RT-Thread上的CAN驱动和应用

主要内容

1 CAN总线的帧格式介绍

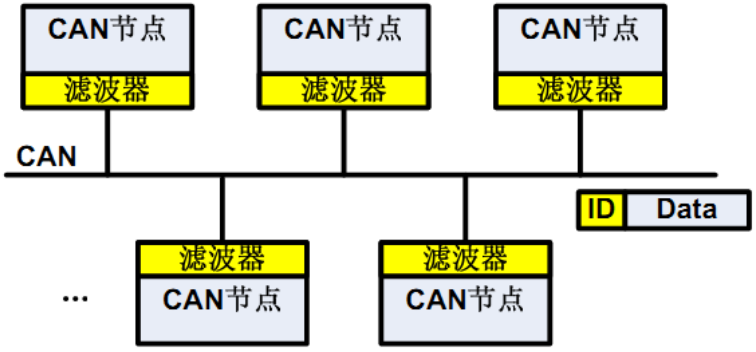
2 RT-Thread上的CAN驱动编写

3 CAN数据处理线程编写

一、CAN总线的帧格式介绍

1 CAN总线的介绍

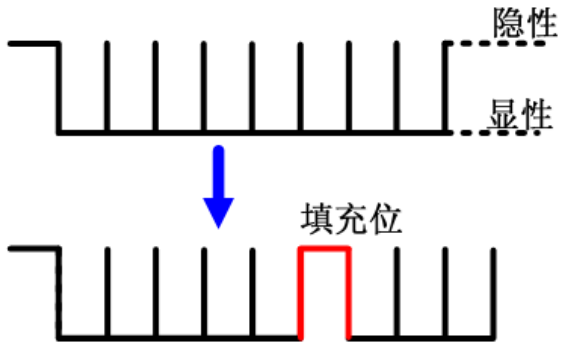
- CAN的特性
  - 多主站结构，各节点平等，优先权由报文ID确定
  - 每个报文的内容通过标识符识别，标识符在网络中是唯一的
    - 标识符描述了数据的含义
    - 某些特定的应用对标识符的分配进行了标准化
  - 根据需要可进行相关性报文过滤



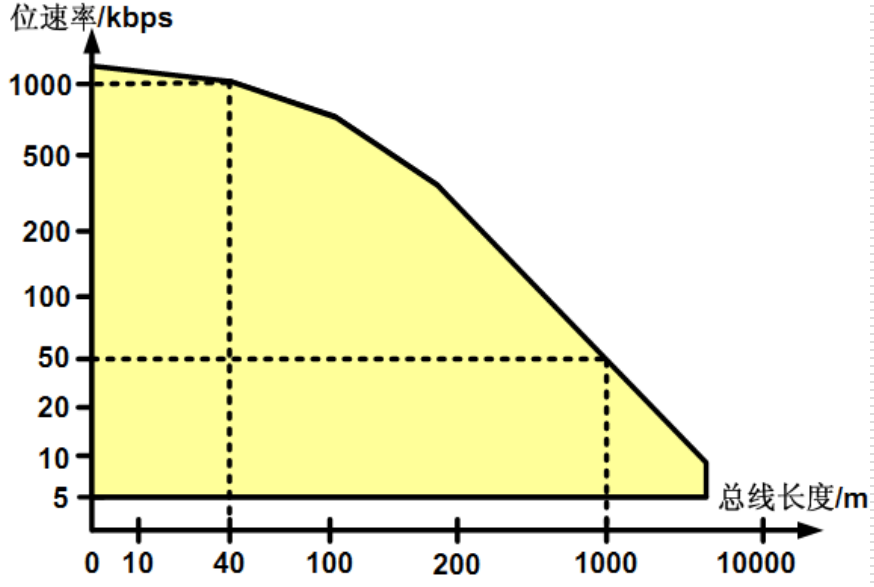
1 CAN总线的介绍

- CAN的特性
  - 使用双绞线作为总线介质，传输速率可达1Mbps，总线长度<=40米
  - 采用NRZ和位填充的位编码方式

NRZ和位填充



位速率与总线长度的关系



1 CAN总线的介绍

- CAN标准
  - CAN2.0版本
    - 2.0A—将29位ID视为错误
    - 2.0B被动—忽略29位ID的报文
    - 2.0B主动—可处理11位和29位两种ID的报文

