摘要

区域控制网络总线是当今汽车企业中使用的最为广泛的一类现场总线。在全球生产的超过50％的汽车里都使用这一协议作为内部的传输总线。它的高质特性是基于其设计良好的协议。它是一种广播式的异步通信协议，有着独特而良好的同步方法，可以检测到各种各样的错误并且相应地处理这些错误，因此它是非常健壮的。它所能够支持的传输速率最大可至1Mbit/s。传统的CAN协议是使用自底向上的设计方法，根据现有的单元芯片搭建而不是系统本身的功能需求，这样对系统的修改更新带来了非常的大的麻烦。笔者受当今数字设计潮流的影响，使用Verilog硬件描述语言和自顶向下的设计方法设计并实现了CAN协议的发送部分和错误处理部分。笔者使用QuartusII软件进行编译与综合，使用ModelSim软件进行仿真与模拟。经过细致的测试后，笔者将程序下载到Altera公司的FPGA板上。发送方与接受方的联结使用构成了一个能够完整执行CAN协议的传输芯片。最终，由笔者设计的这一个IP核经受了严格的测试。

Abstract

Controller Area Network (CAN) is the most frequently used field bus protocol in automation industry. It has been used as the main transmission protocol in over 50% cars produced all over the world. Its superior quality is based on its well-designed protocol. It is an asynchronized communication protocol using broadcasting theory. It can detect various errors and handle the errors accordingly, so it is very robust. And it can support up to 1M bit/s transmission speed. Traditional implementation of CAN protocol includes schematic design and manual labor. This whole process has to be involved with accessible chips rather than the strict function demand. Influenced by the current digital design trend, the author utilizes Verilog Hardware Description Language and Top-Down method to design and implement the transmission and error handling part of CAN protocol using Quartus II to compile and synthesize the whole project and ModelSim to simulate the whole process. After meticulous test, the author downloaded the program on Altera ACEX1K FPGA. The combination of transmission part and receiving part constructs an intact system that fully implements CAN protocol. Finally, the IP core co-coded by the author stands rigorous test.

目录

1. 背景介绍

1.1 IP复用技术、FPGA技术、现代数字电路设计与硬件描述语言

1.1.1 IP复用技术

IP（Intellectual Property 知识产权） *内核是满足特定规范，并能在设计中复用的功能模块。根据功能不同，内核可进行参数。*IP内核主要有三种：软核、硬核、固核。*软核通常以可综合的HDL 提供，因此具有较高的灵活性，并与具体的实现工艺无关，其主要缺点是缺乏对时序、面积和功耗的预见性以及不易进行知识产权的保护。硬核则以经过完全的布局布线的网表形式提供，这种硬核既具有可预见性，同时还可以针对特定工艺或购买商进行功耗和尺寸上的优化，但硬核缺乏灵活性且可移植性差。固核则是软核和硬核的折衷。*大多数应用于FPGA 的IP 内核均为软核，软核有助于用户调节参数并增强可复用性，并以加密方式给出，以保护知识产权。

1.1.2 FPGA技术

现场可编程门阵列（FPGAs）是作为可编程逻辑设备（PLDs）和专用集成电路（ASICs）的一种替代方案在1984年开发出来的。顾名思义，FPGAs的最大好处就是可以快速编程。FPGAs并不像先前其它PLD设备那样只能进行一次编程，它能够（大多数情况下）进行反复编程，这样便给设计者们提供了多次机会来调整他们的电路。

使用FPGAs没有高额的、因不可重用工程（NRE）而带来的花费，而且，它使得由冗长的、令人极端头疼的电路模板制造工作所产生的等待时间也大为减少。通常，使用FPGA技术进行开发设计的时候，由于对一个特定的逻辑设计要进行多次重复的工作，因此使得这个过程有点类似于软件设计。所以，使用FPGAs作为实现平台时，经常会带来创新的设计。

FPGAs融合了逻辑离散并且规模较小的PLDs的低复杂性特点，以及价格昂贵的ASICs的可定制的特点。FPGAs由一组用软件配置的逻辑块组成，而可编程的输入/输出块则包围着这些逻辑块，它们之间用可编程的内部网络连接。

就在前几年，规模最大的FPGA设计中系统门的数量数以万计，并且只能工作在40MHz的频率下。在那个时候，为了设计出最先进的部件，往往要花费150美元以上。而今天，FPGAs提供了数百万逻辑门的容量，300MHz的工作频率，不到10美元的花费，以及一些功

能完整的模块，例如处理器和存储器等。

可见，FPGA是目前为止实现硬件设计软件化的最合适（功能强大且价格低廉）的硬件基础。

1.1.3现代数字电路设计的特点与优势

传统的电路设计基于印刷电路版和手工方式，是一种自顶向上的方式，即根据目前有什么样的芯片单元从而搭建出一个系统，来完成设计任务，这使任务的完成的灵活性降低，无法较为灵活地更改设计。即使是系统需求的一个微小的改动，也需要重新构建系统，这无疑使电路工程师的工作强度大大增加。更为重要的是，当今的IC设计已步入了百万门级超大规模，没有任何一个工程师团队可以以传统的方式完成一个如此大规模的系统。在这种背景下，基于HDL(Hardware Description Language硬件描述语言)应运而生。除了上述HDL设计方式较传统设计方式的“量”上的优势，其更为重要的“质”的优势还在于：首先，这种基于语言的设计是独立于当前的技术，即，也许当前的技术实现一个逻辑功能需要10微秒的延迟，而一年后的技术可使此延迟降到5微秒，但由于语言层对这些具体的技术是不可见的，所以一年前编写的代码依然可用。其次，HDL语言是实现IP复用技术的一种方便途径。而其最重要的优势在于HDL语言可被直接综合在FPGA上，生成相应的电路，从而避开了复杂的手工设计过程（如卡诺图的化简等）。总之，应用HDL语言进行数字设计是当前技术的最前沿也是应用最为广泛的技术。例如，我国自行开发的“龙芯”CPU芯片就是采用“Verilog+FPGA”的逻辑设计模式，并且其水平已达到奔腾二代的水准，并且使用了其FPGA上150万个逻辑单元的85%，可见其规模之大及此技术功能之强大。

1.1.4硬件描述语言

目前主流的硬件描述语言分为两种，一是VHDL语言，一是Verilog语言，这两种语言都是经过IEEE协会认证为标准的硬件描述语言。

VHDL语言全称为Very High Speed Integrated Circuit Hardware Description Language,即高速集成电路硬件描述语言，是最早由美国国防部开发的一门语言，语言结构复杂、严谨，是国内外**教学**的首选语言。但是这门语言不适于初学者学习，但一旦掌握此门语言将对硬件描述语言的整体把握打下基础。其于1987年被列为国际标准。

Verilog HDL语言全称是Verilog Hardware Description Language，全称为Verilog Hardware Description Language，即Verilog 硬件描述语言。它是由GDA（Gateway Design Automation）公司于1983年首创的，并于1995年成为IEEE的标准，于2001年又推出了其改进版。因为这种语言好学易懂，在较短的时间内可以掌握，在工业生产中应用较广，如上述的龙芯工程以及Cisco的路由器都是使用的这种语言。

但是，作为硬件描述语言共通的地方在于其对电路设计的行为级描述，以语言的方式来描述一个模块（Verilog）或一个实体(VHDL)，来表达实现每个功能的模块，定义好输入

输出接口，再在模块的内部定义其各种行为。具体到Verilog上，就是用wire、register来表示连线与寄存器，采用各种触发信号（应用于时序逻辑），电平变化信号（应用于组合逻辑），采用某种控制逻辑来完成对电路的设计。

综上，在此次毕设中，我们采用的是Verilog语言以及Altera公司提供的Acex FPGA板，采用编写软核的方法，对一个总线协议实现了电路设计。

1.2 现场总线的由来、CAN现场总线的发展历史、在业界的使用及协议本身

1.2.1 现场总线

现场总线：现场总线是连接智能现场设备和自动化系统的数字式、双向传输、多分支结构的通信网络，它的关键标志是能支持双向、多节点、总线式的全数字通讯。目前使用的现场总线有：AnyBus，CAN ，Profibus，Fieldbus，WorldFIP，P-NET，LonWorks，INTERBUS，DNET，CNET，LIGHTBUS，MODBUS，CC-LINK。在现场总线中，CAN是唯一被国际标准化组织批准的现场总线。

1.2.2 CAN总线的发展历史

CAN（Controller Area Network），区域控制网络总线，是由德国的Bosch公司（德国BOSCH公司系德国著名的跨国公司，以生产电动工具而闻名于世。在2003年世界500强企业中排名第94位，年销售额为369亿欧元，其中汽车配件销售额为242亿欧元，为全球最大的汽车配件供应商）制定的一套基于汽车内部信息传输的总线,为串行通讯协议，能有效地支持具有很高安全等级的分布实时控制。

在汽车电子行业里，使用CAN 连接发动机控制单元、传感器、防刹车系统、等等，其传输速度可达1 Mbit/s。同时，可以将CAN 安装在卡车本体的电子控制系统里，诸如车灯组、电气车窗等等，用以代替接线配线装置。

其发展历程如下：

1983年　Bosch公司内部的一个关于现场总线的项目正式立项

1986年　CAN总线项目正式推出，标志着CAN协议的正式出台

1987年　由Intel和Philips两家公司分别研制出第一块基于CAN总线的传输芯片，标志着CAN总线在工业中大规模使用的开始。

1991年　Bosch公司的CAN改进版（CAN2.0）推出，这一款协议加大扩充了CAN总线中可加载的不同类别节点数量。同年，Kvaser（世界上最好的CAN解决方案公司）推出了基于CAN更高层的协议，标志着CAN协议的高层应用开始被广泛利用。

1992年　CiA(CAN in Automation 即自动化中的CAN应用)联盟建立，标志着国际化的用户和制造商的联盟形成，该组织制定出CAN的应用层协议。同年世界上第一辆使用CAN协议的梅塞德斯－奔驰轿车下线。

1993年　CAN协议由ISO组织正式宣布，定为ISO11898协议

1994年　由CiA举办的第一届国际CAN大会召开

1995年　同Allen-Bradley（后被Rockwell公司吞并）公司推出了DeviceNet协议，同年ISO11898修订版出台,CiA也推出了CANopen协议

2000年　CAN的时间驱动通信协议产生

可见，CAN以迅猛之势占领全球市场，据不完全统计，CAN总线占据了全球汽车内部总线市场的一半多，各种名牌轿车无不使用CAN协议。

CAN 的应用范围很广，从高速的网络到低价位的多路接线都可以使用CAN。除了汽车电子行业中，CAN在其他领域中也占有着重要的位置。据笔者的了解，CAN总线至少在以下的行业中得到了广泛的应用：单片机多机连网及局域网、大学食堂的划卡系统、防火灾预警系统、变电站实时控制信息采集等。

1.2.3 CAN协议本身

CAN协议主要分为两层：物理层和数据链路层。

1.2.3.1 物理层

物理层规定了CAN总线传输的物理介质（双绞线）以及其逻辑电平的表示方－－采用差分信号，有电压差时表示逻辑“0”，即显性位；无电压差时表示逻辑“1”，即隐性位“1”。总线采用线与逻辑，即若干个隐性位可被一个显性位湮灭，以保证在一个时刻内总线上只有一个固定的数值。报文的位流根据不归零方法编码，在整个位时间里，位的电平要为显性，要么为隐性。

CAN的传输速率最高可至1Mbit/s，此时最大的传输距离为40米，在不同的系统中可设置不同的速率，但是在同一个系统中这个速率是固定的。

1.2.3.2 数据链路层

数据链路层是CAN协议的核心部分。它规定了如下几个方面：

1.2.3.2.1帧类型（由于这是在数据链路层，所以传输的基本单元是帧）

- 数据帧：数据帧携带数据从发送器至接收器。其由7 个不同的位场组成：帧起始、仲裁场、控制场、数据场、CRC 场、应答场、帧结尾。数据场的长度可以为0。其中，

帧起始，它标志数据帧和远程帧的起始，由一个单独的“显性”位组成。只在总线空闲时，才允许站开始发送（信号）。所有的站必须同步于首先开始发送信息的站的帧起始前沿。仲裁场由11位ID和1位RTR组成，ID位标志着节点的唯一标识值，相当于个自的“主码”。RTR标志着此帧是传递数据还是索要数据。在ID场中，最高7位不可以全为“1”。控制场由2个保留位（其中一位在以后的扩展定义中标志了是否是标准帧还是扩展帧），之后是4位的数据长度控制场，标志着数据的长度从1－8。CRC场是对从帧起始至数据场结束的循环冗余检验。其生成多项式为：$X_{15}+X_{14}+X_{10}+X_8+X_7+X_4+X_3+1$，根据协议中的算法产生15位的CRC校验码。CRC场的最后一位是CRC结束标志位，为隐性。ACK场是由接收方发送的1位显性位来湮灭发送方发送的1位隐性位，标志着数据的传输全部接到。ACK场的最后一位是ACK结束标志位，为隐性。帧结尾是连续7位隐性位，标志着一帧的发送完成。

- 远程帧：总线单元发出远程帧，请求发送具有同一识别符的数据帧。
- 错误帧：任何单元检测到一总线错误就发出错误帧。错误帧由“错误标志”（6位）和“错误结尾”（8位）两部分组成。在不同的错误情况下，错误标志可以有两种即全为显性位或全为隐性位。
- 过载帧：过载帧用以在先行的和后续的数据帧（或远程帧）之间提供一附加的延时。数据帧（或远程帧）通过帧间空间与前述的各帧分开。

1.2.3.2.2 CAN总线的位定时及同步

CAN总线的位定时起到了指挥棒的作用，它把标称位时间划分成了几个不重叠时间的片段，它们是：同步段（SYNC_SEG）传播时间段（PROP_SEG），相位缓冲段1（PHASE_SEG1），相位缓冲段2（PHASE_SEG2）。

CAN节点的同步是指接收方与发送方的同步。同步分为两类一类是硬同步，一类是重同步。硬同步指在总线空闲时，接收方检测到了一个下降沿，所有接收节点进入硬同步。重同步是指在一帧的送过程中，接收节点检测到了一个下降沿，则它们进入重同步。

1.2.3.2.3 CAN协议中的总线仲裁

在传输过程中由于传输ID的优先级不同（显性为高，隐性为低），会出现仲裁，每次优先级低的传输节点退出发送，直到下次总线空闲。

1.2.3.2.4 CAN协议的错误种类

共五类，分别为位错误、位填充错误、CRC错误、ACK错误、CRC错误。

1.2.3.2.5 CAN协议的错误限制

分为三类节点：活跃节点（错误较少）、被动节点（较多错误）、总线关闭（错误过多被关闭）。

1.2.3.2.6 发送方和接收方的定义

- 发送方：产生报文的单元称作这个报文的发送方。当总线空闲或该单元失去仲裁时，这个单元就不是发送方。
- 接收方：如果总线不空闲且此单元不是发送方，则其是接收方。

2．需求分析及前期总体设计

本着自顶向下的设计理念，首先要确定需求分析。

2.1需求分析及相应的解决方案

1．本组的课设是基于Nios系统、Avalon总线的汽车电子的模拟过程，所以要首先确定与对外的接口及功能（见表2.1及表2.2）。
2．本组的实验只涉及为数不多的控件单元，所以决定采用CAN的标准格式（即11位ID，可以标志2048种节点，足够满足本组的需求）。同时由于传输量不大，省去过载帧的引入，同时也省去三次采样的工作（即减少相应的滤波和去除毛刺的工作）。因此可以把接收缓冲区定得较小。
3．本组的传输速率不需要很高，可以将基准脉冲适当调慢。
4．需要很好的处理出错的功能，按照协议将所有的错误判断都列入，并设计好重发机制。还要对错误种类进行识别。
5．对发生错误多的节点要实施控制，进行错误限制处理。
6．在未发时，可以按照需要中止下一帧的发送。
7．实现发送时的总线仲裁。并在退出发送后可以在适当时候进行重发。
8．要实现较好的纠错功能和同步传输功能，使用CRC技术和位填充技术。
9．要实现对不同种类节点的识别。

2.2 整体设计框图

根据上述功能，本人进行的前期整体设计如下：

2.2.1最项层模块

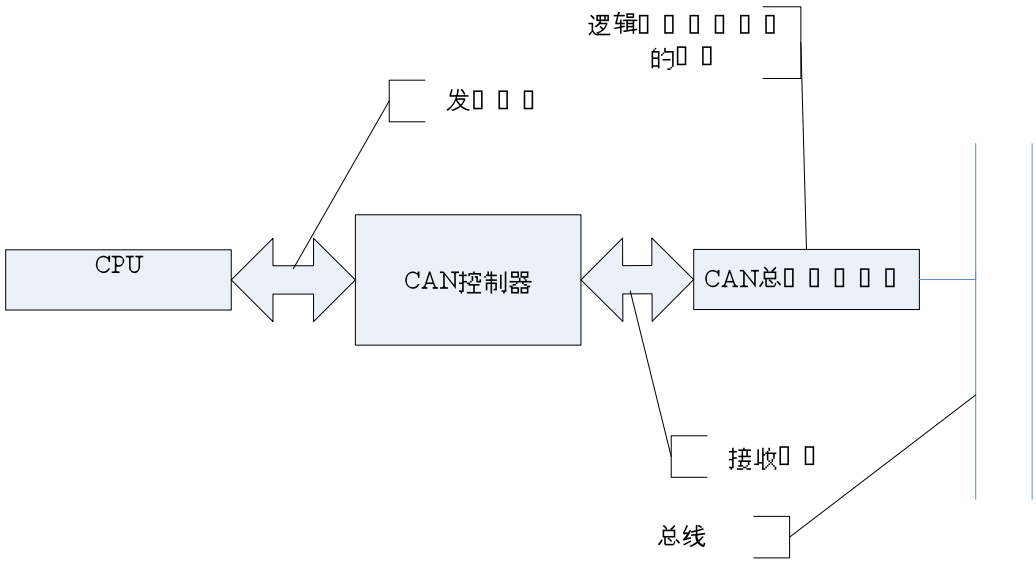


图2.1 CAN控制器的顶层模块图

在这一模块中，CPU可以是中心计算机，也可以是其它的网络层或应用层的控制单元，其通过一系列的接口定义实现对CAN控制器的设定。CAN控制器里是实现了CAN协议的逻辑控制单元，实现了发送、接收功能。CAN总线驱动器是把CAN控制器提供的逻辑信号转化为差分电平信号。总线是一对双绞线。

2.2.2对CAN控制器的划分

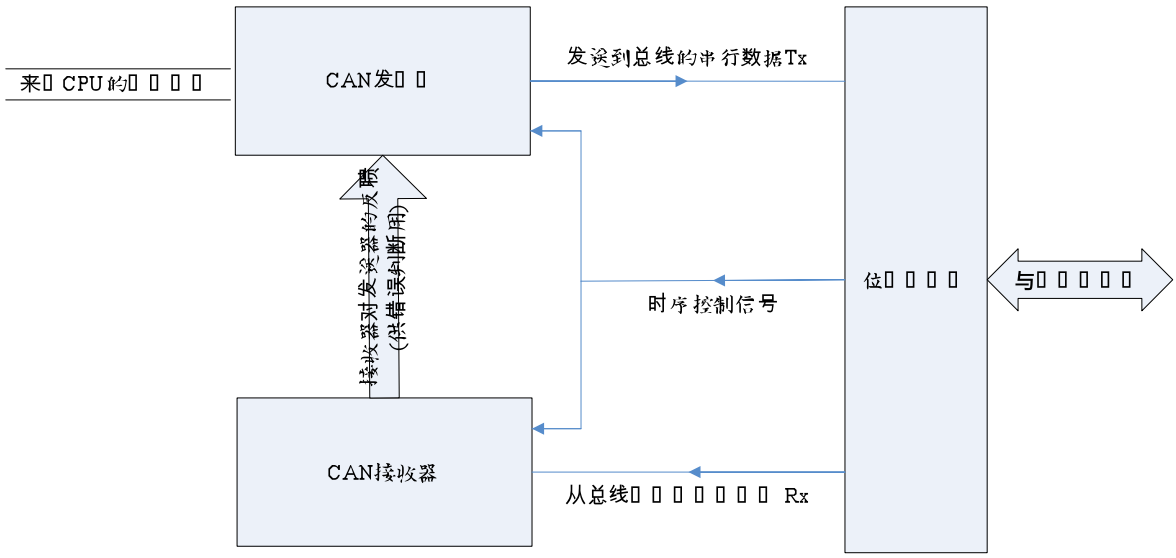


图2.2对CAN控制器划分的模块图

其中，CAN发送器把来自CPU的并行数据记录然后在位时序单元的控制下，通过CAN驱动器将数据打到总线上；而CAN接收器在位时序单元的控制下，通过CAN驱动器从总线上获得当前总线值。

