GigaDevice Semiconductor Inc.

GD32H73x_75x CAN 过滤器使用指南

应用笔记

AN284

1.0 版本

(2026 年 08 月)

目录

目录 1
图索引 2
表索引 3
1. 前言 4
2. 过滤器使用介绍 5
2.1. 过滤方式 5
2.2. 过滤数据配置 7
3. 过滤器使用举例 8
3.1. 禁能 FIFO，RPFQEN=0 8
3.2. 禁能 FIFO，RPFQEN=1 11
3.3. 使能 FIFO，RPFQEN=0 14
3.4. 使能 FIFO，RPFQEN=1 18
4. 版本历史 22

图索引

图 2-1. 接收邮箱的 ID 与 MASK 配置对应关系(1) 5
图 2-2. Rx FIFO 的 ID 与 MASK 配置对应关系 6

表索引

表 1-1. 适用产品 4
表 2-1. 过滤数据配置 7
表 2-2. Rx FIFO 标识符过滤表(RPFQEN = 1 时) 7
表 2-3. Rx FIFO 标识符过滤表(RPFQEN = 0 时) 8
表 3-1. CAN 模块参考配置 9
表 3-2. 主函数参考配置 10
表 3-3. CAN 模块参考配置 11
表 3-4. 主函数参考配置 12
表 3-5. CAN 模块参考配置 15
表 3-6. 主函数参考配置 16
表 3-7. CAN 模块参考配置 18
表 3-8. 主函数参考配置 19
表 4-1. 版本历史 22

1. 前言

GD32H73x_75x CAN 模块具有强大的过滤功能，由于模块的接收部分可配置为只有接收邮箱，也可以配置为共同使用接收邮箱与接收 FIFO，其中接收邮箱的期望 ID 与接收 FIFO 的期望 ID 设置方式不同，接收 FIFO 的期望 ID 设置更加灵活，另外 CAN 模块也提供了私有过滤器寄存器，上述这些原因导致 CAN 模块的过滤应用非常复杂。本文将介绍 CAN 模块过滤器所有配置情况、相关的库函数使用、以及使用举例。本文档适用型号参考*表 1-1. 适用产品*。

表 1-1. 适用产品

产品系列	型号
GD32H73x	GD32H737系列
GD32H75x	GD32H757、GD32H759系列

2. 过滤器使用介绍

CAN 总线控制器集成了可灵活配置的邮箱系统用于 CAN 帧的发送和接收。邮箱系统包含一组邮箱，用于存储控制数据，时间戳，消息标识符和消息数据，最大支持 32 个邮箱。每个 CAN 控制器内有 512 字节 RAM 空间用于邮箱或 FIFO 描述符的配置。

当 CAN 开启 FIFO 接收时，邮箱占用的 RAM 空间将用于接收 FIFO 描述符。接收 FIFO 具有标识符过滤的功能，最大支持 104 个扩展标识符的过滤，或者 208 个标准标识符的过滤，或者 416 个对标识符部分 8 位的过滤，最多有 32 个标识符过滤表元素可通过接收 FIFO/邮箱私有过滤寄存器进行配置。

当禁能Rx FIFO时：

- 如果CAN_CTL0寄存器的RPFQEN位为0，则使用CAN_RMPUBF寄存器来配置所有接收邮箱的过滤数据配置。
- 如果CAN_CTL0寄存器的RPFQEN位为1，则使用CAN_RFIFOMPFx(x = 0..31)寄存器来分别配置接收邮箱的过滤数据配置。

当使能Rx FIFO时：

- 如果CAN_CTL0寄存器的RPFQEN位为0，则使用CAN_RMPUBF寄存器来配置所有接收邮箱的过滤数据配置，使用CAN_RFIFOPUBF和CAN_RFIFOMPFx(x = 0..31)寄存器来配置所有Rx FIFO标识符过滤表元素，并且所有这些寄存器的值的配置必须相同。
- 如果CAN_CTL0寄存器的RPFQEN位为1，则使用CAN_RFIFOMPFx(x=0..31)寄存器来配置由CAN_CTL2寄存器RFFN[3:0]位域设置的Rx FIFO标识符过滤表元素以及接收邮箱(由于接收邮箱描述符和Rx FIFO描述符不能同时占用同一个区域的RAM，因此用一组寄存器进行独立控制过滤数据的配置)，由CAN_RFIFOPUBF寄存器来配置剩余所有的Rx FIFO标识符过滤表元素。

2.1. 过滤方式

GD32H73x_75x CAN模块过滤器采用ID MASK方式进行过滤。

用于配置过滤数据的寄存器主要有：CAN_RMPUBF，CAN_RFIFOMPFx(x = 0..31)，和CAN_RFIFOPUBF。第一个寄存器专用于接收邮箱过滤，最后一个寄存器专用于Rx FIFO过滤，中间的寄存器同时适用于接收邮箱和Rx FIFO过滤。

