前言

在 CAN 协议里，报文的标识符不代表节点的地址，而是跟报文的内容相关的。因此，发送者以广播的形式把报文发送给所有的接收者。节点在接收报文时,根据标识符(CAN ID)的值决定软件是否需要该报文；如果需要，就拷贝到 SRAM 里；如果不需要，报文就被丢弃且无需软件的干预。

为满足这一需求，bxCAN 为应用程序提供了 14 个位宽可变的、可配置的过滤器组(13~0)，以便只接收那些软件需要的报文。硬件过滤的做法节省了 CPU 开销，否则就必须由软件过滤从而占用一定的 CPU 开销。每个过滤器组 x 由 2 个 32 位寄存器，CAN_FxR0 和 CAN_FxR1 组成。

为了让大家了解 STM32 的 bxCAN 的接收过滤机制，首先大家需要了解几个概念。

2 几个重要的概念

2.1 过滤器组

STM32 总共提供 14 个过滤器组来处理 CAN 接收过滤问题，每个过滤器组包含两个 32 位寄存器 CAN_FxR0 和 CAN_FxR1 组成，在设置为屏蔽位模式下，其中一个作为标识符寄存器，另一个作为屏蔽码寄存器。过滤器组中的每个过滤器，编号(叫做过滤器号)从 0 开始，到某个最大数值（这时最大值并非 13，而是取决于 14 个过滤器组的模式和位宽的设置，当全部配置为位宽为 16，且为标识符列表模式时，最大编号为 14*4-1=55）。

2.2 过滤器的过滤模式

STM32 提供两种过滤模式供用户设置：屏蔽位模式和标识符列表模式。

2.2.1 屏蔽位模式

为了过滤出一组标识符，应该设置过滤器组工作在屏蔽位模式。

在屏蔽位模式下，标识符寄存器和屏蔽寄存器一起，指定报文标识符的任何一位，应该按照“必须匹配”或“不用关心”处理。

2.2.2 标识符列表模式

为了过滤出一个标识符，应该设置过滤器组工作在标识符列表模式。

在标识符列表模式下，屏蔽寄存器也被当作标识符寄存器用。因此，不是采用一个标识符加一个屏蔽位的方式，而是使用 2 个标识符寄存器。接收报文标识符的每一位都必须跟过滤器标识符相同。

2.3 过滤器的位宽

每个过滤器组的位宽都可以独立配置，以满足应用程序的不同需求。根据位宽的不同，每个过滤器组可提供：

●1 个 32 位过滤器，包括：STDID[10:0]、EXTID[17:0]、IDE 和 RTR 位

●2 个 16 位过滤器，包括：STDID[10:0]、IDE、RTR 和 EXTID[17:15]位

2.3 过滤器组的过滤模式和位宽设置

过滤器组可以通过相应的 CAN_FMR 寄存器（CAN 过滤器主控寄存器）配置。但是不是什么时候都可以直接配置，在配置一个过滤器组前，必须通过清除 CAN_FAR 寄存器（CAN 过滤器激活寄存器）的 FACT 位，把它设置为禁用状态。然后才能设置或设置过滤器组的配置。

- 通过设置 CAN_FS1R（CAN 过滤器位宽寄存器）的相应 FSCx 位，可以配置一个过滤器组的位宽。
- 通过 CAN_FM1R（CAN 过滤器模式寄存器）的 FBMx 位，可以配置对应的屏蔽/标识符寄存器的标识符列表模式或屏蔽位模式。*(见后续 3.2 节)*

应用程序不用的过滤器组，应该保持在禁用状态。

关于过滤器配置，可参见下图:

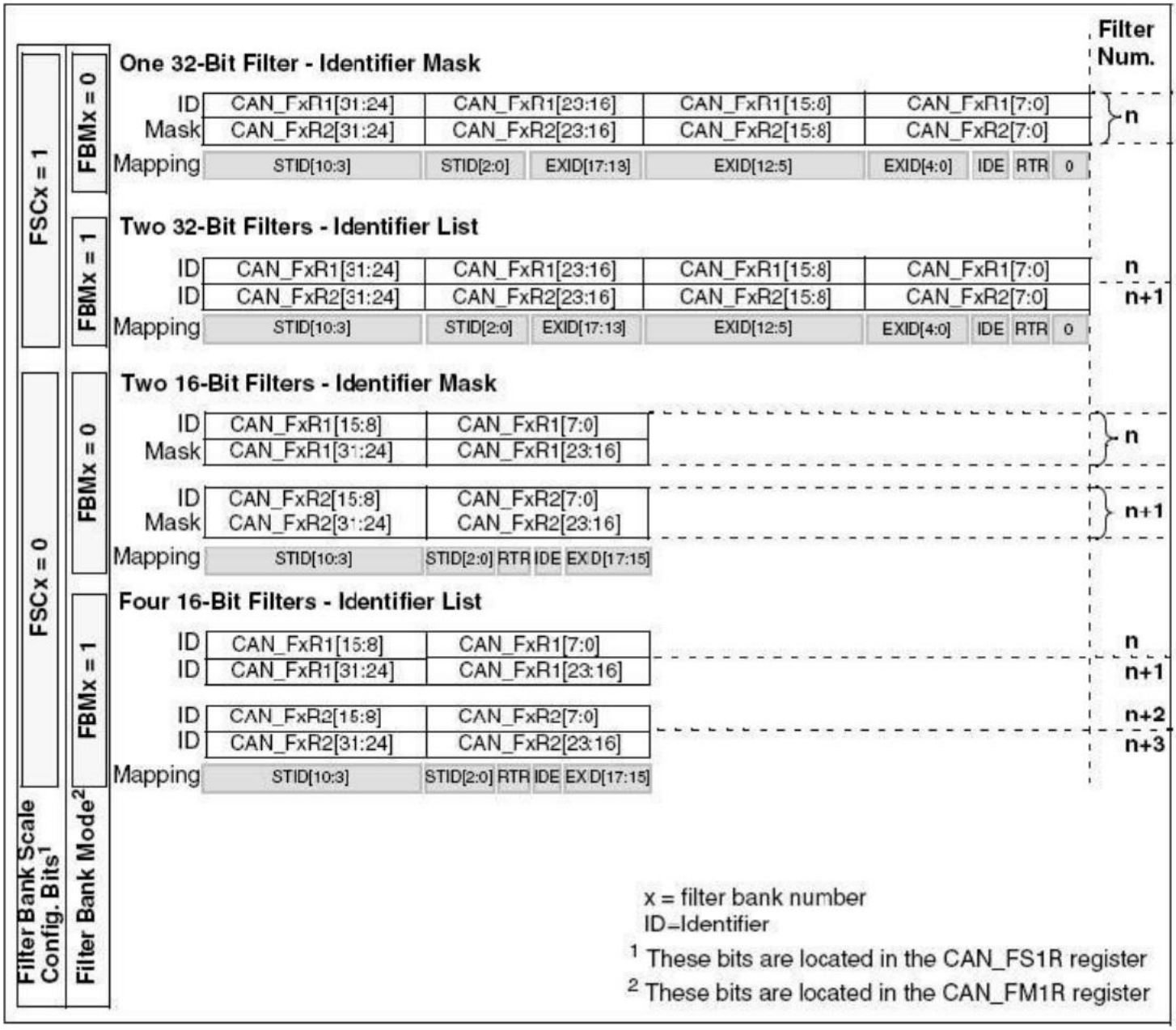

图 **1**

2.4 过滤器匹配序号

一旦收到的报文被存入 FIFO，就可被应用程序访问。通常情况下，报文中的数据被拷贝到 SRAM 中；为了把数据拷贝到合适的位置，应用程序需要根据报文的标识符来辨别不同的数据。bxCAN 提供了过滤器匹配序号，以简化这一辨别过程。

