OBTCAN-IP 核

用户手册

（版本：V2.1）

珠海欧比特控制工程股份有限公司

地址:广东省珠海市唐家东岸白沙路 1 号欧比特科技园　邮编:519080

电话:0756-3391979　传真:0756-3391980　网址:www.myorbita.net

修改记录

版本变更	日期	修改内容	备注
V2.1	2015-02-22	对文件格式、版式等进行修订后，重新发布。	

目 录

表目录

图目录

1 简介

OBT-CAN 实现了 CAN 2.0B 协议，支持 BasicCAN 和 PeliCAN 模式，这两种模式可以通过时钟分频寄存器选择。在 BasicCAN 和 PeliCAN 两种模式下寄存器的映射有所不同。

OBT-CAN 控制器共有 32 个寄存器，地址分别为 0-31。在 CPU 看来，CAN 控制器相当于存储器地址映射的 I/O 设备，CPU 对 CAN 控制器的所有操作都是通过访问寄存器实现的。

OBT-CAN 控制器主要特征

- PCA82C200 模式（即默认的 BasicCAN 模式）
- 扩展的接收缓冲器（64 字节先进先出 FIFO）
- 和 CAN2.0B 协议兼容
- 同时支持 11 位和 29 位标识符
- 位速率可达 1Mbits/s
- PeliCAN 模式扩展功能
  - 可读/写访问的错误计数器
  - 可编程的错误报警限制
  - 最近一次错误代码寄存器
  - 对每一个 CAN 总线错误的中断
  - 具体控制位控制的仲裁丢失中断
  - 单次发送（无重发）
  - 只听模式（无确认、无活动的出错标志）
  - 支持热插拔（软件位速率检测）
  - 验收过滤器扩展（4 字节代码，4 字节屏蔽）
  - 自身信息接收（自接收请求）

2 结构框图

CAN 控制器主要由寄存器、位定时逻辑和位流处理器 3 个模块组成，如图 1-1 所示。

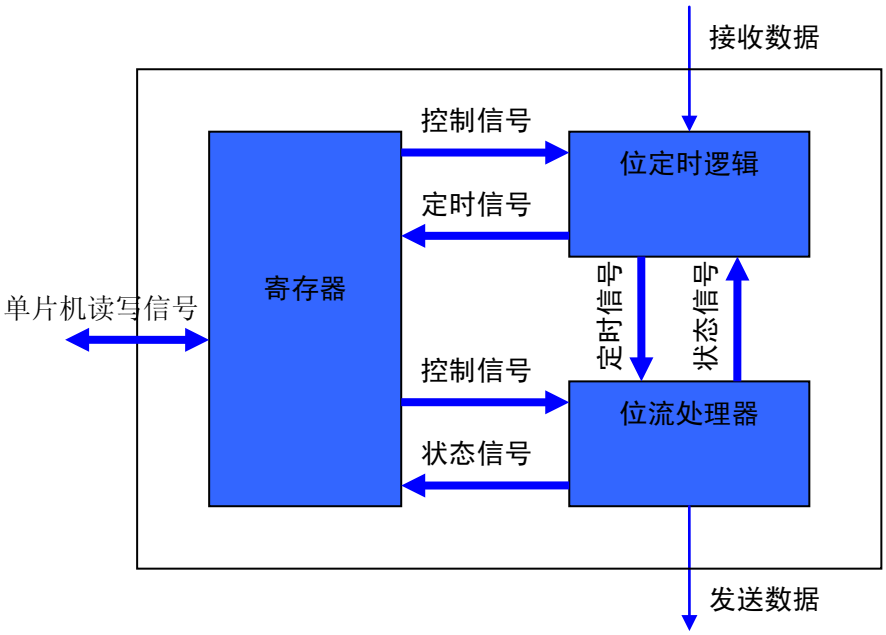


图 2-1 CAN 控制器结构框图

CAN 寄存器是与 CPU 连接的模块，CPU 对 CAN 控制器的所有操作都是通过寄存器进行的。

位定时逻辑实现的主要功能是检测 CAN 总线输入信号和来自位流处理器的 CAN 总线输出信号，输出采样点、采样位、发送点和同步信号等位定时信息。

位流处理器实现的主要功能是输入来自寄存器的寄存器信息和来自位定时逻辑的采样点、采样位、发送点和同步信号等位定时信息，对采样位的位流进行处理，并输出状态信息和 CAN 总线输出信号。

表 2-1 OBTCAN-IP 核外部接口信号说明

序号	信号名称	信号方向	默认状态	信号描述
1	rst_n	I	—	外部复位信号高电平有效
2	ale	I	—	ALE 输入信号
3	rd_n	I	—	微控制器的/RD 信号

序号	信号名称	信号方向	默认状态	信号描述
				(Intel 模式)或 E 使能信号(Motorola 模式)
4	wr_n	I	-	微控制器的/wr 信号(Intel 模式)或 RD/(/WR)信号(Motorola 模式)
5	port_io	I/O	—	多路地址/数据总线
6	cs_n	I	—	片选信号，低电平有效
7	clk_i	I	—	时钟信号输入
8	rx_i	I	—	接收数据信号端
9	tx_o	0	—	发送数据信号端
10	Mode	I	—	模式选择信号
11	irq_n	0	-	中断请求信号
12	clkout_o	0	—	CAN 分频后时钟信号输出

3 BasicCAN模式寄存器

3.1 BasicCAN模式寄存器映射

表 3-1 BasicCAN 地址分配

地址	工作模式		复位模式	
	读	写	读	写
0	控制寄存器	控制寄存器	控制寄存器	控制寄存器
1	(0xFF)	命令寄存器	(0xFF)	命令寄存器
2	状态寄存器	—	状态寄存器	—
3	中断寄存器	—	中断寄存器	—
4	(0xFF)	—	验收代码寄存器	验收代码寄存器
5	(0xFF)	—	验收屏蔽寄存器	验收屏蔽寄存器
6	(0xFF)	—	总线定时 0 寄存器	总线定时 0 寄存器
7	(0xFF)	—	总线定时 1 寄存器	总线定时 1 寄存器
8	(0x00)	—	(0x00)	—
9	(0x00)	—	(0x00)	—
10	发送识别码(10-3)	发送识别码(10-3)	(0xFF)	—
11	发送识别码(2-0)、RTR、DLC	发送识别码(2-0)、RTR、DLC	(0xFF)	—

