BlueNRG-LP、BlueNRG-LPS 2.4 GHz 无线电私有驱动程序

引言

本文档介绍 BlueNRG-LP、BlueNRG-LPS 2.4 GHz 无线电私有底层驱动程序，它提供对 BlueNRG-LP 和 BlueNRG-LPS 设备的访问权限，以便在不使用蓝牙链路层的情况下发送和接收数据包。使用中心数据结构和 API 的应用程序可以控制数据包的不同特征，例如：间隔、通道频率、数据长度等。

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 BlueNRG-LP 和 BlueNRG-LPS 无线电操作

BlueNRG-LP 和 BlueNRG-LPS 2.4 GHz 无线电底层驱动接口控制 2.4 GHz 无线电。此外，它还与采用慢速 32kHz 时钟的唤醒定时器、RAM 存储器，以及处理器相互作用。

RAM 用于存储无线电设置、当前无线电状态、接收到的数据，以及待发送的数据。无线电底层驱动可以管理最多 8 种不同的无线电配置（也称为状态机）。

有几个功能是由无线电自主管理的，无需处理器干预：

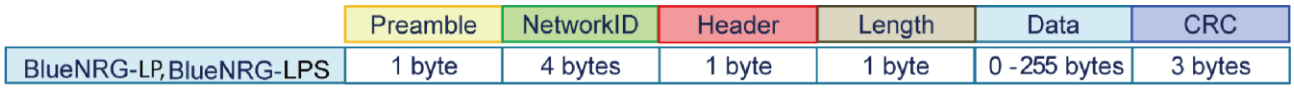
- 数据包加密
- 通信定时
- 睡眠管理

随附部分附加功能，与蓝牙低功耗标准（如蓝牙通道利用率）关联性较高。

2 数据包格式

BlueNRG-LP 和 BlueNRG-LPS 中只使用一种数据包格式，如下所示。

图 1. 数据包格式



一个数据包由六个字段组成，其中只有四个是用户可访问的：

- 默认情况下，前导码长度为 1 字节。但是，用户可以通过 RADIO_SetPreambleRep()定义前导码的重复次数。
- **NetworkID** 是设备的地址，以 4 个字节表示。接收设备只接受这样的数据包：其 NetworkID 字段与自身地址中的 NetworkID 字段相同。NetworkID 应该满足以下规则：
  - 其中连续的 0 或 1 不超过 6 个
  - 其 4 个八位字节不都相等
  - 转换不超过 24 个
  - 在最有效的 6 位中至少有 2 个转换

  用户可以通过 API RADIO_SetTxAttributes()或 API HAL_RADIO_SetNetworkID()访问 NetworkID 字段。
- **Header** 可以接受任何值，其长度为 1 字节。它可以作为一个字节的数据，但未对该字段进行加密。
- **Length** 表示数据字段的长度。用户为待发送的数据包设置该值，或者从接收到的数据包中读取该值。

  BlueNRG-LP、BlueNRG-LPS 链路层可以接收的最大有效负载字节数（带加密或不带加密）为 255。用户可以通过 API RADIO_SetMaxRecievedLength()在硬件层设置该阈值（从 0 到 255）。

  对于 BlueNRG-LP 和 BlueNRG-LPS，长度字段的最大值为 255，但有一些例外。如果启用了加密，则数据字段的最大长度必须减去 4 字节。这 4 字节是为添加到数据包中的 MIC 字段保留的，如图 2. 启用了加密时的数据包格式中所示。启用了加密时的数据包格式。下表包含关于长度字段的概要。

表 1. 长度字段的值（以字节为单位）

产品	数据通道	数据通道（带加密功能）	广播通道	广播通道（带加密功能）
BlueNRG-LP, BlueNRG-LPS	255	251	255	251

为了避免由于接收到的错误长度字段（在数据包中，CRC 校验失败）而导致内存损坏，用户必须为收到的数据包（包括 2 字节的 header 字段，以及 data 字段）保留最大内存

- **Data** 可以接受任何值，其长度由 length 字段决定。用户定义一个内存缓冲区，以便设置 heade 字段、length 字段，以及 data 据字段，如下所示：

```
PacketBuffer[0] = 0x01; // Header field
PacketBuffer[1] = 5; // Length field
PacketBuffer[2] = 0x02; // Data byte 1
PacketBuffer[3] = 0x03; // Data byte 2
PacketBuffer[4] = 0x04; // Data byte 3
PacketBuffer[5] = 0x05; // Data byte 4
PacketBuffer[6] = 0x06; // Data byte 5
```

如果启用了加密，则只对 data 字段进行加密。其他字段（包括 header 字段和 length 段）不加密。

- **CRC** 用于识别损坏的数据包。它的长度为 3 字节，在发送和接收时由无线电分别进行生成和校验。用户可以配置 CRC 计算的初始值，但在广播通道中除外（此处的初始值设置为 0x555555）。CRC 硬件特性可以禁用。这意味着硬件不会在传输时附加 CRC，也不会在接收时校验 CRC。

图 2. 启用了加密时的数据包格式

	Preamble	NetworkID	Header	Length	Data	MIC	CRC
BlueNRG-LP,BlueNRG-LPS	1 byte	4 bytes	1 byte	1 byte	0 -251 bytes	4 bytes	3 bytes

3 无线电底层驱动框架

3.1 说明

无线电底层驱动包含四个文件：

- rf_driver_ll_radio.h
- rf_driver_ll_radio.c
- rf_driver_hal_radio.h
- rf_driver_hal_radio.c

3.2 API 架构

无线电底层驱动接口提供了一组 API（文件 rf_driver_ll_radio.c），允许处理以下功能：

- 无线电初始化
- 加密
- 设置接收器和发射器 Phy（1 Mbps, 2 Mbps, S = 2, S = 8）
- 通信信道管理
- 设置网络 ID、CRC 初始值、发射功率等级
- 设置可接收数据包的最大长度和接收超时时间
- 测试指令（定频发送 tone）

