SJA1000 独立的 CAN 控制器应用指南

目录

1．介绍

SJA1000 是一个独立的 CAN 控制器，它在汽车和普通的工业应用上有先进的特征。由于它和 PCA82C200 在硬件和软件都兼容，因此它将会替代 PCA82C200。SJA1000 有一系列先进的功能，适合于多种应用，特别在系统优化、诊断和维护方面非常重要。

本文是要指导用户设计基于 SJA1000 的完整的 CAN 节点。同时本文还提供典型的应用电路图和编程的流程图。

2．概述

SJA1000 独立的 CAN 控制器有 2 个不同的操作模式：

- BasicCAN 模式（和 PCA82C200 兼容）；
- PeliCAN 模式

BasicCAN 模式是上电后默认的操作模式。因此，用 PCA82C200 开发的已有硬件和软件可以直接在 SJA1000 上使用，而不用作任何修改。

PeliCAN 模式是新的操作模式，它能够处理所有 CAN2.0B 规范的帧类型。而且它还提供一些增强功能使 SJA1000 能应用于更宽的领域。

2.1 SJA1000 的特征

SJA1000 的特征能分成 3 组：

（1） 已建立的 PCA82C200 功能：这组的功能已经在 PCA82C200 里实现。

（2） 改良的 PCA82C200 功能：这组功能的部份已经在 PCA82C200 里实现。但是，在 SJA1000 里，这些功能在速度、大小和性能方面得到了改良。

（3） PeliCAN 模式的增强功能：在 PeliCAN 模式里，SJA1000 支持一些错误分析功能支持系统诊断、系统维护、系统优化。而且这个模式里也加入了对一般 CPU 的支持和系统自身测试的功能。

SJA1000 所有的特征包括它们在应用中主要的优点都被列在下面的表中。

表 1 SJA1000 应用中的优点

已建立的 PCA82C200 功能

灵活的微处理器接口	允许接口大多数微型处理器或微型控制器。
可编程的 CAN 输出驱动器	对各种物理层的分界面。
CAN 位频率高达 1Mbit/s	SJA1000 覆盖了位频率的所有范围，包括高速应用。

提高的 PCA82C200 功能

CAN2.0B(passive)	SJA1000 的 CAN2.0B passive 特征允许 CAN 控制器接收有 29 位标识符的报文。
64 个字节接收 FIFO	接收 FIFO 可以存储高达 21 个报文，这延长了最大中断服务时间，避免了数据超载。
24MHz 时钟频率	微处理器的访问更快和 CAN 的位定时选择更多。
接收比较器旁路	减少内部延迟，由于改进的位定时编程，使 CAN 总线长度更长。

PeliCAN 模式的增强功能

CAN2.0B（active）	CAN2.0B active 支持带有 29 位标识符的网络扩展应用。
发送缓冲器	有 11 位或 29 位标识符的报文的单报文发送缓冲器。
增强的验收滤波器	两个验收滤波器模式，支持 11 位和 29 位标识符的滤波。
可读的错误计数器	支持错误分析，在原型阶段和在正常操作期间可用于：诊断、系统维护、系统优化。
可编程的出错警告界限	

错误代码捕捉寄存器	
出错中断	
仲裁丢失捕捉中断	支持系统优化包括报文延迟时间的分析。
单次发送	使软件命令最小化和允许快速重载发送缓冲器。
仅听模式	SJA1000 能够作为一个认可的 CAN 监控器操作，可以分析 CAN 总线通信或进行自动位速率检测。
自测试模式	支持全部 CAN 节点的功能自测试或在一个系统内的自接收。

2.2 CAN 节点结构

通常，每个 CAN 模块能够被分成不同的功能块。SJA1000 用使应用[3]、[4]、[5]最优化的 **CAN 收发器**连接到 CAN 总线。收发器控制从 CAN 控制器到总线物理层或相反的逻辑电平信号。

上面一层是一个 **CAN 控制器**，它执行在 CAN 规范[8]里规定的完整是 CAN 协议。它通常用于报文缓冲和验收滤波。

而所有这些 CAN 功能都由一个**模块控制器**控制，它负责执行应用的功能。例如，控制执行器、读传感器和处理人机接口（MMI）。

如图 1 所示，SJA1000 独立的 CAN 控制器通常位于微型控制器和收发器之间，大多数情况下这个控制器是一个集成电路。

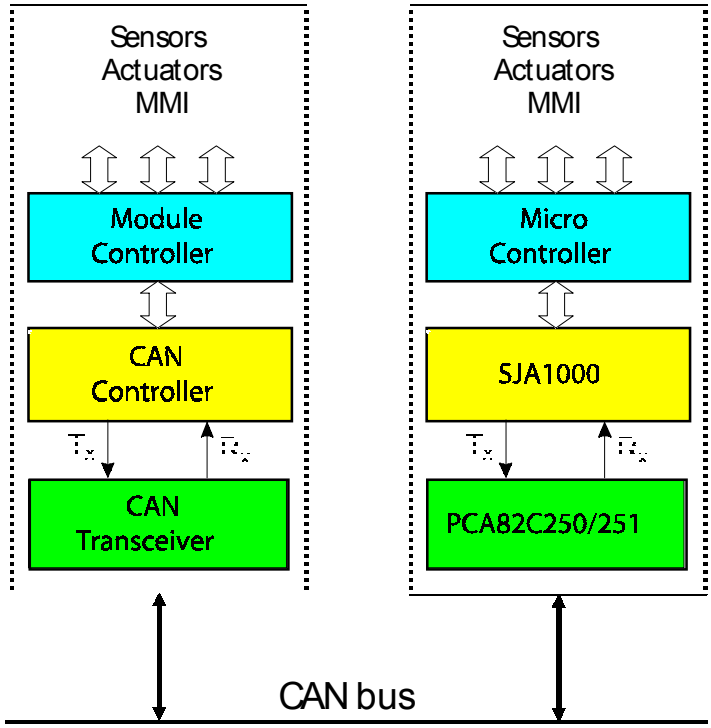


图 1 CAN 模块装置

2.3 结构图

下图是 SJA1000 的结构图。

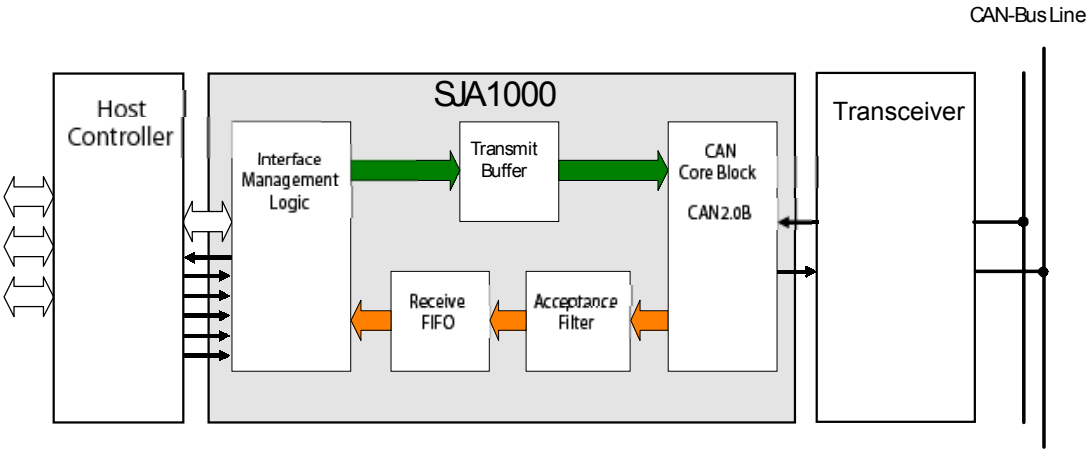


图 2 SJA1000 的结构图

根据 CAN 规范，**CAN 核心模块**控制 CAN 帧的发送和接收。

接口管理逻辑负责连接外部主控制器，该控制器能可以是微型控制器或任何其他器件。经过 SJA1000 复用的地址/数据总线访问寄存器和控制读/写选通信号都在这里处理。另外除了 PCA82C200 已有的 BasicCAN 功能，还加入了一个新的 PeliCAN 功能。因此，附加的寄存器和逻辑电路主要在这块里生效。

SJA1000 的**发送缓冲器**能够存储一个完整的报文（扩展的或标准的）。当主控制器初始化发送，接口管理逻辑会使 CAN 核心模块从发送缓冲器读 CAN 报文。

当收到一个报文时，CAN 核心模块将串行位流转换成用于**验收滤波器的**并行数据。通过这个可编程的滤波器，SJA1000 能确定主控制器要接收哪些报文。

所有收到的报文由验收滤波器验收并存储在**接收 FIFO**。储存报文的多少由工作模式决定，而最多能存储 32 个报文。因为数据超载可能性被大大降低，这使用户能更灵活地指定中断服务和中断优先级。

3．系统

为了连接到主控制器，SJA1000 提供一个复用的地址/数据总线和附加的读/写控制信号。SJA1000 可以作为主控制器外围存储器映射的 I/O 器件。

3.1 SJA1000 的应用

SJA1000 的寄存器和管脚配置使它可以使用各种各样集成或分立的 CAN 收发器。由于有不同的微控制器接口，应用可以使用不同的微控制器。

图 3 所示是是一个包括 80C51 微型控制器和 PCA82C251 收发器的典型 SJA1000 应用。CAN 控制器功能像是一个时钟源，复位信号由外部复位电路产生。在这个例子里，SJA1000 的片选由微控制器的 P2.7 口控制。否则，这个片选输入必须接到 VSS。它也可以通过地址译码器控制，例如，当地址/数据总线用于其他外围器件的时侯。

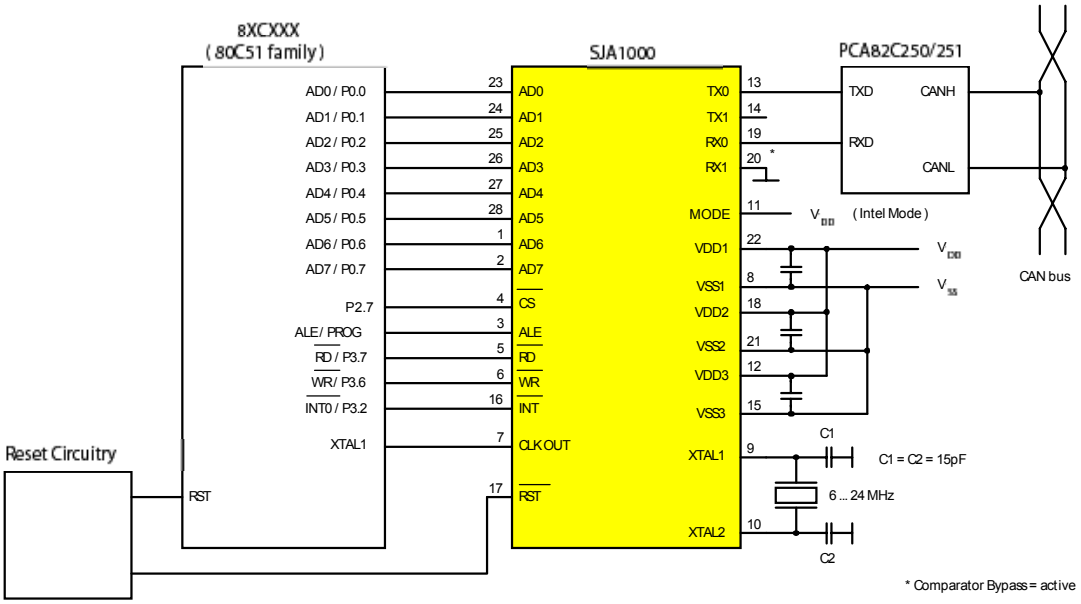


图 3 典型的 SJA1000 应用

3.2 电源

SJA1000 有三对电源引脚，用于 CAN 控制器内部不同的数字和模拟模块。

VDD1/VSS1：内部逻辑 （数字）

VDD2/VSS2：输入比较器 （模拟）

VDD3/VSS3：输出驱动器 （模拟）

为了有更好的 EME 性能，电源应该分隔开来。例如为了抑制比较器的噪声，VDD2 可以用一个 RC 滤波器来退耦。

3.3 复位

为了使 SJA1000 正确复位，CAN 控制器的 XTAL1 管脚必须连接一个稳定的振荡器时钟（见 3.4 节）。引脚 17 的外部复位信号要同步并被内部延长到 15 个 t_{XTAL}。这保证了 SJA1000 所有寄存器能够正确复位（见[1]）。要注意的是上电后的振荡器的起振时间必须要考虑。

3.4 振荡器和时钟策略

SJA1000 能用片内振荡器或片外时钟源工作。另外 CLKOUT 管脚可被使能，向主控制器输出时钟频率。图 4 显示了 SJA1000 应用的四个不同的定时原理。如果不需要 CLKOUT 信号，可以通过置位时钟分频寄存器（Clock Off=1）关断。这将改善 CAN 节点的 EME 性能。CLKOUT 信号的频率可以通过时钟分频寄存器改变：

$f_{CLKOUT} = f_{XTAL}$ / 时钟分频因子（1，2，4，6，8，10，12，14）。

上电或硬件复位后，时钟分频因子的默认值由所选的接口模式（引脚 11）决定。如果使用 16MHz 的晶振，Intel 模式下 CLKOUT 的频率是 8 MHz。Motorola 模式中，复位后的时钟分频因子是 12，这种情况 CLKOUT 会产生 1.33MHz 的频率。

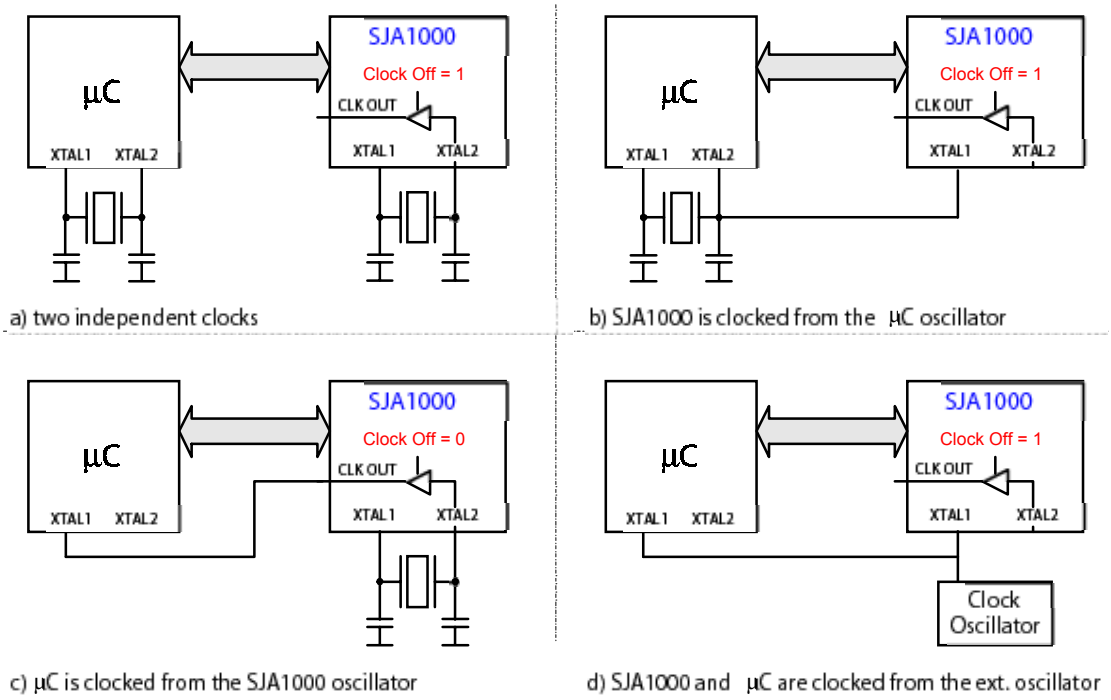


图 4 时钟策略

3.4.1 睡眠和唤醒

置位命令寄存器的进入睡眠位（BasicCAN 模式）或模式寄存器（PeliCAN 模式）的睡眠模式位后，如果没有总线活动和中断等待，SJA1000 就会进入睡眠模式。振荡器在 15 个 CAN 位时间内保持运行状态。此时，微型控制器用 CLKOUT 频率来计时，进入自己的低功耗模式。如果出现三个唤醒条件之中的一个[1]，振荡器会再次启动并产生一个唤醒中断。振荡器稳定后，CLKOUT 频率被激活。

3.5 CPU 接口

SJA1000 支持直接连接到两个著名的微型控制器系列：80C51 和 68xx。通过 SJA1000 的 MODE 引脚可选择接口模式：

Intel 模式： MODE＝高

Motorola 模式： MODE＝低

地址/数据总线和读/写控制信号在 Intel 模式和 Motorola 模式的连接如图 5 所示。Philips 基于 80C51 系列的 8 位微控制器和 XA 结构的 16 位微型控制器都使用 Intel 模式。

为了和其他控制器的地址 / 数据总线和控制信号匹配，必须要附加逻辑电路。但是必须确保在上电期间不产生写脉冲。另一个方法在这个时候使片选输入是高电平，禁能 CAN 控制器。

