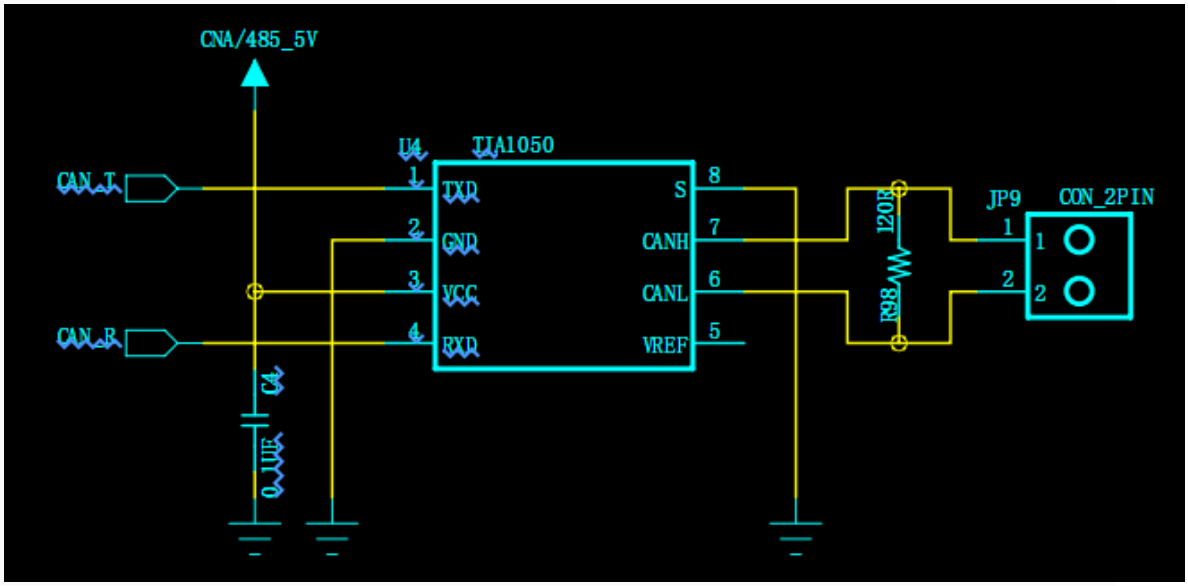
STM32 的 CAN 通信实现（代码+图示）

1.CAN 是控制器局域网络(Controller Area Network, CAN)的简称

（理论知识不做讲解了，太多了）

2.芯片选用：TJA1050



差分信号输入，

这里的显性电平 CANH 和 CANL 压差是 2V 左右，逻辑上表示“0”

两线之间没有压差 CANH 和 CANL 都是 2.5V 左右，表示逻辑“1”

（二）实现代码

1.

```
/* 配置 CAN 模块*/

CAN_Config();

void CAN_Config(void)

{

CAN_GPIO_Config();    //此处用 PB8/PB9

CAN_NVIC_Config();

CAN_Mode_Config();

CAN_Filter_Config();

}
```

2.对于 CAN_NVIC_Config();

看下 CAN 的中断分类

CAN 的中断由发送中断、接收 FIFO 中断、错误中断构成。发送中断由三个发送邮箱任意一个为空的事件构成。接收 FIFO 中断分为 FIFO0 和 FIFO1 的中断，接收 FIFO 收到新的报文或报文溢出的事件可以引起中断。

