蓝牙的跳频算法

原创 闪光light 于 2021-10-08 16:52:57 发布 4347 收藏 29 版权

文章标签： 算法 蓝牙

目标：

阅读蓝牙协议，了解蓝牙的跳频算法，并使用 matlab 进行验证。

内容：

1. 了解蓝牙跳频算法，包括其与非连接态与连接态的信道选择。

2. 使用matlab对其进行验证

一.跳频算法简介

由于蓝牙工作的频段在2.4Ghz，是很多无线设备所使用的频段，极其容易受到干扰，因此，蓝牙采用跳频算法来减少其在通信时所受到的干扰。

蓝牙在2.4GHz ISM频段中定义了40个射频 信道。这些射频信道被分为三种：广播（advertising）、周期（periodic）和数据（data）信道。广播信道使用所有40个射频频道来发现设备、启动连接和广播数据。其中3个射频频道，称为“主要广播频道”，用于初始广播和所有传统广播活动，以及37个射频信道，称为“次要广播信道”，用于所涉及的大部分通信。数据信道使用多达37个射频信道在连接的设备之间进行通信。进行通信的两个设备使用相同的物理信道。为了实现这一点，他们的收发器必须同时调谐到相同的射频信道。当蓝牙进行通信时，设备间会不断变化所使用的物理信道，相连接的两个蓝牙设备必须依照相同的信道变化规律以使得两设备在相同的信道进行通信，这就是蓝牙的跳频算法。

蓝牙的信道划分由下表所示：

RF Channel	RF Center Frequency	Channel Index	Channel Type		
			Data	Primary Advertising	Secondary Advertising
0	2402 MHz	37		●	
1	2404 MHz	0	●		●
2	2406 MHz	1	●		●
…	…	…	…	…	…
11	2424 MHz	10	●		●
12	2426 MHz	38		●	
13	2428 MHz	11	●		●
14	2430 MHz	12	●		●
…	…	…	…	…	…
38	2478 MHz	36	●		●
39	2480 MHz	39		●	

