CAN通信

CAN 是 Controller Area Network 的缩写（以下称为 CAN），是 ISO 国际标准化的串行通信 协议。在当前的汽车产业中，出于对安全性、舒适性、方便性、低公害、低成本的要求，各种 各样的电子控制系统被开发了出来。由于这些系统之间通信所用的数据类型及对可靠性的要求 不尽相同，由多条总线构成的情况很多，线束的数量也随之增加。为适应“减少线束的数量”、 “通过多个 LAN，进行大量数据的高速通信”的需要，1986 年德国电气商博世公司开发出面 向汽车的 CAN 通信协议。此后，CAN 通过 ISO11898 及 ISO11519 进行了标准化，现在在欧 洲已是汽车网络的标准协议。

CAN 控制器根据两根线上的电位差来判断总线电平。总线电平分为显性电平和隐性电平， 二者必居其一。发送方通过使总线电平发生变化，将消息发送给接收方。

物理层：

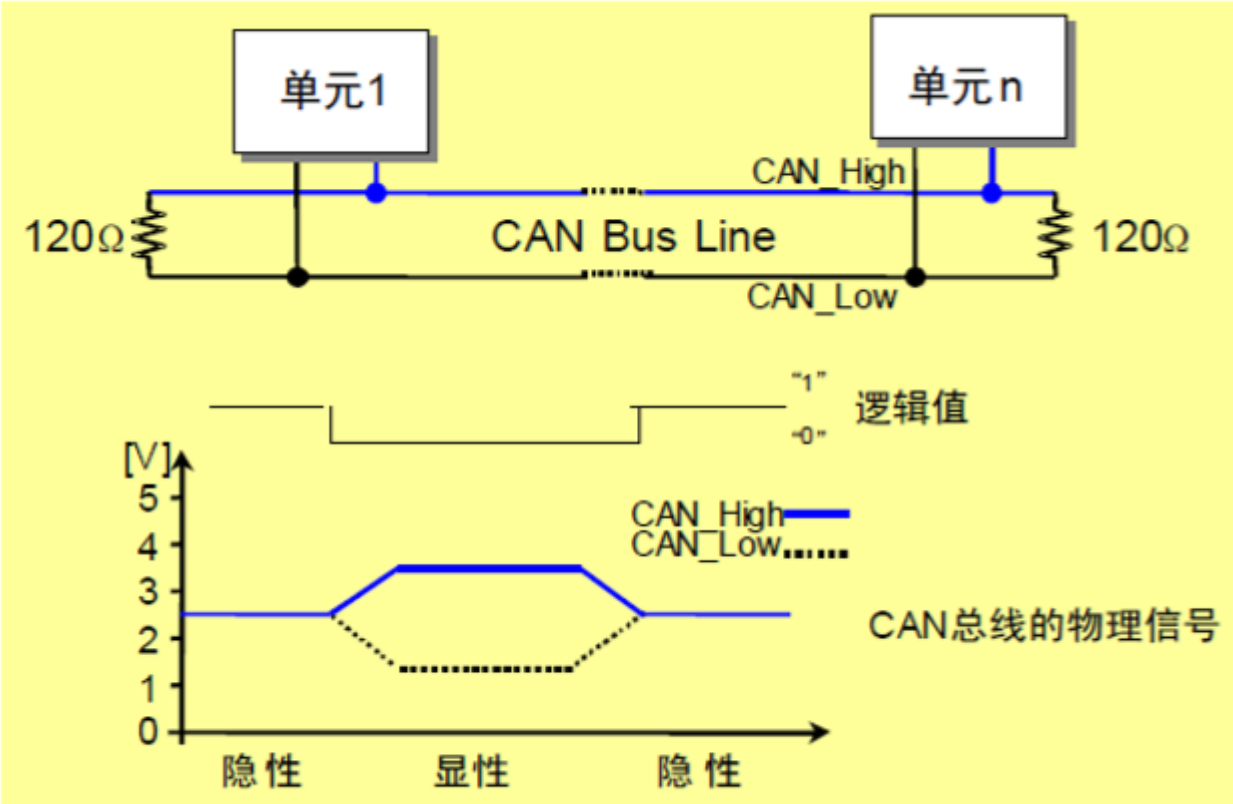


图 34.1.1 ISO11898 物理层特性

从特性可以看出，显性电平对应逻辑0，CAN_H 和 CAN_L 之差为 2.5V 左右。隐性电平对应逻辑1，CAN_H和CAN_L之差为0V.。在总线上显性电平具有优先权，只要有一个 单元输出显性电平，总线上即为显性电平。而隐形电平则具有包容的意味，只有所有的单元都 输出隐性电平，总线上才为隐性电平（显性电平比隐性电平更强）。另外，在 CAN 总线的起止 端都有一个 120Ω的终端电阻，来做阻抗匹配，以减少回波反射。

