BlueNRG-LP、BlueNRG-LPS 定时器模块

引言

BlueNRG-LP、BlueNRG-LPS 是超低功耗的 BLE 单模片上系统，符合 Bluetooth®规范。其架构核心是 32 位的 Cortex-M0+。

本文档介绍管理 BlueNRG-LP、BlueNRG-LPS 链路控制器定时器的软件模块的特性和功能。不同硬件定时器的详细描述详见在无线电控制器参考手册。

定时器模块库由两层不同程度的抽象组成，允许应用程序对与设备唤醒、用户超时或预配置的无线电事务触发相关的事件进行编程。

因此，任何 BLE 和私有无线电应用均基于定时器模块库。

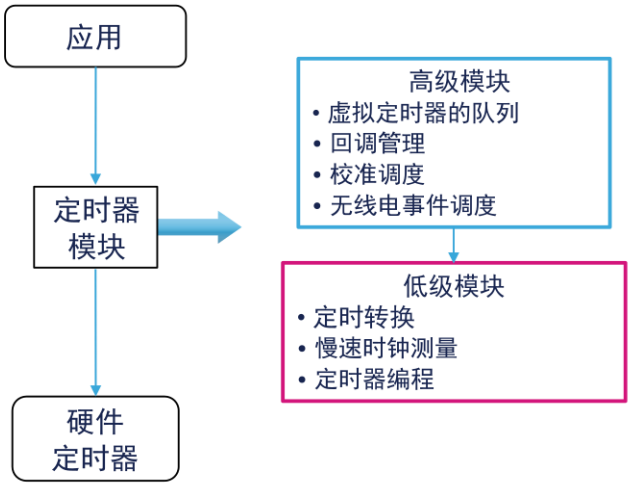
注意：　本文内容适用于 BlueNRG-LP 和 BlueNRG-LPS 设备。BlueNRG-LP 设备和平台的任何参考也适用于 BlueNRG-LPS 设备和平台。必要时，会着重标明具体的区别。

1 模块描述

模块包括四个文件：**rf_driver_hal_vtimer.c**、 **rf_driver_hal_vtimer.h**、**rf_driver_ll_timer.c**，以及**rf_driver_ll_timer.h**。这些文件代表将应用程序与硬件分开的两个层。

第一层通过允许虚拟化设备上可用资源的软件结构抽象硬件计时器。第二层直接连接到硬件，可将以“不依赖于硬件的单位”表示的时间转换为“依赖于硬件的单位”，反之亦然。这些转换考虑了硬件定时器的计数速率。

图 1. 定时器模块组件



2 虚拟定时器

BlueNRG-LP、BlueNRG-LPS 链路控制器提供定时器计数器，用于唤醒处于低功耗模式阶段的设备（而非用于触发无线电操作）。

定时器模块利用单个定时器的硬件资源，实现多个虚拟定时器的分配。

对虚拟定时器数量的唯一限制是设备上的可用内存空间。

虚拟定时器类似于普通定时器。例如，用户可以对虚拟定时器进行编程，以便在特定的时间执行某些动作。

从应用程序的角度来看，虚拟定时器是一个软件结构，除了到期时间外，还包含指向某些用户数据和回调的指针。回调是在时间耗尽时执行的例程。

该软件抽象允许在应用程序定义的虚拟定时器之间共享硬件定时器的功能（如下所述）。

虚拟定时器启动后，其实例将被放置在按到期时间排序的队列中。如果虚拟定时器先于队列中的其他事件运行，则将其置于顶部，并对硬件定时器进行编程。否则，当轮次达到时，虚拟定时器会在其他已经启动的定时器之间发生。

当一个虚拟定时器到期后，内部状态机负责执行链接到刚刚到期的虚拟定时器的回调，并为队列中的下一个定时器保留硬件计数器。

虚拟定时器的超时被视为绝对时间。这意味着，它（例如）像日历上某个特定时间的事件一样发生。

2.1 虚拟时基

在定时器模块内部，时间根据系统时间单位（STU）进行测量。它与硬件振荡器的变化无关，并直接暴露给用户。各超时事件均以 STU 表示。一个 STU 等于 625/256µs（约 2.4414µs），可以轻松表达蓝牙协议规定的时间。只有在对真正的计数器进行编程，以 STU 表示的时间才会在硬件定时器计数单元中进行转换。

