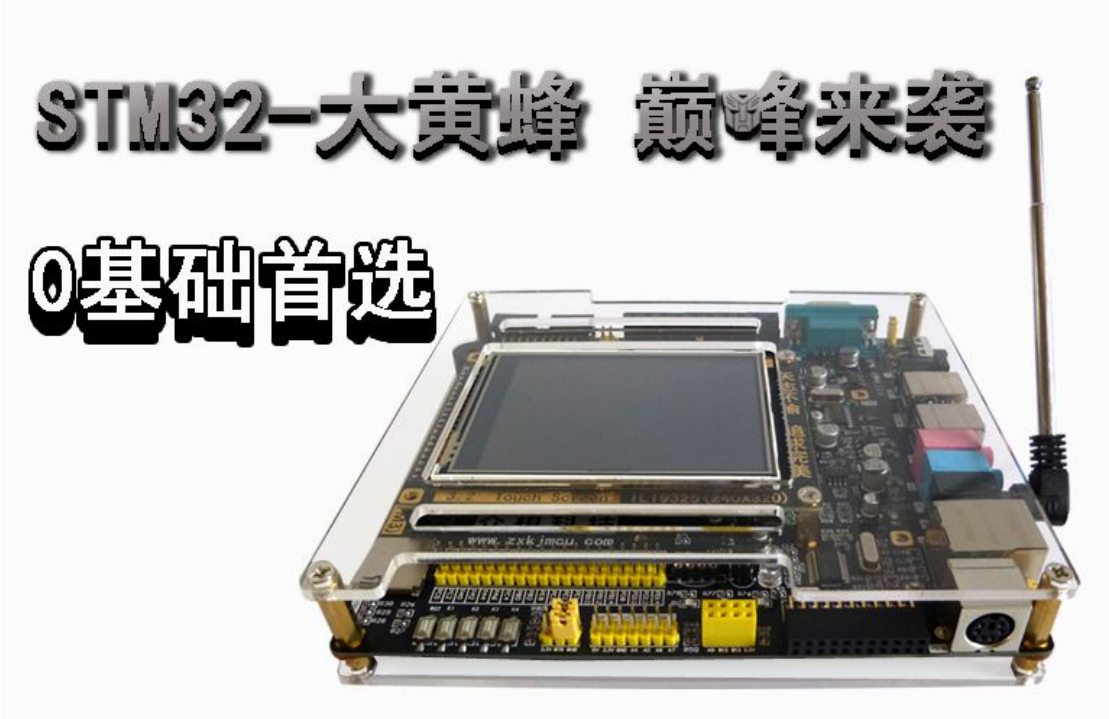
学 ARM 从 STM32 开始

STM32 开发板库函数教程--实战篇



官方网站：http://www.zxkjmcu.com
官方店铺：http://zxkjmcu.taobao.com
官方论坛：http://bbs.zxkjmcu.com
刘洋课堂：http://school.zxkjmcu.com

4.10 STM32 CAN 通讯原理及实验

4.10.1 概述

4.10.1.1 CAN 概念

控制器局域网总线（CAN，Controller Area Network）是一种用于实时应用的串行通讯协议总线，它可以使用双绞线来传输信号，是世界上应用最广泛的现场总线之一。CAN 协议由德国的 Robert Bosch 公司开发，用于汽车中各种不同元件之间的通信，以此取代昂贵而笨重的配电线束。该协议的健壮性使其用途延伸到其他自动化和工业应用。CAN 协议的特性包括完整性的串行数据通讯、提供实时支持、传输速率高达 1Mb/s、同时具有 11 位的寻址以及检错能力。

CAN 总线是一种多主方式的串行通讯总线，基本设计规范要求有高的位速率，高抗电子干扰性，并且能够检测出产生的任何错误。CAN 总线可以应用于汽车电控制系统、电梯控制系统、安全监测系统、医疗仪器、纺织机械、船舶运输等领域。

4.10.1.2 CAN 总线的特点

1、具有实时性强、传输距离较远、抗电磁干扰能力强、成本低等优点；

2、采用双线串行通信方式，检错能力强，可在高噪声干扰环境中工作；

3、具有优先权和仲裁功能，多个控制模块通过 CAN 控制器挂到 CAN-bus 上，形成多主机局部网络；

4、可根据报文的 ID 决定接收或屏蔽该报文；

5、可靠的错误处理和检错机制；

6、发送的信息遭到破坏后，可自动重发；

7、节点在错误严重的情况下具有自动退出总线的功能；

8、报文不包含源地址或目标地址，仅用标志符来指示功能信息、优先级信息。

4.10.1.3 CAN 总线的工作原理

CAN 总线使用串行数据传输方式，可以 1Mb/s 的速率在 40m 的双绞线上运行，也可以使用光缆连接，而且在这种总线上总线协议支持多主控制器。CAN 与 I2C 总线的许多细节很类似，但也有一些明显的区别。

当 CAN 总线上的一个节点(站)发送数据时，它以报文形式广播给网络中所有节点。对每个节点来说，无论数据是否是发给自己的，都对其进行接收。每组报文开头的 11 位字符为标识符，定义了报文的优先级，这种报文格式称为面向内容的编址方案。在同一系统中标识符是唯一的，不可能有两个站发送具有相同标识符的报文。当几个站同时竞争总线读取时，这种配置十分重要。

当一个站要向其它站发送数据时，该站的 CPU 将要发送的数据和自己的标识符传送给本站的 CAN 芯片，并处于准备状态；当它收到总线分配时，转为发送报文状态。CAN 芯片将数据根据协议组织成一定的报文格式发出，这时网上的其它站处于接收状态。每个处于接收状态的站对接收到的报文进行检测，判断这些报文是否是发给自己的，以确定是否接收它。

由于 CAN 总线是一种面向内容的编址方案，因此很容易建立高水准的控制系统并灵活地进行配置。我们可以很容易地在 CAN 总线中加进一些新站而无需在硬件或软件上进行修改。当所提供的新站是纯数据接收设备时，数据

