STM32 的 can 现场总线实验心得

最近在搞 stm32 实验板的 can 现场总线实验，之前只是搞过 STC51 的串口通信，相比之下，发觉 can 总线都挺复杂的。开始时，知道自己是新手，只知道 can 总线跟串行通信，485 通信，I2C 通信一样都是用来传输数据通信的，对其工作原理一窍不通，还是从基础开始看书看资料，先了解它的基本原理吧。

原来 can 总线有以下特点：

主要特点

支持CAN协议2.0A和2.0B主动模式
波特率最高可达1兆位/秒
支持时间触发通信功能

发送

3个发送邮箱
发送报文的优先级特性可软件配置
记录发送SOF时刻的时间戳

接收

3级深度的2个接收FIFO
14个位宽可变的过滤器组－由整个CAN共享
标识符列表
FIFO溢出处理方式可配置
记录接收SOF时刻的时间戳

可支持时间触发通信模式

禁止自动重传模式
16位自由运行定时器
定时器分辨率可配置
可在最后2个数据字节发送时间戳

管理

中断可屏蔽
邮箱占用单独 1 块地址空间，便于提高软件效率

看完这些特点后，疑问一个一个地出现，

1. 什么是时间触发功能？
2. 发送邮箱是什么来的？
3. 报文是什么来的？
4. 什么叫时间戳？
5. 什么叫接收 FIFO？
6. 什么叫过滤器？

好了，带着疑问往下看，看完一遍后，

报文:

报文包含了将要发送的完整的数据信息

发送邮箱:

共有 3 个发送邮箱供软件来发送报文。发送调度器根据优先级决定哪个邮箱的报文先被发送。

接收过滤器:

共有14个位宽可变/可配置的标识符过滤器组，软件通过对它们编程，从而在引脚收到的报文中选择它需要的报文，而把其它报文丢弃掉。

接收FIFO

共有2个接收FIFO，每个FIFO都可以存放3个完整的报文。它们完全由硬件来管理

工作模式

bxCAN 有 3 个主要的工作模式：初始化、正常和睡眠模式。

初始化模式

*软件通过对CAN_MCR寄存器的INRQ位置1，来请求bxCAN进入初始化模式，然后等待硬件对CAN_MSR寄存器的INAK位置1来进行确认
*软件通过对CAN_MCR寄存器的INRQ位清0，来请求bxCAN退出初始化模式，当硬件对 CAN_MSR 寄存器的 INAK 位清 0 就确认了初始化模式的退出。
*当bxCAN处于初始化模式时，报文的接收和发送都被禁止，并且CANTX引脚输出隐性位（高电平）

正常模式

在初始化完成后，软件应该让硬件进入正常模式，以便正常接收和发送报文。软件可以通过
对CAN_MCR寄存器的INRQ位清0，来请求从初始化模式进入正常模式，然后要等待硬件对CAN_MSR寄存器的INAK位置1的确认。在跟CAN总线取得同步，即在CANRX引脚上监测到11个连续的隐性位（等效于总线空闲）后，bxCAN才能正常接收和发送报文。
过滤器初值的设置不需要在初始化模式下进行，但必须在它处在非激活状态下完成（相应的FACT位为0）。而过滤器的位宽和模式的设置，则必须在初始化模式下，进入正常模式前完成。

睡眠模式（低功耗）

*软件通过对CAN_MCR寄存器的SLEEP位置1，来请求进入这一模式。在该模式下，bxCAN的时钟停止了，但软件仍然可以访问邮箱寄存器。
*当bxCAN处于睡眠模式，软件想通过对CAN_MCR寄存器的INRQ位置1，来进入初始化式，那么软件必须同时对SLEEP位清0才行
*有2种方式可以唤醒（退出睡眠模式）bxCAN：通过软件对SLEEP位清0，或硬件检测CAN总线的活动。

工作流程

那么究竟can是怎样发送报文的呢？

发送报文的流程为：

应用程序选择1个空发送邮箱；设置标识符，数据长度和待发送数据；

然后对CAN_TIxR寄存器的TXRQ位置1，来请求发送。TXRQ位置1后，邮箱就不再是空邮箱；而一旦邮箱不再为空，软件对邮箱寄存器就不再有写的权限。TXRQ位置1后，邮箱马上进入挂号状态，并等待成为最高优先级的邮箱，参见发送优先级。一旦邮箱成为最高优先级的邮箱，其状态就变为预定发送状态。一旦CAN总线进入空闲状态，预定发送邮箱中的报文就马上被发送（进入发送状态）。一旦邮箱中的报文被成功发送后，它马上变为空邮箱；硬件相应地对CAN_TSR寄存器的RQCP和TXOK位置1，来表明一次成功发送。