管理这些设置的 API 列表如下：

- RADIO_Init()
- RADIO_SetEncryptionCount()
- RADIO_SetEncryptionAttributes()
- RADIO_SetEncryptFlags()
- RADIO_EncryptPlainData()
- RADIO_Set_ChannelMap()
- RADIO_SetChannel()
- RADIO_SetTxAttributes()
- RADIO_SetBackToBackTime()
- RADIO_SetTxPower()
- RADIO_SetReservedArea()
- RADIO_MakeActionPacketPending()
- RADIO_SetPhy()
- RADIO_SetMaxRecievedLength()
- RADIO_SetPreambleRep()
- RADIO_SetDefaultPreambleLen()
- RADIO_DisableCRC()
- RADIO_StopActivity()
- RADIO_StartTone()
- RADIO_StopTone()

大多数 API 修改状态机的参数（作为参数传递）。另一方面，部分参数呈全局性，即对所有状态机均有效。其中之一是调用 RADIO_SetGlobalReceiveTimeout()设置的接收超时。此值设置接收窗口的持续时间（以微秒为单位）。

无线电底层驱动使用由 ActionPackets 链表组成的中心数据结构。ActionPacket 与上述 API 共同定义完整的传输或接收操作。其中还包括一些回调函数，允许用户定义一系列操作。

ActionPacket 由输入字段和输出字段组成。输入字段用于配置动作，而输出字段则用于保存已经执行动作的信息。字段信息详见下表。

表 2. ActionPacket 结构

参数名称	输入/输出	总结
StateMachineNo	IN	该参数表示该操作的状态机编号。0 到 7
ActionTag	IN	当前动作的配置。 ActionTag 表中的标志位详细信息
MaxReceiveLength	IN	设置链路控制器可以接收的最大字节数。该值介于 0 至 255 字节之间
WakeupTime	IN	如果是相对时间，则包含以微秒为单位的唤醒时间。如果是绝对时间，则以系统时间单位（STU）表示。更多关于 STU 的信息，详见 BlueNRG-LP 和 BlueNRG-LPS 定时器模块应用笔记
*next_true	IN	如果 condRoutine() 返回 TRUE，则指向下一个 ActionPacket
*next_false	IN	如果 condRoutine() 返回 FALSE，则指向下一个 ActionPacket
(*condRoutine) (ActionPacket*)	IN	在 ActionPackets 链表中决定下一个动作所必需的用户回调。这个 routine 对时间要求紧迫，必须在 45us 内结束。
(*dataRoutine) (ActionPacket*, ActionPacket*)	IN	用于管理数据的用户回调
*data	IN/OUT	对于 TX，指向的数组包含待发送的数据（header、length 和 data 字段)。 对于 RX，指向的数组复制接收到的数据。如果是 RX，则数组必须有“第 2 节 数据包格式”中说明的最大尺寸
timestamp_receive	OUT	该字段包含数据包被接收时的时间戳。它将在 dataRoutine()回调函数中使用。仅限 RX。 它以 STU 的形式表示。一个 STU 等于 625/256 微秒。
status	OUT	状态寄存器包含与动作有关的信息。
rssi	OUT	数据包被接收时的 RSSI。仅限 RX。

ActionTag 是一个位掩码，用于启用无线电的不同特性（由 ActionPacket 使用）。下表介绍这些参数。

表 3. ActionTag 字段描述

位	名称	说明
7	TIMESTAMP_POSITION	该位设置时间戳的位置是在数据包的开头或者结尾。 0：数据包的结尾 - 当解调器接收到数据包的最后一位或当最后一个位传输位已从传输块中移出时。 1：数据包的开头 - 当解调器检测到前导码 + 存取地址时。仅适用于接收
6	INC_CHAN	该位激活自动信道增量。API RADIO_SetChannel()(1)设置增量的值。 0：无增量 1：自动增量
5	RELATIVE	该位决定 ActionPacket 的 WakeupTime 字段是绝对时间还是当前时间的相对时间。 0：绝对 1：相对
4	WHITENING_DISABLE	该位决定是否禁用白化 0：启用白化 1：禁用白化
3	RESERVED	RESERVED
2	TIMER_WAKEUP	在 Radio 句柄中，该位决定动作（RX 或 TX）的执行是基于连续操作（BackToBack）时间执行还是基于 WakeupTime。 如果是第一个动作，则忽略此位，因为它在执行时将始终基于 WakeupTime。 0：基于连续操作（BackToBack）时间（默认 150 µs）。 1：基于 WakeupTime
1	TXRX	该位决定动作是 RX 动作，还是 TX 动作。 1：TX 动作 0：RX 动作
0	PLL_TRIG	该位激活射频 PLL 校准。 0：禁用射频校准。 1：启用射频校准。 用户应该在第一个动作中对其进行设置

1. *在广播通道中，跳频被限制为 1 跳。*

ActionPacket 状态字段的位表示由最后一个无线电操作触发的中断的映射。ActionPacket 的状态字段的描述如下。可参照“BlueNRG-LP 无线电控制器”参考手册（RM0480）获取全部详情。

表 4. **Status_table**

位名	位位置	说明
RCVOK	31	接收数据无错误。
RCVCRCERR	30	接收数据失败（CRC 错误）。 仅当至少检测到前导码和存取地址时，才会引发此错误。
TIMECAPTURETRIG	29	在时间捕获寄存器中捕获的时间。
RCVCMD	28	接收的指令。
RCVNOMD	27	PDU 数据包报头中嵌入的接收到的 MD 位为零。
RCVTIMEOUT	26	接收超时（未发现前导码）。
DONE	25	接收/发送已完成。
TXOK	24	之前发送的数据包已经被对端设备成功接收。
RCVLENGTHERROR	18	接收的有效负载长度超过允许的最大值。
PREVTRANSMIT	6	前一个事件是发送(1)或接收(0)。