传输协议不要求独立的部分有物理目的地址。它允许分布过程同步化，即总线上控制器需要测量数据时，可由网上获得，而无须每个控制器都有自己独立的传感器。

4.10.1.4 CAN 总线的应用

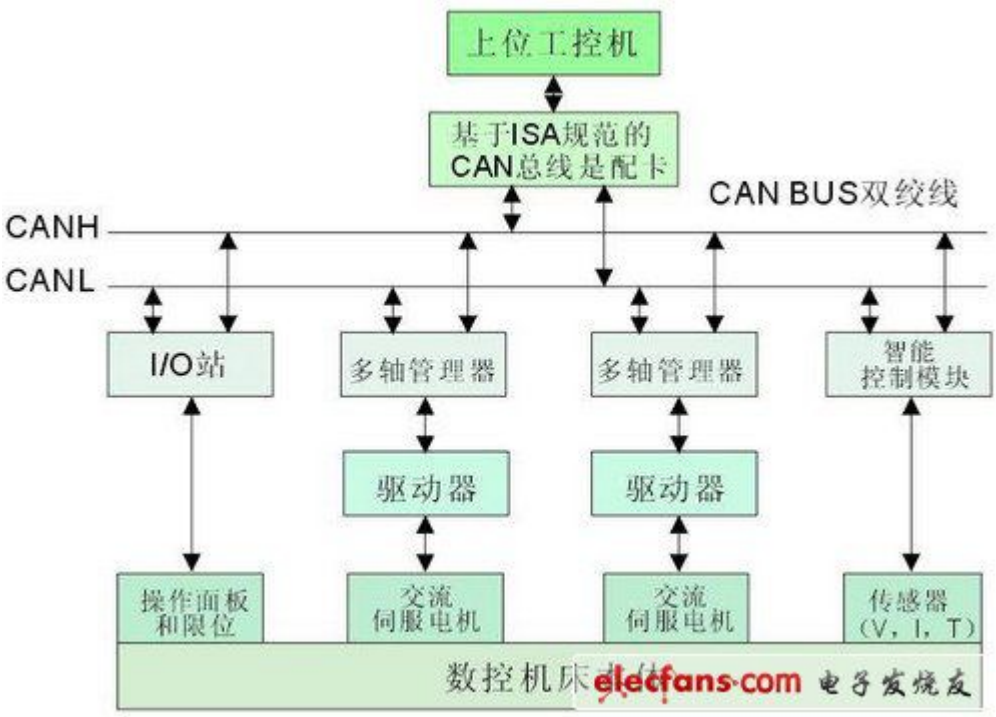
CAN 总线在组网和通信功能上的优点以及其高性价比据定了它在许多领域有广阔的应用前景和发展潜力。这些应用有些共同之处：CAN 实际就是在现场起一个总线拓扑的计算机局域网的作用。不管在什么场合，它负担的是任一节点之间的实时通信，但是它具备结构简单、高速、抗干扰、可靠、价位低等优势。CAN 总线最初是为汽车的电子控制系统而设计的，目前在欧洲生产的汽车中 CAN 的应用已非常普遍，不仅如此，这项技术已推广到火车、轮船等交通工具中。

1、汽车制造中的应用

2、大型仪器设备中的应用

3、工业控制中的应用

随着计算机技术、通信技术和控制技术的发展,传统的工业控制领域正经历着一场前所未有的变革,而工业控制的网络化,更拓展了工业控制领域的发展空间,带来新的发展机遇。在广泛的工业领域，CAN 总线可作为现场设备级的通信总线，而且与其他的总线相比，具有很高的可靠性和性能价格比。这将是 CAN 技术开发应用的一个主要的方向。



4、智能家庭和生活小区管理中的应用

5、机器人网络互联中的应用

4.10.2 STM bxCAN 主要特点

STM bxCAN 在 CAN 的基础上增加了一些新的特色，功能更丰富一些，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
- 可变的过滤器组：在互联网产品中，CAN1 和 CAN2 分享 28 个过滤器组，其它的 STM32F103xx 系列产品中有 14 个过滤器组
- 标识符列表
- 记录接收 SOF 时刻的时间戳

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
- 可在最后 2 个数据字节发送时间戳管理
- 中断可以屏蔽

