NXP ZigBee3.0 协议栈中软件定时器分析

(Shaozhong.Liang@nxp.com)

1. 硬件定时器

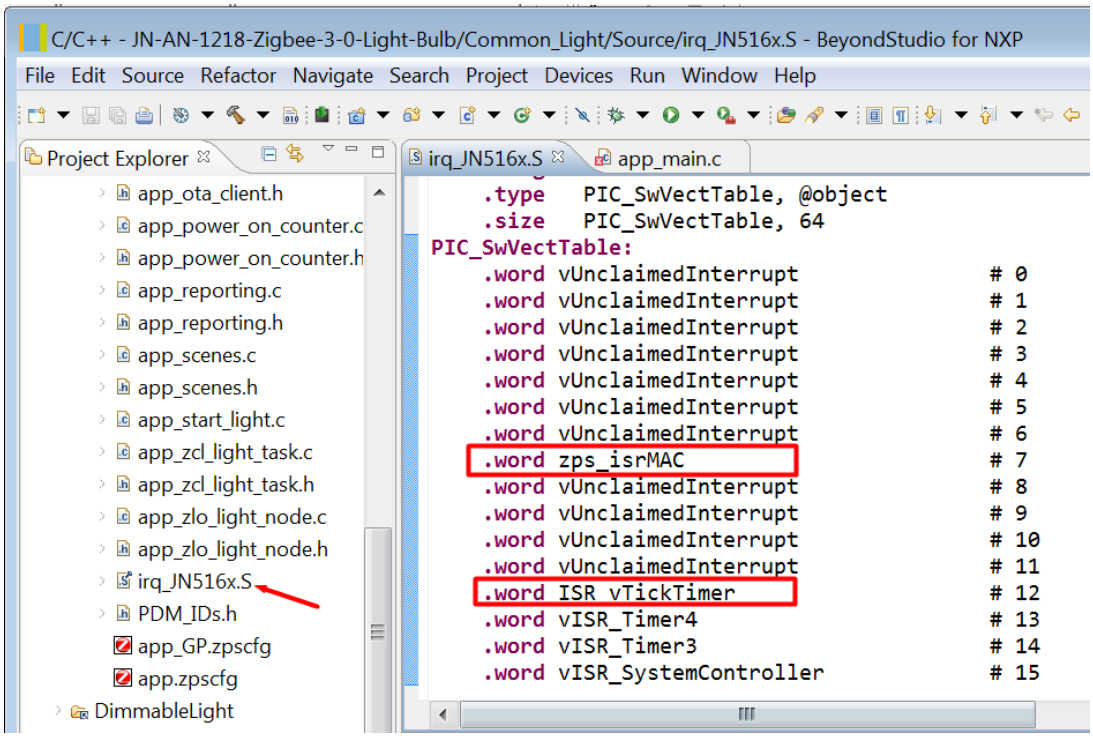
ZigBee 3.0 协议中存在很多需要定时处理的事务，包括一次性超时处理和周期性超时处理。 在协议栈里面涉及到两类定时器：

一类是硬件定时器，对应 JN-516x 芯片上的几个 Hardware Timer 定时器。

另一类是软件定时器，通过 API 函数添加到软定时器链表，再由硬件定时器中断触发进行减计数。

NXP ZigBee3.0 协议栈采用了数组和链表的方式管理软件定时器。SDK 提供了 API 启动逻辑上的软件定时器，利用硬件定时器作为时间操作的基本单元。设置时间操作的最小精度为 1ms，每 1ms 硬件定时器便产生一个时间中断。每次硬件定时器中断更新后，任务调度程序就将全部节点中的时间计数减 1，如果有软定时器计数减到 0，删除这个软定时器，并调用相应的定时器回调函数进行相应的事件处理。

在 ZigBee3.0 SDK 中，有二个与硬件定时器相关的中断服务函数，ISR_vTickTimer 和 zps_isrMAC。其中 zps_isrMAC 用于 MAC 层、NWK 层、APL 层等与协议栈底层相关的定时器。由于这部分没有开放相关源代码，这里不作深入分析，更多细节可以参考$(JN-SW-4170)\Components\ZPSTSV\Include\zps_tsv.h 头文件。下面以 JN-AN-1218-Zigbee-3-0-Light-Bulb 应用代码为例分析 ISR_vTickTimer 相关的代码。



2. ZTIMER 软件定时器

在$(JN-SW-4170)\Components\ZigbeeCommon\Source\ZTimer.c 文件中提供了 ZigBee Timer Module 软件定时器的源代码。下面是 Z Timer 的关键接口函数。

```
PUBLIC ZTIMER_teStatus ZTIMER_eInit(ZTIMER_tsTimer *psTimers, uint8 u8NumTimers);
PUBLIC void ZTIMER_vSleep(void);
PUBLIC void ZTIMER_vWake(void);
PUBLIC void ZTIMER_vTask(void);
PUBLIC ZTIMER_teStatus ZTIMER_eOpen(uint8 *pu8TimerIndex, ZTIMER_tpfCallback pfCallback, void *pvParams, uint8 u8Flags);
PUBLIC ZTIMER_teStatus ZTIMER_eClose(uint8 u8TimerIndex);
PUBLIC ZTIMER_teStatus ZTIMER_eStart(uint8 u8TimerIndex, uint32 u32Time);
PUBLIC ZTIMER_teStatus ZTIMER_eStop(uint8 u8TimerIndex);
PUBLIC ZTIMER_teState ZTIMER_eGetState(uint8 u8TimerIndex);
```