2.2.3对CAN发送功能模块的具体划分

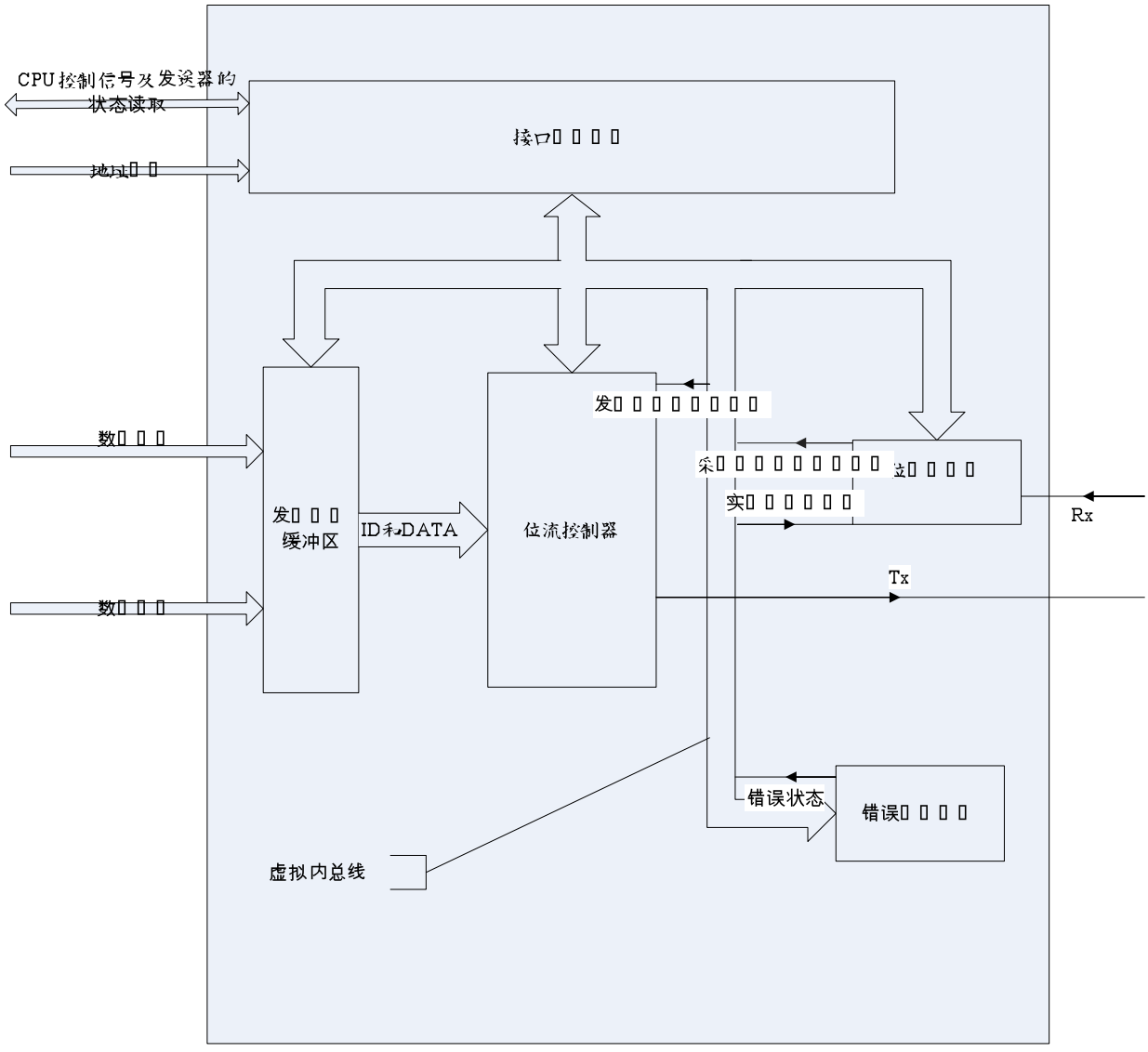


图2.3 CAN控制器发送功能模块

其中，接口逻辑单元为CAN控制器对外的接口，其中规定了CAN内部寄存器的地址映射。

发送数据缓冲区存放着即将发送的数据或远程帧。

位流控制器是发送部分的核心，其中含发送状态机的定义，规定了总线仲裁、错误重发机制、并根据桢的格式发送相应的位。

位时序逻辑控制发送每一位的时序。

2.2.4对CAN发送器每个模块的具体说明

2.2.4.1对外部的接口（含接收部分）

表2.1 CAN控制器的对外接口表

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	模式寄存器
1	状态寄存器	命令寄存器	状态寄存器	命令寄存器
2	Bus timing 0		Bus timing 0	Bus timing 0
3	Bus timing 1		Bus timing 1	Bus timing 1
4	错误代码捕捉寄存器ECC		错误代码捕捉寄存器ECC	
5	接收错误计数器		接收错误计数器	接收错误计数器
6	发送错误计数器		发送错误计数器	发送错误计数器
7	（保留）	（保留）	（保留）	（保留）
8	RX ID[10:3]	TX ID[10:3]	ACR0	ACR0
9	RX ID[2:0] RTR DLC	TX ID[2:0] RTR DLC	ACR1	ACR1
10	RX DATA1	TX DATA1	AMR0	AMR0
11	RX DATA2	TX DATA2	AMR1	AMR1
12	RX DATA3	TX DATA3		
13	RX DATA4	TX DATA4		
14	RX DATA5	TX DATA5		
15	RX DATA6	TX DATA6		
16	RX DATA7	TX DATA7		
17	RX DATA8	TX DATA8		

表2.2 CAN寄存器功能表的说明

名称	位	功能	说明
模式寄存器mode	Mode.0	标识当前是否处于复位模式	1．处于 0．不处于

状态寄存器 status	Status.5	标识当前CAN结点是否参与总线发送（若错误太多，CAN结点会自动和总线脱离，不再进行发送操作）	1．不参与总线发送 0．参与总线发送
	Status.4	标识是否处于发送状态	1．处于 0．不处于
	Status.3	标识一次发送行为是否完成	1．完成 0．未完成
	Status.2	标识发送缓冲区是否有效	1．有效，可写 2．无效，不可写
	Status.1	标识该帧数据是否溢出	1. 溢出 0. 不溢出
	Status.0	标识接收缓冲区是否有效	1:有效,可读 0:无效,不可读
命令寄存器	Command.2	中止发送	1．发出中止命令 0．无操作
	Command.1	请求发送	1．发出发送命令 0．无操作
	Command.0	释放缓冲区	1. 发出释放命令 0. 无操作
总线时序寄存器0	BIT0.7	SJW.2	SJW的第二位
	BIT0.7	SJW.1	SJW的第一位
	BIT0.6	SJW.0	SJW的第零位
	BIT0.5	BRP.5	波特率的第五位
	BIT0.4	BRP.4	波特率的第四位
	BIT0.3	BRP.3	波特率的第三位
	BIT0.2	BRP.2	波特率的第二位
	BIT0.1	BRP.1	波特率的第一位
	BIT0.0	BRP.0	波特率的第零位
	BIT1.6	SEG2.2	延时段2第二位
	BIT1.5	SEG2.1	延时段2第一位
	BIT1.4	SEG2.0	延时段2第零位

	BIT1.3	SEG1.3	延时段1第三位
	BIT1.2	SEG1.2	延时段1第二位
	BIT1.1	SEG1.1	延时段1第一位
	BIT1.0	SEG1.0	延时段1第零位
错误代码捕捉寄存器ECC	ECC.5	标识是发送方的错误还是接收方的错误	0：发送错 1：接收错
	ECC.7 ECC.6 ECC.5	标识检测到的错误种类	001:位错误 010:格式错误r 011:填充错误 100:ACK错误 101:CRC错误
接收错误计数器【8：0】REC		接收方错误计数	9位计数器0-511
发送错误计数器【8：0】TEC		发送方错误计数	9位计数器0-511
ACR0		过滤ID[10:3]	验收过滤码0
ACR1		过滤ID[2:0]	验收过滤码1
AMR0		ID[10:3]屏蔽码	验收屏蔽码0
AMR1		ID[2:0]屏蔽码	验收屏蔽码1
RX BUFFER ID[10:3]			接收帧头缓冲1
RX BUFFER ID[2:0],RTR,DLC			接收帧头缓冲2
RX BUFFER DATA1			接收缓冲数据1
RX BUFFER DATA2			接收缓冲数据2
RX BUFFER DATA3			接收缓冲数据3
RX BUFFER DATA4			接收缓冲数据4
RX BUFFER DATA5			接收缓冲数据5
RX BUFFER DATA6			接收缓冲数据6
RX BUFFER DATA7			接收缓冲数据7
RX BUFFER DATA8			接收缓冲数据8
TX BUFFER ID[10:3]			发送帧头缓冲1
TX BUFFER ID[2:0] RTR DLC			发送帧头缓冲2
TX BUFFER DATA1			接收缓冲数据1
TX BUFFER DATA2			接收缓冲数据2
TX BUFFER DATA3			接收缓冲数据3
TX BUFFER DATA4			接收缓冲数据4
TX BUFFER DATA5			接收缓冲数据5

TX BUFFER DATA6			接收缓冲数据6
TX BUFFER DATA7			接收缓冲数据7
TX BUFFER DATA8			接收缓冲数据8

2.2.4.2 接口逻辑的框图

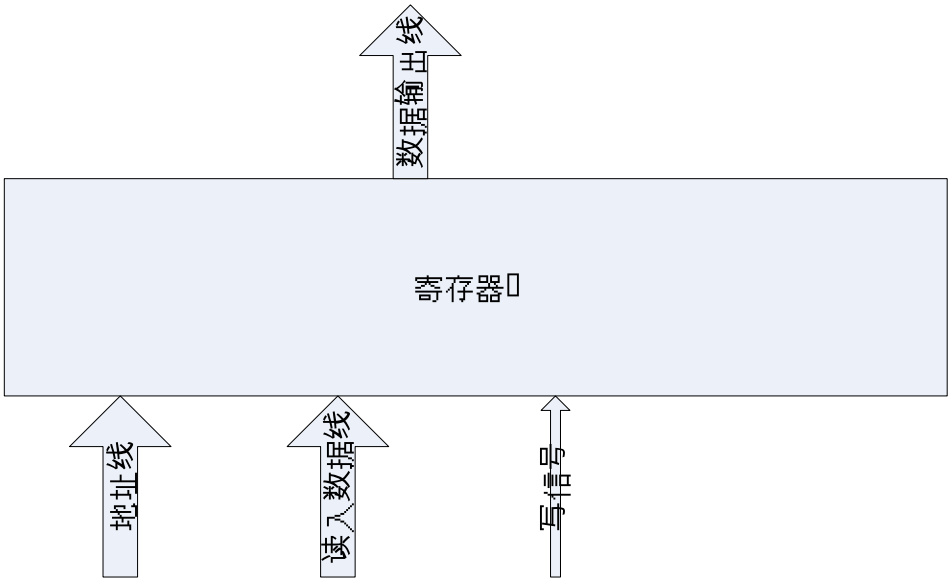


图2.4逻辑接口的框图

此处与Avalon总线相一致，由8位地址总线输入需要操作的寄存器地址，8位输入数据线输入数据，8位数据输出线输出相应的数据；寄存器默认是读数据，在（1位）写信号有效的情况下写入数据。

2.2.4.3位时序逻辑框图

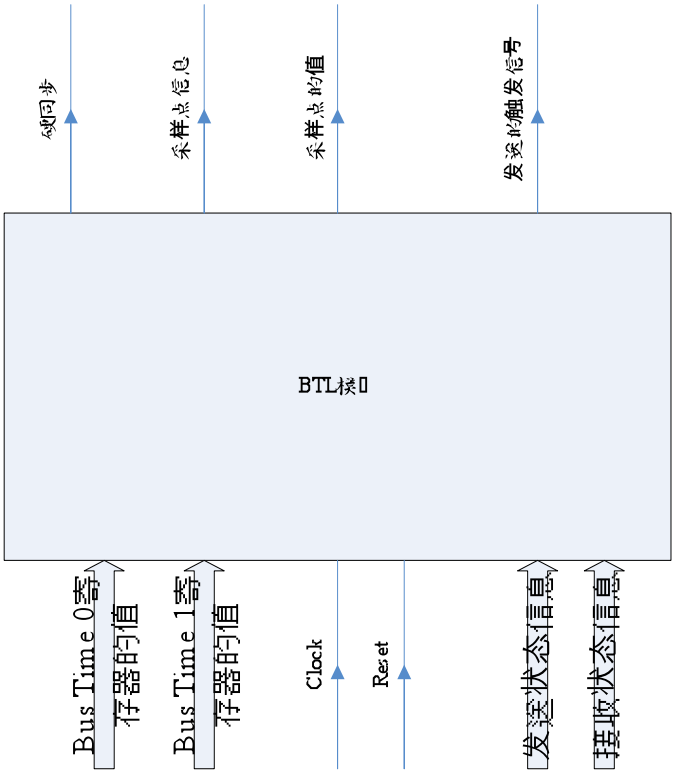


图2.5位时序逻辑框图

其中，sample_point为采样点sampled_bit为采样值hard_sync为硬同步信号　tx_point为发送一位的触发信号。这一模块根据CAN协议规定的时序逻辑以及同步方法来产生相应的时序信号，并根据同步规则进行相应的时序调整。

2.2.4.4 CRC检验码生成模块的框图

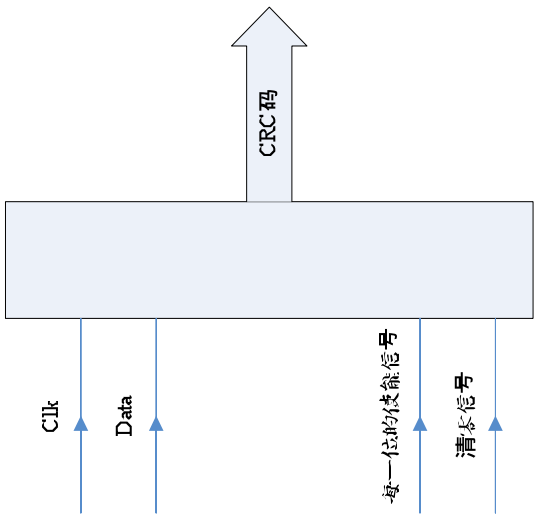


图2.6 CRC检验码模块框图

其中，data: 需要计算的每一位值enable: 计算CRC时输入每一位的使能信号initialize: CRC清零信号output:输出的CRC值，此处按照协议给出的生成多项式及其算法运算获得结果，从每帧的最开始处即进行CRC运算，直到数据发送的结束运算停止。

2.2.4.5 位填充模块的框图

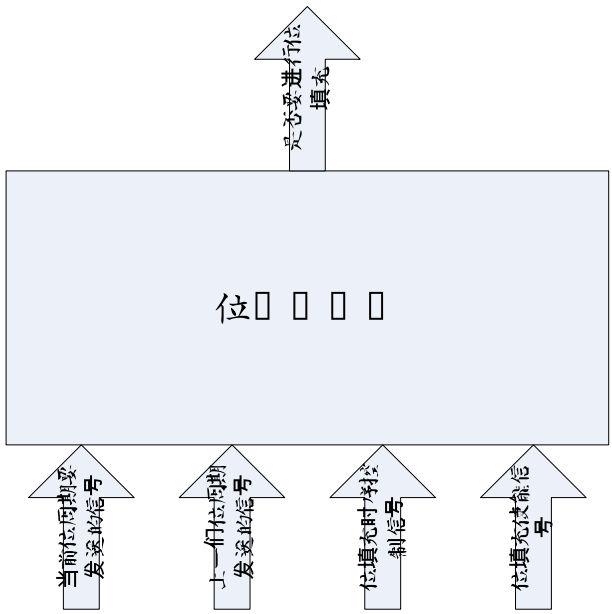


图2.7 位填充模块框图

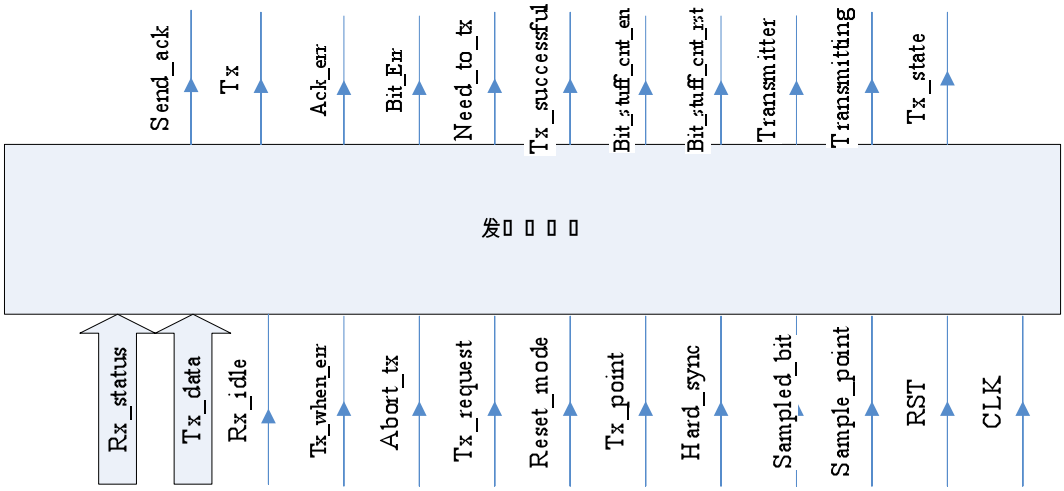
位填充模块根据CAN协议，实行每连续五个连续电平信号加入一个反向电平，是重同步机制的重要组成部分。这个模块的功能周期始于发送ID位，终止于发送完成ACK位。

2.2.4.6 发送状态机模块

这一模块是发送部分的主要工作模块，它要实现的功能是

(1) 根据BTL模块产生的位周期内的发送信号，输出“发送缓冲区”内的相应位，并在CRC段内发送CRC模块产生的以CRC校验码，最后发出相应的数据结束信号。
(2) 在总线空闲或默认情况下发隐性位。
(3) 在发送出现错误或监听到总线上有错误的时候，根据EML模块提供的值发送相应的错误信号。
(4) 在发送错误完成之后，重发需要进行发送的数据（或远程）帧。
(5) 进行总线仲裁，并在总线空闲时重发丢失仲裁的帧。
(6) 检测位错误与应答错误。
(7) 向位填充模块输出位填充使能信号和重置信号。
(8) 发送应答位，标志发送完成。

(9) 根据CAN协议对“错误被动节点”进行发送延迟。
(10)进行总线上的缓冲工作，即在不同帧之间插入间隔帧，防止总线过载。
(11)实现中止发送的功能。



输□ ：
(1)reg Tx_data□: 待□ □ □ □ □
(2)abort_tx: 中止待□ □ □ □ □
(3)reset_mode:软□ □ □
(4tx_point 发□ □ □ □ □ □ □ □ □
(5)hard_sync:硬同步信□
(6)sampled_bit:采□ □
(7)sample_point 采□ □ □
(8)Rx_state: 发□ □ □ □ □ □ □
(9)rx_idle: 总线□ □ □ □
(10) tx_when_err: 发□ □ □ □ □ □ □ □ □

输□ ：
(1)send_ack: 发□ ACK的□ □ □ □
(2)tx: 发□ □ □ □ □
(3)ack_err:检测□ □ ACK错误
(4)bit_err:检测□ □ □ □ □
(5) need_to_tx:准□ □ □ ，即□ □ □ □ □ □ □ □ □ □
(6)tx_successful: 发□ □ □
(7)bit_stuff_cnt_en:位□ □ □ □ □ □ □
(8)bit_stuff_cnt_rst: 位□ □ □ □ □ □ □
(9)transmitter: 是否是□ □ ′ RTR帧□ □ □ □
(10)transmitting 是否在□ □ (所有□ ，包括□ □ □)
(11)tx_state:发□ □ □