以 STU 为单位的时间累积在一个 64 位长的全局变量上。如果一个数字手表每 24 小时溢出清零一次，则定时器模块的时基需要一百多万年才会溢出清零。

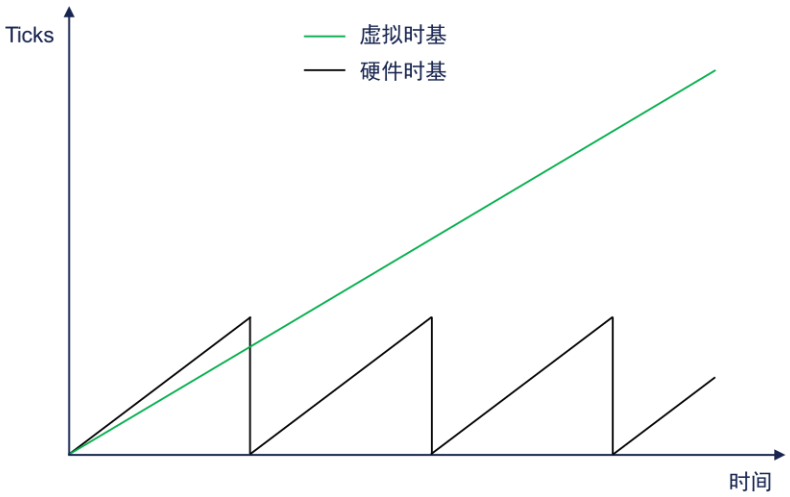
而从不溢出清零的时基由于能够显示事件的先后顺序，以及判断某事件是否为过去事件，因而具有相当高的实用性。

然而，此类时基必须处理硬件定时器的有限长度，因此也被称为“虚拟时基”。为了正确地累积时间，必须在每次硬件定时器溢出清零之前至少更新一次虚拟时基变量。

这一重要机制存在于内部，不对用户负责，由一个专用的虚拟定时器保持活动状态，且该定时器在初始化阶段由模块自动待命。

此类特殊的虚拟定时器通过定时器硬件容量允许的最大可能值定期进行编程。这意味着，处于低功耗模式的设备会被周期性唤醒，以执行时基维护。BlueNRG-LP、BlueNRG-LPS 大约每 138 分钟唤醒一次。

图 2. 虚拟和硬件时基



3 低速振荡器和校准过程

除了由虚拟定时器共享的定时器之外，另一个定时器能够触发无线电活动。在大多数情况下，无线电事务的定时是一个关键方面，必须保证一定的准确性。

低速振荡器为 BlueNRG-LP、BlueNRG-LPS 链路控制器定时器馈送信号。根据配置，低速振荡器源可以是外部 XO 或内部 RO。与外部振荡器不同，内部振荡器的速度可以改变，因其会受到温度的影响。这意味着，如果由内部振荡器计时，则定时器可以根据不固定的周期进行计数。

之后，相同的超时可对应于不同的时间间隔，具体取决于时钟频率。如果采用内部振荡器，为了保证程序化超时的准确性，要定期考虑频率的变化。在这种情况下，用于维持虚拟时基的同一虚拟定时器也负责在每个校准间隔启动校准过程，该过程包括针对利用稳定时钟源的内部振荡器，测量其一定数量的周期。测得频率之后，下一次超时（总是以 STU 表示）在机器时间单位（MTU）的硬件计数器单位中以更高的精度进行转换。

本质上，由低速振荡器的频率测量充当转换因子，允许以 STU 表示的时间转换为以 MTU 表示的时间，反之亦然。

初始化阶段结束且用户定义校准间隔后，低速振荡器频率测量由固件自主管理（作为时基维护机制）。

低速振荡器的标称频率为 32.768 kHz。如果 BlueNRG-LP、BlueNRG-LPS 配备了以标称频率运行的外部晶体振荡器，则不需要校准流程。BlueNRG-LP、BlueNRG-LPS 也可以采用外部振荡器，运行到不同的速度。在最后一种情况下，为了评估振荡器的频率，只需要进行第一次校准。