地址	工作模式		复位模式	
	读	写	读	写
12	发送数据字节 1	发送数据字节 1	（0xFF）	—
13	发送数据字节 2	发送数据字节 2	（0xFF）	—
14	发送数据字节 3	发送数据字节 3	（0xFF）	—
15	发送数据字节 4	发送数据字节 4	（0xFF）	—
16	发送数据字节 5	发送数据字节 5	（0xFF）	—
17	发送数据字节 6	发送数据字节 6	（0xFF）	—
18	发送数据字节 7	发送数据字节 7	（0xFF）	—
19	发送数据字节 8	发送数据字节 8	（0xFF）	—
20	接收识别码（10-3）	—	接收识别码（10-3）	—
21	接收识别码（2-0）、RTR、DLC	—	接收识别码（2-0）、RTR、DLC	—
22	接收数据字节 1	—	接收数据字节 1	—
23	接收数据字节 2	—	接收数据字节 2	—
24	接收数据字节 3	—	接收数据字节 3	—
25	接收数据字节 4	—	接收数据字节 4	—
26	接收数据字节 5	—	接收数据字节 5	—
27	接收数据字节 6	—	接收数据字节 6	—
28	接收数据字节 7	—	接收数据字节 7	—
29	接收数据字节 8	—	接收数据字节 8	—
30	（0x00）	—	（0x00）	—
31	时间分频寄存器	时间分频寄存器	时间分频寄存器	时间分频寄存器

3.2 控制寄存器

控制寄存器包含中断使能位和复位请求位。

表 3-2 控制寄存器（CR）

符号.位	名称	功能	硬件复位	软件复位
CR.7	—	保留（总是 0）。	0	0
CR.6	—	保留（总是 0）。	0	0
CR.5	—	保留（总是 1）。	1	1
CR.4	溢出中断使能	1：使能；0：禁能。	x	x
CR.3	错误中断使能	1：使能；0：禁能。	x	x
CR.2	发送中断使能	1：使能；0：禁能。	x	x
CR.1	接收中断使能	1：使能；0：禁能。	x	x
CR.0	复位请求	1：停止当前传输并进入复位模式； 0：返回工作模式。	1	1

- x：复位不影响该寄存器或位。

3.3 命令寄存器

往寄存器的相应位写 1 将引起被支持的动作。

表 3-3 命令寄存器（CMR）

符号.位	名称	功能	硬件复位	软件复位
CMR.7	—	保留（总是 1）。	注 1	注 1
CMR.6	—	保留（总是 1）。		
CMR.5	—	保留（总是 1）。		
CMR.4	—	保留（总是 1）。		
CMR.3	清除数据溢出	清除数据溢出状态位。		
CMR.2	释放接收缓冲器	释放当前接收缓冲器以便于新的接收。		
CMR.1	停止发送	停止尚未开始的发送。		
CMR.0	发送请求	开始发送缓冲器中报文的发送。		

- 注 1：读命令寄存器的结果总为“1111 1111”。

3.4 状态寄存器

状态寄存器反映模块的当前状态并且是只读的。

表 3-4 状态寄存器（SR）

符号.位	名称	功能	硬件复位	软件复位
SR.7	总线状态	模块总线关闭，且此时无总线活动时，为 1。	0	x
SR.6	错误状态	至少有一个错误计数器达到或超过 CPU 报警限制。	0	x
SR.5	发送状态	正在发送报文时，为 1。	0	0
SR.4	接收状态	正在接收报文时，为 1。	0	0
SR.3	发送完毕	最后一个报文发送成功时，为 1。	1	x
SR.2	发送缓冲器状态	为 1 时，CPU 可以向发送缓冲器中写入数据。	1	1
SR.1	数据溢出状态	FIFO 中无空间导致报文丢失时，为 1。	0	0
SR.0	接收缓冲器状态	接收 FIFO 中有可用报文时，为 1	0	0

- x：复位不影响该寄存器或位。

3.5 中断寄存器

中断寄存器通知 CPU 是什么引起了中断。只有在控制寄存器里相应的中断允许位置 1 时中断位才置 1。

表 3-5 中断寄存器（IR）

符号.位	名称	功能	硬件复位	软件复位
IR.7	—	保留（总是 1）。	1	1
IR.6	—	保留（总是 1）。	1	1
IR.5	—	保留（总是 1）。	1	1
IR.4	—	保留（总是 0）。	0	0
IR.3	数据溢出中断	若 SR.1 由 0 变为 1，置位。	0	0
IR.2	错误中断	若错误状态或总线状态发生变化，置位。	0	x
IR.1	发送中断	若发送缓冲器被释放，置位。	0	0
IR.0	接收中断	FIFO 不空时，置位。	0	0

- x: 复位不影响该寄存器或位。

3.6 发送缓冲寄存器

发送缓冲存储来自 CPU 的将要通过本模块发送的数据。在 BasicCAN 模式下只有标准帧格式报文可以被发送和接收，扩展帧格式报文将被忽略。

表 3-6 发送缓冲器

地址	名称	位							
		7	6	5	4	3	2	1	0
10	识别码 1	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3
11	识别码 1	ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0
12	发送数据 1	发送字节 1							
13	发送数据 2	发送字节 2							
14	发送数据 3	发送字节 3							
15	发送数据 4	发送字节 4							
16	发送数据 5	发送字节 5							
17	发送数据 6	发送字节 6							
18	发送数据 7	发送字节 7							
19	发送数据 8	发送字节 8							