二. 蓝牙跳频算法的实现

2.1 广播信道跳频

蓝牙的链路层状态分为以下五种状态：

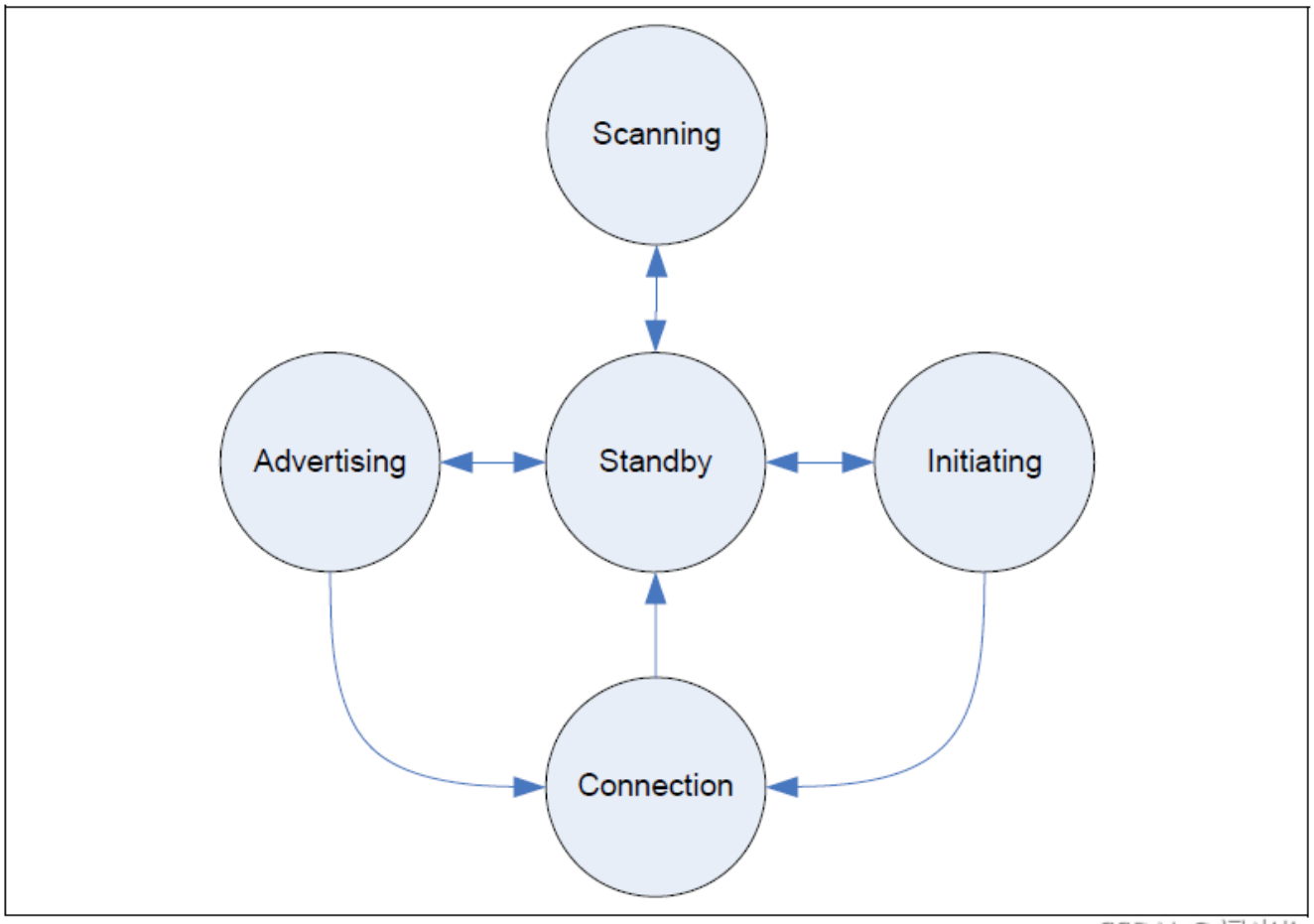


Figure 1.1: State diagram of the Link Layer state machine

其中，standby 不进行数据的发送，Initiating和Scanning没有对信道选择规则进行限制，因此，跳频态只在 Advertising 和Connection状态中进行。

当蓝牙位于广播状态时，每隔一个固定的时间进行一次广播事件，如下图所示：

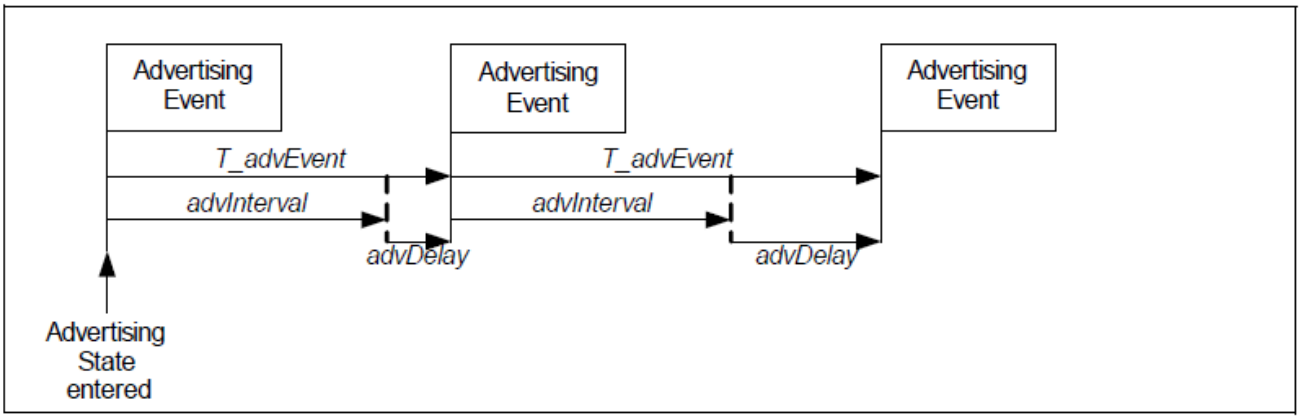
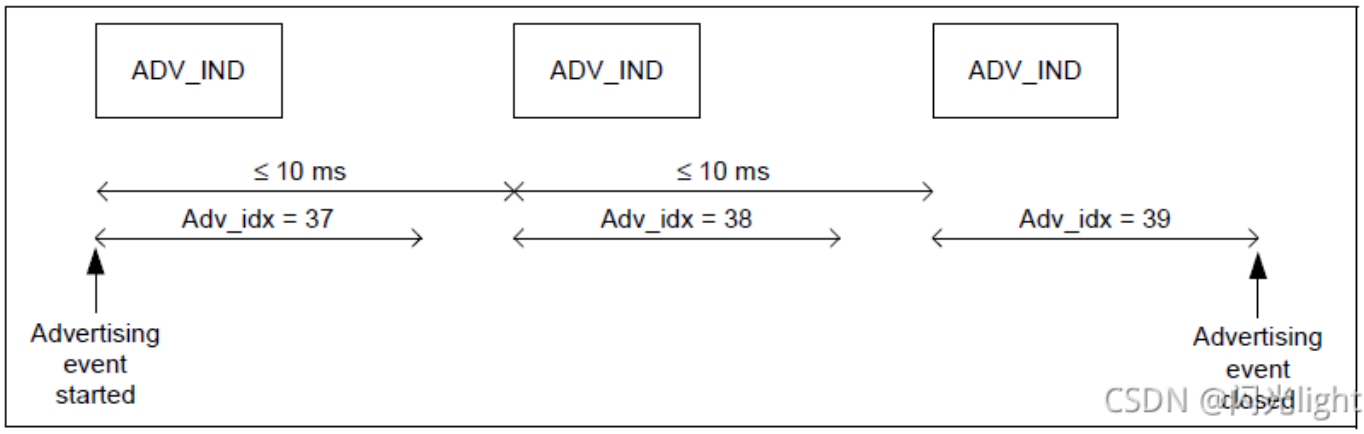