4.10.3 STM bxCAN 初学者需要关注的几个重点

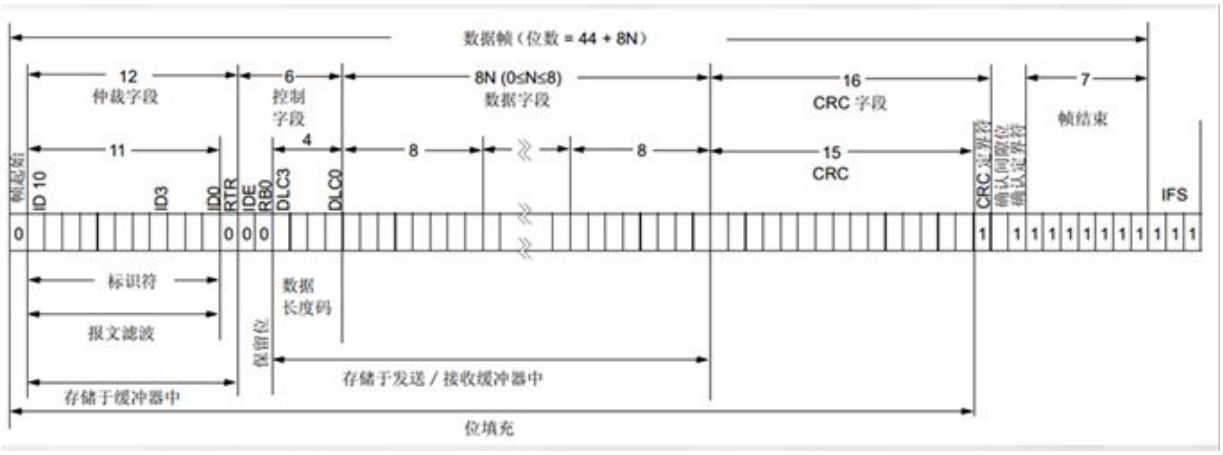
1、隐性位与显性位

CAN 总线为“隐性”（逻辑 1）时，CAN_H 和 CAN_L 的电平为 2.5V（电位差为 0V）；

CAN 总线为“显性”（逻辑 0）时，CAN_H 和 CAN_L 的电平为 3.5V 和 1.5V（电位差为 2.5V）；

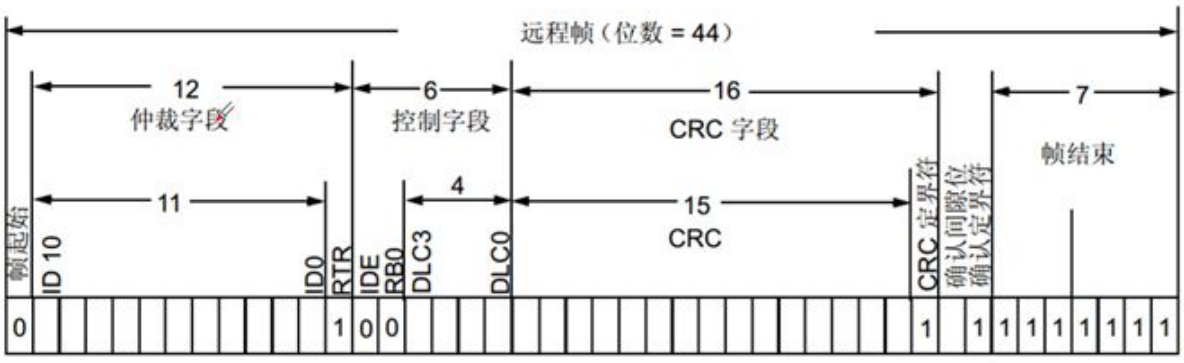
2、数据帧类型

(1)标准数据帧由这几部分组成：12 位冲裁字段、6 位控制字段、数据字段（8N）、16 位 CRC 校验字段、7 位帧结束。

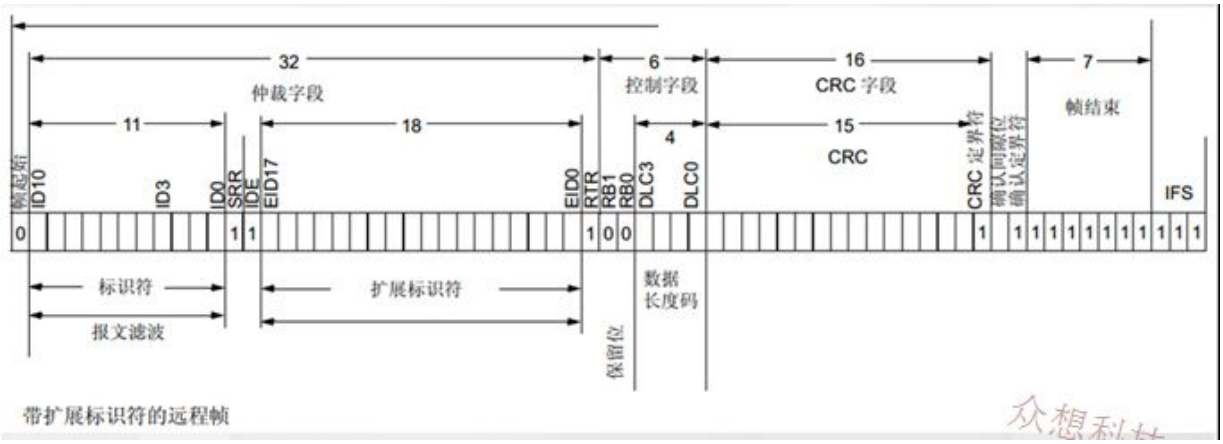


(2)扩展数据帧由这几部分组成：32 位冲裁字段、6 位控制字段、数据字段（8N）、16 位 CRC 校验字段、7 位帧结束。

(3)标准远程帧由这几部分组成：16 位冲裁字段、6 位控制字段、16 位 CRC 校验字段、7 位帧结束。



(4)标准远程帧由这几部分组成：32 位冲裁字段、6 位控制字段、16 位 CRC 校验字段、7 位帧结束。



4.10.4 STM bxCAN 控制和状态寄存器

4.10.4.1 CAN 主控制寄存器（CAN_MCR）

CAN 主控制寄存器是一个 32 位的寄存器。

地址偏移量：0x00

复位值：0x0001 0002

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															DBF
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESET	保留							TICM	ABOM	AWUM	NART	RFLM	TXFP	SLEEP	TXRQ
rw													rw	rw	rw

位详细说明

位 31	LOW2：邮箱 2 最低优先级标志（Lowest priority flag for mailbox 2） 当多个邮箱在等待发送报文，且邮箱 2 的优先级最低时，硬件对该位置‘1’。
位 30	LOW1：邮箱 1 最低优先级标志（Lowest priority flag for mailbox 1） 当多个邮箱在等待发送报文，且邮箱 1 的优先级最低时，硬件对该位置‘1’。
位 29	LOW0：邮箱 0 最低优先级标志（Lowest priority flag for mailbox 0） 当多个邮箱在等待发送报文，且邮箱 0 的优先级最低时，硬件对该位置‘1’。 注：当有 1 个邮箱在等待，则 LOW[2:0]被清‘0’。
位 28	TME2：发送邮箱 2 空（Transmil mailbox 2 empty） 当邮箱 2 中没有等待发送的报文时，硬件对该位置‘1’。
位 27	TME1：发送邮箱 1 空（Transmil mailbox 1 empty） 当邮箱 1 中没有等待发送的报文时，硬件对该位置‘1’。
位 26	TME0：发送邮箱 0 空（Transmil mailbox 0 empty） 当邮箱 0 中没有等待发送的报文时，硬件对该位置‘1’。
位 25:24	CODE[1:0]：邮箱号（Mailbox code） 当有至少 1 个发送邮箱为空时，这 2 位表示下一个空的发送邮箱号。 当所有的发送邮箱都为空时，这 2 位表示优先级最低的那个发送邮箱号。
位 23	ABRQ2：邮箱 2 终止发送（Abort request for mailbox 2） 软件对该位置‘1’，可以终止邮箱 2 的发送请求，当邮箱 2 的发送报文被清除时硬件对该位清‘0’。如果邮箱 2 中没有等待发送的报文，则对该位置‘1’没有任何效果。
位 22:20	保留位，硬件强制其值为 0
位 19	TERR2：邮箱 2 发送失败（Transmilssion error of mailbox 2） 当邮箱 2 因为出错而导致发送失败时，对该位置‘1’。
位 18	ALST2：邮箱 2 仲裁丢失（Arbitration lost for mailbox 2） 当邮箱 2 因为出错而导致发送失败时，对该位置‘1’。
位 17	TXOK2：邮箱 2 发送成功（Transmilssion OK of mailbox 2） 每次在邮箱 2 进行发送尝试后，硬件对该位进行更新：

