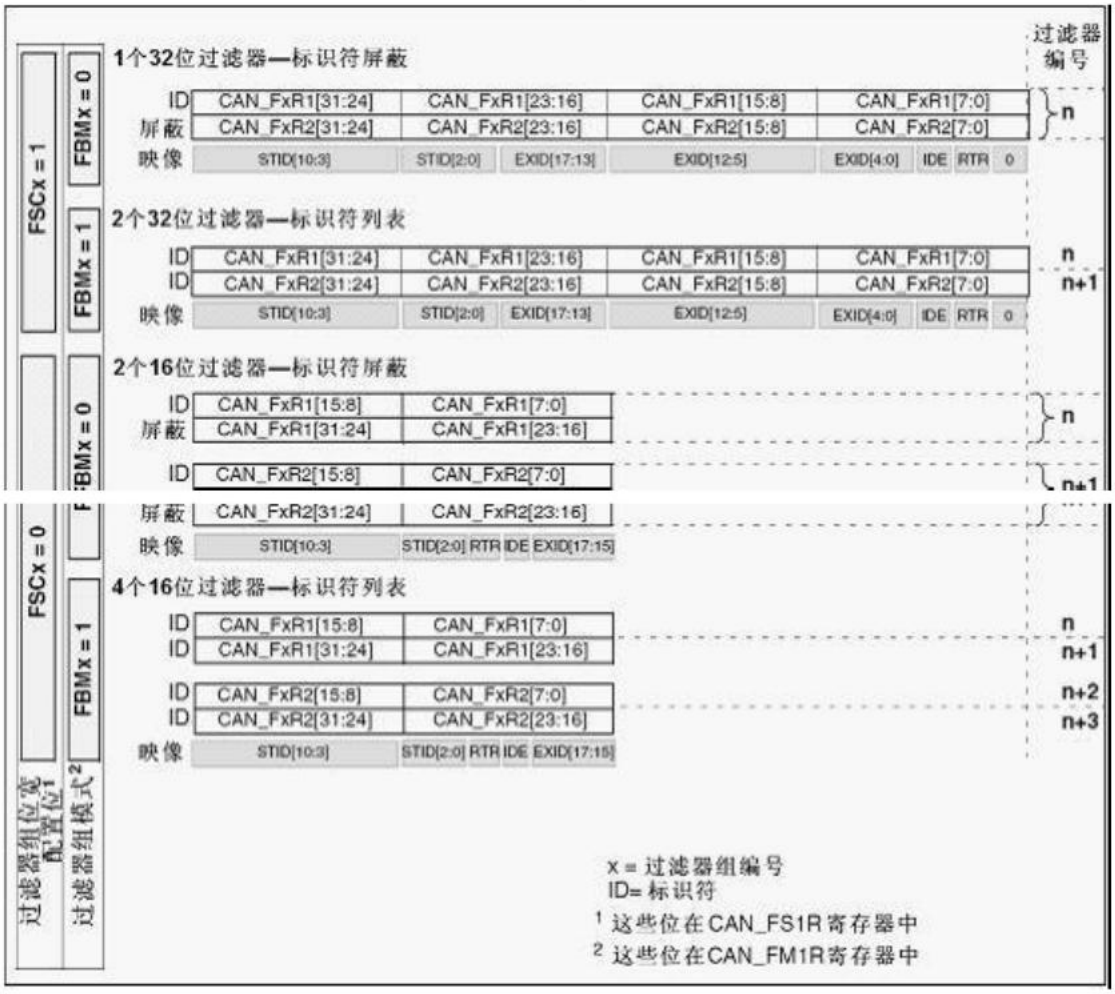
STM32 CAN 过滤器总结

在 STM32 互联型产品中，CAN1 和 CAN2 分享 28 个过滤器组,其它 STM32F103xx 系列产品中有 14 个过滤器组,用以对接收到的帧进行过滤。