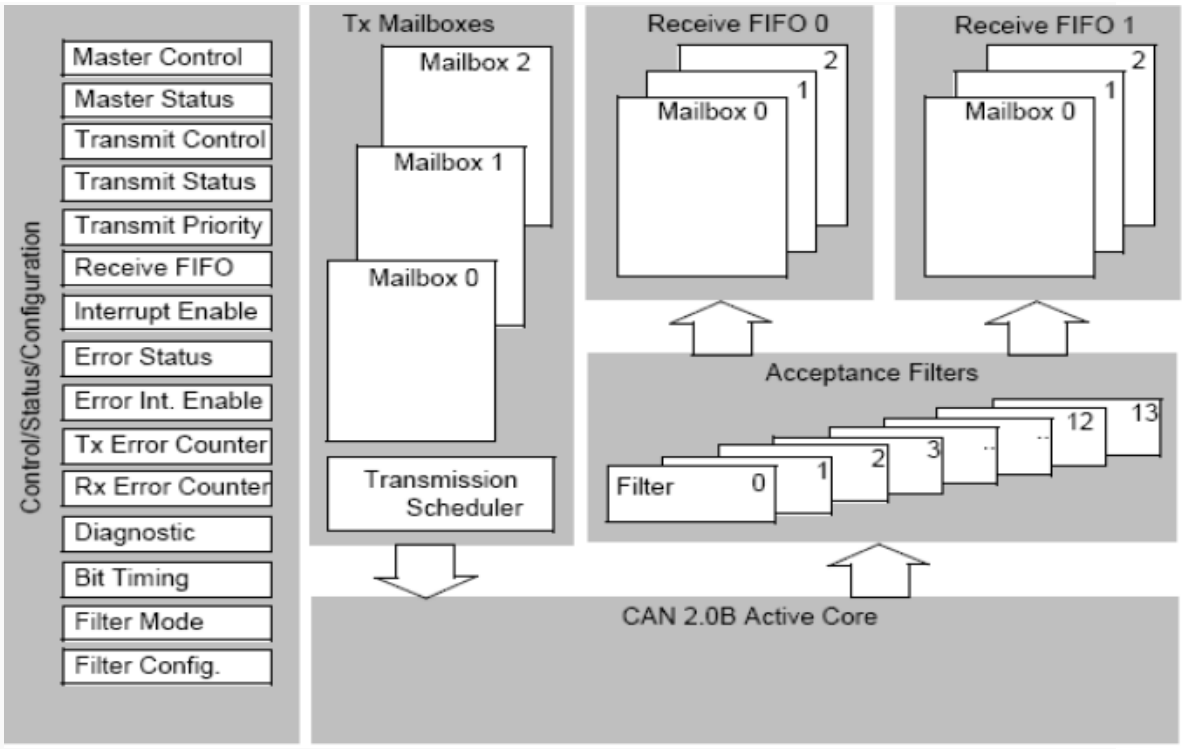
本实验中使用的 RX0 中断通道即为 FIFO0 中断通道，当 FIFO0 收到新报文时，引起中断，我们就在相应的中断服务函数读取这个新报文。

```
static void CAN_NVIC_Config(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
   /* Configure one bit for preemption priority */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
   /*中断设置*/
   NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn;//CAN1 RX0
中断
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;//抢占优先级 0
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;     //子优先级 0
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}
```

3.CAN_Mode_Config();　　//CAN 不像 I2C 那样有片选，他背身有许多协议，也要进行选择，所以对应代码的结构体也多（集体可看 CAN 手册）