4 如何编写应用程序

编写应用程序有两种方式：前者基于主要由四个 API 组成的 HAL 层，后者基于使用 ActionPacket 数据结构。

4.1 HAL 层方法

最简单的方法是使用 HAL 无线电驱动（文件 rf_driver_hal_radio.c）中提供的一组 API，对无线电进行配置以完成以下操作：

- 发送一个数据包
- 发送一个数据包，然后等待数据包被接收（ACK）
- 等待一个数据包
- 等待一个数据包。如果收到数据包，则回应一个数据包（ACK）

在这种情况下，用户不需要使用 ActionPacket 来配置无线电的操作，但需要一个指向用户回调的指针，以根据执行的操作来处理不同的信息：

- TX 动作：IRQ 状态
- RX 动作：IRQ 状态、RSSI、时间戳，以及接收的数据

在中断模式下调用用户回调，特别是在具有最高优先级的 BLE_TX_RX_IRQHandler()中。

每个 API 的第二个参数是相对时间（以微秒为单位），表示从调用 API 的那一刻起下一个无线电活动的开始时间。该延迟必须足够长，否则将无法对无线电计时器进行编程，同时会返回错误代码。

用户可以选择所需的时间，无需考虑无线电为其设置使用的时间。之后，传递给 API 的延迟表示第一个比特被传输或接收窗口开始的时间。

图 3. 相对时间

4.2 带 HAL 层的 TX 示例

如下例所示，无线电被编程为周期性发送数据包，且两个连续数据包之间的时间为 10 ms。每个数据包含 3 个字节的数据。

注意： *无线电初始化之后是定时器模块初始化。*

该示例并未直接使用 ActionPacket 结构，而是通过 HAL 层的 API 对其进行使用。

```
uint8_t send_packet_flag = 1;
uint8_t packet[5];
int main(void)
{
    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);
    }
    /* Radio configuration*/
    RADIO_Init();
    /* Timer Init */
    HAL_VTIMER_Init(&VTIMER_InitStruct);
    /* Set the Network ID */
    HAL_RADIO_SetNetworkID(0x88DF88DF);
    /* Set the RF output power at max level */
    RADIO_SetTxPower(MAX_OUTPUT_RF_POWER);
    packet[0] = 1; /* Header field */
    packet[1] = 3; /* Length field */
    packet[2] = 2; /* Data */
    packet[3] = 3;
    packet[4] = 4;
    while(1) {
        HAL_VTIMER_Tick();
        /* If ready, a new action is scheduled */
        if(send_packet_flag == 1) {
            send_packet_flag = 0;
            /* Schedule the action with the parameter - channel, wakeupTime, data,
            dataCallback */
            HAL_RADIO_SendPacket(22, 10000, packet, TxCallback );
        }
    }
    return 0;
}
void BLE_TX_RX_IRQHandler(void)
{
    RADIO_IRQHandler();
}
```

定义用户回调 TxCallback()是为了重新调度另一个发送数据包动作，如下所示:

```
uint8_t TxCallback(ActionPacket* p, ActionPacket* next)
{
    /* Check if the TX action is ended */
    if( p ->status & BLUE_STATUSREG_PREVTRANSMIT) != 0) {
    /* Triggers the next transmission */
    send_packet_flag = 1;
    }
    return TRUE;
}
```

4.3 带 HAL 层的 RX 示例

在下面的示例中，无线电被编程为定期进入 RX 状态。每个 RX 操作之间的延迟为 9 ms。由此可确保在第一次良好接收后，RX 设备总是在 TX 设备（在“第 4.2 节 带 HAL 层的 TX 示例”中配置）开始发送数据包之前 1 ms（保护时间）被唤醒。

注意： *有效负载的最大长度设为 255 字节。*

RX 超时设为 20 ms。该值足够大，可以确保至少接收一个数据包。

该示例并未直接使用 ActionPacket 结构，而是通过 HAL 层的 API 对其进行使用。

```
uint8_t rx_flag = 1;
uint8_t packet[MAX_PACKET_LENGTH];

int main(void)
{
    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);
    }
    /* Radio configuration*/
    RADIO_Init();
    /* Timer Init */
    HAL_VTIMER_Init(&VTIMER_InitStruct);
    /* Set the Network ID */
    HAL_RADIO_SetNetworkID(0x88DF88DF);

    while(1) {
        HAL_VTIMER_Tick();
        /* If ready, a new action is scheduled */
        if(rx_flag == 1) {
          rx_flag = 0;

          /* Schedule the action with the parameter – channel, wakeupTime,
          receive buffer, receive timeout, max received length, dataCallback */
          HAL_RADIO_ReceivePacket ( 22, 9000, packet, 20000, 255, RxCallback );
        }
    }
    return 0;
}
void BLE_TX_RX_IRQHandler(void)
{
    RADIO_IRQHandler();
}
```

如果数据包已经收到，则定义用户回调 RxCallback()是为了重新安排另一轮接收并检索信息：

```
uint8_t RxCallback(ActionPacket* p, ActionPacket* next)
{
  /* Check if the RX action is ended */
  if( (p->status & BLUE_STATUSREG_PREVTRANSMIT) == 0) {
    /* Check if a packet with a valid CRC is received */
    if((p->status & BLUE_INTERRUPT1REG_RCVOK) != 0) {
      /* Retrieve the information from the packet */
      // p->data contains the data received: header field | length field | data field
      // p->rssi
      // p->timestamp_receive
    }
    /* Check if a RX timeout occurred */
    else if( (p->status & BLUE_INTERRUPT1REG_RCVTIMEOUT) != 0) {
    }
    /* Check if a CRC error occurred */
    else if ((p->status & BLUE_INTERRUPT1REG_RCVCRCERR) != 0) {
    }
  }
  /* Triggers the next reception */
  rx_flag = 1;
  return TRUE;
}
```

4.4 ActionPacket 方法

最灵活的方法是根据无线电需要执行的动作来声明一定数量的 ActionPacket。然后，用户以待执行操作的描述来填充这些结构。对于每个 ActionPacket，必须调用 API RADIO_SetReservedArea() 来初始化 ActionPacket 本身的信息。

如要开始执行 ActionPacket，则必须调用 API RADIO_MakeActionPacketPending()。在此之后，应用程序可以：

- 确保另一个 ActionPacket 被调用（通过将 ActionPacket 链接在一起），然后决定哪个 ActionPacket 在条件例程中执行。
- 通过再次调用 API RADIO_MakeActionPacketPending()来重新激活无线电执行。

