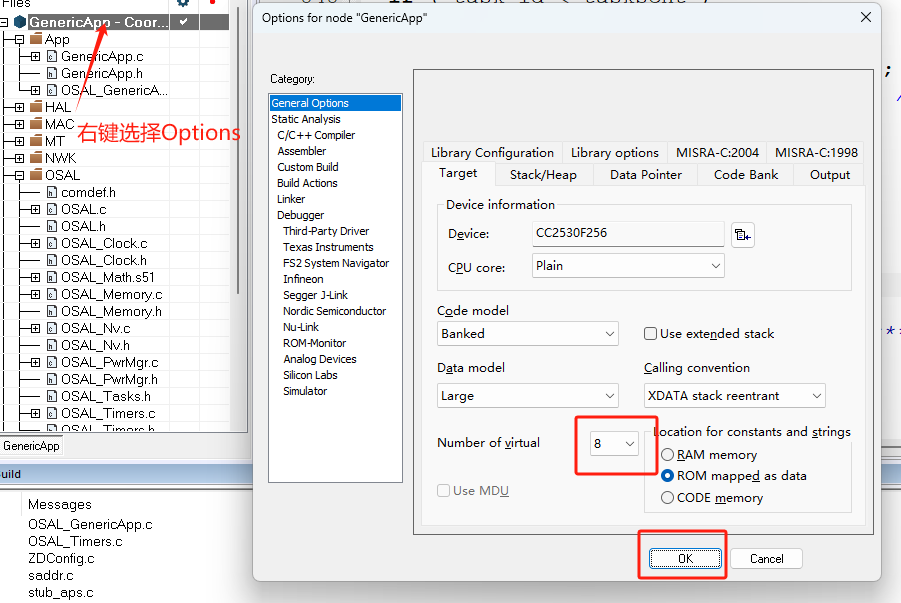


替换启动文件



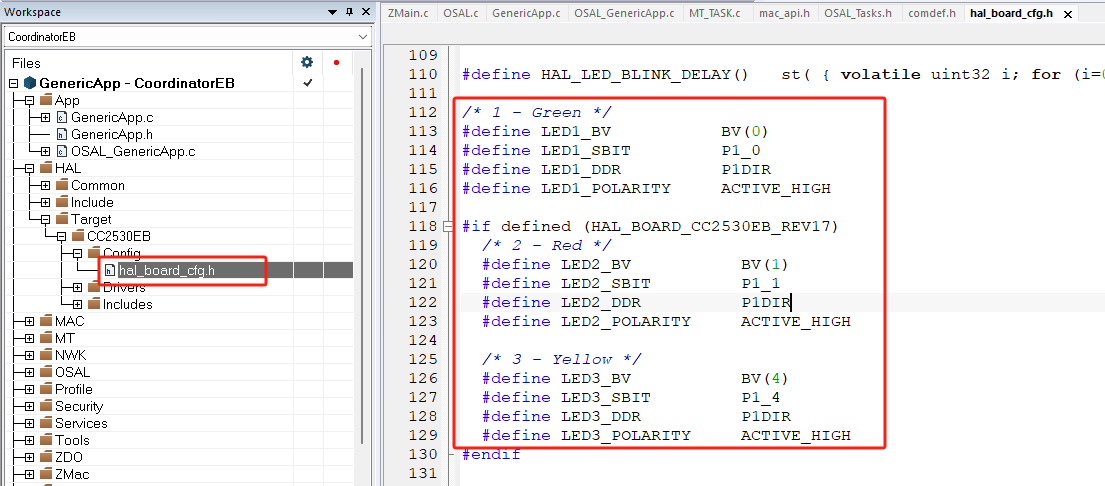


更改配置

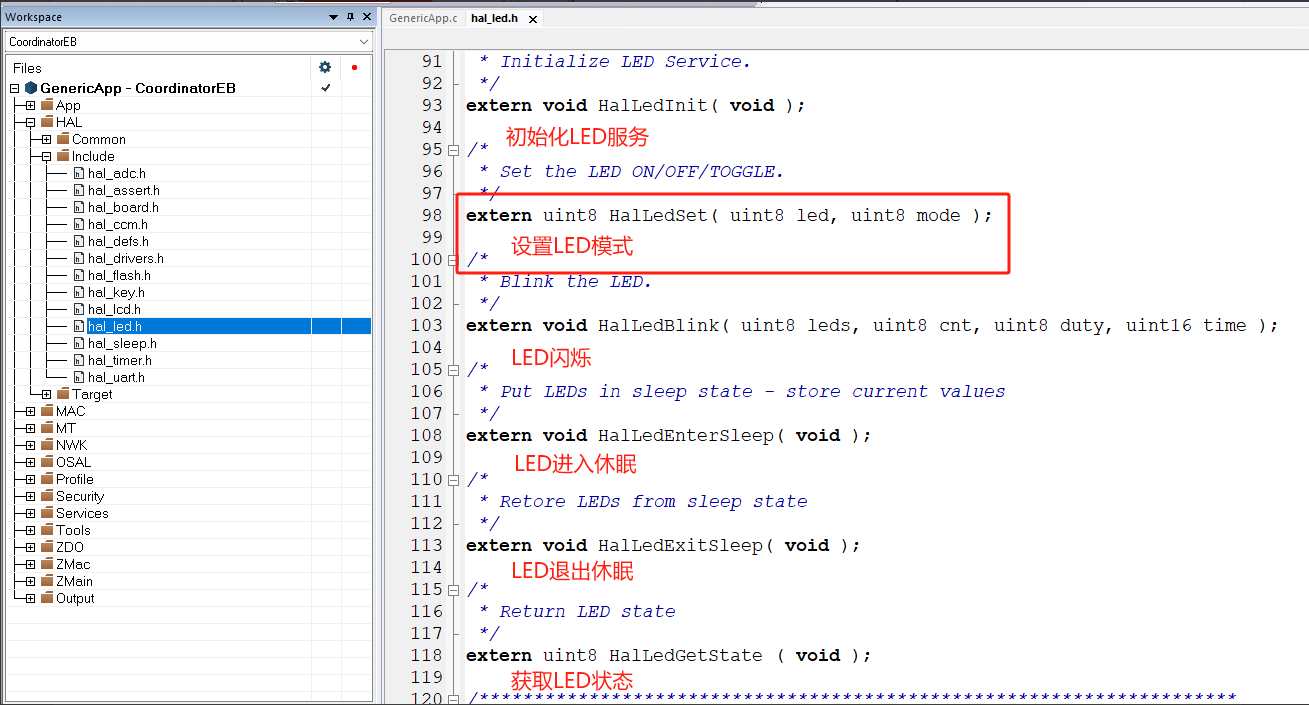


更改完成后再次编译

打开硬件接口文件：hal\_board\_cfg.h，可以发现示例中的三个灯的引脚与板子上相同那么我们就无需更改



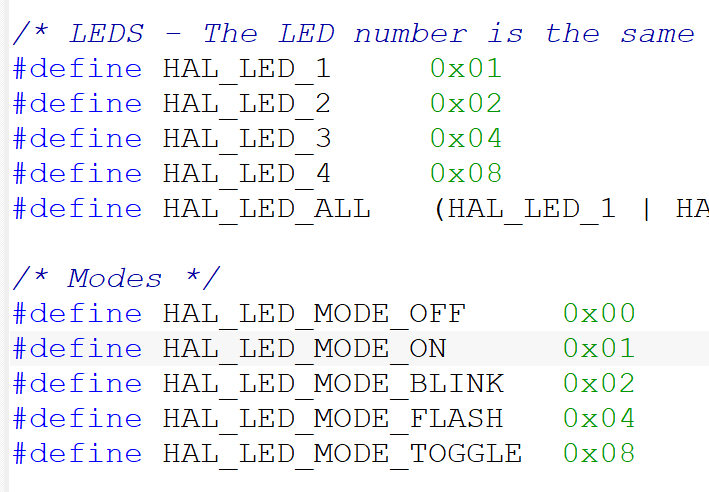
在hal\_led.h文件中可以看到我们能够使用的关于LED灯的函数接口