4.CAN_Filter_Config(); 过滤器程序

看下 CAN 的主要框图（M3 里面的 CAN）



M3 里面有三个发送邮箱，把要发送的数据打包成报文，然后把它发送到 CAN 网络总线上；

接收器 先经过滤器，M3 里面有 14 个过滤器，过滤的时候是根据报文的 ID 来过滤的，ID 相同的话，才能过去（相当于一个闸门）

接收到数据后（这里有两个 FIFO ，0、1 ），我们的实验用的是 放到 FIFO 的 Mailbox 0；

代码写法根基三部分：

1.Tx Mailboxes 2.Acceptance Filters 3.Receive FIFO

```
static void CAN_Filter_Config(void)
{
   CAN_FilterInitTypeDef  CAN_FilterInitStructure;

  /*CAN过滤器初始化*/
  CAN_FilterInitStructure.CAN_FilterNumber=0;           //过滤器组0
  CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask; //工作在标识符屏蔽位模式
  CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;  //过滤器位宽为单个32位。
  /* 使能报文标示符过滤器按照标示符的内容进行比对过滤，扩展ID不是如下的就抛弃掉，是的话，会存入FIF00。 */

    CAN_FilterInitStructure.CAN_FilterIdHigh= (((u32)0x1314<<3)&0xFFFF0000)>>16;        //要过滤的ID高位
    CAN_FilterInitStructure.CAN_FilterIdLow= (((u32)0x1314<<3)|CAN_ID_EXT|CAN_RTR_DATA)&0xFFFF; //要过滤的ID低位
    CAN_FilterInitStructure.CAN_FilterMaskIdHigh= 0xFFFF;      //过滤器高16位每位必须匹配
    CAN_FilterInitStructure.CAN_FilterMaskIdLow= 0xFFFF;       //过滤器低16位每位必须匹配
  CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_Filter_FIFO0 ;       //过滤器被关联到FIF00
  CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;       //使能过滤器
  CAN_FilterInit(&CAN_FilterInitStructure);
  /*CAN通信中断使能*/
  CAN_ITConfig(CAN1, CAN_IT_FMP0, ENABLE);
}
```

STM32 的 ID 过滤方式有两种，一种为标识符列表模式，它把要接收报文的 ID 列成一个表，要求报文 ID 与列表中的某一个标识符完全相同才可以接收，可以理解为白名单管理。（说白了就是有一个标志位相同，然后就联通了）另一种称为标识符屏蔽模式，它把可接收报文 ID 的某几位作为列表，这几位被称为屏蔽位，可以把它理解成关键字搜索，只要屏蔽位(关键字)相同，就符合要求。即这种模式只要求报文 ID 的屏蔽位与列表中标识符相应屏蔽位相同 ，报文就被保存到接收 FIFO。（标志位的某几位检验，相同的话就通过了）

2.什么叫做报文（就是帧）

在原始数据段的前面加上传输起始标签、片选(识别)标签、控制标签，在数据的尾段加上 CRC 校验标签、应答标签和传输结束标签。把这些内容按特定的格式打包好，就可以用一个通道表达各种信号了，各种各样的标签就如同 SPI 中各种通道上的信号，起到了协同传输的作用。当整个数据包被传输到其它设备时，只要这些设备按格式去解读，就能还原出原始数据。类似这样的数据包就被称为 CAN 的数据帧。

帧	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。
帧间隔	用于将数据帧及遥控帧与前面的帧分离开来的帧。

看下数据帧

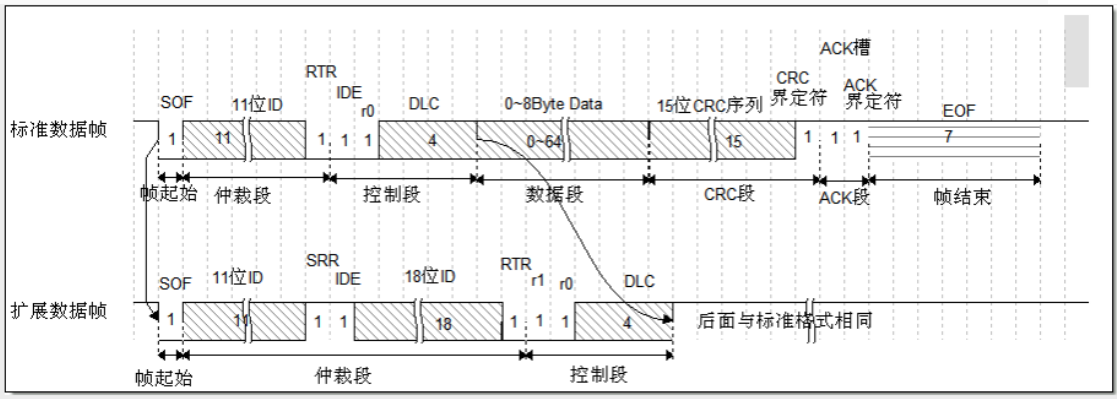
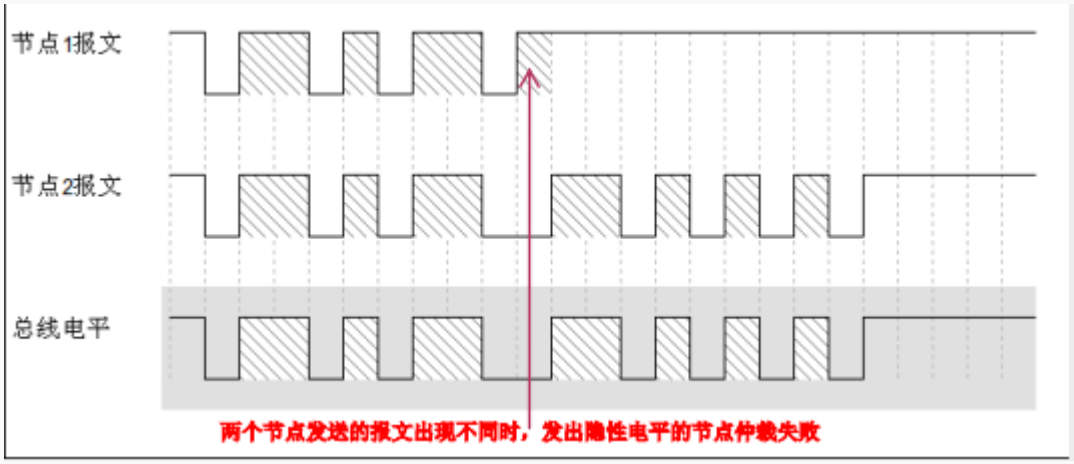