图2.8发送状态机模块框图

2.2.4.7 错误管理逻辑单元（EML）

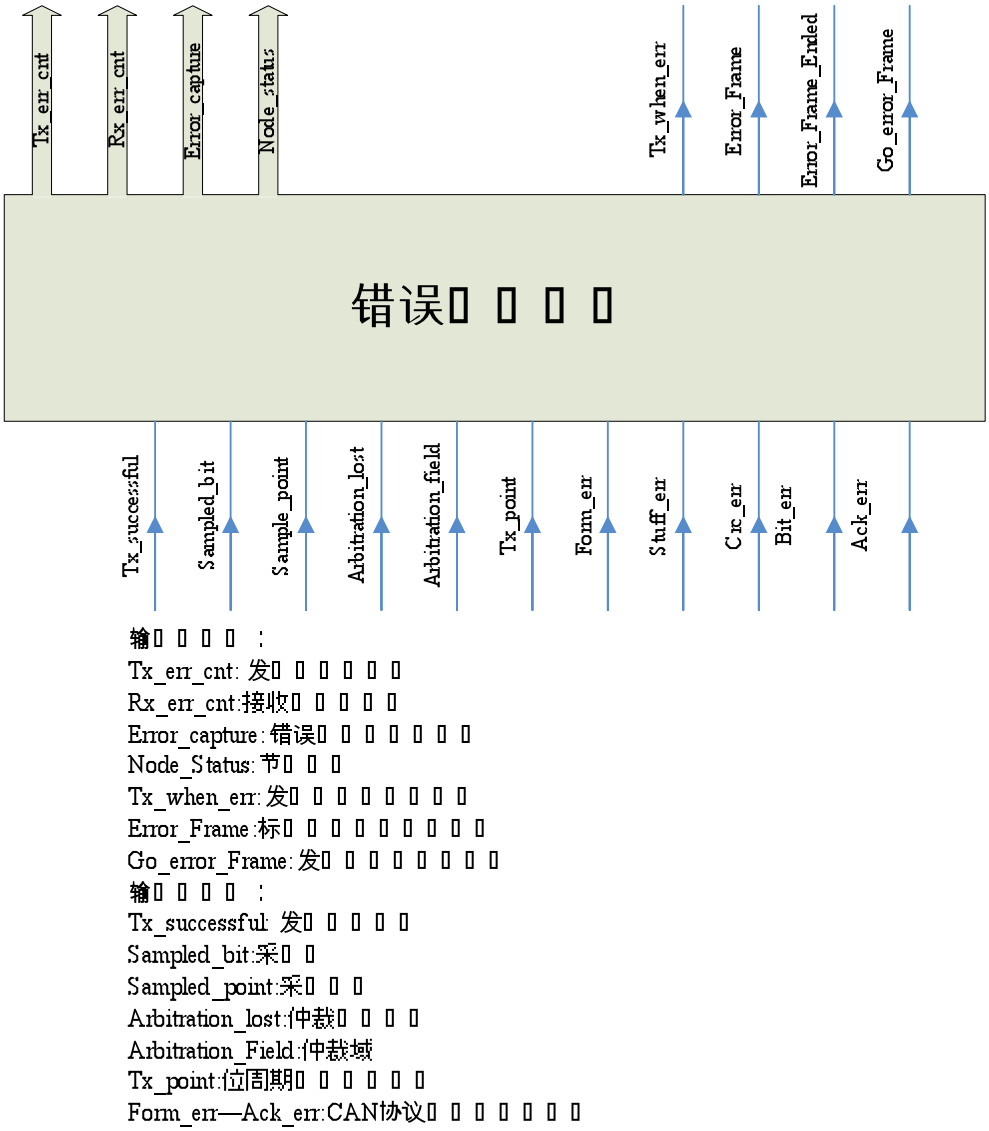


图2.9 错误管理模块框图

这一模块负责对发送、接收时产生的错误进行管理，其主要功能是：

（1） 收集五种错误信息，产生发送错误帧信号。

（2） 根据节点类型，发送相应的错误帧，即决定在发送错误帧时每一个发送位周期内所要发送的值

（3） 根据发送错误和接收错误两类不同的错误对发送错误计数器和接收错误计数器实行管理。

（4） 根据CAN协议对错误限制的规定，完成对节点类型的设定。

（5） 给出错误帧结束信号。

2.2.5发送流程图

在确立了上述的功能模块后，制定出针对本组CAN核的一套发送的流程图。

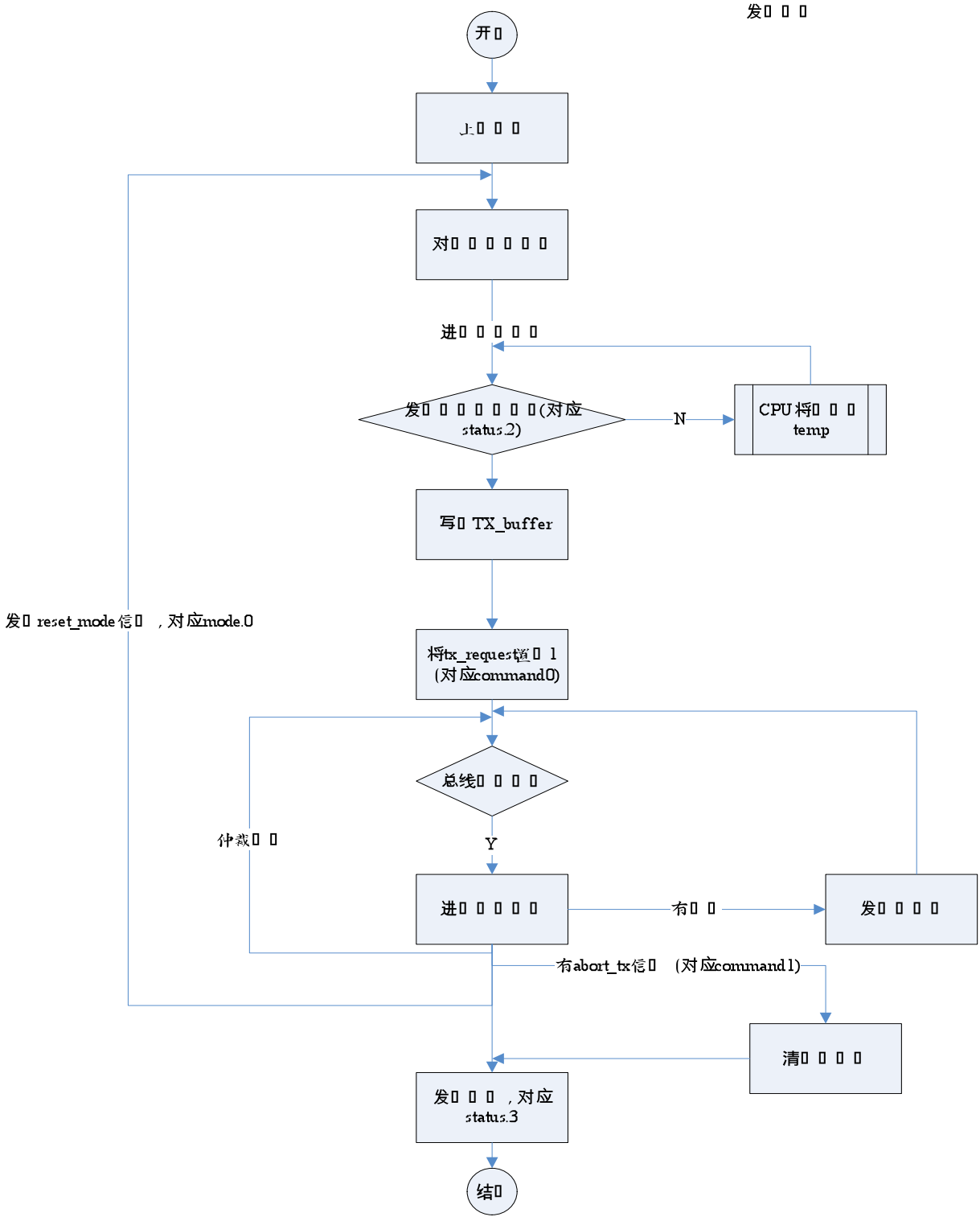


图2.10 发送操作流程图

3．详细设计及分析

3.1 对Verilog 里面阻塞与非阻塞赋值的一些讨论

阻塞赋值类似于平时经常使用的编程语言，是过程型赋值语言。而非阻塞赋值是硬件描述语言中与平时所学的软件编程语言中区别较大的一种赋值语言。或者说，软件编程语言里是没有这种赋值形式的。非阻塞赋值是一种并行的赋值，即语句之间没有先后顺序之分，在同一个触发信号的作用下，上一句对一个变量的赋值，不会影响下一句对该变量的使用，这在时序逻辑中是非常有用处的。一个经典的理论认为，在组合逻辑中应使用阻塞赋值；而在时序逻辑中应使用非阻塞赋值。但根据笔者的经验，这在绝大多数情况下是成立的。但对一些特殊的情况，比如对一些时序要求不很严格的时序逻辑(比如起到存储作用的单元)使用非阻塞赋值也是可以的。正确理解硬件描述语言里的阻塞与非阻塞是整个学习过程的一个难点，许多有着丰富设计经验的工程师也往往不能对此理解十分清楚。笔者由于学习这门语言的时间不长，并不对其中所有细节掌握得都很好，但根据自己的经验，在实际过程中，需要至少认清以下这一点：

assign 之于 阻塞 与 <=之于非阻塞

assign语句是非常有用的语句，它可以产生一些对要求非常严格的控制信号或触发信号。它的使用方法经常是这样的“assign 变量x= (一个条件表达式)”，如果这个表达式里涉及的值是涉及到用非阻塞赋值方式赋的值y，那么，y在此刻对x呈现的是已经更改过的值。

而如果另外一个非阻塞赋值方式赋的值z也由y来决定，那么在同一个时钟的跳变沿到来时，y对z呈现的值则是还没有被更改过的值。

3.2位定时逻辑单元（BIT TIMING LOGIC）

3.2.1位时序的解释

CAN总线是是一种多主异步的串行传输总线。每一个单独的CAN节点都有一个时钟，其内部时序是由这个时钟提供的信号来控制的。首先，要确认如下几个概念：

A. 标称位速率

标称位速率为一理想的发送器在没有重新同步的情况下每秒发送的位数量。CAN协议规定最快传输速度为1M bit/s(最大传输距离为40米)。

B. 标称位时间

标称位时间 = 1 /标称位速率

C.可以把标称位时间划分成了几个不重叠时间的片段，它们是：

C.1同步段（SYNC_SEG）

位时间的同步段用于同步总线上不同的节点。所有的跳变沿在理想状况下会出现在这一段内。

C.2传播时间段（PROP_SEG）

传播段用于补偿网络内的物理延时时间。它是总线上输入比较器延时和输出驱动器延时总和的两倍。在工程实践中，常把这一段归并到相位缓冲段1内。

C.3相位缓冲段1（PHASE_SEG1）相位缓冲段2（PHASE_SEG2）

相位缓冲段用于补偿边沿阶段的错误。这两个段可以通过重新同步加长或缩短。采样点是读总线电平并解释各位的值的一个时间点。采集点位于相位缓冲段1（PHASE_SEG1）之后。而发送点是用来向总线发送一个电位值的时间点。发送点位于相位缓冲段2(PHASE_SEG1)之后。

下面是一个位周期的示意

图3.1符合严格定义位周期示意图

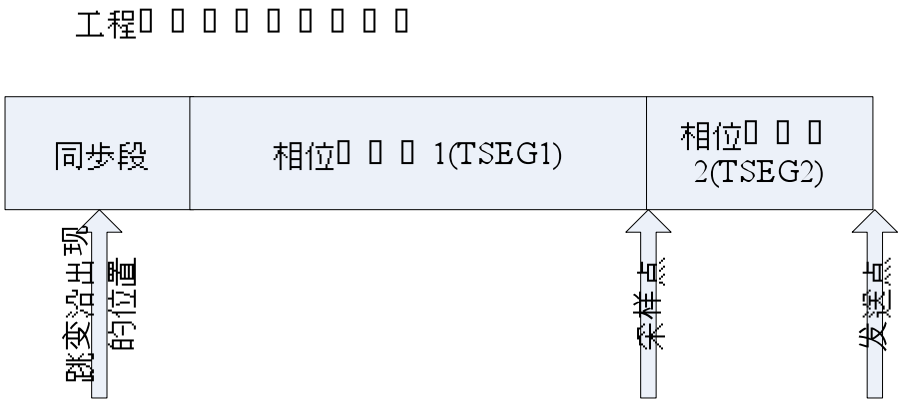


图3.2工程实践中采用的位周期示意图

D．信息处理时间（INFORMATION PROCESS TIME）

信息处理时间是一个以采样点作为起始的时间段。它的时间长度是采样点后用于计算后续位的位电平的时间长度；通常是一个基准脉冲的长度，即系统时钟的一个周期。

E．时间份额（TIME QUANTUM）

时间份额是派生自振荡器周期的固定时间单元。存在有一个可编程的预比例因子，其整体数值范围为1—32 的整数，以最小时间份额为起点，时间份额的长度为：时间份额（TIME QUANTUM）＝ m * 最小时间份额（MINIMUM TIME QUANTUM）（m 为预比

例因子）。这个时间份额也被称为CAN的时钟周期，它的本质是对最小时间份额(系统时钟)的分频器。之所以其被称为CAN的时钟周期，是因为同步段、相位缓冲段1、相位缓冲段2都是以这个周期为基准来设定的(设定方法见下)。设定m的方法见寄存器设定bus_timing_0寄存器的方法；在工程实践中，通常把时间份额设为2个最小时间份额长度，即，将bus_timing_0设为1000_0000。

F．时间段的长度（Length of Time Segments）

同步段（SYNC_SEG）为1 个时间份额； 传播段（PROP_SEG）的长度可设置为1，2，…8 个时间份额；缓冲段1 （PHASE_SEG1）的长度可设置为1，2，…，8 个时间份额；相位缓冲段2（PHASE_SEG2）的长度为阶段缓冲段1（ PHASE_SEG1）和信息处理时间（INFORMATIONPROCESSING TIME）之间的最大值；信息处理时间少于或等于2 个时间份额。一个位时间总的的时间份额值可以设置在8—25 的范围。设定TSEG1和TSEG2两段的方法见bus_timing_1寄存器的使用方法。在工程实践中，通常把同步段设为1个时间份额，把TESG1设为5个时间份额，把TSEG2设为4个时间份额；即一个位周期是10个时间份额(20个系统时钟)长度，即，将bus_timing_1设为0011_0100。在理想状况下(即不存在时钟差异的情况下)，这个模块起的作用仅仅是对基准脉冲的一个两次分频，同时在第二次分频中，确定出同步段、采样点和发送点的位置。

3.2.2 位时序的接口定义

```
module can_btl
(
  clk,//系统时钟
  rst,//系统复位
  rx,//总线接收值
  tx,//向总线发送值

  /* Bus Timing 0 register */
  baud_r_presc,//波特率
  sync_jump_width,//最大中转宽度

  /* Bus Timing 1 register */
  time_segment1,//TSEG1长度
  time_segment2,//TSEG2长度
  //triple_sampling,

  /* Output signals from this module */
  sample_point,//采样点
```

```
    sampled_bit,//采样值
    sampled_bit_q,//上一位周期内的采样值
    tx_point,//发送点
    hard_sync,//硬同步

    /* Output from can_bsp module */
    rx_idle,//总线空闲
    rx_inter,//进入帧间歇
    transmitting,//节点处于发送状态
    transmitter,//是发送节点
    go_rx_inter,//即将进入帧间歇
    tx_next,//在总线上将要打上的值

    go_error_frame,//发送错误帧的触发信号
    go_tx,//准备发送
    send_ack,//进入ACK状态
    node_error_passive,//错误被动型节点
    quant_cnt//对位周期内时间份额的计数
);
```

3.2.3 位时序的流程图表示

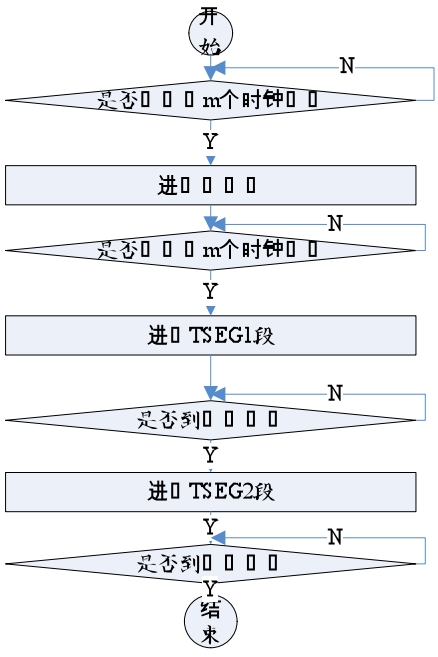


图3.3位时序算法流程图

需要解释的是：
(1)m个时钟周期代表一个时间份额
(2)检测是否到达采样点，即检测是否在进入TSEG1段后又来了规定长度1的时间份额，是否到达发送点是检测在进入TSEG2段后是否又来了规定长度2的时间份额。规定长度1和规定长度2即TSEG1的长度和TSEG2的长度。
(3)状态机在当前位周期结束后自动进入下一个位周期。

3.2.4 位时序的理想状态

下图是对btl在理想状态下的分析

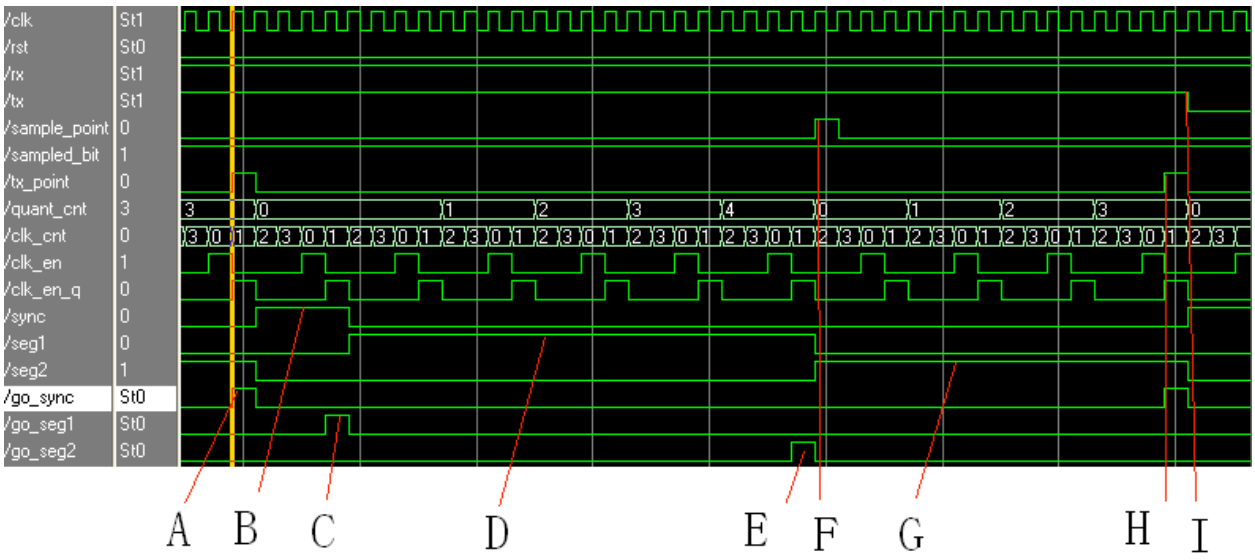


图3.4位时序理想状态波形图

需要解释的是：

变量说明：

Clk:系统时钟 Rst:系统复位信号Rx:总线上接收到的值 Tx:打到总线上的值 sample_point:采样 Sampled_bit: 从总线上采样的值 tx_point: 发送点 quant_cnt: 时间份额的个数 clk_cnt: 作为第一次分频的计数器 clk_en: 第一次分频的产生信号 clk_en_q: 对clk_en做一个系统周期长度的锁存 sync:表示节点处于同步段 seg1:表示节点处于TSEG1段，seg2:表示节点处于TSEG2段 go_sync: sync的触发信号 go_seg1: seg1的触发信号 go_seg2: seg2的触发信号。