接收邮箱的ID与MASK配置对应关系如*图2-1. 接收邮箱的ID与MASK配置对应关系*所示。Rx FIFO的ID与MASK配置对应关系如*图2-2. Rx FIFO的ID与MASK配置对应关系*所示。

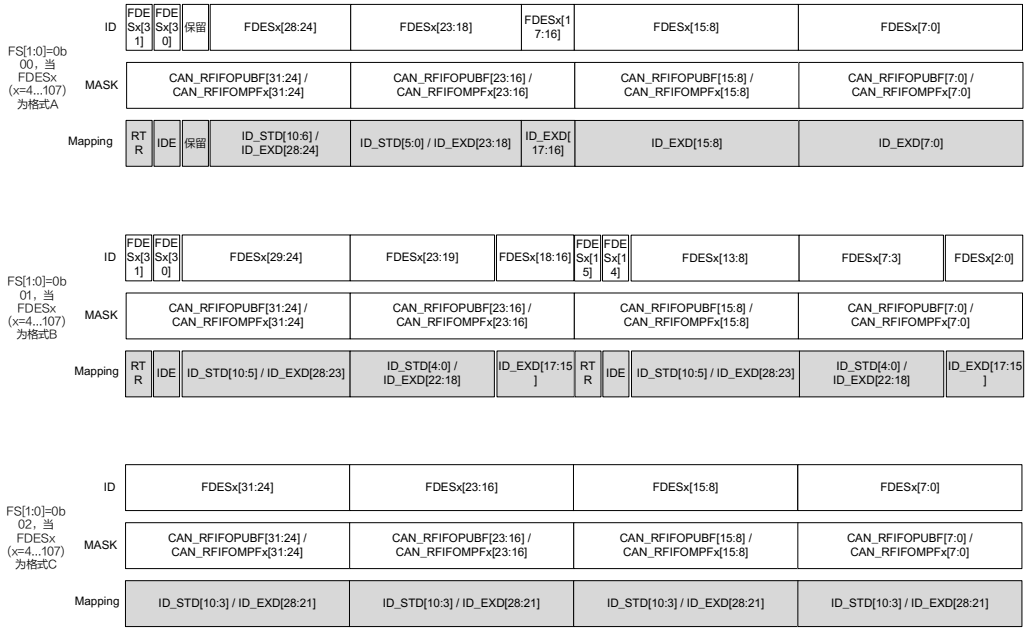
图 2-1. 接收邮箱的 ID 与 MASK 配置对应关系(1)

ID	MDES0[20]	MDES0[21]	保留	MDES1[28:24]	MDES1[23:18]	MDES1[17:16]	MDES1[15:8]	MDES1[7:0]
MASK	CAN_RMPUBF[31:24]] / CAN_RFIFOMPFx[31:24]				CAN_RMPUBF[23:16] / CAN_RFIFOMPFx[23:16]		CAN_RMPUBF[15:8] / CAN_RFIFOMPFx[15:8]	CAN_RMPUBF[7:0] / CAN_RFIFOMPFx[7:0]
Mapping	RTR	IDE	保留	ID_STD[10:6] / ID_EXD[28:24]	ID_STD[5:0] / ID_EXD[23:18]	ID_EXD[17:16]	ID_EXD[15:8]	ID_EXD[7:0]

注：

1. 由can_mailbox_config函数的传参mdpara.rtr值配置MDES0[20]，由can_mailbox_config函数的传参mdpara.ide值配置MDES0[21]，由can_mailbox_config函数的传参mdpara.id值配置MDES1[28:18] / MDES1[28:0]。

图 2-2. Rx FIFO 的 ID 与 MASK 配置对应关系



注：

1. 当FDESx (x=4...107)为格式A时，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].remote_frame值配置FDESx[31]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].extended_frame值配置FDESx[30]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].id值配置FDESx[28:18] / FDESx[28:0]。
2. 当FDESx (x=4...107)为格式B时：
   - 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].remote_frame值配置FDESx[31]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].extended_frame值配置FDESx[30]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i].id值配置FDESx[29:19] / FDESx[29:16]；
   - 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+1].remote_frame值配置FDESx[15]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+1].extended_frame值配置FDESx[14]，由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+1].id值配置FDESx[13:3] / FDESx[13:0]。
3. 当FDESx (x=4...107)为格式C时：
   - 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i]. id值配置FDESx[31:24]；
   - 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+1]. id值配置FDESx[23:16]；
   - 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+2]. id值配置

FDESx[15:8];

