CANOpen 系列教程 06

# 结合代码理解 CAN 底层收发数据（含 STM32 例程)

作者：strongerHuang

申明：该文档仅供个人学习使用

| | |
|---|---|
| 归类 | CANOpen 系列教程 |
| 标签 | CAN、 CANOpen、 CanFestival |
| 网站 | http://www.strongerhuang.com |

# 目 录

# 一、写在前面

该教程前面讲述了许多关于 CAN 协议的一些概念，可能许多初学者看的云里雾里，那么本文将结合代码让大家理解之前讲述的内容。

因为关注我的人大部分都在学习 STM32，我将结合 **STM32F103、标准外设库例程**来让大家理解之前讲述的一些概念。

本文主要讲述内容：
**1. 传输数据相关参数**
**2. 位时序及传输波特率**

为方便大家理解，我将在最后提供对应例程「**CANOpen 系列教程 06_CAN 底层收发例程**」。当然，本文只讲述代码中部分内容，某些配置参数放在后面讲述。

本文章收录于【CANOpen 系列教程】，在我的博客分类“CANOpen 系列教程”也能查找到。

为了方便大家平时公交、地铁、外出办事也能用手机随时随地查看该教程，该系列教程也同步更新于微信公众号【EmbeddedDevelop】，关注微信公众号回复【CANOpen 系列教程】即可查看。

# 二、传输数据相关参数

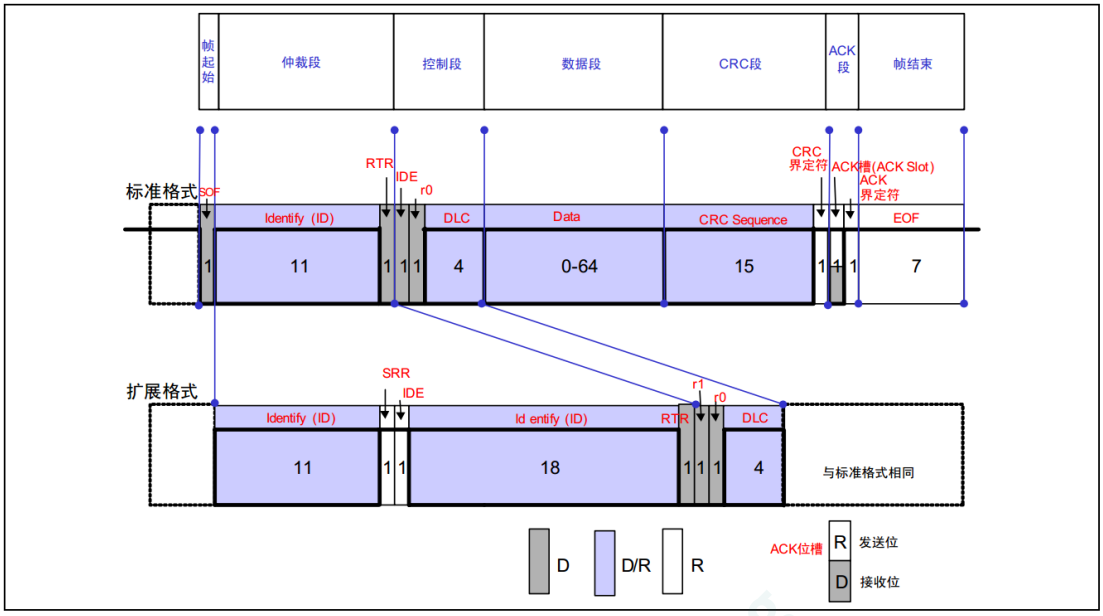
传输数据相关参数其实主要就是前面《CANOpen 系列教程 04》讲述的“**帧类型及格式说明**”那一章节内容，建议先看下那些概念内容。