Figure 4.3: Advertising events perturbed in time using advDelay

advInterval应为20 ms至10485.759375 s范围内0.625 ms的整数倍。

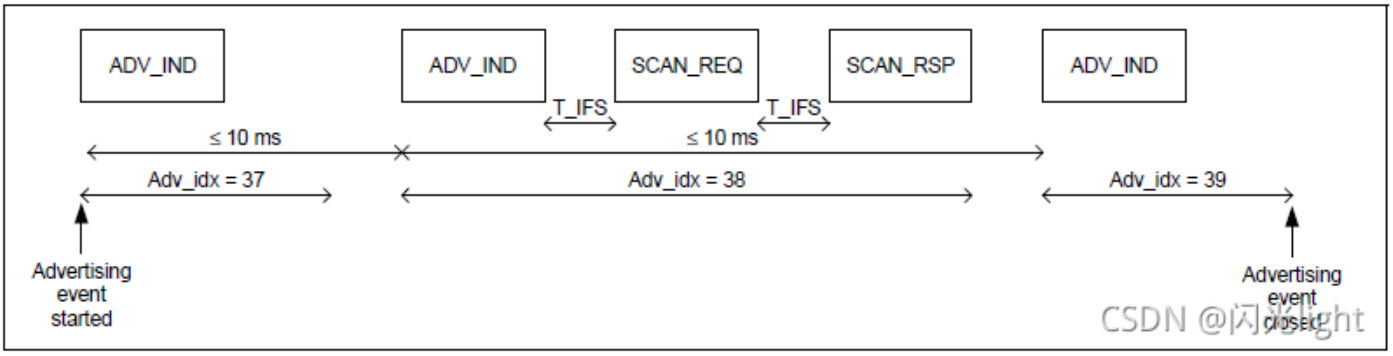
advDelay是一个伪随机值，其范围为0 ms到10 ms，由链路层为每个广播事件生成。

当没有连接请求时，每个广播事件的信道选择如下：

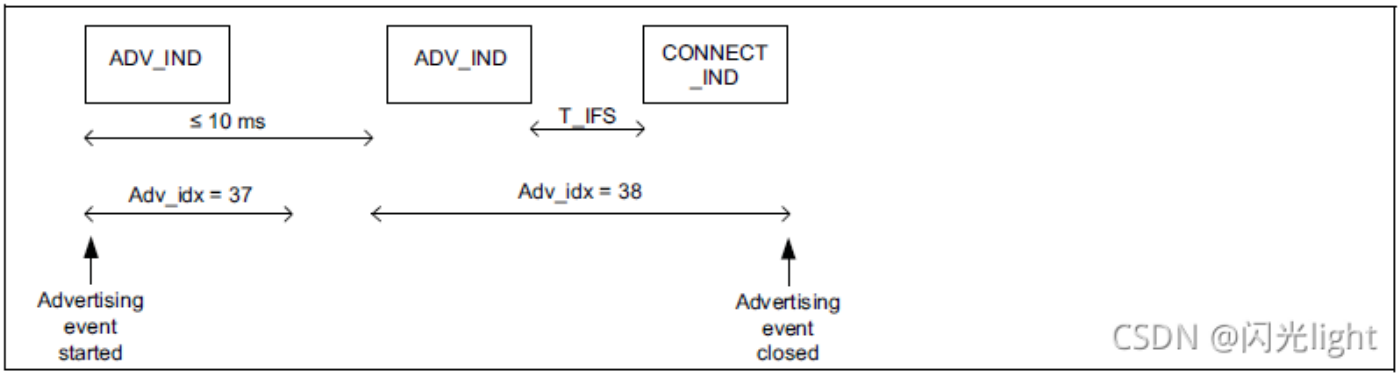


每次广播信道都是自37开始，至39结束，周而复始。

如果设备接收到包含其设备地址的SCAN_REQ PDU，则该设备应在同一广播信道上回复SCAN_RSP PDU。发送SCAN_RSP PDU后，或者设备不允许扫描，应移动到下一个使用的主要广播信道以发送另一个ADV IND PDU，或者关闭广播事件。



如果设备接收到包含其设备地址的CONNECT_IND PDU，则链路层应退出广播状态，并按照转换到连接状态。如果设备不进入连接状态，则应移动到下一个使用的广播信道以发送另一个ADV_IND PDU，或关闭广播事件。