协议层：CAN 协议是通过以下 5 种类型的帧进行的。

□ 数据帧 □ 遥控帧 □ 错误帧 □ 过载帧 □ 间隔帧

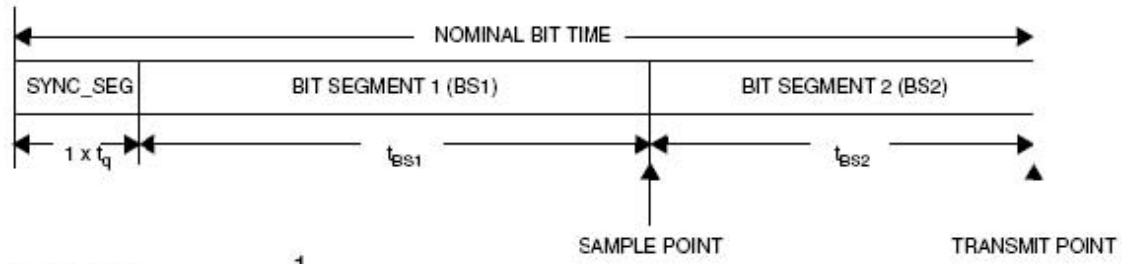
帧类型	帧用途
数据帧	用于发送单元向接收单元传送数据的帧
遥控帧	用于接收单元向具有相同 ID 的发送单元请求数据的帧
错误帧	用于当检测出错误时向其它单元通知错误的帧
过载帧	用于接收单元通知其尚未做好接收准备的帧
间隔帧	用于将数据帧及遥控帧与前面的帧分离开来的帧

其中数据帧一般由7个段构成：

帧起始	仲裁段	控制段	数据段	CRC段	ACK段	帧结束

（1） 帧起始。表示数据帧开始的段。
（2） 仲裁段。表示该帧优先级的段。
（3） 控制段。表示数据的字节数及保留位的段。
（4） 数据段。数据的内容，一帧可发送 0~8 个字节的数据。
（5） CRC 段。检查帧的传输错误的段。
（6） ACK 段。表示确认正常接收的段。
（7） 帧结束。表示数据帧结束的段。

协议这一块比较复杂，有兴趣自己查询资料了解了解。



$$BaudRate = \frac{1}{NominalBitTime}$$

$$NominalBitTime = 1 \times t_q + t_{BS1} + t_{BS2}$$

with:

$t_{BS1} = t_q \times (TS1[3:0] + 1)$,

$t_{BS2} = t_q \times (TS2[2:0] + 1)$,

$t_q = (BRP[9:0] + 1) \times t_{PCLK}$

where t_q refers to the Time quantum

t_{PCLK} = time period of the APB clock,

BRP[9:0], TS1[3:0] and TS2[2:0] are defined in the CAN_BTR Register.

CAN波特率=APB总线频率/BRP分频器/(1+TBS1+TBS2)。

CAN初始化步骤：

1、配置相关引脚的复用功能（AF9），使能 CAN 时钟。

我们要用 CAN，第一步就要使能 CAN 的时钟，CAN 的时钟通过 APB1ENR 的第 25 位来 设置。其次要设置 CAN 的相关引脚为复用输出，这里我们需要设置 PA11（CAN1_RX）和 PA12 （CAN1_TX）为复用功能（AF9），并使能 PA 口的时钟。具体配置过程如下：

```
GPIO_InitTypeDef GPIO_Initure;

__HAL_RCC_CAN1_CLK_ENABLE();          //使能 CAN1 时钟
__HAL_RCC_GPIOA_CLK_ENABLE();         //开启 GPIOA 时钟

GPIO_Initure.Pin=GPIO_PIN_11|GPIO_PIN_12;      //PA11,12
GPIO_Initure.Mode=GPIO_MODE_AF_PP;             //推挽复用
GPIO_Initure.Pull=GPIO_PULLUP;                 //上拉
GPIO_Initure.Speed=GPIO_SPEED_FAST;            //快速
GPIO_Initure.Alternate=GPIO_AF9_CAN1;          //复用为 CAN1
HAL_GPIO_Init(GPIOA,&GPIO_Initure);            //初始化
```