每组过滤器包括了 2 个可配置的 32 位寄存器：CAN_FxR0 和 CAN_FxR1。这些过滤器相当于关卡，每当收到一条报文时，CAN 要先将收到的报文从这些过滤器上"过"一下，能通过的报文是有效报文，收进相关联 FIFO（FIFO1 或 FIFO2），不能通过的是无效报文(不是发给"我"的报文)，直接丢弃。

（标准 CAN 的标志长度是 11 位。扩展格式 CAN 的标志长度是 29。CAN2.0A 协议规定 CAN 控制器必须有一个 11 位的标识符。CAN2.0B 协议中规定 CAN 控制器的标示符长度可以是 11 位或 29 位。STM32 同时支持 CAN2.0A/CAN2.0B 协议。）



每组过滤器组有两种工作模式：标识符列表模式和标识符屏蔽位模式。

标识符屏蔽位模式：可过滤出一组标识符。此时，这样 CAN_FxR0 中保存的就是标识符匹配值，CAN_FxR1 中保存的是屏蔽码，即 CAN_FxR1 中如果某一位为 1，则 CAN_FxR0 中相应的位必须与收到的帧的标志符中的相应位吻合才能通过过滤器；CAN_FxR1 中为 0 的位表示 CAN_FxR0 中的相应位可不必与收到的帧进行匹配。

标识符列表模式：可过滤出一个标识。此时 CAN_FxR0 和 CAN_FxR1 中的都是要匹配的标识符，收到的帧的标识符必须与其中的一个吻合才能通过过滤。

注意：CAN_FilterIdHigh 是指高 16 位 CAN_FilterIdLow 是低 16 位应该将需要得到的帧的和过滤器的设置值左对齐起。

所有的过滤器是并联的，即一个报文只要通过了一个过滤器，就是算是有效的。

按工作模式和宽度，一个过滤器组可以变成以下几中形式之一：

(1) 1 个 32 位的屏蔽位模式的过滤器。

(2) 2 个 32 位的列表模式的过滤器。

(3) 2 个 16 位的屏蔽位模式的过滤器。

(4) 4 个 16 位的列表模式的过滤器。

每组过滤器组有两个 32 位的寄存器用于存储过滤用的"标准值"，分别是 FxR1，FxR2。

在 32 位的屏蔽位模式下：

有 1 个过滤器。

FxR2 用于指定需要关心哪些位，FxR1 用于指定这些位的标准值。

在 32 位的列表模式下：

有两个过滤器。

FxR1 指定过滤器 0 的标准值 FxR2 指定过滤器 1 的标准值。

收到报文的标识符只有跟 FxR1 与 FxR1 其中的一个完全相同时，才算通过。

在 16 位的屏蔽位模式下：

有 2 个过滤器。

FxR1 配置过滤器 0，其中，[31-16]位指定要关心的位，[15-0]位指定这些位的标准值。

FxR2 配置过滤器 1，其中，[31-16]位指定要关心的位，[15-0]位指定这些位的标准值。

在 16 位的列表模式下：

有 4 个过滤器。

FxR1 的[15-0]位配置过滤器 0，FxR1 的[31-16]位配置过滤器 1。

FxR2 的[15-0]位配置过滤器 2，FxR2 的[31-16]位配置过滤器 3。

STM32 的 CAN 有两个 FIFO，分别是 FIFO0 和 FIFO1。为了便于区分，下面 FIFO0 写作 FIFO_0，FIFO1 写作 FIFO_1。