- 由can_rx_fifo_filter_table_config函数的传参id_filter_table[i+3]. id值配置FDESx[7:0]。

2.2. 过滤数据配置

过滤数据的配置方法总结如下：

表 2-1. 过滤数据配置

接收配置	RPFQEN	CAN_RMPUBF(1)	CAN_RFIFOPUBF(2)	CAN_RFIFOMPFx(x = 0..31)(3)
禁能FIFO，即只有邮箱	0	作为过滤器	-	-
	1	-	-	作为过滤器
使能FIFO，即邮箱与FIFO都存在(4)	0	作为接收邮箱过滤器，见*表2-3. Rx FIFO标识符过滤表(RPFQEN = 0时)*表注1	作为接收FIFO过滤器，所有过滤寄存器值配置必须相同，见*表2-3. Rx FIFO标识符过滤表(RPFQEN = 0时)*表注2	
	1	-	作为剩余的FIFO标识符元素过滤器，见*表2-2. Rx FIFO标识符过滤表(RPFQEN = 1时)*表注3	作为接收邮箱和接收FIFO的过滤器，过滤的标识符对象见*表2-2. Rx FIFO标识符过滤表(RPFQEN = 1时)*表注1和注2

注：

1. 对应 can_init 函数中的 can_parameter_init->mb_public_filter 成员值；
2. 对应 can_rx_fifo_config 函数中的 can_fifo_para_init->fifo_public_filter 成员值。当 FIFO 使能且 RPFQEN 位为 0 时，can_rx_fifo_config 函数的 can_fifo_para_init->fifo_public_filter 成员值也将同时配置 CAN_RFIFOMPFx(x = 0...31) 寄存器为相同值。
3. 由 can_private_filter_config 函数按传参 index 来配置 CAN_RFIFOMPFx(x = 0...31)。
4. 当不使用 can_mailbox_config 函数配置接收邮箱，只使用 can_rx_fifo_filter_table_config 函数配置 Rx FIFO 时，则只使用 Rx FIFO 进行接收。

表 2-2. Rx FIFO 标识符过滤表(RPFQEN = 1 时)

RFFN[3:0]	Rx FIFO标识符过滤表元素数目	Rx FIFO占用的空间	可用的邮箱(1)	私有过滤的元素(2)	公共过滤的元素(3)
0000	8	邮箱0 - 7	邮箱8 - 31	元素0 - 7	无
0001	16	邮箱0 - 9	邮箱10 - 31	元素0 - 9	元素10 - 15
0010	24	邮箱0 - 11	邮箱12 - 31	元素0 - 11	元素12 - 23
0011	32	邮箱0 - 13	邮箱14 - 31	元素0 - 13	元素14 - 31
0100	40	邮箱0 - 15	邮箱16 - 31	元素0 - 15	元素16 - 39
0101	48	邮箱0 - 17	邮箱18 - 31	元素0 - 17	元素18 - 47
0110	56	邮箱0 - 19	邮箱20 - 31	元素0 - 19	元素20 - 55

RFFN[3:0]	Rx FIFO标识符过滤表元素数目	Rx FIFO占用的空间	可用的邮箱(1)	私有过滤的元素(2)	公共过滤的元素(3)
0111	64	邮箱0 - 21	邮箱22 - 31	元素0 - 21	元素22 - 63
1000	72	邮箱0 - 23	邮箱24 - 31	元素0 - 23	元素24 - 71
1001	80	邮箱0 - 25	邮箱26 - 31	元素0 - 25	元素26 - 79
1010	88	邮箱0 - 27	邮箱28 - 31	元素0 - 27	元素28 - 87
1011	96	邮箱0 - 29	邮箱30 - 31	元素0 - 29	元素30 - 95
1100	104	邮箱0 - 31	无	元素0 - 31	元素32 - 103
其他	104	邮箱0 - 31	无	元素0 - 31	元素32 - 103

注:

1. 由CAN_RFIFOMPFx(x = 0...31)寄存器配置过滤的邮箱;
2. 由CAN_RFIFOMPFx(x = 0...31)寄存器配置过滤的Rx FIFO标识符过滤表元素;
3. 由CAN_RFIFOPUBF寄存器配置过滤的Rx FIFO标识符过滤表元素。

表 2-3. Rx FIFO 标识符过滤表(RPFQEN = 0 时)

RFFN[3:0]	Rx FIFO标识符过滤表元素数目	Rx FIFO占用的空间	可用的邮箱(1)	公共与私有过滤的元素(2)
0000	8	邮箱0 - 7	邮箱8 - 31	元素0 - 7
0001	16	邮箱0 - 9	邮箱10 - 31	元素0 - 15
0010	24	邮箱0 - 11	邮箱12 - 31	元素0 - 23
0011	32	邮箱0 - 13	邮箱14 - 31	元素0 - 31
0100	40	邮箱0 - 15	邮箱16 - 31	元素0 - 39
0101	48	邮箱0 - 17	邮箱18 - 31	元素0 - 47
0110	56	邮箱0 - 19	邮箱20 - 31	元素0 - 55
0111	64	邮箱0 - 21	邮箱22 - 31	元素0 - 63
1000	72	邮箱0 - 23	邮箱24 - 31	元素0 - 71
1001	80	邮箱0 - 25	邮箱26 - 31	元素0 - 79
1010	88	邮箱0 - 27	邮箱28 - 31	元素0 - 87
1011	96	邮箱0 - 29	邮箱30 - 31	元素0 - 95
1100	104	邮箱0 - 31	无	元素0 - 103
其他	104	邮箱0 - 31	无	元素0 - 103