图 18-2 数据帧的结构

数据帧以一个显性位(逻辑 0)开始，以 7 个连续的隐性位(逻辑 1)结束。在它们之间，分为仲裁段、控制段、数据段、CRC 段和 ACK 段。

扩展与标准 仲裁段不一样，扩展 11+18 = 29 位

仲裁段：告诉是发还是收（几个同时通信的时候，那那根的信号，拉低了就是谁的了）如：



总电平接收了节点 2 的^_^（can 的神奇之处）

CRC：发送与接收的对应

IDE：用于区分标准帧与扩展帧

r0、r1 都是显性位

3.CAN 通信报文内容设置

void CAN_SetMsg(void)

4.发送消息，“ABCD”

CAN_Transmit(CAN1, &TxMessage);

总体流程图（例程对应）：

和从机。本实验中的流程图如下：

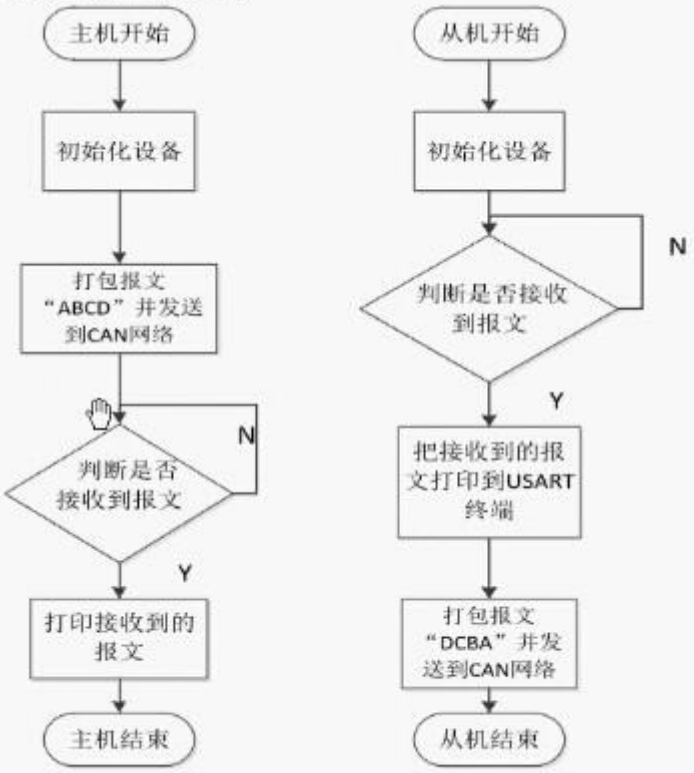


图 18-11 主机及从机的执行流程

while(flag == 0xff); //flag =0 ,success ， 此时产生一个中断（CAN 中断）

对于从机的 CAN_Config(); 是一模一样

不一样的是从机先上电

中断函数时一样的

CAN 线接的时候 CANH 接 CANH, CANL 接 CANL, 不能对接