3.7 接收缓冲寄存器

位于地址 20 至 29 的接收缓冲是 64 字节接收 FIFO 的可见部分。它的结构与发送缓冲器相同。

3.8 接收过滤寄存器

应用接收过滤代码和接收过滤屏蔽寄存器，报文可以根据它们的标识符（ID）被过滤。

11 位的标识符的高 8 位与接收过滤代码寄存器中相应的接收过滤屏蔽寄存器中设为 0 的位比较，如果匹配则储存进 FIFO。

该寄存器不受硬件复位和软件复位的影响。

4 PeliCAN模式寄存器

4.1 PeliCAN模式寄存器映射

表 4-1 PeliCAN 地址分配

地址	工作模式				复位模式	
	读		写		读	写
0	模式寄存器		—		模式寄存器	模式寄存器
1	（0x00）		命令寄存器		（0x00）	命令寄存器
2	状态寄存器		—		状态寄存器	—
3	中断寄存器		—		中断寄存器	—
4	中断使能寄存器		中断使能寄存器		中断使能寄存器	中断使能寄存器
5	（0x00）		—		（0x00）	—
6	总线定时 0 寄存器		—		总线定时 0 寄存器	总线定时 0 寄存器
7	总线定时 1 寄存器		—		总线定时 1 寄存器	总线定时 1 寄存器
8	（0x00）		—		（0x00）	—
9	（0x00）		—		（0x00）	—
10	（0x00）		—		（0x00）	—
11	仲裁丢失捕捉寄存器		—		仲裁丢失捕捉寄存器	—
12	错误代码捕捉寄存器		—		错误代码捕捉寄存器	—
13	错误报警限制寄存器		—		错误报警限制寄存器	错误报警限制寄存器
14	接收错误计数器		—		接收错误计数器	接收错误计数器
15	发送错误计数器		—		发送错误计数器	发送错误计数器
16	接收 SFF 帧信息	接收 EFF 帧信息	发送 SFF 帧信息	发送 EFF 帧信息	验收代码 0 寄存器	验收代码 0 寄存器
17	接收识别码 1	接收识别码 1	发送识别码 1	发送识别码 1	验收代码 1 寄存器	验收代码 1 寄存器
18	接收识别码 2	接收识别码 2	发送识别码 2	发送识别码 2	验收代码 2 寄存器	验收代码 2 寄存器
19	接收数据 1	接收识别码 3	发送数据 1	发送识别码 3	验收代码 3 寄存器	验收代码 3 寄存器
20	接收数据 2	接收识别码 4	发送数据 2	发送识别码 4	验收屏蔽 0 寄存器	验收屏蔽 0 寄存器
21	接收数据 3	接收数据 1	发送数据 3	发送数据 1	验收屏蔽 1 寄存器	验收屏蔽 1 寄存器

地址	工作模式				复位模式	
	读		写		读	写
22	接收数据 4	接收数据 2	发送数据 4	发送数据 2	验收屏蔽 2 寄存器	验收屏蔽 2 寄存器
23	接收数据 5	接收数据 3	发送数据 5	发送数据 3	验收屏蔽 3 寄存器	验收屏蔽 3 寄存器
24	接收数据 6	接收数据 4	发送数据 6	发送数据 4	保留（0x00）	—
25	接收数据 7	接收数据 5	发送数据 7	发送数据 5	保留（0x00）	—
26	接收数据 8	接收数据 6	发送数据 8	发送数据 6	保留（0x00）	—
27	FIFO	接收数据 7	—	发送数据 7	保留（0x00）	—
28	FIFO	接收数据 8	—	发送数据 8	保留（0x00）	—
29	接收报文计数器		—		接收报文计数器	—
30	（0x00）		—		（0x00）	—
31	时间分频寄存器		时间分频寄存器		时间分频寄存器	时间分频寄存器

4.2 模式寄存器

表 4-2 模式寄存器（MOD）

符号.位	名称	功能	硬件复位	软件复位
MOD.7	—	保留（总是 0）。	0	0
MOD.6	—	保留（总是 0）。	0	0
MOD.5	—	保留（总是 0）。	0	0
MOD.4	—	保留（总是 0）。	0	0
MOD.3	验收滤波模式	1：单滤波模式；0：双滤波模式。	0	x
MOD.2	自检测模式	1：控制器进入自检测模式。	0	x
MOD.1	仅听模式	1：控制器进入仅听模式。	0	x
MOD.0	复位模式	1：停止当前传输并进入复位模式。 0：返回工作模式。	1	1

- x：复位不影响该寄存器或位。

4.3 命令寄存器

往寄存器的相应位写 1 将引起被支持的动作。

表 4-3 命令寄存器（CMR）

符号.位	名称	功能	硬件复位	软件复位
CMR.7	—	保留（总是 0）。	0	0
CMR.6	—	保留（总是 0）。	0	0
CMR.5	—	保留（总是 0）。	0	0
CMR.4	自接收请求	1：发送并同时接收一个报文。	0	0
CMR.3	清除数据溢出	1：清除数据溢出状态位。	0	0
CMR.2	释放接收缓冲器	1: 释放当前接收缓冲器以便于新的接收。	0	0
CMR.1	停止发送	1：停止尚未开始的发送。	0	0
CMR.0	发送请求	1：开始发送缓冲器中报文的发送。	0	0

- x：复位不影响该寄存器或位。

4.4 状态寄存器

状态寄存器反映模块的当前状态并且是只读的。

表 4-4 状态寄存器

符号.位	名称	功能	硬件复位	软件复位
SR.7	总线状态	1：模块总线关闭，且此时无总线活动。	0	x
SR.6	错误状态	1：至少有一个错误计数器达到或超过报警限制。	0	x
SR.5	发送状态	1：正在发送报文。	1	x
SR.4	接收状态	1：正在接收报文。	1	x
SR.3	发送完毕	1：最后一个报文发送成功。	1	x
SR.2	发送缓冲器状态	1：CPU 可以向发送缓冲器中写入数据。	1	1
SR.1	数据溢出状态	1：FIFO 中无空间导致报文丢失。	0	0
SR.0	接收缓冲器状态	1：接收 FIFO 中有可用报文。	0	0

- x：复位不影响该寄存器或位。
- SR.4 与 SR.5 在总线关闭后会产生 11 个连续的高电平。

4.5 中断寄存器

中断寄存器通知 CPU 是什么引起了中断。只有在中断允许寄存器里相应的中断允许位置 1 时中断位才置 1。

表 4-5 中断寄存器（IR）

符号.位	名称	功能	硬件复位	软件复位
IR.7	总线错误中断	若检测到总线上有错误，置位。	0	0
IR.6	仲裁丢失中断	若模块已经丢失仲裁，置位。	0	0
IR.5	错误被动中断	若模块处于错误主动与错误被动之间。	0	0

符号.位	名称	功能	硬件复位	软件复位
IR.4	—	保留（总是 0）。	0	0
IR.3	数据溢出中断	若 SR.1 由 0 变为 1，置位。	0	0
IR.2	错误中断	若错误状态或总线状态发生变化，置位。	0	x
IR.1	发送中断	若发送缓冲器被释放，置位。	0	0
IR.0	接收中断	FIFO 不空时，置位。	0	0

- x：复位不影响该寄存器或位。

4.6 中断允许寄存器

在中断允许寄存器里可以允许/禁止独立的中断源。如果被允许，则中断寄存器里的相应位可以被置 1，同时将产生一个中断。

表 4-6 中断允许寄存器（IER）

符号.位	名称	功能	硬件复位	软件复位
IER.7	总线错误中断使能	1：使能；0：禁能。	x	x
IER.6	仲裁丢失中断使能	1：使能；0：禁能。	x	x
IER.5	错误被动中断使能	1：使能；0：禁能。	x	x
IER.4	—	保留。	x	x
IER.3	数据溢出中断使能	1：使能；0：禁能。	x	x
IER.2	错误中断使能	1：使能；0：禁能。	x	x
IER.1	发送中断使能	1：使能；0：禁能。	x	x
IER.0	接收中断使能	1：使能；0：禁能。	x	x