	0：上次发送尝试失败 1：上次发送尝试成功 当邮箱 2 进行发送请求被成功完成后，硬件对该位置‘1’。
位 16	RQCP2：邮箱 2 请求完成（Request completed mailbox 2） 当上次对邮箱 2 的请求（发送或终止）完成后，硬件对该位置‘1’。 软件对该位写‘1’也可以对其清‘0’；当硬件接收到发送请求时也对该位清‘0’。 该位被清‘0’时，邮箱 2 的其它发送状态位（TXOK2、ALST2 和 TERR2）也被清‘0’
位 15	ABRQ1：邮箱 1 终止发送（Abort request for mailbox 1） 软件对该位置‘1’，可以终止邮箱 1 的发送请求，当邮箱 1 的发送报文被清除时硬件对该位清‘0’。如果邮箱 1 中没有等待发送的报文，则对该位置‘1’没有任何效果。
位 14:12	保留位，硬件强制其值为 0
位 11	TERR1：邮箱 1 发送失败（Transmilssion error of mailbox 1） 当邮箱 1 因为出错而导致发送失败时，对该位置‘1’。
位 10	ALST1：邮箱 1 仲裁丢失（Arbitration lost for mailbox 1） 当邮箱 1 因为出错而导致发送失败时，对该位置‘1’。
位 9	TXOK1：邮箱 1 发送成功（Transmilssion OK of mailbox 1） 每次在邮箱 1 进行发送尝试后，硬件对该位进行更新： 0：上次发送尝试失败 1：上次发送尝试成功 当邮箱 1 进行发送请求被成功完成后，硬件对该位置‘1’。
位 8	RQCP1：邮箱 1 请求完成（Request completed mailbox 1） 当上次对邮箱 1 的请求（发送或终止）完成后，硬件对该位置‘1’。 软件对该位写‘1’也可以对其清‘0’；当硬件接收到发送请求时也对该位清‘0’。 该位被清‘0’时，邮箱 1 的其它发送状态位（TXOK1、ALST1 和 TERR1）也被清‘0’
位 7	ABRQ0：邮箱 0 终止发送（Abort request for mailbox 0） 软件对该位置‘1’，可以终止邮箱 0 的发送请求，当邮箱 0 的发送报文被清除时硬件对该位清‘0’。如果邮箱 0 中没有等待发送的报文，则对该位置‘1’没有任何效果。
位 6:4	保留位，硬件强制其值为 0
位 3	TERR0：邮箱 0 发送失败（Transmilssion error of mailbox 0） 当邮箱 0 因为出错而导致发送失败时，对该位置‘1’。
位 2	ALST0：邮箱 0 仲裁丢失（Arbitration lost for mailbox 0） 当邮箱 0 因为出错而导致发送失败时，对该位置‘1’。
位 1	TXOK0：邮箱 0 发送成功（Transmilssion OK of mailbox 0） 每次在邮箱 0 进行发送尝试后，硬件对该位进行更新： 0：上次发送尝试失败 1：上次发送尝试成功 当邮箱 0 进行发送请求被成功完成后，硬件对该位置‘1’。
位 0	RQCP0：邮箱 0 请求完成（Request completed mailbox 0） 当上次对邮箱 0 的请求（发送或终止）完成后，硬件对该位置‘1’。 软件对该位写‘1’也可以对其清‘0’；当硬件接收到发送请求时也对该位清‘0’。 该位被清‘0’时，邮箱 0 的其它发送状态位（TXOK1、ALST1 和 TERR1）也被清‘0’

4.10.4.2 CAN 接收 FIFO 0 寄存器（CAN_RF0R）

地址偏移量：0x0C

复位值：0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留										RFOM0	ROVR0	FULL0	保留	FMP0[1:0]	
res										rs	rc w1	rc w1		r	r

位详细说明

位 31:6	保留位，硬件强制其值为 0
位 5	RFOM0：释放接收 FIFO 0 输出邮箱（Release FIFO 0 output mailbox） 软件通过对该位置‘1’来释放接收 FIFO 的输出邮箱，如果 FIFO 接收为空，那么对该位置‘1’没有任何效果。即只有当 FIFO 中有报文时对该位置‘1’才有意义。如果 FIFO 中有 2 个以上的报文，由于当输出邮箱被释放时，硬件对该位置‘0’。
位 4	FOVR0：FIFO 0 溢出（FIFO 0 overrrun） 当 FIFO 0 已满，又收到新的报文且报文符合过滤条件，硬件对该位置‘1’。 该位由软件清‘0’。
位 3	FULL0：FIFO 0 满（FIFO 0 full） 当 FIFO 0 中有 3 个报文时，硬件对该位置‘1’。该位由软件清‘0’。
位 2	保留位，硬件强制其值为 0
位 1:0	FMP0[1:0]：FIFO 0 报文数目（FIFO 0 message pending） FIFO 0 报文数目，这两位反映了当前接收 FIFO 0 中存放的报文数目。 每当 1 个新的报文被存入接收 FIFO 0,硬件就对 FMP0 加 1。 每当软件对 RFOM0 位写‘1’来释放输出邮箱，FMP0 就被减 1，直到其为 0。

4.10.4.3 CAN 接收 FIFO 1 寄存器（CAN_RF1R）

地址偏移量：0x10

复位值：0x00

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留										RFOM1	ROVR1	FULL1	保留	FMP1[1:0]	
res										rs	rc w1	rc w1		r	r

位详细说明

位 31:6	保留位，硬件强制其值为 0
位 5	RFOM1：释放接收 FIFO 1 输出邮箱（Release FIFO 1 output mailbox） 软件通过对该位置‘1’来释放接收 FIFO 的输出邮箱，如果 FIFO 接收为空，那么对该