注:

1. 由CAN_RMPUBF寄存器配置过滤的邮箱;
2. 由相同配置值的CAN_RFIFOPUBF和CAN_RFIFOMPFx(x = 0...31)寄存器配置过滤的Rx FIFO标识符过滤表元素。

3. 过滤器使用举例

3.1. 禁能 FIFO，RPFQEN=0

使用邮箱进行接收，适用于接收不超过 32 组相同过滤规律的 ID 组(每个邮箱可以单独设置期

望 ID)，邮箱使用公共的过滤数据配置。当前过滤数据配置为接收所有帧，配置 1 个接收邮箱。参考配置如下。

表 3-1. CAN 模块参考配置

```
void can_config(void)
{
    can_parameter_struct can_parameter;

    /* initialize CAN register */
    can_deinit(CAN0);
    can_deinit(CAN1);
    /* initialize CAN */
    can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);

    /* initialize CAN parameters */
    can_parameter.internal_counter_source = CAN_TIMER_SOURCE_BIT_CLOCK;
    can_parameter.self_reception = DISABLE;
    can_parameter.mb_tx_order = CAN_TX_HIGH_PRIORITY_MB_FIRST;
    can_parameter.mb_tx_abort_enable = ENABLE;
    can_parameter.local_priority_enable = DISABLE;
    can_parameter.mb_rx_ide_rtr_type = CAN_IDE_RTR_FILTERED;
    can_parameter.mb_remote_frame = CAN_STORE_REMOTE_REQUEST_FRAME;
    can_parameter.rx_private_filter_queue_enable = DISABLE; // RPFQEN=0
    can_parameter.edge_filter_enable = DISABLE;

    can_parameter.protocol_exception_enable = DISABLE;
    can_parameter.rx_filter_order = CAN_RX_FILTER_ORDER_MAILBOX_FIRST;
    can_parameter.memory_size = CAN_MEMSIZE_32_UNIT;
    /* filter configuration */
    can_parameter.mb_public_filter = 0x0;     // when set filter value 0x0, means the IDE & ID bits of
the received frame are not concerned
    can_parameter.resync_jump_width = 1;
    can_parameter.prop_time_segment = 2;
    can_parameter.time_segment_1 = 4;
    can_parameter.time_segment_2 = 3;
    /* 125Kbps */
    can_parameter.prescaler = 80;

    /* initialize CAN */
    can_init(CAN0, &can_parameter);
    can_init(CAN1, &can_parameter);

    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_Message_IRQn, 0, 0);
```

```
    /* enable CAN MB0 interrupt */
    can_interrupt_enable(CAN0, CAN_INT_MB0);

    can_operation_mode_enter(CAN1, CAN_NORMAL_MODE);
    can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
}
```

表 3-2. 主函数参考配置

```
int main(void)
{
    /* configure systick */
    ... ...
    /* configure board */
    ... ...
    /* configure GPIO */
    ... ...
    /* initialize CAN and filter */
    can_config();

    can_struct_para_init(CAN_MDSC_STRUCT, &transmit_message);
    can_struct_para_init(CAN_MDSC_STRUCT, &receive_message);
    /* initialize transmit message */
    transmit_message.rtr = 0;
    transmit_message.ide = 0;
    transmit_message.code = CAN_MB_TX_STATUS_DATA;
    transmit_message.brs = 0;
    transmit_message.fdf = 0;
    transmit_message.prio = 0;
    transmit_message.data_bytes = 8;
    /* tx message content */
    transmit_message.data = (uint32_t *)(tx_data);
    transmit_message.id = 0x55;

    receive_message.rtr = 0;
    receive_message.ide = 0;
    receive_message.code = CAN_MB_RX_STATUS_EMPTY;
    /* rx mailbox */
    receive_message.id = 0x55;
    receive_message.data = (uint32_t *)(rx_data);
    can_mailbox_config(CAN0, 0, &receive_message);
    can_tx_state = RESET;

    while(1) {
```

```
        /* press WAKEUP key to trigger message transmission */
        ... ...
        /* wait for prior transmission finished */
        if((RESET == can_tx_state) || (SET == can_flag_get(CAN1, CAN_FLAG_MB1))){
            can_tx_state = SET;
            can_flag_clear(CAN1, CAN_FLAG_MB1);
            /* transmit message */
            can_mailbox_config(CAN1, 1, &transmit_message);
        }
        /* check the reception result */
    }
}
```