首先，用户必须预先定义 ZTIMER_tsTimer 定时器数组的大小：

```
#define APP_ZTIMER_STORAGE      (APP_NUM_STD_TMRS + APP_NUM_GP_TMRS + APP_NUM_NTAG_TMRS + APP_NUM_ZCL_MS_TMRS)
#define BDB_ZTIMER_STORAGE      (1 + BDB_INCLUDE_NS_TIMER + BDB_INCLUDE_FB_TIMER + BDB_INCLUDE_TL_TIMER)
PRIVATE ZTIMER_tsTimer asTimers[APP_ZTIMER_STORAGE + BDB_ZTIMER_STORAGE];

PUBLIC void APP_vInitResources(void)
{
    /* Initialise the Z timer module */
    ZTIMER_eInit(asTimers, sizeof(asTimers) / sizeof(ZTIMER_tsTimer));

    /* Create Z timers */
    ZTIMER_eOpen(&u8TimerButtonScan,    APP_cbTimerButtonScan,    NULL, ZTIMER_FLAG_PREVENT_SLEEP);
    ZTIMER_eOpen(&u8TimerZCL,           APP_cbTimerZclTick ,      NULL, ZTIMER_FLAG_PREVENT_SLEEP);
    ZTIMER_eOpen(&u8TimerId,            APP_cbTimerId,            NULL, ZTIMER_FLAG_PREVENT_SLEEP);
     ……..
}
```

APP_vInitResources()
→ZTIMER_eInit(…)
→vAHI_TickTimerInterval(16000); //启动 1ms 定时计数
→vAHI_TickTimerIntEnable(TRUE); //允许 1ms 定时中断
→vAHI_TickTimerConfigure(E_AHI_TICK_TIMER_RESTART);
→ZTIMER_eOpen(…) //注册用户定时器回调函数

在 1ms 硬件定时中断 ISR 服务函数中增加 Tick 计数。

```
PUBLIC void ISR_vTickTimer(void)
{
    vAHI_TickTimerIntPendClr();
    if(ZTIMER_sCommon.u8Ticks < 0xff)
    {
        ZTIMER_sCommon.u8Ticks++;
    }
}
```

在应用主循环函数中将会调用 ZTIMER_vTask 检查 Tick 计数器。如果有硬件定时器中断触发，轮询定时器数组。如果软定时器计数减到 0，调用定时器回调函数。

APP_vMainLoop()
→ ZTIMER_vTask()

```
PUBLIC void ZTIMER_vTask(void)
{
……..
    /* If no ticks to process, exit */
    if(ZTIMER_sCommon.u8Ticks == 0)
    {
        return;
    }
    /* Decrement the tick counter */
    ZTIMER_sCommon.u8Ticks--;
```

```
        /* Process all of the timers */
        for(n = 0; n < ZTIMER_sCommon.u8NumTimers; n++)
        {
            psTimer = &ZTIMER_sCommon.psTimers[n];

            /* If this timer is not opened and running, skip it */
            if(psTimer->eState != E_ZTIMER_STATE_RUNNING)
            {
                continue;
            }
            /* Decrement the time */
            psTimer->u32Time--;

            /* If the timer has not expired, move on to the next one */
            if(psTimer->u32Time > 0)
            {
                continue;
            }
            psTimer->eState = E_ZTIMER_STATE_EXPIRED;

            /* If this timer should prevent sleeping while running, decrement the activity count */
            if(psTimer->u8Flags & ZTIMER_FLAG_PREVENT_SLEEP)
            {
                PWRM_eFinishActivity();
            }
            /* If the timer has    a valid callback, call it */
            if(psTimer->pfCallback != NULL)
            {
                psTimer->pfCallback(psTimer->pvParameters);
            }
        }
    }
```

如果是低功耗睡眠设备，JN-5169 在进入睡眠之前，将会关闭 Tick Timer 硬件定时器。被唤醒后，需要重新恢复 Tick Timer 运行。

```
    PWRM_CALLBACK(PreSleep)
    {
        ……..
        /* Put ZTimer module to sleep (stop tick timer) */
        ZTIMER_vSleep();
        ……..
     }

    PWRM_CALLBACK(Wakeup)
    {
        ……..
        ZTIMER_vWake();
        ……..
     }
```

