GigaDevice Semiconductor Inc.

GD32H73x_75x CAN Filter User Guide

Application Note
AN284

Revision 1.0

(Aug. 2026)

Table of Contents

List of Figures

List of Tables

1. Preface

The GD32H73x_75x CAN module features powerful filtering capabilities. The module's reception section can be configured to use only Rx mailboxes or to utilize both Rx mailboxes and Rx FIFOs. The configuration methods for expected IDs in Rx mailboxes and Rx FIFOs differ, with the expected ID settings for Rx FIFOs being more flexible. Additionally, the CAN module provides private filter registers. These features make the filtering application for the CAN module highly complex. This document will introduce all configuration scenarios for the CAN module filters, the usage of related firmware library functions, and usage examples. The applicable products are as shown in *Table 1-1. Applicable product*.

Table 1-1. Applicable product

Product series	Model
GD32H73x	GD32H737 series
GD32H75x	GD32H757, GD32H759 series

2. Filter usage introduction

The CAN bus controller integrates a flexibly configurable mailbox system for the transmission and reception of CAN frames. The mailbox system consists of a set of mailboxes used to store control data, timestamps, message identifiers, and message data, supporting up to 32 mailboxes. Each CAN controller contains a 512-byte RAM space for the configuration of mailbox or FIFO descriptors.

When CAN enables FIFO reception, the RAM space occupied by mailboxes will be used for receiving FIFO descriptors. The reception FIFO supports identifier filtering functionality, with a maximum of 104 extended identifier filters, 208 standard identifier filters, or 416 partial 8-bit identifier filters. Up to 32 identifier filter table elements can be configured through the reception FIFO/mailbox private filter registers.

When Rx FIFO is disabled:

- If the RPFQEN bit in the CAN_CTL0 register is set to 0, the CAN_RMPUBF register is used to configure the filter data for all reception mailboxes.
- If the RPFQEN bit in the CAN_CTL0 register is set to 1, the CAN_RFIFOMPFx (x = 0..31) register is used to individually configure the filter data for reception mailboxes.

When Rx FIFO is enabled:

- If the RPFQEN bit in the CAN_CTL0 register is 0, the CAN_RMPUBF register is used to configure the filter data for all receive mailboxes, the CAN_RFIFOPUBF and CAN_RFIFOMPFx (x = 0..31) registers are used to configure all Rx FIFO identifier filter table elements, and the values of all these registers must be configured identically.
- If the RPFQEN bit in the CAN_CTL0 register is 1, the CAN_RFIFOMPFx (x=0..31) registers are used to configure the Rx FIFO identifier filter table elements set by the RFFN[3:0] bit field in the CAN_CTL2 register, as well as the receive mailboxes (since the receive mailbox descriptors and Rx FIFO descriptors cannot simultaneously occupy the same RAM area, a separate set of registers is used to independently control the filter data configuration), and the remaining Rx FIFO identifier filter table elements are configured using the CAN_RFIFOPUBF register.

2.1. Filter method

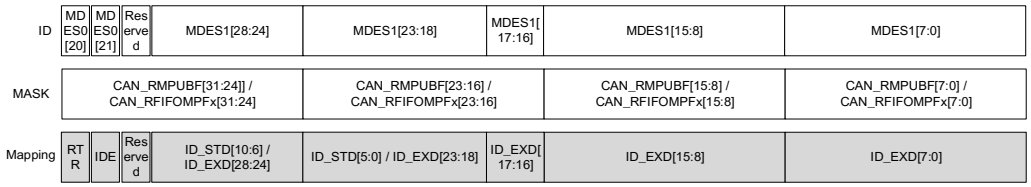
The GD32H73x_75x CAN module filter adopts the ID MASK method for filtering.

The registers used to configure filter data mainly include: CAN_RMPUBF, CAN_RFIFOMPFx (x = 0..31), and CAN_RFIFOPUBF. The first register is dedicated for Rx mailbox filtering, the last register is dedicated for Rx FIFO filtering, and the middle register is applicable for both Rx mailbox and Rx FIFO filtering.

The ID and MASK configuration relationship for the Rx mailbox is shown in ***Figure 2-1. ID and MASK configuration relationship for Rx mailbox***. The ID and MASK configuration

relationship for the Rx FIFO is shown in ***Figure 2-2. ID and MASK configuration relationship for Rx FIFO***.