所有进一步的操作都在中断模式下处理（就像 HAL 层方法那样），但在这种情况下，用户将处理两个回调函数：

- Condition routine：condRoutine()

  它提供当前 ActionPacket 的结果，并返回 TRUE 或 FALSE。根据这一点，链接到当前 ActionPacket 的下一个 ActionPacket 是从两种可能性中选择的：

  – next_true ActionPacket1
  – next_false ActionPacket2

  该机制的目的是区分调度程序的下一个动作。例如，RX 动作中的条件例程可以决定：

  – 调度 next_true ActionPacket，前提是收到的数据包良好（ActionPacket1）
  – 调度 next_false ActionPacket，前提是收到的数据包不符合要求（ActionPacket2）

- Data routine：dataRoutine()

  它提供接收或传输的数据、RSSI、时间戳等信息。此外，它应该根据最后接收的数据，修改下一个数据包的发送数据。该步骤也可用来修改待发送的数据包数据。

多个回调的目标是通过避免时效性瓶颈，使用户能够访问无线电的总体性能。其目标是在条件例程中对时效性方面进行绑定，而框架的其余部分并不注重时效性。

此举的优势在于，可令用户将代码拆分为更小的例程，从而改善编程的结构。

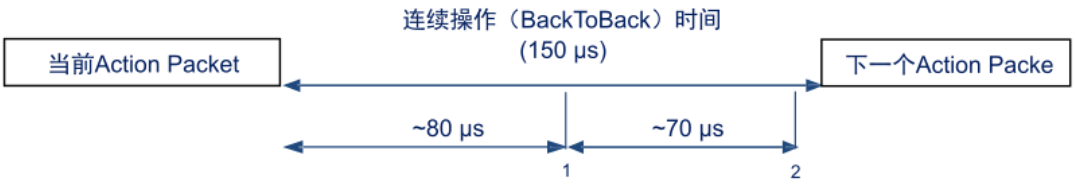
时效性要求最高的操作是在 conroutine()中执行的。此处会根据一些标准选择下一个活动，之后再动态地添加补丁。很明显，这必须在无线电控制器开始其操作之前发生，否则补丁将被忽略。

条件例程的最大执行时间大约是连续操作（BackToBack）时间减去无线电用于 RF 设置的时间。目前，为便于设置，连续操作（BackToBack）场景中的无线电需要达到 70µs。如果连续操作（BackToBack）时间为 150µs，则 condRoutine()有大约 80µs 的时间将指针设置到下一个活动。

因此，dataRoutine()将剩余时间用来修改（例如）待发送的数据。

下图总结了不同回调的时机安排。

图 4. 回调时机



1：在此之前，条件例程必须结束。

2：在此之前，必须完成下一个数据包的更新。

4.5 带 ActionPacket 的 TX 示例

如下例所示，无线电被编程为周期性发送数据包，且两个连续数据包之间的时间为 10 ms。每个数据包含 3 个字节的数据。ActionPacket 结构用于定义该操作。

```
int8_t packet[5];
ActionPacket txAction;

int main(void)
{
   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);
   }
   /* Radio configuration*/
   RADIO_Init();
   /* Timer Init */
   HAL_VTIMER_Init(&VTIMER_InitStruct);
   /* Set the channel (22) and the channel increment (0) */
   RADIO_SetChannel(STATE_MACHINE_0, 22, 0);
   /* Sets of the Network ID and the CRC initial value */
   RADIO_SetTxAttributes(STATE_MACHINE_0,0x88DF88DF, 0x555555);
   /* Set the RF output power at max level */
   RADIO_SetTxPower(MAX_OUTPUT_RF_POWER);
   packet[0] = 1; /* Header field */
   packet[1] = 3; /* Length field */
   packet[2] = 2; /* Data */
   packet[3] = 3;
   packet[4] = 4;
   /* Builds ActionPacket (txAction) for sending a
   single packet and schedules
   the next ActionPacket as itself (txAction) */
   txAction.StateMachineNo = STATE_MACHINE_0;
   /* Make a TX action with time relative to Wakeup
   Timer and enable the PLL calibration */
   txAction.ActionTag = RELATIVE | TIMER_WAKEUP | TXRX |
   PLL_TRIG;
   /* 10 ms before operation */
   txAction.WakeupTime = 10000;
   /* Pointer to the data to send */
   txAction.data = packet;
   /* Pointer to next ActionPacket: txAction */
   txAction.next_true = &txAction;
   /* Do nothing */
   txAction.next_false = NULL_0;
   /* Condition routine for selecting next ActionPacket*/
   txAction.condRoutine = conditionRoutine;
   /* Data routine called after conditionRoutine */
   txAction.dataRoutine = dataRoutine;
   /* Records the ActionPacket information */
   RADIO_SetReservedArea(&txAction);
   /* Execute the ActionPacket */
   RADIO_MakeActionPacketPending(&txAction);
   while(1) {
        HAL_VTIMER_Tick();
   }
   return 0;
}

void BLE_TX_RX_IRQHandler(void)
{
   RADIO_IRQHandler();
}
```

条件回调触发执行下一个预定的 ActionPacket (rxAction)。尽管在这种情况下不使用数据回调，但仍须进行定义。

```
uint8_t conditionRoutine(ActionPacket* p)
{
   /* Check if the TX action is ended */
   if( p ->status & BLUE_STATUSREG_PREVTRANSMIT) != 0) {
   }
   /* The TRUE schedules the next_true action: txAction */
   return TRUE;
}

uint8_t dataRoutine(ActionPacket* p, ActionPacket* next)
{
   return TRUE;
}
```

4.6 带 ActionPacket 的 RX 示例

在下面的示例中，无线电被编程为定期进入 RX 状态。每个 RX 操作之间的延迟为 9 ms。由此可确保在第一次良好接收后，RX 设备总是在 TX 设备（在“第 4.5 节 带 ActionPacket 的 TX 示例”中配置）开始发送数据包之前 1 ms（保护时间）被唤醒。RX 超时设为 20 ms。该值足够大，可以确保至少接收一个数据包。