CAN 总线传输的内容主要就是**发送和接收**，下面我将结合代码中发送和接收参数来让大家理解我们编程控制 CAN 总线上那些参数。

## 2.1 CAN 总线数据帧

本文提供例程主要是 CAN 总线使用最多的数据帧为例，让大家理解数据帧传输由我们编程控制的参数。

先看下图回顾一下数据帧格式，其中有些字段内容由控制器自动完成，如：帧起始，CRC 校验。**而有些就是由我们编程控制，如：ID、数据等**。




数据帧的构成

## 2.2 CAN 发送代码

### A.CAN 发送数据结构

下图主要就是 CAN 发送由我们编程控制的参数。其实你会发现，主要就是上面帧格式中部分内容：ID、IDE、RTR、DLC、Data。

stm32f10x_can.h

```
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 transmitted. This parameter can be a value
                        of @ref CAN_identifier_type */

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

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

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

### B.发送配置参数

下图为实际发送配置的参数。

main.c

```
64   /* 传输参数 */
65   TxMessage.StdId = 0x321;                    //标准帧ID
66   TxMessage.ExtId = 0x01;                     //扩展帧ID(暂未用)
67   TxMessage.IDE = CAN_ID_STD;                 //标准帧(或扩展帧)
68   TxMessage.RTR = CAN_RTR_DATA;               //数据帧(或远程帧)
69   TxMessage.DLC = 6;                          //长度
70   for(i=0; i<TxMessage.DLC; i++)
71   {
72     TxMessage.Data[i] = 0xA0 + i;             //数据A0 A1 A2 A3 A4 A5
73   }
```

## 2.3 CAN 接收代码

CAN 接收其实和发送类似，CAN 总线上的字段就那些，在接收端接收的那些参数无非就是发送端发送出来的那些参数。

针对 STM32 内部集成的 CAN，接收数据结构多了一个 FMI 参数，大概意思就是接收消息邮箱筛选器索引。

stm32f10x_can.h

```
173 typedef struct
174 {
175   uint32_t StdId;  /*!< Specifies the standard identifier.
176                         This parameter can be a value between 0 to 0x7FF. */
177
178   uint32_t ExtId;  /*!< Specifies the extended identifier.
179                         This parameter can be a value between 0 to 0x1FFFFFFF. */
180
181   uint8_t IDE;     /*!< Specifies the type of identifier for the message that
182                         will be received. This parameter can be a value of
183                         @ref CAN_identifier_type */
184
185   uint8_t RTR;     /*!< Specifies the type of frame for the received message.
186                         This parameter can be a value of
187                         @ref CAN_remote_transmission_request */
188
189   uint8_t DLC;     /*!< Specifies the length of the frame that will be received.
190                         This parameter can be a value between 0 to 8 */
191
192   uint8_t Data[8]; /*!< Contains the data to be received. It ranges from 0 to
193                         0xFF. */
194
195   uint8_t FMI;     /*!< Specifies the index of the filter the message stored in
196                         the mailbox passes through. This parameter can be a
197                         value between 0 to 0xFF */
198 } CanRxMsg;
```

**接收操作**

为方便初学者理解，这里这要就是使用中断接收 CAN 总线数据，在中断函数里面打印也主要是用于测试，一般实际项目打印不会出现在中断函数（打印相对耗时）。

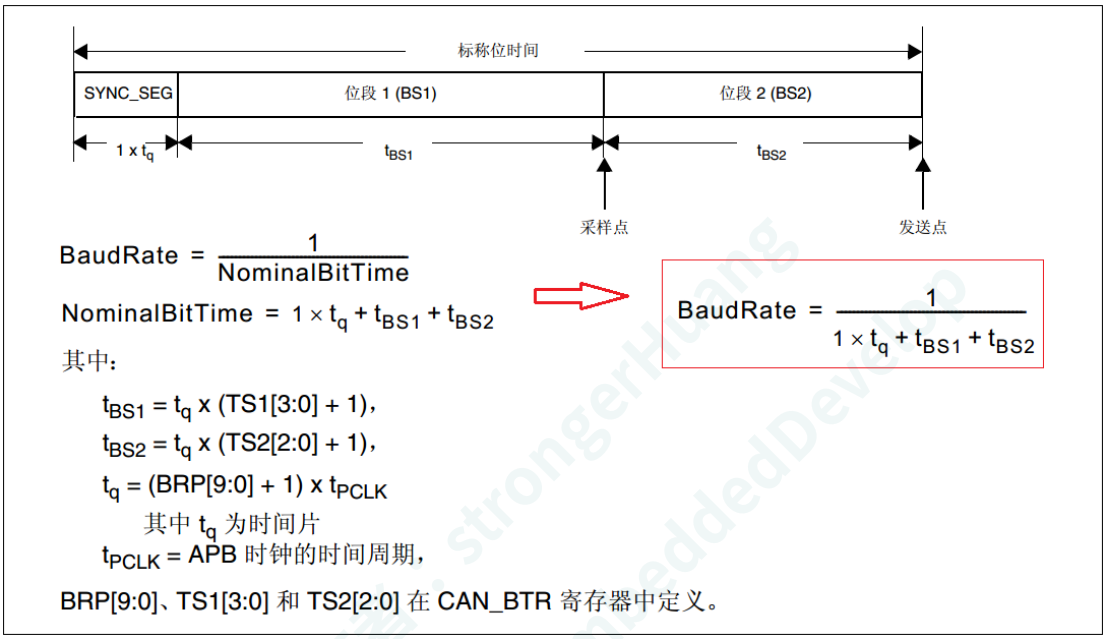
stm32f10x_it.c