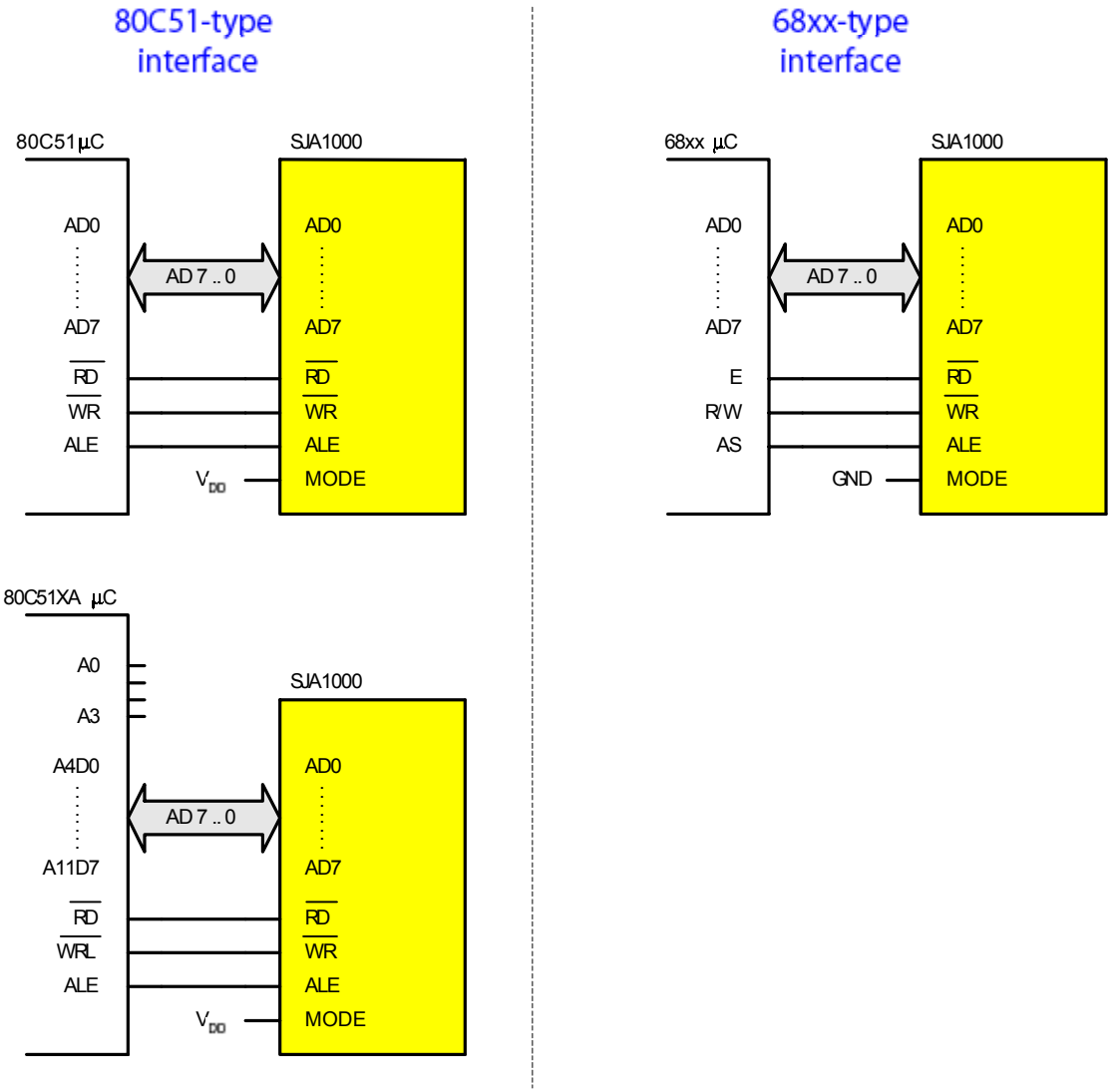


图 5 SJA1000 的 CPU 时钟接口

3.6 物理层接口

为了和 PCA82C200 兼容，SJA1000 包括一个模拟接收输入比较器电路。如果收发器的功能由分立元件实现，就要用刀这个集成的比较器。

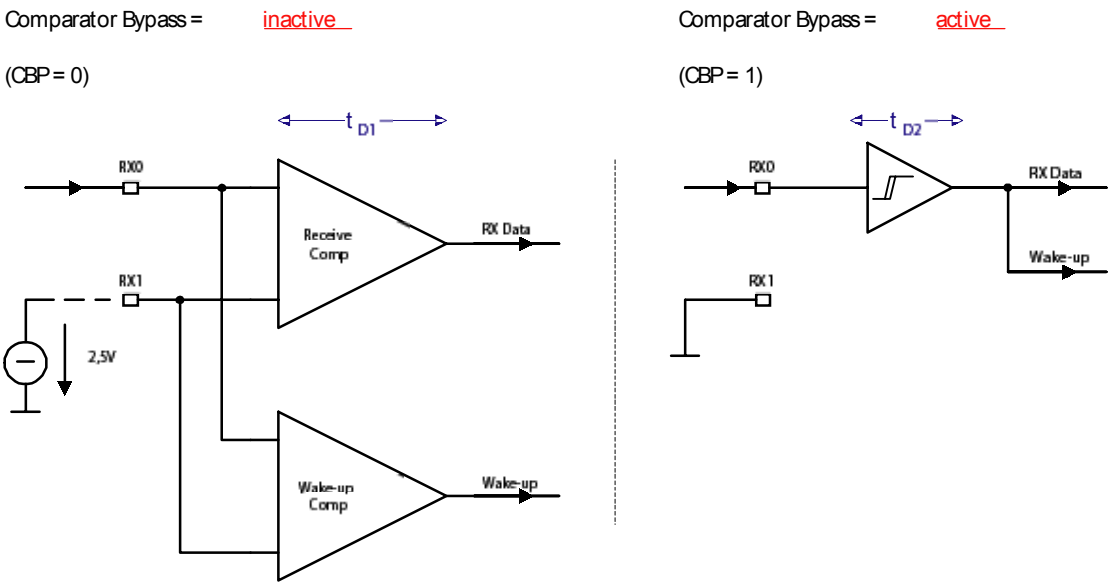


图 6 SJA1000 的接收输入比较器

如果使用外部集成收发器电路，而且没有在时钟分频寄存器里使能比较器旁路功能，RX1 输出要被连接到 2.5V 的参考电压（现有的收发器电路参考电压输出）。图 6 显示了两种设置的相应电路：CBP=激活和 CBP=非激活。另外唤醒信号的通道被下拉。对于使用集成的收发器电路的所有新应用，我们建议激活（使用）SJA1000 的比较器旁路功能（图 7）。如果这个功能被使能，施密特触发器的输入有效，内部的传播延迟 t_{D2} 比接收比较器的延迟 t_{D1} 要小得多。它对最大的总线长度[6]有正面的影响。另外，休眠模式的电流将显著降低。

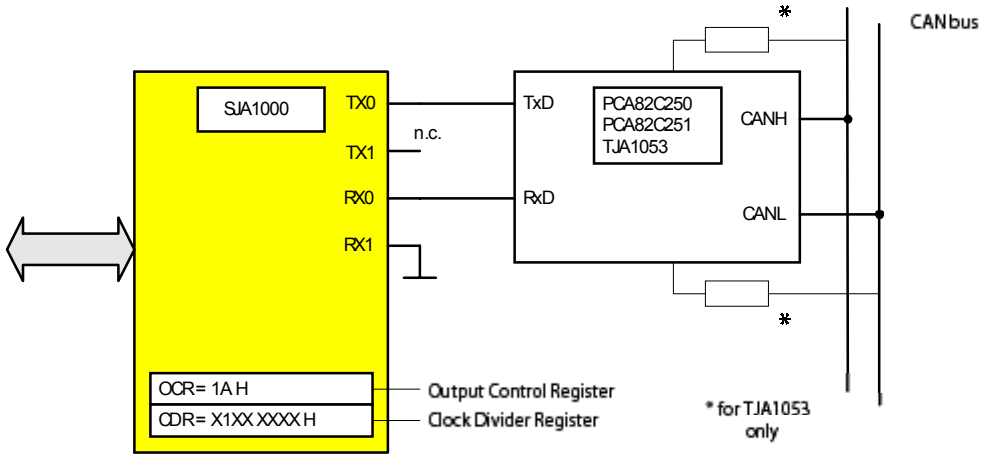


图 7 带有集成收发器电路的标准应用

4．CAN 通迅的控制

4.1 控制 SJA1000 的基本功能和寄存器

SJA1000 的功能配置和行为由主控制器的程序执行。因此 SJA1000 能满足不同属性的 CAN 总线系统的要求。主控制器和 SJA1000 之间的数据交换经过一组寄存器（控制段）和一个 RAM（报文缓冲器）完成。RAM 的部分的寄存器和地址窗口组成了发送和接收缓冲器，对于主控制器来说就象是外围器件寄存器。

表 2 根据它们在系统的作用分组列出了这些寄存器。

注意，一些寄存器只在 PeliCAN 模式有效，控制寄存器就仅在 BasicCAN 模式里有效。而且一些寄存器是只读的或只写的，还有一些只能在复位模式中访问。

关于寄存器的读（和 / 或）写访问、位定义和复位值等更多信息，可在数据表[1]中找到。

表 2 SJA1000 内部寄存器的分类

使用类型	寄存器名称（符号）	寄存器地址		功能
		PeliCAN 模式	BasicCAN 模式	
选择不同的操作模式的要素	模式（MOD）	0	—	选择睡眠模式、验收滤波器模式、自测试模式、只听模式和复位模式
	控制（CR）	—	0	在 BasicCAN 模式里选择复位模式
	命令（CMR）	—	1	BasicCAN 模式的睡眠模式命令
	时钟分频器（CDR）	31	31	在 CLKOUT 设置时钟信号（引脚 7）选择 PeliCAN 模式、比较器旁路模式、TX1（管脚 14）输出模式
设定 CAN 通讯的要素	验收码（ACR） 验收屏蔽（AMR）	16~29 20~23	4， 5	验收滤波器位的模式选择
	总线定时寄存器 0（BTR0） 总线定时寄存器 1（BTR1）	6 7	6 7	位定时参数的设置
	输出控制（OCR）	8	8	输出驱动器属性的选择
	命令（CMR）	1	1	自接收、清除数据超载、释放接收缓冲器、中止传输和传输请求的命令
	状态（SR）	2	2	报文缓冲器的状态、CAN 核心模块的状态
	中断（IR）	3	3	CAN 中断标志。
	中断使能（IER）	4	—	在 PeliCAN 模式使能和禁能中断
	控制（CR）	—	0	在 BasicCAN 模式使能和禁能中断事件
复杂的错误检测和分析的要素	仲裁丢失捕捉（ALC）	11	—	显示仲裁丢失位的位置
	错误代码捕捉（ECC）	12	—	显示最近一次的错误类型和位置
	出错警告界限（EWLR）	13	—	产生出错警告中断的阀值选择
	RX 错误计数（RXERR）	14	—	反映接收错误计数器的当前值
	TX 错误计数（TXERR）	14，15	—	反映发送错误计数器的当前值
	Rx 报文计数器（RMC）	29	—	接收 FIFO 里的报文数量
	RX 缓冲器起始地址（RBSA）	30	—	显示接收缓冲器提供的报文的当前内部 RAM 地址
信息缓冲器	发送缓冲器（TXBUF）	16~28	10~19	
	接收缓冲器（RXBUF）	16~28	20~29	

4.1.1 发送缓冲器/接收缓冲器

要在 CAN 总线上发送的数据被载入 SJA1000 的存储区，这个存储区叫“发送缓冲器”。从 CAN 总线上收到的数据也存储在 SJA1000 的存储区，这个存储区叫“接收缓冲器”。这些缓冲器包括 2，3 或 5 个字节的标识符和帧信息（取决于模式和帧类型），而最多可以包含 8 个数据字节。关于报文缓冲器各位的定义和组成等更多信息，见数据表[1]。

- BasicCAN 模式： 缓冲器长 10 个字节（见表 3）
  - 2 个标识符字节。
  - 最多 8 个数据字节。
- PeliCAN 模式： 这些缓冲器是 13 个字节长（见表 4）
  - 1 字节帧信息。

– 2 个或 4 个标识符字节（标准帧或扩展帧）
– 最多 8 个数据字节。

表 3 BasicCAN 模式里的 RX 和 TX 缓冲器

CAN 地址（十进制）	名称	组成和注释
TX 缓冲器：10 RX 缓冲器：20	标识符字节 1	8 位标识符
TX 缓冲器：11 RX 缓冲器：21	标识符字节 2	3 位标识符，1 位远程传输请求位，4 位数据长度代码，表示数据字节的数量
TX 缓冲器：12~19 RX 缓冲器：22~29	数据字节 1~8	由数据长度代码指明，最多 8 个数据字节

表 4 PeliCAN 模式里的 RX 缓冲器[1]（读访问）和 TX 缓冲器（写访问[2]）

CAN 地址（十进制）	名称	组成和标注
16	帧信息	1 位说明，如果报文包括一个标准帧或扩展帧 1 位远程传输请求位 4 位数据长度码，说明数据字节的数量
17，18	标识符字节 1，2	标准帧：11 位标识符 扩展帧：16 位标识符
19，20	标识符字节 3，4	仅扩展帧：13 个标识符
帧类型 标准帧：19~26 扩展帧：21~28	数据字节 1~8	由数据长度代码说明，最多 8 个数据字节

1．整个接收 FIFO（64 个字节）能通过 CAN 地址 32~95 访问（见 5.1 节）。
2．TX 缓冲器的读访问可通过 CAN 地址 96~108 完成（也见 5.1 节）。

4.1.2 验收滤波器

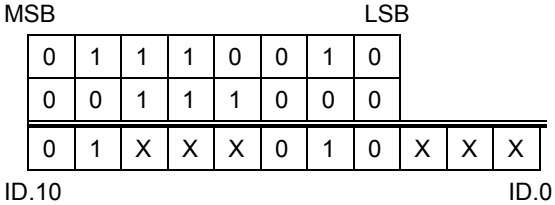
独立的 CAN 控制器 SJA1000 装配了一个多功能的验收滤波器，该滤波器允许自动检查标识符和数据字节。使用这些有效的滤波方法，可以防止对于某个节点无效的报文或报文组存储在接收缓冲器里。因此降低了主控制器的处理负载。

滤波器由验收码寄存器和屏蔽寄存器根据数据表[1]给定算法来控制。接收到的数据会和验收代码寄存器中的值进行逐位比较。接收屏蔽寄存器定义与比较相关的位的位置（0=相关，1=不相关）。只有收到报文的**相应**的位与验收代码寄存器相应的位相同，报文才会被接收。

BasicCAN 模式里的验收滤波

SJA1000 在这个模式可以即插即用地取代 PCA82C200（硬件和软件）。因此验收滤波功能与 PCA82C200 的一样，也可以使用。这个滤波器是由两个 8 位寄存器——验收码寄存器（ACR）和验收屏蔽寄存器（AMR）控制。CAN 报文标识符的高 8 位和这些寄存器里值相比较，见图 8。因此可以定义若干组的标识符为被任何一个节点接收。

例子：
验收码寄存器（ACR）包括：
验收屏蔽寄存器（AMR）包括：
带有 11 位的标识符信息被接收
（X=无关）



在验收屏蔽寄存器里是“1”的位置上，标识符相应的位可以是任何值。这对于三个最低位也一样。因此在这个例子里可以接收 64 个不同的标识符。标识符其他的位必须等于验收代码寄存器相应位的值。

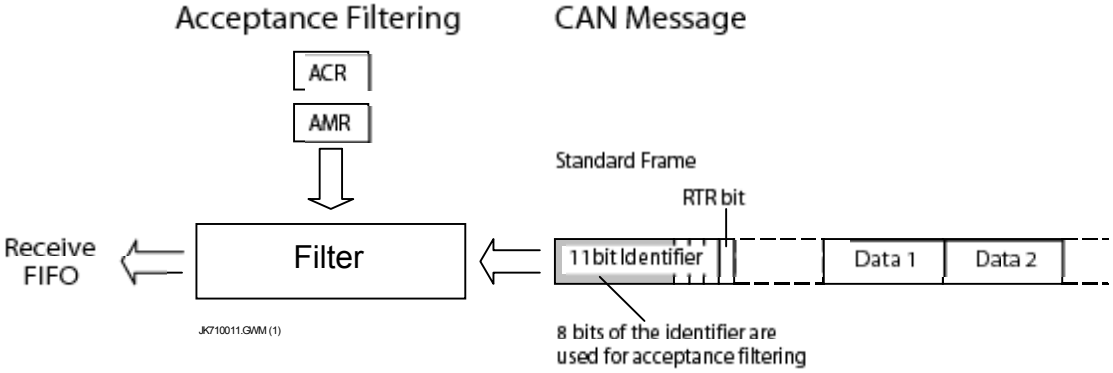


图 8 BasicCAN 模式的验收滤波

PeliCAN 模式的验收滤波

PeliCAN 模式的验收滤波已被扩展：4 个 8 位的验收码寄存器（ACR0，ACR1，ACR2 和 ACR3）和验收屏蔽寄存器（AMR0，AMR1，AMR2 和 AMR3）可以用多种方法滤波报文。如图 9 和图 10 所示，这些寄存器可用于控制一个长的滤波器或两个短的滤波器。报文的哪些位用于验收滤波，取决于收到的帧（标准帧或扩展帧）和选择的滤波器模式（单滤波器或双滤波器）。有关报文的哪些位和验收码和屏蔽位相比较的更多信息请看表 5。从图和表可以看出，标准帧的验收滤波可以包括 RTR 位甚至数据字节。对于不需要经过验收滤波的报文位（例如报文组被定义为接受），验收屏蔽寄存器必须相应的位位置上置“1”。

如果报文不包括数据字节（例如：是一个远程帧或者数据长度码为零）但是验收滤波包括数据字节，则如果标识符直到 RTR 位都有效的话，报文会被接收。

例 1：

假设前面描述的同样的 64 个标准帧报文要在 PeliCAN 模式里滤波，可以通过使用一个长滤波器完成（单滤波器模式）。

验收代码寄存器（ACRn）和验收屏蔽寄存器（AMRn）包括：

n	0	1(高四位)	2	3
ACRn	01XX X010	XXXX	XXXX XXXX	XXXX XXXX
AMRn	0011 1000	1111	1111 1111	1111 1111
接收的报文(ID.28~ID.18,RTR)	01xx x010 xxxx			

（“X”＝不相关，“x”＝任意，只使用了 ACR1 和 AMR1 的高四位）。

在验收屏蔽寄存器是“1”的位置上，标识符相应的位可以是任何值，譬如远程发送请求位和数据字节 1 和 2 的位。

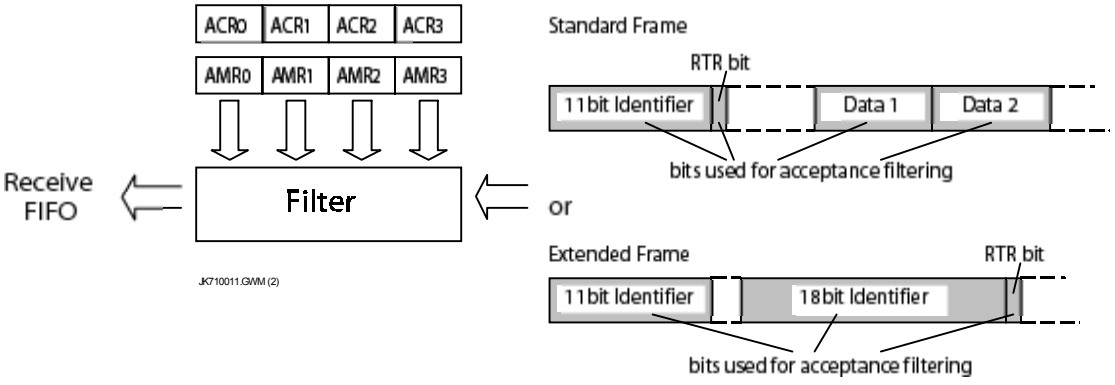


图 9 PeliCAN 模式的验收滤波（单滤波器模式）

例 2：

假设下面 2 个有标准帧标识符的报文在标识符不用进一步译码就被接收。数据和远程帧必须被正确接收。数据字节不要求验收滤波。

报文 1：（ID.28）1***0***11 ***1***100 101（ID.18）

信息 2：（ID.28）1***1***11 ***0***100 101（ID.18）

使用单滤波器模式可以接收到四个报文而不仅是要求的两个：

N	0	1(高 4 位)	2	3
ACRn	1X11 X100	101X	XXXX XXXX	XXXX XXXX
AMRn	0100 1000	0001	1111 1111	1111 1111
接收的报文（ID.28~ID.18，RTR）	1011 0100	101x		
	1111 0100	101x	（报文 2）	
	1011 1100	101x	（报文 1）	
	1111 1100	101x		