3. ZCL 软件定时器

ZCL 层也需要多个软件定时器，例如 Alarm Cluster，Automatic ZCL attribute reporting，OTA Cluster 等等。为此，ZCL 层提供另外一套软件定时器框架。由于 ZCL Timer 只服务于 ZCL 层，与用户应用代码无关。

```
PUBLIC void vZCL_EventHandler(
                        tsZCL_CallBackEvent         *psZCLCallBackEvent);
PUBLIC teZCL_Status eZCL_TimerRegister(
                        teZCL_TimerMode             eTimerMode,
                        uint32                      u32UTCTime,
                        tfpZCL_ZCLCallBackFunction  pfZCLCallBackFunction);
```

以 ZCL 的 Attribute Report Manager 为例，分析 ZCL Timer 的调用流程。在应用初始化阶段注册 ZCL Timer 回调函数 vReportTimerClickCallback，实现周期上报属性功能。

APP_ZCL_vInitialise()
→ eZCL_Initialise()
→ eZCL_CreateZCL()
→ eZCL_CreateTimer()
→ eZCL_CreateOptionalManagers()
→ eZCL_CreateReportManager()
→ eZCL_TimerRegister()

```
PUBLIC   teZCL_Status eZCL_CreateReportManager(
                    uint8       u8NumberOfReports,
                    uint16      u16SystemMinimumReportingInterval,
                    uint16      u16SystemMaximumReportingInterval)
{
    ……
    // add timer click function to ZCL
    if(eZCL_TimerRegister(E_ZCL_TIMER_CLICK_MS, 0, vReportTimerClickCallback)!= E_ZCL_SUCCESS)
    {
        return(E_ZCL_FAIL);
    }
    // initialise structure
    for(i=0;i<u8NumberOfReports;i++)
    {
        /* add all header slots to the to free list */
        vDLISTaddToHead(&psZCL_Common->lReportDeAllocList, (DNODE *)&psZCL_Common->psReportRecord[i]);
        // initialise
    }
    return(E_ZCL_SUCCESS);
}
```

JN-AN-1218-Zigbee-3-0-Light-Bulb 应用代码启动了一个 ZTIMER 软件定时器，每 10ms 调用一次 APP_cbTimerZclTick 回调函数，实现 ZCL 定时事件。

```
PUBLIC void APP_cbTimerZclTick(void *pvParam)
{
    static uint32 u32Tick10ms = 9;
    static uint32 u32Tick1Sec = 99;
```

```
    tsZCL_CallBackEvent sCallBackEvent;

    ZTIMER_eStart(u8TimerTick, ZTIMER_TIME_MSEC(10));
    ……
    /* Wrap the 1 second    counter and provide 1Hz ticks to cluster */
    if(u32Tick1Sec > 99)
    {
        u32Tick1Sec = 0;
        sCallBackEvent.pZPSevent = NULL;
        sCallBackEvent.eEventType = E_ZCL_CBET_TIMER;
        vZCL_EventHandler(&sCallBackEvent);
    }
    ……
}
```

在 APP_cbTimerZclTick 中进行计时，每一秒触发一次 vZCL_EventHandler 事件处理函数。函数 boExpiredCheck 将会检查是否有 ZCL 软件定时器到达，并调用之前注册的回调函数。

vZCL_EventHandler ()
→ vZCL_TimerSchedulerUpdate()
→ boExpiredCheck()

```
PRIVATE bool boExpiredCheck(
                        tsZCL_TimerRecord           *psTimerRecord,
                        teZCL_CallBackEventType     eEventType)
{
    ……
    switch(psTimerRecord->eTimerMode)
    {
        case(E_ZCL_TIMER_CLICK):
        {
            if(E_ZCL_CBET_TIMER == eEventType)              {
                // always call user callback after 1 sec
                psTimerRecord->pfZCLCallBackFunction(&psZCL_Common->sTimerCallBackEvent);
            }
            break;
        }
        case(E_ZCL_TIMER_CLICK_MS):
        {
            // always call user callback
            psTimerRecord->pfZCLCallBackFunction(&psZCL_Common->sTimerCallBackEvent);
            break;
        }
        ……
    }
    ……
}
```

4. 软件定时器注意事项

A. 由于 JN-5169 ZigBee 协议栈不支持动态内存分配，全部软件定时器节点必须在源代码中预先定义，并分配内存。新增加 ZTIMER 必须修改 asTimers[]数组定义。

B. 软件定时器被启动后，会调用 PWRM_eStartActivity 函数，阻止 CPU 进入睡眠状态。如果低功耗睡眠设备无法进入睡眠模式，很有可能是某些软件定时器(可能是 TSV Timer，ZTimer，ZCL Timer)仍在运行状态。

C. 如果是低功耗睡眠设备，JN-5169 在进入睡眠之前，将会关闭 Tick Timer 硬件定时器。被唤醒后，需要重新恢复 Tick Timer 运行。否则系统的 Tick Timer 硬件定时器将会失效。

D. ZCL Timer 软件定时器需要 E_ZCL_CBET_TIMER/E_ZCL_CBET_TIMER_MS 定时器事件驱动。否则，ZCL 层相关的定时操作将会失效(Reporting Timer, Identify Timer, OTA Timer, Green Power Timer 等等)。