2、设置CAN工作模式及波特率。

这一步通过先设置 CAN_MCR 寄存器的 INRQ 位，让 CAN 进入初始化模式，然后设置 CAN_MCR 的其他相关控制位。再通过 CAN_BTR 设置波特率和工作模式（正常模式/环回模式） 等信息。 最后设置 INRQ 为 0，退出初始化模式。

在库函数中，提供了函数 HAL_CAN_Init 用来初始化 CAN 的工作模式以及波特率， HAL_CAN_Init 函数体中，在初始化之前，会设置 CAN_MCR 寄存器的 INRQ 为 1 让其进入初 始化模式，然后初始化 CAN_MCR 寄存器和 CRN_BTR 寄存器之后，会设置 CAN_MCR 寄存 器的 INRQ 为 0 让其退出初始化模式。所以我们在调用这个函数的前后不需要再进行初始化模式设置。

```
1  HAL_StatusTypeDef HAL_CAN_Init(CAN_HandleTypeDef* hcan);
```

```
typedef struct
{
  uint32_t Prescaler;
  uint32_t Mode;
  uint32_t SJW;
  uint32_t BS1;
  uint32_t BS2;
  uint32_t TTCM;
  uint32_t ABOM;
  uint32_t AWUM;
  uint32_t NART;
  uint32_t RFLM;
  uint32_t TXFP;
}CAN_InitTypeDef;
```

这个结构体看起来成员变量比较多，实际上参数可以分为两类。前面 5 个参数是用来设置 寄存器 CAN_BTR，用来设置模式以及波特率相关的参数，设置模式的参数是 Mode ，我们实验中用到回环模式CAN_MODE_LOOPBACK 和常规模式 CAN_MODE_NORMAL。其他设置波特率相关的参数 Prescaler，SJW，BS1 和 BS2 分别用来设置波特率分频器，重新同步跳跃宽度以及时 间段 1 和时间段 2 占用的时间单元数。后面 6 个成员变量用来设置寄存器 CAN_MCR，也就是 设置 CAN 通信相关的控制位.初始化示例：

```
CAN_HandleTypeDef    CAN1_Handler;   //CAN1 句柄
CanTxMsgTypeDef      TxMessage;      //发送消息
CanRxMsgTypeDef      RxMessage;      //接收消息
CAN1_Handler.Instance=CAN1;
CAN1_Handler.pTxMsg=&TxMessage;   //发送消息
CAN1_Handler.pRxMsg=&RxMessage;   //接收消息
CAN1_Handler.Init.Prescaler=6;    //分频系数(Fdiv)为 brp+1
CAN1_Handler.Init.Mode= CAN_MODE_LOOPBACK;      //模式设置:回环模式
CAN1_Handler.Init.SJW= CAN_SJW_1TQ;    //重新同步跳跃宽度为 tsjw+1 个时间单位
CAN1_Handler.Init.BS1= CAN_BS1_8TQ;    //tbs1范围CAN_BS1_1TQ~CAN_BS1_16TQ
CAN1_Handler.Init.BS2= CAN_BS2_6TQ;    //tbs2 范围 CAN_BS2_1TQ~CAN_BS2_8TQ
CAN1_Handler.Init.TTCM=DISABLE; //非时间触发通信模式
CAN1_Handler.Init.ABOM=DISABLE; //软件自动离线管理
CAN1_Handler.Init.AWUM=DISABLE; //睡眠模式通过软件唤醒
CAN1_Handler.Init.NART=ENABLE;  //禁止报文自动传送
CAN1_Handler.Init.RFLM=DISABLE;  //报文不锁定,新的覆盖旧的
CAN1_Handler.Init.TXFP=DISABLE;  //优先级由报文标识符决定
HAL_CAN_Init(&CAN1_Handler);
```

3、设置滤波器。

在 HAL 库中，提供了函数 HAL_CAN_ConfigFilter 用来初始化 CAN 的滤波器相关参数。HAL_CAN_ConfigFilter 函数体中，在初始化滤波器之前，会设置 CAN_FMR 寄存器的 FINIT 位为 1 让其进入初始化模式，然后初始化 CAN 滤波器相关的寄存器之后，会设置 CAN_FMR 寄存器的 FINIT 位为 0 让其退出初始化模式。

所以我们在调用这个函数的前后不需要再进行初 始化模式设置。

```
HAL_StatusTypeDef HAL_CAN_ConfigFilter(CAN_HandleTypeDef* hcan,  CAN_FilterConfTypeDef* sFilterConfig);
```