根据过滤器优先级规则，过滤器匹配序号和报文一起，被存入邮箱中。因此每个收到的报文，都有与它相关联的过滤器匹配序号。

过滤器匹配序号可以通过下面两种方式来使用：

- 把过滤器匹配序号跟一系列所期望的值进行比较
- 把过滤器匹配序号当作一个索引来访问目标地址

对于标识符列表模式下的过滤器(非屏蔽方式的过滤器)，软件不需要直接跟标识符进行比较。

对于屏蔽位模式下的过滤器，软件只须对需要的那些屏蔽位(必须匹配的位)进行比较即可。

在给过滤器编号时，并不考虑过滤器组是否为激活状态。另外，每个 FIFO 各自对其关联的过滤器进行编号,如下图：

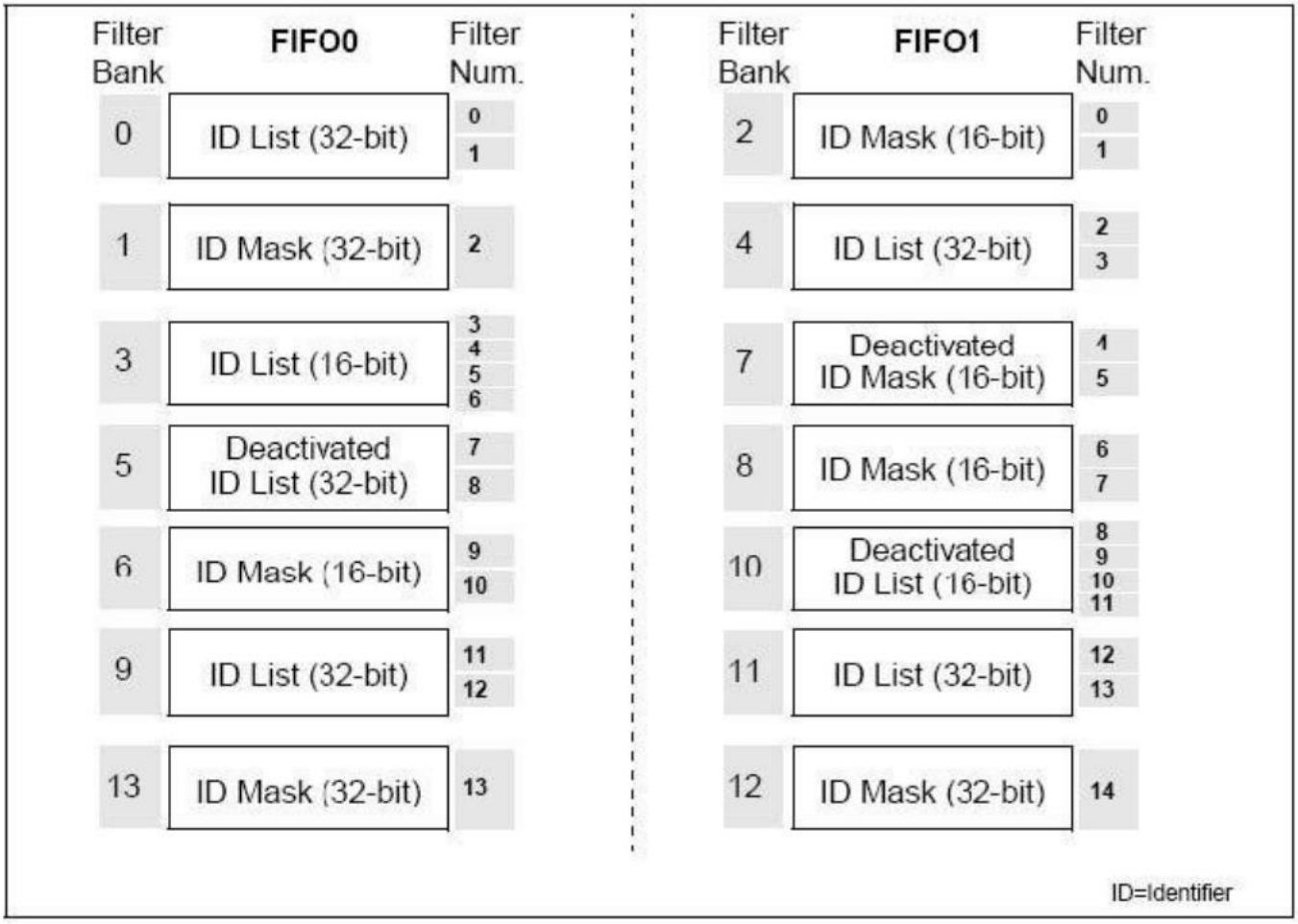

图 **2**

2.5 过滤器优先级规则

根据过滤器的不同配置，有可能一个报文标识符能通过多个过滤器的过滤；在这种情况下，存放在接收邮箱中的过滤器匹配序号，根据下列优先级规则来确定：

- 位宽为 32 位的过滤器，优先级高于位宽为 16 位的过滤器
- 对于位宽相同的过滤器，标识符列表模式的优先级高于屏蔽位模式
- 位宽和模式都相同的过滤器，优先级由过滤器号决定，过滤器号小的优先级高

如下图:

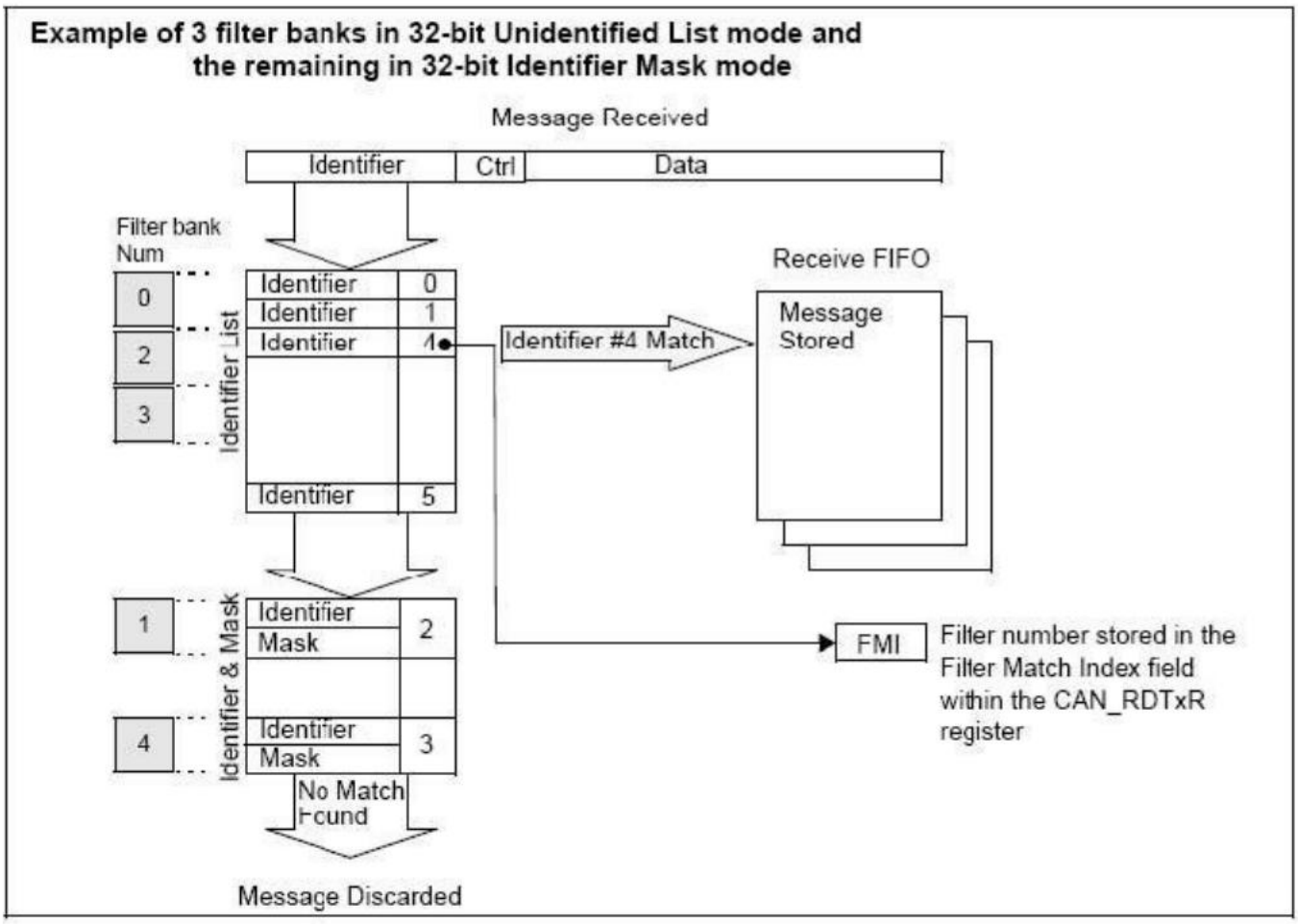

图 **3**

如上图，在接收一个报文时，其标识符首先与配置在标识符列表模式下的过滤器相比较；如果匹配上，报文就被存放到相关联的 FIFO 中，并且所匹配的过滤器的序号（这时为 4）被存入过滤器匹配序号中。如同例子中所显示，报文标识符跟#4 标识符匹配，因此报文内容和 FMI4 被存入 FIFO。

如果没有匹配，报文标识符接着与配置在屏蔽位模式下的过滤器进行比较。

如果报文标识符没有跟过滤器中的任何标识符相匹配，那么硬件就丢弃该报文，且不会对软件有任何打扰。

3 与过滤器相关的寄存器

3.1 CAN 过滤器主控寄存器 (CAN_FMR)

地址偏移量: 0x200

复位值: 0x2A1C 0E01

注：该寄存器的非保留位完全由软件控制。

图 **4**

位 31:1	保留位，强制为复位值。
位 0	FINIT：过滤器初始化模式 针对所有过滤器组的初始化模式设置。 0: 过滤器组工作在正常模式； 1: 过滤器组工作在初始化模式。

3.2 CAN 过滤器模式寄存器 (CAN_FM1R)

地址偏移量: 0x204

复位值: 0x0000 0000

注：只有在设置 CAN_FMR(FINIT=1)，使过滤器处于初始化模式下，才能对该寄存器写入。

图 **5**

位 31:14	保留位，硬件强制为 0
位 13:0	FBMx：过滤器模式 过滤器组 x 的工作模式。 0: 过滤器组 x 的 2 个 32 位寄存器工作在标识符屏蔽位模式； 1: 过滤器组 x 的 2 个 32 位寄存器工作在标识符列表模式。

3.3 CAN 过滤器位宽寄存器 (CAN_FS1R)

地址偏移量: 0x20C

复位值: 0x0000 0000

注：只有在设置 CAN_FMR(FINIT=1)，使过滤器处于初始化模式下，才能对该寄存器写入。

图 **6**

位 31:14	保留位，硬件强制为 0
位 13:0	FSCx：过滤器位宽设置 过滤器组 x(13～0)的位宽。 0：过滤器位宽为 2 个 16 位； 1：过滤器位宽为单个 32 位。