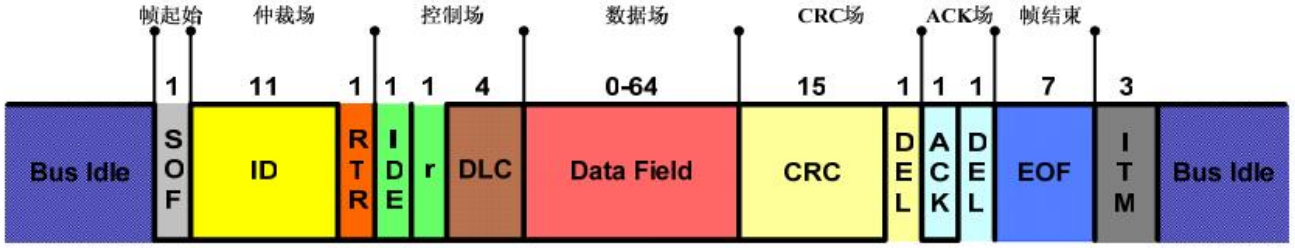
	11位ID数据帧	29位ID数据帧
CAN 2.0B Active	OK	OK
CAN 2.0B Passive	OK	容纳
CAN 2.0A	OK	总线错误

1 CAN总线的介绍

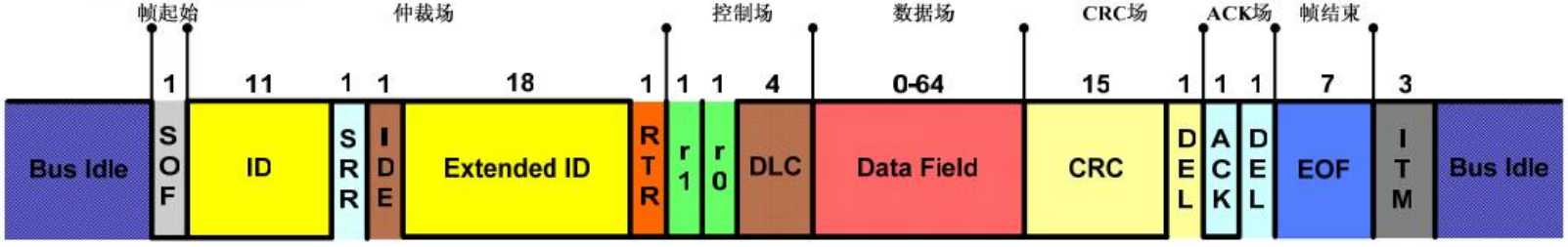
- CAN的帧格式
  - 数据帧➔携带从发送节点至接收节点的数据
  - 远程帧➔向其他节点请求发送具有同一标识符的数据帧
  - 帧间空间➔数据帧（或远程帧）通过帧间空间与前述的各帧分开
  - 错误帧➔节点检测到错误后发送错误帧
  - 超载帧➔在先行的和后续的数据帧（或远程帧）之间附加一段延时