用户将处理尽可能少的硬件定时器计数单位和振荡器频率，只需应对以 STU 为单位的时间即可。

3.1 校准间隔

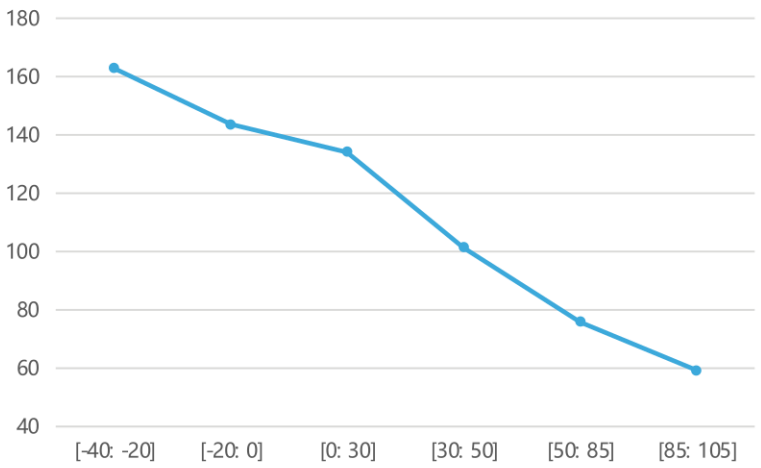
校准间隔是一个参数，可以在初始化阶段进行设置，以决定设备必须多久进行一次内部振荡器的频率测量。

如前所述，如果采用外部晶体振荡器，用户可以忽略此参数，将其置零。

定期测量内部振荡器频率的主要原因是补偿由于温度变化而引起的频率变化，并保证超时具有一定准确性（特别是在无线电操作期间）。

温度变化对内部振荡器频率的影响程度由频率温度灵敏度来描述。频率温度灵敏度在工作温度范围[-40°C：105°C]内的趋势如下。

图 3. 频率温度灵敏度（ppm/°C）



频率温度灵敏度是指温度变化一度所引起的频率误差。因此，为了计算内部振荡器必须多久测量一次，即找到校准间隔的值，有必要确定温度如何变化以及允许的最大误差。

假设温度以 0.1°C/s 的速率变化，频率温度灵敏度为 160 ppm/°C。为了确保误差低于 500ppm，必须至少每 31 秒测量一次内部振荡器的频率。

一旦设置了校准间隔，固件就会在每个校准间隔自主安排一个校准过程。校准过程持续大约 800 µs。

如果设备已经处于活动状态，则提前启动校准过程，并相应地对下一个校准事件进行编程。

4 定时器模块示例

在下面的章节中，将描述一些典型的用例。

在所有利用 BlueNRG-LP、BlueNRG-LPS 链路层定时器的应用程序内部，总是调用三个 API。

HAL_VTIMER_Init()。它根据低速振荡器的类型和高速时钟的启动时间来初始化定时器模块。此外，它还启动负责触发校准过程和虚拟时基维护操作的虚拟定时器。

该信息包含在下面定义的专用结构中：

```
typedef struct HAL_VTIMER_InitS {
      /* XTAL startup in 2.44 us unit */
      uint16_t XTAL_StartupTime;
      /* Enable initial estimation of the frequency
      of the Low Speed Oscillator */
      BOOL EnableInitialCalibration;
      /* Periodic calibration interval in ms,
      to disable set to 0 */
      uint32_t PeriodicCalibrationInterval;
} HAL_VTIMER_InitType;
```

XTAL_StartupTime 是高速时钟稳定下来所需的时间。该值以系统时间单位表示，对于无线电操作的定时特别有用。

标志 *EnableInitialCalibration* 允许在初始化期间估算低速振荡器的频率。一般情况下，如果采用外部晶体振荡器，则可将该标志置零以禁用初始估算。

PeriodicCalibrationInterval 以毫秒为单位，表示低速振荡器的测量频率（视温度变化）。同样在这种情况下，如果采用外部晶体振荡器，则可选择等于零的校准间隔来禁用定期校准。

HAL_VTIMER_Tick()。该 API 在应用程序主循环中调用。它负责管理虚拟定时器队列，检查虚拟定时器是否到期，管理硬件定时器资源的共享机制以及到期定时器的用户回调执行。

如果 PeriodicCalibrationInterval 不等于零，则会在校准定时器到期后定期启动校准过程。如果校准定时器尚未到期，但设备处于活动状态，则可提前启动校准过程。