（“X”＝不相关，“x”＝任意，只使用了 ACR1 和 AMR1 的高四位）。

这个结果不满足不进一步解码而接收两条信息的要求。

使用双滤波器可以得到正确的结果：

	滤波器 1			滤波器 2	
n	0	1	3 低四位	2	3 高四位
ACRn	1011 1100	101X XXXX	… XXXX	1111 0100	101X …
AMRn	0000 0000	0001 1111	… 1111	0000 0000	0001 …
接收的信息（ID.28~ID.18，RTR）	1011 1100 101X （报文 1）			1111 0100 101X （报文 2）	

（“X”＝不相关，“x”＝任意）。

报文 1 被滤波器 1 接收，报文 2 被滤波器 2 接收。如果报文至少被两个滤波器中的一个接收，报文就被存到接收 FIFO。这种方法可满足于这种要求。

例 3：

在这个例子里，使用一个长的验收滤波器滤波一组带有扩展帧标识符的报文。

n	0	1	2	3（高六位）
ACRn	1011 0100	1011 000X	1100 XXXX	0011 0XXX
AMRn	0000 0000	0001 0001	0000 1111	0000 0111
接收的报文（ID.28~ID.18，RTR）	1011 0100	1011 000x	1100 xxxx	0011 0x

（“X”＝不相关，“x”＝任意，只使用了 ACR1 和 AMR1 的高六位）。

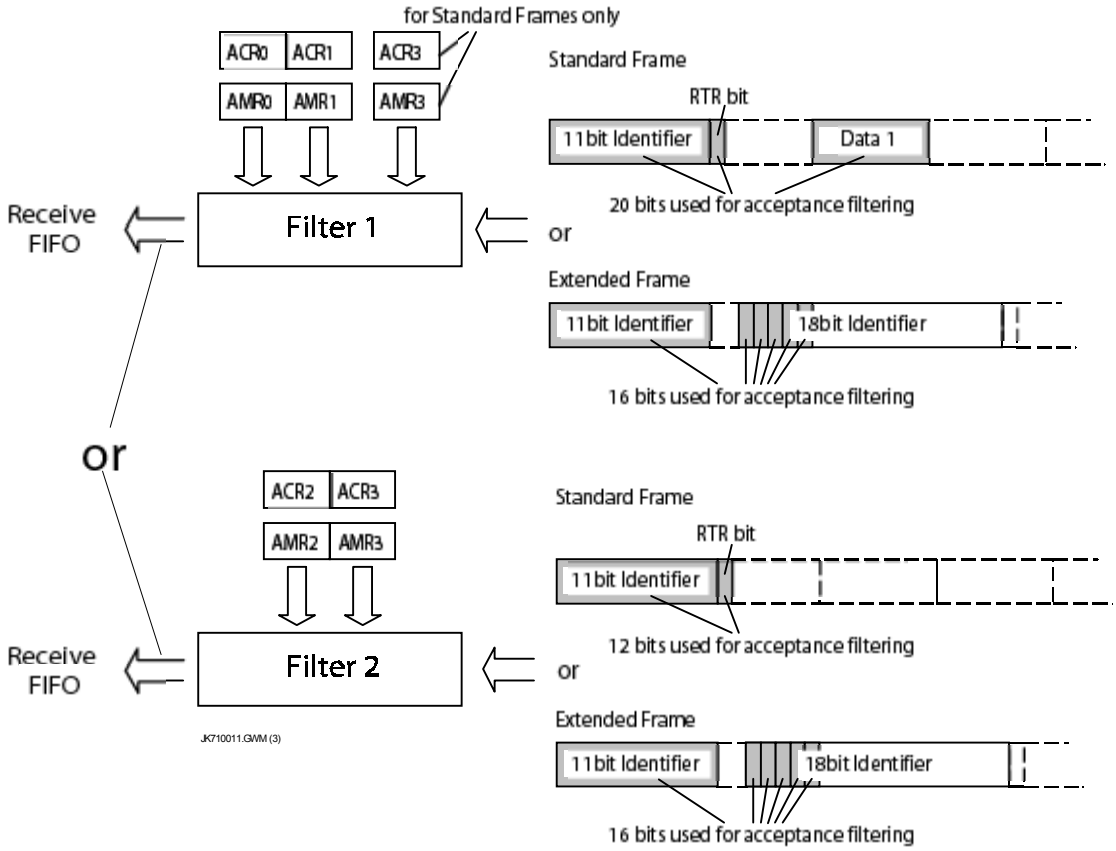


图 10 PeliCAN 模式的验收滤波（双滤波器模式）

例 4：

有些使用标准帧系统仅用 11 位标识符和头两个数据字节识别报文。如果报文超过 8 个数据字节，头两个数据字节定义为报文头和使用分段存储协议就会使用像这样的协议，例如 DeviceNet。对于这种系统类型，SJA1000 除了滤波 11 位标识符和 RTR 位外，在单滤波器模式里能滤波两个数据字节，在双滤波器模式里能过滤一个数据字节（除了 11 位标识符和 RTR 位）。

下面的例子显示了用双滤波器模式，在这种系统里有效地滤波报文：

	滤波器 1								滤波器 2			
n	0		1		…	3 低四位			2		3 高四位	
ACRn	1110	1011	0010	1111	…			1001	1111	0100	XXX0	…
AMRn	0000	0000	0000	0000	…			0000	0000	0000	1110	…
接收的报文	1110	1011	0010	1111	…			1001	1111	0100	xxx 0	
	标识符+RTR		头一个数据字节						标识符		RTR	

（“X”＝不相关，“x”＝任意）。

滤波器 1 滤波的报文有：

- 标识符“11101011001”
- RTR＝“0”，也就是说是数据帧，以及
- 数据字节“11111001”（这是指例如 DeviceNet：一个信息的所有段都被过滤）。

滤波器 2 用来过滤一组 8 个报文，其中报文有：

- 标识符“11110100 000”到“11110100111”，以及
- RTR＝“0”，也就是数据帧。

表 5 PeliCAN 模式的验收滤波器总结

帧类型	单滤波器模式（图 9）	双滤波器模式（图 10）
标准	验收的报文位： - 11 位标识符 - RTR 位 - 第一个数据字节（8 位） - 第二个数据字节（8 位） 使用的验收码寄存器和屏蔽寄存器： - ACR0 或 ACR1/ACR2/ACR3 的高四位 - AMR0 或 AMR1/AMR2/AMR3 的高四位 （接收屏蔽寄存器的未使用的位应设为“1”）	滤波器 1 验收的报文位： - 11 位标识符 - RTR 位 - 第一个数据字节（8 位） 使用的验收码寄存器和屏蔽寄存器： - ACR0/ACR1 或 ACR3 的低四位 - AMR0/AMR1 或 AMR3 的低四位 滤波器 2 用于验收测试的报文位： - 11 位标识符 - RTR 位 使用的验收码寄存器和验收屏蔽寄存器： - ACR2 或 ACR3 的高四位 - AMR2 或 AMR3 的高四位
扩展	用于验收的报文位： - 11 位基本的标识符 - 18 位扩展的标识符 - RTR 位 使用的验收码和验收屏蔽寄存器： - ACR0/ACR1/ACR2 或 ACR3 的高六位 - AMR0/AMR1/AMR2 或 AMR3 的高六位 （验收屏蔽寄存器的未使用的位应设为“1”）	滤波器 1 用于验收的报文位： - 11 位基本标识符 - 扩展标识符的 5 个最高位 使用的验收码和验收屏蔽寄存器： - ACR0/ACR1 和 AMR0/AMR1 滤波器 2 用于测试验收的报文位： - 11 位基本的标识符 - 扩展标识符的 5 个最高位 使用的验收码和验收屏蔽寄存器： - ACR2/ACR3/和 AMR2/AMR3

4.2 CAN 通讯的功能

通过 CAN 总线建立通讯的步骤是：

- 系统上电后
  - 根据 SJA1000 的硬件和软件连接设置主控制器
  - 根据选择的模式、验收滤波、位定时等等设置 CAN 控制器的通讯。这也是在 SJA1000 硬件复位后进行。
- 在应用的主过程中
  - 准备要发送的报文并激活 SJA1000 发送它们。
  - 对被 CAN 控制器接收的报文起作用。
  - 在通讯期间，对发生的错误起作用。

图 11 表示了程序的总体流程。接下来会详细地解说那些直接控制 SJA1000 的流程。

4.2.1 初始化

如上面提到的一样，独立的 CAN 控制器 SJA1000 必须在上电或硬件复位后设置 CAN 通讯。在由主控制器操作期间，它可能会发送一个（软件）复位请求，SJA1000 会被重新配置（再次初始化）。流程图如图 12。本章还会给出一个使用 80C51 派生的微控制器编程的例子。

上电后，主控制器在运行完自己的特殊复位程序后进入 SJA1000 的设置程序。由于图 11 的“configure control lines...”部份和使用的控制器有关，所以这里不作讨论。但是这章的例子主要显示了如何配置一个 80C51 派生器件。

初始化过程的描述见图 12。假设上电后独立 CAN 控制器在管脚 17 得到一个复位脉冲（低电平），使它进入复位模式。在设置 SJA1000 的寄存器前，主控制器通过读复位模式/请求标志来检查 SJA1000 是否已达到复位模式，因为要得到配置信息的寄存器仅在复位模式可写。

在复位模式中，主控制器必须配置下面的 SJA1000 控制段寄存器：

- 模式寄存器（仅在 PeliCAN 模式），为应用选择下面的工作模式：
  - 验收滤波器模式
  - 自我测试模式
  - 仅听模式
- 时钟分频寄存器，定义
  - 使用 BasicCAN 模式还是 PeliCAN 模式
  - 是否使能 CLKOUT 管脚
  - 是否旁路 CAN 输入比较器
  - TX1 输出是否用作专门的接收中断输出
- 验收码寄存器和验收屏蔽寄存器
  - 定义接收报文的验收码
  - 对报文和验收码进行比较的相关位定义验收屏蔽码
- 总线定时寄存器，见[6]
  - 定义总线的位速率
  - 定义位周期内的采样点（位采样点）
  - 定义在一个位周期里采样的数量
- 输出控制寄存器
  - 定义 CAN 总线输出管脚 TX0 和 TX1 的输出模式：正常输出模式、时钟输出模式、双相输出模式或测试输出模式
  - 定义 TX0 和 TX1 输出管脚配置：悬空、下拉、上拉或推挽以及极性。

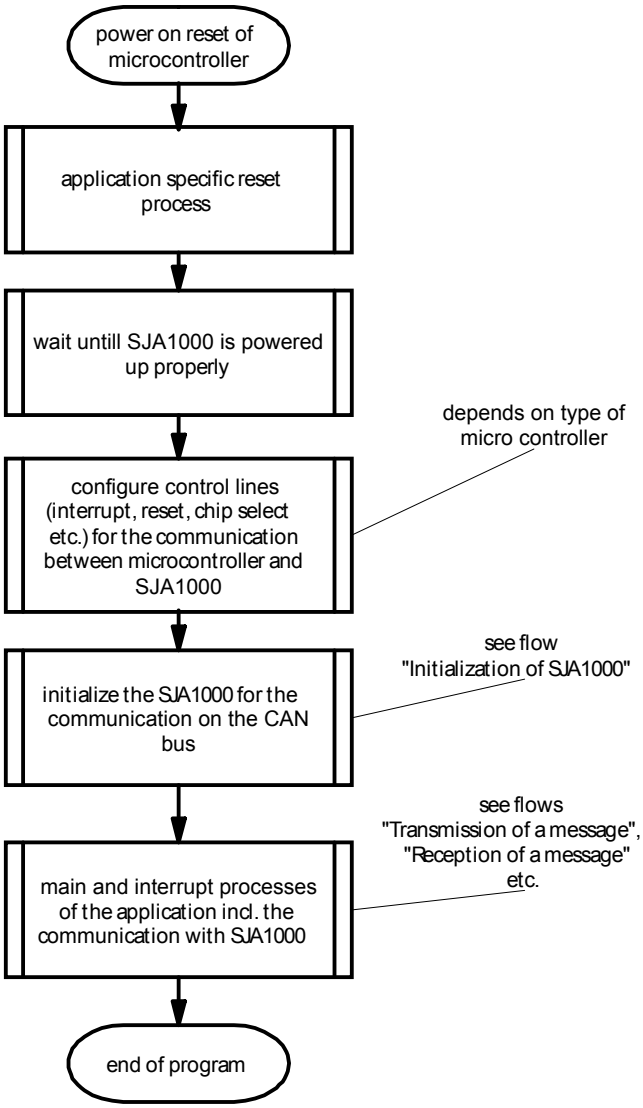


图 11　程序的总体流程图

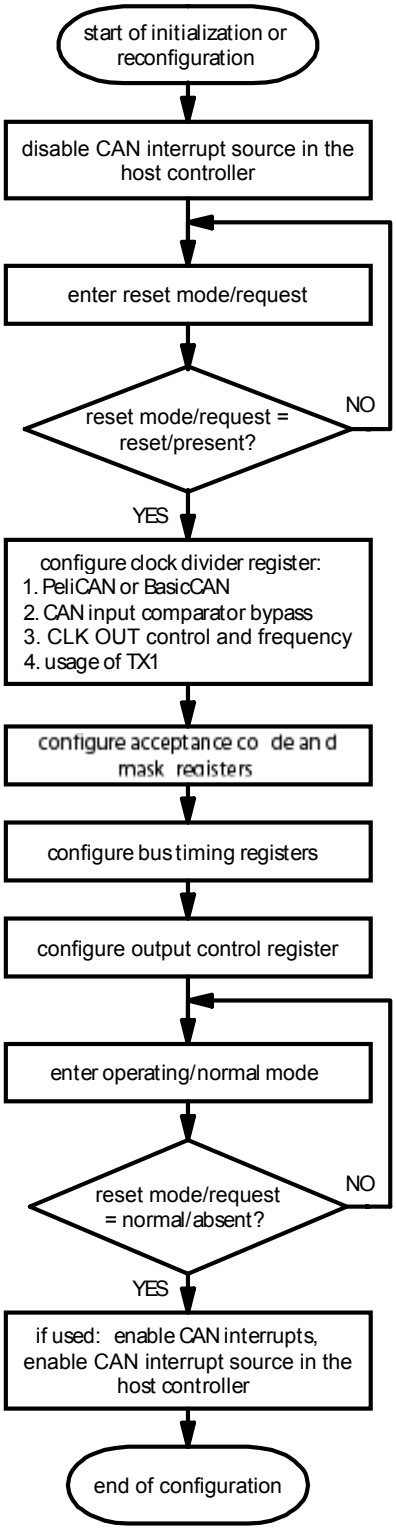


图 12 “SJA1000 的初始化”流程图

在将这个信息发送到 SJA1000 的控制段后，SJA1000 会清除复位模式/请求标志进入工作模式。要必须先检查标志是否确实被清除，是否进入了工作模式才能进行下一步的操作。这通过循环读标志实现。

在硬件复位等待期间（管脚 17 是低电平），不能清除复位模式/请求标志，因为这将迫使复位模式/请求标志变成“复位/存在”（查阅数据表[1]可得到进一步的信息）。因此这个循环是不断尝试清除标志和检查

是否成功离开复位模式。

进入工作模式后，CAN 控制器的中断可被使能（如果适合的话）。

例子：SJA1000 的配置和初始化

这个例子是基于图 3 的应用例子。在下面编程例子里，假设微型控制器 S87C654 是主控制器。它以 SJA1000 输出的时钟作为时钟信号。在上电期间，一个复位电路为微型控制器和 CAN 控制器提供硬件复位信号。在复位[1]期间，SJA1000 的时钟分频寄存器清零。因此 CAN 控制器进入 BasicCAN 模式，且时钟输出使能，当晶振起振后，CAN 控制器能够给 S87C654 传送时钟信号。管脚 11 的时钟频率是 $f_{CLK}/2$，它支持 80C51 系列控制器。收到这个时钟信号后，微控制器启动自己的复位过程（如图 11 所示）。

在附录里，给出了不同的常数和变量等等的定义。变量在 BasicCAN 和 Pelican 模式里的的含意可以不同。例如“InterruptEnReg”在 BasicCAN 模式里是指控制寄存器但在 Pelican 模式里是指中断使能寄存器。编程使用的是 C 语言。

在这个例子里，假设 CAN 控制器要被初始化然后在 PeliCAN 模式里使用。这很容易地就可以从 BasicCAN 模式相应的初始化程序实现。

第一步必须在主控制器和 SJA1000 之间设定一个通讯链路（片选，中断等等），如图 11 的“configure Control lines...”。

```
/*定义中断优先级和控制（电平ー激活，见 4.2.5 章）                               */
PX0=PRIORITY_HIGH；                   /*设 CAN 有一个高优先级中断              */
IT0＝INTLEVELACT；                    /*中断 0 为电平激活                      */

/*使能 SJA1000 的通讯接口                                                      */
CS＝ENABLE_N;                         /*SJA1000 接口使能                       */

/*通讯连接的定义结束－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－ */
```