ActionPacket 结构用于定义该操作。

```
ActionPacket rxAction;

int main(void)
{
      HAL_VTIMER_InitType VTIMER_InitStruct =
      {HS_STARTUP_TIME, INITIAL_CALIBRATION,
      CALIBRATION_INTERVAL};
      if (SystemInit(SYSCLK_64M,BLE_SYSCLK_32M)!= SUCCESS) {
           /* Error during system clock configuration
           take appropriate action */
           while(1);
      }

      /* Radio configuration*/
      RADIO_Init();
      /* Timer Init */
      HAL_VTIMER_Init(&VTIMER_InitStruct);
      /* Set the channel (22) and the channel increment (0) */
      RADIO_SetChannel(STATE_MACHINE_0, 22, 0);
      /* Sets of the Network ID and the CRC initial value */
      /* RX timeout 20 ms*/
      RADIO_SetGlobalReceiveTimeout(20000);
      rxAction. MaxReceiveLength = 255;
      RADIO_SetTxAttributes(STATE_MACHINE_0, 0x88DF88DF, 0x555555);
      /* Builds ActionPacket (rxAction) to make a reception and schedules
      the next ActionPacket at itself (rxAction) */
      rxAction.StateMachineNo = STATE_MACHINE_0;
      /* Make a RX action with time relative to Wakeup Timer and enable the PLL calibration */
      rxAction.ActionTag = RELATIVE | TIMER_WAKEUP | PLL_TRIG;
      /* 9 ms before operation */
      rxAction.WakeupTime = 9000;
      /* Pointer to the array where the data are received */
      rxAction.data = packet;
      /* Pointer to next ActionPacket: rxAction */
      rxAction.next_true = &rxAction;
      /* Do nothing */
      rxAction.next_false = NULL_0;
      /* Condition routine for selecting next ActionPacket*/
      rxAction.condRoutine = conditionRoutine;
      /* Data routine called after conditionRoutine : RSSI, RX timestamps,
      data received or data modification before next transmission*/
      rxAction.dataRoutine = dataRoutine;
      /* Records the ActionPacket information */
      RADIO_SetReservedArea(&rxAction);
      /* Execute the ActionPacket */
      RADIO_MakeActionPacketPending(&rxAction);
      while(1) {
           HAL_VTIMER_Tick();
      }
      return 0;
}
void BLE_TX_RX_IRQHandler(void)
{
   RADIO_IRQHandler();
}
```

条件回调触发执行下一个预定的 ActionPacket (rxAction)。尽管在这种情况下不使用数据回调，但仍须进行定义。

```
uint8_t conditionRoutine(ActionPacket* p)
{
   /* Check if the RX action is ended */
   if( (p->status & BLUE_STATUSREG_PREVTRANSMIT) == 0) {
      /* Check if a packet with a valid CRC is received */
      if((p->status & BLUE_STATUSREG_RCVOK) != 0) {
      }
      /* Check if a RX timeout occurred */
      else if( (p->status & BLUE_INTERRUPT1REG_RCVTIMEOUT) != 0) {
      }
      /* Check if a CRC error occurred */
      else if ((p->status & BLUE_INTERRUPT1REG_RCVCRCERR) != 0) {
      }
   }
   /* Triggers the next reception */
   return TRUE;
}

uint8_t dataRoutine(ActionPacket* p, ActionPacket* next)
{
   /* Check if the RX action is ended */
   if( (p->status & BLUE_STATUSREG_PREVTRANSMIT) == 0) {
      /* Check if a packet with a valid CRC is received */
      if((p->status & BLUE_STATUSREG_RCVOK) != 0) {
         /* Retrieve the information from the packet */
         // p->data contains the data received: header field | length field | data field
         // p->rssi
         // p->timestamp_receive
      }
      /* Check if a RX timeout occurred */
      else if( (p->status & BLUE_INTERRUPT1REG_RCVTIMEOUT) != 0) {
      }
      /* Check if a CRC error occurred */
      else if ((p->status & BLUE_INTERRUPT1REG_RCVCRCERR) != 0) {
      }
   }
   return TRUE;
}
```

5 BlueNRG-LP 和 BlueNRG-LPS 私有无线（OTA）固件

本章节介绍基于无线电底层驱动的 BlueNRG-LP、BlueNRG-LPS 私有无线（OTA）固件升级，它提供对 BlueNRG-LP 和 BlueNRG-LPS 设备的访问权限，以便在不使用蓝牙链路层的情况下发送和接收数据包。

本节介绍两个角色：服务器和客户端。

前一个节点负责通过 OTA 方式向客户端节点发送二进制镜像。

后一个节点作为复位管理程序选择运行哪个应用程序：OTA 客户端应用程序与服务器节点通信，以获取二进制镜像并更新其 Flash 存储器；或者之前加载的应用程序（通过 OTA 或其他方式）。