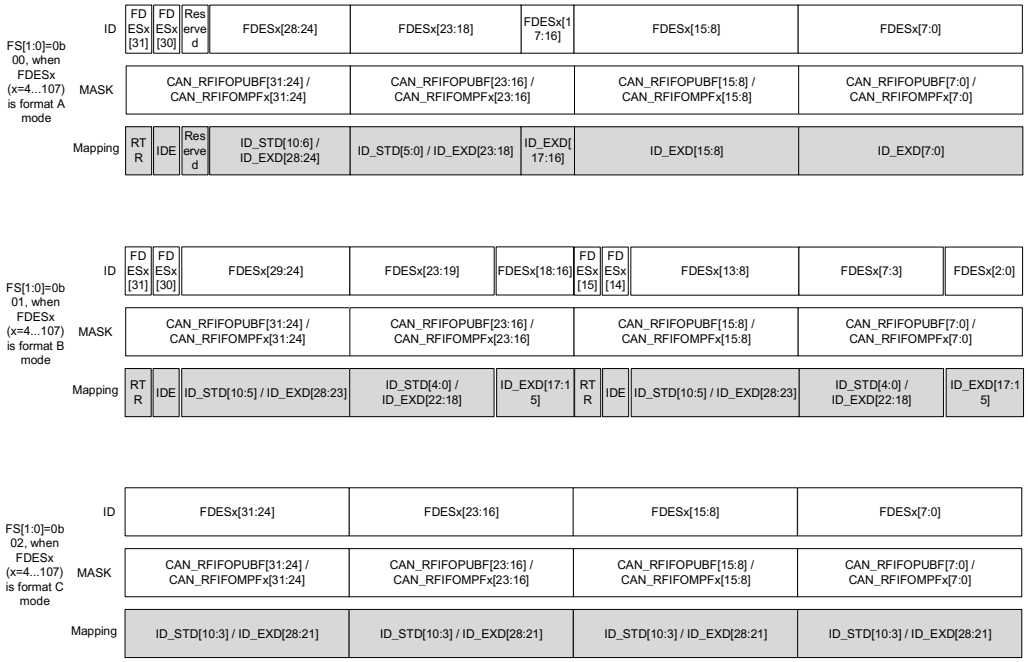
Figure 2-1. ID and MASK configuration relationship for Rx mailbox(1)



Note:

1. The value of mdpara.rtr passed by the can_mailbox_config function is used to configure MDES0[20], the value of mdpara.ide passed by the can_mailbox_config function is used to configure MDES0[21], and the value of mdpara.id passed by the can_mailbox_config function is used to configure MDES1[28:18] / MDES1[28:0].

Figure 2-2. ID and MASK configuration relationship for Rx FIFO



Note:

1. When FDESx (x=4...107) is in format A, the value of id_filter_table[i].remote_frame passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[31], the value of id_filter_table[i].extended_frame passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[30], and the value of id_filter_table[i].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[28:18] / FDESx[28:0].

2. When FDESx (x=4...107) is in format B:

■ The value of id_filter_table[i].remote_frame passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[31], the value of id_filter_table[i].extended_frame

passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[30], and the value of id_filter_table[i].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[29:19] / FDESx[29:16].

- The value of id_filter_table[i+1].remote_frame passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[15], the value of id_filter_table[i+1].extended_frame passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[14], and the value of id_filter_table[i+1].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[13:3] / FDESx[13:0].

3. When FDESx (x=4...107) is in format C:

- The value of id_filter_table[i].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[31:24].
- The value of id_filter_table[i+1].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[23:16].
- The value of id_filter_table[i+2].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[15:8].
- The value of id_filter_table[i+3].id passed by the can_rx_fifo_filter_table_config function is used to configure FDESx[7:0].

2.2. Filter data configuration

The filter data configuration method is summarized as follows:

Table 2-1. Filter data configuration

Rx configuration	RPFQEN	CAN_RMPUBF(1)	CAN_RFIFOPUBF(2)	CAN_RFIFOMPFx(x = 0..31) (3)
Disable FIFO, only mailbox	**0**	As filter.	-	-
	1	-	-	As filter.
Enable FIFO, both mailbox and FIFO exist(4)	**0**	As a Rx mailbox filter, see ***Table 2-3. Rx FIFO identifier filter table (when RPFQEN = 0)***, Note 1.	As a Rx FIFO filter, all filter register values must be identically configured, see ***Table 2-3. Rx FIFO identifier filter table (when RPFQEN = 0)***, Note 2.	
	1	-	As a filter for remaining FIFO identifier elements, see ***Table 2-2. Rx FIFO identifier filter table (when RPFQEN = 1)***, Note 3.	As a filter for the Rx mailbox and Rx FIFO, the filtered identifier target is shown in ***Table 2-2. Rx FIFO identifier filter table (when RPFQEN = 1)***, see

				Note 1 and 2.