3.4 CAN 过滤器 FIFO 关联寄存器 (CAN_FFA1R)

地址偏移量: 0x214

复位值: 0x0000 0000

注：只有在设置 CAN_FMR(FINIT=1)，使过滤器处于初始化模式下，才能对该寄存器写入。

图 **7**

位 31:14	保留位，硬件强制为 0。
位 13:0	FFAx：过滤器位宽设置 报文在通过了某过滤器的过滤后，将被存放到其关联的 FIFO 中。 0：过滤器被关联到 FIFO0； 1：过滤器被关联到 FIFO1。

3.5 CAN 过滤器激活寄存器 (CAN_FA1R)

地址偏移量: 0x21C

复位值: 0x0000 0000

图 **7**

位 31:14	保留位，硬件强制为 0。
位 13:0	FACTx：过滤器激活 软件对某位设置 1 来激活相应的过滤器。只有对 FACTx 位清 0，或对 CAN_FMR 寄存器的 FINIT 位设置 1 后，才能修改相应的过滤器寄存器 x(CAN_FxR[0:1])。 0：过滤器被禁用； 1：过滤器被激活。

3.6 CAN 过滤器组 x 寄存器 (CAN_FiRx) (i＝0..13，x=1..2)

地址偏移量：0x240h..0x2AC

复位值：未定义位

注：共有 14 组过滤器：i=0..13。每组过滤器由 2 个 32 位的寄存器，CAN_FiR[2:1]组成。只有在 CAN_FaxR 寄存器（CAN 过滤器激活寄存器）相应的 FACTx 位清'0'，或 CAN_FMR 寄存器（CAN 过滤器主控寄存器）的 FINIT 位为'1'时，才能修改相应的过滤器寄存器。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
FB31	FB30	FB29	FB28	FB27	FB26	FB25	FB24	FB23	FB22	FB21	FB20	FB19	FB18	FB17	FB16
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
FB15	FB14	FB13	FB12	FB11	FB10	FB9	FB8	FB7	FB6	FB5	FB4	FB3	FB2	FB1	FB0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 8

位 31:0	FB[31:0]：过滤器位 • 当为标识符模式时： 寄存器的每位对应于所期望的标识符的相应位的电平。 0：期望相应位为显性位； 1：期望相应位为隐性位。 • 当为屏蔽位模式时： 寄存器的每位指示是否对应的标识符寄存器位一定要与期望的标识符的相应位一致。 0：不关心，该位不用于比较； 1：必须匹配，到来的标识符位必须与滤波器对应的标识符寄存器位相一致。

注：根据过滤器位宽和模式的不同设置，过滤器组中的两个寄存器的功能也不尽相同。。关于过滤器的映射，功能描述和屏蔽寄存器的关联，请参见 2 节标识符过滤。

屏蔽位模式下的屏蔽/标识符寄存器，跟标识符列表模式下的寄存器位定义相同。

4 代码实例

4.1 CAN ID 值的结构分析

在讲到代码实例之前，首先大家都弄懂一件事，当给定一个 CAN ID，如 0x1800f001,当然这个是扩展 ID，这里要问的是，这个 CAN ID 的值本身包含两部分，即基本 ID 与扩展 ID，即么你知道这个扩展 ID0x1800f001 的哪些位是基本 ID，哪些位又是扩展 ID？（在基本 CANID 格式下不存在这个问题）

在回答这个问题之前我们来看看 ISO11898 的定义，如下图：

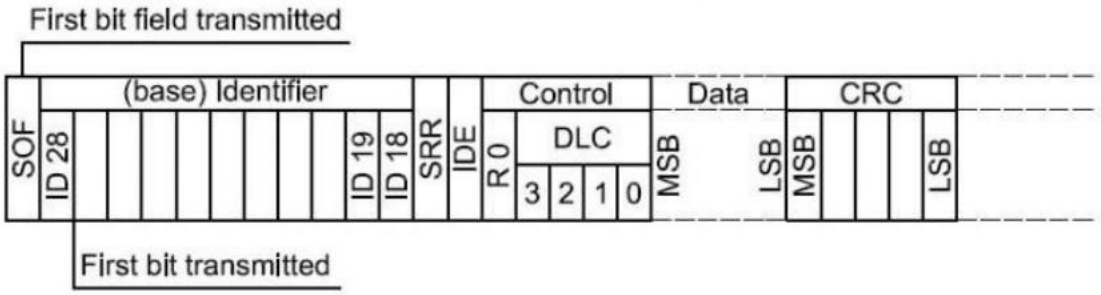

Figure 10 — Order of bit transmission (base format)