5.1 OTA 服务器应用程序

OTA 服务器应用程序负责通过无线方式向客户端节点发送二进制镜像。采用 YMODEM 通信协议，通过 UART 端口获取图像。

服务器状态机通过使用两个状态机例程实现：

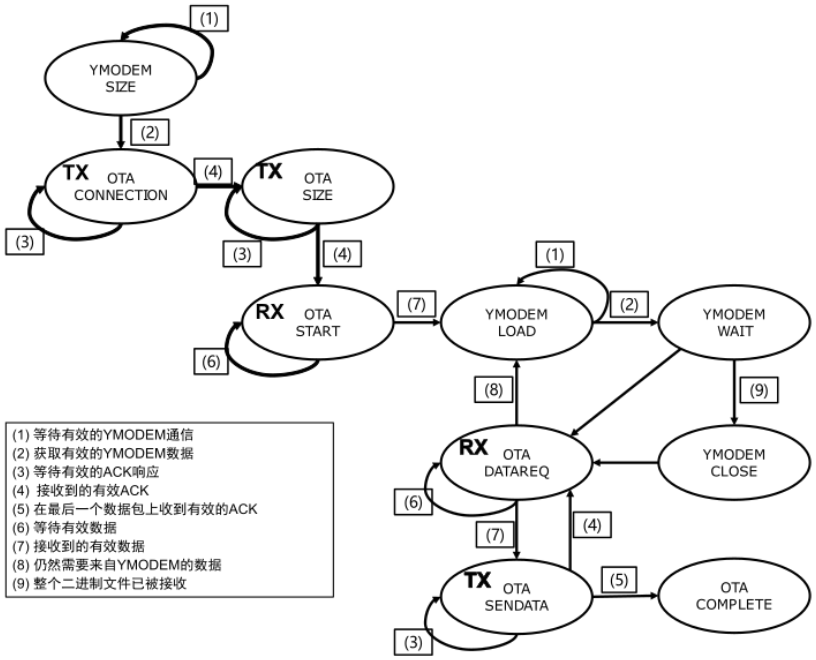
- 专用于 YMODEM 协议的状态机
- 专用于 OTA 协议的状态机

状态机具体由令牌决定。

5.1.1 OTA 服务器状态机

下图显示了为 OTA 服务器节点（包括 OTA 协议和 YMODEM 协议）而实现的状态机。

图 5. OTA 服务器状态图



有 4 种 YMODEM 状态：

- **SIZE**：第一次 YMODEM 通信，由用户提供二进制文件的大小。

- **LOAD**：主状态，可以接收最多 1kB 数据（二进制镜像）并将其存储在 RAM 内存中。
- **WAIT**：暂态，用于检查整个二进制镜像是否已被接收。
- **CLOSE**：该状态负责在接收到整个二进制镜像后关闭 YMODEM 通信。

通过 UART YMODEM 的任何通信问题都用应用程序的常规中止来处理。应用程序重新开始。

共有 6 种 OTA 状态：

- **CONNECTION**：一旦接收到二进制镜像文件的大小（YMODEM size 状态），就启动一个新的固件更新序列，服务器自此开始定期发送“连接数据包”，以显示连接的可行性。如果收到对“连接数据包”的 ACK 响应，则认为已接入客户端。
- **SIZE**：在此状态下，服务器向客户端发送一个“大小数据包”，说明可发送二进制镜像的大小。
- **START**：在此状态下，客户机要求服务器启动 OTA 固件更新。客户端根据在 SIZE 状态下接收到的二进制镜像的大小，可以发送“启动数据包”或“不启动数据包”。如果服务器接收到“启动数据包”，则 OTA 固件更新开始。否则，服务器将进入 CONNECTION 状态，寻找下一个连接。
- **DATAREQ**：服务器从客户端接收待发送的数据包数量（序列号）。
- **SENDATA**：服务器将请求的数据包（通过序列号）发送到客户端。所有数据包都具有相同的长度，但最后一个数据包的长度可以减少。如果仍然没有获取数据请求，服务器将进入 YMODEM LOAD 状态并获取二进制镜像的下一部分。
- **COMPLETE**：当客户端确认最后一个数据包，整个二进制镜像就传输完成，OTA 操作结束。

OTA 操作期间的射频通信通过重传进行管理，并预编程一定数量的重试。如果单个状态期间的重试次数超过了配置的最大重试次数，则 OTA 操作将被中止，应用程序将重新启动。

5.1.2 OTA 服务器数据包帧

所使用的数据包帧基于无线电底层驱动框架的数据包格式

图 6. 数据包帧格式

Preamble	NetworkID	Header	Length	Data	CRC
1 byte	4 bytes	1 byte	1 byte	variable	3 bytes

数据包的报头用于提供与服务器实际操作状态有关的信息，而数据包的数据提供诸如二进制镜像大小或二进制镜像数据块之类的信息。NetworkID 可由用户配置，且对服务器和客户端而言必须相同。

服务器发送的数据包列表如下：

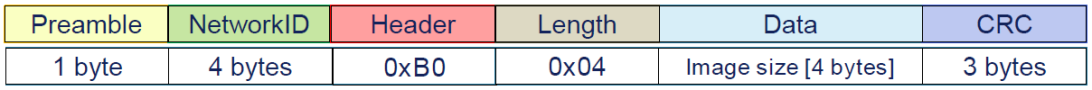
- **连接数据包**：不包含数据字段。报头设为 0xA0。

图 7. 连接数据包

Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xA0	0x00	3 bytes

- **大小数据包**：包含 4 字节的数据，其中包含有关 MSB first 中二进制镜像大小的信息。

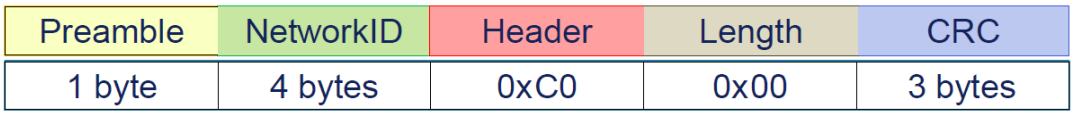
图 8. 大小数据包



Preamble	NetworkID	Header	Length	Data	CRC
1 byte	4 bytes	0xB0	0x04	Image size [4 bytes]	3 bytes

- **“Start ACK”数据包：**用来确认客户端启动数据包的响应数据包。

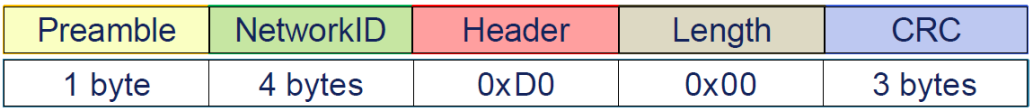
图 9. “Start ACK”数据包



Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xC0	0x00	3 bytes

- **“Data request ACK”数据包：**用来确认客户端数据请求数据包的响应数据包。

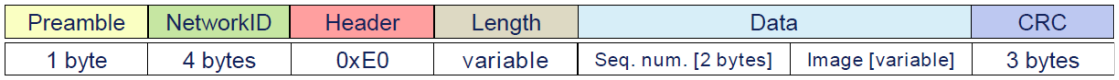
图 10. “Data request ACK”数据包



Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xD0	0x00	3 bytes

- **“Send data（发送数据）”数据包：**包含二进制镜像中的数据块的数据包。序列号用于同步客户端和服务器端，并用于重传机制。

图 11. “发送数据”数据包



Preamble	NetworkID	Header	Length	Data		CRC
1 byte	4 bytes	0xE0	variable	Seq. num. [2 bytes]	Image [variable]	3 bytes