Note:

1. Corresponds to the mb_public_filter member value in the can_parameter_init -> can_init function.
2. Corresponds to the fifo_public_filter member value in the can_fifo_para_init -> can_rx_fifo_config function. When FIFO is enabled and the RPFQEN bit is 0, the fifo_public_filter member value in the can_fifo_para_init -> can_rx_fifo_config function will also configure CAN_RFIFOMPFx (x = 0...31) registers to the same value.
3. CAN_RFIFOMPFx (x = 0...31) are configured by the can_private_filter_config function based on the passed index.
4. When the can_mailbox_config function is not used to configure the Rx mailbox, and only the can_rx_fifo_filter_table_config function is used to configure the Rx FIFO, then only the Rx FIFO is used for reception.

Table 2-2. Rx FIFO identifier filter table (when RPFQEN = 1)

RFFN[3:0]	Number of Rx FIFO identifier filter table elements	Space occupied by Rx FIFO	Available mailbox (1)	Private filter elements (2)	Public filter elements (3)
0000	8	Mailbox 0 - 7	Mailbox 8 - 31	Elements 0 - 7	None
0001	16	Mailbox 0 - 9	Mailbox 10 - 31	Elements 0 - 9	Elements 10 - 15
0010	24	Mailbox 0 - 11	Mailbox 12 - 31	Element 0 - 11	Element 12 - 23
0011	32	Mailbox 0 - 13	Mailbox 14 - 31	Element 0 - 13	Element 14 - 31
0100	40	Mailbox 0 - 15	Mailbox 16 - 31	Elements 0 - 15	Element 16 - 39
0101	48	Mailbox 0 - 17	Mailbox 18 - 31	Element 0 - 17	Element 18 - 47
0110	56	Mailbox 0 - 19	Mailbox 20 - 31	Element 0 - 19	Element 20 - 55
0111	64	Mailbox 0 - 21	Mailbox 22 - 31	Element 0 - 21	Element 22 - 63
1000	72	Mailbox 0 - 23	Mailbox 24 - 31	Elements 0 - 23	Element 24 - 71
1001	80	Mailbox 0 - 25	Mailbox 26 - 31	Element 0 - 25	Element 26 - 79
1010	88	Mailbox 0 - 27	Mailbox 28 - 31	Element 0 - 27	Element 28 - 87
1011	96	Mailbox 0 - 29	Mailbox 30 - 31	Element 0 - 29	Element 30 - 95
1100	104	Mailbox 0 - 31	None	Elements 0 - 31	Element 32 -

RFFN[3:0]	Number of Rx FIFO identifier filter table elements	Space occupied by Rx FIFO	Available mailbox (1)	Private filter elements (2)	Public filter elements (3)
					103
Others	104	Mailbox 0 - 31	None	Elements 0 - 31	Element 32 - 103

Note:

1. Mailboxes filtered by the CAN_RFIFOMPFx (x = 0...31) registers.
2. Rx FIFO identifier filter table elements filtered by the CAN_RFIFOMPFx (x = 0...31) registers.
3. Rx FIFO identifier filter table elements filtered by the CAN_RFIFOPUBF register.

Table 2-3. Rx FIFO identifier filter table (when RPFQEN = 0)

RFFN[3:0]	Number of Rx FIFO identifier filter table elements	Space occupied by Rx FIFO	Available mailbox (1)	Elements filtered by public and private filters (2)
0000	8	Mailbox 0 - 7	Mailbox 8 - 31	Elements 0 - 7
0001	16	Mailbox 0 - 9	Mailbox 10 - 31	Elements 0 - 15
0010	24	Mailbox 0 - 11	Mailbox 12 - 31	Elements 0 - 23
0011	32	Mailbox 0 - 13	Mailbox 14 - 31	Elements 0 - 31
0100	40	Mailbox 0 - 15	Mailbox 16 - 31	Element 0 - 39
0101	48	Mailbox 0 - 17	Mailbox 18 - 31	Element 0 - 47
0110	56	Mailbox 0 - 19	Mailbox 20 - 31	Element 0 - 55
0111	64	Mailbox 0 - 21	Mailbox 22 - 31	Element 0 - 63
1000	72	Mailbox 0 - 23	Mailbox 24 - 31	Element 0 - 71
1001	80	Mailbox 0 - 25	Mailbox 26 - 31	Element 0 - 79
1010	88	Mailbox 0 - 27	Mailbox 28 - 31	Element 0 - 87
1011	96	Mailbox 0 - 29	Mailbox 30 - 31	Element 0 - 95
1100	104	Mailbox 0 - 31	None	Element 0 - 103
Others	104	Mailbox 0 - 31	None	Element 0 - 103