每组过滤器组必须关联且只能关联一个 FIFO。复位默认都关联到 FIFO_0。

所谓"关联"是指假如收到的报文从某个过滤器通过了，那么该报文会被存到该过滤器相连的 FIFO。

从另一方面来说，每个 FIFO 都关联了一串的过滤器组，两个 FIFO 刚好瓜分了所有的过滤器组。

每当收到一个报文，CAN 就将这个报文先与 FIFO_0 关联的过滤器比较，如果被匹配，就将此报文放入 FIFO_0 中。

如果不匹配，再将报文与 FIFO_1 关联的过滤器比较，如果被匹配，该报文就放入 FIFO_1 中。

如果还是不匹配，此报文就被丢弃。

每个 FIFO 的所有过滤器都是并联的，只要通过了其中任何一个过滤器,该报文就有效。

如果一个报文既符合 FIFO_0 的规定，又符合 FIFO_1 的规定，显然，根据操作顺序，它只会放到 FIFO_0 中。

每个 FIFO 中只有激活了的过滤器才起作用，换句话说，如果一个 FIFO 有 20 个过滤器，但是只激话了 5 个，那么比较报文时，只拿这 5 个过滤器作比较。

一般要用到某个过滤器时，在初始化阶段就直接将它激活。

需要注意的是，每个 FIFO 必须至少激活一个过滤器，它才有可能收到报文。如果一个过滤器都没有激活，那么是所有报文都报废的。

一般的，如果不想用复杂的过滤功能，FIFO 可以只激活一组过滤器组，且将它设置成 32 位的屏蔽位模式，两个标准值寄存器(FxR1，FxR2)都设置成 0。这样所有报文均能通过。（STM32 提供的例程里就是这么做的！）

STM32 CAN 中，另一个较难理解的就是过滤器编号。

过滤器编号用于加速 CPU 对收到报文的处理。

收到一个有效报文时， CAN 会将收到的报文 以及它所通过的过滤器编号， 一起存入接收邮箱中。CPU 在处理时，可以根据过滤器编号，快速的知道该报文的用途，从而作出相应处理。