- x：复位不影响该寄存器或位。

4.7 仲裁丢失捕捉寄存器

表 4-7 仲裁丢失捕捉寄存器（ALC）

符号.位	名称	功能	硬件复位	软件复位
ALC.7:5	—	保留（总是 0）。	0	x
ALC.4:0	位编号	仲裁时丢失的位编号。		

- x：复位不影响该寄存器或位。

4.8 错误代码捕捉寄存器

表 4-8 仲裁丢失捕捉寄存器（ECC）

符号.位	名称	功能	硬件复位	软件复位
ECC.7:6	错误代码	错误代码编号。	0	x

符号.位	名称	功能	硬件复位	软件复位
ECC.5	方向	1：接收；0：发送。	0	x
ECC.4:0	段	帧中出错的部分。	0	x

- x：复位不影响该寄存器或位。

表 4-9 错误代码说明（ECC.7:6）

ECC.7:6	说明
0	位错误
1	格式错误
2	填充错误
3	其它

表 4-10 错误代码说明（ECC.4:0）

ECC.4:0	说明
0x03	帧起始
0x02	ID.28 – ID.21
0x06	ID.20 – ID.18
0x04	SRTR 位
0x05	IDE 位
0x07	ID.17 – ID.13
0x0F	ID.12 – ID.5
0x0E	ID.4 – ID.0
0x0C	RTR 位
0x0D	保留位 1
0x09	保留位 0
0x0B	数据长度代码
0x0A	数据段
0x08	CRC 序列
0x18	CRC 界定符
0x19	应答通道
0x1B	应答界定符
0x1A	帧结束
0x12	间断
0x11	主动错误标记
0x16	被动错误标记
0x13	支配位误差
0x17	错误界定符
0x1C	过载标记

4.9 错误报警限制寄存器

该寄存器允许设置 CPU 错误警告的限制。默认值是 96。注意该寄存器只在复位模式下可写。

4.10 接收错误计数器

该寄存器显示接收错误计数器的值。它在复位模式下可写。总线关闭事件会把它复位为 0。

4.11 发送错误计数器

该寄存器显示发送错误计数器的值。它在复位模式下可写。总线关闭事件会把它复位为 0。

4.12 发送缓冲寄存器

发送缓冲被映射为地址 16 至 28 并且是只写的。发送缓冲的结构取决于将要发送的是标准帧（SFF）还是扩展帧（EFF），如下所示:

表 4-11 发送缓冲器

地址	写（SFF）	写（EFF）
16	发送帧信息	发送帧信息
17	发送识别码 1	发送识别码 1
18	发送识别码 2	发送识别码 2
19	发送数据 1	发送识别码 3
20	发送数据 2	发送识别码 4
21	发送数据 3	发送数据 1
22	发送数据 4	发送数据 2
23	发送数据 5	发送数据 3
24	发送数据 6	发送数据 4
25	发送数据 7	发送数据 5
26	发送数据 8	发送数据 6
27	—	发送数据 7
28	—	发送数据 8

表 4-12 发送帧信息（此位段在 SFF 和 EFF 帧中相同）

7	6	5	4	3	2	1	0
FF	RTR	—	—	DLC.3	DLC.2	DLC.1	DLC.0

- 7：FF 选择帧格式，1 = EFF，0 = SFF。
- 6：对于远程发送请求帧 RTR 应被置为 1。
- 5:4：不考虑。
- 3:0：DLC 指定数据长度代码并且应该是 0 到 8 之间的数值。如果大于 8，则 8 个字节将被发送。

表 4-13 发送标识符 1（此位段在 SFF 帧和 EFF 帧中相同）

7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

- 7:0：标识符的高 8 位。

表 4-14 发送标识符 2，SFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	—	—	—	—	—

- 7:5：SFF 标识符的低 3 位。
- 4:0：不考虑。

表 4-15 发送标识符 2，EFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

- 7:0：29 位 EFF 标识符的第 20 到第 13 位。

表 4-16 发送标识符 3，EFF 帧

7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

- 7:0：29 位 EFF 标识符的第 12 到第 5 位。

表 4-17 发送标识符 4，EFF 帧

7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	—	—	—

- 7:3：29 位 EFF 标识符的第 4 到第 0 位。
- 2:0：不考虑。

数据位段:

对于 SFF 帧，数据位段位于地址 19 到 26，对于 EFF 帧，位于 21 到 28。数据从位于最低地址的 MSB（最高位字节）开始发送。

4.13 接收缓冲寄存器

表 4-18 接收缓冲寄存器

地址	读（SFF）	读（EFF）
16	接收帧信息	接收帧信息
17	接收识别码 1	接收识别码 1
18	接收识别码 2	接收识别码 2
19	接收数据 1	接收识别码 3
20	接收数据 2	接收识别码 4
21	接收数据 3	接收数据 1
22	接收数据 4	接收数据 2
23	接收数据 5	接收数据 3
24	接收数据 6	接收数据 4
25	接收数据 7	接收数据 5
26	接收数据 8	接收数据 6
27	FIFO 中下一个报文的接收帧信息	接收数据 7
28	FIFO 中下一个报文的接收识别码 1	接收数据 8

表 4-19 接收帧信息（此位段在 SFF 和 EFF 帧中相同）

7	6	5	4	3	2	1	0
FF	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0

- 7：已接收报文的帧格式，1 = EFF，0 = SFF。
- 6：RTR 帧时为 1。
- 5:4：总为 0。
- 3:0：DLC 指定数据长度代码。

表 4-20 接收标识符 1（此位段在 SFF 帧和 EFF 帧中相同）

7	6	5	4	3	2	1	0
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

- 7:0：标识符的高 8 位。

表 4-21 接收标识符 2，SFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	RTR	0	0	0	0

- 7:5：SFF 标识符的低 3 位。
- 4：RTR 帧时为 1。
- 3:0：总为 0。

表 4-22 接收标识符 2，EFF 帧

7	6	5	4	3	2	1	0
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

- 7:0：29 位 EFF 标识符的第 20 到第 13 位。

表 4-23 接收标识符 3，EFF 帧

7	6	5	4	3	2	1	0
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

- 7:0：29 位 EFF 标识符的第 12 到第 5 位。

表 4-24 接收标识符 4，EFF 帧

7	6	5	4	3	2	1	0
ID.4	ID.3	ID.2	ID.1	ID.0	RTR	0	0

- 7:3：29 位 EFF 标识符的第 4 到第 0 位。
- 2：RTR 帧时为 1。
- 1:0：不考虑。

数据位段:

对于接收到的 SFF 帧，数据段位于地址 19 到 26，对于 EFF 帧则位于 21 到 28。

4.14 验收过滤寄存器

验收过滤器可以用来过滤掉不符合特定要求的报文。如果一个报文被过滤掉，它将不被放进接收 FIFO 里面，CPU 也不必处理它。

有两种不同的过滤模式：单过滤和双过滤。模式寄存器的第 3 位控制使用哪种模式。在单过滤模式下只使用一个 4 个字节的过滤器。在双过滤模式下使用两个更小的过滤器，如果匹配其中任何一个，则报文被接收。每个过滤器由两部分组成：接收代码和接收屏蔽。代码寄存器用来指定匹配的格式而屏蔽寄存器则指定不考虑的位。总共 8 个寄存器被用作接收过滤器，如下表所示。注意它们只在复位模式下被读写。