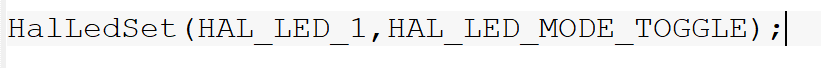
我们常用的函数为，设置LED模式



这个函数需要填写的参数为

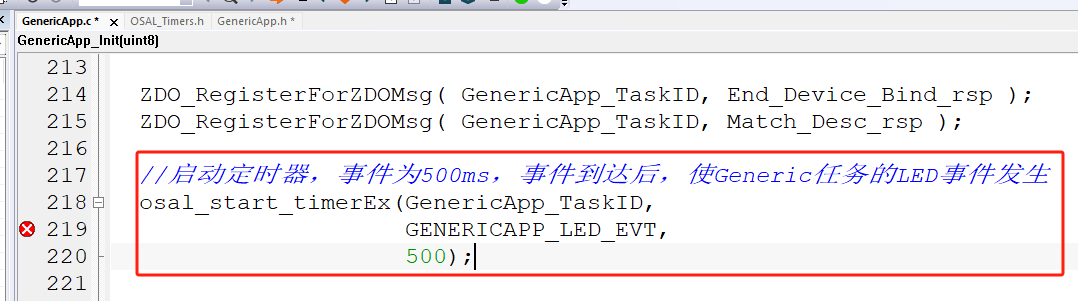


例如我们想要让LED1翻转就可以这样调用函数

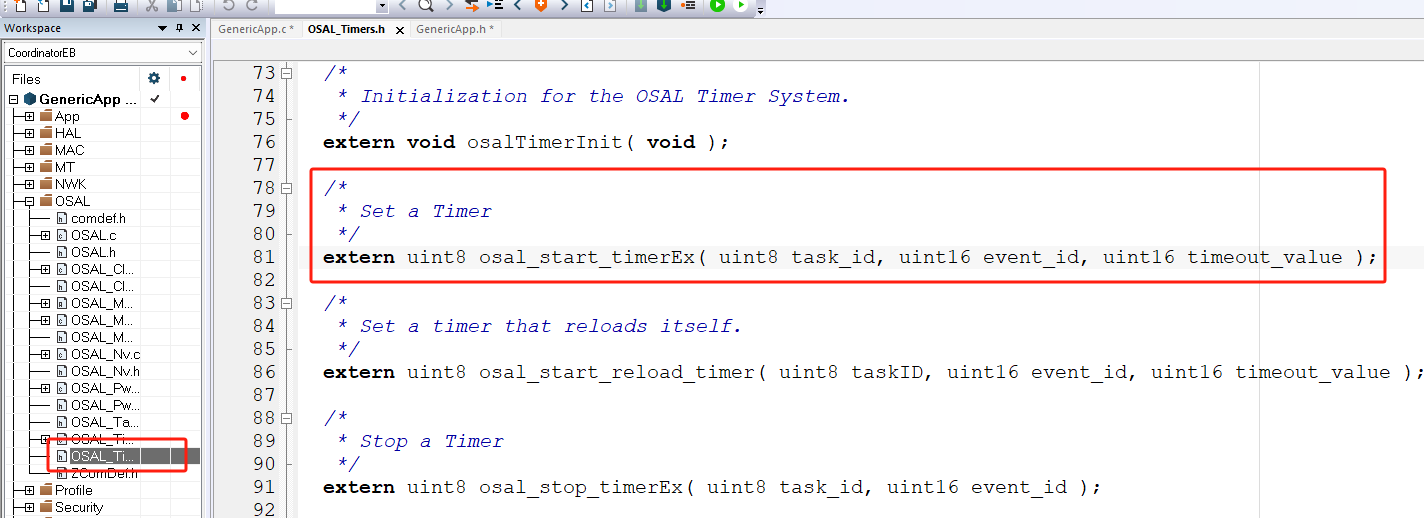


利用软件定时器产生事件，事件发生后LED灯状态翻转：

在GenericApp.c中的void GenericApp\_Init( uint8 task\_id )函数中启动软件定时器