	位置‘1’没有任何效果。即只有当 FIFO 中有报文时对该位置‘1’才有意义。如果 FIFO 中有 2 个以上的报文，由于当输出邮箱被释放时，硬件对该位置‘0’。
位 4	FOVR1：FIFO 1 溢出（FIFO 1 overrrun） 当 FIFO 1 已满，又收到新的报文且报文符合过滤条件，硬件对该位置‘1’。 该位由软件清‘0’。
位 3	FULL1：FIFO 1 满（FIFO 0 full） 当 FIFO 0 中有 3 个报文时，硬件对该位置‘1’。该位由软件清‘0’。
位 2	保留位，硬件强制其值为 0
位 1:0	FMP1[1:0]：FIFO 1 报文数目（FIFO 1 message pending） FIFO 1 报文数目，这两位反映了当前接收 FIFO 1 中存放的报文数目。 每当 1 个新的报文被存入接收 FIFO 1,硬件就对 FMP1 加 1。 每当软件对 RFOM1 位写‘1’来释放输出邮箱，FMP1 就被减 1，直到其为 0。

4.10.4.4 CAN 发送邮箱标识符寄存器（CAN_TIxR）（x=0..2）

地址偏移量：0x108 0x190 0x1A0

复位值：0xXXXXXXXX ,X=未定义位（除了第 0 位，复位时 TXRQ=0）

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STID[10:0]/EXID[28:18]											EXID[17:13]				
rw											rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXID[12:0]													IDE	RTR	TXRQ
rw										rw	rw	rw	rw	rw	rw

位详细说明

位 31:21	STID[10:0]/EXID[28:18]:标准标识符或扩展标识符 Standard identifier or extended identifier） 依据 IDE 位的内容，这些位或是标准标识符，或是扩展身份标识的高字节。
位 20:3	EXID[17:0]：扩展标识符（Extended identifier） 扩展身份标识的低字节。
位 2	IDE：标识符选择（identifier extension） 该位决定发送邮箱中报文私用的标识符类型 0：使用标准标识符 1：使用扩展标识符
位 1	RTR：远程发送请求（Remote transmission request） 0：数据帧 1：远程帧
位 0	TXRQ：发送数据请求（Transmit mailbox request） 由软件对其置‘1’，来请求发送邮箱的数据。当数据发送完成，邮箱为空时，硬件对其清‘0’。

4.10.4.5 CAN 时序寄存器（CAN_BTR）

地址偏移量：0x10C

复位值：0x0123 0000 ,注释：当 CAN 处于初始化模式时，该寄存器只能由软件访问

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SILM	LBKM					SJW[1:0]			TS2[2:0]			TS1[3:0]			
rw	rw	res				rw	rw	res	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						BRP[9:0]									
res						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位详细说明

位 31	SILM:静默模式（用于调试） 0：正常状态； 1：允许环回模式。
位 30	LBKM:环回模式（用于调试） 0：禁止环回模式； 1：允许环回模式。
位 29:26	保留位，硬件强制为 0。
位 25:24	SJW[2:0]：重新同步跳跃宽度（Resynchronization jump width） 为了重新同步,该位域定义了 CAN 硬件在每位中可以延长或缩短多少个时间单元的上限。 Trjw=tcan * （sjw[1:0]+1）
位 23	保留位，硬件强制为 0。
位 22:20	TS2[2:0]：时间段 2（Time segment 2） 该位域定义了时间段 2 占用了多少个时间单元 Tbs2=tcan * （ts2[2:0]+1）
位 19:16	TS1[3:0]：时间段 1（Time segment 2） 该位域定义了时间段 1 占用了多少个时间单元 Tbs1=tcan * （ts1[3:0]+1）,关于位时间特性的详细信息，请参考参考手册
位 15:10	保留位，硬件强制为 0。
位 9:0	BRP[9:0]：波特率分频器（Band rate prescaler） 该位域定义了时间单元（tq）的时间长度 Tq=（brq[9:0]+1）*tpclk

4.10.4.6 CAN 波特率计算公式

CAN 波特率=系统时钟/分频数/（1*tq+tBS1+tBS2）

其中 tBS1=tq*(TS1[3:0]+1)

tBS2=tq*(TS2[2:0]+1)

Tq=（BRP[9:0]+1)*tPCLK

Tq 表示一个时间单元

tPCLK=APB 时钟的时间周期

BPR[9:0],TS1[3:0]和 TS2[2:0]在 CAN_BTR 寄存器中定义总体配置保持

tBS1>=tBS2， tBS2>=1 个 CAN 时钟周期， tBS2>=2tSJW

计算波特率要遵循以上的方式和规定，公式看起来比较晦涩难懂，要仔细品味才能逐渐理解，不过在编写程序的时候也有一些简便的方法，我们先做出来了波特率设置表格，把常用的波特率（16 种）设置参数都详细的列出来，大家可以采用查表的方式很容易的查阅和应用。

波特率设置

CAN 波特率	参数设置
50KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=120;
62.5KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=96;
80KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=75;
100KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=60;
125KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=48;
200KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=30;
250KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=24;

400KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_5tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=10;
500KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_3tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=12;
800KBPS	CAN_SJW=CAN_SJW_1tq; CAN_BS1=CAN_BS1_5tq; CAN_BS2=CAN_BS2_2tq; CAN_Prescaler=5;

4.10.4.7 CAN 屏蔽滤波功能

1、屏蔽位模式

在屏蔽位模式下，标识符寄存器和屏蔽寄存器一起，指定报文标识符的任何一位，应该按照“必须匹配”或者“不用关心”处理。

2、标识符列表模式

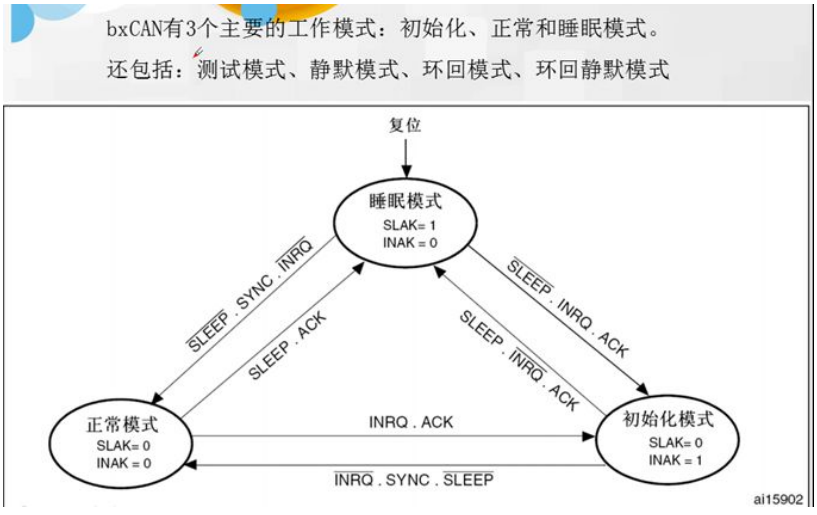
在标识符列表模式下，屏蔽寄存器也被当作标识符寄存器用。因此，不是采用一个标识符加一个屏蔽位的方式，而是使用 2 个标识符寄存器。接收报文标识符的每一位都必须跟过滤器标识符相同。