过滤器初始化参考实例代码：

```
/**
* @brief CAN外设过滤器初始化
* @param can结构体
* @retval None
*/
HAL_StatusTypeDef CanFilterInit(CAN_HandleTypeDef* hcan)
{
  CAN_FilterTypeDef  sFilterConfig;

  sFilterConfig.FilterBank = 0;
  sFilterConfig.FilterMode = CAN_FILTERMODE_IDMASK;
  sFilterConfig.FilterScale = CAN_FILTERSCALE_32BIT;
  sFilterConfig.FilterIdHigh = 0x0000;
  sFilterConfig.FilterIdLow = 0x0000;
  sFilterConfig.FilterMaskIdHigh = 0x0000;
  sFilterConfig.FilterMaskIdLow = 0x0000;
  sFilterConfig.FilterFIFOAssignment = CAN_RX_FIFO0;
  sFilterConfig.FilterActivation = ENABLE;
  sFilterConfig.SlaveStartFilterBank = 14;

    if(hcan == &hcan1)
    {
        sFilterConfig.FilterBank = 0;
    }
    if(hcan == &hcan2)
    {
        sFilterConfig.FilterBank = 14;
    }

  if(HAL_CAN_ConfigFilter(hcan, &sFilterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_CAN_Start(hcan) != HAL_OK)        //开启CAN
  {
    Error_Handler();
  }

  if (HAL_CAN_ActivateNotification(hcan, CAN_IT_RX_FIFO0_MSG_PENDING) != HAL_OK)
  {
    Error_Handler();
  }

    return HAL_OK;
}
```

4.发送接收消息

在初始化 CAN 相关参数以及过滤器之后，接下来就是发送和接收消息了。

发送消息的函数是：

```
HAL_StatusTypeDef HAL_CAN_Transmit(CAN_HandleTypeDef* hcan, uint32_t Timeout);
```

接受消息的函数是：

```
HAL_StatusTypeDef HAL_CAN_Receive(CAN_HandleTypeDef* hcan,  uint8_t FIFONumber, uint32_t Timeout);
```

使用范例：

```
CanFilterInit(&hcan1);          //初始化CAN1过滤器(应写在主函数中)

/**
 * @brief CAN通信接收**中断**回调函数
 * @param CAN序号
 * @retval None
 */
void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan)
{
    CAN_RxHeaderTypeDef   RxHeader;
    if(HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, CanReceiveData) != HAL_OK)
  {
    Error_Handler();            //如果CAN通信数据接收出错，则进入死循环
  }
  CanDataReceive(RxHeader.StdId);   //进行电机数据解析
}
```

从代码中可以看出，初始化完成之后，只需要调用CAN的过滤器初始化函数即可使用CAN通信了。

CAN通信的发送函数如下*（此处以大疆的电机通信协议为例）*：

```
/**
* @brief ID为1~4的电机信号发送函数
* @param ID为1~4的各个电机的电流数值
* @retval None
*/
void CanTransmit_1234(CAN_HandleTypeDef *hcanx, int16_t cm1_iq, int16_t cm2_iq, int16_t cm3_iq, int16_t
cm4_iq)
{
  CAN_TxHeaderTypeDef TxMessage;

    TxMessage.DLC=0x08;//数据长度（最大为8）
  TxMessage.StdId=0x200;//标准标识符
  TxMessage.IDE=CAN_ID_STD;//使用标准帧
  TxMessage.RTR=CAN_RTR_DATA;//数据帧
  uint8_t TxData[8];

    TxData[0] = (uint8_t)(cm1_iq >> 8);
    TxData[1] = (uint8_t)cm1_iq;
    TxData[2] = (uint8_t)(cm2_iq >> 8);
    TxData[3] = (uint8_t)cm2_iq;
    TxData[4] = (uint8_t)(cm3_iq >> 8);
    TxData[5] = (uint8_t)cm3_iq;
```

```
    TxData[6] = (uint8_t)(cm4_iq >> 8);
    TxData[7] = (uint8_t)cm4_iq;

    if(HAL_CAN_AddTxMessage(hcanx,&TxMessage,TxData,(uint32_t*)CAN_TX_MAILBOX0)!=HAL_OK)
    {
         Error_Handler();       //如果CAN信息发送失败则进入死循环
    }
}
```

CAN通信有一点，如果要用CAN2，必须要使能CAN1.