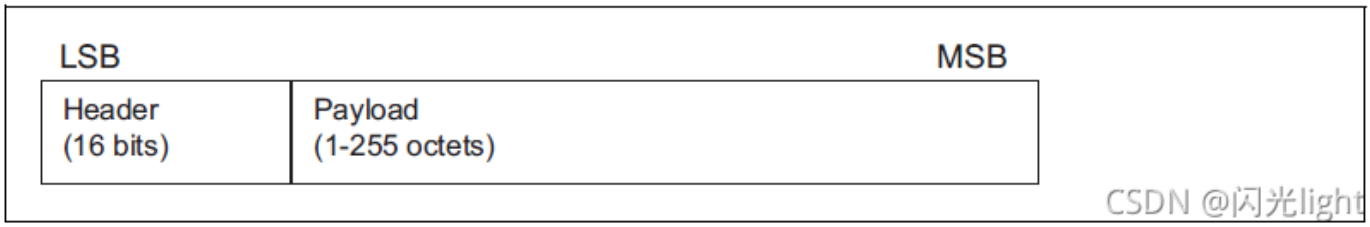


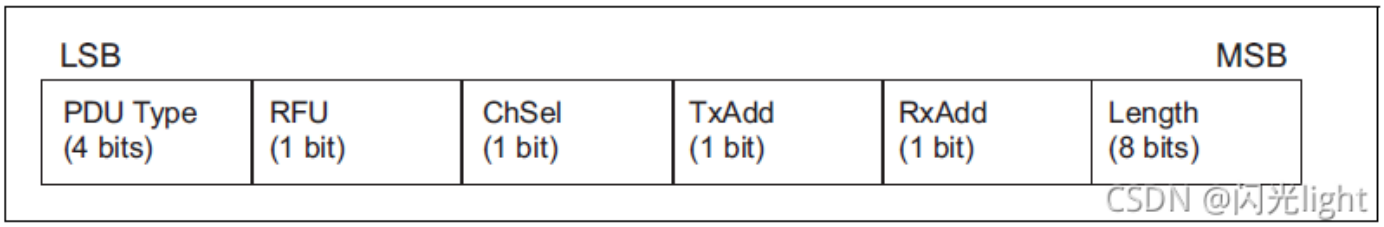
总而言之，在广播状态，信道的变化是在三个广播信道上按顺序进行轮流交替 。

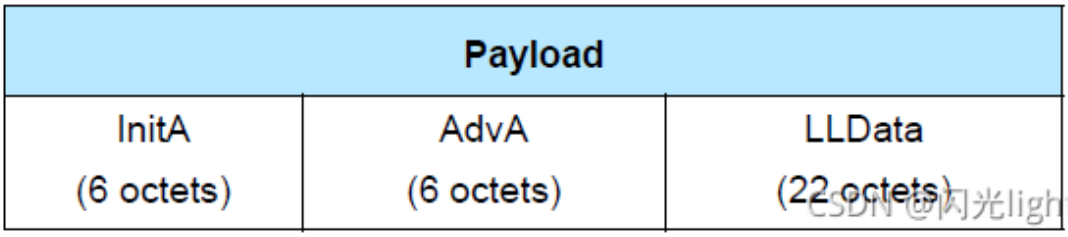
2.2 数据信道跳频

主设备链路层应将数据信道分为可用信道（用于连接）和不可用信道（不用于连接）。这称为通道图（channel map）。所用通道的最小数量应为2个。主设备可以向链路层提供信道分类信息。链路层可以使用主设备提供的信息。从设备应在CONNECT_IND

PDU中从主设备接收通道图。如果主设备更改通道图，应通知从设备。CONNECT_IND结构如下：







LLData									
AA	CRCInit	WinSize	WinOffset	Interval	Latency	Timeout	ChM	Hop	SCA
(4 octets)	(3 octets)	(1 octet)	(2 octets)	(2 octets)	(2 octets)	(2 octets)	(5 octets)	(5 bits)	(3 bits)

Figure 2.13: LLData field structure in CONNECT_IND and AUX_CONNECT_REQ PDU's payload

如上面的几幅图所示，主设备会发送带有通道图和跳频算法的选择的CONNECT_IND给从设备，从设备按照Header中的ChSel的值选择跳频算法（蓝牙包含两种跳频算法），按照LLData中的ChM值确定通道图，按照Hop中的值来确定跳频算法1的参数。下图就是一个使用wireshark软件截取的一个CONNECT_IND包的结构：

```
> Nordic BLE Sniffer
v Bluetooth Low Energy Link Layer
    Access Address: 0x8e89bed6
  v Packet Header: 0x22e5 (PDU Type: CONNECT_IND, ChSel: #2, TxAdd: Random, RxAdd: Random)
      .... 0101 = PDU Type: 0x5 CONNECT_IND
      ...0 .... = Reserved: 0
      ..1. .... = Channel Selection Algorithm: #2
      .1.. .... = Tx Address: Random
      1... .... = Rx Address: Random
      Length: 34
    Initiator Address: 47:a7:fb:f0:27:14 (47:a7:fb:f0:27:14)
    Advertising Address: f3:67:dd:f3:ac:a7 (f3:67:dd:f3:ac:a7)
  v Link Layer Data
      Access Address: 0xd24beac7
      CRC Init: 0xbcb51a
      Window Size: 2 (2.5 msec)
      Window Offset: 13 (16.25 msec)
      Interval: 36 (45 msec)
      Latency: 0
      Timeout: 500 (5000 msec)
    > Channel Map: ffffffff1f
      ...0 1011 = Hop: 11
      101. .... = Sleep Clock Accuracy: 31 ppm to 50 ppm (5)
    CRC: 0x83a124
```

```
0000  03 35 00 03 2c 57 02 0a  01 26 30 00 00 e2 e6 15   ·5··,W·· ·&0·····
0010  b8 d6 be 89 8e e5 22 14  27 f0 fb a7 47 a7 ac f3   ·····"· '···G···
0020  dd 67 f3 c7 ea 4b d2 1a  b5 bc 02 0d 00 24 00 00   ·g···K·· ·····$··
0030  00 f4 01 ff ff ff ff 1f  ab c1 85 24               ········ ···$
```


通道图为FFFFFFFF1F（1111 1111 1111 1111 1111 1111 1111 1111 0001 1111），表示在此次连接中，第5、6、7信道不可用。