如果发送失败，由于仲裁引起的就对CAN_TSR寄存器的ALST位置1，由于发送错误引起的就对 TERR 位置 1。

原来发送的优先级可以由标识符和发送请求次序决定：

由标识符决定

当有超过1个发送邮箱在挂号时，发送顺序由邮箱中报文的标识符决定。根据CAN协议，标

识符数值最低的报文具有最高的优先级。如果标识符的值相等，那么邮箱号小的报文先被发

送。

由发送请求次序决定

通过对CAN_MCR寄存器的TXFP位置1，可以把发送邮箱配置为发送FIFO。在该模式下，发送的优先级由发送请求次序决定。

该模式对分段发送很有用。

时间触发通信模式

在该模式下，CAN硬件的内部定时器被激活，并且被用于产生时间戳，分别存储在CAN_RDTxR/CAN_TDTxR寄存器中。内部定时器在接收和发送的帧起始位的采样点位置被采样，并生成时间戳（标有时间的数据）。

接着又是怎样接收报文的呢?

接收管理

接收到的报文，被存储在3级邮箱深度的FIFO中。FIFO完全由硬件来管理，从而节省了CPU

的处理负荷，简化了软件并保证了数据的一致性。应用程序只能通过读取FIFO输出邮箱，来读取FIFO中最先收到的报文。

有效报文

根据CAN协议，当报文被正确接收（直到EOF域的最后1位都没有错误），且通过了标识符
过滤，那么该报文被认为是有效报文。

接收相关的中断条件

* 一旦往FIFO存入1个报文，硬件就会更新FMP[1:0]位，并且如果CAN_IER寄存器的FMPIE位为1，那么就会产生一个中断请求。
* 当FIFO变满时（即第3个报文被存入），CAN_RFxR寄存器的FULL位就被置1，并且如果CAN_IER寄存器的FFIE位为1，那么就会产生一个满中断请求。
* 在溢出的情况下，FOVR位被置1，并且如果CAN_IER寄存器的FOVIE位为1，那么就会产生一个溢出中断请求

标识符过滤

在CAN协议里，报文的标识符不代表节点的地址，而是跟报文的内容相关的。因此，发送者以广播的形式把报文发送给所有的接收者。（注：不是一对一通信，而是多机通信）节点在接收报文时－根据标识符的值－决定软件是否需要该报文；如果需要，就拷贝到SRAM里；如果不需要，报文就被丢弃且无需软件的干预。
为满足这一需求，bxCAN为应用程序提供了14个位宽可变的、可配置的过滤器组（13~0），以便只接收那些软件需要的报文。硬件过滤的做法节省了CPU开销，否则就必须由软件过滤从而占用一定的CPU开销。每个过滤器组x由2个32位寄存器，CAN_FxR0和CAN_FxR1组成。