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。

4.10.4.8 bxCAN 工作模式

bxCAN 有 3 种主要工作模式：初始化、正常和睡眠模式。还包括：测试模式、静默模式、环回模式、环回静默模式。



当上电复位时处于睡眠模式，处于低功耗工作；我们设置成进入初始化模式，程序初始化；接着就进入正常工作模式，处于满负荷工作状态。这三种模式相互之间都可以直接转换，不需要按照顺序执行，这就取决于编写程

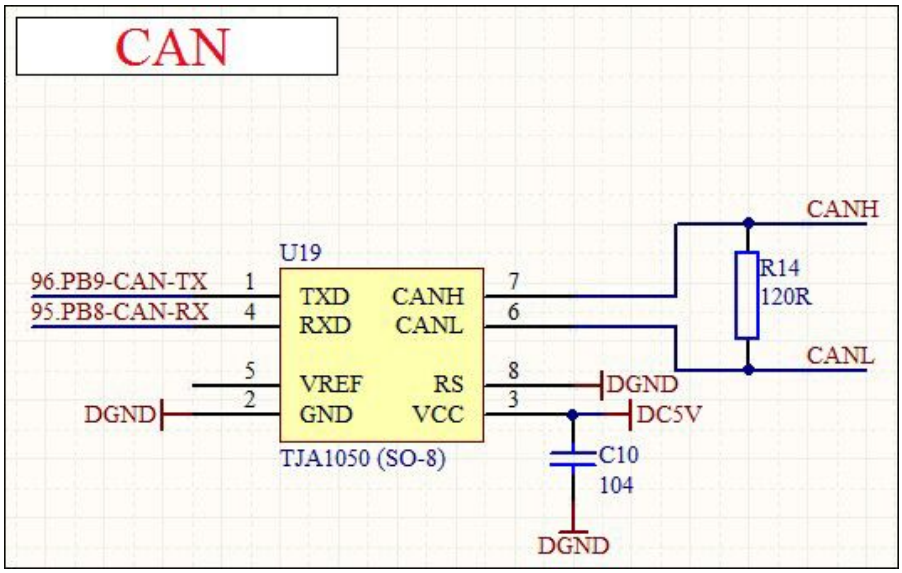
序人员的设计思想。

4.10.5 实验目的

通过编写 CAN 程序，熟悉各种寄存器的使用和程序模块设计，初步了解 CAN 总线的功能。通过本公司的 USB-CAN 转接器把开发板上的数据通过 CAN 通道传送到 PC 机上显示。

4.10.6 硬件设计

利用实验板上的 CAN 通讯电路，可以很方便的实现这个功能。



图一 CAN 通讯应用电路

4.10.7 CAN 总线通讯软件设计

1、打开 CAN 时钟；

2、编写端口映射函数；

3、配置 CAN 相关设置参数

4.10.7.1 STM32 库函数文件

```
stm32f10x_gpio.c
stm32f10x_rcc.c
Misc.c  // 中断控制字（优先级设置）库函数
stm32f10x_can.c  // CAN 通讯函数
```

本节实验及以后的实验我们都是用到库文件，其中 stm32f10x_gpio.h 头文件包含了 GPIO 端口的定义。stm32f10x_rcc.h 头文件包含了系统时钟配置函数以及相关的外设时钟使能函数，所以我们要把这两个头文件对应的 stm32f10x_gpio.c 和 stm32f10x_rcc.c 加到工程中；Misc.c 库函数主要包含了中断优先级的设置，tm32f10x_can.c 库函数主要包含 CAN 通讯设置，这些函数也要添加到函数库中。以上库文件包含了本次实验所有要用到的函数使用功能。

4.10.7.2 自定义头文件

```
pbdata.h
pbdata.c
```

同时我们自己也创建了两个公共的文件，这两个文件主要存放我们自定义的公共函数和全局变量，以方便以后每个功能模块之间传递参数。

4.10.7.3 pbdata.h 文件里的内容是

```
#ifndef _pbdata_H
#define _pbdata_H

#include "stm32f10x.h"
#include "misc.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_tim.h"
#include "stm32f10x_can.h"
#include "sttdio.h"

void RCC_HSE_Configuration(void); //定义函数
void delay(u32 nCount);
void delay_us(u32 nus);
void delay_ms(u16 nms);

#endif
```

语句 #ifndef、#endif 是为了防止 pbdata.h 文件被多个文件调用时出现错误提示。如果不加这两条语句，当两个文件同时调用 pbdata 文件时，

会提示重复调用错误。

4.10.7.4 pbdata.c 文件里的内容是

```
#include "pbdata.h" //很重要，引用这个头文件

void RCC_HSE_Configuration(void) //HSE 作为 PLL 时钟，PLL 作为 SYSCLK
{
   RCC_DeInit(); /*将外设 RCC 寄存器重设为缺省值 */
   RCC_HSEConfig(RCC_HSE_ON);    /*设置外部高速晶振（HSE） HSE 晶振打开(ON)*/

   if(RCC_WaitForHSEStartUp() == SUCCESS) {  /*等待 HSE 起振,  SUCCESS：HSE 晶振稳定且就绪*/

   RCC_HCLKConfig(RCC_SYSCLK_Div1);/*设置 AHB 时钟(HCLK)RCC_SYSCLK_Div1——AHB 时钟 = 系统时*/
   RCC_PCLK2Config(RCC_HCLK_Div1); /*设置高速 AHB 时钟(PCLK2)RCC_HCLK_Div1——APB2 时钟 = HCLK*/
   RCC_PCLK1Config(RCC_HCLK_Div2); /*设置低速 AHB 时钟(PCLK1)RCC_HCLK_Div2——APB1 时钟 = HCLK / 2*/

   RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);/*设置 PLL 时钟源及倍频系数*/
   RCC_PLLCmd(ENABLE);    /*使能 PLL */
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) ; /*检查指定的 RCC 标志位(PLL 准备好标志)设置与否*/

   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);  /*设置系统时钟（SYSCLK） */
   while(RCC_GetSYSCLKSource() != 0x08);      /*0x08：PLL 作为系统时钟 */
 }
}
void delay(u32 nCount)
{
    for(;nCount!=0;nCount--);
}
/*******************************************************************************
* 名    称：delay_us(u32 nus)
* 功    能：微秒延时函数
* 入口参数：u32  nus
* 出口参数：无
* 说    明：
* 调用方法：无
*******************************************************************************/
void delay_us(u32 nus)
```

```
{
    u32 temp;
   SysTick->LOAD = 9*nus;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达

    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}

/*****************************************************************************
* 名    称：delay_ms(u16 nms)
* 功    能：毫秒延时函数
* 入口参数：u16 nms
* 出口参数：无
* 说    明：
* 调用方法：无
*****************************************************************************/
void delay_ms(u16 nms)
{
   u32 temp;
    SysTick->LOAD = 9000*nms;
   SysTick->VAL=0X00;//清空计数器
   SysTick->CTRL=0X01;//使能，减到零是无动作，采用外部时钟源
   do
   {
    temp=SysTick->CTRL;//读取当前倒计数值
    }while((temp&0x01)&&(!(temp&(1<<16))));//等待时间到达
    SysTick->CTRL=0x00; //关闭计数器
    SysTick->VAL =0X00; //清空计数器
}
```