5.2 OTA 客户端应用程序

OTA 客户端应用为复位管理器应用，在启动时决定，是“跳转”到用户应用程序，还是激活客户端 OTA 固件更新应用程序。

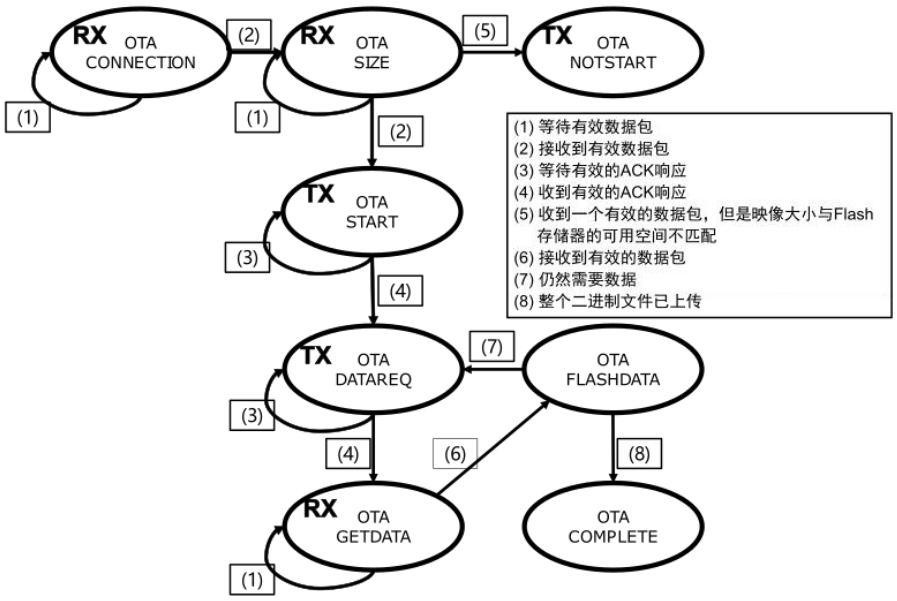
客户端 OTA 固件更新应用程序通过一个状态机实现，该状态机管理 OTA 协议并在用户 Flash 中加载获取的二进制镜像。

OTA 客户端应用程序内存大小低于 Flash 存储器的 8 kB。

5.2.1 OTA 客户端状态机

下图显示了为 OTA 客户端节点实现的 OTA 协议的状态机。

图 12. OTA 客户端状态图



共有 8 种 OTA 状态：

- **CONNECTION**：这是客户端的开始状态。它能够查找来自服务器的有效“连接数据包”。如果接收到“连接数据包”，则回发一个 ACK 响应。此动作建立与服务器的连接。
- **SIZE**：在此状态下，客户端从服务器获取“大小数据包”。
- **START**：在此状态下，客户端向服务器发送“开始数据包”。这会导致 OTA 固件更新启动。仅当二进制镜像的大小与用户 Flash 存储器匹配时，才会发送“开始数据包”。
- **NOTSTART**：二进制镜像的大小对于客户端的用户 Flash 存储器而言过大。因此，OTA 固件更新不会启动。
- **DATAREQ**：客户端向服务器发送所需数据包的数量。数量计算的依据是二进制镜像的大小和服务器可以发送的最大字节数（起初针对客户端和服务器所定义）
- **SENDATA**：客户端获得请求的数据包
- **FLASHDATA**：在此状态下，数据包中的数据存储在缓冲区中，一旦缓冲区的大小大于或等于页面大小，则所有缓冲区实际上都写入用户 Flash 存储器中。此操作也在接收到二进制镜像的最后一块后执行。
- **COMPLETE**：一旦整个二进制镜像上传到用户 Flash 存储器中，OTA 操作就完成了。

为了具有一定的重试次数，OTA 操作期间的 RF 通信通过重传进行管理。如果单个状态期间的重试次数超过了配置的最大重试次数，则 OTA 操作将被中止，应用程序将重新启动。

5.2.2 OTA 客户端数据包帧

所使用的数据包帧基于无线电底层驱动框架的数据包格式。

图 13. 数据包帧格式（客户端）

Preamble	NetworkID	Header	Length	Data	CRC
1 byte	4 bytes	1 byte	1 byte	variable	3 bytes

数据包的报头用于提供与客户端实际操作的状态有关的信息，而数据包的数据用于请求二进制镜像的特定块。NetworkID 可由用户配置，且对服务器和客户端而言必须相同。

客户端发送的数据包列表如下：

- **“Connection ACK”数据包：**是对服务器连接数据包的响应。

图 14. “Connection ACK”数据包

Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xA0	0x00	3 bytes

- **“Size ACK”数据包：**是对服务器大小数据包的响应。

图 15. “Size ACK”数据包

Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xB0	0x00	3 bytes

- **“Start”数据包：**用于启动 OTA 操作。

图 16. “Start”数据包

Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xC0	0x00	3 bytes

- **“NotStart”数据包：**用于不启动 OTA 操作。

图 17. “Not start”数据包

Preamble	NetworkID	Header	Length	CRC
1 byte	4 bytes	0xF0	0x00	3 bytes

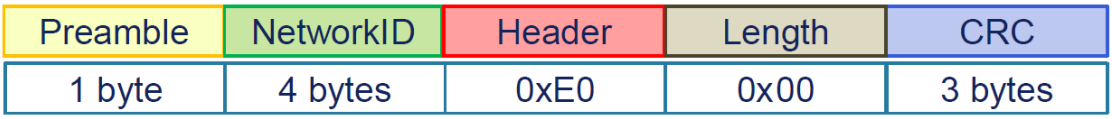
- **“Data request”数据包：**用于向服务器请求特定的数据包。

图 18. “Data request”数据包

Preamble	NetworkID	Header	Length	Data	CRC
1 byte	4 bytes	0xD0	0x02	Seq. num. [2 bytes]	3 bytes