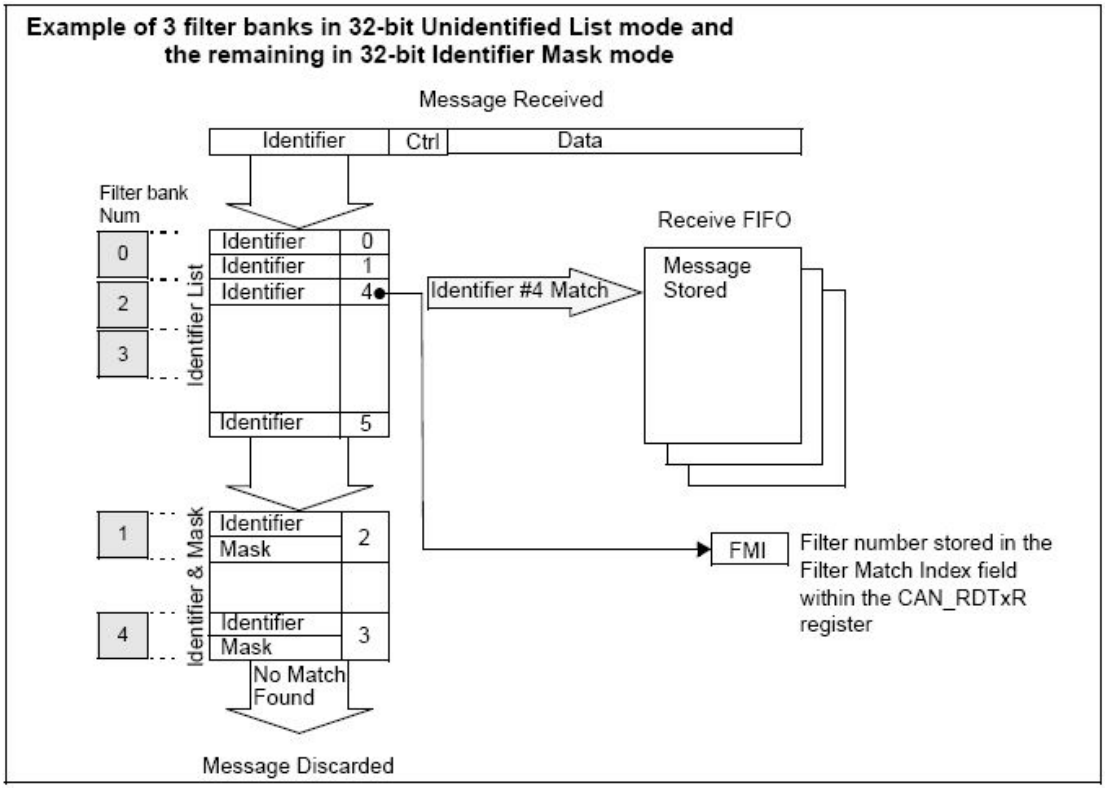
过滤器的模式的设置

通过设置CAN_FMOR的FBMx位，可以配置过滤器组为标识符列表模式或屏蔽位模式。
为了过滤出一组标识符，应该设置过滤器组工作在屏蔽位模式。
为了过滤出一个标识符，应该设置过滤器组工作在标识符列表模式。
应用程序不用的过滤器组，应该保持在禁用状态。

过滤器优先级规则

位宽为32位的过滤器，优先级高于位宽为16位的过滤器
对于位宽相同的过滤器，标识符列表模式的优先级高于屏蔽位模式
位宽和模式都相同的过滤器，优先级由过滤器号决定，过滤器号小的优先级高

图128 过滤器机制的例子



上面的例子说明了bxCAN的过滤器规则：在接收一个报文时，其标识符首先与配置在标识符列表模式下的过滤器相比较；如果匹配上，报文就被存放到相关联的FIFO中，并且所匹配的过滤器的序号被存入过滤器匹配序号中。如同例子中所显示，报文标识符跟#4标识符匹配，因此报文内容和FMI4被存入FIFO。如果没有匹配，报文标识符接着与配置在屏蔽位模式下的过滤器进行比较。如果报文标识符没有跟过滤器中的任何标识符相匹配，那么硬件就丢弃该报文，且不会对软件有任何打扰。

接收邮箱（FIFO）

在接收到一个报文后，软件就可以访问接收FIFO的输出邮箱来读取它。一旦软件处理了报文（如把它读出来），软件就应该对CAN_RFxR寄存器的RFOM位进行置1，来释放该报文，以便为后面收到的报文留出存储空间。

中断

bxCAN占用4个专用的中断向量。通过设置CAN中断允许寄存器(CAN_IER)，每个中断源都可以单独允许和禁用。

发送中断可由下列事件产生：

— 发送邮箱0变为空，CAN_TSR寄存器的RQCP0位被置1。
— 发送邮箱1变为空，CAN_TSR寄存器的RQCP1位被置1。

— 发送邮箱2变为空，CAN_TSR寄存器的RQCP2位被置1。

FIFO0中断可由下列事件产生：

— FIFO0接收到一个新报文，CAN_RF0R寄存器的FMP0位不再是‘00’。
— FIFO0变为满的情况，CAN_RF0R寄存器的FULL0位被置1。
— FIFO0发生溢出的情况，CAN_RF0R寄存器的FOVR0位被置1。

FIFO1中断可由下列事件产生：

— FIFO1接收到一个新报文，CAN_RF1R寄存器的FMP1位不再是‘00’。
— FIFO1变为满的情况，CAN_RF1R寄存器的FULL1位被置1。
— FIFO1发生溢出的情况，CAN_RF1R寄存器的FOVR1位被置1。