为了表示清楚，在此使用m的值是4，即在第一次分频时，将系统周期4分频，可见，clk_en的信号是每隔4个系统时钟产生一次正跳变；使用的缓冲时间段仍是通常工程中采用的方法即1-5-4的分隔方法。所有与缓冲时间段的信号及采样点的信号都是使用clk_en_q为触发信号(这个信号是对第一次分频产生的电位信号的锁存)，而发送点的触发信号是使用clk_en为触发信号，以保证发送能够迅速完成，使下一位周期到来时，总线上已经为发送的值。

A. 此处是go_sync信号(由clk_en_q和上一个位周期的seg2段与时间份额计数器内的值3共同触发)，注意此时其仍在上一个位周期的TSEG2段。
B. 一个系统周期过后，此节点便进入同步段状态。
C. 此处是go_seg1信号(由clk_en_q 和sync段共同触发，因为同步段只有一个时间份额的长度)，此时节点仍处在SYNC段。
D. 一个系统周期过后，此节点便进入SEG1状态。

E. 此处是go_seg2信号(由clk_en_q和本位周期内seg1段与时间份额计数器内的值4共同触发)，此时节点仍处在SEG1状态。

F. 这一时间点上有两个事件发生：一是节点进入SEG2状态，二是在clk_en_q、quant_cnt值为4、节点处于SEG1这三个条件作用下采样点的出现。可见采样点是出现在SEG1段的最后以及SEG2段的最初。

G. 此时节点仍处于SEG2状态。

H. 这一时间点上也有两个事件发生：一是下一位周期内的go_sync信号产生；二是在clk_en、seg2、quant_cnt值为3这三个条件共同作用下tx_point信号的正跳变。注意此时节点仍处在本位周期的SEG2段，这并不像sample_point那样实际上是处在下一个缓冲段内，这样做的目的是保证在下一个系统时钟的上升沿来到时，可以把需要发送的值打到总线上，从而满足CAN协议对“发送要即时”的要求，也即在下个位周期的开始总线上就是已经打好的所需发送的值。

I. 此时处于下一个位周期内，同时节点也向总线上打出相应的值(上一位周期内是1，这一位周期内是0，这可以从波形里清楚地读出)。

这样的设计使一个位周期里的不同段紧密衔接在一起，并根据协议要求，发出采样信号与发送信号，以此为基础就可以实现同步传输的功能(即在同一时钟作用下，无时钟偏差)。

在此附上两个使用同一时钟的CAN节点，图中可清楚地表示出两者在同一时间点上都处在同一位周期的时间段上(假定其在同一时刻上电，若在不同时间上电，则其相位差保持不变)。

两个在理想状态下的CAN节点的时序图：

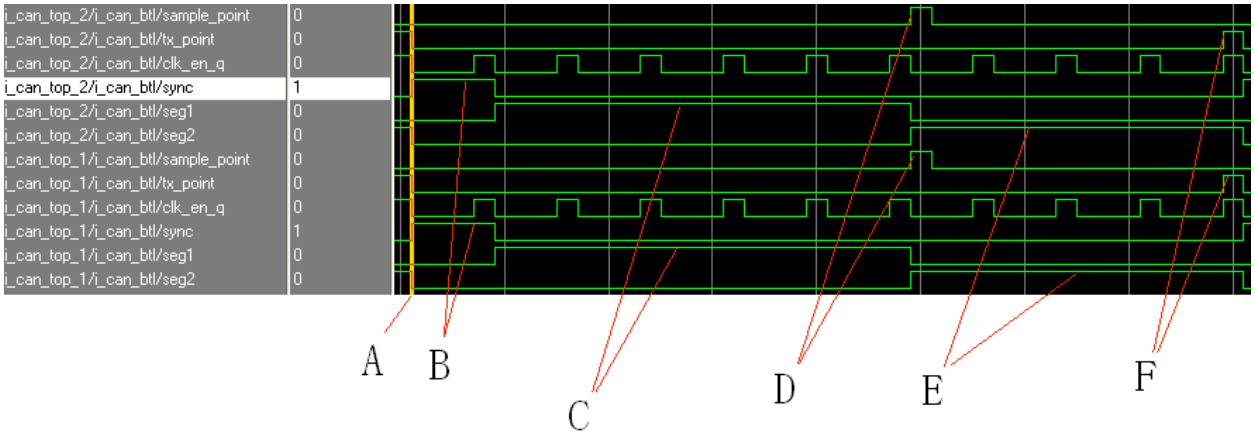


图3.5理想状态下两个CAN节点位时序相互间的影响

可见，A点是一个位周期的开始；B点是两者同在SYNC段；C点是两者同在SEG1段；D点是两者均处于采样状态；E点是两者同在SEG2段；F点是两者均处于发送点状态。

由上述的分析可知，在同时上电，时钟周期完全匹配的情况下，发送与接收的时序是容易控制的。但是，事实上并非总是这样的，如何使发送与接收两方互相知道彼此在做什

么，怎样使双方协调一致，具体到CAN节点上，就是如何使两者的位周期内部的三个时间段吻合(或是说大致吻合，因为同时上电是很难做到的，但实现两级分频的方法，可是使其状态相差保持在一个时间份额内，即不会影响发送点与接收点，因为这两者的时间距离通常在四个时间份额以上)。这便引出了下一个话题：

3.2.5同步

CAN协议是异步传输，但是其核思想仍然有同步传输的影子，这使同步这一概念显得更为重要。CAN中的同步是在下降沿时进行同步，这也是为什么发送方要进行位填充的原因(详见下面的位填充部分)。首先，要确定CAN协议里的同步种类以及其对位定时逻辑的影响及CAN协议的一个调整参数SJW。在CAN协议中，同步有两种，分别是**硬同步**(hard synchronization)和**重同步**(resynchronization)。CAN协议规定在对延长或缩短一个位周期时，调整值(若干个时间份额)有一个上限**SJW**(Segment Jump Width)，当位周期被调整的值小于此值时，位周期可以精确补偿，否则只能补偿这一上限值。

硬同步是指当发送方发出一帧时，首先要发送一个显性位(即帧起始符)，由于之前总线上一直处于空闲状态，即总线上一直是隐性位，突然出现一个隐性位，这标志着通信的开始，接收方捕捉到这一下降沿后，即进行相应的时序调整，这一过程就是硬同步。当检测到硬同步时，接收方在下一系统周期内自动进入TSEG1段。如果，接收方此时处于TSEG2段，那其发送点立即有效，因为接收方也可能有数据要发送，此时其发送数据帧则不再发送帧起始符，而直接开始发送ID场。硬同步在传输一帧时只能发生一次。

重同步是指接收方在接收时，每检测到一个下降沿，就要进行相应的调整，这是一个更为复杂的同步过程，重同步在一个位周期内只能发生一次。下面演示三种下降沿可能到来的位置。

A.标准位置

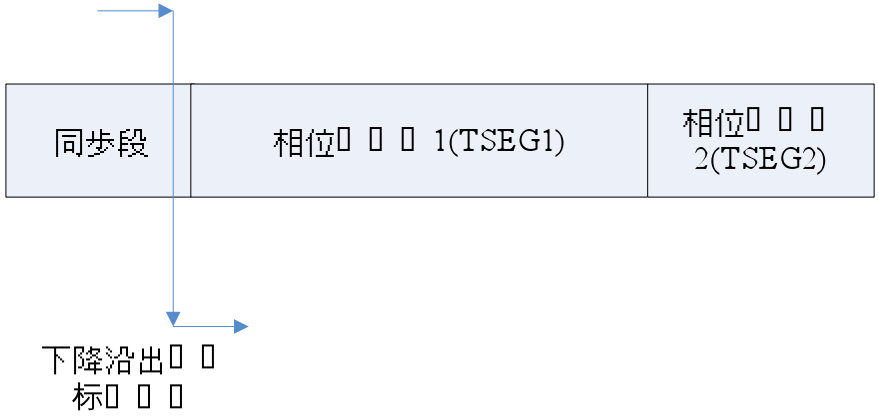


图3.6重同步时下降沿到来的标准位置

B.较晚到来

当下降沿出现在图示位置时，表示发收双方是同步的，不需要进行同步的处理。

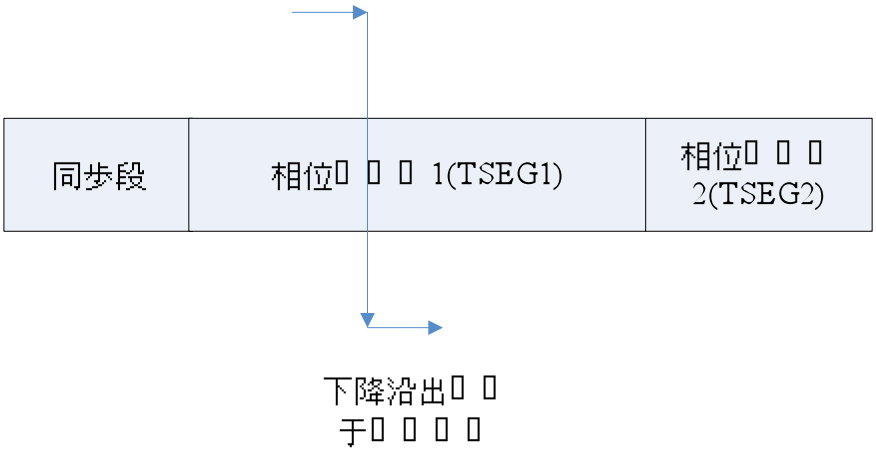


图3.7重同步时下降沿较晚到来的位置

当下降沿位于TSEG1段时，表明这个下降沿来得晚了，其与TSEG1段的起始位置的时间差e就是晚 来的那段时间。根据CAN协议，如果e>SJW，那么只能将TSEG1段延迟SJW长度，否则就将其延长e时间长度，即模拟下降沿来后TSEG1规定长度时间后进行采样，再之后TSEG2规定长度之后进行发送；从而保证后序的采样点与发送点和发送方保持一致。这时，发送点与采样点还是依照之前的规定发出。

C．较早到来

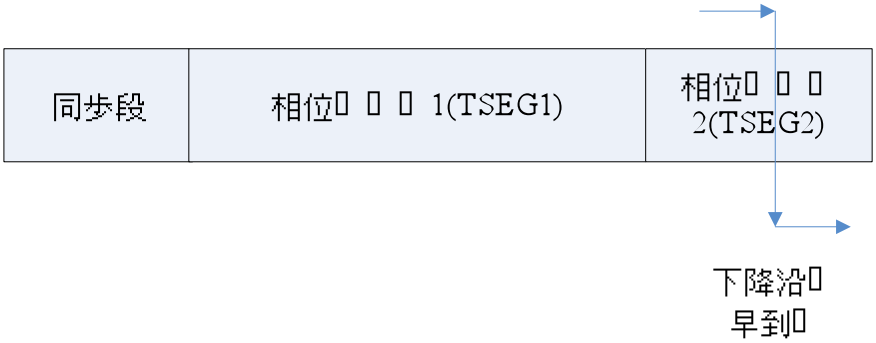


图3.8 重同步时下降沿较早到来的位置

当下降沿位于TSEG2段时，表明这个下降沿来得早了，其与TSEG2段的起始位置的时间差e就是早来的那段时间。如果e>SJW，则只能将TSEG2缩短SJW长度；否则，可以直接跳入下一个位周期，即缩短e个长度。由于要模拟出这个下降沿是在同步段的结束位置到来这一现象，所以此时直接跳入TSEG1段，并且立即发出tx_point信号。

在实现这一算法时，对延长TSEG1段和缩短TSEG2段，采用的是两种不同的方法。当延长TSEG1段时，使用一个delay计数器来记录TSEG1所要延长的时间，这个delay计数器里的值就是要延迟的时间，具体来说，如果quant_cnt(在TSEG1中所走的时间长度)小于SJW，则delay里的值为quant_cnt，否则其为SJW的值。当要缩短TSEG2段时，采用sync_window这样一个窗口，这个窗口表示早到的下降沿与TSEG2的结束点的差小于SJW，若小于SJW，则此窗口为正数值，也即表示这个下降沿落到了这个窗口内，可以精确补偿早到的这一段时间；否则，只有自行等待一段时间，直到下降沿落到这个窗口内，再做处理(跳入下一位

周期的TSEG1段)，但此时补偿的时间只是SJW的长度，而无法精确补偿。

至此，对同步算法已经全部讨论完毕，结合上述讨论的位定时逻辑，现在可以给出在位定时逻辑中几个最重要的信号的触发条件：

同步段的触发信号(go_sync):clk_en_q有效，在TSEG2段已走过规定时间tseg2并且没有发生任何同步事件。

进入TSEG1段的触发信号(go_seg1):clk_en_q有效，当前处于同步段或此时发生任一同步事件(需对同步做出调整)。

进入TSEG2段的触发信号(go_seg2):clk_en_q有效，当前处于SEG1段并且没有硬同步发生且已在TSEG1段走过tseg1长度时间。若发生重同步，则需再延迟delay计数器里的数值。

采样点触发信号(sample_point): 处在TSEG1段结束，与TSEG2段的触发信号相同。

发送点触发信号(tx_point): 处在TSEG2段结束，或者在TSEG段内发生任一同步事件。

硬同步信号(hard_sync): 总线空闲，总线上接收的值为0，而上一采样点的采样值为1且是这一桢内还没有其它硬同步信号。

重同步信号(resync): 在传输ID场到应答结束位中总线上接收到的值是0，而上一采样点的采样值为1且是这一位周期内还没有其它重同步信号。

需要说明的是，接收方与发送方的同步绝不是两者位周期内某一时刻的处处状态相同，同步段、TSEG1、TSEG2的引入，只是为了辅助给出采样点和发送点的正确输出。所要达到的目的只是一个：发送方发出的信号，接收方能够正确的接到。而事实上，由于物理条件的因素，甚至是编程本身对时序的使用上，发送方与接收方的位周期的每一时刻的状态是不会完全相同的。在CAN总线传输中，由于这是一种广播式的传输，发送方占主导地位，**接收方的时序要与发送方看齐**，这使得接收方在重同步之前的时序永远要落后发送方若干个节拍。因此，接收方需要进行重同步，发送方并不需要进行重同步，只有在发送隐性位而收到显示位的时候(即有另一个节点此时占据总线发送权时)才进行重同步。这样做的好处是，发送方不需要经常调整时序(通常是延长TSEG1段)，因而传送速率加快，从而导致接收方的接收速率也加快，可以提高系统的最大传输速度。但是其实现起来有一定难度，需要对时序有非常好的控制，而且分析起时序来也很困难，因为虽然是发送点与采样点不会产生冲突，但是发送方与接收方的位周期的状态往往相差一段距离。所以在以使用这种方法为主的同时，笔者也实验采用了另一种方法即发送方在检测到重同步时，也进行相应的重同步算法，这样虽然降低了传输速率，但使波形模拟更易看懂，而且使收、发两方的状态十分接近。

下面来演示传送中的(硬、重)同步现象：

3.2.5.1 硬同步

这是两个CAN节点都要进行发送，但每个节点的系统时钟不同，在此将其设为相差

3%，虽然这已超过了CAN协议的规定，但由于采用的一级分频是两分频，非常小，这使两个节点的误差降低，现在处于可以进行正确传输的环境。

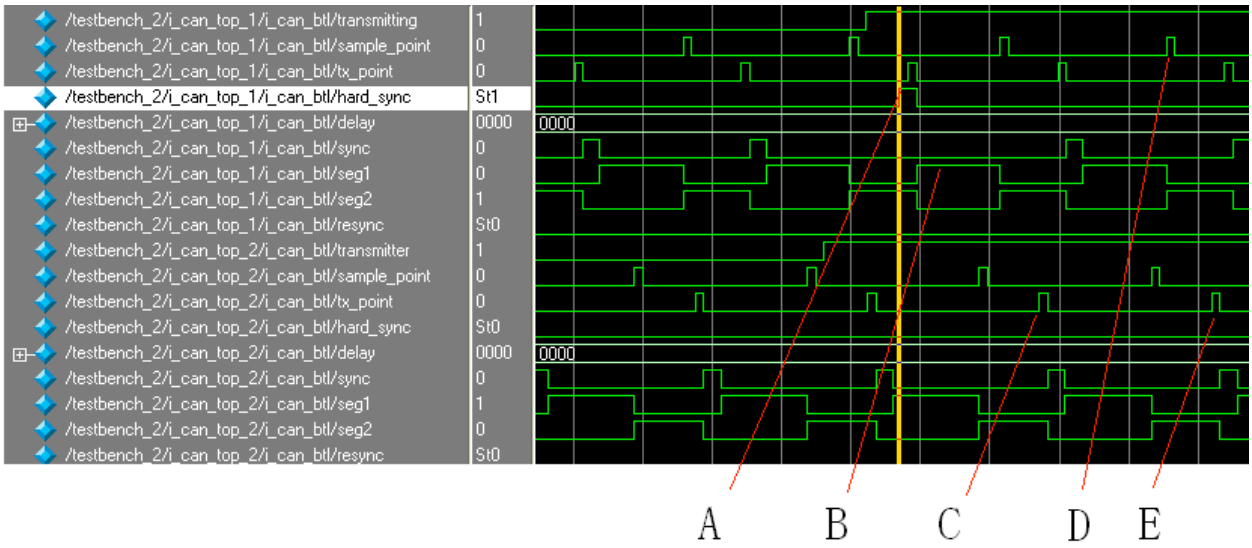


图3.9硬同步时的波形显示

由图可见，节点2首先检测到总线空闲，抢到对总线的控制权向总线上打上帧起始符。说明：

A．节点1检测到总线上的硬同步。并在clk_en的作用下，发送点有效。

B．节点1跳过同步段，直接进入TSEG1段。

C、D、E：分别是节点2的发送点、节点1的采样点、节点2的采样点，可见，它们都已被正确分开，即不会发生冲突。这是因为节点2还要进入同步段，而节点1可以直接进入TSEG1，从而变得非常接近节点2的状态。