3.2. 禁能 FIFO，RPFQEN=1

使用邮箱进行接收，适用于接收不超过 32 组不同过滤规律的 ID 组(每个邮箱可以单独设置期望 ID)。当前使用 8 个接收邮箱，配置 4 个标准帧，4 个扩展帧。参考配置如下。

表 3-3. CAN 模块参考配置

```
void can_config(void)
{
    can_parameter_struct can_parameter;

    /* initialize CAN register */
    can_deinit(CAN0);
    can_deinit(CAN1);
    /* initialize CAN */
    can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);

    /* initialize CAN parameters */
    can_parameter.internal_counter_source = CAN_TIMER_SOURCE_BIT_CLOCK;
    can_parameter.self_reception = ENABLE;
    can_parameter.mb_tx_order = CAN_TX_HIGH_PRIORITY_MB_FIRST;
    can_parameter.mb_tx_abort_enable = ENABLE;
    can_parameter.local_priority_enable = DISABLE;
    can_parameter.mb_rx_ide_rtr_type = CAN_IDE_RTR_FILTERED;
    can_parameter.mb_remote_frame = CAN_STORE_REMOTE_REQUEST_FRAME;
    can_parameter.rx_private_filter_queue_enable = ENABLE;    // RPFQEN=1
    can_parameter.rx_filter_order = CAN_RX_FILTER_ORDER_MAILBOX_FIRST;
    can_parameter.memory_size = CAN_MEMSIZE_32_UNIT;
    /* filter configuration */
    can_parameter.mb_public_filter = 0x0; // this is not used for filtering, can_private_filter_config()
is used for filter configuration
    can_parameter.resync_jump_width = 1;
```

```
    can_parameter.prop_time_segment = 2;
    can_parameter.time_segment_1 = 4;
    can_parameter.time_segment_2 = 3;
    /* 250Kbps */
    can_parameter.prescaler = 20;

    /* initialize CAN */
    can_init(CAN0, &can_parameter);
    can_init(CAN1, &can_parameter);

    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_Message_IRQn, 0, 0);

    /* enable CAN MB0 interrupt */
    can_interrupt_enable(CAN0, CAN_INT_MB0);
    can_interrupt_enable(CAN0, CAN_INT_MB1);
    can_interrupt_enable(CAN0, CAN_INT_MB2);
    can_interrupt_enable(CAN0, CAN_INT_MB3);
    can_interrupt_enable(CAN0, CAN_INT_MB4);
    can_interrupt_enable(CAN0, CAN_INT_MB5);
    can_interrupt_enable(CAN0, CAN_INT_MB6);
    can_interrupt_enable(CAN0, CAN_INT_MB7);

    /* concern filtering MB0~MB7 IDE bit, and ID fields */
    //when set filter value 0x7FFFFFFF, means the IDE & ID bits of the received frame should totally
match target descriptor (configured by can_mailbox_config(CAN0, i, &receive_message[i]))
    can_private_filter_config(CAN0, 0, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 1, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 2, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 3, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 4, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 5, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 6, 0x7FFFFFFF);
    can_private_filter_config(CAN0, 7, 0x7FFFFFFF);

    can_operation_mode_enter(CAN1, CAN_NORMAL_MODE);
    can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
}
```