Note:

1. Mailbox configured by the CAN_RMPUBF register.
2. Rx FIFO identifier filter table elements configured by the CAN_RFIFOPUBF and CAN_RFIFOMPFx (x = 0...31) registers with the same configuration value.

3. Example of filter usage

3.1. Disable FIFO, RPFQEN=0

Use mailbox for reception, suitable for receiving up to 32 groups of IDs with the same filtering data (independently configure the desired ID for each mailbox). The mailboxes use shared filtering data configuration. Currently, filtering data is configured as accept all frames, and one reception mailbox is configured. Refer to the configuration below.

Table 3-1. CAN module configuration example

```
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);
}
```

Table 3-2. Main function configuration example

```
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. Disable FIFO, RPFQEN=1

Use mailbox for reception, suitable for receiving up to 32 groups of IDs with different filtering data (independently configure the desired ID for each mailbox). Currently, 8 reception mailboxes are used, configured with 4 standard frames and 4 extended frames. Refer to the configuration below.

Table 3-3. CAN module configuration example

```
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);
}
```

Table 3-4. Main function configuration example

```
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. Enable FIFO, RPFQEN=0

Both reception mailboxes and Rx FIFO can be used for reception, suitable for receiving up to 4 groups of IDs with different filtering data, with each group containing up to 104 IDs with the same filtering data. Independently configure the desired ID for mailboxes and FIFO identifier filter table elements.

In the example, 4 groups of fully filtered IDs and 4 groups of partially filtered IDs are received. Configured in format A, refer to ***Table 2-3. Rx FIFO identifier filter table (when RPFQEN = 0)***, RFFN[3:0] is configured as 0x0 in format A, supporting 8 complete standard identifiers or complete extended identifiers.

In the example, ID groups 0-3 (0b1000000x, 0b1000001x, 0b1000010x, 0b1000011x) are configuring the filter data through the value of the `mb_public_filter` member in `can_parameter_init`. ID groups 4-7 (0x55-0x58 configured in elements 0-3) are configuring the filter data through the value of the `fifo_public_filter` member in `can_fifo_para_init`. For the filter table elements 4-7, users need to use the `can_rx_fifo_filter_table_config()` function to configure an additional 4 IDs to avoid receiving unexpected data. Refer to the configuration below.

The example verifies that data frames with IDs 0x55-0x58 and 0x80-0x87 can be correctly received, while data frames with other IDs are filtered out.

Table 3-5. CAN module configuration example

```
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);
}
```

Table 3-6. Main function configuration example

```
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. Enable FIFO, RPFQEN=1

Both the receive mailbox and Rx FIFO are capable of receiving, suitable for scenarios requiring reception of up to 128 ID groups of different filtering data + up to 288 ID groups of identical filtering data. Mailbox and FIFO identifier filter table elements can be individually set with the desired IDs.

In the example, 52 different ID groups are configured for reception, and filtering data for each ID is configured independently. Configured as Format B, refer to ***Table 2-2. Rx FIFO identifier filter table (when RPFQEN = 1)***, RFFN[3:0] should be configured as 0x9 in Format B, supporting 64 complete standard identifiers or 14 bits of extended identifiers.

In the example, a total of 52 IDs which configured in elements 0-25 (IDs 0-51) are configuring the filter data using the `can_private_filter_config()` function. The remaining 108 IDs which configured in elements 26-79 (IDs 52-159) are configuring the filter data through the value of the `fifo_public_filter` member in `can_fifo_para_init`. Users need to use the `can_rx_fifo_filter_table_config()` function to configure an additional 108 IDs to avoid receiving unexpected data. Refer to the configuration below.

The example verifies that data frames with IDs 0x55-0x88 can be correctly received, while data frames with other IDs are filtered out.

Table 3-7. CAN module configuration example

```
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);
}
```

Table 3-8. Main function configuration example

```
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. Revision history

Table 4-1. Revision history

Revision No.	Description	Date
1.0	Initial release	Aug. 12, 2026

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.