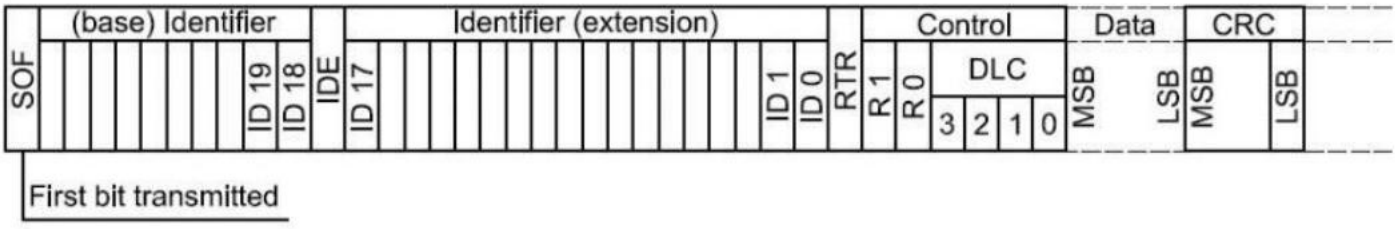

Figure 11 — Order of bit transmission (extended format)

图 **9**

如上图，基本格式不存在扩展 ID，而扩展格式中 ID0~ID17 为 Extension ID,而 ID18~ID28 为 Base ID.

因此 CAN ID 值 0x1800f001 用二进制表示为：0b 0001 1000 0000 0000 1111 0000 0000 0001,用括号分别区别为：0b 000[1 1000 0000 00][00 1111 0000 0000 0001]，红色部分为扩展 ID，蓝色部分为基本 ID。那么知道这些有什么用呢？接下来的代码示例中你就会有什么用了。

4.2 位宽为 32 位的屏蔽模式

在此种模式下中过滤多个 CAN ID，此时，过滤器包含两个寄存器，屏蔽码寄存器和标识符寄存器。此模式下最多只存在一个屏蔽过滤器。

如下图所示：

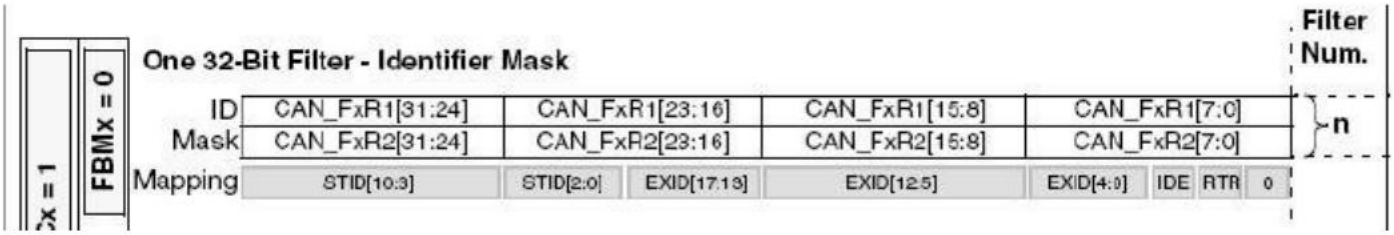

图 **10**

如上图，上面的 ID 为标识符寄存器，中间部分的 MASK 为屏蔽码寄存器。每个寄存器都是 32 位的。最下边显示的是与 CAN ID 各位定位的映射关系。由 4.1 的知识很快可以发现，上图最下边的映射关系恰好等于扩展 CAN 值左移 3 位再补上 IDE（扩展帧标识），RTR（远程帧标志）。

因此，我们初步得出这样的推论：对于一个扩展 CAN ID，不能单纯地将它看到的一个数，而应该将它看成两部分，基本 ID 和扩展 ID(当然标准 CAN ID 只包含基本 ID 部分)，过滤器屏蔽码寄存器和标识符寄存器也应该看成多个部分，然后问题就变成了如何将 CAN ID 所表示的各部分如何针对过滤器寄存器各部分对号入座的问题了。
对号入座的方法多种多样，但万变不离其心，主要是掌握其核心思想即可：1:在各种过滤器模式下，CAN ID 与寄存器相应位置一定要匹配；2：在屏蔽方式下，屏蔽码寄存器某位为 1 表示接收到的 CAN ID 对应的位必须对验证码寄存器对应的位相同。

下面给出一个代码例子，假设我们要接收多个 ID：0x7e9,0x1800f001,前面为标准 ID，后面为扩展 ID，要同时能接收这两个 ID，那么该如何设置这个过滤器呢?