表 3-4. 主函数参考配置

```
int main(void)
{
    /* configure systick */
    ... ...
```

```
/* configure board */
... ...
/* configure GPIO */
... ...
/* initialize CAN and filter */
can_config();

/* standard frame */
for(i=0; i<4; i++){
    can_struct_para_init(CAN_MDSC_STRUCT, &transmit_message[i]);
    can_struct_para_init(CAN_MDSC_STRUCT, &receive_message[i]);
    /* initialize transmit message */
    transmit_message[i].rtr = 0;
    transmit_message[i].ide = 0;
    transmit_message[i].code = CAN_MB_TX_STATUS_DATA;
    transmit_message[i].brs = 0;
    transmit_message[i].fdf = 0;
    transmit_message[i].prio = 0;
    transmit_message[i].data_bytes = 8;
    /* tx message content */
    transmit_message[i].data = (uint32_t *)(tx_data);
    transmit_message[i].id = 0x55+i;

    receive_message[i].rtr = 0;
    receive_message[i].ide = 0;
    receive_message[i].code = CAN_MB_RX_STATUS_EMPTY;
    /* rx mailbox */
    receive_message[i].id = 0x55+i;
    receive_message[i].data = (uint32_t *)(rx_data);
    can_mailbox_config(CAN0, i, &receive_message[i]);
}

/* extended frame */
for(i=4; i<8; i++){
    can_struct_para_init(CAN_MDSC_STRUCT, &transmit_message[i]);
    can_struct_para_init(CAN_MDSC_STRUCT, &receive_message[i]);
    /* initialize transmit message */
    transmit_message[i].rtr = 0;
    transmit_message[i].ide = 1;
    transmit_message[i].code = CAN_MB_TX_STATUS_DATA;
    transmit_message[i].brs = 0;
    transmit_message[i].fdf = 0;
    transmit_message[i].prio = 0;
    transmit_message[i].data_bytes = 8;
```

```
        /* tx message content */
        transmit_message[i].data = (uint32_t *)(tx_data);
        transmit_message[i].id = 0x10000055+i;

        receive_message[i].rtr = 0;
        receive_message[i].ide = 1;
        receive_message[i].code = CAN_MB_RX_STATUS_EMPTY;
        /* rx mailbox */
        receive_message[i].id = 0x10000055+i;
        receive_message[i].data = (uint32_t *)(rx_data);
        can_mailbox_config(CAN0, i, &receive_message[i]);
    }

    i = 0;
    while(1) {
        /* press WAKEUP key to trigger message transmission */
        ... ...
        /* transmit message */
        can_mailbox_config(CAN1, 1, &transmit_message[i]);
        i++;
        if(i > 7){
            i = 0;
        }
        ... ...
        /* check the reception result */
    }
}
```

3.3. 使能 FIFO，RPFQEN=0

接收邮箱和 Rx FIFO 均可接收，适用于需要接收不超过 4 组不同过滤规律，每组不超过 104 中相同过滤规律的 ID 组，邮箱和 FIFO 标识符过滤表元素可以单独设置期望 ID。

例子中接收 4 组全过滤 ID，4 组部分过滤 ID。配置为格式 A，参考*表 2-3. Rx FIFO 标识符过滤表(RPFQEN = 0 时)*，RFFN[3:0]在格式 A 时配置为 0x0 即可，支持 8 个完整的标准标识符或完整的扩展标识符。

此时第 0-3 个 ID 组(0b1000000x，0b1000001x，0b1000010x，0b1000011x)由 can_parameter_init->mb_public_filter 成员值配置过滤数据，第 4-7 个 ID 组(由第 0-3 个元素配置的 0x55-0x58)由 can_fifo_para_init->fifo_public_filter 成员值配置过滤数据。对于第 4-7 个标识符过滤表元素，用户需要使用 can_rx_fifo_filter_table_config()函数配置额外的 4 个 ID，以避免接收到预期意外的数据。参考配置如下。

例子验证结果能正确接收到 ID 为 0x55-0x58，0x80-0x87 的数据帧，过滤掉其他 ID 的数据帧。

表 3-5. CAN 模块参考配置

```
void can_config(void)
{
    can_parameter_struct can_parameter;
    can_fifo_parameter_struct can_fifo_parameter;

    uint8_t i = 0;

    /* initialize CAN register */
    can_deinit(CAN0);
    can_deinit(CAN1);
    /* initialize CAN */
    can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);

    /* initialize CAN parameters */
    can_parameter.internal_counter_source = CAN_TIMER_SOURCE_BIT_CLOCK;
    can_parameter.self_reception = ENABLE;
    can_parameter.mb_tx_order = CAN_TX_HIGH_PRIORITY_MB_FIRST;
    can_parameter.mb_tx_abort_enable = ENABLE;
    can_parameter.local_priority_enable = DISABLE;
    can_parameter.mb_rx_ide_rtr_type = CAN_IDE_RTR_FILTERED;
    can_parameter.mb_remote_frame = CAN_STORE_REMOTE_REQUEST_FRAME;
    can_parameter.rx_private_filter_queue_enable = DISABLE; // RPFQEN=0
    can_parameter.rx_filter_order = CAN_RX_FILTER_ORDER_MAILBOX_FIRST;
    can_parameter.memory_size = CAN_MEMSIZE_32_UNIT;
    /* filter configuration */
    can_parameter.mb_public_filter = 0xFFFBFFFF; // used for 0-3nd IDs filtering
    can_parameter.resync_jump_width = 1;
    can_parameter.prop_time_segment = 2;
    can_parameter.time_segment_1 = 4;
    can_parameter.time_segment_2 = 3;
    /* 250Kbps,70% */
    can_parameter.prescaler = 20;

    /* initialize CAN */
    can_init(CAN0, &can_parameter);
    can_init(CAN1, &can_parameter);

    /* Rxfifo configuration */
    can_fifo_parameter.dma_enable = DISABLE;
    can_fifo_parameter.filter_format_and_number = CAN_RXFIFO_FILTER_A_NUM_8;
    can_fifo_parameter.fifo_public_filter = 0xFFFFFFFF; // used for 4-7nd ID and extra 4 IDs filtering
    can_rx_fifo_config(CAN0, &can_fifo_parameter);
```

```
    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_Message_IRQn, 0, 0);

    /* enable CAN RxFIFO not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INT_MB5);
    /* enable Rx interrupt */
    can_interrupt_enable(CAN0, CAN_INT_MB8);
    can_interrupt_enable(CAN0, CAN_INT_MB9);
    can_interrupt_enable(CAN0, CAN_INT_MB10);
    can_interrupt_enable(CAN0, CAN_INT_MB11);

    can_operation_mode_enter(CAN1, CAN_NORMAL_MODE);
    can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
}
```