表 4-25 验收过滤寄存器

偏移地址	说明	硬件复位	软件复位
16	验收代码 0 寄存器（ACR0）	x	x
17	验收代码 1 寄存器（ACR1）	x	x
18	验收代码 2 寄存器（ACR2）	x	x
19	验收代码 3 寄存器（ACR3）	x	x
20	验收屏蔽 0 寄存器（ACM0）	x	x
21	验收屏蔽 1 寄存器（ACM1）	x	x
22	验收屏蔽 2 寄存器（ACM2）	x	x
23	验收屏蔽 3 寄存器（ACM3）	x	x

4.14.1 单过滤模式，标准帧

当在单过滤模式下接收一个标准帧，寄存器 ACR0:3 将会以以下的方式与接收到的报文比较：

- ACR0.7:0 和 ACR1.7:5 与 ID.28:18 比较。
- ACR1.4 与 RTR 位比较。
- ACR1.3:0 未使用。
- ACR2 和 ACR3 与数据字节 1 和 2 比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

4.14.2 单过滤模式，扩展帧

当在单过滤模式下接收一个扩展帧，寄存器 ACR0:3 将会以以下的方式与接收到的报文比较：

- ACR0.7:0 和 ACR1.7:0 与 ID.28:13 比较。
- ACR2.7:0 和 ACR3.7:3 与 ID.12:0 比较。
- ACR3.2 与 RTR 位比较。
- ACR3.1:0 未使用。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

4.14.3 双过滤模式，标准帧

当在双过滤模式下接收一个标准帧，寄存器 ACR0:3 将会以以下的方式与接收到的报文比较：

过滤器 1:

- ACR0.7:0 和 ACR1.7:5 与 ID.28:18 比较。
- ACR1.4 与 RTR 位比较。
- ACR1.3:0 与数据字节 1 的高半字节比较。
- ACR3.3:0 与数据字节 1 的低半字节比较。

过滤器 2:

- ACR2.7:0 和 ACR3.7:5 与 ID.28:18 比较。
- ACR3.4 与 RTR 位比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

4.14.4 双过滤模式，扩展帧

当在双过滤模式下接收一个扩展帧，寄存器 ACR0:3 将会以以下的方式与接收到的报文比较：

过滤器 1:

- ACR0.7:0 和 ACR1.7:0 与 ID.28:13 比较。

过滤器 2:

- ACR2.7:0 和 ACR3.7:0 与 ID.28:13 比较。

AMR 寄存器里相应的位选择是否比较的结果没关系。屏蔽寄存器里一个置 1 的位表示不考虑。

4.15 接收报文计数器

位于地址 29 的接收报文计数器保持当前储存在接收 FIFO 里的报文的数量。最高 3 位总为 0。

4.16 公共寄存器

在 BasicCAN 和 PeliCAN 模式下有 3 个公共寄存器，它们具有相同的地址和相同的功能。它们是时钟分频寄存器和总线定时寄存器 0 和 1。

4.17 模式选择寄存器

此寄存器的功能是选择 PeliCAN 和 BasicCAN 模式。

表 4-26 时钟分频寄存器（CDR）

符号.位	名称	功能	硬件复位	软件复位
CDR.7	CAN 模式	1：PeliCAN；0：BasicCAN。	0	x
CDR.6:4	—	保留（总是 0）。	0	0
CDR.3:0	—	保留。	0	x

- x：复位不影响该寄存器或位。

4.18 总线定时 0 寄存器

表 4-27 总线定时 0 寄存器（BTR0）

符号.位	名称	功能	硬件复位	软件复位
BTR0.7:6	SJW	同步跳跃宽度。	0	x

符号.位	名称	功能	硬件复位	软件复位
BTR0.5:0	BRP	波特速率预设值。	0	x

- x: 复位不影响该寄存器或位。

CAN Core 的系统时钟由以下计算:

$$t_{scl} = 2 \times t_{clk} \times (BRP + 1) \quad \text{(式 3-1)}$$

其中 tscl 是系统时钟。

同步跳跃宽度定义了在一次重同步中一个位周期里有多少个时钟周期（tscl）可以调整。

4.19 总线定时 1 寄存器

总线定时 1 寄存器（BTR1）的位分配（地址 7）。

表 4-28 总线定时 1 寄存器（BTR1）

符号.位	名称	功能	硬件复位	软件复位
BTR1.7	SAM	1: 对总线采样 3 次。 0: 对总线采样 1 次。	0	x
BTR1.6:4	TSEG2	时间段 2。	0	x
BTR1.3:0	TSEG1	时间段 1。	0	x

- x: 复位不影响该寄存器或位。

CAN 总线的位周期由 CAN 的系统时钟和时间段 1 和 2 决定，如下面的等式所示:

$$t_{tseg1} = t_{scl} \times (TSEG1 + 1) \quad \text{（式 3-2)}$$

$$t_{tseg2} = t_{scl} \times (TSEG2 + 1) \quad \text{(式 3-3)}$$

$$t_{bit} = t_{tseg1} + t_{tseg2} + t_{scl} \quad \text{(式 3-4)}$$

附加的 t_{scl} 项来自初始的同步段。采样在位周期的 TSEG1 和 TSEG2 之间完成。

5 时序图

5.1主机接口时序图

5.1.1 INTEL主机模式下的操作时序

当 8051 单片机模式设为 INTEL 模式（即 MODE＝1，代码对应信号为 mode）的时候，8051 对 OBTCAN 的读操作和写操作的时序分别如图 5-1 和图 5-2 所示：