4.10.8 STM32 系统时钟配置 SystemInit()

每个工程都必须在开始时配置并启动 STM32 系统时钟。

4.10.9 GPIO 引脚时钟使能

```
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);//功能复用 IO 时钟使能
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1,ENABLE);//CAN 总线时钟使能
```

本节实验用到了 PB 端口，所以要把 PB 端口的时钟打开；

4.10.10 GPIO 管脚电平控制函数

在主程序中采用 while(1)循环语句，等待 CAN 通讯中断的到来。

```
    while(1);
    {
        //TxMessage.StdId=0xFE00>>5;
        //TxMessage.ExtId=0;
        //TxMessage.IDE=CAN_ID_STD;
        TxMessage.StdId=0;
        TxMessage.ExtId=0xFFFFFFFF>>3;

        TxMessage.IDE=CAN_ID_EXT;
        //数据帧
        TxMessage.RTR=CAN_RTR_DATA;
        //远程帧
        TxMessage.RTR=CAN_RTR_REMOTE;
        TxMessage.DLC=8;

        TxMessage.Data[0]=0x11;
        TxMessage.Data[1]=0x22;
        TxMessage.Data[2]=0x33;
        TxMessage.Data[3]=0x44;
        TxMessage.Data[4]=0x55;
        TxMessage.Data[5]=0x66;
        TxMessage.Data[6]=0x77;
        TxMessage.Data[7]=0x88;
        //CAN 发送数据
        CAN_Transmit(CAN1,&TxMessage);
        delay_ms(1000);
    }
}
```

4.10.11 stm32f10x_it.c 文件里的内容是

```
    #include "stm32f10x_it.h"
    #include "stm32f10x_rcc.h"
    #include "misc.h"
    #include "pbdata.h"

    void USB_LP_CAN1_RX0_IRQHandler(void)
```

```
{
    CanRxMsg RxMessage;
    CanTxMsg TxMessage;

    //CAN 接收
    CAN_Receive(CAN1,CAN_FIFO0,&RxMessage);

    TxMessage.StdId=RxMessage.StdId;
    TxMessage.ExtId=RxMessage.ExtId;
    TxMessage.IDE=RxMessage.IDE;
    TxMessage.RTR=RxMessage.RTR;
    TxMessage.DLC=RxMessage.DLC;

    TxMessage.Data[0]=RxMessage.Data[0];
    TxMessage.Data[1]=RxMessage.Data[1];
    TxMessage.Data[2]=RxMessage.Data[2];
    TxMessage.Data[3]=RxMessage.Data[3];
    TxMessage.Data[4]=RxMessage.Data[4];
    TxMessage.Data[5]=RxMessage.Data[5];
    TxMessage.Data[6]=RxMessage.Data[6];
    TxMessage.Data[7]=RxMessage.Data[7];
    //CAN 发送数据
    CAN_Transmit(CAN1,&TxMessage);
}
```