1 CAN总线的介绍

- 数据帧
  - 标准帧

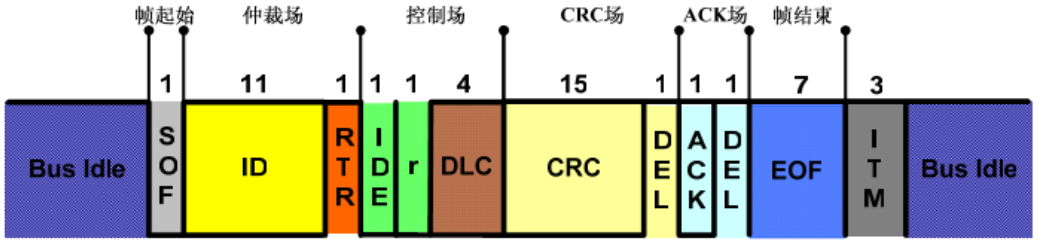


  - 扩展帧

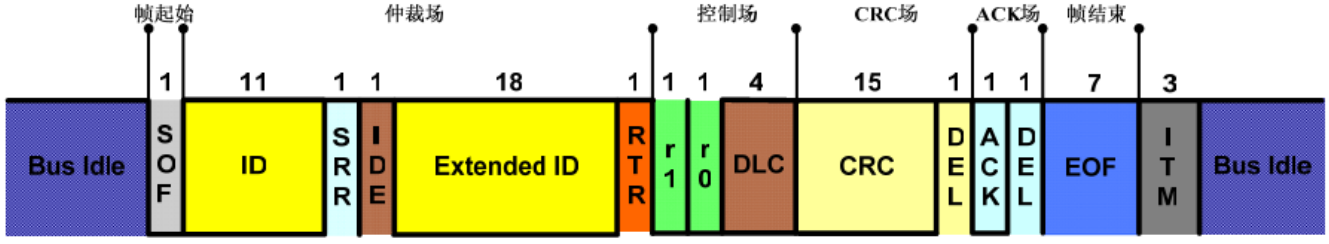


1 CAN总线的介绍

- 远程帧
  - 对应标准数据帧的远程帧



  - 对应扩展数据帧的远程帧



二、RT-Thread上的CAN驱动编写

二、RT-Thread上的CAN驱动编写

```
device->type        = RT_Device_Class_CAN;
device->rx_indicate = can_rx_indicate;
device->tx_complete = RT_NULL;
device->init        = rt_can_init;
device->open        = rt_can_open;
device->close       = rt_can_close;
device->read        = rt_can_read;
device->write       = rt_can_write;
device->control     = rt_can_control;
device->user_data   = can;
```

驱动的编写主要是实现以上函数的编写，其中以设备读和写的函数尤其重要，最终要将设备注册到内核中。

二、RT-Thread上的CAN驱动编写

建立了一种**CAN**设备类型和两个数据结构：

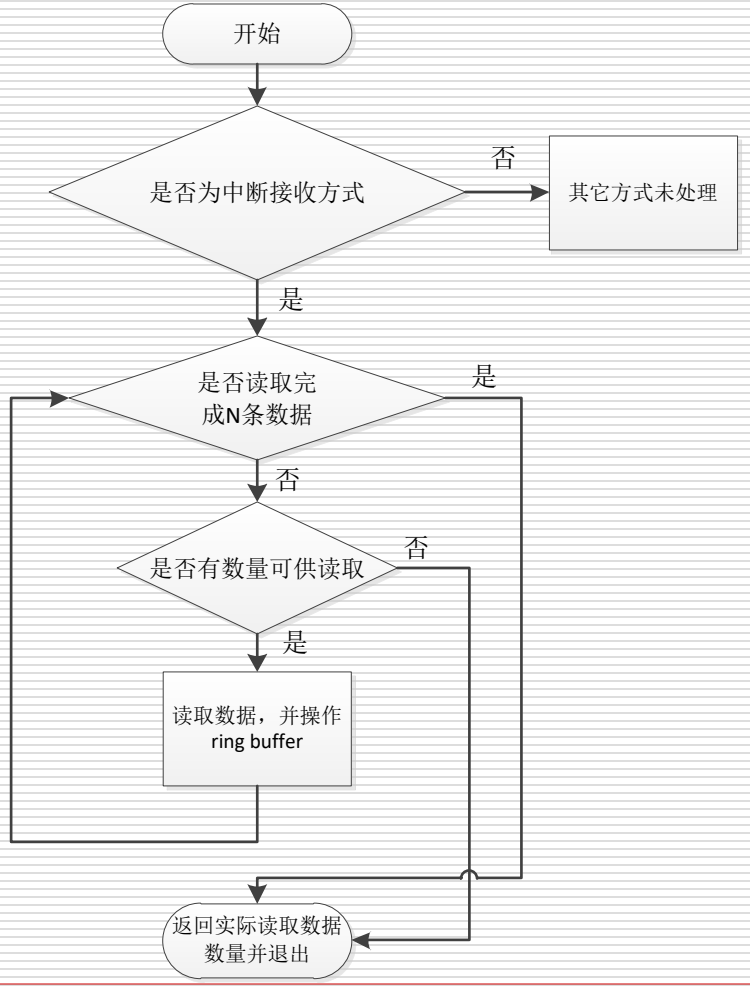
```
struct stm32_can_device
{
        CAN_TypeDef* can_device;

        /* rx structure */
        struct stm32_can_int_rx* int_rx;

        /* tx structure */
        struct stm32_can_int_tx* int_tx;
};
struct stm32_can_int_rx
{
        CanRxMsg  can_rx_buffer[CAN_RX_BUFFER_SIZE];
        rt_uint32_t read_index, save_index;
};

struct stm32_can_int_tx
{
        CanTxMsg  can_tx_buffer[CAN_TX_BUFFER_SIZE];
        rt_uint32_t read_index, save_index;
};
```