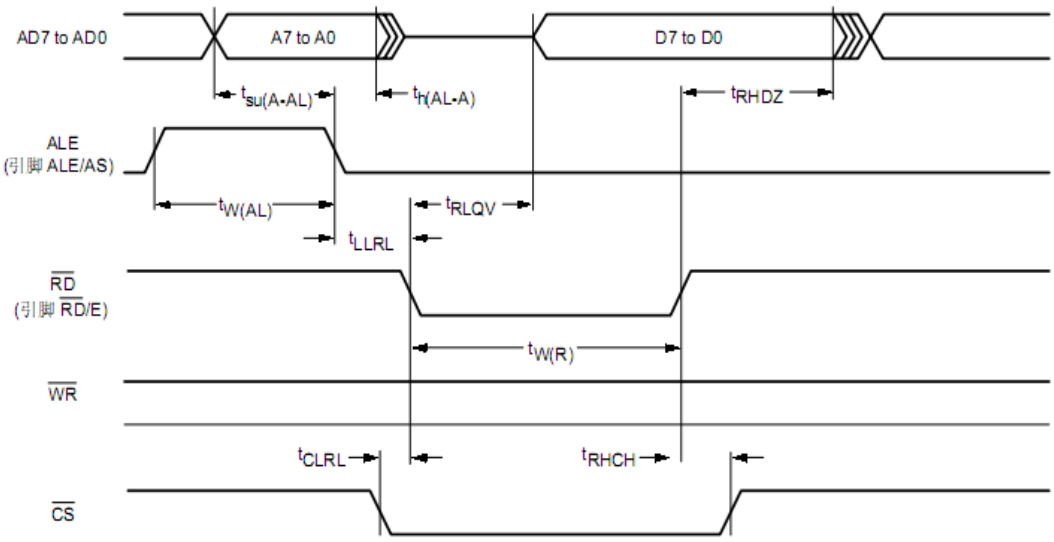


图 5-1 INTEL 模式读时序图

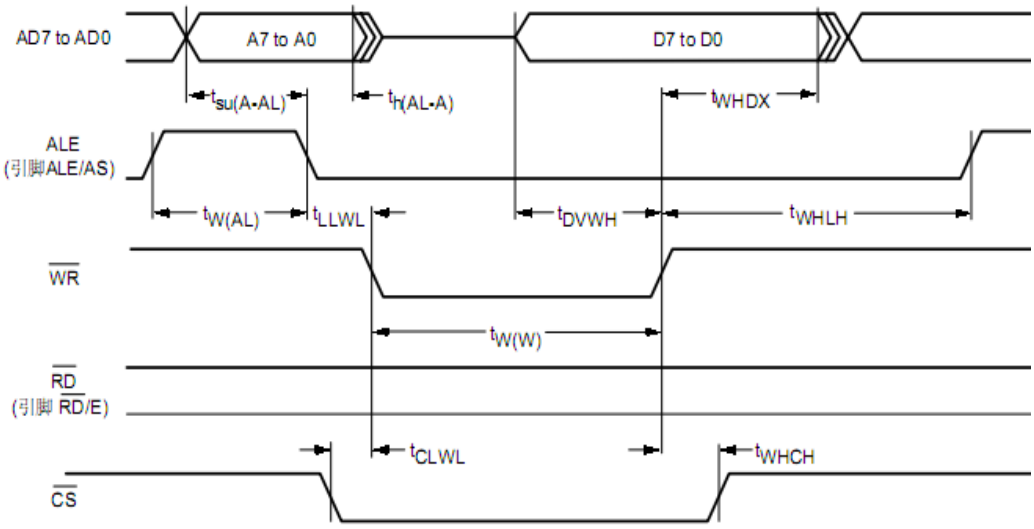


图 5-2 INTEL 模式写时序图

5.1.2 MOTOROLA主机模式下的操作时序

当 8051 单片机模式设为 MOTOROLA 模式（即 MODE＝0，代码对应信号为 mode）的时候，8051 对 OBTCAN 的读操作和写操作的时序分别如图 5-3 和图 5-4 所示：

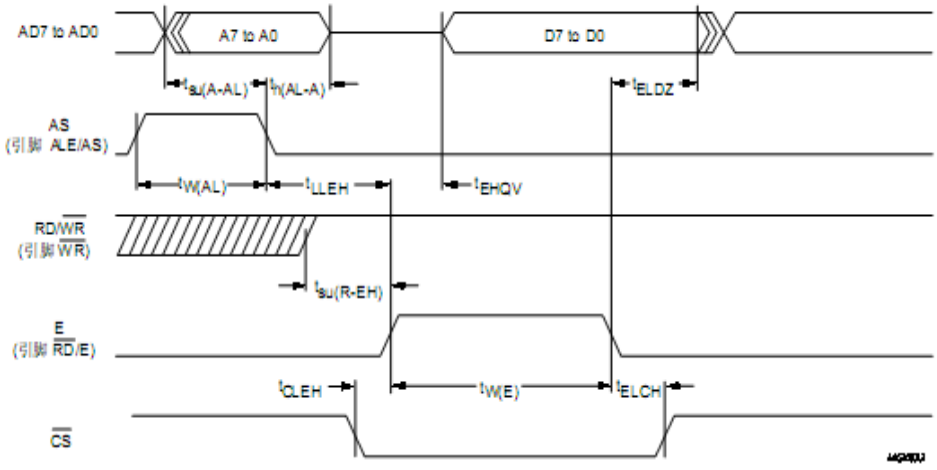


图 5-3 MOTOROLA 模式读时序图

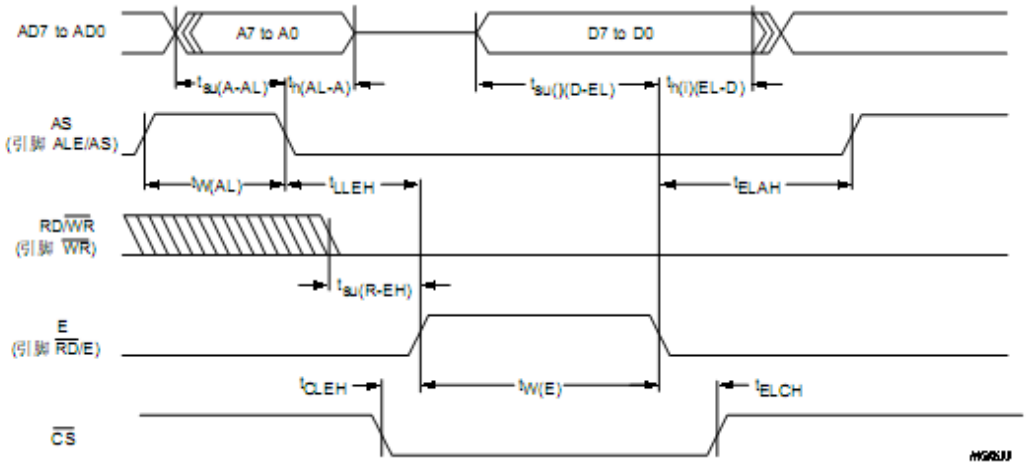
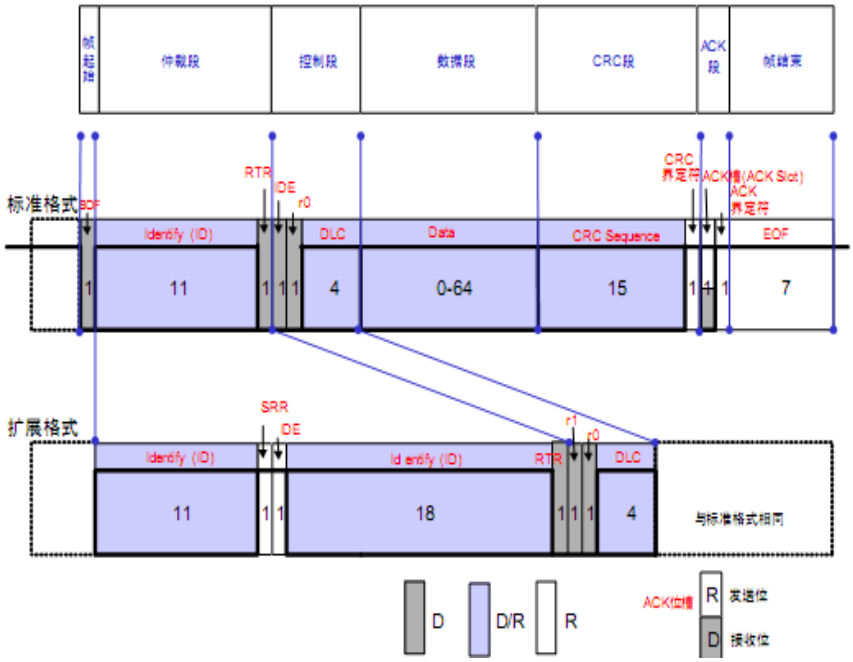