第二步是初始化 SJA1000 的所有内部寄存器。因为一些寄存器在仅复位模式期间可被写，所以在写入之前必须检查。上电后，SJA1000 被设定为复位模式，如果复位模式已被置位，在循环里面可以检查到。

```
/*中断禁能，如果使用（上电后不需要）                                           */
EA＝DISABLE；                         /*所有中断禁能                           */
SAJIntEn=DISABLE;                     /*来自 SJA100 的外部中断禁能             */

/*设定复位模式/请求位（注：上电后，SJA1000 处于 BasicCAN 模式）
在超时和出现错误信号后跳出循环                                                 */
while((ModeControlReg & RM_RR_Bit)= =ClrByte)
{
/*其他位而不是复位模式/请求位没有改变。                                        */
ModeControlReg = ModeControlReg｜RM_RR_Bit；
}
/*根据图 3 给定的硬件设定时钟分频寄存器
选择 PeliCAN 模式
旁路 CAN 输入比较器作为外部收发器使用
为控制器 S87C654 选择时钟                                                      */
```

```
ClockDiv 标识符 eReg=CANMode_Bit｜CBP_Bit｜DivBy2；
/*如果需要（在上电后总是必须的），CAN 中断禁能，
（写 SJA1000 中断使能/控制寄存器）                                    */
InterruptEnReg=ClrIntEnSJA；

/*定义验收代码和屏蔽                                                  */
AcceptCode0Reg=ClrByte；
AcceptCode1Reg=ClrByte；
AcceptCode2Reg=ClrByte；
AcceptCode3Reg=ClrByte；
AcceptMask0Reg=DontCare；        /*接收任何标识符                    */
AcceptMask1Reg=DontCare；        /*接收任何标识符                    */
AcceptMask2Reg=DontCare；        /*接收任何标识符                    */
AcceptMask3Reg=DontCare；        /*接收任何标识符                    */

/*配置总线定时                                                        */
/*位频率＝1Mbit/s@24MHz，总线被采样一次                               */
BusTiming0Reg=SJW_MB_24｜Prec_MB_24；
BusTiming1Reg=TSEG2_MB_24｜TSEG1_MB_24；

/*配置 CAN 输出：TX1 悬空，TX0 推挽
正常输出模式                                                        */
OutControlReg = Tx1Float｜Tx0PshPull｜NormalMode；

/*离开复位模式/请求，也就是转向操作模式，
S87C654 的中断使能
但 SJA1000 的 CAN 中断禁能，这可以在一个系统里面分别完成           */

/*清除复位模式位，选择双验收滤波器模式
关闭自我测试模式和仅听模式，
清除休眠模式（唤醒）                                                */
do                               /*等待，直到 RM_RR_Bit 清零          */
/*在超时和出现错误后跳出循环                                          */
{
ModeControlReg = ClrByte；
}while((ModeControlReg&RM_RR_Bit) != ClrByte)；

SJAIntEn = ENABLE；              /*SJA1000 的外部中断使能             */
EA       = ENABLE；              /*所有中断使能                       */

/*------------------------------ SJA1000 初始化例子的结束 ----------------------------------------------*/
```

4.2.2 传输

根据 CAN 协议规范[8]，报文的传输由 CAN 控制器 SJA1000 独立完成。主控制器必须将要发送传送

到发送缓冲器的报文，然后将命令寄存器里的“发送请求”标志置位。发送过程可由 SJA1000 的中断请求控制或由查询控制段的状态标志控制。

中断控制的发送

根据图 13 给出的控制器的主要过程，CAN 控制器的发送中断以及为和 SJA1000 通讯主控制器使用的外部中断使能而且优先级高于启动发送（也由中断控制）。中断使能标志是位于 BasicCAN 模式的控制寄存器和 PeliCAN 模式的中断使能寄存器（见表 2 及[1]）。

当 SJA1000 正在发送报文时，发送缓冲器被写锁定。所以在放置一个新报文到发送缓冲器之前，主控制器必须检查状态寄存器（见[1]）的“发送缓冲器状态”标志（TBS）。

- 发送缓冲器被锁定：

  主控制器将新报文暂时存放在它自己的存储器里并设置一个标志，表示一个报文正在等待发送。如何处理可以设计来保存几个要发送的报文的临时存储计器是软件设计者的问题。启动传输报文会在中断服务程序中处理，程序在当前运行的发送末端被初始化。

  从 CAN 控制器收到中断（见图 13 的中断处理过程）后，主控制器会检查中断类型。如果是发送中断，它会检查是否有更多的报文要被发送。一个正在等待的报文会从从临时存储器复制到发送缓冲器，表示要发送更多信息的标志被清除。置位命令寄存器（见[1]）的“发送请求”（TR）标志，使 SJA1000 启动发送。

- 发送缓冲器被释放：

  主控制器将新报文写入发送缓冲器并置位命令寄存器（见[1]）的 “发送请求”（TR）标志，这将使 SJA1000 启动发送。在发送成功结束时，CAN 控制器产生会一个发送中断。

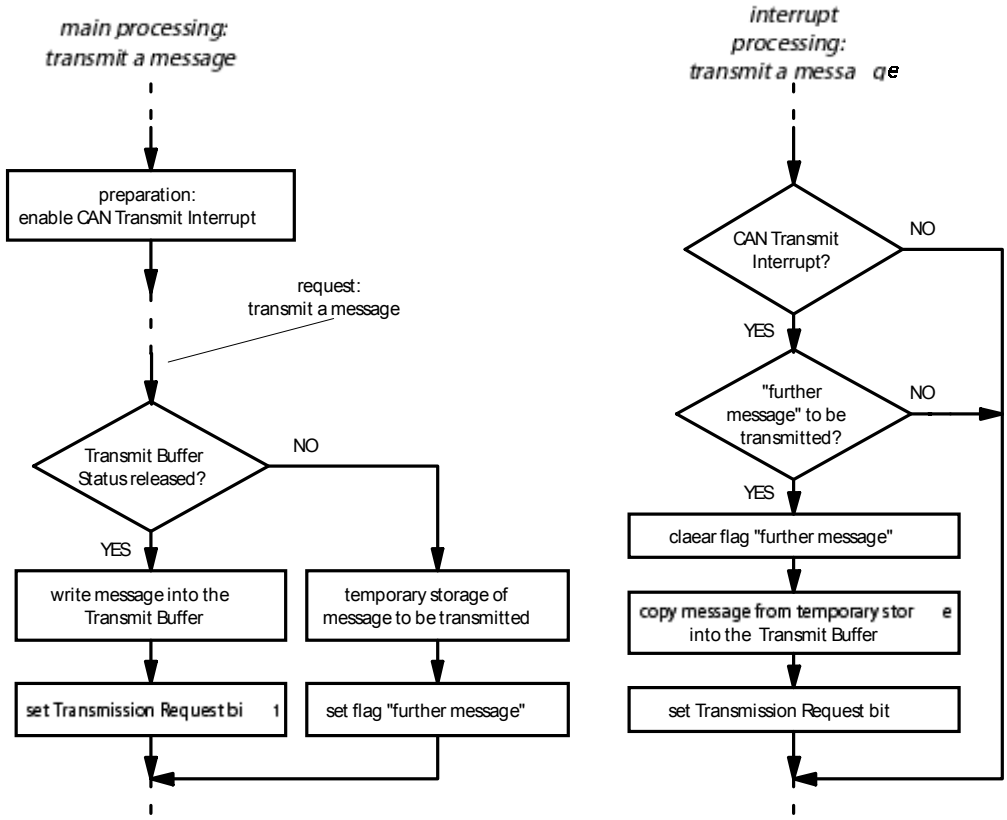


图 13 “发送一个报文”的流程图（中断控制）

查询控制的发送

流程如图 14 所示。CAN 控制器的发送中断在这类传输控制中禁能。

只要 SJA1000 正在发送报文，发送缓冲器就被写锁定。因此在将新报文放入发送缓冲器之前，主控制器必须检查状态寄存器（见[1]）的“发送缓冲器状态”标志（TBS）。

- 发送缓冲器被锁定：
  周期查询状态寄存器，主控制器等待，直到发送缓冲器被释放。
- 发送缓冲器被释放：
  主控制器将新的报文写入发送缓冲器并置位命令寄存器（见[1]）的“发送请求”（TR）标志，此时 SJA1000 将启动发送。

PeliCAN 模式的例子：

在附录里，给出了不同的常数和变量等等的定义。变量在 BasicCAN 和 Pelican 模式里的的含意可以不同。例如“InterruptEnReg”在 BasicCAN 模式里是指控制寄存器但在 Pelican 模式里是指中断使能寄存器。编程使用的是 C 语言。

根据 4.2.1 节给出的例子，初始化 CAN 控制器后，可启动正常的通讯：

```
:
/*等待，直到发送缓冲器被释放                                          */
Do
{
/*等待时，启动查询定时器并运行一些任务
在超时和出现错误后跳出循环                                          */
}while((statusReg & TBS_Bit) != TBS_Bit);

/*释放发送缓冲器，信息可写入缓冲器                                    */
/*在这个例子里，会发送一个标准帧信息                                  */
TxFrameInfo = 0x08;                 /*SFF(data)，DLC＝8             */
TxBuffer1   = 0xA5;                 /*标识符 1 ＝A5，（1010，0101）  */
TxBuffer2   = 0x20;                 /*标识符 2 ＝20，（0010，0000）  */
TxBuffer3   = 0x51;                 /*data1 =51                     */
 •
 •
TxBuffer10  = 0x58;                 /*data8 =58                     */

/*启动发送                                                          */
 CommandReg   =TR_Bit;              /*置位发送请求位                */
 •
 •
```

状态寄存器的 TS 和 RS 标志能检测 CAN 控制器是否已达到空闲状态。TBS 标志和 TCS 标志可以检查是否成功发送。

BasicCAN 模式的例子：

在附录里，给出了不同的常数和变量等等的定义。变量在 BasicCAN 和 Pelican 模式里的的含意可以不同。例如“InterruptEnReg”在 BasicCAN 模式里是指控制寄存器但在 Pelican 模式里是指中断使能寄存

器。编程使用的是 C 语言。

根据 4.2.1 节给出的例子，初始化 CAN 控制器后，可启动正常的通讯：

```
:
/*等待，直到发送缓冲器被释放                                          */
Do
{
/*等待时，启动查询定时器并运行一些任务
在超时和出现错误后跳出循环                                          */
}while((StatusReg & TBS_Bit) != TBS_Bit);

/*发送缓冲器被释放，信息可写入缓冲器                                  */
/*BasicCAN 模式里只有标准帧信息。                                    */
TxBuffer1    = 0xA5;          /* 标识符 1＝A5，（1010，0101）       */
TxBuffer2    = 0x28;          /* 标识符 2＝28，（0010，0000）（DLC＝8）*/
TxBuffer3    = 0x51;          /* data1=51                           */
•
TxBuffer10   = 0x58;          /* data8=58                           */
/*启动发送                                                          */
CommandReg = TR_Bit;          /*置位发送请求位                      */
•
```

TBS－和 TCS－标志用于检查是否成功发送。

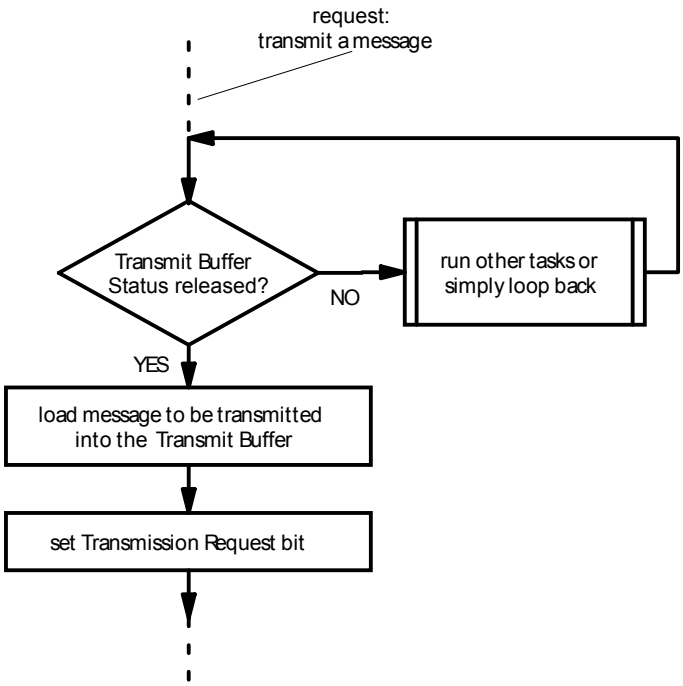


图 14 “发送一个报文”的流程图（查询控制）

4.2.3 中止发送

一个已经请求发送的报文，可通过置位命令寄存器[1]的相应位执行“中止发送”命令中止发送。这个

功能可用于例如：发送一个比现在的报文更紧急的报文，而这个报文已被写入发送缓冲器，但是直到现在没有被成功地发送。

图 15 显示了一个使用发送中断的流程。这个流程显示了为了发送更高优先级的报文而中止当前发送的报文的情况。不同原因的中止报文发送要求不同的中断流程。

一个相应的流程能从查询控制发送的处理中得到。

以免报文由于不同的原因仍然等待处理，发送缓冲器会锁定（见图 15 的主流程图）。如果要求发送一个紧急报文，置位命令寄存器里中止发送位。当这条等待处理的报文已被成功的发送或中止后，发送缓冲器被释放，同时产生一个发送中断被。在中断流程，要检查状态寄存器的发送完成标志，确定前面的发送是否成功。状态“未完成”表示发送被中止。在这种情况下，主控制器要运行一个特殊程序来处理中止发送，例如：在检查后重复发送中止的报文（如果它仍然有效的话）。

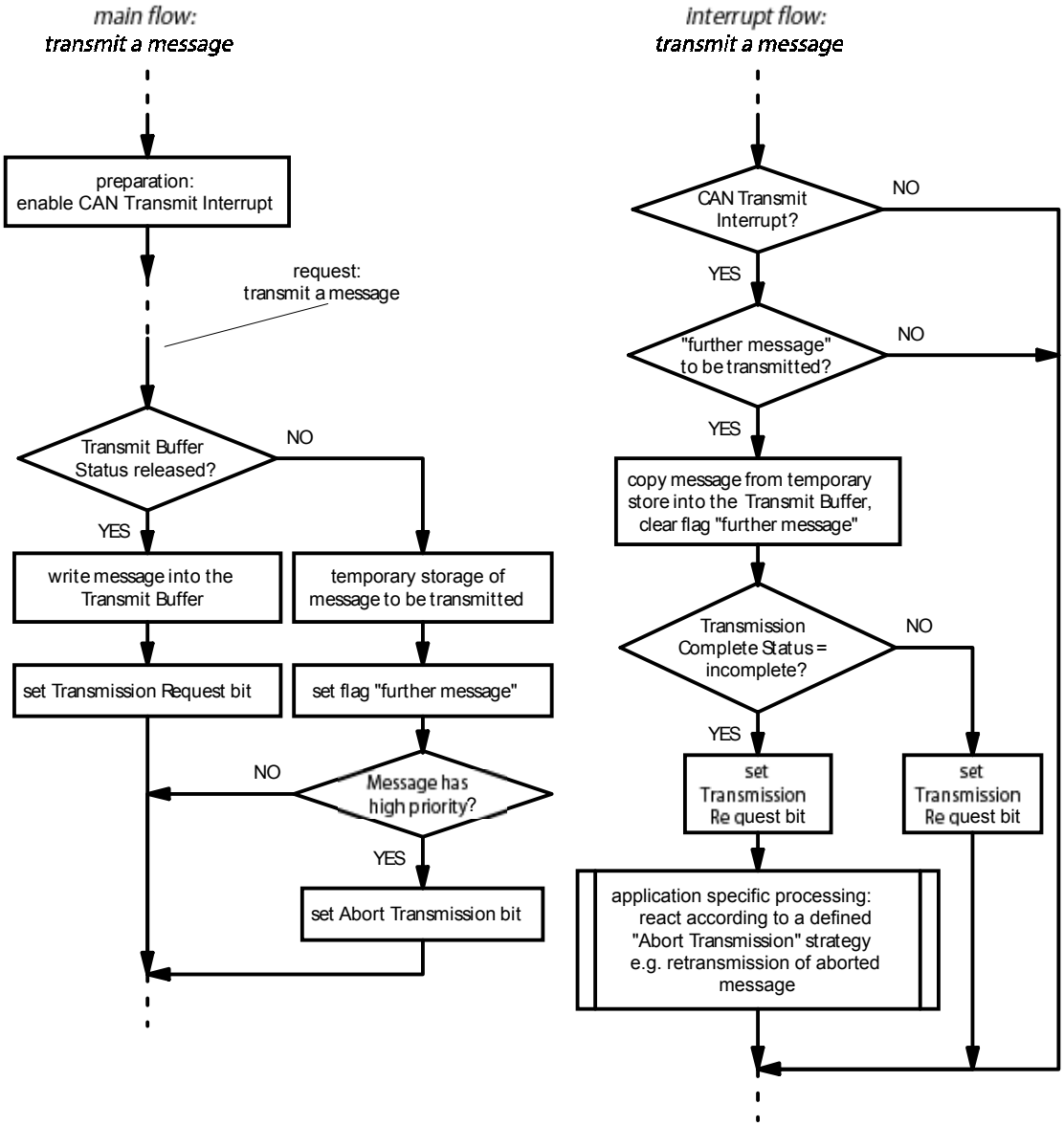


图 15 “中止发送一个报文”的流程图（中断控制）

4.2.4 接收

根据 CAN 协议规范[8]，报文的接收由 CAN 控制器 SJA1000 独立完成。收到的报文放在接收缓冲器(见 4.1.1 和 5.1)。可以发送给主控制器的报文，由状态寄存器的接收缓冲器状态标志“RBS”（见[1]）和接收

中断标志“RI”（见[1]）标出（如果使能）。主控制器会将这条信息发送到本地的报文存储器，然后释放接收缓冲器并对报文操作。发送过程能被 SJA1000 的中断请求或查询 SJA1000 的控制段状态标志来控制。