错误和状态变化中断可由下列事件产生：

— 出错情况，关于出错情况的详细信息请参考CAN错误状态寄存器(CAN_ESR)。
— 唤醒情况，在CAN接收引脚上监视到帧起始位(SOF)。
— CAN 进入睡眠模式。

工作流程大概就是这个样子，接着就是一大堆烦人的 can 寄存器，看了一遍总算有了大概的了解，况且这么多的寄存器要一下子把他们都记住是不可能的。根据以往的经验，只要用多几次，对寄存器的功能就能记住。

好了，到读具体实验程序的时候了，这时候就要打开“STM32 库函数”的资料

因为它里面有 STM32 打包好的库函数的解释，对读程序很有帮助。

下面是主程序：

```
int main(void)
{
//  int press_count = 0;
   char data = '0';
   int sent = FALSE;
```

```
#ifdef DEBUG
  debug();
#endif

  /* System Clocks Configuration */
  RCC_Configuration();

  /* NVIC Configuration */
  NVIC_Configuration();

  /* GPIO ports pins Configuration */
  GPIO_Configuration();

  USART_Configuration();

  CAN_Configuration();

  Serial_PutString("\r\n      伟      研      科      技
http://www.gzweiyan.com\r\n");
  Serial_PutString("CAN test\r\n");
```

```
  while(1){
    if(GPIO_Keypress(GPIO_KEY, BUT_RIGHT)){
      GPIO_SetBits(GPIO_LED, GPIO_LD1);   //检测到按键按下

      if(sent == TRUE)
        continue;
      sent = TRUE;
      data++;
      if(data > 'z')
        data = '0';
      CAN_TxData(data);
    }
    else{   //按键放开
      GPIO_ResetBits(GPIO_LED, GPIO_LD1);
      sent = FALSE;
    }
  }
}
```

前面的 RCC、NVIC、GPIO、USART 配置和之前的实验大同小异，关键是分析 CAN

_Configuration()

函数如下：

```
void CAN_Configuration(void)//CAN配置函数
{
  CAN_InitTypeDef        CAN_InitStructure;
  CAN_FilterInitTypeDef  CAN_FilterInitStructure;

  /* CAN register init */
  CAN_DeInit();
//  CAN_StructInit(&CAN_InitStructure);

  /* CAN cell init */
  CAN_InitStructure.CAN_TTCM=DISABLE;//禁止时间触发通信模式
  CAN_InitStructure.CAN_ABOM=DISABLE;//，软件对CAN_MCR寄存器的INRQ位进行置1
                                       随后清0后，一旦硬件检测
                                     //到128次11位连续的隐性位，就退出离线
状态。

  CAN_InitStructure.CAN_AWUM=DISABLE;//睡眠模式通过清除CAN_MCR寄存器的
                                       SLEEP位，由软件唤醒

  CAN_InitStructure.CAN_NART=ENABLE;//DISABLE;CAN报文只被发送1次，不管发送
                                     的结果如何（成功、出错或仲裁丢失）

  CAN_InitStructure.CAN_RFLM=DISABLE;//在接收溢出时FIFO未被锁定，当接收FIFO
                                       的报文未被读出，下一个收到的报文会覆
                                       盖原有
                                                          //的报文

  CAN_InitStructure.CAN_TXFP=DISABLE;//发送FIFO优先级由报文的标识符来决定
//  CAN_InitStructure.CAN_Mode=CAN_Mode_LoopBack;
  CAN_InitStructure.CAN_Mode=CAN_Mode_Normal; //CAN硬件工作在正常模式
  CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;//重新同步跳跃宽度1个时间单位
  CAN_InitStructure.CAN_BS1=CAN_BS1_8tq;//时间段1为8个时间单位
  CAN_InitStructure.CAN_BS2=CAN_BS2_7tq;//时间段2为7个时间单位
  CAN_InitStructure.CAN_Prescaler = 9; //(pclk1/((1+8+7)*9)) = 36Mhz/16/9 =
                                       250Kbits设定了一个时间单位的长度9
  CAN_Init(&CAN_InitStructure);

  /* CAN filter init 过滤器初始化*/
  CAN_FilterInitStructure.CAN_FilterNumber=0;//指定了待初始化的过滤器0
  CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;//指定了过
                                       滤器将被初始化到的模式为标识符屏
                                       蔽位模式
```

```
  CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;//给出了过滤器位宽1个32位过滤器

  CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;//用来设定过滤器标识符（32位位宽时为其高段位，16位位宽时为第一个）

  CAN_FilterInitStructure.CAN_FilterIdLow=0x0000;//用来设定过滤器标识符（32位位宽时为其低段位，16位位宽时为第二个

  CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;//用来设定过滤器屏蔽标识符或者过滤器标识符（32位位宽时为其高段位，16位位宽时为第一个

  CAN_FilterInitStructure.CAN_FilterMaskIdLow=0x0000;//用来设定过滤器屏蔽标识符或者过滤器标识符（32位位宽时为其低段位，16位位宽时为第二个

  CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_FIFO0;//设定了指向过滤器的FIFO0

  CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;//使能过滤器
  CAN_FilterInit(&CAN_FilterInitStructure);

  /* CAN FIFO0 message pending interrupt enable */
  CAN_ITConfig(CAN_IT_FMP0, ENABLE);//使能指定的CAN中断
}
```