3.2.5.2重同步

3.2.5.2.1 可以被精确补偿的重同步

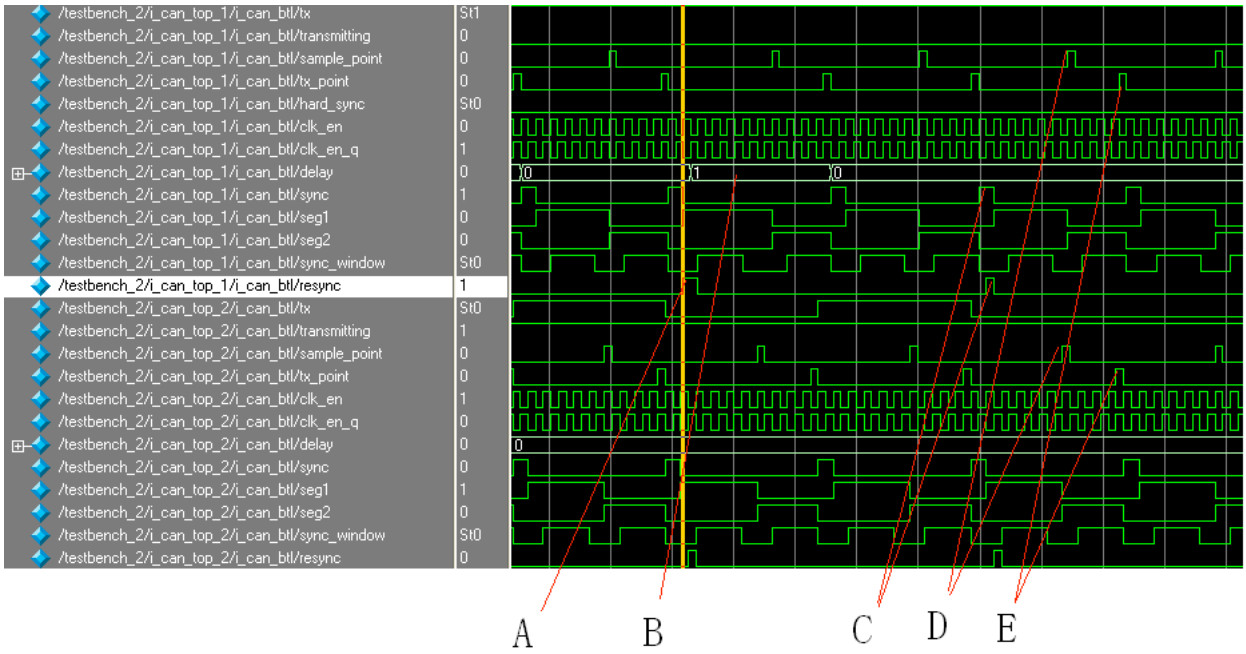


图3.10 可被精确补偿的重同步波形显示

此时，节点1是接收方，而节点2是发送方。

A．此时节点1检测到一个下降沿，并且检测到这个下降沿落在了TSEG1段，需要进行重同步。

B．经计算，延长TSEG1段1个单元，进行精确补偿。

C．此时节点1又一次检测到一个下降沿，由于上一次的重同步且为精确补偿，使这次这个下降沿直接落在同步段内，即不需要重同步。

D、E． 重同步后的发送点与采样点时序配合得很好。

3.2.5.2.2 不可被精确补偿的重同步

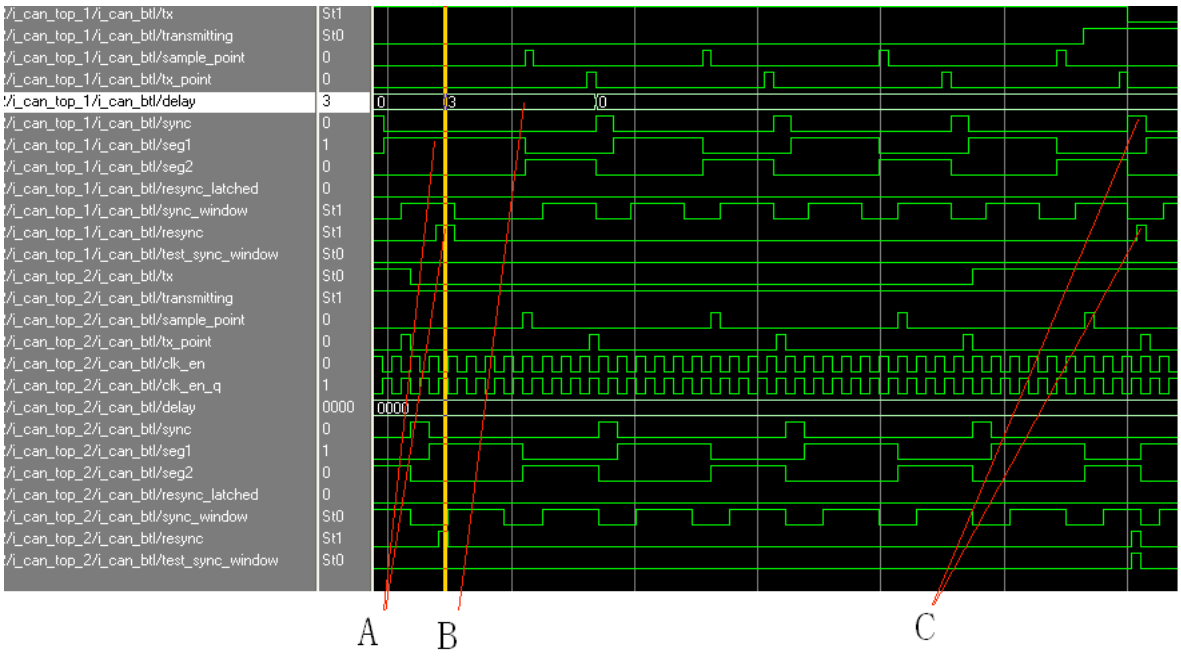


图3.11不可被精确补偿的重同步波形显示

这是在极限状态下的波形，仿真采用两个系统时钟偏差达5.5%的CAN节点来进行测试，而在实际情况中这是很少见的，即绝多数情况下延迟或早到是可以被精确补偿的。

A．接收方检测到下降沿，且在TSEG1段，但可看出，这个下降沿来得非常晚

B．经计算，接收方需要向后延迟超过SJW个时间份额，只能在此进行不精确补偿。

C．经过上一次的重同步，下降沿已经落到同步段内，无需重同步。

另：下降沿位于TSEG2处的重同步(有别于上面的下降沿处于TSEG1的重同步)

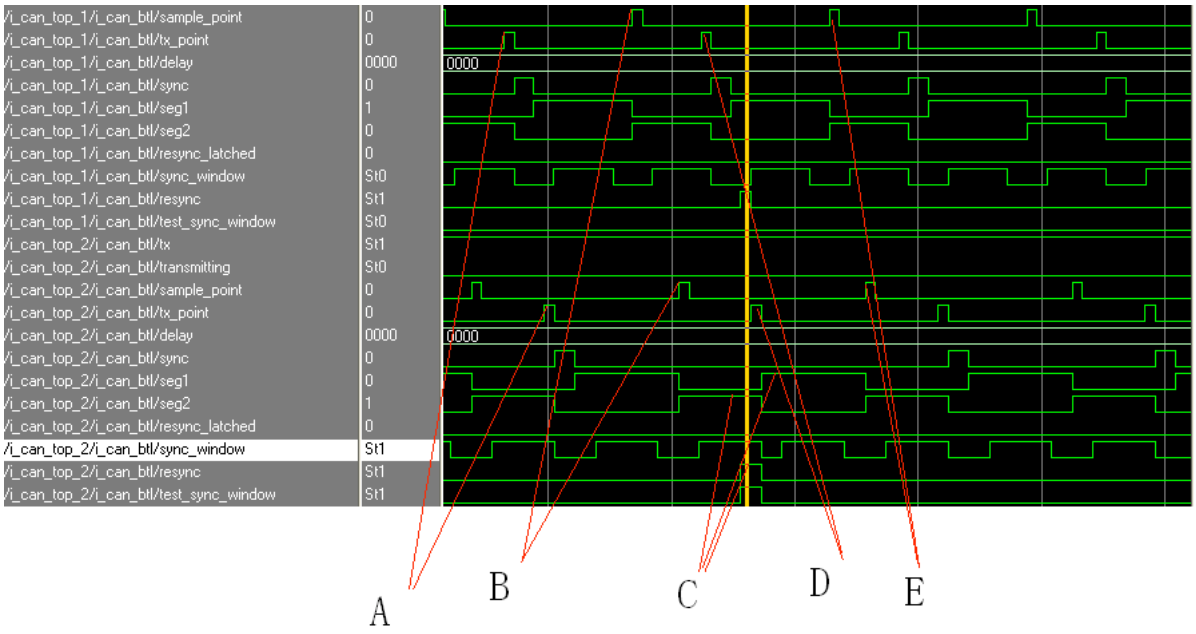


图3.12下降沿位于TSEG2处的重同步波形显示

A、B：两节点发送点与采样点已有所偏差。

C：下降沿落到TSEG2段，缩短TSEG2(因为这里只相差一个时间份额，所以缩短的功能并不能显现)，之后取消进入同步段，直接进入TSEG1段。

D、E：经过重同步，发送点、采样点的偏差降低。

3.2.5.2.3 发送丢失总线控制之后的重同步

当发送方丢失对总线的控制之后，如丢失仲裁或等待接收方发送ACK位时，需要向另一方同步。

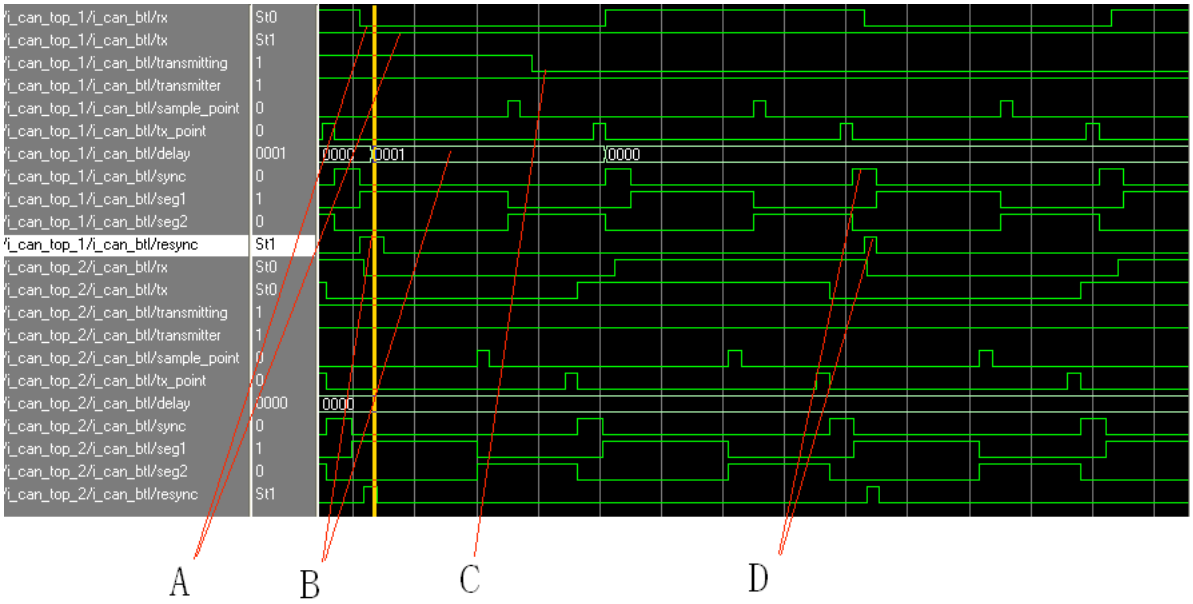


图3.13发送方丢失对总线控制时的重同步波形演示

A、发送的是1而接收到的是0，已经丢失对总线的控制

B、本节点向此时的发送方重同步，发现下降沿落在TSEG1段，对TSEG1延迟。

C、已经停止传送数据。

D、由于上一次的重同步，此时已经与发送方同步。

3.2.5.3另一种重同步的方法

当发送方也经常进行重同步时，可以使接收方与发送方在进行重同步之后可以达到时序状态非常的一致，对有偏差的节点进行非常见效的消除偏差。但是，它所带来的负作用是，若发送tx与接收值rx的延迟过长，超过一个时间份额，这就使发送方检测的下降沿永远不会落在同步段内，则经常需要重同步，而重同步里以延长TSEG1段为主，这使得传输

一帧的时间加长，即降低了传输速率，经大量实践显示，这种降幅在5%-10% 左右。其带来的另一个负作用是，在这种情况下进行的某些重同步不是必需的，而是由节点的物理性质决定的，尤其是它只能对下降沿都发生在收发两者TSEG1的情况进行精确地处理，对下降沿落在接收方的TSEG2段这种情况不能进行很好的处理，这使得其对发收双方的系统周期偏差的要求非常严格，并且多使用了逻辑计算资源。因此，在工程实践中这种方法是不被采用的，但是这种方法较容易实现。下图是对这一种同步方法的说明。

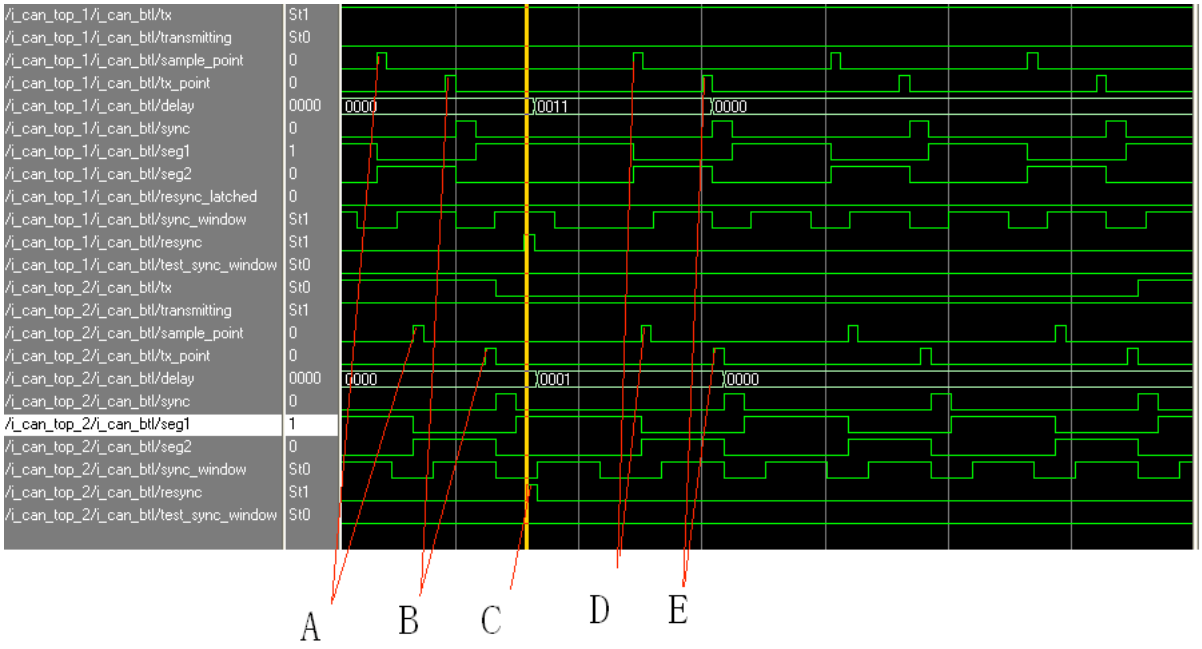


图3.14另一种重同步方法的波形显示

A、B：采样点与发送点的偏差已经非常明显。

C：下降沿同时落入发送方和接收方的TSEG1段，两者同时对TSEG1段延迟。

D、E：经重同步后，采样点与发送点的偏差减少到很小。

3.3位填充机制的具体实现

在讨论完同步机制后，来讨论发送方的位填充，因为这是实现同步的必须手段同。CAN协议规定在传送五个相同电平的信号后，发送方自动在后面加一个反向的信号。这样，如果是连续发送五个“1”,在第六位便自动补“0”，构成一个下降沿；若连续发送了五个“0”,则之后添加一个“1”,构成一个上升沿,在这之后最多五位之后根据位同步，又产生一个下降沿，从而起到同步效果。

3.3.1 位填充模块的接口定义

module can_tx_bit_stuff

```
(
bit_de_stuff_tx,//向外输出，标志是否要进行位填充
clk,//系统时钟
rst,//复位信号
reset_mode,//复位模式
bit_stuff_cnt_en,//位填充的始能信号
bit_de_stuff_reset,//位填充计数器的复位信号
tx_point_q,//发送点的锁存信号
tx,//当前要发送的值
tx_q//上一个位周期内发送的值
);
```

3.3.2 位填充算法的实现

这个算法的实现较为简单，但需要弄清位填充的始能信号何时有效，协议规定在发送数据帧或远和调用帧时，从帧开始到CRC结束位需要进行位填充。发送ACK位、帧结束、帧间隔以及错误帧时不进行位填充。

下面是位填充实现的状态转换图:

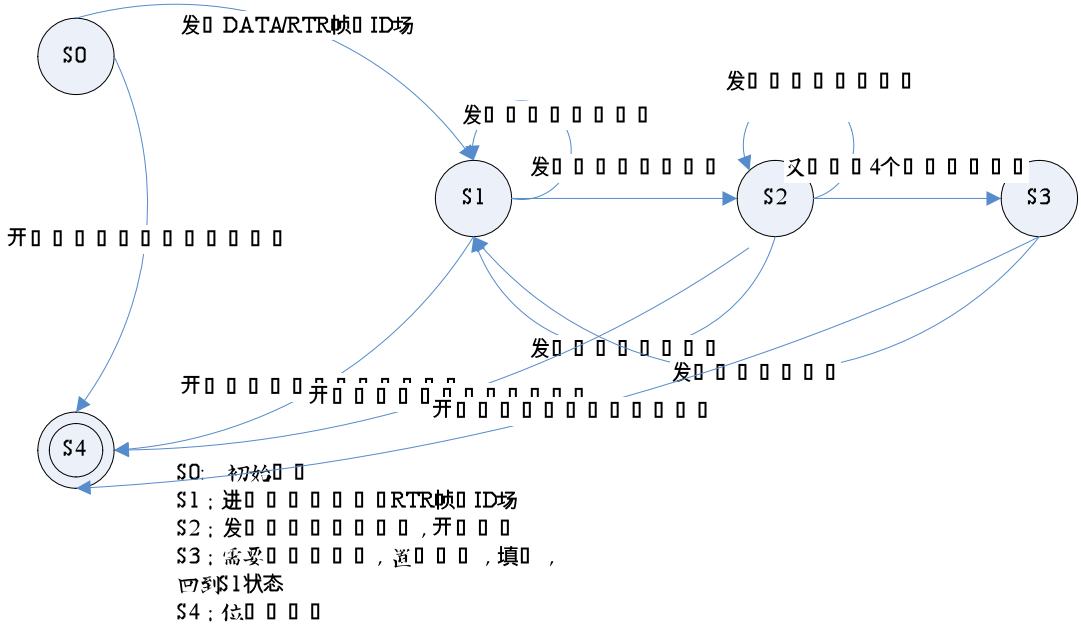
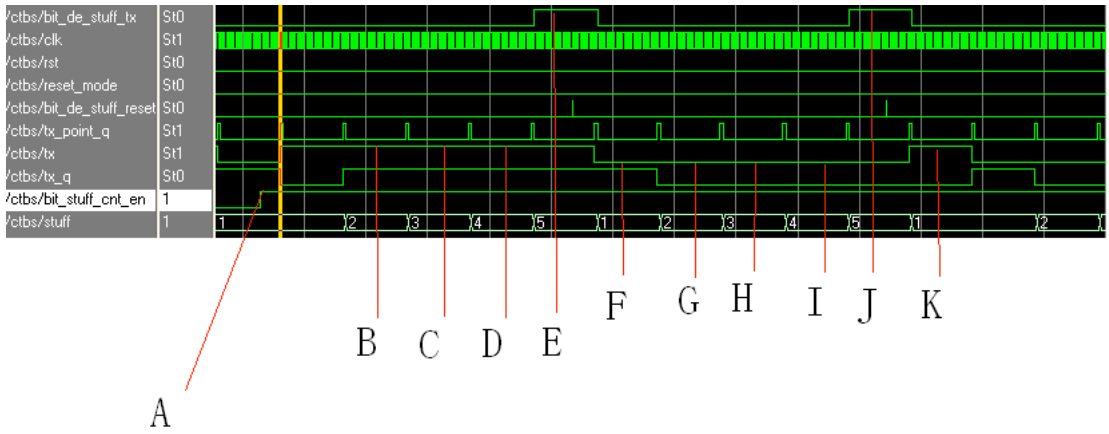


图3.15位填充的状态转换图

3.3.3位填充的效果演示



3.16位填充的效果演示波形图

A. 位填充使能信号有效
B. 发送第二个“1”
C. 发送第三个“1”
D. 发送第四个“1”
E. 发送第五个“1”，此时位填充有效，需要填充。
F. 填充了一个“0”。
G. 发送第二个“0”(上面填充的一个“0”也计入位填充序列)
H. 发送第三个“0”
I. 发送第四个“0”
J. 发送第五个“0”，此时位填充有效，需要填充。
K. 填充了一个“1”

3.4循环冗余校验码机制的具体实现

*循环码是一类重要的线性分组码。自1957年Prange开始研究以来，至今已得到了广泛的使用。它发展迅速的原因在于：编码和译码可用简单的具有反馈的移位寄存器来实现，并可用现代代数进行分析和构造。*显然，循环码对于CAN这类串行通信总线的纠错是非常合适的。CAN协议中给出了针对CAN总线的生成多项式$X^{15}+X^{14}+X^{10}+X^{8}+X^{7}+X^{4}+X^{3}+1$的算法。但在直接使用此算法之前，笔者想先简单地讨论一下这个算法电路的由来。