图 5-4 MOTOROLA 模式写时序图

5.2总线信号数据帧组成



6 应用案例

6.1 基于 8051 单片机的CAN IP核应用方案

此应用方案的系统分成四个子模块，分别是主控制器模块（8051 单片机）、FPGA 模块（烧写 CAN IP 核）、显示数据的接口模块（UART 接口）、电源模块。主要框图为：

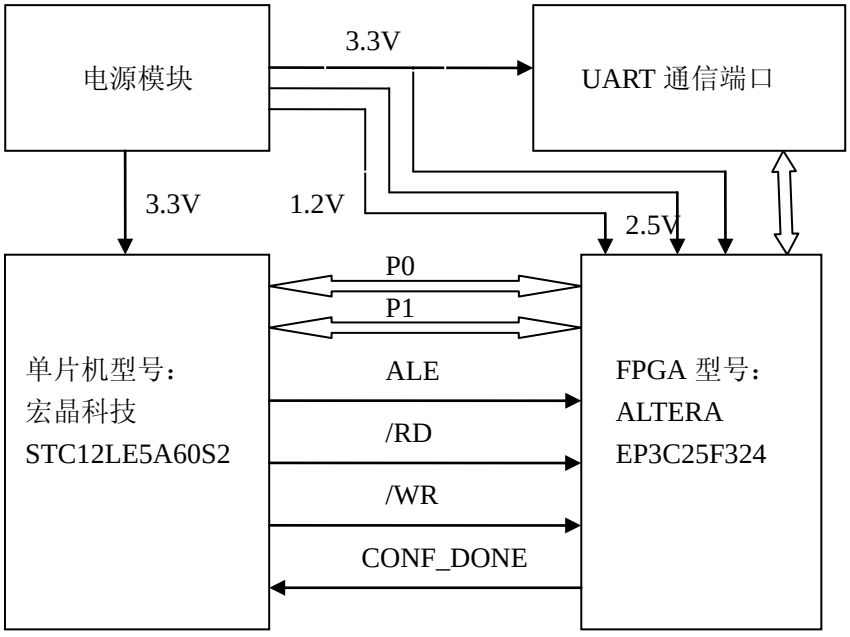


图 6-1 CAN IP 核应用方案图

基于 8051 的 CAN 应用系统既可用来作检验 CAN IP 核的测试系统，又可以用来作发送或是接收 CAN 总线信号的仪器。

7 附录

7.1附录一：Basic模式下发送数据示例程序

Can1_Basic_Transmit_ini()

```
signed char Can1_Basic_Transmit_ini()                    //发送数据
{
 printf("Basic Transmit...    \n");
 printf("CAN1DOS = %#x\n", CAN1DOS);
 printf("CAN1TBS = %#x\n", CAN1TBS);
 printf("CAN1TCS = %#x\n", CAN1TCS);
 printf("CAN1RS = %#x\n", CAN1RS);
 printf("CAN1TS = %#x\n", CAN1TS);
 printf("CAN1ES = %#x\n", CAN1ES);
 printf("CAN1BS = %#x\n", CAN1BS);
*/
//       while(1);
         if(CAN1TBS == 0)
         {
         printf("CAN1TBS = %#x\n", (int)CAN1TBS);
                return -1;
         };
/*
 while(CAN1TBS == 0)
 {
         printf("CAN1SR = 0x%x\n",(int)CAN1SR);
         DelayNMs(200);
 }
         printf("xxxx\n");
*/
//       CAN1TX_ID1 = 0xea;
//       CAN1TX_ID2_RTR_DLC =0x28;
 CAN1TCF = 0xea28;

 CAN1TDBB1 = 0x21;
 CAN1TDBB2 = 0x31;
 CAN1TDBB3 = 0x41;
```

```
CAN1TDBB4 = 0x51;
CAN1TDBB5 = 0x61;
CAN1TDBB6 = 0x71;
CAN1TDBB7 = 0x81;
CAN1TDBB8 = 0x91;
printf("CAN1TX_ID1 = %#x\n", (int)CAN1TX_ID1);
printf("CAN1TX_ID2_RTR_DLC = %#x\n", (int)CAN1TX_ID2_RTR_DLC);
printf("CAN1TDBB1 = %#x\n", (int)CAN1TDBB1);
printf("CAN1TDBB2 = %#x\n", (int)CAN1TDBB2);
printf("CAN1TDBB3 = %#x\n", (int)CAN1TDBB3);
printf("CAN1TDBB4 = %#x\n", (int)CAN1TDBB4);
printf("CAN1TDBB5 = %#x\n", (int)CAN1TDBB5);
printf("CAN1TDBB6 = %#x\n", (int)CAN1TDBB6);
printf("CAN1TDBB7 = %#x\n", (int)CAN1TDBB7);
printf("CAN1TDBB8 = %#x\n", (int)CAN1TDBB8);

printf("CAN1SR = %#x\n", (int)CAN1SR);
printf("发送中。。。 \n");

while(CAN1TS == 1);
CAN1CMD = 0x1;
DelayNMs(5);
printf("CAN1SR = %#x\n", (int)CAN1SR);
//      while(1);
return 0;
//      printf("发送完成。。。\n");
```