二、RT-Thread上的CAN驱动编写

```
typedef struct
{
        uint32_t StdId;  /*!< Specifies the standard identifier.
                            This parameter can be a value between 0 to 0x7FF. */

        uint32_t ExtId;  /*!< Specifies the extended identifier.
                            This parameter can be a value between 0 to 0x1FFFFFFF. */

        uint8_t IDE;     /*!< Specifies the type of identifier for the message that
                            will be received. This parameter can be a value of
                            @ref CAN_identifier_type */

        uint8_t RTR;     /*!< Specifies the type of frame for the received message.
                            This parameter can be a value of
                            @ref CAN_remote_transmission_request */

        uint8_t DLC;     /*!< Specifies the length of the frame that will be received.
                            This parameter can be a value between 0 to 8 */

        uint8_t Data[8]; /*!< Contains the data to be received. It ranges from 0 to
                            0xFF. */

        uint8_t FMI;     /*!< Specifies the index of the filter the message stored in
                            the mailbox passes through. This parameter can be a
                            value between 0 to 0xFF */
} CanRxMsg;
```

二、RT-Thread上的CAN驱动编写

CAN设备的读函数流程图



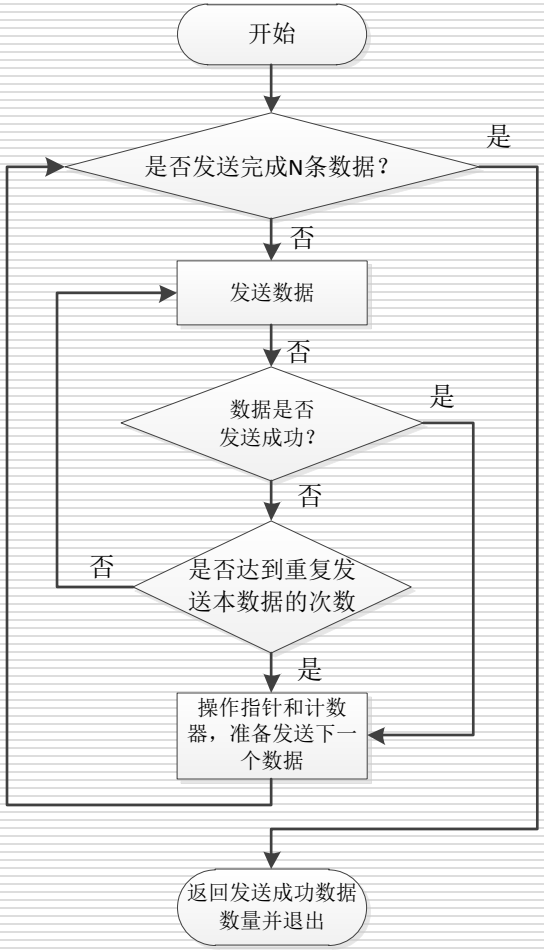
二、RT-Thread上的CAN驱动编写

CAN设备的读操作：

```
static rt_size_t rt_can_read (rt_device_t dev, rt_off_t pos, void*
buffer, rt_size_t size)
{
    CanRxMsg* ptr;
    rt_err_t err_code;
    rt_uint8_t length;
    struct stm32_can_device* can;

    ptr = (CanRxMsg *)buffer;
    err_code = RT_EOK;
    length = size;
    can = (struct stm32_can_device*)dev->user_data;

    if (dev->flag & RT_DEVICE_FLAG_INT_RX)
    {
        /* interrupt mode Rx */
        while (size)
        {
        rt_base_t level;

        /* disable interrupt */
        level = rt_hw_interrupt_disable();
        if (can->int_rx->read_index != can->int_rx->save_index)
        {
                size--;
                *ptr = can->int_rx->can_rx_buffer[can->int_rx-
        >read_index];
                /* move to next position */
                ptr ++;
                can->int_rx->read_index ++;
                if (can->int_rx->read_index >=
                CAN_RX_BUFFER_SIZE)
                can->int_rx->read_index = 0;
        }
        else
        {
                /* set error code */
                err_code = -RT_EEMPTY;

                /* enable interrupt */
                rt_hw_interrupt_enable(level);
                break;
        }
        /* enable interrupt */
        rt_hw_interrupt_enable(level);
        }
}
else
{
                /* polling mode */

}
/* set error code */
rt_set_errno(err_code);
return (length- size);
}
```