3.4.1 CRC算法的由来

在循环码的编码电路中，一定会有一个除法电路。所以首先来看一个除以g(x)的多项式除法电路。

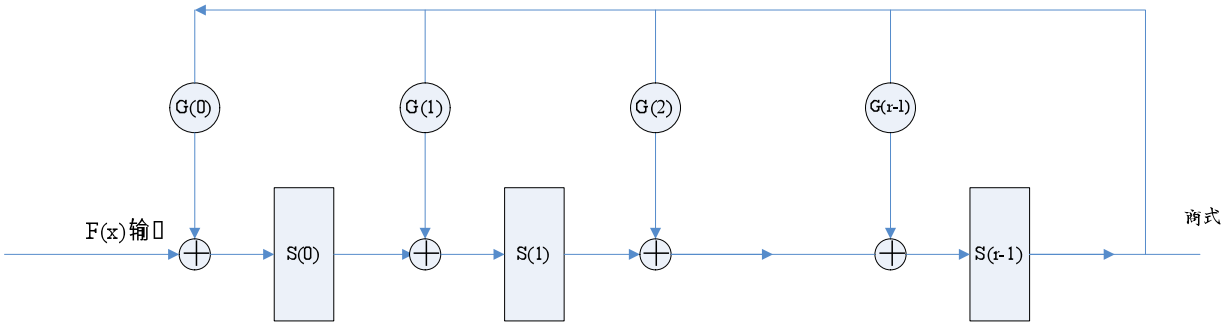


图3.17多项式除法电路

图中，有r级移位寄存器，分别用S0,S1,S(r-1)表示，r为多项式g(x)的最高次数，G(0)—G(r-1)代表g(x)的相应项系数。当G(i)等于1时，表明导线连通；当G(i)等于0时，表明导线断开。F(x)表示串行输入的每一位。由分析时序可知，当F(x)为0时，S(1)—S(r-1)进行右移一位的移位运算；当F(x)等于1时，S(0)—S(r-1)与G(0)—G(r-1)进行按位异或，所得结束置入S(0)—S(r-1)中。可证明，S(0)—S(r-1)中就是由低到高存放的计算所得的余数，也就是所求的循环冗余码。

但是，在实际编码中，需要提高F(x)的次数，即在F(x)的后面填入r个零(这和CAN协议中对CRC码字的生成方法是一致的，在CAN中是向F(x)后面加入15个零)。在除法电路上稍加改动，就可以生成对应这一功能的电路。

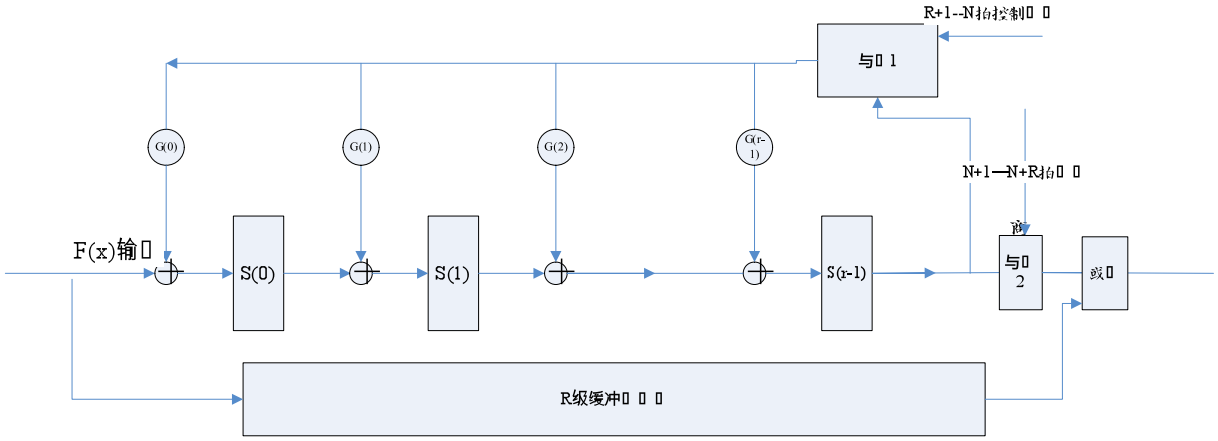


图3.18改进后的CRC校验编码电路

但是，这一电路的缺点在于在前R拍内，各级寄存器只处于移位状态，没有进行除法运算，时间开销较大。为克服这一缺点，可将F(x)直接加到最右一级寄存器输出端的异或门上，并将这个异或值作为与门1的反馈，这样就可以减少R个节拍的开销。最终的电路图如下：

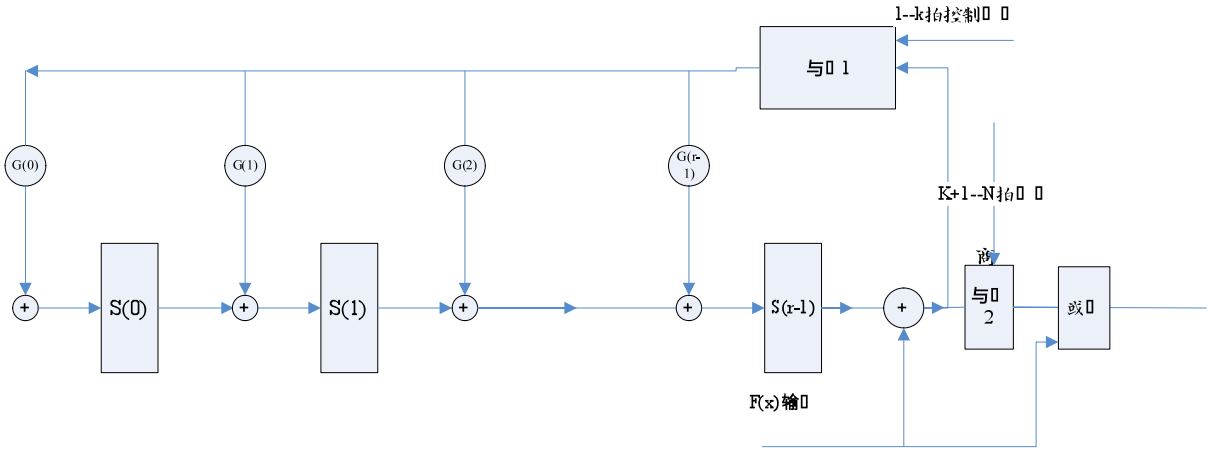


图3.19最终的CRC校验编码电路

这个电路图就与CAN协议给出的那个CRC产生算法非常吻合，在此给出其算法的伪码表示方法：

步骤1：r级寄存器左移一位，末位补零。

步骤2：当前输入位F(x)与r级寄存器最高位异或。

步骤2.1: 若步骤2的值为1，则r级寄存器与H4599异或(此数即是g(r-1),g(r-2)…g(0)的系数按权加和)

步骤2.2: 若步骤2的值为0，则r级寄存器值不变

步骤3: r级寄存器的值做为CRC码字输出

3.4.2 CRC模块的接口定义

```
module can_crc (
    clk, //系统时钟
    rst,//系统复位
    sample_point,//采样点
    nxtbit,//串行输入值
    bit_stuff,//是否是位填充位
    crc_enable,   //CRC模块的使能信号
    go_crc,       //CRC模块的退出信号
    crc_rg//CRC寄存器
);
```

CRC模块的使能信号对于总线上的第一个发送方来说是进入发送状态起有效，对于其它节点来说是它们检测到硬同步时有效；CRC模块的退出信号是当总线进入接收CRC码字状态时有效。

3.5 逆置数据模块的具体实现

因为笔者采用对发送缓冲区内的数据进行指针从低到高取值发送的策略，所以对从CPU发送来的数据要有一个逆置的过程。比如CPU要CAN发送1011_0110这个8位数据,首先要经过逆置模块将其逆置成0110_1101,再通过发送状态机模块将其从低到高发出，即发送序列为1_0_1_1_0_1_1_0,这也就是发送方打在总线上的数据，这和CPU所要求发送的数据完全吻合。

3.5.1逆置模块的接口定义

```
module can_ibo
(
   di,//8位输入数据
   do//8位输出数据
);
```

3.5.2 ibo算法实现

这个逆置的算法非常简单，就是将输入端的7-0位赋给输出端的0-7位。

3.6错误管理模块的具体设计

CAN总线提供了功能非常强大的纠错误处理，它可以检测到五种错误。这五种错误分别是位错误、填充错误、格式错误、应答错误和CRC错误。位错误是指发送方发送的值与接收方在总线上采样的值不同。在一些特定情况下，即使发送值与采样的值不同，也不产生位错误。填充充错误是指接收方在接收数据或远程帧时从总线上接收到六个相同电位的值。格式错误是指CAN协议规定作为某些场的结束位必须是隐性位，若在这些位上接收到显性位，则发生格式错。应答错误指接收方在应答位发送一个显性位来湮灭发送方此时发送的隐性位，发送方若此时采样到隐性位，则发生应答错误。CRC错误是指接收方接收到的CRC码与自己算得的CRC码不同。针对发送方的错误是两种：位错误和应答错误。针对接收方的错误是其余三种。使用了这五种检测错误的机制后，使得CAN总线的纠错能力非常强，也使CAN总线检测不到的错误非常少，误码率在一千亿分之一以下，这相当于以0.5Mbit/s的传输速率，25%的总线负载，每年运行2000小时，这样运行1000年，才会有一个没被检测到的错误。

当CAN节点检测到错误时，就会根据CAN协议的错误限制发送相应的错误帧。首先要明确CAN协议中节点错误的类型。CAN节点共三种节点错误类型：

错误主动型(error active) “错误主动”的单元可以正常地参与总线通讯并在错误被检测到时发出主动错误标志。

错误被动型 (error passive)“错误被动”的单元不允许发送主动错误标志。“错误被动”的单元参与总线通讯而且在错误被检测到时只发出被动错误标志。而且发送错误帧之后，“错误被动”单元将在预设下一个发送之前处于挂起状态。

总线关闭型(bus off)：“总线关闭”的单元不允许对总线上有任何影响。

这三种类型节点是通过CAN节点内部的**发送错误计数器**和**接收错误计数器**共同决定的。当发送错误计数器和接收错误计数器的值都在128之内时，节点为错误主动型；当两个计数器有一方在128之上且在255这下，节点为错误被动型；当发送错误计数器在255之上，节点为总线关闭型，这时只有在总线上监测到连续128个隐性位才能将错误计数器清零，节点进入错误主动型。

针对错误主动型，和错误被动型两类节点，CAN协议有两种错误帧：“错误主动帧”前6位错误标志位为显性位，后8位错误结束位为隐性位；“错误被动帧”前6位错误标志位为隐性位，后8位错误结束位也为隐性位。因为在发送错误帧时不进行位填充，所以其它节点**至少**会在错误标志位处发现位填充错误，重而停止发送或接收正常帧，转而发送错误帧，即当总线上发现1个错误时，其它节点都会在至多六个位周期内检测到错误并发送错误帧，由于总线现与逻辑的使用，这会使实际上总线上的错误标志位为6(所有节点同一时刻发现错误，并同时开始发送错误标志位)-12(某些节点直到第一个节点发送了6位显性位时才检测到位填充错误，之后开始发送6个显性位)。

EML模块的具体功能就是：(一)对发生的错误进行相应的错误计数，以确定节点的错误类型；(二)根据节点的错误类型，组织相应的错误帧；(三)根据发生的那五种错误类型，记录错误代码捕捉器，并记录这是发送方还是接收方的错误。

在近一步讨论具体算法之前，先来明确一下EML的接口定义。

3.6.1错误管理逻辑单元的接口定义

```
module can_eml
(
    //输出信号
go_error_frame,//发送错误帧的触发信号
    tx_when_err,//发送错误帧的相应位
    error_frame_ended,//错误帧发送完成
    error_frame,//处在发送错误帧的状态
    error_cnt1,//错误帧标志位的计数器
    error_cnt2,//错误帧结束位的计数器
    enable_error_cnt2,//错误帧结束位开始计数的使能信号
    error_capture_code,//错误代码捕捉寄存器
```

```
    tx_err_cnt,//发送错误计数器
    rx_err_cnt,//接收错误计数器
    node_error_passive,//错误被动型
    node_error_active,//错误主动型

    node_bus_off,//总线关闭
    node_bus_off_on,//节点在总线上
    set_reset_mode,//节点由总线关闭型恢复为可用时的信号

//输入信号：
clk,//系统时钟
    rst,//复位信号
    reset_mode,//软复位信号
    sample_point,//采样点
    sampled_bit,//采样值
    sampled_bit_q,//采样值的锁存
    bus_free,//连续检测到总线上11个1
    tx,//要送的值
    tx_successful,//发送成功
    arbitration_lost,//仲裁丢失
    arbitration_field,//仲裁域
    form_err,//格式错
    ack_err,//应答错
    bit_err,//位错
    crc_err,//CRC错
    stuff_err,//填充错
    go_rx_eof,//接收帧结束的触发信号
    go_rx_ack_lim,//接收ACK结束位的触发信号
    tx_point,//发送点
    transmitting,//正在发送
    transmitter//发送节点

);
```

3.6.2具体算法的实现

3.6.2.1 发送错误计数器与接收计数器的算法实现

这两个计数器的算法是由CAN协议中“故障界定“的规则给出的。

故障界定的规则如下：

1. 当接收器检测到一个错误，接收错误计数就加1。在发送主动错误标志或过载标志期间所检测到的错误为位错误时，接收错误计数器值不加1。

2. 当错误标志发送以后,接收器检测到的第一个位为“显性”时，接收错误计数值加8。

3. 当发送器发送一错误标志时，发送错误计数器值加8。

例外情况1：

发送器为“错误被动”，并检测到应答错误（注：此应答错误由检测不到一“显性”应答 以及当发送被动错误标志时检测不到一“显性”位而引起）。

例外情况2：

发送器因为填充错误而发送错误标志（注：此填充错误发生于仲裁期间。引起填充错误是由于：填充位〈填充位〉位于RTR 位之前，并已作为“隐性”发送，但是却被监视为“显性”）。

例外情况1 和例外情况2 时，发送错误计数器值不改变。

4. 发送主动错误标志或过载标志时，如果发送器检测到位错误，则发送错误计数器值加8。

5. 当发送主动错误标志或过载标志时，如果接受器检测到位错误（位错误），则接收错误计数器值加8。

6. 在发送主动错误标志、被动错误标志或过载标志以后，任何节点最多容许7个连续的“显性”位。

以下的情况，每一发送器将它们的发送错误计数值加8，及每一接收器的接收错误计数值加8：当检测到第14个连续的“显性”位后；在检测到第8个跟随着被动错误标志的连续的“显性”位以后；在每一附加的8个连续“显性”位顺序之后。

7. 报文成功传送后（得到应答及直到帧末尾结束没有错误），发送错误计数器值减1，除非已经是0。

8. 如果接收错误计数值介于1 和127 之间，在成功地接收到报文后（直到ACK 间隙接收没有错误，

及成功地发送了应答位），接收错误计数器值减1。如果接收错误计数器值是0，则它保持0，如果大于127，则它会设一值介于119 到127 之间。

9. 当发送错误计数器值等于或超过128 时，或当接收错误计数器值等于或超过128 时，节点为“错误被动”。让节点成为“错误被动”的错误条件致使节点发出主动错误标志。

10.当发送错误计数器值大于或等于256 时，节点为“总线关闭”。

11. 当发送错误计数器值和接收错误计数器值都小于或等于127 时，“错误被动”的节点重新变为“错误主动”。

12. 在总线监视到128 次出现11个连续“隐性”位之后，“总线关闭”的节点可以变成“错误主动”（不再是“总线关闭”），它的错误计数值也被设置为0。

明确了这些规则后，可以给出相应的流程图

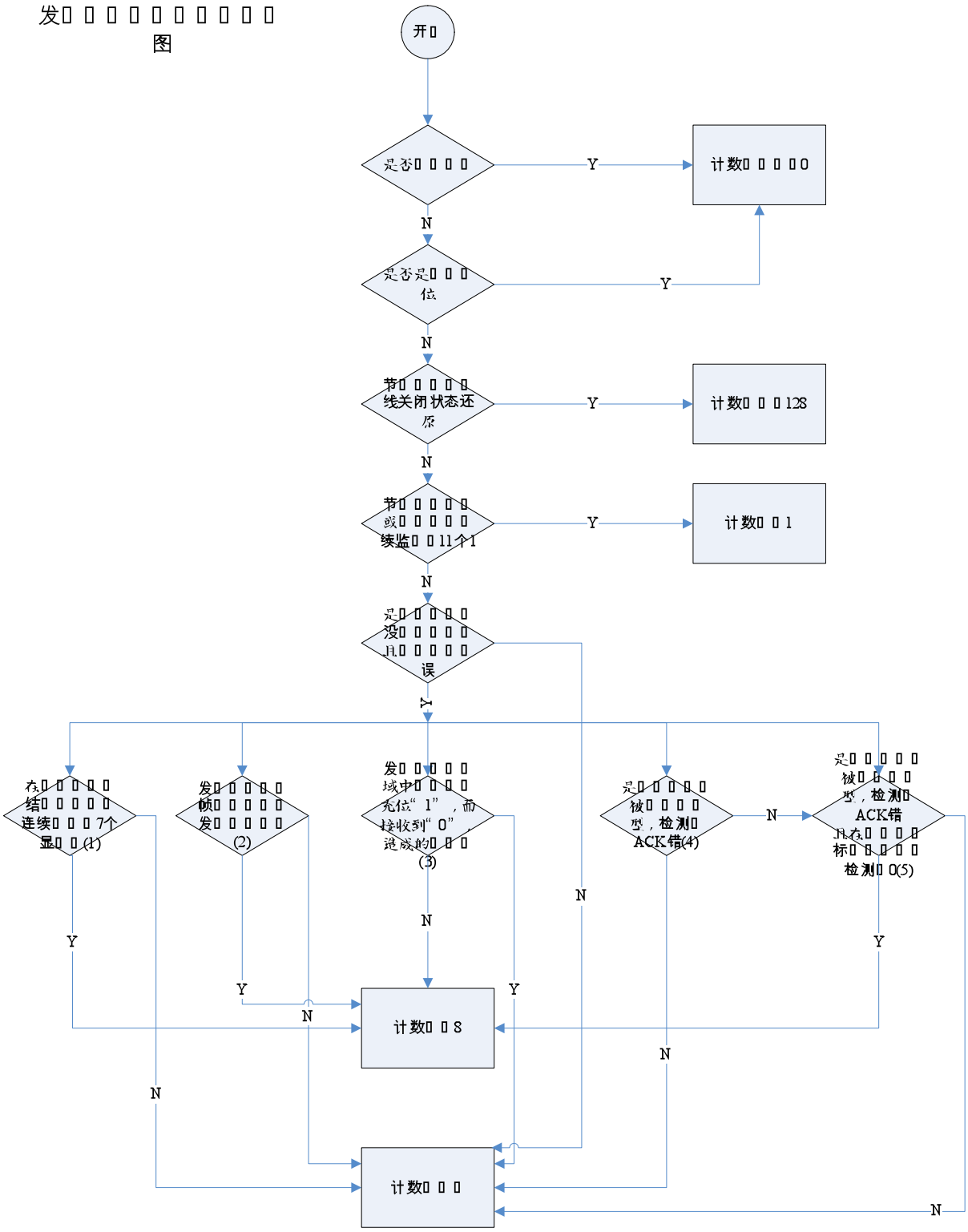


图3.20发送错误计数器的算法流程图

注： (1)不是发送状态机或接收状态机能够检测到的错误，只有在EML模块里可以发现
(2)对应着发送错误帧时可能发生的位错误
(3)对应着“故障界定”中的规则5
(4)(5)对应着“故障界定”中的规则3

接收错误计数器的流程图 见下页图3.21

注：
(1) 正确接收到一帧是指接收到一帧的应答位的结束符。
(2) 与发送错误计数器中的注(2)相对应
(3) 对应着规则二
(4) 对应着规则五
(5) 对应着规则六

综上，便是发送错误计数器和接数错误计数器的计数方法，在设计时笔者使用的相应触发信号都是只在一个系统时钟内有效，以保证每一个错误只被记录一次。在设计中，需要辅助使用四个计数器：(1)错误主动节点发送错误标志时的计数器对错误被动节点发送(2)错误被动节点发送错误标志时的计数器(因为CAN协议规定被动节点在发送错误标志时，需要检测到六个相同电位，才可以进行发送错误结束位的状态) (3)发送错误结束位时的计数器(主动节点和被动节点可以共用这一计数器)(4)主动节点发送完错误标志后可以允许的接收到的显性位的计数器(主要用于判断发错误帧时可能发生的位错误)

3.6.2.2节点错误类型的具体算法以及错误代码捕捉寄存器的实现

这是两个非常简单的算法。当发送错误计数器的值大于或等256时，节点类型为总线关闭型；当发送错误计数器或接收错误计数器大于或等于128且不是总线关闭型时，节点为错误被动型；除上述两种情况外，总线为错误主动型。