不用过滤器编号其实也是可以的， 这时候 CPU 就要分析所收报文的标识符， 从而知道报文的用途。

由于标识符所含的信息较多，处理起来就慢一点了。

STM32 使用以下规则对过滤器编号：

(1) FIFO_0 和 FIFO_1 的过滤器分别独立编号，均从 0 开始按顺序编号。

(2) 所有关联同一个 FIFO 的过滤器，不管有没有被激活，均统一进行编号。

(3) 编号从 0 开始，按过滤器组的编号从小到大，按顺序排列。

(4) 在同一过滤器组内，按寄存器从小到大编号。FxR1 配置的过滤器编号小，FxR2 配置的过滤器编号大。

(5) 同一个寄存器内，按位序从小到大编号。[15-0]位配置的过滤器编号小，[31-16]位配置的过滤器编号大。

(6) 过滤器编号是弹性的。 当更改了设置时，每个过滤器的编号都会改变。

但是在设置不变的情况下，各个过滤器的编号是相对稳定的。

这样，每个过滤器在自己在 FIFO 中都有编号。

在 FIFO_0 中，编号从 0 -- (M-1)， 其中 M 为它的过滤器总数。

在 FIFO_1 中，编号从 0 -- (N-1),，其中 N 为它的过滤器总数。

一个 FIFO 如果有很多的过滤器,，可能会有一条报文，在几个过滤器上均能通过，这时候,，这条报文算是从哪儿过来的呢？

STM32 在使用过滤器时，按以下顺序进行过滤：

(1) 位宽为 32 位的过滤器，优先级高于位宽为 16 位的过滤器。

(2) 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。

(3) 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。

按这样的顺序,报文能通过的第一个过滤器,就是该报文的过滤器编号,被存入接收邮箱中。

Eg.stm32 的 can 总线的配置如下：

```
CAN_InitStructure.CAN_TTCM=DISABLE;
CAN_InitStructure.CAN_ABOM=DISABLE;
CAN_InitStructure.CAN_AWUM=DISABLE;
CAN_InitStructure.CAN_NART=DISABLE;
CAN_InitStructure.CAN_RFLM=DISABLE;
CAN_InitStructure.CAN_TXFP=DISABLE;
CAN_InitStructure.CAN_Mode=CAN_Mode_Normal;
//CAN_Mode_LoopBack
//CAN_Mode_Normal
/* Baudrate = 125kbps*/
CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
CAN_InitStructure.CAN_BS1=CAN_BS1_2tq;
```

```
    CAN_InitStructure.CAN_BS2=CAN_BS2_3tq;
    CAN_InitStructure.CAN_Prescaler = 48;  /*设定了一个时间单位的长度，它的范围是 1
到 1024。其      实就是分频数 APB1*/
     CAN_Init(&CAN_InitStructure);
     /* CAN filter init */
     CAN_FilterInitStructure.CAN_FilterNumber=0;//选择过滤器 0
     CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;//指定过滤器被设
置为标识符屏蔽模式
     CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;//给出过滤器位宽为
32 位
 下面根据设置的参数不同来决定 can 总线 can 总线的配置情况：
1、对扩展数据帧进行过滤:(只接收扩展数据帧)
     CAN_FilterInitStructure.CAN_FilterIdHigh   = (((u32)slave_id<<3)&0xFFFF0000)>>1
6;
     CAN_FilterInitStructure.CAN_FilterIdLo=
               (((u32)slave_id<<3)|CAN_ID_EXT|CAN_RTR_DATA)&0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdHigh  = 0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdLow   = 0xFFFF;
(注：标准帧数据帧、标准远程帧和扩展远程帧均被过滤)
2、对扩展远程帧过滤:(只接收扩展远程帧)
     CAN_FilterInitStructure.CAN_FilterIdHigh   = (((u32)slave_id<<3)&0xFFFF0000)>>1
6;
     CAN_FilterInitStructure.CAN_FilterIdLow   = (((u32)slave_id<<3)|CAN_ID_EXT|CAN
_RTR_REMOTE)&0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdHigh  = 0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdLow   = 0xFFFF;
3、对标准远程帧过滤:(只接收标准远程帧)
     CAN_FilterInitStructure.CAN_FilterIdHigh   = (((u32)slave_id<<21)&0xffff0000)>>1
6;
     CAN_FilterInitStructure.CAN_FilterIdLow   = (((u32)slave_id<<21)|CAN_ID_STD|CA
N_RTR_REMOTE)&0xffff;
     CAN_FilterInitStructure.CAN_FilterMaskIdHigh  = 0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdLow   = 0xFFFF;
4、对标准数据帧过滤:(只接收标准数据帧)
     CAN_FilterInitStructure.CAN_FilterIdHigh   = (((u32)slave_id<<21)&0xffff0000)>>1
6;
     CAN_FilterInitStructure.CAN_FilterIdLow   = (((u32)slave_id<<21)|CAN_ID_STD|CA
N_RTR_DATA)&0xffff;
     CAN_FilterInitStructure.CAN_FilterMaskIdHigh  = 0xFFFF;
     CAN_FilterInitStructure.CAN_FilterMaskIdLow   = 0xFFFF;
5、对扩展帧进行过滤:(扩展帧不会被过滤掉)
     CAN_FilterInitStructure.CAN_FilterIdHigh   = (((u32)slave_id<<3)&0xFFFF0000)>>1
6;
     CAN_FilterInitStructure.CAN_FilterIdLow   = (((u32)slave_id<<3)|CAN_ID_EXT)&0xF
```

FFF;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0xFFFF;
CAN_FilterInitStructure.CAN_FilterMaskIdLow = 0xFFFC;
6、对标准帧进行过滤:(标准帧不会被过滤掉)
CAN_FilterInitStructure.CAN_FilterIdHigh = (((u32)slave_id<<21)&0xffff0000)>>16;
CAN_FilterInitStructure.CAN_FilterIdLow = (((u32)slave_id<<21)|CAN_ID_STD)&0xffff;
CAN_FilterInitStructure.CAN_FilterMaskIdHigh = 0xFFFF;
CAN_FilterInitStructure.CAN_FilterMaskIdLow = 0xFFFC;
注：slave_id 为要过滤的 id 号。