二、RT-Thread上的CAN驱动编写

CAN设备的写函数流程图



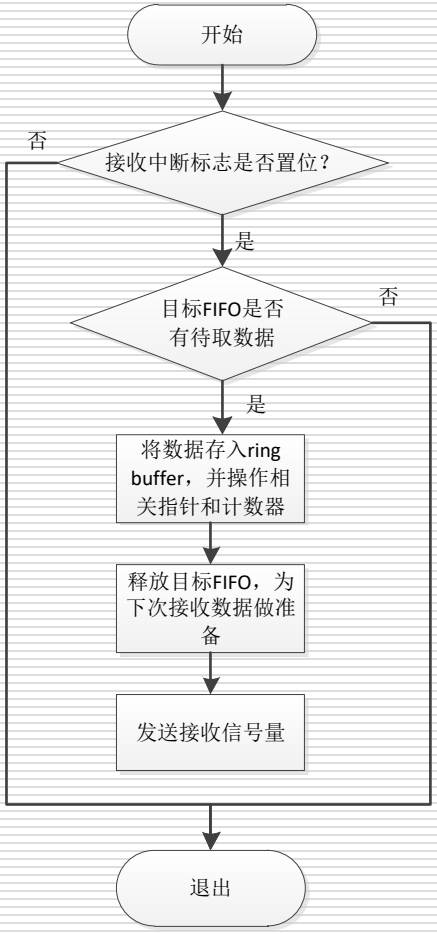
二、RT-Thread上的CAN驱动编写

CAN设备的写操作：

```
static rt_size_t rt_can_write(rt_device_t dev, rt_off_t pos, const void* buffer, rt_size_t size)
{
    CanTxMsg* ptr;
    rt_err_t err_code;
    uint8_t transmit_mailbox;
    rt_size_t fail_times = 0,length = size;
    struct stm32_can_device* can;

    err_code = RT_EOK;
    ptr = (CanTxMsg*)buffer;
    can = (struct stm32_can_device*)dev->user_data;

    if (dev->flag & RT_DEVICE_FLAG_INT_TX)
    {
            /* interrupt mode Tx, does not support */
    }
    else
    {
        while(size)
        {
            /* polling mode */
            do
            {
                can->int_tx->save_index ++;
                if (can->int_tx->save_index >= CAN_TX_TRY_TIMES)
                {
                    fail_times ++;
                        break;
                }
                transmit_mailbox = CAN_Transmit(can->can_device,  tr);
            }
            while(CAN_TransmitStatus(can-          >can_device,transmit_mailbox) != CAN_TxStatus_Ok);
            ptr ++;
            size --;
            can->int_tx->save_index = 0;
        }
    }
    /* set error code */
    rt_set_errno(err_code);

    return (length - fail_times);
}
```