因为在一个时刻内，只能有一个错误发生，所以可以把这五种错误映射到错误代码捕捉寄存器的前三位上，它们分别是：位错误—001，格式错—010，填充错—011，应答错—100，CRC错—101；这个寄器的第四位记录是发送方还是接收方。需要注意的是，这种设计对CPU提出一定的要求。因为CAN节点在任一时刻内只能有一个错误发生，但可以有几个错误连续的发生，比如CRC错和ACK错经常结伴产生，这就需要CPU要时时对错误代码捕捉器进行询查，一旦发生错误后，要立刻记录下来。

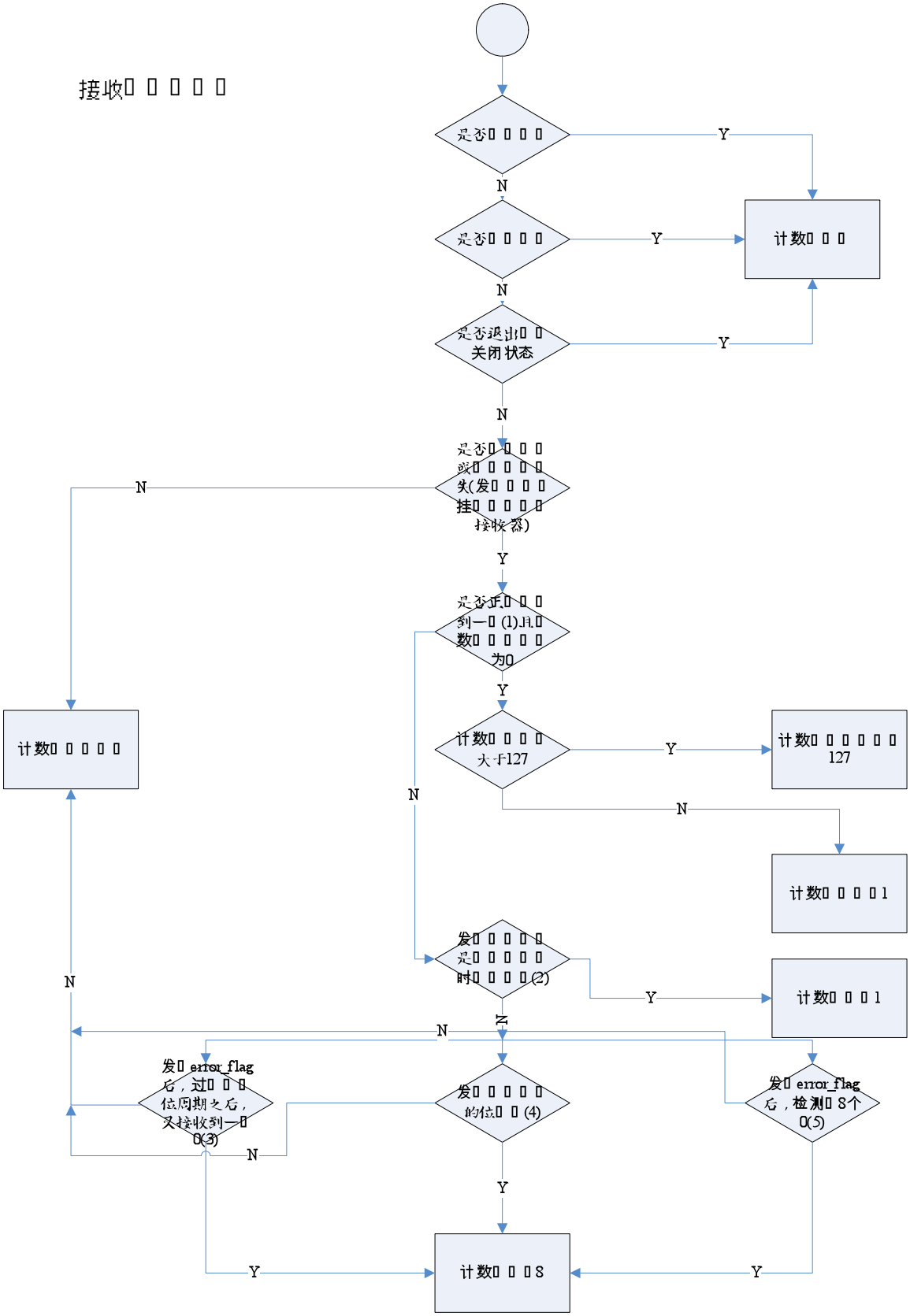
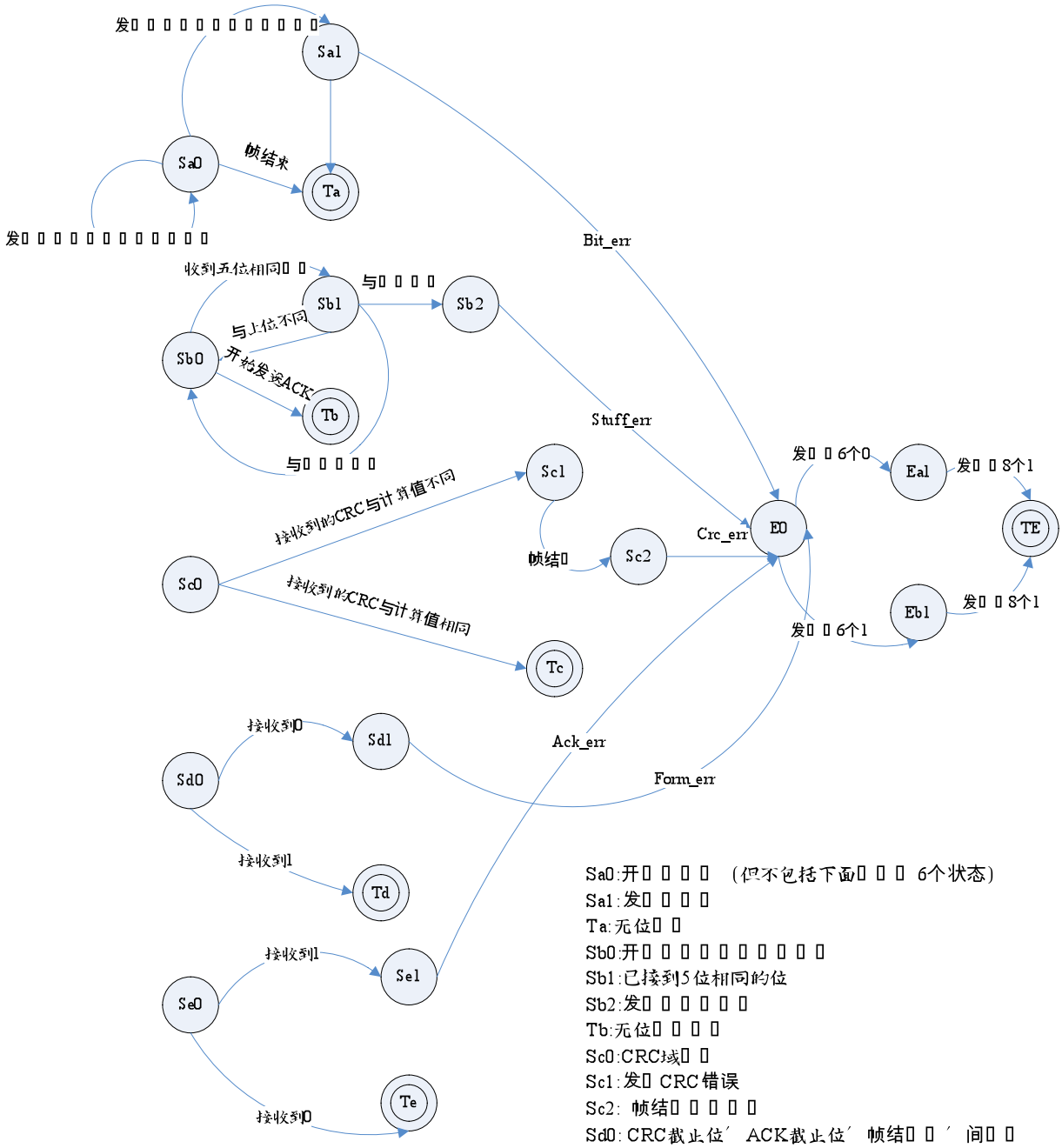


图3.21 接收错误计数器的算法流程图

3.6.2.3 组织发送错误帧的状态转换



Sa0:开 （但不包括下面 6个状态）
Sa1:发
Ta:无位
Sb0:开
Sb1:已接到5位相同的位
Sb2:发
Tb:无位
Sc0:CRC域
Sc1:发 CRC错误
Sc2: 帧结
Sd0: CRC截止位′ ACK截止位′ 帧结 ′ 间
Sd1: 发
Td: 无格式
Se0:ACK应
Se1: 发
Te: 无
E0 ：错误处
Ea1: error_active节 error_flag
Eb1: error_passive节 error_flag
TE:错误帧发

图3.22错误帧组织的状态转换图

3.6.2.4 三种错误节点类型发送的错误帧

3.6.2.4.1 错误主动型

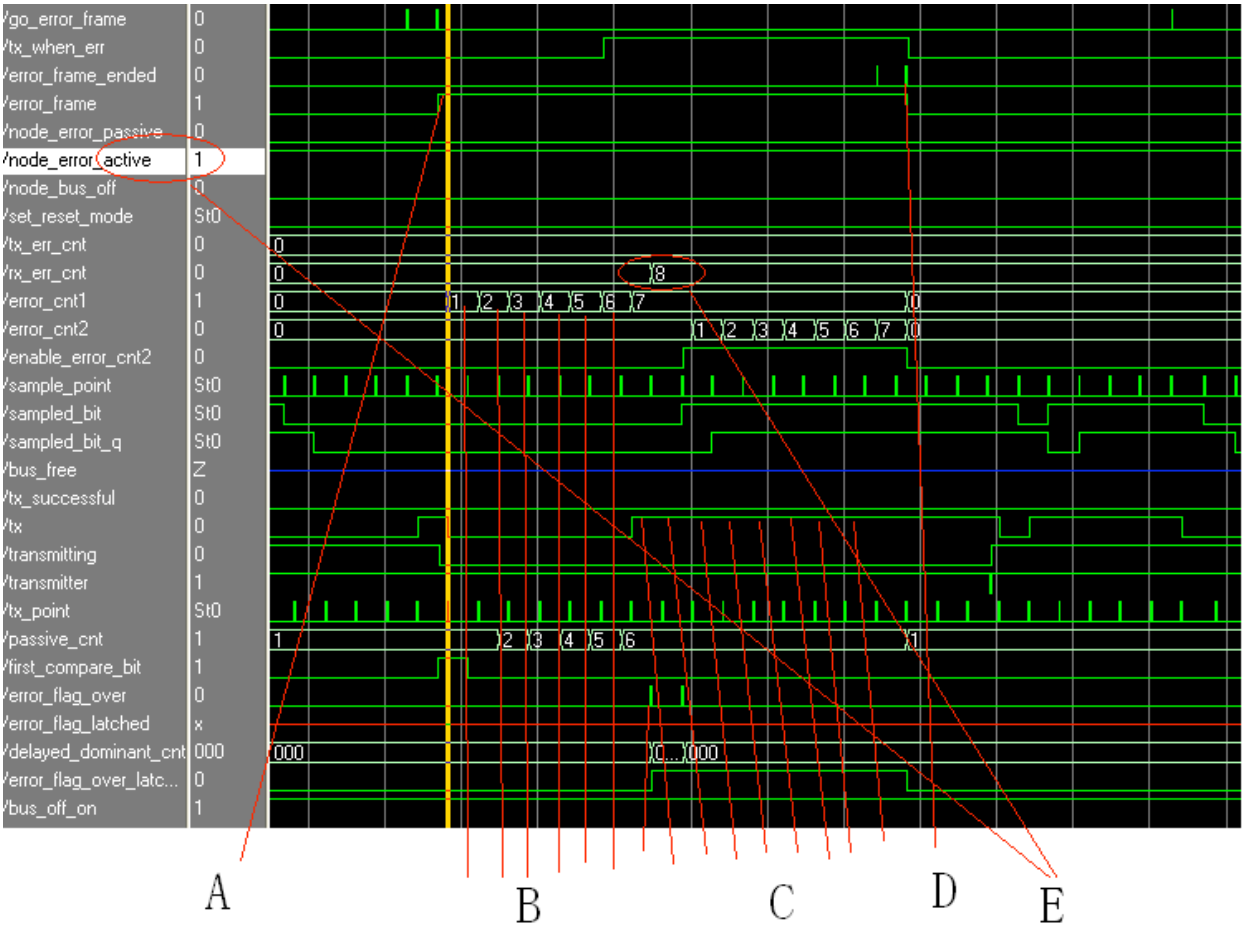


图3.23 主动错误节点的错误帧波形演示

注：

A、 产生错误

B、 发送6位显性位

C、 发送8位隐性位

D、 错误帧结束

E、 接收错误计数器为8，发送错误寄存器为0，是错误主动型

3.6.2.4.2错误被动型

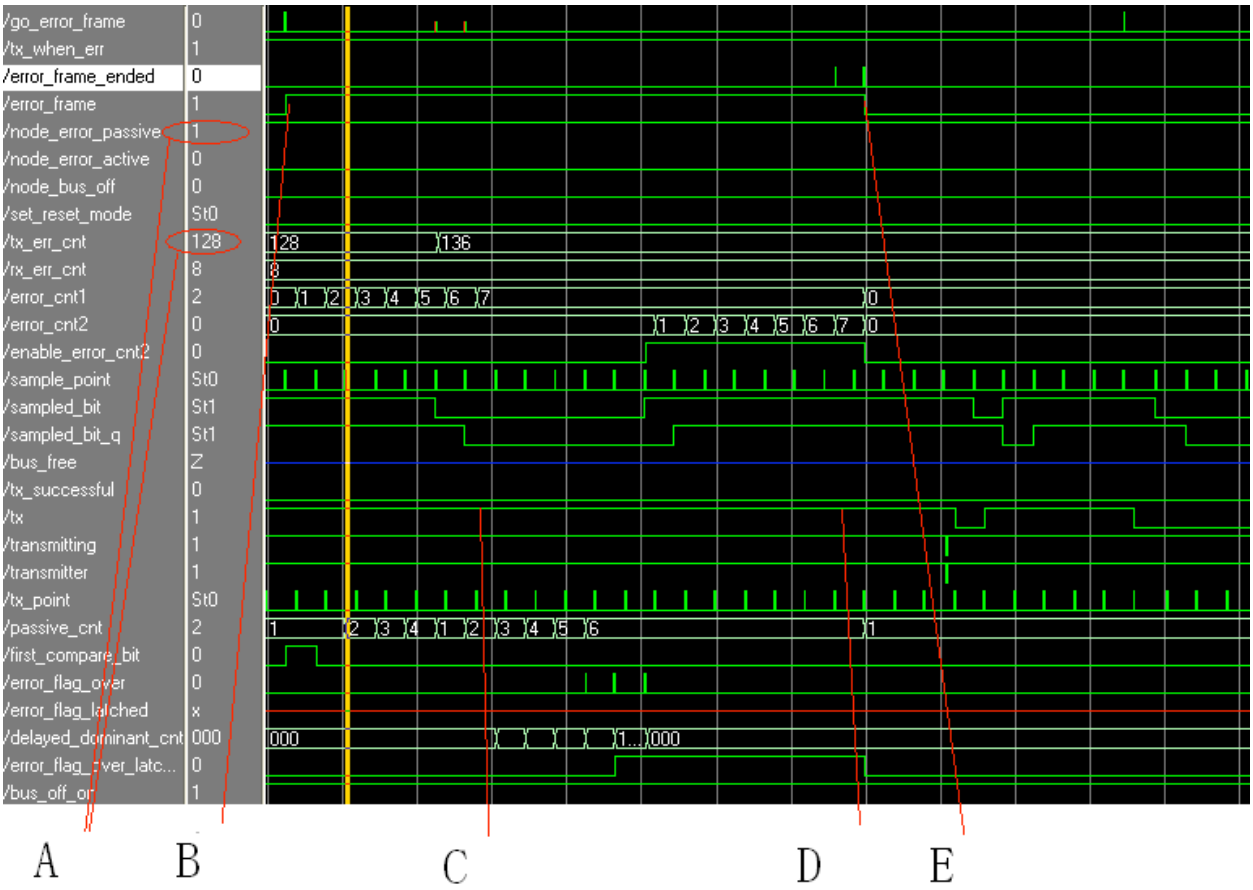


图3.24被动型错误节点错误帧的波形演示

注：A、发送错误计数器128为错误被动型。

B、检测到错误，预备发错误帧。

C、由于是被动型，先连续发6位隐性位作为错误标志。

D、再连续发8位隐性位作为错误结束标志。

E、错误帧发送结束。

3.6.2.4.3 总线关闭型

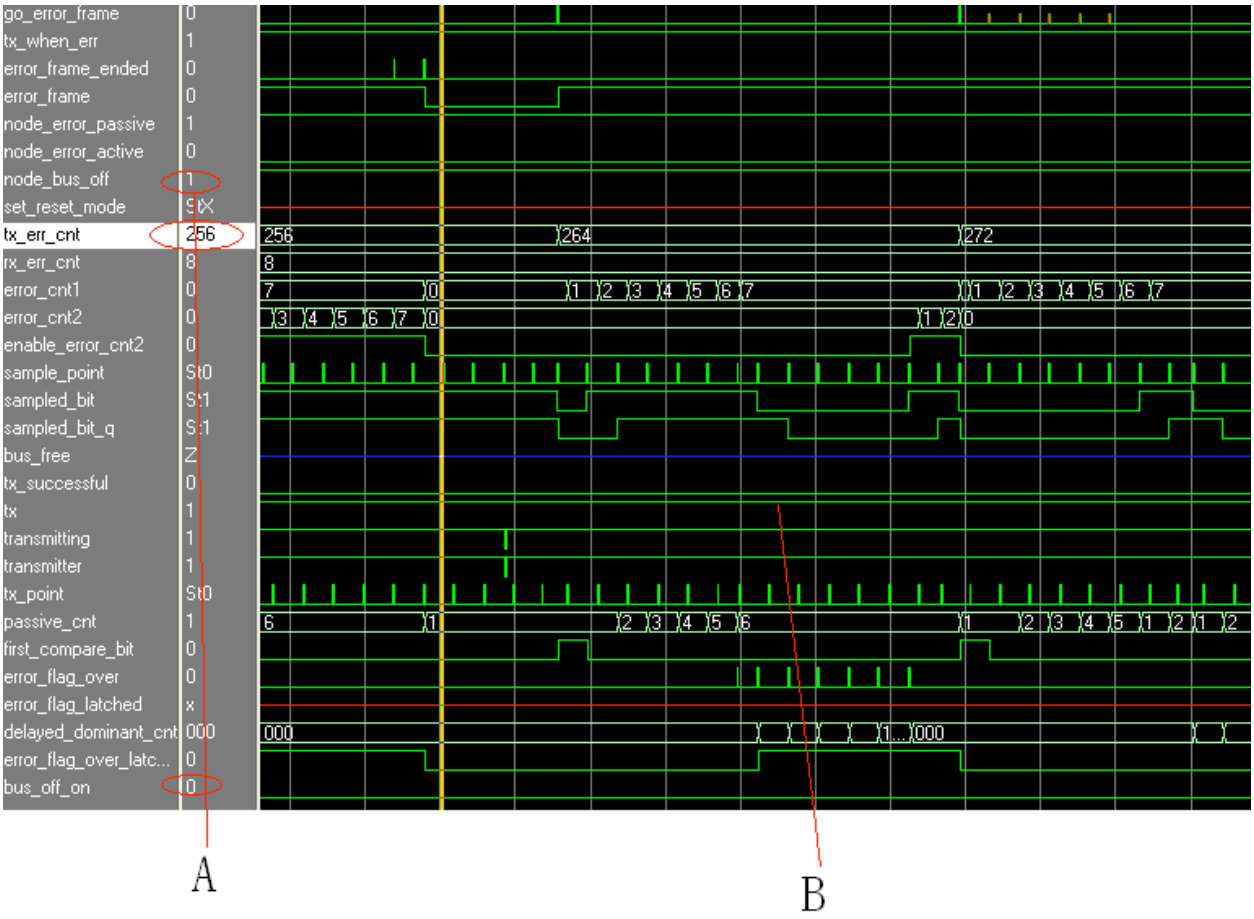


图3.25总线关闭型的节点的错误帧波形演示

注：A、发送错误计数器值为256，现处于总线关闭状态

B、恒发送1，不再参与发送(虽然发送数据帧还没有完成)，由于错误计数器的状态机并不因此停止,它还可能检测到位错误，从而发错误帧，但由于在发送状态机此时强制发送的都是1，对总线没有任何影响。在这里只需设定一个限制值，不另计数器超过这个值即可(否则计数值会进行循环加法)。