2.2.1 跳频算法1

通道选择算法#1包括两个阶段：计算未映射的信道数，然后将该索引映射到所用通道图中。整体算法如下图所示：

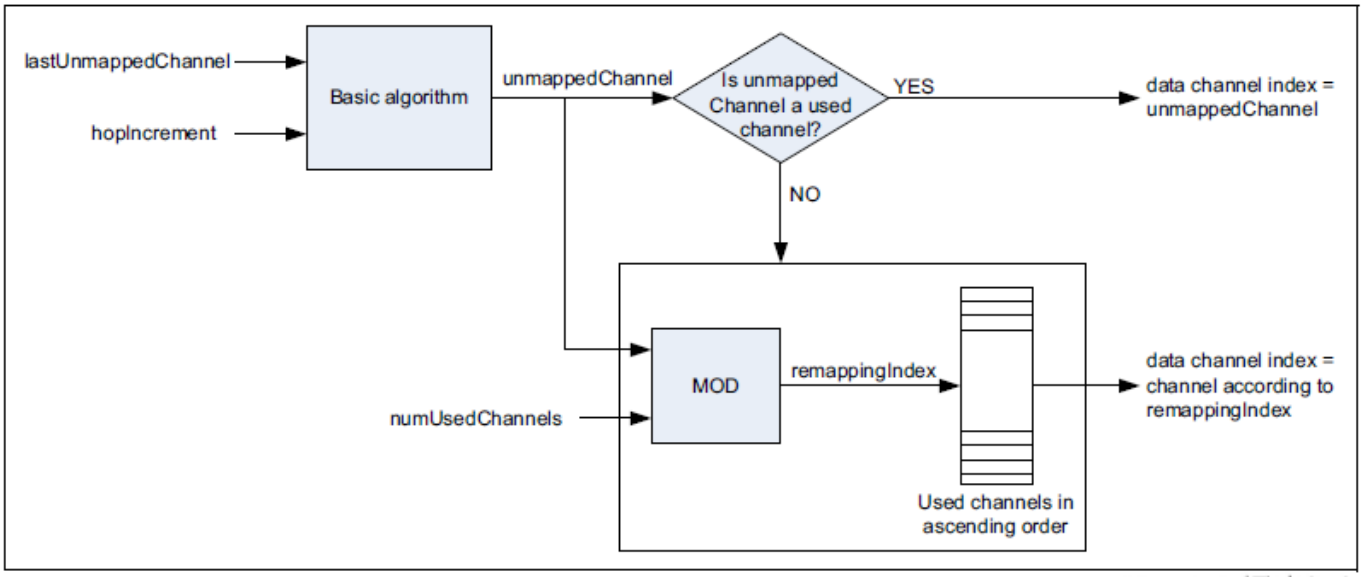


Figure 4.32: Block diagram of data Channel Selection Algorithm #1

unmappedChannel和lastUnmappedChannel是两个连续连接事件的未映射通道数。

unmappedChannel被称为当前连接事件的未映射通道索引。lastUnmappedChannel是上一个连接事件的未映射通道索引。*hopIncrement*处于5-16之间的一个随机值。对于连接的第一个连接事件，LastUnappedChannel应为0。在连接事件开始时，应使用以下基本算法计算未映射通道索引：

$$unmappedChannel = (lastUnmappedChannel + hopIncrement) \bmod 37$$

当连接事件关闭时，LastUnappedChannel应设置为UnappedChannel的值。如果根据信道映射，unmappedChannel位于通道图中，信道选择算法#1应unmappedChannel的值作为该次连接事件的数据信道。如果根据通道映射，unmappedChannel是不可用，则应使用以下算法将未映射通道重新映射到通道映射中使用的通道之一：

$$remappingIndex = unmappedChannel \bmod numUsedChannels$$

其中NumusedChannel是可使用通道数。将构建一个重新映射表，该表按升序包含所有使用的通道，从零开始索引。然后使用remappingIndex从重新映射表中选择连接事件的数据通道索引，重新映射表指的是可用信道从低到高排列。

举个例子

第一次连接事件

Channel map [36:0] =11110_00000000_11100000_00000110_00000000b；

LastUnappedChannel = 0；

Used channels in ascending order is [9, 10, 21, 22, 23, 33, 34, 35, 36]，N=9（可用信道数）

假设输入 hopIncrement = 7

unmappedChannel = (0+7)mod37 = 7

由于 7 不属于可用信道表中：

remappingIndex = 7mod N = 7

则 data channel index = Used channels in ascending order[7] = 35;

第二次连接事件

Channel map [36:0] =11110_00000000_11100000_00000110_00000000b；

LastUnappedChannel = 7；

Used channels in ascending order is [9, 10, 21, 22, 23, 33, 34, 35, 36].

hopIncrement = 7

unmappedChannel = (7+7)mod37 = 14

由于14 不属于可用信道表中：

remappingIndex = 14mod 9= 5

data channel index = Used channels in ascending order[5] = 33;

由此可见，与跳频算法1相关的是LastUnappedChannel ， Used channels in ascending order，hopIncrement三个参数，matlab实现方式如下：

```
function unmappedChannel = Basic_algorithm(lastUnmappedChannel,hopIncrement)
unmappedChannel = mod((lastUnmappedChannel + hopIncrement),37);
end
```

```
clear;
channel_map = [9 10 21 22 23 33 34 35 36];
lastUnmappedChannel = 0;
hopIncrement = 7;
unmappedChannel = Basic_algorithm(lastUnmappedChannel,hopIncrement);
if ismember(unmappedChannel,channel_map)
    channel_index = unmappedChannel;
else
    remappingIndex = mod(unmappedChannel,length(channel_map(:)));
    channel_index = channel_map(remappingIndex+1);
end
```