7.2附录二：Basic模式下接收数据示例程序

Can1_Basic_Receive ()

```
signed char Can1_Basic_Receive( unsigned int *ID,char *buff,unsigned int *DLE )
{
 unsigned char i;
 signed char error;
 if( ( CAN1RBS == 1 )||( CAN1DOS == 1 ) )
 {
        *DLE = CAN1RDLC;
        if( CAN1RDLC >8 )
        {
              *DLE = 8;
        }
        *ID = CAN1RID;
        if( CAN1RRTR == 0 )                          //数据帧
```

```
        {
            for( i = 0; i < *DLE;i++ )
            {
                *buff++ = *(&CAN1RDBB1 + i);
            }
            error = 0;
        }
        else
        {
            error = -2;                                    //请求帧
        }
                                                           //
        CAN1CDO = 1;
        CAN1RRB = 1;
    }
    else
    {
        error =   -1;                                      //无数据
    }
    return error;
}
```

7.3附录三：Peli模式下发送数据示例程序

Can1_Peli_Transmit_ini()

```
signed char Can1_Peli_Transmit_ini()                       //发送数据
{
 printf("Peli Transmit..... \n");
 printf("CAN1DOS = %#x\n", CAN1DOS);
 printf("CAN1TBS = %#x\n", CAN1TBS);
 printf("CAN1TCS = %#x\n", CAN1TCS);
 printf("CAN1RS = %#x\n", CAN1RS);
 printf("CAN1TS = %#x\n", CAN1TS);
 printf("CAN1ES = %#x\n", CAN1ES);
 printf("CAN1BS = %#x\n", CAN1BS);
*/
//      while(1);
 while(STATUS_0&0x10 != 0);
 while(STATUS_0&0x0c == 0);

//      printf("xxxx\n");

//      CAN1TX_ID1 = 0xea;
```

```
//      CAN1TX_ID2_RTR_DLC =0x28;
 TX_FI_SFF_0    = 0x88;              //ff=1 SFF 标准格式，ff=0 EFF 扩展格式  DLC.3=1  数据长度 8
 TX_ID1_EFF_0 = 0x12;
 TX_ID2_EFF_0 = 0x34;
 TX_ID3_EFF_0 = 0x56;
 TX_ID4_EFF_0 = 0x78;

 TX_DATA1_EFF_0 = 0x21;
 TX_DATA2_EFF_0 = 0x31;
 TX_DATA3_EFF_0 = 0x41;
 TX_DATA4_EFF_0 = 0x51;
 TX_DATA5_EFF_0 = 0x61;
 TX_DATA6_EFF_0 = 0x71;
 TX_DATA7_EFF_0 = 0x81;
 TX_DATA8_EFF_0 = 0x91;
 printf("TX_FI_SFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7060)))));
 printf("TX_ID1_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7061)))));
 printf("TX_ID2_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7062)))));
 printf("TX_ID3_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7063)))));
 printf("TX_ID4_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7064)))));
 printf("TX_DATA1_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7065)))));
 printf("TX_DATA2_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7066)))));
 printf("TX_DATA3_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7067)))));
 printf("TX_DATA4_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7068)))));
 printf("TX_DATA5_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X7069)))));
 printf("TX_DATA6_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X706a)))));
  printf("TX_DATA7_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X706b)))));
 printf("TX_DATA8_EFF_0 = %#x\n", (int)*(((unsigned char xdata*)((0X706c)))));

 COMMAND_0 = 0x10;
 DelayNMs(5);
 printf("STATUS_0 = %#x\n", (int)STATUS_0);
      while((STATUS_0&0x08) == 0 )
 {
        printf("sending...\n");
        printf("STATUS_0 = %#x\n", (int)STATUS_0);
 }

      DelayNMs(2000);
      printf("发送完成。。。\n");
 return 0;
//      printf("发送完成。。。\n");
}
```

7.4附录四：Peli模式下接收数据示例程序

Can1_Peli_Receive()

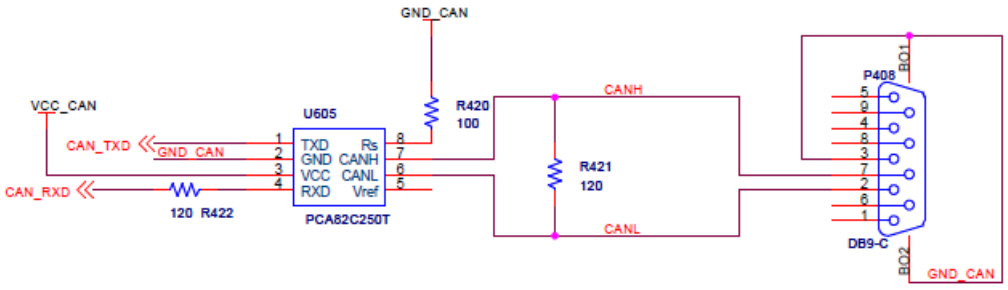
```
void Can1_Peli_Receive()
  {
      unsigned char i, n, TempData;
      unsigned char PeliCANReceiveData[ 13 ] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
      while(STATUS_0 != 0xd)
      {
            printf("no message receive...\n");
            printf("the stats is %#x\n",(int)STATUS_0);
            DelayNMs(1000);
      };
      printf("have a message ...\n");
      PeliCANReceiveData[ 0 ] = RX_FI_SFF_0;                  //从数据接收缓冲器中读数据
      printf("PeliCANReceiveData[0] = %#x\n", (int)*(((unsigned char xdata*)((0X7010)))));
      n = PeliCANReceiveData[ 0 ]&0x0F;                       //提取数据帧个数
      n = n + 4;
      for( i = 0; i < n; i++ )                                //读标识符及数据帧
           {
      PeliCANReceiveData[ 1 + i ] = (int)*(((unsigned char xdata*)((0X7011 + i))));
      printf("PeliCANReceiveData[%d] = %#x\n", (int)(i+1),(int)*(((unsigned char xdata*)((0X7011 + i)))));
           };
      COMMAND_0 = 0x04;                                       //释放数据接收缓冲器

      TempData = ARBITRATION_LOST_CAPTURE_0;                  //读仲裁丢失捕捉寄存器和错误捕捉寄
存器
      TempData = ERROR_CODE_CAPTURE_0;
      while( (STATUS_0&0x01) == 1 )                           //判缓冲区是否已空
      {
      COMMAND_0 = 0x04;
      }
      return ;
 }
```

8 订货信息

序号	产品型号	产品描述
1	OBTIP-CAN-F	ASIC 版本固核（ASIC 网表）
2	OBTIP-CAN -V	FPGA 版本固核（FPGA 网表）

3	OBTIP-CAN -S	软核（RTL 源码）

9 典型接口图



10 资源利用情况

Altera Cyclone III: LE: 24,593 Memory bits: 5,248
Xilinx Virtex2: Slices : 3,500 LUTs : 6,109 Memory bits : 5,248 Flip-Flops :1,239
Xilinx Virtex5: Slices : 1,346 LUTs : 3,410 Memory bits : 5,248 Flip-Flops :1,260
Actel ProASIC3: D-flip-flops(CORE) : 27,308