查询控制接收

流程如图 16 所示，CAN 控制器在这种接收类型下接收中断禁能。

主控制器如常读 SJA1000 的状态寄存器，检查接收缓冲状态标志（RBS）看是否收到一个报文。这些标志的定义位于控制段的寄存器（见[1]）。

- 接收缓冲器状态标志表示“空”，也就是没有收到报文。
  主控制器继续当前的任务直到收到检查接收缓冲器状态的新请求。

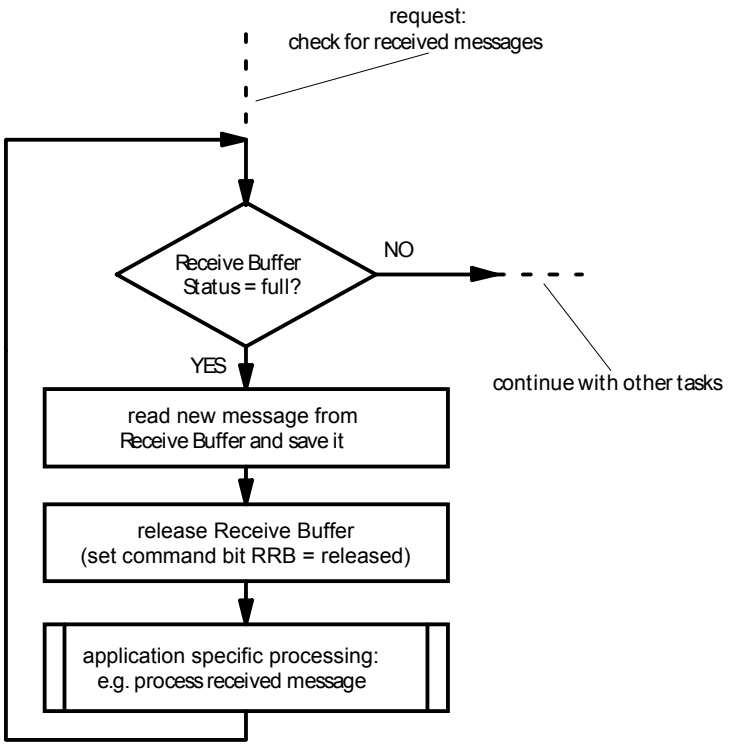


图 16 “接收一个报文”的流程图（查询控制）

- 接收缓冲器状态标志表示“满”，也就是说收到一个或多个报文：
  主控制器从 SJA1000 得到第一个报文，然后通过置位命令寄存器的相应位，发送一个释放接收缓冲器命令。如图 16 所示，主控制器在检查更多信息报文前可以处理每个收到的报文。但也可以通过再次查询接收缓冲器状态位立即检查更多报文，并将在以后一起处理所有收到的报文。在这种情况下，主控制器的本地报文存储器必须足够大，可以存储多于一条报文。在已经发送和处理一条或所有报文后，主控制器继续执行其他的任务。

中断控制接收

根据图 17 给出的控制器的主要过程，CAN 控制器的接收中断以及为和 SJA1000 通迅主控制器的外部中断使能而且优先级高于中断控制报文。中断使能标志位于控制寄存器里（对于 BasicCAN 模式）或位于中断使能寄存器里（对于 PeliCAN 模式）——见表 2 和[1]。

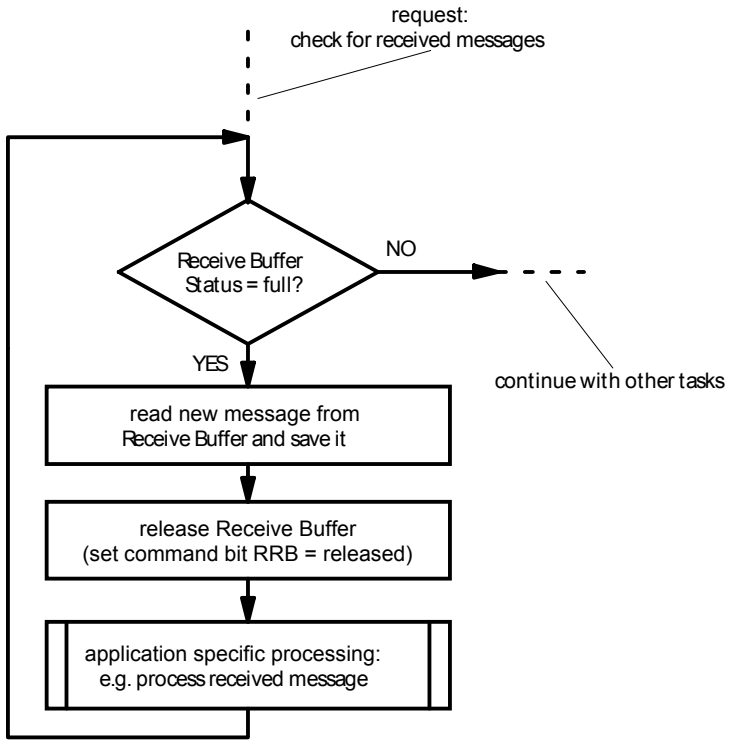


图 17 “接收一个报文”的流程图（中断控制）

如果 SJA1000 已接收一个报文，而且报文已通过验收滤波器并放在接收 FIFO，那么会产生一个接收中断。因此主控制器能立刻作用，将收到的报文发送到自己的报文存储器，然后通过置位命令寄存器的相应标志“RRB”（见[1]）发送一个释放接收缓冲器命令。接收 FIFO 里的更多报文将产生一个新的接收中断，因此不可能将所有在接收 FIFO 中的有效信息在一个中断周期内读出。和这个方法相反，图 18 显示了一个将所有信息一次读出的过程。在释放了接收缓冲器后，SJA1000 会检查状态寄存器中接收缓冲器状态（RBS）看是否有更多报文，而所有有效的信息都会被循环读出。

如图 17 所示，整个接收过程在一个中断程序中完成，而且和主程序没有相互作用。如果可行的话，报文的处理甚至也可以在中断程序里完成。

例子：

在附录里，给出了不同的常数和变量等等的定义。变量在 BasicCAN 和 Pelican 模式里的的含意可以不同。例如“InterruptEnReg”在 BasicCAN 模式里是指控制寄存器但在 Pelican 模式里是指中断使能寄存器。编程使用的是 C 语言。

根据 4.2.1 节给出的例子，初始化 CAN 控制器后，可启动正常的通讯：

1．部分主程序

```
:
/*接收中断使能                                                    */
InterruptEnReg＝RIE_Bit;
:
```

2．中断 0 服务程序的部分程序

```
:
/*从 SJA1000 读中断寄存器的内容并临时保存，所有中断标志被清除（在 PeliCAN 模式里，
接收中断（RI）被首先清除，当给出释放缓冲器命令时）
*/
```

```
CANInterrupt = InterruptReg;
 •
 •
/*检查接收中断和读一个或所有接收到的信息                                    */
 If    (RI_VarBit ==YES)                    /*检测到接收中断                 */
{
      /*从 SJA1000 得到接收缓冲器的内容，并将它存入控制器的内部存储器，
      可以立刻对帧信息和数据长度代码解码并适当地取出。                       */
       •
       •
      /*释放接收缓冲器，接收中断标志被清除，新的信息将产生一个新中断          */
      CommandReg=RRB_Bit;                  /*释放接收缓冲器                 */
}
:
```

数据超载处理

在接收 FIFO 满了但还接收其他报文的时侯，就会通过置位状态寄存器中的数据超载状态位（如果使能）通知主控制器有数据超载的情况，SJA1000 会产生一个数据超载中断。

如果运行在数据超载的状态下，由于主控制器没有足够的时间及时从接收缓冲器取收到的报文而变得极度超载。一个表示数据丢失的数据超载信号，可能会导致系统矛盾。通常一个系统应该设计成：收到的信息要被足够快地传输和处理避免产生数据超载。如果数据超载不能避免，那么主控制器应该执行一个特殊的处理程序来处理这些情况。

图 18 是有关处理数据超载中断的程序流程。

在已经传输这条报文后（该报文产生接收中断并释放接收缓冲器），会通过读接收缓冲器状态来检查在接收 FIFO 中是否还有有效报文。因此在继续下一步之前，所有的信息都能从接收 FIFO 取出。当然在中断中读一条报文并且处理它（可能的话），要比 SJA1000 接收一条新报文更快点。否则主控制器将一直在中断里读报文。

检测到数据超载后，可以根据“数据超载”策略启动一个异常处理。这个策略可以在两种情况下决定：

- 数据超载和接收中断一起发生：信息可能已经丢失。
- 数据超载发生时，没有检测到接收中断：信息可能已经丢失，接收中断可能禁能。

主控制器怎样对这些情况采取相应的动作由系统设计者决定。

相应的处理也可以在查询控制报文接收中处理。

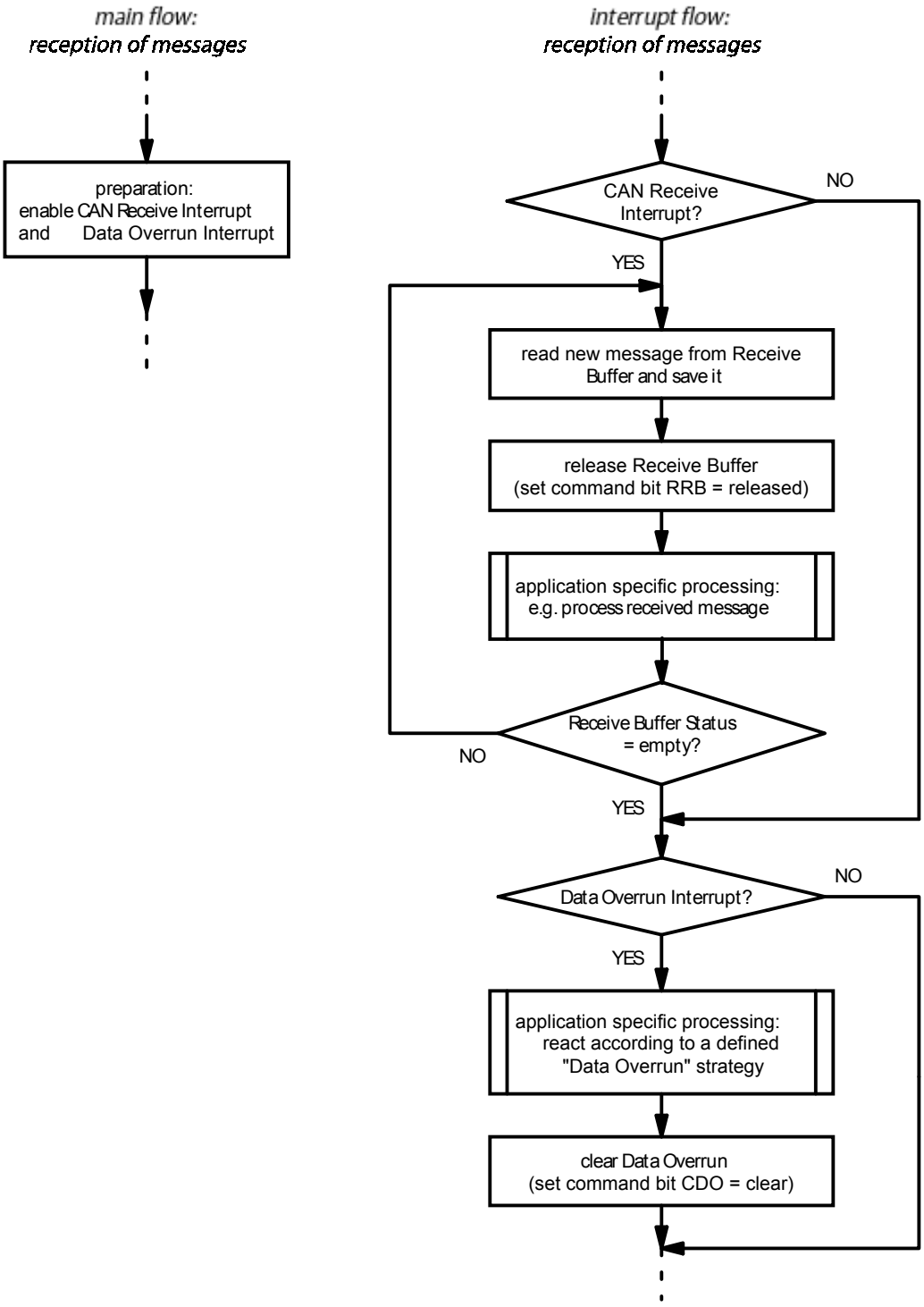


图 18 “数据超载和接收报文”的流程图（中断控制）

4.2.5 中断

在 PeliCAN 模式里，SJA1000 有 8 个不同的中断（在 BasicCAN 模式里仅有 5 个），这些中断可使主控制器立即作用在 CAN 控制器的某些状态上。

一旦 CAN 产生中断，SJA1000 就将中断输出（管脚 16）设为低电平，直到主控制器通过读 SJA1000 的中断寄存器对中断采取相应措施；或在 PeliCAN 模式里，释放接收缓冲器后产生接收中断。在主控制器

这个动作后，SJA1000 将输出中断跳到高电平。如果这段时间有更多中断，或接收 FIFO 里有更多有效报文，SJA1000 立刻将中断输出再次设为低电平。因此输出仅在很短的时间里保持高电平。处理中断请求的握手信号和在两个中断之间的高电平脉冲要求主控制器的中断由电平触发。

图 19 的流程给出所有可能中断的概述，详细的描述可以参考本应用指南。在这个流程里不同的中断被处理的次序仅是一种可能的解决方法。中断被处理的次序很大程度上取决于系统和它所要求的行为。这必须由整个系统的设计者决定。

发送、接收和数据超载中断的进行的动作已经在前面讨论过了。

图 20、21、22 详细地给出唤醒中断、仲裁丢失中断和三个不同出错中断的流程。所有的出错中断可以执行系统的一个通用错误策略程序。这个策略完成：在开发期的系统优化和在操作期的系统自动优化和系统维护。仲裁丢失中断也可以用于系统优化和维护。见下面的章节和数据表[1]可获得关于不同的错误信号、仲裁丢失处理和相关的信息的详细资料。