2.2.2 跳频算法2

跳频算法2在蓝牙5.0之后的版本使用更广，其算法流程如下图：

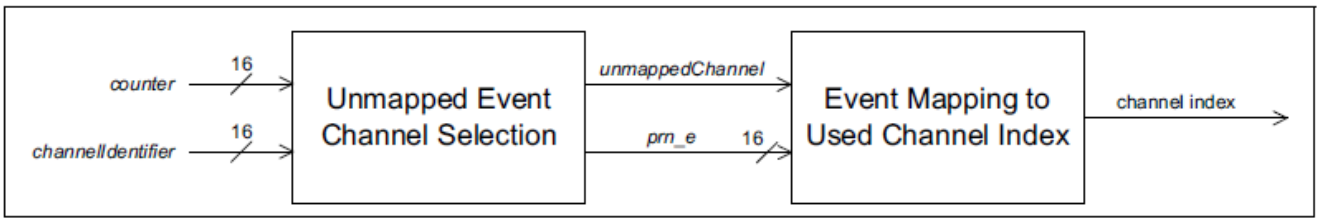


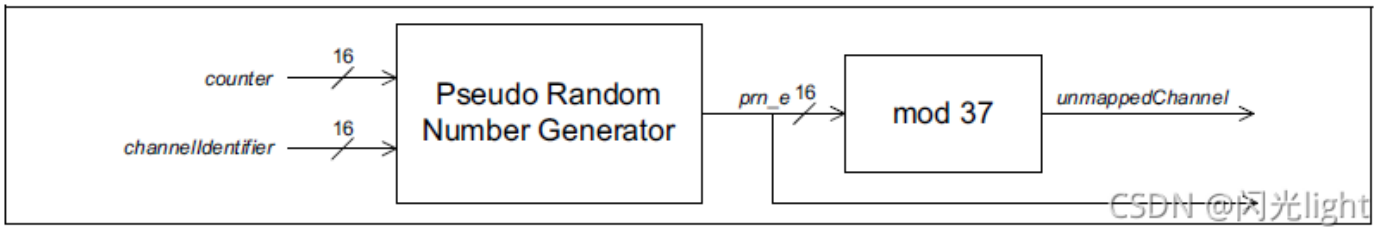
Figure 4.33: General block diagram of Channel Selection Algorithm #2

counter是事件数计数器，每经过一个连接事件计数器加一，范围为：0x0000-0xFFFF；

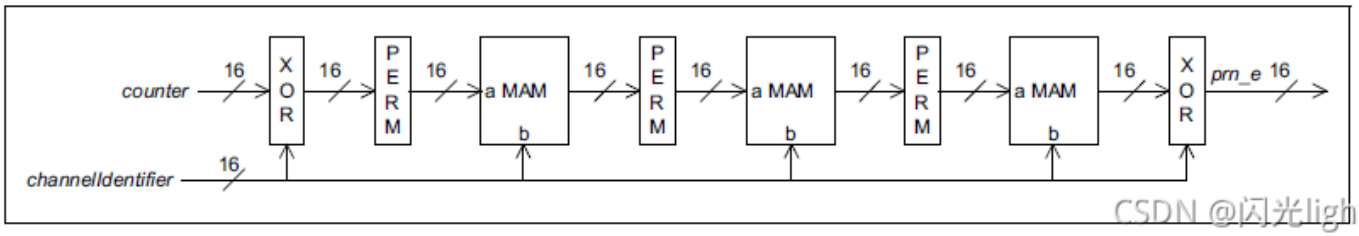
channelIdentifier的值由以下表达式得出：

channelIdentifier = (Access Address31-16) XOR (Access Address15-0)