[cpp]view plaincopyprint?

```
1.  CAN_FilterInitTypeDefCAN_FilterInitStructure;
2.  U16 std_id =0x7e9;
3.  U32 ext_id =0x1800f001;
4.  U32 mask =0;
5.
6.  CAN_FilterInit(&CAN_FilterInitStructure); //初始化 CAN_FilterInitStructrue 结构体变量
7.  CAN_FilterInitStructure.CAN_FilterNumber=0; //设置过滤器组 0，范围为 0~13
8.  CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask; //设置过滤器组 0 为屏蔽模式
9.  CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit; //设置过滤器组 0 位宽为 32 位
10.
11. //标识位寄存器的设置
12. //ext_id<<3 对齐，见上图 9，再>>16 取高 16 位
13. CAN_FilterInitStructure.CAN_FilterIdHigh=（(ext_id<<3) >>16) &0xffff; //设置标识符寄存器高字节。
14. CAN_FilterInitStructure.CAN_FilterIdLow=(U16)(ext_id<<3) | CAN_ID_EXT; //设置标识符寄存器低字节
15. //这里也可以这样设置
16. //CAN_FilterInitStructure.CAN_FilterIdHigh=std_id<<5; //设置标识符寄存器高字节.这里为什么是左移 5
    位呢？从上图可以看出，CAN_FilterIdHigh 包含的是 STD[0~10]和 EXID[13~17],标准 CAN ID 本身是不包含扩展
    ID 数据，因此为了要将标准 CAN ID 放入此寄存器，标准 CAN ID 首先应左移 5 位后才能对齐.
17. //CAN_FilterInitStructure.CAN_FilterIdLow=0|CAN_ID_EXT; //设置标识符寄存器低字节,这里也可以设置
    为 CAN_ID_STD
18.
19. //屏蔽寄存器的设置
20. //这里的思路是先将标准 CAN ID 和扩展 CAN ID 对应的 ID 值先异或后取反，为什么？异或是为了找出两个 CAN ID
    有哪些位是相同的，是相同的位则说明需要关心，需要关心的位对应的屏蔽码位应该设置为 1，因此需要取反一下。
    最后再整体左移 3 位。
21. mask =(std_id<<18);//这里为什么左移 18 位？因为从 ISO11898 中可以看出，标准 CAN ID 占 ID18~ID28，为了
    与 CAN_FilterIdHigh 对齐，应左移 2 位，接着为了与扩展 CAN 对应，还应该再左移 16 位，因此，总共应左移 2+16
```

```
    =18 位。也可以用另一个方式来理解：直接看 Mapping 的内容，发现 STDID 相对 EXID[0]偏移了 18 位,因此左移
    18 位.
22. mask ^=ext_id;//将对齐后的标准 CAN 与扩展 CAN 异或后取反
23. mask =~mask;
24. mask <<=3;//再整体左移 3 位
25. mask |=0x02; //只接收数据帧，不接收远程帧
26. CAN_FilterInitStructure.CAN_FilterMaskIdHigh=(mask>>16)&0xffff; //设置屏蔽寄存器高字节
27. CAN_FilterInitStructure.CAN_FilterMaskIdLow=mask&0xffff; //设置屏蔽寄存器低字节
28.
29. CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_FIFO0; //此过滤器组关联到接收 FIFO0
30. CAN_FilterInitStructure.CAN_FilterActivation=ENABLE; //激活此过滤器组
31. CAN_FilterInit(&CAN_FilterInitStructure); //设置过滤器
```

总结可知，当过滤器为屏蔽模式时，标识符寄存器对应的 ID 内容可为任意一需求接收的 ID 值，当同时要接收标准帧和扩展帧时，标识符寄存器对应 IDE 位也随意设置，屏蔽寄存器的 IDE 位设置为 0，表示不关心标准帧还是扩展帧。而屏蔽寄存器对应的 ID 内容为各需求接收的 ID 值依次异或的结果再取反。