- **“Send data ACK”数据包：**用于对来自服务器的数据进行确认。

图 19. **“Send data ACK”数据包**



5.3 OTA 固件升级场景

以下是 OTA 固件升级操作的场景。

该场景由两台设备组成。

- 服务器 - 运行应用程序 RADIO_OTA_ResetManager，配置 OTA_Server_Ymodem。
- 客户端- 运行应用程序 RADIO_OTA_ResetManager，配置 OTA_Client。

固件应用程序位于 DK 文件夹\Projects\Peripheral_Examples\MIX\RADIO\RADIO_OTA_ResetManager 中。

OTA 固件升级要遵循的步骤如下：

1. 启动 BlueNRG-LP、BlueNRG-LPS 板件及其各自的应用程序 OTA_Server_Ymodem 和 OTA_Client。

   OTA_Server_Ymodem 板必须用 USB 线连接到 PC 上。
2. 使用 TeraTerm 等串行终端程序打开板件（采用 OTA_Server_Ymodem 配置）的 COM 口。
3. 选择 YMODEM 标准文件的传输方式。

   在 Terterm 中，操作步骤为：打开菜单 File，然后选择 transfer，再选择 YMODEM 并按“send（发送）”。
4. 选择要上传的二进制镜像（.bin 文件）。要注意的是！必须按照“第 5.4 章节 如何添加 OTA 客户端功能”的说明生成二进制镜像。
5. YMODEM 传输完成后，客户端板件的 OTA 固件升级也就结束。

在发布的 DK 中，可以找到一个示例应用程序，其中也包含为 OTA 客户机功能保留的两个配置。示例应用程序 TxRx 通过无线电底层驱动，演示点对点通信。配置如下：

- TX_Use_OTA_ResetManager.
- RX_Use_OTA_ResetManager.

对于这两种配置，“第 5.4 章节 如何添加 OTA 客户端功能”中说明了相关步骤。STEVAL-IDB0011V1 的按钮 PUSH2 用于运行 OTA 客户端功能，以更新相应的镜像。

示例位于\Projects\Peripheral_Examples\MIX\RADIO\RADIO_TxRx 中。

5.4 如何添加 OTA 客户端功能

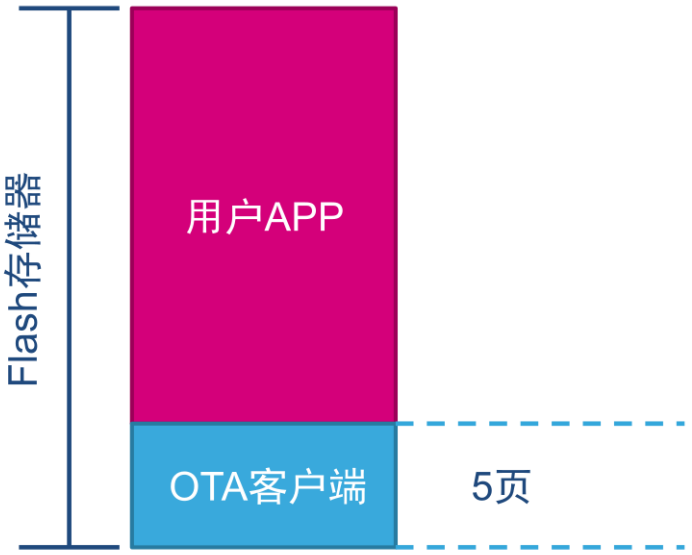
为了将 OTA 客户端功能集成到现有应用程序中，用户必须遵循以下步骤：

1. 为客户端应用程序保留 Flash 存储器的前 5 页（10 kB）。链接器符号“MEMORY_FLASH_APP_OFFSET”可以在该范围内使用如下：

   MEMORY_FLASH_APP_OFFSET =0x2800

   示例应用程序 TxRx、配置 TX_Use_OTA_ResetManager 或 RX_Use_OTA_ResetManager 可作为参考。
2. 在应用程序项目中包含文件“radio_ota.c”和“radio_ota.h”。这些文件位于\Library\BLE_Application\OTA。文件包含 API OTA_Jump_To_Reset_Manager()，用于设置 RAM 变量 ota_sw_activation，然后复位系统。
3. 定义一个触发器，用于从用户应用程序跳转到 OTA 客户端应用程序。该触发器用于调用函数 OTA_Jump_To_Reset_Manager()。

图 20. OTA 客户端 Flash 存储器布局



版本历史

表 5. 文档版本历史

日期	版本	变更
2020 年 7 月 20 日	1	初始版本。
2021 年 3 月 29 日	2	增加了表 4. Status_table。 更新了表 3. ActionTag 字段描述，“第 4.2 节 带 HAL 层的 TX 示例”和“第 4.3 节 带 HAL 层的 RX 示例”。
2022 年 4 月 6 日	3	更新了：简介、“第 3.1 节 描述”、“第 3.2 节 API 架构”、“第 4.1 节 HAL 层方法”，以及“第 5.3 节 OTA 固件升级场景”。 在整个文档中增加了 BlueNRG-LPS 参考文件。

目录

表格索引

图片目录

重要通知 - 请仔细阅读

意法半导体公司及其子公司（“意法半导体”）保留随时对 ST 产品和/或本文档进行变更、更正、增强、修改和改进的权利，恕不另行通知。买方在订货之前应获取关于意法半导体产品的最新信息。意法半导体产品的销售依照订单确认时的相关意法半导体销售条款。

买方自行负责对意法半导体产品的选择和使用，意法半导体概不承担与应用协助或买方产品设计相关的任何责任。

意法半导体不对任何知识产权进行任何明示或默示的授权或许可。

转售的意法半导体产品如有不同于此处提供的信息的规定，将导致意法半导体针对该产品授予的任何保证失效。

ST 和 ST 标志是意法半导体的商标。关于意法半导体商标的其他信息，请访问 www.st.com/trademarks。其他所有产品或服务名称是其各自所有者的财产。本文档中的信息取代本文档所有早期版本中提供的信息。