HAL_VTIMER_TimeoutCallback()。该 API 在专用定时器 IRQ 处理程序中调用。它在硬件定时器到期时执行，向应用程序发出信号。***注意，该 API 不属于用户定义的回调***

```
void CPU_WKUP_IRQHandler(void)
{
   HAL_VTIMER_TimeoutCallback();
}
```

4.1 启动和停止虚拟定时器

虚拟定时器是一种结构，定义如下：

```
typedef struct VTIMER_HandleTypeS {
  uint64_t expiryTime; /* Absolute timeout expressed in STU */
  VTIMER_CallbackType callback; /* User callback */
  BOOL active; /* Managed by the internal tick */
  struct VTIMER_HandleTypeS *next; /* Managed by the internal tick */
  void *userData; /* Pointer to user data */
}
```

一旦在应用程序中声明了 VTIMER_HandleType，用户即可定义回调，并在定时器启动后传递所需的超时。如果需要，用户还可以定义一些由定时器句柄搭载的数据。

下面的应用示例演示如何启动虚拟定时器。使用专用 API **HAL_VTIMER_StartTimerMs()**将到期时间表示为以毫秒为单位的相对时间间隔

```
void callback(void *handle)
{
   printf(“Timer Callback after one second! \r\n”);
}

int main(void)
{
   uint32_t delay = 1000; /* One second delay */
   HAL_VTIMER_InitType VTIMER_InitStruct = {HS_STARTUP_TIME,
   INITIAL_CALIBRATION,CALIBRATION_INTERVAL};
   /* System initialization function */
   if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS) {
      /* Error during system clock configuration take appropriate action */
      while(1);
   }
   HAL_VTIMER_Init(&VTIMER_InitStruct);
   timerHandle.callback = callback;
   HAL_VTIMER_StartTimerMs(&timerHandle, delay);
   while(1) {
      HAL_VTIMER_Tick();
   }
}
```

当定时器最终到期后，回调将在 HAL_VTIMER_Tick()中触发。

如果定时器已经启动，则不能再次启动。在这种情况下，API 返回一个错误代码，并且虚拟定时器没有插入队列中。

如果延迟过短，则认为定时器已经到期，并执行相关回调。

超时也可以表示为以 STU 为单位的绝对时间。在这种情况下，使用 **HAL_VTIMER_StartTimerSysTime()**

```
static VTIMER_HandleType timerHandle;
void callback(void *handle)
{
   printf(“Timer Callback after one second! \r\n”);
}

int main(void)
{
   uint32_t delay = 409600; /* number of system units in one second */
   HAL_VTIMER_InitType VTIMER_InitStruct = {HS_STARTUP_TIME, INITIAL_CALIBRATION,
   CALIBRATION_INTERVAL};
   /* System initialization function */
   if (SystemInit(SYSCLK_64M, BLE_SYSCLK_32M) != SUCCESS) {
      /* Error during system clock configuration take appropriate action */
      while(1);
   }
   HAL_VTIMER_Init(&VTIMER_InitStruct);
   timerHandle.callback = callback;
   HAL_VITMER_StartTimerSysTime(&timerHandle, TIMER_GetCurrentSysTime() + delay);
   while(1) {
      HAL_VTIMER_Tick();
   }
}
```

得到以 STU 为单位的当前时间，并加上与 1 秒相对应的系统时间单位数。因此，到期时间变成以 STU 表示的绝对时间。

虚拟定时器启动后，可以通过调用 **HAL_VTIMER_StopTimer()**在到期之前进行停止。

```
HAL_VTIMER_StopTimer(&timerHandle);
```

5 无线电定时器

BlueNRG-LP、BlueNRG-LPS 提供另一个定时器，专门用于触发无线电事务（可以是发送或接收）。特别是，定时器模块库提供了一种可能性 - 对与无线电操作相关的两个不同事件进行编程：

- 通过空中传输的第一个传输比特
- 接收窗口的开始

注意，定时器模块只编程无线电定时器，而不为传输或接收配置无线电。此外，定时器模块没有对背靠背通信的超时进行编程。一个专用的软件库可以完成这两项任务。若需更多信息，请参考无线电驱动程序用户手册。无线电定时器不像虚拟定时器那样在定时器队列中虚拟化（无必要性），但即使在这种情况下，用户也必须将前述两事件之一的到期时间（以 STU 为单位）表示为将来的绝对时间。此外，无线电定时器在软件结构中被抽象出来，但无需用户的任何动作。如果超时时间太近，则对定时器进行编程的请求将被拒绝并返回错误代码。