表 3-6. 主函数参考配置

```
int main(void)
{
    /* configure systick */
    ... ...
    /* configure board */
    ... ...
    /* configure GPIO */
    ... ...
    /* initialize CAN and filter */
    can_config();

    /* the expected 0-3 IDs configuration */
    for(i=0; i<4; i++){
        fifo_element[i].extended_frame = CAN_STANDARD_FRAME_ACCEPTED;
        fifo_element[i].remote_frame = CAN_DATA_FRAME_ACCEPTED;
        fifo_element[i].id = 0x55 + i;
    }
    /* the rest unused IDs configuration */
    for(i=4; i<8; i++){
        fifo_element[i].extended_frame = CAN_STANDARD_FRAME_ACCEPTED;
        fifo_element[i].remote_frame = CAN_DATA_FRAME_ACCEPTED;
        fifo_element[i].id = 0x55;
    }
    can_rx_fifo_filter_table_config(CAN0, fifo_element);
    can_rx_fifo_clear(CAN0);

    /* the expected 4-7 IDs configuration */
    for(i=0; i<4; i++){
```

```
        can_struct_para_init(CAN_MDSC_STRUCT, &receive_message[i]);
        receive_message[i].rtr = 0;
        receive_message[i].ide = 0;
        receive_message[i].code = CAN_MB_RX_STATUS_EMPTY;
        /* rx mailbox */
        receive_message[i].id = 0x80+i*2;
        receive_message[i].data = (uint32_t *)(rx_data);
        can_mailbox_config(CAN0, i+8, &receive_message[i]);
    }
    /* standard frame */
    /* in this demo, we test 16 different IDs, the 0-3 ID (due to perfect filtering) frames will be received by CAN0 fifo, the 4-11 ID (due to partial filtering) frames will be received by mailbox, others will be filtered */
    for(i=0; i<16; i++){
        can_struct_para_init(CAN_MDSC_STRUCT, &transmit_message[i]);
        /* initialize transmit message */
        transmit_message[i].rtr = 0;
        transmit_message[i].ide = 0;
        transmit_message[i].code = CAN_MB_TX_STATUS_DATA;
        transmit_message[i].brs = 0;
        transmit_message[i].fdf = 0;
        transmit_message[i].prio = 0;
        transmit_message[i].data_bytes = 8;
        /* tx message content */
        transmit_message[i].data = (uint32_t *)(tx_data);
        if(i < 4){
            transmit_message[i].id = 0x55+i;
        }else{
            transmit_message[i].id = 0x80+i-4;
        }
    }

    i = 0;
    while(1) {
        /* press WAKEUP key to trigger message transmission */
        ... ...
        /* transmit message */
        can_mailbox_config(CAN1, 1, &transmit_message[i]);
        i++;
        if(i>=16){
            i=0;
        }
        ... ...
        /* check the reception result */
```

```
    }
}
```

3.4. 使能 FIFO，RPFQEN=1

接收邮箱和 Rx FIFO 均可接收，适用于需要接收不超过 128 组不同过滤规律的 ID 组+不超过 288 组相同过滤规律的 ID 组，邮箱和 FIFO 标识符过滤表元素可以单独设置期望 ID。

例子中接收 52 个不同 ID 组，独立配置每个 ID 的过滤数据。配置为格式 B，参考*表 2-2. Rx FIFO 标识符过滤表(RPFQEN = 1 时)*，RFFN[3:0]在格式 B 时配置为 0x9 即可，支持 64 个完整的标准标识符或扩展标识符中的 14 个 bit。

此时第 0-25 元素共 52 个 ID(第 0-51 个 ID)由 can_private_filter_config()函数配置过滤数据，剩余第 26-79 元素共 108 个 ID(第 52-159 个 ID)由 can_fifo_para_init->fifo_public_filter 成员值配置过滤数据，用户需要使用 can_rx_fifo_filter_table_config()函数配置额外的 108 个 ID，以避免接收到预期意外的数据。参考配置如下。