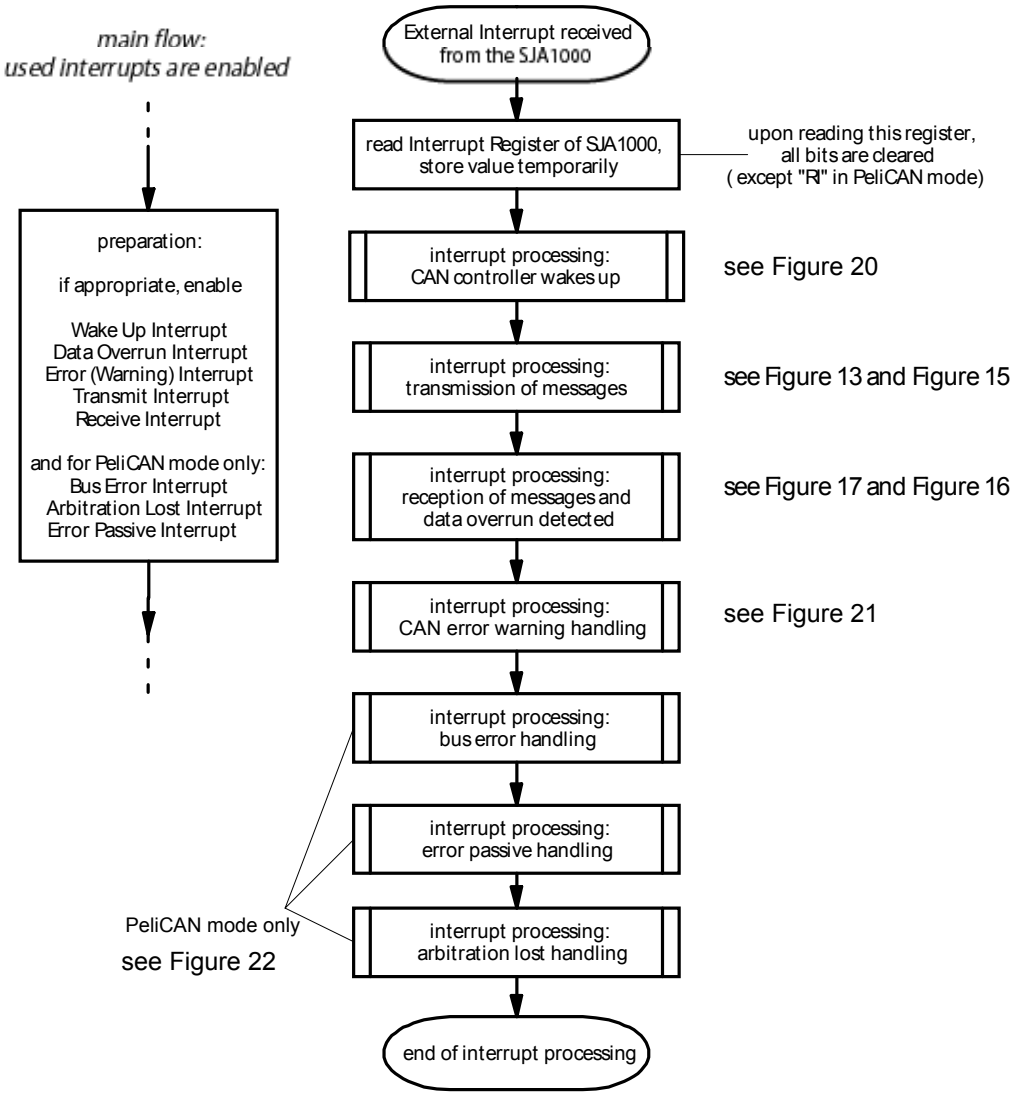


图 19　总体的中断流程

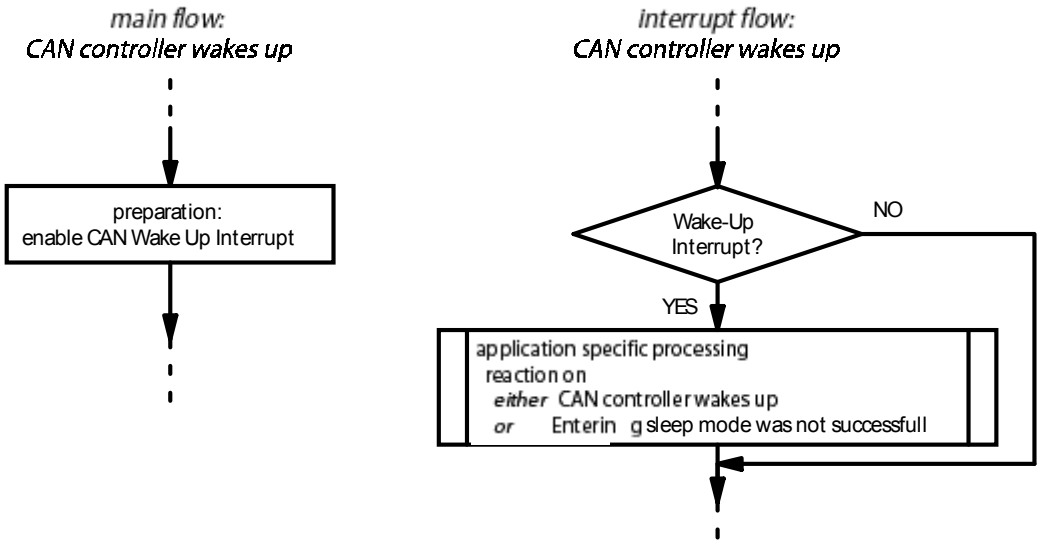


图 20 “CAN 控制器唤醒”的流程图

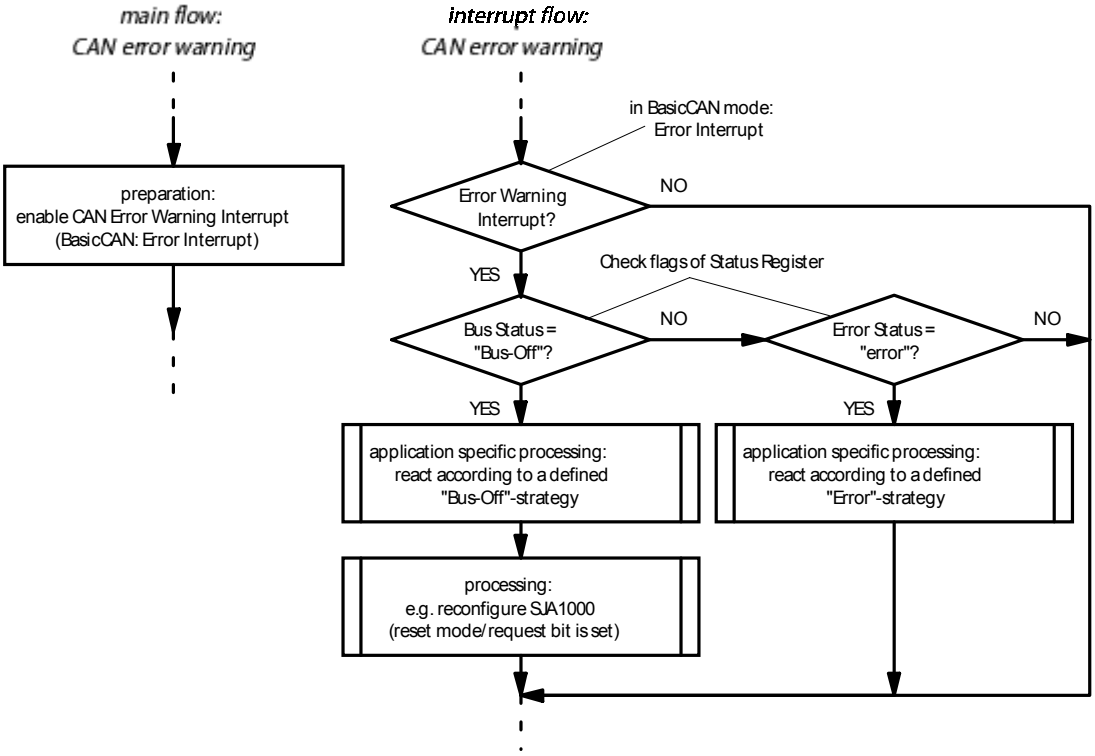


图 21 “出错中断”的流程图

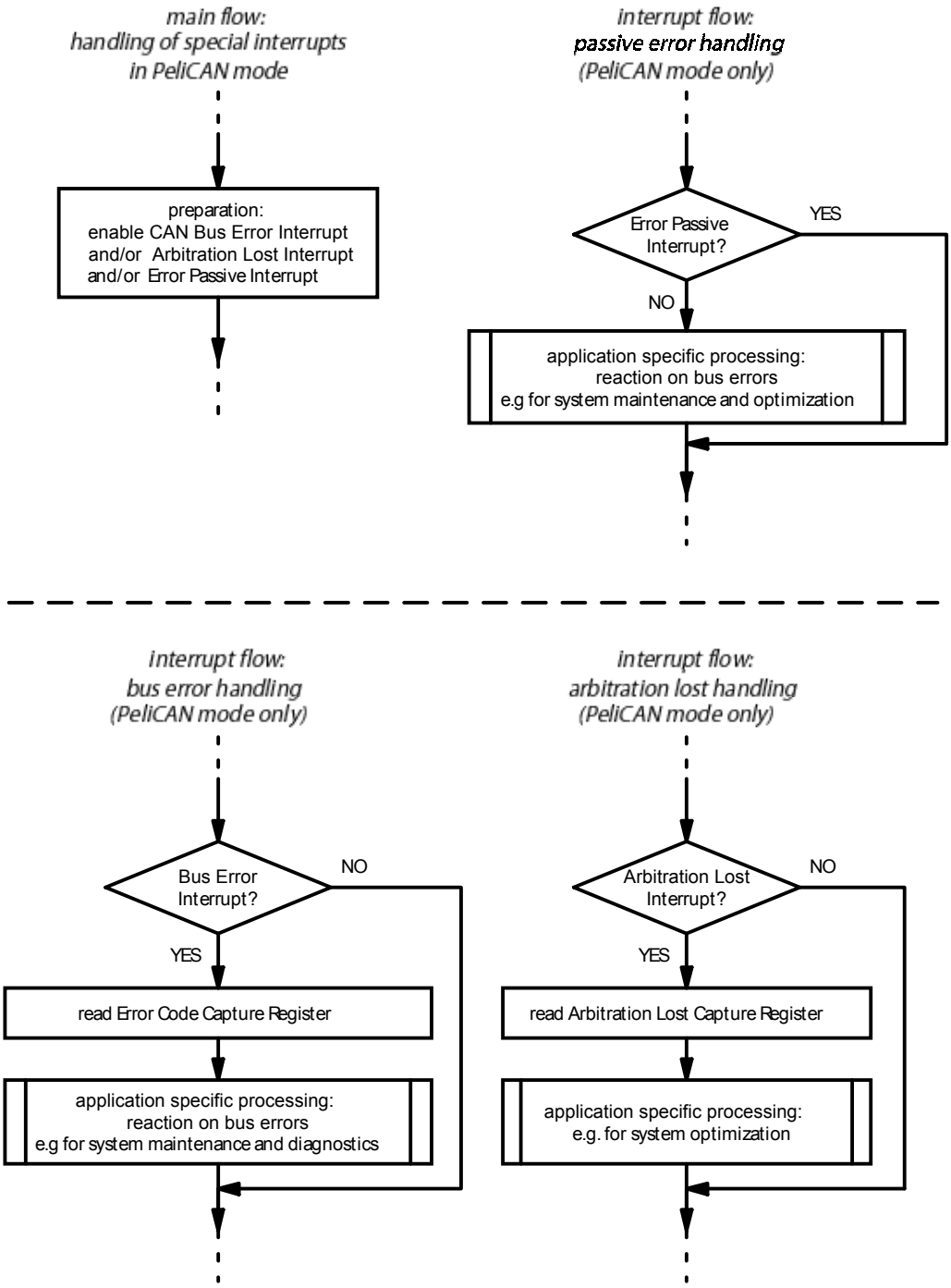


图 22 “处理特殊的 PeliCAN 中断”流程图

5．PELICAN 模式的功能

5.1 接收 FIFO/报文计数器/直接 RAM 访问

SJA1000 寄存器和报文缓冲器对于主控制器来说是外围寄存器，它们可以通过复用的地址/数据总线寻址。在不同的模式（操作或复位）可以访问不同的寄存器。正常操作的地址范围是：Address0~31。它包括用于初始化、状态和控制的寄存器。而且 CAN 报文存储器位于地址 16 和 28 之间。在主控制器写访问时，用户能够寻址 CAN 控制器的发送缓冲器，在读访问时，读出接收缓冲器的内容。

除了上面所说的范围外，整个接收 FIFO 映射在 CAN 地址 32 和 95 之间（见图 23）。而且，是内部 80 个字节 RAM 一部分的 SJA1000 发送缓冲器在 CAN 地址 96 和 108 之间。

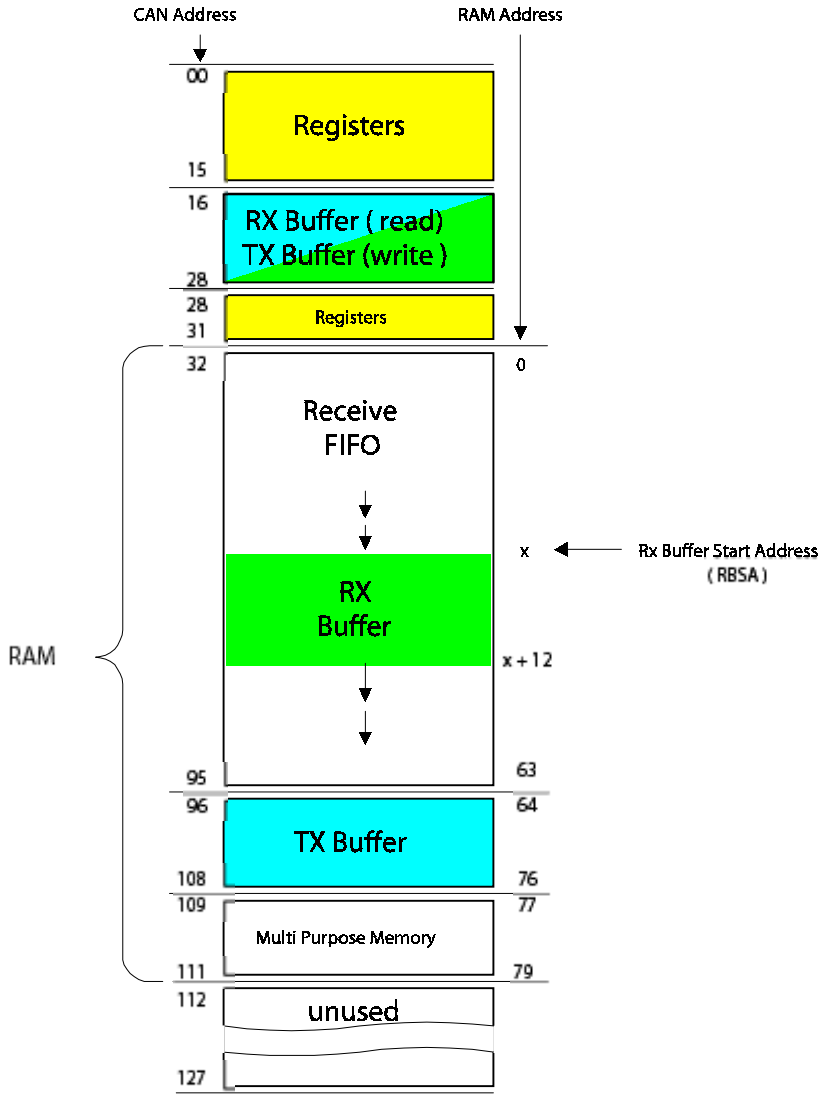


图 23　寄存器和 RAM 地址分配

直接访问 RAM 时，可以读发送缓冲器和完整的接收 FIFO。

在 PeliCAN 模式里，接收 FIFO 能够存储高达 n=21 条报文。用下面的方程，可以算出报文的最大数量：

$$n = \frac{64}{3 + data\_length\_code}$$

接收缓冲器被定义为一个 13 字节的窗口，总是包括接收 FIFO 当前的接收报文。如图 24 所示，下面的部份或全部的报文都在接收缓冲器窗口。

但是，在“释放接收缓冲器”命令之前，接收 FIFO 里的下一个收到的报文在接收缓冲器窗口（从 CAN 地址 16 开始）将完全可以看到。

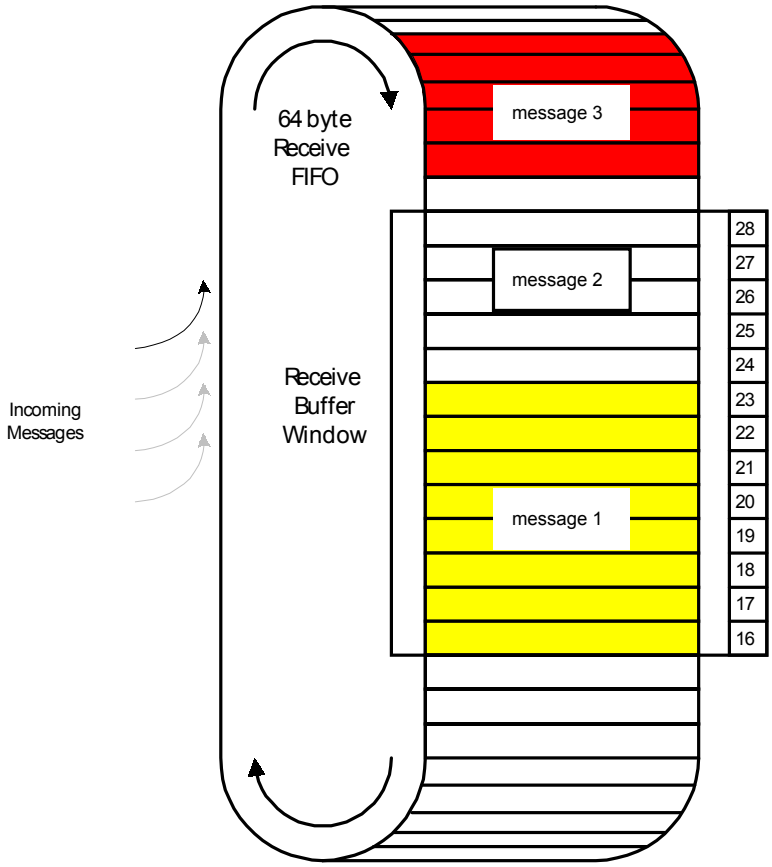


图 24　接收 FIFO

为了分析的需要，SJA1000 另外提供两个寄存器，处理接收报文：

- Rx 缓冲器起始地址寄存器（RBSA）允许接收 FIFO 范围里识别单个 CAN 报文。
- Rx 报文计数器寄存器，表示接收 FIFO 里的当前存储的报文数量。

图 23 显示了物理 RAM 地址和 CAN 地址之间的关系。

5.2　错误分析功能

基于错误计数器的值，每个 CAN 控制器能够在三种错误状态之一中工作：错误激活、错误认可或总线离线。如果错误计数器的值都在 0~127 之间，CAN 控制器是错误激活的。此时产生错误激活标志（6 个显性位）。如果一个错误计数器的值在 128－255 之间，SJA1000 是错误认可的。此时，在检测到错误前，产生认可错误标志（6 个隐性位）。如果发送错误计数器的值高于 255，则到达总线离线状态。在这种状态下，自动置位复位请求位，SJA1000 对总线没有影响。如图 25 所示，总线离线状态只能在主控制器用命令“Reset Request = 0”退出。这将启动总线离线恢复定时器，发送错误计数器计数 128 个总线释放信号。计数结束后，两个错误计数器都是 0，器件再次处于错误激活状态。

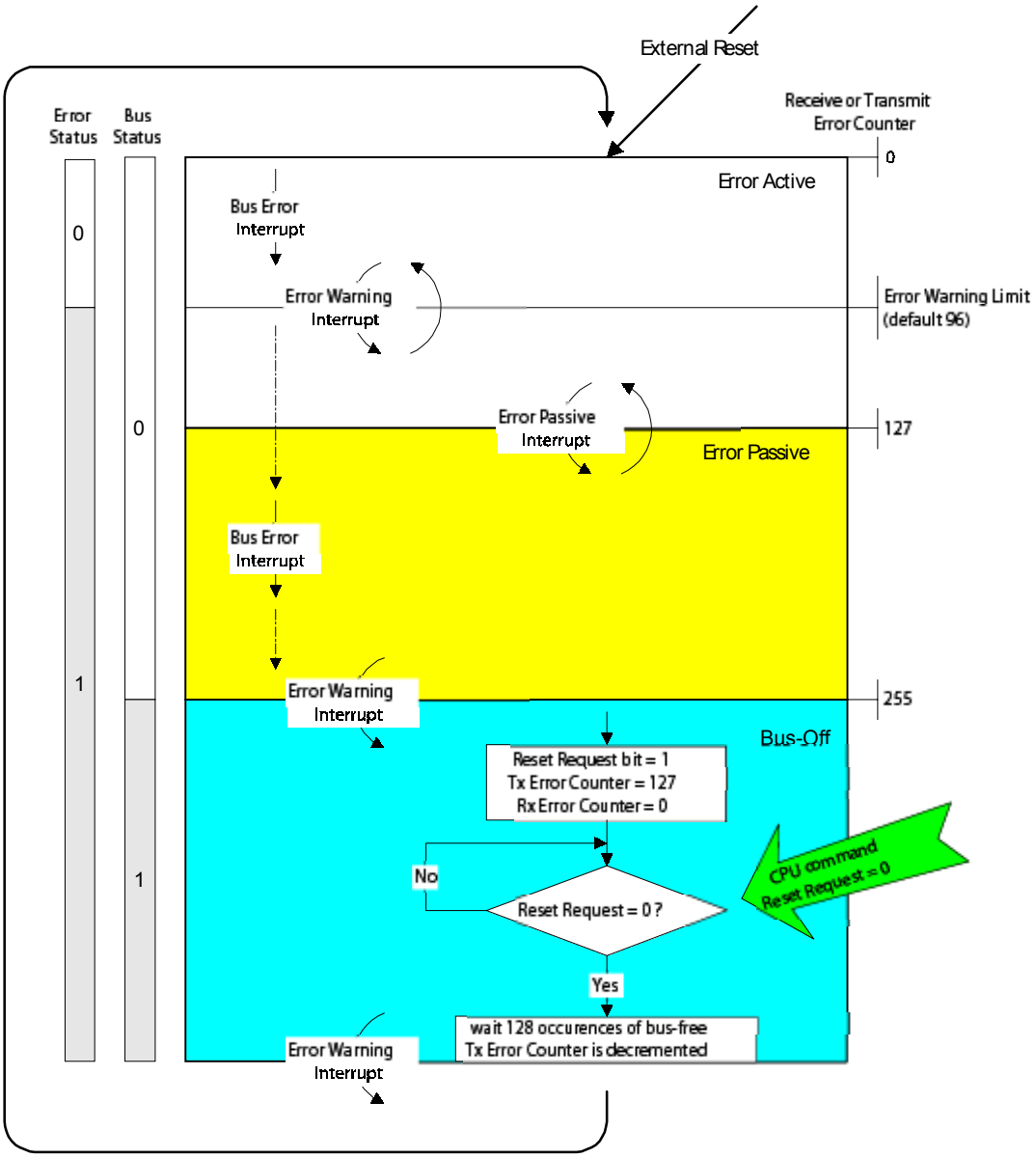


图 25 SJA1000 的出错中断

而且图上显示在不同的错误状态下错误和总线状态的值。

5.2.1 错误计数器

如上面描述，CAN 的错误状态和发送错误计数器和接收错误计数器的值直接有关。

为了仔细研究错误界定，支持 SJA1000 的增强的错误分析功能，CAN 控制器提供可读的错误计数器。另外，在复位模式，允许对于两个错误计数器进行写访问。

5.2.2 出错中断

见图 25，使用了 3 个错误中断源来向主控制器发信出错的状态。每个中断都能在中断使能寄存器里分别使能。

总线出错中断:

在 CAN 总线上检测到任何一个错误都会产生中断。

出错警告中断：

如果超过出错警告界限，产生出错警告中断。而且它在 CAN 控制器进入总线离线状态和在此之前再一次进入错误激活状态也会产生这个中断。SJA1000 的出错警告界限在复位模式中可编程。复位后的默认值是 96。

错误认可中断：

如果错误状态从错误激活变成错误认可或相反，将产生错误认可中断。

5.2.3 错误码捕捉

如前几部分描述，SJA1000 可以执行在 CAN2.0B 规范[8]定义的所有错误界定。每个 CAN 控制器处理错误的整个过程是完全自动的。但是，为了向用户提供某个错误的详细信息，SJA1000 提供了错误代码捕捉功能。无论什么时候发生 CAN 总线错误，它都会强制产生相应的总线出错中断。同时，当前位的位置被捕捉入错误代码捕捉寄存器。在主控制器将捕捉的数据读出前，它都会被保存在寄存器中。然后捕捉机制再次激活。寄存器可以内容区分四种错误类型：格式出错、填充出错、位出错和其他错误。如图 26 所示，寄存器还另外表明在错误是在报文的接收还是发送期间发生。这个寄存器中的五个位表示 CAN 帧内错误的位位置，更多信息请看下面的表和数据表。

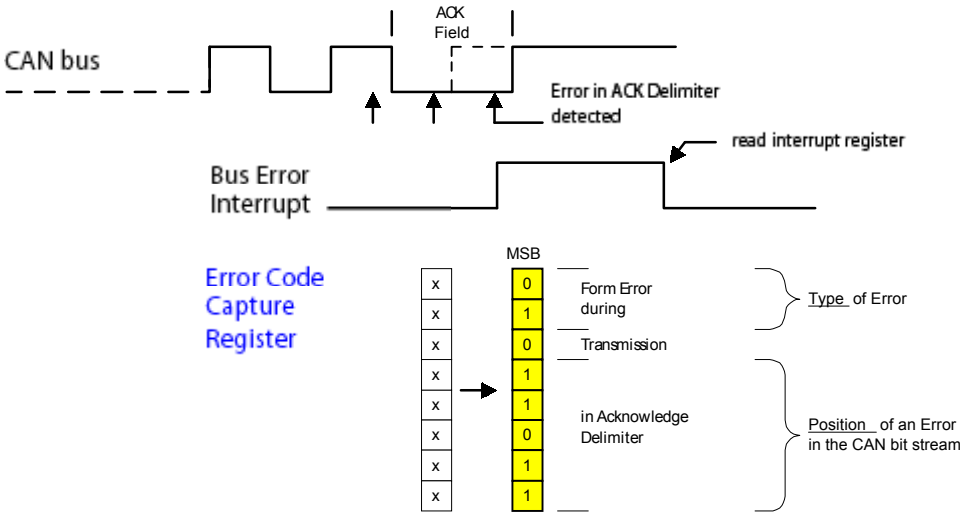


图 26 错误码捕捉功能举例

CAN 规范定义了：CAN 总线上的每个位只有特殊类型的错误。下面两张表显示了在 CAN 报文发送和接收期间可能出现的所有错误。左边的部份包括位置和错误的类型，这些由错误码捕捉寄存器捕捉。每张表的右边部分是将错误码转换成上层的错误描述，可以直接从寄存器内容知道其含意。通过使用这些表格，能得到有关错误计数器的变化和在器件发送和接收管脚的错误状态的更多信息。使用这些表时，例如在错误分析软件里，可以详细地分析每一个错误状态。关于 CAN 错误类型和位置的信息能用于错误统计和系统维护或在系统优化期间进行纠正。

表 6 接收时可能出现的错误

错误码捕捉				
CAN 位流里的错误位置	错 误类型	RX 错误计数	描述	
标识符 SRR、IDE 和 RTR 位 保留位 数据长度码 数据场 CRC 序列	填充	＋1	收到 5 个电平相同的连续的位	－－
CRC 定界符	格式 填充	＋1 ＋1	Rx = 显性 收到超过 5 个电平相同的连续的位	位必须是隐性
应答位	位	＋1	TX=显性，但 RX=隐性	不能写显性位
应答定界符[1]	格式	＋1	RX=显性，或 检测到 CRC 错误 [1]	临界的总线定时或总线长度 CRC 序列不正确
帧结束	格式 其他	＋1 ±0	RX=头六位是显性 RX=最后一位的显性	－－ 反应：发出超载标志，如果发送器重新发送，数据可能重复
间隔	其他	±0	RX=显性	反应：接收器发出超载标志
激活错误标志	位	＋8	TX=显性，但 RX=隐性	不能写显性位
容许的显性位（Tolerate Dominant Bit）	其他	＋8	RX=出错标志后的第一位是显性 RX=出错或过载标志后有超过 7 位显性位	
错误定界符	格式 其他	＋1 ±0	RX=头七位是显性位 RX=定界符的最后一位是显性位	－－ 发送超载标志
超载标志	位	＋8	TX=显性，但 RX＝隐性	不能写显性位

[1] 如果 CRC 不正确，应答定界符内的错误是“格式错误”。

表 7 发送时可能出现的错误

错误代码捕捉		TX 错误计数	描述	
CAN 位流里的错误位置	错误类型			
帧起始	位	＋8	Tx=显性，但 Rx=隐性	不能写显性位
标识符	位 填充	+8 ±0	Tx=显性，但 Rx=隐性 Tx 隐性，但 Rx=显性	不能写显性位 －－
SRR 位	位 填充	+8 ±0	Tx=显性，但 Rx=隐性 Tx=隐性，但 Rx=显性	不能写显性位 －－
IDE 和 RTR 位	位 填充	+8 +8	Tx=显性，但 Rx=隐性 Tx=隐性，但 Rx=显性	不能写显性位 －－
保留位 数据长度码 数据场 CRC 序列	位	+8	Tx=显性，但 Rx=隐性	不能写显性位
CRC 定界符	格式	+8	Rx=显性	位必须为隐性
应答隙	其他 其他	+8 ±0	Rx=隐性（错误激活） Rx=隐性（错误认可）	没有应答 没有应答，节点可能单独在总线上
应答定界符	格式	+8	Rx=显性	临界的总线定时或总线长度
帧结束	格式 其他	+8 +8	Rx=头六个位是显性位 Rx=最后一位是显性位	－－ 帧已经被一些节点接收，再次发送可能导致接收器里数据重复
间隔	其他	±0	Rx=显性	来自于“旧”CAN 控制器的超载标志
激活错误标志 过载标志	位	＋8	Tx=显性，但 Rx=隐性	不能写显性位
允许显性位（Tolerate Dominant Bit）	格式	＋8	Rx=在激活错误标志或过载标志后有超过 7 个显性位	－－
错误定界符	格式 其他	＋8 ±0	Rx=头七位是显性位 Rx=定界符的最后一位是显性位	－－ 来自于“旧”CAN 控制器的超载标志
认可错误标志	其他	＋8	Rx=显性（错误认可）	没有收到应答，节点不是单独在总线上

5.3 仲裁丢失捕捉

SJA1000 能够确定 CAN 位流丢失仲裁的确切位置。并立即产生“仲裁丢失中断”。而且，这个位的号码被捕捉到仲裁丢失捕捉寄存器。主控制器读出这个寄存器的内容后，仲裁丢失捕捉功能被再次激活。

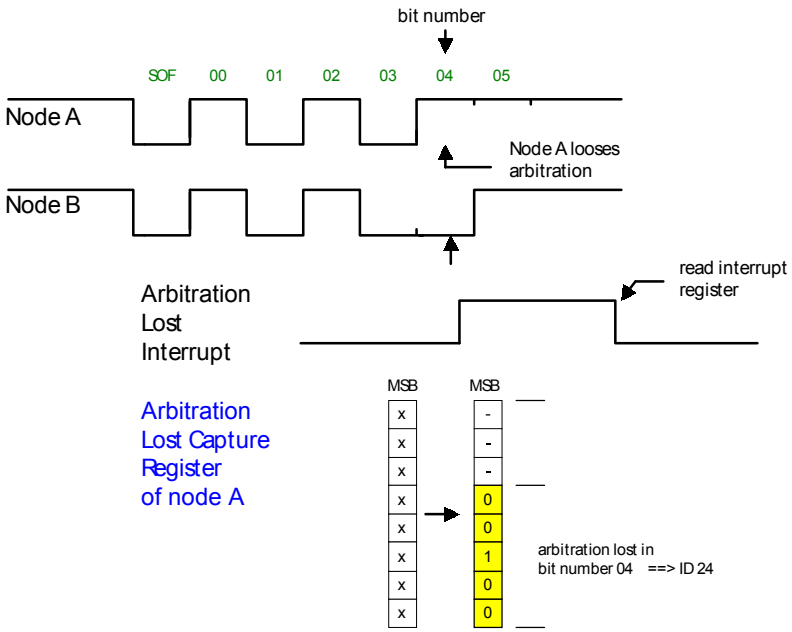


图 27 仲裁丢失捕捉功能

在这个功能的帮助下，SJA1000 能够监控每个 CAN 总线的访问。诊断或在系统配置期间，可以识别仲裁失败的所有位置。

下面这个例子显示了钟裁丢失功能怎样使用。

首先使能中断使能寄存器里的仲载丢失中断。中断后，中断寄存器的内容就被保存起来。如果仲裁丢失中断标志被置位，表示仲裁丢失捕捉寄存器的内容被分析过了。

例子：**仲载丢失**

```
…
InterruptEnReg＝ALIE_Bit;                    /*仲裁丢失中断使能                    */
…
/*-----------------------------------中断服务程序-------------------------------------------------------------- */
…
int_reg_copy＝InterruptReg;                  /*保存中断寄存器内容                  */
…
if    (int_reg_copy & ALIE_Bit)
      candat = ArbLostCapReg;                /*读仲裁丢失捕捉寄存器                */
…
…
```

5.4 单次发送

在一些应用中，自动重发 CAN 报文没有意义：它造成一个节点几次仲裁和数据变得无效。

为了请求一个“单次发送”，CAN 控制器必须完成下面的步骤：

1．发送请求

2．等待发送状态

3．中止发送

通过同时置位命令位 CMR.0 和 CMR.1，处理软件能将初始化“单次发送”选项减少到一个命令。

在这种情况下，没有必要查询状态位，主控制器能集中于其他任务上。“单次发送”功能能与 SJA1000

的仲裁丢失和错误代码捕捉功能完美结合。

如果仲裁丢失或发生错误，SJA1000 不会重新发送报文。一旦置位状态寄存器的发送状态位，内部发送请求位就自动清除。

使用两个捕捉寄存器的附加信息，一个报文是否被重发由用户控制。

如在 5.7 章里描述的，单次发送能和自我测试模式一起使用。

5.5 仅听模式

在仅听模式里，SJA1000 不能在 CAN 总线上写显性位。激活错误标志或超载标志不能都写，成功接收后的应答信息也不会给出。

错误就像在错误认可模式里处理。错误分析功能如：错误码捕捉和出错中断就像在正常操作模式一样工作。

但是，错误计数器的状态被冻结。

可以接收报文，但不可以发送。因此，这个模式可用于自动的位速率检测，见 5.6 节和其他带有监控功能的应用。

注意，在进入仅听模式之前，必须进入复位模式。

例子：**仅听模式**

```
…
ModeControlReg＝RM_RR_Bit;                /*进入复位模式                */
ClockDiv 标识符 eReg ＝CANMode_ Bit;      /*PeliCAN 模式                */
…
ModeControlReg＝LOM_Bit;                  /*进入仅听模式                */
                                          /*离开复位模式                */
…
```

5.6 自动位速率检测

自动位速率已有的测试和错误概念的主要缺点是产生的 CAN 出错帧不被接受。SJA1000 支持 PeliCAN 模式的自动位速率检测的请求。这里将简短地描述一个不影响网络运行操作的应用例子。

在仅听模式，SJA1000 不能发送报文也不能产生出错帧。这个模式里只能接收报文。软件里预定义的表格包含所有可能的位速率以及它们的位定时参数。在启动用最高位速率接收报文之前，SJA1000 使能接收中断和出错中断。如果在 CAN 总线产生了错误，软件会转向下一个较低的位速率。在成功地接收到一个报文后，SJA1000 已经检测到正确的位速率而且能转向正常工作模式。从现在起，这个节点能够象系统其他激活的 CAN 节点一样工作。

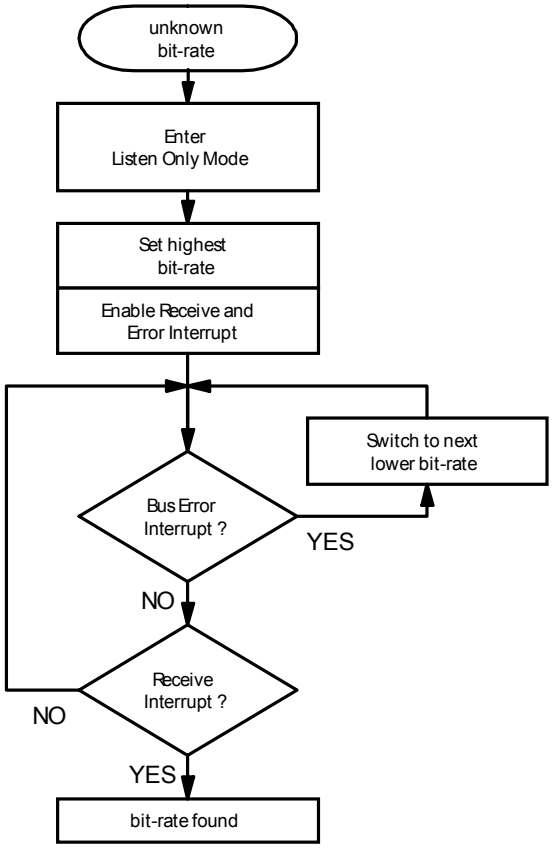


图 28 位速率检测的算法

5.7 CAN 的自测试

SJA1000 支持两种不同的自测试：

- 局部自测试
- 总体自测试

局部自测试，例如能很好地用于单个节点测试，因为它不需要来自于其他节点的应答。此时，SJA1000 必须处于“自测试模式”（模式寄存器）并发出“自接收请求”命令。

对于总体自测试，在操作模式下 SJA1000 执行同样的命令。但是，在运行系统中，需要 CAN 的应答。

注意：在这两种情况下，物理层接口包括带有终端的 CAN 总线必须有效。发送或自接收通过置位命令寄存器的相应位初始化。

SJA1000 提供三个命令位用于 CAN 发送和自接收的初始化。表 8 显示了所有可能的组合（取决于所选的工作模式）。

表 8 CAN 发送请求命令

命令	CMR=	成功操作后的中断	自我测试模式	操作模式
自接收请求	0x10	RX 和 TX	局部自测试	总体自测试
发送请求	0x01	TX	正常发送[1]	正常发送
单次发送	0x03	TX	没有重发的发送[1]	没有重发的发送
单次发送和自我接收请求	0x12	RX 和 TX	无重发的局部自测试	无重发的总体自测试

下面的例子表示了初始化局部自测试的基本编程元素，。

[1] 有或没有重发的正常发送通常在工作模式里完成。

例：局部自测试

```
…
        ModeControlReg   =RM_RR_Bit;          /*进入复位模式          */
         ClockDivideReg  =CANMode_Bit;        /*PeliCAN 模式          */
        ModeControlReg   =STM_Bit;           /*进入自我测试模式      */
                                             /*离开复位模式          */
              TxFramInfo =0x03;              /*填满发送缓冲器        */
               TxBuffer1 =0x53;              /*                      */
…
               TxBuffer5 =0xAA;              /*最后一个发送的字节    */

            CommandReg   =SRR_Bit;           /*自我接收请求          */
……….
if    (RxBuffer1 != TxBufferRd1)    comparison    =false;
if    (RxBuffer2 != TxBufferRd2)    comparison    =false;
```

5.8 接收同步脉冲的产生

在成功地接收到报文后，SJA1000 允许 TX1 管脚上产生一个脉冲（如果报文被完整地存储在接收 FIFO 里）。这个脉冲在时钟分频驱动寄存器里使能，在第 6 位“帧结束”持续期间内激活。因此，它能用于通用的事件触发任务，例如：作为一个专用的触发中断源或将在下面讨论的分布式系统内的总体时钟同步。

在分布式系统内，很难用一个不带额外同步信号线的系统时钟。所有连接到总线上的节点都有本地时钟（带有时钟相移）。假设 CAN 网络内的一个节点作为“主”时钟，网络里其余的时钟都同步到主时钟。

自身接收请求特征和每个 SJA1000 在接收到信息后的一定时间内产生一个脉冲，可以同步分布式系统里的系统时钟。

在图 29，一个系统主机发送“自接收报文”到 CAN 总线上。报文接收后，每个节点包括主机，都产生一个接收同步脉冲。这个脉冲使每个从机节点的定时器复位。同时主机节点用这个脉冲去捕捉主时钟值 t_M。