二、RT-Thread上的CAN驱动编写

CAN设备的接收中断处理函数流程图



二、RT-Thread上的CAN驱动编写

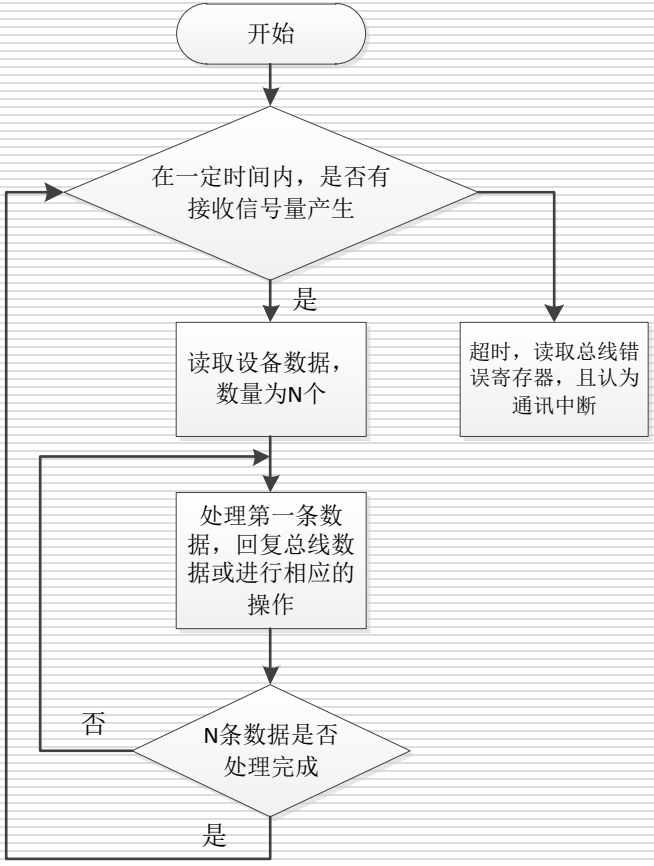
CAN设备的接收中断处理函数

```
void rt_hw_can_rev_isr(rt_device_t device, uint8_t can_fifo)
{
        uint32_t it_status;
        struct stm32_can_device* can = (struct stm32_can_device*) device->user_data;

        if(can_fifo == CAN_FIFO0)
                it_status = CAN_IT_FMP0;
        else if(can_fifo == CAN_FIFO1)
                it_status = CAN_IT_FMP1;
        if(CAN_GetITStatus(can->can_device, it_status) != RESET)
        {
                /* interrupt mode receive */
                RT_ASSERT(device->flag & RT_DEVICE_FLAG_INT_RX);

                /* save on rx buffer */
                while (CAN_MessagePending(can->can_device, can_fifo) > 0)
                {
                        rt_base_t level;

                        /* disable interrupt */
                        level = rt_hw_interrupt_disable();

                        /* save character */
                        CAN_Receive(can->can_device, can_fifo, &(can->int_rx->can_rx_buffer[can->int_rx->save_index]));
                        can->int_rx->save_index ++;
                        if (can->int_rx->save_index >= CAN_RX_BUFFER_SIZE)
                                can->int_rx->save_index = 0;

                        /* if the next position is read index, discard this 'read char' */
                        if (can->int_rx->save_index == can->int_rx->read_index)
                        {
                                can->int_rx->read_index ++;
                                if (can->int_rx->read_index >= CAN_RX_BUFFER_SIZE)
                                        can->int_rx->read_index = 0;
                        }
                        CAN_FIFORelease(can->can_device, can_fifo);
                        /* enable interrupt */
                        rt_hw_interrupt_enable(level);
                }

                /* invoke callback */
                if (device->rx_indicate != RT_NULL)
                {
                        rt_size_t rx_length;

                        /* get rx length */
                        rx_length = can->int_rx->read_index > can->int_rx->save_index ?
                                CAN_RX_BUFFER_SIZE - can->int_rx->read_index + can->int_rx->save_index :
                                can->int_rx->save_index - can->int_rx->read_index;

                        device->rx_indicate(device, rx_length);
                }
        }
}
```

三、CAN数据处理线程编写

三、CAN数据处理线程编写

CAN数据处理线程流程图



三、CAN数据处理线程编写

CAN设备的读操作：

```
void can_thread_entry(void* parameter)
{
uint8_t rx_buf_offset = 0;
rt_err_t err;
CanTxMsg tmsg;
CanRxMsg rmsg[10];
while(1)
{
       err= rt_sem_take(&can_rx_sem,RT_TICK_PER_SECOND * 60);
      if(err == RT_EOK)
      {
            register rt_base_t temp;
           temp = rt_hw_interrupt_disable();
           can_rx_sem.value = 0;
           rt_hw_interrupt_enable(temp);
           while( 1 )
           {
              uint8_t len;
              uint8_t i = 0;

              rx_buf_offset = 0;
// read data.
             len = rt_device_read(&dev_can,0, &rmsg[rx_buf_offset],
CAN_RX_BUFFER_SIZE - rx_buf_offset);
            if(len > 0)
            {
                 can_con_state = 1;
                 can_rx_led_flash();
            }
           if( rx_buf_offset == 0 )
           {
                if( len == 0 )
                 {
                    break;
                 }
           }
           rx_buf_offset += len;
```

```
while(rx_buf_offset)
{
        switch(rmsg[i].FMI)
        {
              case 0:
              {
                   tmsg.ExtId = REGISTER_CODE |
CAN_DIRECTION;
                   tmsg.StdId = 0;
                   tmsg.IDE = CAN_ID_EXT;//À©Õ¹Öi
                   tmsg.RTR = CAN_RTR_DATA;//Êý¾ÝÖi
                   tmsg.DLC = 8;//Êý¾Ý³¤¶È
                   memset(&tmsg.Data[0],0,8);
                   rt_device_write(&dev_can,0, &tmsg, 1);
                   can_tx_led_flash();
                   rt_event_send(&can_event, (1 << 0));
                   break;
              }
               case 1:
                   rt_event_send(&can_event, (1 << 1));
                   break;

               default:break;
           }
          rx_buf_offset --;
           i ++;
}

}
}
else if(err == RT_ETIMEOUT)
{
//读取错误状态寄存器，根据需要显示或上传
can_con_state = 0;
}
}
}
```