例子验证结果能正确接收到 ID 为 0x55-0x88 的数据帧，并过滤掉其他 ID 的数据帧。

表 3-7. CAN 模块参考配置

```
void can_config(void)
{
    can_parameter_struct can_parameter;
    can_fifo_parameter_struct can_fifo_parameter;

    uint8_t i = 0;

    /* initialize CAN register */
    can_deinit(CAN0);
    can_deinit(CAN1);
    /* initialize CAN */
    can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);

    /* initialize CAN parameters */
    can_parameter.internal_counter_source = CAN_TIMER_SOURCE_BIT_CLOCK;
    can_parameter.self_reception = ENABLE;
    can_parameter.mb_tx_order = CAN_TX_HIGH_PRIORITY_MB_FIRST;
    can_parameter.mb_tx_abort_enable = ENABLE;
    can_parameter.local_priority_enable = DISABLE;
    can_parameter.mb_rx_ide_rtr_type = CAN_IDE_RTR_FILTERED;
    can_parameter.mb_remote_frame = CAN_STORE_REMOTE_REQUEST_FRAME;
    can_parameter.rx_private_filter_queue_enable = ENABLE; // RPFQEN=1
    can_parameter.rx_filter_order = CAN_RX_FILTER_ORDER_MAILBOX_FIRST;
    can_parameter.memory_size = CAN_MEMSIZE_32_UNIT;
    /* filter configuration */
```

```
    can_parameter.mb_public_filter = 0x0; // this is not used for filtering, can_fifo_parameter
->fifo_public_filter and can_private_filter_config() is used for filter configuration
    can_parameter.resync_jump_width = 1;
    can_parameter.prop_time_segment = 2;
    can_parameter.time_segment_1 = 4;
    can_parameter.time_segment_2 = 3;
    /* 250Kbps,70% */
    can_parameter.prescaler = 20;

    /* initialize CAN */
    can_init(CAN0, &can_parameter);
    can_init(CAN1, &can_parameter);

    /* Rxfifo configuration */
    can_fifo_parameter.dma_enable = ENABLE;
    can_fifo_parameter.filter_format_and_number = CAN_RXFIFO_FILTER_B_NUM_64;
    can_fifo_parameter.fifo_public_filter = 0xFFFFFFFF; // used for 52-159nd IDs filtering
    can_rx_fifo_config(CAN0, &can_fifo_parameter);

    /* used for 0-51nd IDs filtering */
    for(i=0; i<=25; i++){
        can_private_filter_config(CAN0, i, 0xFFFFFFFF);
    }

    can_operation_mode_enter(CAN1, CAN_NORMAL_MODE);
    can_operation_mode_enter(CAN0, CAN_NORMAL_MODE);
}
```

表 3-8. 主函数参考配置

```
int main(void)
{
    /* configure systick */
    ... ...
    /* configure board */
    ... ...
    /* configure GPIO */
    ... ...
    /* configure DMA */
    ... ...
    /* initialize CAN and filter */
    can_config();

    /* the expected 52 IDs configuration */
    for(i=0; i<52; i++){
```

```
        fifo_element[i].extended_frame = CAN_STANDARD_FRAME_ACCEPTED;
        fifo_element[i].remote_frame = CAN_DATA_FRAME_ACCEPTED;
        fifo_element[i].id = 0x55 + i;
    }
    /* the rest unused IDs configuration */
    for(i=52; i<160; i++){
        fifo_element[i].extended_frame = CAN_STANDARD_FRAME_ACCEPTED;
        fifo_element[i].remote_frame = CAN_DATA_FRAME_ACCEPTED;
        fifo_element[i].id = 0x55;
    }
    can_rx_fifo_filter_table_config(CAN0, fifo_element);
    can_rx_fifo_clear(CAN0);

    /* standard frame */
    /* in this demo, we test 64 different IDs, the 0-51 ID frames will be received by CAN0 fifo, the other ID frames will be filtered */
    for(i=0; i<64; i++){
        can_struct_para_init(CAN_MDSC_STRUCT, &transmit_message[i]);
        /* initialize transmit message */
        transmit_message[i].rtr = 0;
        transmit_message[i].ide = 0;
        transmit_message[i].code = CAN_MB_TX_STATUS_DATA;
        transmit_message[i].brs = 0;
        transmit_message[i].fdf = 0;
        transmit_message[i].prio = 0;
        transmit_message[i].data_bytes = 8;
        /* tx message content */
        transmit_message[i].data = (uint32_t *)(tx_data);
        transmit_message[i].id = 0x55+i;
    }

    i = 0;
    while(1) {
        /* press WAKEUP key to trigger message transmission */
        ... ...
        /* transmit message */
        can_mailbox_config(CAN1, 1, &transmit_message[i]);
        i++;
        if(i>=64){
            i=0;
        }
        ... ...
        /* check the reception result */
    }
```

```
}
```

4. 版本历史

表 4-1. 版本历史

版本号	说明	日期
1.0	首次发布	2026 年 08 月 12 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.