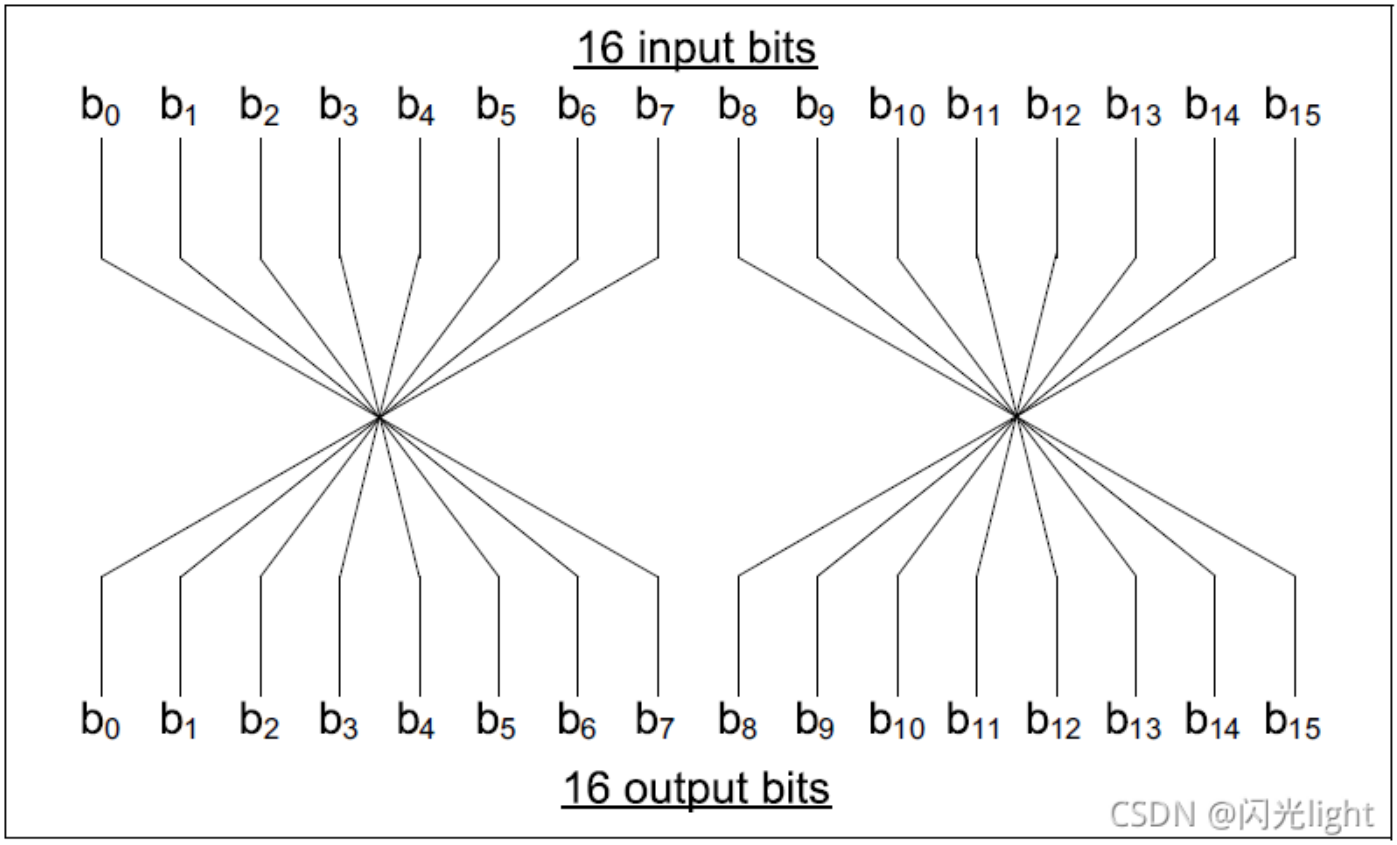
Unmapped Event Channel Selectio模块的计算过程如下:



Pseudo Random Number Generator 模块的结构如下



PERM是置位操作，置换操作包括分别对低8个输入位和高8个输入位进行位反转,结构如下:



matlab代码如下:

```
function y = PERM(in)
%UNTITLED3 此处显示有关此函数的摘要
%    此处显示详细说明
```

```
y(1) = in(8);    y(2) = in(7);
y(3) = in(6);
y(4) = in(5);
y(5) = in(4);
y(6) = in(3);
y(7) = in(2);
y(8) = in(1);
y(9) = in(16);
y(10) = in(15);
y(11) = in(14);
y(12) = in(13);
y(13) = in(12);
y(14) = in(11);
y(15) = in(10);
y(16) = in(9);
end
```