5.1 校准和无线电定时器

如前所述，只有当低速内部振荡器是 BlueNRG-LP、BlueNRG-LPS 链路层定时器的时钟时，才需要校准过程。届时，通过固件适时确保下一个无线电事务的无线电定时器是利用最新的低速频率测量进行编程的，以提高超时的准确性。换言之，如果下一个无线电事务发生在下一个校准事件之后，则定时器不会立即被编程，而是保持待定状态，直至新的频率测量可用。相反，如果下一个校准事件发生在下一个无线电事务之后，则定时器将在收到请求后被编程。由于校准值在 HAL_VTIMER_Tick()内部的线程模式下可用，因此在低速振荡器测量结束后，设置一个余量，以便给模块足够的时间来用新值完成待定的定时器。如果不考虑该余量，则以前属于未来的无线电事件可能会转变为过去，因此无法编程。

5.2 无线电定时器编程示例

无线电定时器只能在事先配置了无线电事务之后被编程。然后，无线电定时器编程 API 可被视为无线电事务配置的最后一步。因此，需要一些特定的信息来正确编程无线电定时器：

- 以 STU 为单位的超时
- 事务（发送或接收）的类型
- 通道频率的 PLL 校准

根据前文参数，为了遵循期望的超时，定时器模块补偿无线电配置所需的时间。在无线电初始化期间，在 RAM 中的一些特定结构中初始化不同的 RF 设置时间。如需关于无线电 RAM 结构的更多详情，请参阅无线电控制器参考手册。

因此，假设无线电已经初始化，事务已经配置，可通过 HAL_VTIMER_SetRadioTimerValue()对无线电定时器进行编程。请注意，定时器模块初始化发生在无线电初始化之后，如“虚拟定时器示例”一节所示。

超时表示为绝对时间。例如，可将传输编程为在一秒钟后触发，如下所示：

```
uint8_t event_type = HAL_VTIMER_TX_EVENT;
uint8_t cal_req = HAL_VTIMER_PLL_CALIB_REQ;
uint32_t timeout = TIMER_GetCurrentSysTime() + 409600;
retVal = HAL_VTIMER_SetRadioTimerValue(timeout,event_type,cal_req);
```

如果超时时间过近，则 API 将返回一个错误代码。无线电定时器可以在触发前停止。但是，如果定时器停止的时间距离超时过近，则可能无法正确清除。换言之，固件始终清除定时器，但定时器可能已被触发，无法再作停止。届时可以使用专用 API 来停止无线电定时器。根据定时器是否被成功清除，相同的 API 将返回不同的值。

```
/**
* @brief 清除预定的最后一个无线电活动，同时禁用无线电定时器。
此外，如果过于接近超时时间，也将返回不同的值
而且可能无法及时清除无线电活动。
@返回 0 - 无线电活动已被成功清除。
@返回 1 - 未能及时清除最后的无线电活动。
@返回 2 - 无法清除最后的无线电活动。
*/
uint8_t HAL_VTIMER_ClearRadioTimerValue(void);
```

6 睡眠管理

定时器模块防止设备在不同情况下进入睡眠状态：

- 触发了一个虚拟定时器，但其相关的回调尚未执行
- 正在进行低速时钟测量
- 下一个无线电事务非常接近
- 设备处于背靠背通信状态

如前多次所述，定时器模块能够自主启动内部虚拟定时器，以执行校准过程结束/或时基维护。如果在应用程序级别请求无定时器的低功耗模式，且未预定无线电定时器和编程虚拟定时器，则定时器模块也会禁用内部虚拟定时器。在这种情况下，如果配置正确，设备只能通过外部源进行唤醒。

版本历史

表 1. 文档版本历史

日期	版本	变更
2020 年 7 月 13 日	1	初始版本。
2022 年 4 月 6 日	2	更新了“简介”和“第 1 节 模块概述”。 在整个文档中增加了 BlueNRG-LPS 参考文件。

目录