再看看发送程序：

```
TestStatus CAN_TxData(char data)
{
  CanTxMsg TxMessage;

  u32 i = 0;
  u8 TransmitMailbox = 0;
```

```
/*
  u32 dataLen;
  dataLen = strlen(data);
  if(dataLen > 8)
    dataLen = 8;
    */
  /* transmit 1 message 生成一个信息 */
  TxMessage.StdId=0x00;// 设定标准标识符
  TxMessage.ExtId=0x1234;// 设定扩展标识符
  TxMessage.IDE=CAN_ID_EXT;// 设定消息标识符的类型
  TxMessage.RTR=CAN_RTR_DATA;// 设定待传输消息的帧类型
/*  TxMessage.DLC= dataLen;
    for(i=0;i<dataLen;i++)
        TxMessage.Data[i] = data[i];
*/
  TxMessage.DLC= 1; //设定待传输消息的帧长度
  TxMessage.Data[0] = data;// 包含了待传输数据


  TransmitMailbox = CAN_Transmit(&TxMessage);//开始一个消息的传输

  i = 0;
  while((CAN_TransmitStatus(TransmitMailbox) != CANTXOK) && (i != 0xFF))//通
                                                          过检查 CANTXOK 位来确认发送
                                                          是否成功
  {
    i++;
  }



  return (TestStatus)ret;
}
```

CAN_Transmit（）函数的操作包括：

1. [选择一个空的发送邮箱]
2. [设置 Id]*
3. [设置 DLC 待传输消息的帧长度]
4. [请求发送]

请求发送语句：

```
CAN->sTxMailBox[TransmitMailbox].TIR |= TMIDxR_TXRQ;//对 CAN_TIxR 寄存器的
```

TXRQ 位置 1，来请求发送

发送方面搞定了，但接收方面呢？好像在主程序里看不到有接收的语句。立刻向师兄求救。
原来是用来中断方式来接收数据，原来它和串口一样可以有两种方式接收数据，一种是中断方式一种是轮询方式，若采用轮询方式则要调用主函数的 CAN_Polling(void)函数。

接着又遇到一个问题，为什么中断函数 CAN_Interrupt(void)的最后要关中断呢？
因为一旦往 FIFO 存入 1 个报文，硬件就会更新 FMP[1:0]位，并且如果 CAN_IER 寄存器的 FMPIE 位为 1，那么就会产生一个中断请求。所以中断函数执行完后就要清除 FMPIE 标志位。这时我才回想起来，原来我对 CAN 的理解还不够，对程序设计的初衷不够明确，于是我重新看了一遍 CAN 的工作原理，这时后我发现比以前容易理解了，可能是因为看了程序以后知道了大概的流程，然后看资料就有了针对性。

发送者以广播的形式把报文发送给所有的接收者（注：不是一对一通信，而是多机通信）节点在接收报文时－根据标识符的值－决定软件是否需要该报文；如果需要，就拷贝到 SRAM里；如果不需要，报文就被丢弃且无需软件的干预。**一旦往FIFO存入1个报文，硬件就会更新FMP[1:0]位，并且如果CAN_IER寄存器的FMPIE位为1，那么就会产生一个中断请求。所以中断函数执行完后就要清除FMPIE标志位。**

By：becarson

http://becarsonfeng.blog.163.com