在下一步，主机将 t_M 值作为一个“参考时序报文”发送到所有从机。在每个从机的简单的加法程序和在 t_S 内重载所有定时器，同步了整个系统时钟。

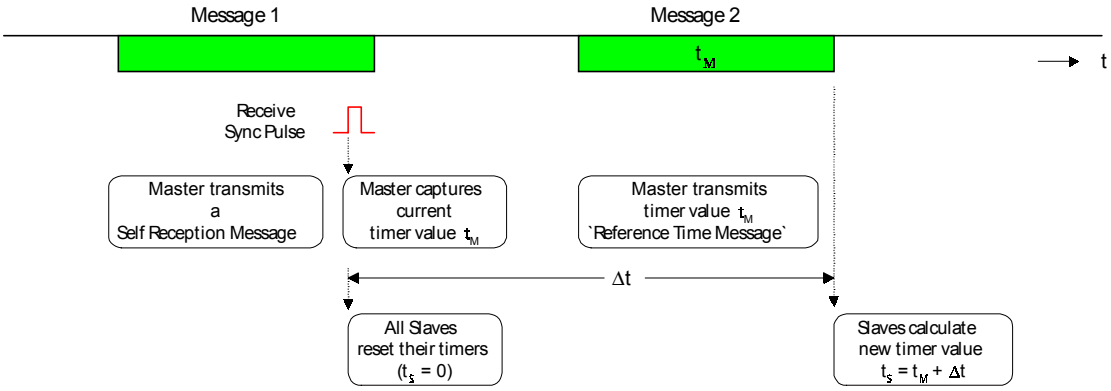


图 29 系统同步的时序图

这种方法的主要好处是简化了复杂的时间标志处理。由于关键路径是由硬件控制确定，所以没有必要使用软件循环计数。而且它独立于网络参数。中断事件可能会在整个周期内发生，但不影响同步过程。

6．参考文献

[1] Data Sheet SJA1000, Philips Semiconductors

[2] Eisele, H. and Jöhnk, E.: PCA82C250/251 CAN Transceiver, Application Note AN96116, Philips Semiconductors, 1996

[3] Data Sheet PCA82C250, Philips Semiconductors, September 1994

[4] Data Sheet PCA82C251, Philips Semiconductors, October 1996

[5] Data Sheet TJA1053, Philips Semiconductors,

[6] Jöhnk, E. and Dietmayer, K.: Determination of Bit Timing Parameters for the CAN Controller SJA1000, Application Note AN97046, Philips Semiconductors, 1997

[7] Data Sheet PCx82C200, Philips Semiconductors, November 1992

[8] CAN Specification Version 2.0, Parts A and B, Philips Semiconductors, 1992

[9] Hank, P.: PeliCAN: A New CAN Controller Supporting Diagnosis and System Optimization, 4th International CAN Conference, Berlin, Germany, October 1997

7．附录

在应用指南里的例子，我们用 C 语言（Keil 的 C 编译器）描述 SJA1000 可能的编程流程。在这些例子里，目标控制器是 Philips Semiconductors 的 S87C654，也可以使用其他 80C51 系列器件。务必确保在主程序里包括对目标器件寄存器正确的说明。

SJA1000 的寄存器和位定义

```
/*定义直接对 8051 的存储区访问                                                    */

#define   XBYTE（(unsigned char volatile xdata  *）0）

/*模式和控制寄存器的地址和位定义                                                  */

#define   ModeControlReg      XBYTE[10]

#define   RM_RR_Bit       0x01          /*复位模式（请求）位                    */
#if defined   (PeliCANMode)
#define   LOM_Bit         0x02          /*仅听模式位                            */
#define   STM_Bit         0x04          /*自我测试模式位                        */
#define   AFM_Bit         0x08          /*验收滤波器模式位                      */
#define   SM_Bit          0x10          /*进入休眠模式位                        */
#endif

/*中断使能和控制寄存器的地址和位定义                                              */
#if defined   (PeliCANMode)
#define   InterruptEnReg      XBYTE[4]      /* PeliCAN 模式                      */
#define   RIE_Bit         0x01          /*接收中断使能位                        */
#define   TIE_Bit         0x02          /*发送中断使能位                        */
#define   EIE_Bit         0x04          /*错误警告中断使能位                    */
#define   DOIE_Bit        0x08          /*数据超载中断使能位                    */
```

```
#define   WUIE_Bit          0x10          /*唤醒中断使能位                    */
#define   EPIE_Bit          0x20          /*错误隐性中断使能位                */
#define   ALIE_Bit          0x40          /*仲裁丢失中断使能位                */
#define   BEIE_Bit          0x80          /*总线错误中断使能位
#else     /*BasicCAN 模式    */
#define   InterruptEnReg    XBYTE[0]      /*  控制寄存器                      */

#define   RIE_Bit           0x02          /*接收中断使能位                    */
#define   TIE_Bit           0x04          /*发送中断使能位                    */
#define   EIE_Bit           0x08          /*错误中断使能位                    */
#define   DOIE_Bit          0x10          /*超载中断使能位                    */
#endif


/*命令寄存器的地址和位定义                                                    */

#define   CommandReg        XBYTE[1]

#define   TR_Bit            0x01          /*发送请求位                        */
#define   AT_Bit            0x02          /*中止发送位                        */
#define   RRB_Bit           0x04          /*释放接收缓冲器位                  */
#define   CDO_Bit           0x08          /*清除数据超载位                    */
#if defined   (PeliCANMode)
#define   SRR_Bit           0x10          /*自身接收请求位                    */
#else     /*BasicCAN 模式    */
#define GTS_Bit             0x10          /*进入睡眠模式位                    */
#endif

/*状态寄存器的地址和位定义
#define   StatusReg         XBYTE[2]                                        */

#define   RBS_Bit           0x01          /*接收缓冲器状态位                  */
#define   DOS_Bit           0x02          /*数据超载状态位                    */
#define   TBS_Bit           0x04          /*发送缓冲器状态位                  */
#define   TCS_Bit           0x08          /*发送完成状态位                    */
#define   RS_Bit            0x10          /*接收状态位                        */
#define   TS_Bit            0x20          /*发送状态位                        */
#define   ES_Bit            0x40          /*错误状态位                        */
#define   BS_Bit            0x80          /*总线状态位                        */

/*中断寄存器的地址和位定义                                                    */
#define   InterruptReg      XBYTE[3]

#define   RI_Bit            0x01          /*接收中断位                        */
```

```
#define   TI_Bit              0x02      /*发送中断位                          */
#define   EI_Bit              0x04      /*错误警告中断位                      */
#define   DOI_Bit             0x08      /*数据超载中断位                      */
#define   WUI_Bit             0x10      /*唤醒中断位                          */
#if defined  (PeliCANMode)
#define   EPI_Bit             0x20      /*错误被动中断位                      */
#define   ALI_Bit             0x40      /*仲裁丢失中断位                      */
#define   BEI_Bit             0x80      /*总线错误中断位                      */
#endif

/*总线定时寄存器的地址和位定义                                               */

#define   BusTiming0Reg       XBYTE[6]
#define   BusTiming1Reg       XBYTE[7]

#define   SAM_Bit             0x80      /*采样模式位
                                        1==总线被采样三次
                                        0==总线被采样一次                   */

/*输出控制寄存器的地址和位定义                                               */

#define   OutControlReg       XBYTE[8]
                                        /*OCMODE1，OCMODE0                  */
#define   BiPhaseMode         0x00      /*双相输出模式                        */
#define   NormalMode          0x02      /*正常输出模式                        */
#define   ClkOutMode          0x03      /*时钟输出模式                        */

                                        /*TX1 的输出管脚配置                  */
#define   OCPOL1_Bit          0x20      /*输出极性控制位                      */
#define   Tx1Float            0x00      /*配置为悬空                          */
#define   Tx1PullDn           0x40      /*配置为下拉                          */
#define   Tx1PullUp           0x80      /*配置为上拉                          */
#define   Tx1PshPull          0Xc0      /*配置为推挽                          */
                                        /*TX0 的输出管脚配置                  */
#define   OCPOL0_Bit          0x04      /*输出极性控制位                      */
#define   Tx0Float            0x00      /*配置为悬空                          */
#define   Tx0PullDn           0x08      /*配置为下拉                          */
#define   Tx0PullUp           0x10      /*配置为上拉                          */
#define   Tx0PshPull          0X18      /*配置为推挽                          */


/*验收代码和屏蔽寄存器的地址定义                                             */
#if defined  (PeliCANMode)
#define   AcceptCode0Reg      XBYTE[16]
```

```
#define    AcceptCode1Reg    XBYTE[17]
#define    AcceptCode2Reg    XBYTE[18]
#define    AcceptCode3Reg    XBYTE[19]
#define    AccepMask0 Reg    XBYTE[20]
#define    AccepMask1 Reg    XBYTE[21]
#define    AccepMask2 Reg    XBYTE[22]
#define    AccepMask3Reg     XBYTE[23]
#else      /*BasicCAN 模式
#define AcceptCodeReg        XBYTE[4]
#define AcceptMaskReg        XBYTE[5]
#endif

/*Rx-缓冲器的地址定义                                                    */
#if defined   (PeliCANMode)
#define    RxFramInFo        XBYTE[16]
#define    RxBuffer1         XBYTE[17]
#define    RxBuffer2         XBYTE[18]
#define    RxBuffer3         XBYTE[19]
#define    RxBuffer4         XBYTE[20]
#define    RxBuffer5         XBYTE[21]
#define    RxBuffer6         XBYTE[22]
#define    RxBuffer7         XBYTE[23]
#define    RxBuffer8         XBYTE[24]
#define    RxBuffer9         XBYTE[25]
#define    RxBuffer10        XBYTE[26]
#define    RxBuffer11        XBYTE[27]
#define    RxBuffer12        XBYTE[28]
#else      /*BasicCAN 模式
#define    RxBuffer1         XBYTE[20]
#define    RxBuffer2         XBYTE[21]
#define    RxBuffer3         XBYTE[22]
#define    RxBuffer4         XBYTE[23]
#define    RxBuffer5         XBYTE[24]
#define    RxBuffer6         XBYTE[25]
#define    RxBuffer7         XBYTE[26]
#define    RxBuffer8         XBYTE[27]
#define    RxBuffer9         XBYTE[28]
#define    RxBuffer10        XBYTE[29]
#endif

/*Tx 缓冲器的地址定义                                                    */
#if defined   (PeliCANMode)
/*仅写地址                                                              */
#define    TxFramInFo        XBYTE[16]
```

```
#define   TxBuffer1          XBYTE[17]
#define   TxBuffer2          XBYTE[18]
#define   TxBuffer3          XBYTE[19]
#define   TxBuffer4          XBYTE[20]
#define   TxBuffer5          XBYTE[21]
#define   TxBuffer6          XBYTE[22]
#define   TxBuffer7          XBYTE[23]
#define   TxBuffer8          XBYTE[24]
#define   TxBuffer9          XBYTE[25]
#define   TxBuffer10         XBYTE[26]
#define   TxBuffer11         XBYTE[27]
#define   TxBuffer12         XBYTE[28]
/*仅读地址
#define   TxFramInFoRd       XBYTE[96]
#define   TxBufferRd1        XBYTE[97]
#define   TxBufferRd2        XBYTE[98]
#define   TxBufferRd3        XBYTE[99]
#define   TxBufferRd4        XBYTE[100]
#define   TxBufferRd5        XBYTE[101]
#define   TxBufferRd6        XBYTE[102]
#define   TxBufferRd7        XBYTE[103]
#define   TxBufferRd8        XBYTE[104]
#define   TxBufferRd9        XBYTE[105]
#define   TxBufferRd10       XBYTE[106]
#define   TxBufferRd11       XBYTE[107]
#define   TxBufferRd12       XBYTE[108]
#else     /*BasicCAN 模式
#define   TxBuffer1          XBYTE[10]
#define   TxBuffer2          XBYTE[11]
#define   TxBuffer3          XBYTE[12]
#define   TxBuffer4          XBYTE[13]
#define   TxBuffer5          XBYTE[14]
#define   TxBuffer6          XBYTE[15]
#define   TxBuffer7          XBYTE[16]
#define   TxBuffer8          XBYTE[17]
#define   TxBuffer9          XBYTE[18]
#define   TxBuffer10         XBYTE[19]
#endif

/*其他寄存器的地址定义
#if defined   (PeliCANMode)
#define   ArbLostCapReg      XBYTE[11]
#define   ErrCodeCapReg      XBYTE[12]
#define   ErrWarnLimitReg    XBYTE[13]
```

```
#define   RxErrCountReg     XBYTE[14]
#define   TxErrCountReg     XBYTE[15]
#define   RxMsgCountReg     XBYTE[29]
#define   RxBufstartAdr     XBYTE[30]
#endif

/*时钟分频寄存器的地址和位定义                                              */

#define   ClockDivideReg    XBYTE[31]

#define   DivBy1            0x07        /*CLKOUT=振荡器频率                 */
#define   DivBy2            0x00        /*CLKOUT=1/2 振荡器频率             */

#define   CLKOff_Bit        0x08        /*时钟关闭位，时钟输出管脚控制位    */
#define   RXINTEN_Bit       0x20        /*用于接收中断的管脚 TX1            */
#define   CBP_Bit           0x40        /*CAN 比较器旁路控制位              */
#define   CANMode_Bit       0x80        /*CAN 模式控制位                    */
```

S87C654 的寄存器和位定义

```
/*端口 2 寄存器“P2”                                                         */
sfr   P2        =0xA0;

Sbit            =0xA7;                  /*端口 2 的 MSB，用于 SJA1000 的片选  */
P2_7
 •
/*端口 3   P3 的复用功能                                                    */
sfr   P3        =0xB0;
 •
Sbit  int0      =0xB2;
 •
/*定时控制寄存器“TCON” */
sfr             =0x88;
TCON

 •
Sbit  IE0       =0x89;                  /*外部中断 0 边缘标志               */
Sbit  IT0       =0x88;                  /*中断 0 类型控制位（边缘或低电平触发）*/
 •

/*中断使能寄存器“IE”                                                        */
sfr    IE       =0xA8;

Sbit  EA        =0xAF;                  /*所有中断使能/禁能标志             */
 •
```

```
Sbit   EX0    =0xA8;                          /*外部中断 0 的使能或禁能位                */
 •

/*中断优先级寄存器“IP” */
sfr      IP      ＝0xB8;
 •
sbit   PX0     ＝0xB8;                         /*外部中断 0 优先级控制                    */
 •
```

例子里的变量和常量的定义

```
/*-硬件/软件连接的定义-------------------------------------------------------------------------------------------- */
/*控制器：S87C654；CAN 控制器：SJA1000（见 11 页上的图 3）                                 */
#define    CS                  P2_7        /*SJA1000 的片选                            */
#define    SJAIntInp           Int0        /*SJA1000 的外部中断 0                      */
#define    SJAIntEn            EX0         /*外部中断 0 使能标志                       */

/*－使用的常量定义------------------------------------------------------------------------------------------------ */

#define    YES                 1
#define    NO                  0

#define    ENABLE              1
#define    DISABLE             0
#define    ENABLE_N            0
#define    DISABLE_N           1

#define    INTLEVELACT         0
#define    INTEDGEACT          1

#define    PRIORITY_LOW        0
#define    PRIORITY_HIGH       1

/*寄存器内容默认（复位）值，清除寄存器                                                     */
#define    ClrByte             0x00

/*常量：清除中断使能寄存器                                                                 */
#if defined   (PeliCANMode)
#define    ClrIntEnSJA         ClrByte
#else
#define    ClrIntEnSJA         ClrByte｜RM_RR_Bit      /*保留复位请求                   */
#endif

/*验收代码和屏蔽寄存器的定义                                                               */
#define    DontCare            0xFF
```

```
/*不同例子的总线定时值的定义----------------------------------------------------------------------------*/

/*例子 AN97076 里总线定时值
   一位率                    ：25kbit/s
   一振荡器频率              ：24MHz，1，0%
   一最大允许传播延迟        ：1630ns
   一最小要求传播延迟        ：120ns
#define    PrescExample     0x02          /*波特率预分频    ：3          */
#define    SJWExample       0xc0          /*SJW             ：4          */
#define    TSEG1Example     0x0A          /*TSEG1           ：11         */
#define    TSEG2Example     0x30          /*TSEG2           ：4          */

/*总线定时值：
   一位率                    ：1MBit/s
   一振荡器频率              ：24MHz，1，0%
   一最大允许传播延迟        ：747ns
   一最小要求传播延迟        ：45ns
#define    Prec_MB_24       0x00          /*波特率预分频器  ：1          */
#define    SJW_MB_24        0x00          /*SJW             ：1          */
#define    TSEG1_MB_24      0x08          /*TSEG1           ：9          */
#define    TSEG2_MB_24      0x10          /*TSEG2           ：2          */

/*总线定时值：
   一位率                    ：100kBit/s
   一振荡器频率              ：24MHz，1，0%
   一最大允许传播延迟        ：4250ns
   一最小要求传播延迟        ：100ns
#define    Prec_kB_24       0x07          /*波特率预分频器  ：8          */
#define    SJW_kB_24        0xc0          /*SJW             ：4          */
#define    TSEG1_kB_24      0x09          /*TSEG1           ：10         */
#define    TSEG2_kB_24      0x30          /*TSEG2           ：4          */

/*总线定时值：
   一位率                    ：1MBit/s
   一振荡器频率              ：16MHz，1，0%
   一最大允许传播延迟        ：623ns
   一最小要求传播延迟        ：23ns
#define    Prec_MB_16       0x00          /*波特率预分频器  ：1          */
#define    SJW_MB_16        0x00          /*SJW             ：1          */
#define    TSEG1_MB_16      0x08          /*TSEG1           ：5          */
#define    TSEG2_MB_16      0x10          /*TSEG2           ：2          */

/*总线定时值：
```

```
    －位率                          ：100kBit/s
    －振荡器频率                    ：16MHz，1，0%
    －最大允许传播延迟              ：4450ns
    －最小要求传播延迟              ：500ns
#define     Prec_kB_16      0x04        /*波特率预分频器    ：5     */
#define     SJW_kB_16       0xC0        /*SJW               ：4     */
#define     TSEG1_kB_16     0x0A        /*TSEG1             ：11    */
#define     TSEG2_kB_16     0x30        /*TSEG2             ：4     */

/*总线定时值的结束----------------------------------------------------------------------------------------------------*/

/*使用过的变量的定义--------------------------------------------------------------------------------------------------*/
/*中断寄存器内容的中间存储内容----------------------------------------------------------------------------------------*/
BYTE bdata CANInterrupt;                        /*位可寻址的字节                */
sbit    RI_BitVar       = CANInterrupt  ^   0;
sbit    TI_ BitVar      = CANInterrupt  ^   1;
sbit    EI_ BitVar      = CANInterrupt  ^   2;
sbit    DOI_ BitVar     = CANInterrupt  ^   3;
sbit    WUI_ BitVar     = CANInterrupt  ^   4;
sbit    EPI_ BitVar     = CANInterrupt  ^   5;
sbit    ALI_ BitVar     = CANInterrupt  ^   6;
sbit    BEI_ BitVar     = CANInterrupt  ^   7;
```