4.10.12 main.c 文件里的内容是

```
#include "pbdata.h"

void RCC_Configuration(void);
void GPIO_Configuration(void);
void NVIC_Configuration(void);
void CAN_Configuration(void);


int main(void)
{
    CanTxMsg TxMessage;

    RCC_Configuration(); //系统时钟初始化
    GPIO_Configuration();//端口初始化
    NVIC_Configuration();
    CAN_Configuration();//CAN 总线初始化

    while(1);
```

```
    {
        //TxMessage.StdId=0xFE00>>5;
        //TxMessage.ExtId=0;
        //TxMessage.IDE=CAN_ID_STD;
        TxMessage.StdId=0;
        TxMessage.ExtId=0xFFFFFFFF>>3;

        TxMessage.IDE=CAN_ID_EXT;
        //数据帧
        TxMessage.RTR=CAN_RTR_DATA;
        //远程帧
        TxMessage.RTR=CAN_RTR_REMOTE;
        TxMessage.DLC=8;

        TxMessage.Data[0]=0x11;
        TxMessage.Data[1]=0x22;
        TxMessage.Data[2]=0x33;
        TxMessage.Data[3]=0x44;
        TxMessage.Data[4]=0x55;
        TxMessage.Data[5]=0x66;
        TxMessage.Data[6]=0x77;
        TxMessage.Data[7]=0x88;
        //CAN 发送数据
        CAN_Transmit(CAN1,&TxMessage);
        delay_ms(1000);
    }
}

void RCC_Configuration(void)
{
    SystemInit();//72m
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_CAN1,ENABLE);
}

void GPIO_Configuration(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    //端口映射
    GPIO_PinRemapConfig(GPIO_Remap1_CAN1,ENABLE);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8;//RX
```

```
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU;
    GPIO_Init(GPIOB,&GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9;//TX
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF_PP;
    GPIO_Init(GPIOB,&GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;

    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

    NVIC_InitStructure.NVIC_IRQChannel = USB_LP_CAN1_RX0_IRQn; //设置 CAN
中断优先级
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

void CAN_Configuration(void)//重点
{
   CAN_InitTypeDef CAN_InitStructure;
   CAN_FilterInitTypeDef CAN_FilterInitStructure;

   CAN_DeInit(CAN1);
   CAN_StructInit(&CAN_InitStructure);

   //关闭时间触发模式
   CAN_InitStructure.CAN_TTCM=DISABLE;
   //关闭自动离线管理
   CAN_InitStructure.CAN_ABOM=DISABLE;
   //关闭自动唤醒模式
   CAN_InitStructure.CAN_AWUM=DISABLE;
   //禁止报文自动重传
   CAN_InitStructure.CAN_NART=DISABLE;
   //FIFO 溢出时报文覆盖源文件
   CAN_InitStructure.CAN_RFLM=DISABLE;
   //报文发送优先级取决于 ID 号
   CAN_InitStructure.CAN_TXFP=DISABLE;
   //正常的工作模式
```

```
    CAN_InitStructure.CAN_Mode=CAN_Mode_Normal;

    // 设置 CAN 波特率  125 KBPS

    CAN_InitStructure.CAN_SJW=CAN_SJW_1tq;
    CAN_InitStructure.CAN_BS1=CAN_BS1_3tq;
    CAN_InitStructure.CAN_BS2=CAN_BS2_2tq;
    CAN_InitStructure.CAN_Prescaler = 48;


    //初始化 CAN
    CAN_Init(CAN1,&CAN_InitStructure);

    //屏蔽滤波
    CAN_FilterInitStructure.CAN_FilterNumber=0;
    //屏蔽模式
    CAN_FilterInitStructure.CAN_FilterMode=CAN_FilterMode_IdMask;
    //32 位寄存器
    CAN_FilterInitStructure.CAN_FilterScale=CAN_FilterScale_32bit;
    //高 16 位
    CAN_FilterInitStructure.CAN_FilterIdHigh=0x0000;
    //低 16 位
    CAN_FilterInitStructure.CAN_FilterIdLow=0;
    //屏蔽位高 16 位
    CAN_FilterInitStructure.CAN_FilterMaskIdHigh=0x0000;
    //屏蔽位低 16 位
    CAN_FilterInitStructure.CAN_FilterMaskIdLow=0;
    //过滤器 0 关联到 FIFO0
    CAN_FilterInitStructure.CAN_FilterFIFOAssignment=CAN_Filter_FIFO0;
    //使能过滤器
    CAN_FilterInitStructure.CAN_FilterActivation=ENABLE;
    //初始化过滤器
    CAN_FilterInit(&CAN_FilterInitStructure);

    //使能接收中断
    CAN_ITConfig(CAN1,CAN_IT_FMP0,ENABLE);
}
```

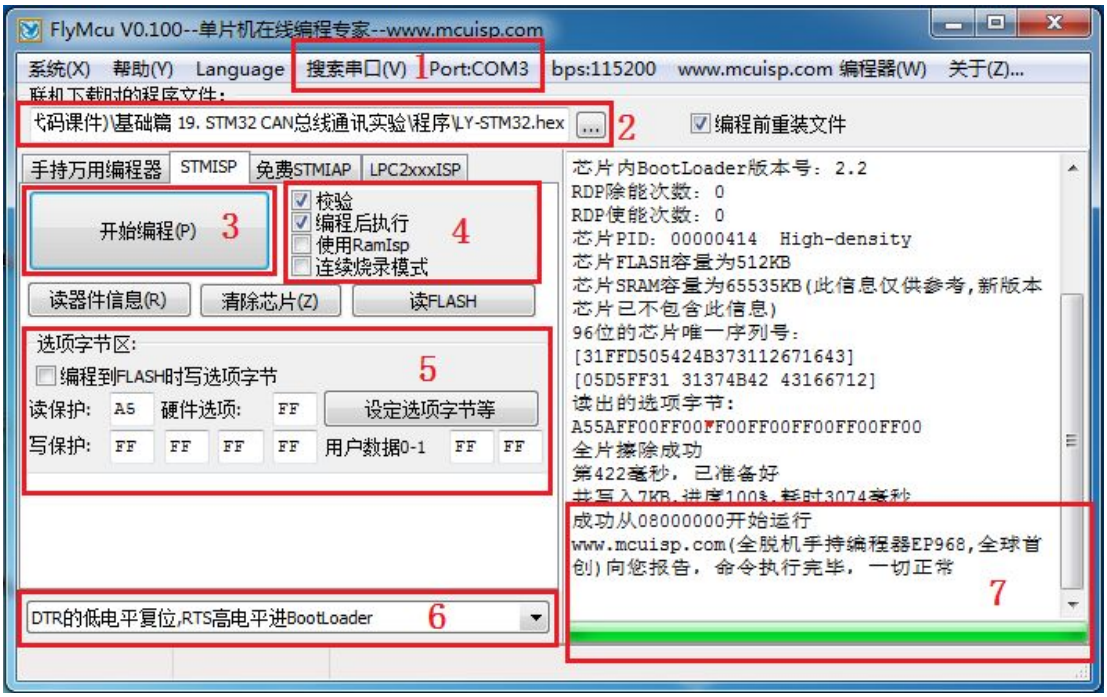
在 main(void)程序体中代码比较多，大部分都是前期课程的延伸，新的知识点不多但比较难，需要初学者好好温习两遍，自己试着编写程序，可以按照自己的想法改变通讯细节，这样理解的会更深刻。

附表 1　方式书写格式说明

序号	CAN 格式	

4.10.13 程序下载

请根据下图所指向的 7 个重点区域配置。其中（1）号区域根据自己机器的实际情况选择，我的机器虚拟出来的串口号是 COM3。（2）号区域请自己选择程序所在的文件夹。（7）号区域当程序下载完后，进度条会到达最右边，并且提示一切正常。（4、5、6）号区域一定要按照上图显示的设置。当都设置好以后就可以直接点击（3）号区域的开始编程按钮下传程序了。



本节实验的源代码在光盘中：（LY-STM32 光盘资料\1.课程\1,基础篇\基础篇 19. STM32 CAN 总线通讯实验\程序）

4.10.14 实验效果图

程序写入实验板后，使用公司开发的多功能监视系统，在 CAN 调试界面中的接收区就会接收到程序发送过来的数据。1#红色框图区域是标准数据帧，2#红色框图区域是扩展数据帧。