4.3 位宽为 32 位的标识符列表模式

在此种模式下，过滤器组包含的两个寄存器含义一样，此模式下只多存在两个标识符列表过滤器如下图：

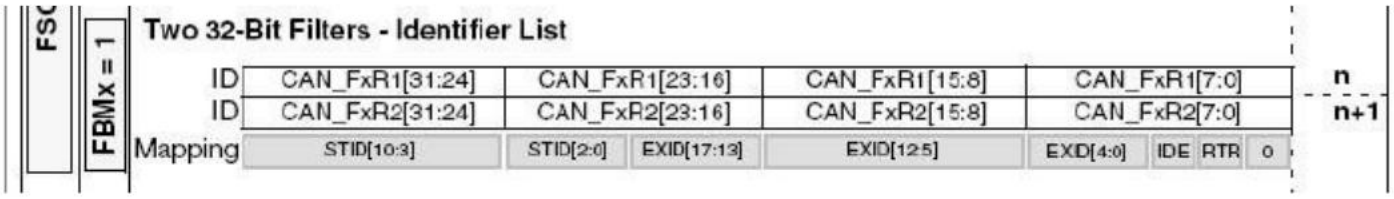

图 11

```
[cpp]view plaincopyprint?
```

```
1.  CAN_FilterInitTypeDefCAN_FilterInitStructure;
2.  U16 std_id =0x7e9;
3.  U32 ext_id =0x1800f001;
4.  CAN_FilterInit(&CAN_FilterInitStructure); //初始化 CAN_FilterInitStructrue 结构体变量
5.  CAN_FilterInitStructure.CAN_FilterNumber=0; //设置过滤器组 0，范围为 0~13
6.  CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdList; //设置过滤器组 0 为标识符列表模式
7.  CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit; //设置过滤器组 0 位宽为 32 位
8.
9.  //设置屏蔽寄存器，这里当标识符寄存器用
10. CAN_FilterInitStructure.CAN_FilterIdHigh=std_id<<5) ; //为什么左移 5 位？与上面相同道理，这里不再
    重复解释
11. CAN_FilterInitStructure.CAN_FilterIdLow=0|CAN_ID_STD; //设置标识符寄存器低字节,CAN_FilterIdLow
    的 ID 位可以随意设置，在此模式下不会有效。
12.
```

```
13. //设置标识符寄存器
14. CAN_FilterInitStructure.CAN_FilterMaskIdHigh=((ext_id<<3)>>16) & 0xffff; //设置屏蔽寄存器高字节
15. CAN_FilterInitStructure.CAN_FilterMaskIdLow=((ext_id<<3)& 0xffff) | CAN_ID_EXT; //设置屏蔽寄存
    器低字节
16.
17. CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_FIFO0; //此过滤器组关联到接收 FIFO0
18. CAN_FilterInitStructure.CAN_FilterActivation=ENABLE; //激活此过滤器组
19. CAN_FilterInit(&CAN_FilterInitStructure); //设置过滤器
```

4.4 位宽为 16 位的屏蔽码模式

在此模式下，最多存在两个屏蔽码过滤器，如下图：

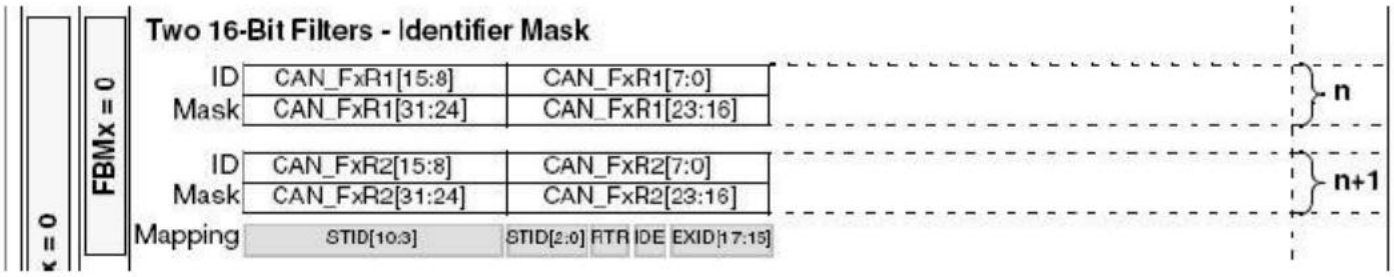

图 12

由上图映射可知，最下面的映射只包含 STDID0~ID10,因此，此模式下的两个屏蔽过滤器只能实现对标准 ID 的过滤。具体代码就不介绍了，参见上图的映射即可。

4.5 位宽为 16 位的标识符列表模式

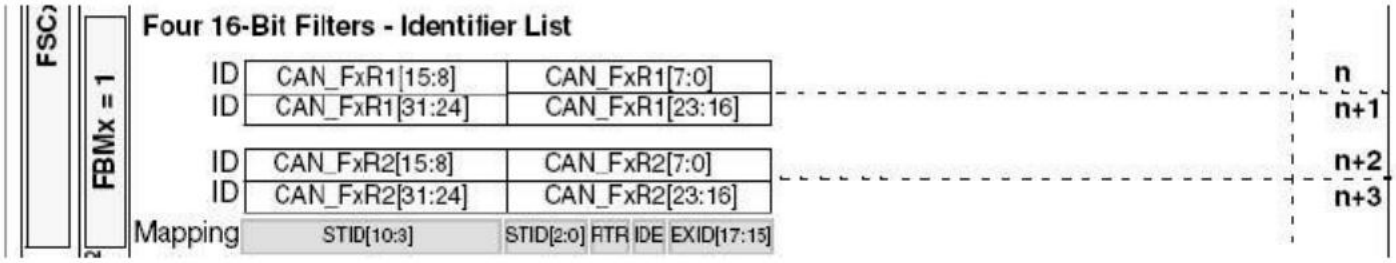

图 13

在此模式下，由于标识符寄存器的高 16 位和低 16 位，屏蔽寄存器的高 16 位和低 16 位都用来做标识符寄存器，因此，最多可存在 4 个标识符过滤器。同样，只能实现对标准帧的过滤。具体代码就不介绍了，参见上图的映射即可。