MAM是一个包含加、成、取余的操作，结构如下：

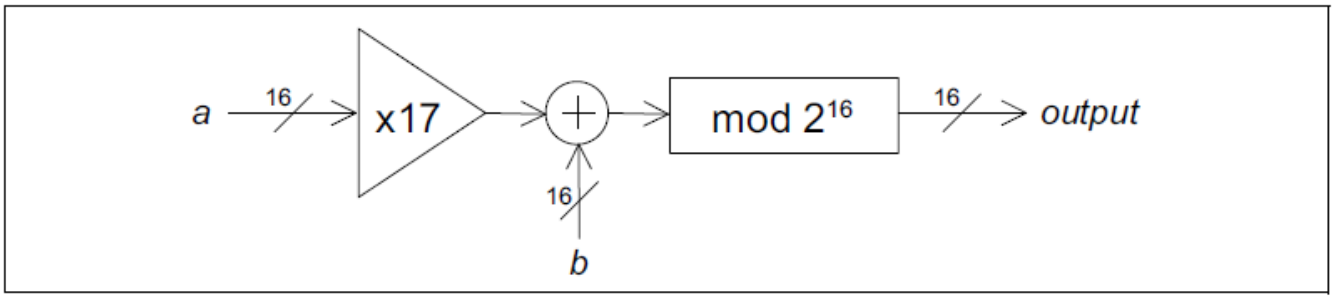


Figure 4.35: Multiply, Add, and Modulo block operation

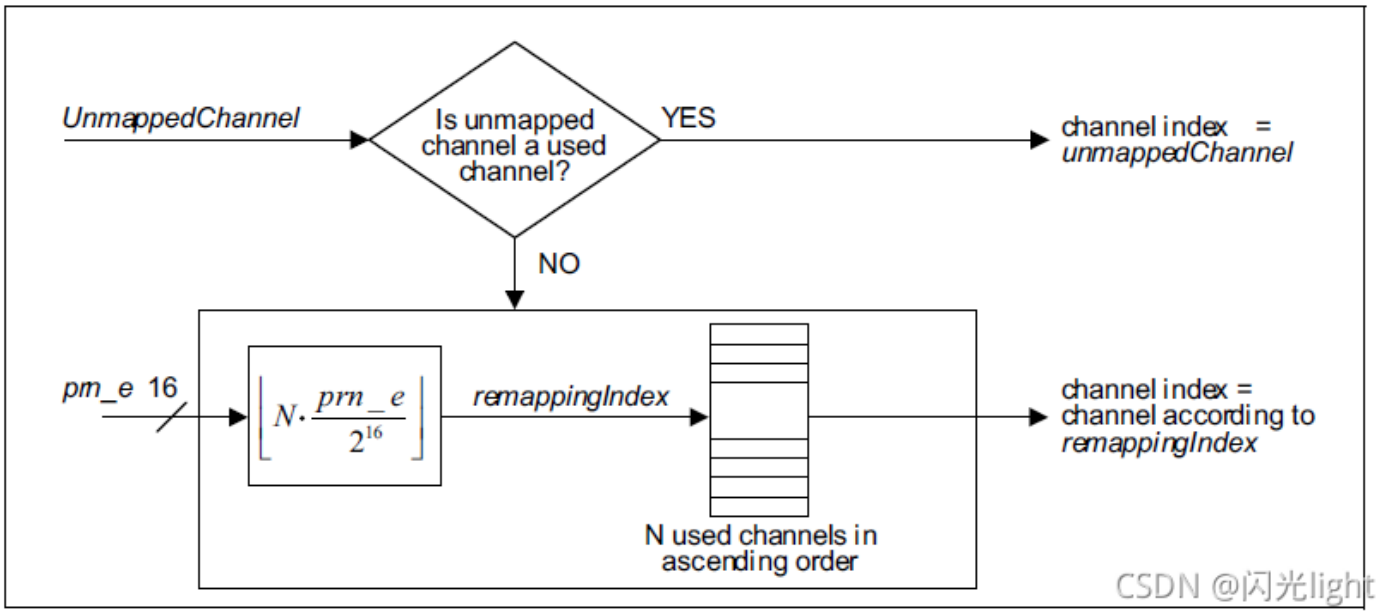
matlab代码实现如下

```
function outputMAM = MAM(a,b)
a = bin2dec(strrep(num2str(a),' ',''));
b = bin2dec(strrep(num2str(b),' ',''));
%先把输入的二进制数组变成十进制数计算
output_dec = mod(a.*17+b, 2.^16);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%把输出变回二进制数组
output_dec = dec2bin(output_dec,16);
outputMAM = str2num(output_dec(:))';
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

整个 Pseudo Random Number Generator 模块的matlab代码如下

```
function prn_e = PRNE(counter,channelIdentifier)
PERM_in_1 = xor(counter,channelIdentifier);
MAM_in_1 = PERM(PERM_in_1);
PERM_in_2 = MAM(MAM_in_1,channelIdentifier);
MAM_in_2 = PERM(PERM_in_2);
PERM_in_3 = MAM(MAM_in_2, channelIdentifier);
MAM_in_3 = PERM(PERM_in_3);
XOR_in_1 = MAM(MAM_in_3, channelIdentifier);
prn_e = xor(XOR_in_1,channelIdentifier);
end
```

Event Mapping to Used Channel Index模块的算法与信道选择算法1的结构类似：对UnmappedChannel的值进行判断，若其位于Channel map中，则该次时间的信道选择就是UnmappedChannel，否则，生成remappingIndex来进行重映射。



prn_e通过以下公式生成remappingIndex：

$$remappingindex = \left\lfloor \left(\frac{N * prn\_e}{2^{16}} \right) \right\rfloor$$

⌊⌋表示向下取整

整体信道选择算法的matlab算法如下：

```
clear;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
channel_map = [9 10 21 22 23 33 34 35 36];
N=9;
Access_Address = '8e89bed6';
```

```
counter = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1];
                                                AA = str_bin(Access_Address);

%AA是Access_Address的二进制列表，这里为[1,0,0,0,1,1,1,0,1,0,0,0,1,0,0,1,1,0,1,1,1,1,1,0,1,1
channelIdentifier = xor(AA(1:16),AA(17:32));    prn_e = PRNE(counter,channelIdentifier
unmappedChannel = bin2dec(strrep(num2str(prn_e),' ',''));
unmappedChannel = mod (unmappedChannel, 37);
if ismember(unmappedChannel,channel_map)
    channel_index = unmappedChannel;
else
prn_e_dec = bin2dec(strrep(num2str(prn_e),' ',''));
remappingIndex = floor(prn_e_dec.*N/(2.^16));
channel_index = channel_map(remappingIndex+1);
end
clc;
```

当设置

channel_map = [9 10 21 22 23 33 34 35 36];
N=9;
Access_Address = '8e89bed6';
counter = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1];

时，matlab仿真的各参数值如下，输出信道为9

AA=[1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 1 0];

channel_index = 9;

prn_e = [0 0 1 1 0 0 0 0 0 1 0 1 1 1 1 1];

prn_e_dec = 1685;

remappingIndex = 20;

unmappedChannel = 0;

当改变Channel map和count的值时，matlab得出的结果与协议中的相同。

Channel map [36:0] = 11111_11111111_11111111_11111111_11111111b.

Counter	1	2	3
prn_e	1685	38301	27475
unmappedChannel	20	6	21
mappedChannel	20	6	21

Channel map [36:0] =11110_00000000_11100000_00000110_00000000b.

The remapping table is [9, 10, 21, 22, 23, 33, 34, 35, 36].

Counter	6	7	8
prn_e	10975	5490	46970
unmappedChannel	23	14	17
mappedChannel	23	9	34