```
144 *******************************************************/
145 void CAN_RX_IRQHandler(void) 中断函数入口是宏定义：#define CAN_RX_IRQHandler   USB_LP_CAN1_RX0_IRQHandler
146 {                            位于bsp_can.h文件
147   uint8_t i;
148   CanRxMsg RxMessage;
149
150   CAN_Receive(CANx, CAN_RX_FIFO, &RxMessage); 这里相当于提取相关参数到Rx数据结构
151
152   /* 打印接收数据(仅限测试使用) */
153   printf("StdId=%d; IDE=%d; RTR=%d; DLC=%d;\n", RxMessage.StdId, RxMessage.IDE, RxMessage.RTR, RxMessage.DLC);
154   for(i=0; i<RxMessage.DLC; i++)
155   {
156     printf("Data[%d]=%d\n", i, RxMessage.Data[i]);
157   }
158 }
```

# 三、位时序及传输波特率

同样，在前面《CANOpen 系列教程 04》中有讲述“位时序及传输波特率”的概念。其实，位时序间接决定了传输的波特率。换句话说，传输的波特率由位时序几个参数决定。

看下图，回顾一下波特率计算公式：




## 3.1 波特率配置代码

结合上图计算公式和下图代码，可以看出位时序中几个参数和波特率的关系。波特率为 1M 时，几个位时序参数可以配置为如下图值：

bsp_can.c

```
64  /* 配置CAN波特率 = 1MBps = 36M/4/(1+3+5) */
65  CAN_InitStructure.CAN_SJW = CAN_SJW_1tq;
66  CAN_InitStructure.CAN_BS1 = CAN_BS1_3tq;
67  CAN_InitStructure.CAN_BS2 = CAN_BS2_5tq;
68  CAN_InitStructure.CAN_Prescaler = 4;
69  CAN_Init(CANx, &CAN_InitStructure);
```

**提示：**

36M 代表 CAN 时钟，具体要看时钟相关配置。

波特率固定，位时序参数可以不同。比如波特率固定为 1M，位时序参数可以为上图配置；也可以修改其中的值，如修改位段 1 为 CAN_BS1_5tq，位段 2 为 CAN_BS2_3tq。只要遵循波特率计算公式即可。

# 四、例程下载

CANOpen 系列教程 06_CAN 底层收发例程：
https://pan.baidu.com/s/1LzD0Epc-Z8vlHsb-sD3WVw
提取码：l2dc

**提示：**
链接后期可能会失效，可回复【CANOpen 系列教程】查看更新链接。

# 五、说明

1.该文档部分文字来自网络，仅供个人学习使用，版权所有，禁止商用。

2. 本文由我一个人编辑并整理，难免存在一些错误。

3.本教程收录于微信公众号「嵌入式专栏」，关注微信公众号回复【CANOpen 系列教程】即可查看全系列教程。

# 六、最后

我的博客：http://www.strongerhuang.com
我的 GitHub：https://github.com/EmbeddedDevelop

我的微信公众号（ID：strongerHuang）还在分享 STM8、STM32、Keil、IAR、FreeRTOS、UCOS、RT-Thread、CANOpen、Modbus...等更多精彩内容，如果想查看更多内容，可以关注我的微信公众号。