该函数的声明位于OSAL\_Timers.h中

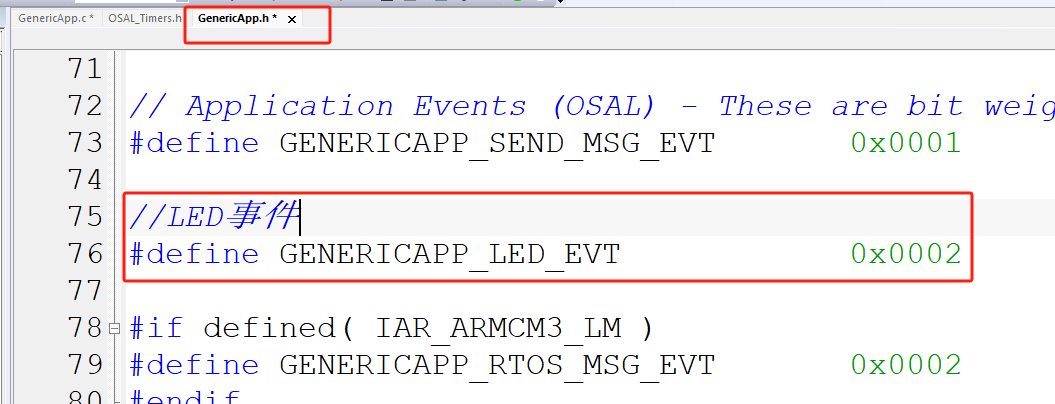


该函数需要三个参数，任务ID、定时结束产生的事件、定时的时间

任务ID：选择GenericApp\_TaskID

定时事件：500ms

但是我们缺少事件，可以在GenericApp.h中进行定义（注意事件是一个十六位的数，因此事件定义的值应该为，0x0001，0x0002，0x0004……）



最后在GenericApp\_ProcessEvent函数中判断LED事件是否发生，如果发生执行相应内容