3.7发送状态机CAN_TX_BSP的具体设计

在讨论完了发送时序的处理、位填充、CRC码的产生以及错误的处理，现在来讨论发送状态机，这也是发送的核心所在。首先要说明的是，**任何一个节点都有发送功能和接收功能。所有节点都在监听总线，也都在发送信号**(只是当其没有发送任务的时候，对总线默认发“1”，不对总线构成影响)。这样在CAN节点内部，**发送方与接收方的信号就可以实现共享**，为优化资源打下了基础。发送状态机进行的主要工作是按照CPU的发送请求，对发送缓冲区的数据一位一位发出。在同一时刻内总线上，可以有多个节点在进行发

送，当发生冲突时，需要进行仲裁，当优先权低的节点丢失仲裁后，变为接收器，直到接收完一帧，并检测到总线空闲且发送命令未被取消时再进行发送。发送的任一时刻若检测到错误(即使是CRC错，虽然根据CAN协议CRC错误是由检测到CRC计算错误和状态机走到ACK结束位时发出，但在这里将这种组合条件视为一种广义的错误)，需与EML单元配合尽快发出错误帧。在发送错误帧后，同仲裁丢失一样，发送器还需继续发送这一帧，直到发送完成。由于一个节点同时总在监听帧的情况(但却不总是在发送帧，一个CAN总线的平均负载率大约在25%，而每个CAN节点发送帧的概率就更小一些)，所以在发送过程中发送状态机可以使用本节点内的接收状态机内的接收信号来进行状态机的驱动，并根据这些信号来发送相应位。

在实际设计中，笔者在这个状态机内部实际使用了两个状态机。**一个状态机是“大”状态机tran_state_machine**，这个状态机记录当前发送的性质，并对其设上优先值。这五类发送性质(接照优先级从高到低)是：复位或总线关闭状态、发错误帧状态、发送数据(或远程)帧状态 、发送应答位状态、发送应答位后的状态。实际上，“发送应答位后的状态”与“复位或总线关闭”状态时对总线的影响是相同的，即都发送“1”；但为了逻辑上的清晰，还是将它们分开。优先级的设计是针对它们对总线的影响和要求其的反应速度决定的，要求反应越快的，其优先级就越高。**另一个状态机是“小”状态机tx_state_machine，这个状态机记录发送帧时的状态** 。这个状态机记录和发送帧相关的信息，它的状态有：空闲状态、帧头状态、数据状态、CRC状态、ACK域状态、帧结束及后续状态。采用这两个状态机，再结合内部的仲裁机制、错误重发机制、CRC模块、位填充模块、EML模块就可以实现CAN发送功能。首先来看一下can_tx_bsp的接口定义。

3.7.1 can_tx_bsp 的接口定义

```
module can_tx_bsp
(
  clk,//系统时钟
  rst,//系统复位

 /*btl对我的输入*/
    tx_point,//发送一位的触发信号
    hard_sync,//硬同步

  /* Mode register */
  reset_mode,//模式寄存器中的软复位信号

  /* Command register */
  tx_request,//发送命令
```

```
    abort_tx,//中止在待发帧

    /*Status register注意这是一个组合逻辑，需要实时赋值，所以都是wire型*/
    node_bus_off,//节点是否处于错误过多状态(无法向总路线上发数据)
    transmit_status,//是否处于发送状态,
    tx_successful,//一帧是否发送完成
    need_to_tx,//缓冲区是否有效，即是否需要发帧

    /* Tx data registers. */
    tx_data_0,//包含帧头ID八位
    tx_data_1,//包含帧头余下的ID三位，及DLC,RTR五位
    tx_data_2,//数据字节1
    tx_data_3,//数据字节2
    tx_data_4,//数据字节3
    tx_data_5,//数据字节4
    tx_data_6,//数据字节5
    tx_data_7,//数据字节6
    tx_data_8,//数据字节7
    tx_data_9,//数据字节8

    /* End: Tx data registers */

    /*接收BSP对我的输入，用于引导状态机*/
    rx_idle,//总路线处于空闲状态,即我可以发送数据帧,起始发数据帧头
    rx_id,//接收状态机处于接收ID状态
    rx_rtr,//接收状态机处于接收RTR状态
    rx_data,//进入发送数据位
    rx_crc,//进入发送CRC位
    rx_ack,//进入发送ACK状态
    go_rx_idle,//进入总线空闲状态的触发信号
    go_rx_id,//进入ID场的触发信号
    go_rx_ack,//进入ACK场的触发信号
    go_rx_crc,//进入发送CRC场的触发信号
    go_rx_inter,//进入发送帧间隔的触发信号
rx_inter,//发送帧间隔的状态
    go_rx_crc_lim,//进入发送CRC  结束位的触发信号
    rx_crc_lim,//CRC结束位
    go_rx_eof,//进入帧结束的触发信号
```

```
    go_rx_ack_lim,//进入接收ACK结束位的触发信号
    rx_eof,//帧结束状态
    sampled_bit,//采样值
    sampled_bit_q,//对采样值的锁存
    sample_point,//采样点
    bus_free,//总线释放，针对空闲或节点错误状态的自减一
    calculated_crc,//计算所得的CRC码
  send_ack,//发送ACK位，只是在本节点为接收节点时才发0
  /*对接收BSP的输出*/
    transmitter,//是否是发送节点
    transmitting,//是否在发送(正常帧或错误帧或ACK)
      /*位填充对我的输入*/
    bit_de_stuff_tx,
    /*我对位填充的输出*/
    bit_stuff_cnt_en, //位填充的使能信号
    bit_de_stuff_reset,//位填充计数器的清零信号
    tx_point_q,//发送点的缓存信号
    tx_q,//发送值的缓冲信号
    /* Tx signal */
    tx,//我方对总线的输出
    tx_next,//我方对总线的输出即将发送的tx

  set_reset_mode,//由于节点错误过多，进入复位状态的触发信号
  bit_err,//检测到的位错误
  ack_err,//检测到的ACK错误
node_error_passive,//节点处于错误被动状态
  tx_state,//发送状态

//eml的输入
    tx_when_err,//发错误帧时对应发的位
    error_frame_ended,//错误帧结束
    error_frame,//发送错误帧
    go_error_frame,//发送错误帧的触发信号
    error_cnt1,//发送错误标志位的计数器
    error_cnt2,//发送错误结束标志的计数器
    enable_error_cnt2,//  发送错误结束标志的计数器的初始信号
//对eml的输出
    arbitration_lost,//仲裁丢失信号
```

arbitration_field,//仲裁场

);

其中，特别需要注意的是transmitting、transmitter、tx_state的区别。Transmitting是指当前节点的发送值会对总线有影响(它在发送数据、远程帧，发送错误帧以及接收方发送ACK位时有效;在丢失总线仲裁、不是发送方时无效)。Transmitter是指一个发送节点，它从有发送请求到总线又一次空闲之间有效，标志着总线上有一个发送节点，是总线的状态。Tx_state标志着是否是在总线上正在发送的节点，是这个节点本身的状态。

3.7.2算法的具体实现

3.7.2.1 “大”状态机的状态转换

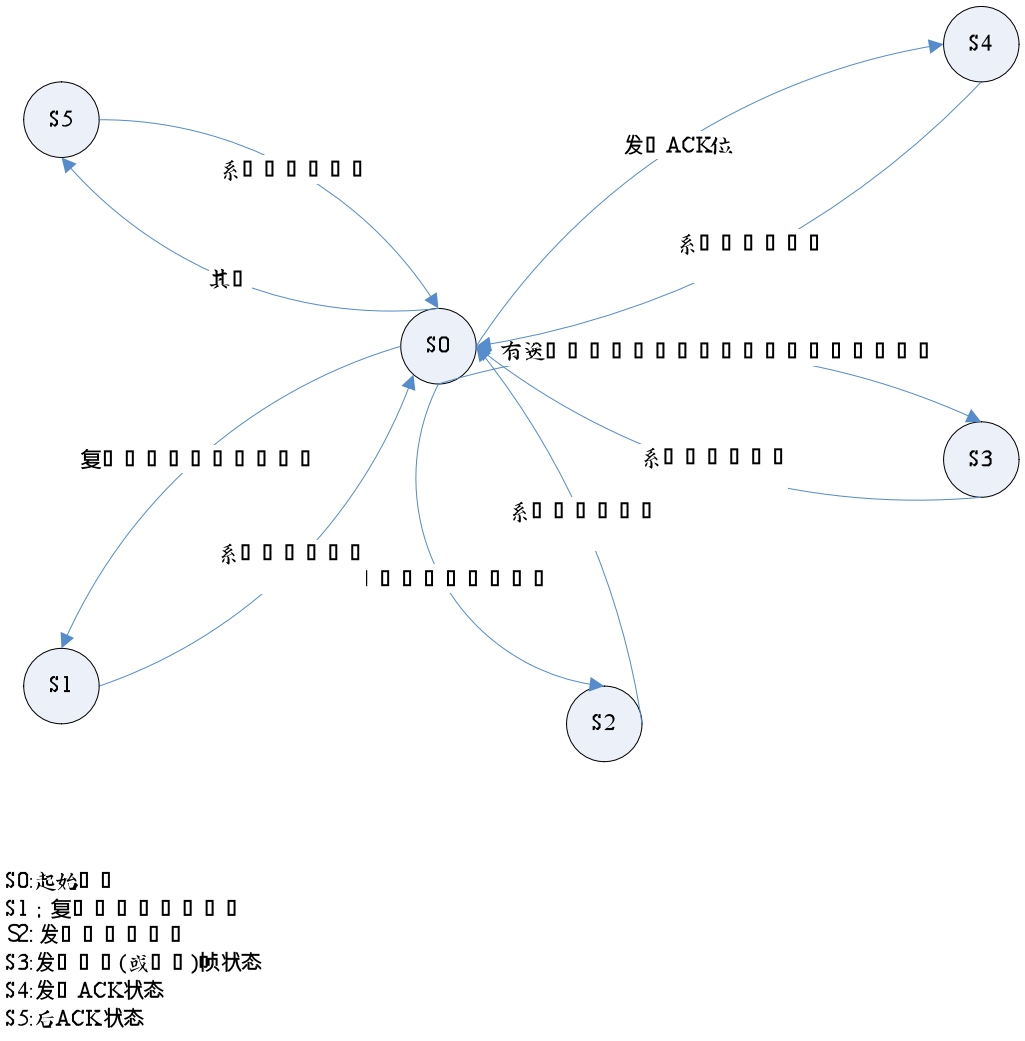


图3.26发送状态机中“大”状态机状态转换图

其中状态S3的触发信号是非常关键的，因为这是发送方进行发送帧和重发机制的最重要部分。它是由三部分构成的(任一部分成立都可以进入状态S3)：

(1) 有发送请求、总线空闲，如果此节点被挂起(因是错误被动节点在发送一帧前，要比正常节点多等待8个隐性位)并已等待足够长或者有发送请求，现处于间歇帧的最后一位，且已从总线检测到一个帧起始位。即使丢失仲裁，也可以等待别的节点发送完毕后，再重发，因为发送请求是只有在发送完成后或当有退出发送的信号到来，且现在不处于发送状态时才会被清零。因此这个信号也可以实现中止发送的功能。

(2) 错误帧发送完毕。这是针对于错误重发机制。

(3) 本身已处于发送状态。

需要注意的是：在这个大状态机和下面的小状态机里都有发送ACK的状态，所不同的是，在这个大状态机里由于ACK的优先级低，所以一旦这个节点是发送方，就会进入S3状态(发送数据状态)，在那里对ACK状态的定义使其在此刻发送“1”。而当大状态机转到这个ACK状态时标志着这是个接收方，所以在ACK这位应该发送“0”。

3.7.2.2小状态机的状态转换

S0:开□ □ □
S1:准□ □ □
S2:进□ □ □ □ □
S3:启动CRC及位□ □
S4:发□ □ □ □ ID位
E0:发□ □ □ □
S5:发□ RTR位
S6:发□ IDE位及保留位0及DLC位
S7:发□ □ □ □
S8:发□ □ □ □
S9:发□ □ □
S10:发□ CRC场
S11:发□ CRC delimiter
S12:发□ ACK
S13:发□ ACK delimiter
S14:发□ 1(帧结□ ′ 间□帧)
S15:发□ □ □

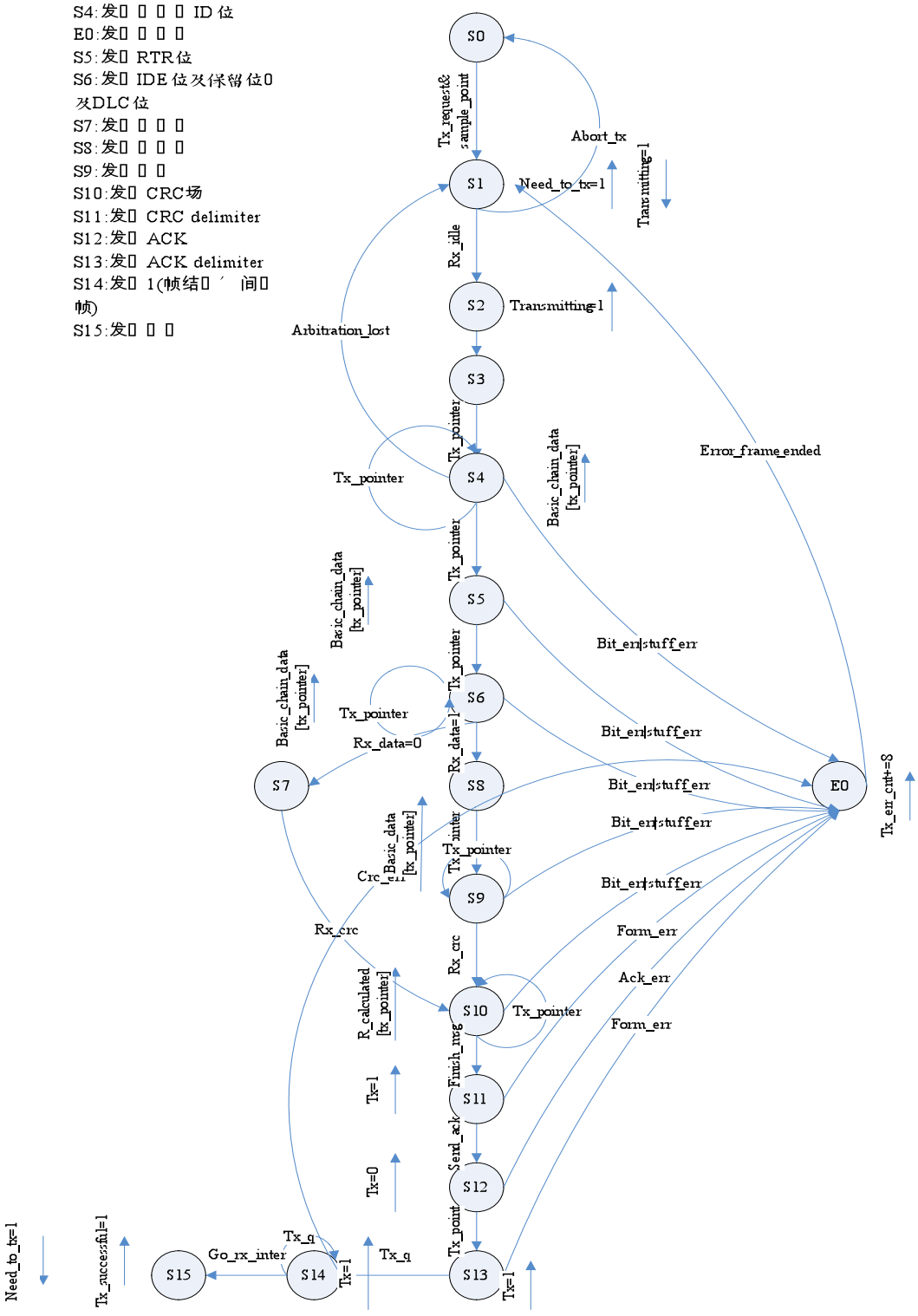


图3.27发送状态机中“小”状态机的状态转换图

对发送一帧，采用分而治之的策略，即把一帧分为若干部分，并对每一个部分使用不同的算法进行发送：

(1) 第一部分：帧头basic_chain。根据CAN的协议，在每一帧开始需有一位显性位，标志着帧开始，首先要把这位加在数组的最低位。其后，根据对外接口的定义，把ID场由高向低(这里的数据已经由ibo模块进行了逆置)地对应填充到这个数组的由低向高各位。同时在ID场后面加入两位“0”，代表着“标准CAN标志位”和“保留位”。最后再添上数据长度控制场的4位数据。下面显示的代码是对这一过程的实现。

Assign basic_chain = {r_tx_data_1[7:4], 2'h0, r_tx_data_1[3:0], r_tx_data_0[7:0], 1'b0};

(2) 第二部分：帧中的数据 basic_chain_data。把要发送的数据排列在这个数组内。下面显示的代码是对这一过程的实现。

assign basic_chain_data = {r_tx_data_9, r_tx_data_8, r_tx_data_7, r_tx_data_6, r_tx_data_5, r_tx_data_4, r_tx_data_3, r_tx_data_2};

在发送过程中，要根据数据长度控制场的长度来发送这里面的数据。这个长度是过对DLC里的数据进行乘8减一的方法实现的，下面显示的代码是对这一过程实现。

assign limited_tx_cnt_std = tx_data_1[3] ? 6'h3f : ((tx_data_1[2:0] <<3) - 1'b1);

(3) 第三部分：CRC场中的数据 r_calculated_crc。发送方将根据本身的接收功能算得的CRC码传输出去。CRC的计算过程是从帧开始位进行直到数据域的完成结束。并在其最高位补上一个零，作为crc_delimiter。

(4) 每四部分：传输完上述信息，作为发送方只需发“1”就可以了，因为对于发送方而言，ACK，ACK_DELIMITER这两位都是发“1”，而之后的帧结束位以及后面的间歇帧也都是发“1”。而作为接收方，当进行到发送ACK位这一状态时，需向总线发送“0”，之后也全部发送“1”。

(5) 这四部分及位填充结合起来需要使用一个发送指针tx_pointer。在发送信息(即前三部分)过程中，每来一个发送点便加一，同时将信号发送出去。但是如果这个发送方是因为检测到一个硬同步而开始发送帧头时，不能发送“帧起始位”，因为这回造成总线仲裁的不正确。但此时这个指针会加一，在下一个发送点到来时就可以直接发送ID场。与之相关的CRC计算过程不会受到影响，因为CRC所采用的串行输入值是接收值，而不是发送值，这样由于“线与”逻辑，还是可以把这个“0”接收到，进行正确的CRC计算。另外与之相关的是位填充，有这种情况下，位填充模块会自动把上一次发送的值记为“0”(尽管并没有发送“0”)，来保证正确的位填充。由于是在不同的数组间进行指挥，还涉及到指针清零的过程。这个清零过程可由6个事件触发：节点复位、总线进入空闲状态、发生错误、发送帧头时指针值为帧头长度减一、发送数据帧头时指针值为数据长度减一、CRC结束位信号。在此基础上，结合“小”状态机发送相应的信息数组或相应位，就可实现对帧的发送。

(6) 发送成功的标志：tx_successful。这个值是在一个传输节点进入到间歇帧状态，并在这个过程中未发生错误及丢失仲裁时有效。之所以不能因为由其进入间歇帧状态就认为

发送成功，是因为这个进入间歇帧状态的信号值是由接收方给出的，而这是对整个总线上状态的反映，所以必须要确定这个信号是由当前这个发送方对总线的影响而发生的。另外，在程序中“发送方”这个信号是从开始发送到总线上的一帧结束有效的。即使这个信号有效，也只能表明这个节点在这个帧的开始时是发送方。只有在这个节点在传输此帧的过程中没有发生错误且没有丢失总线仲裁的情况下，才能确定其是正确的发送了一帧。

附:发送一个完整帧贴图

注：为使贴图紧凑，笔者采用的帧数据长度为1，但需要两张贴图才可以看清楚全过程。

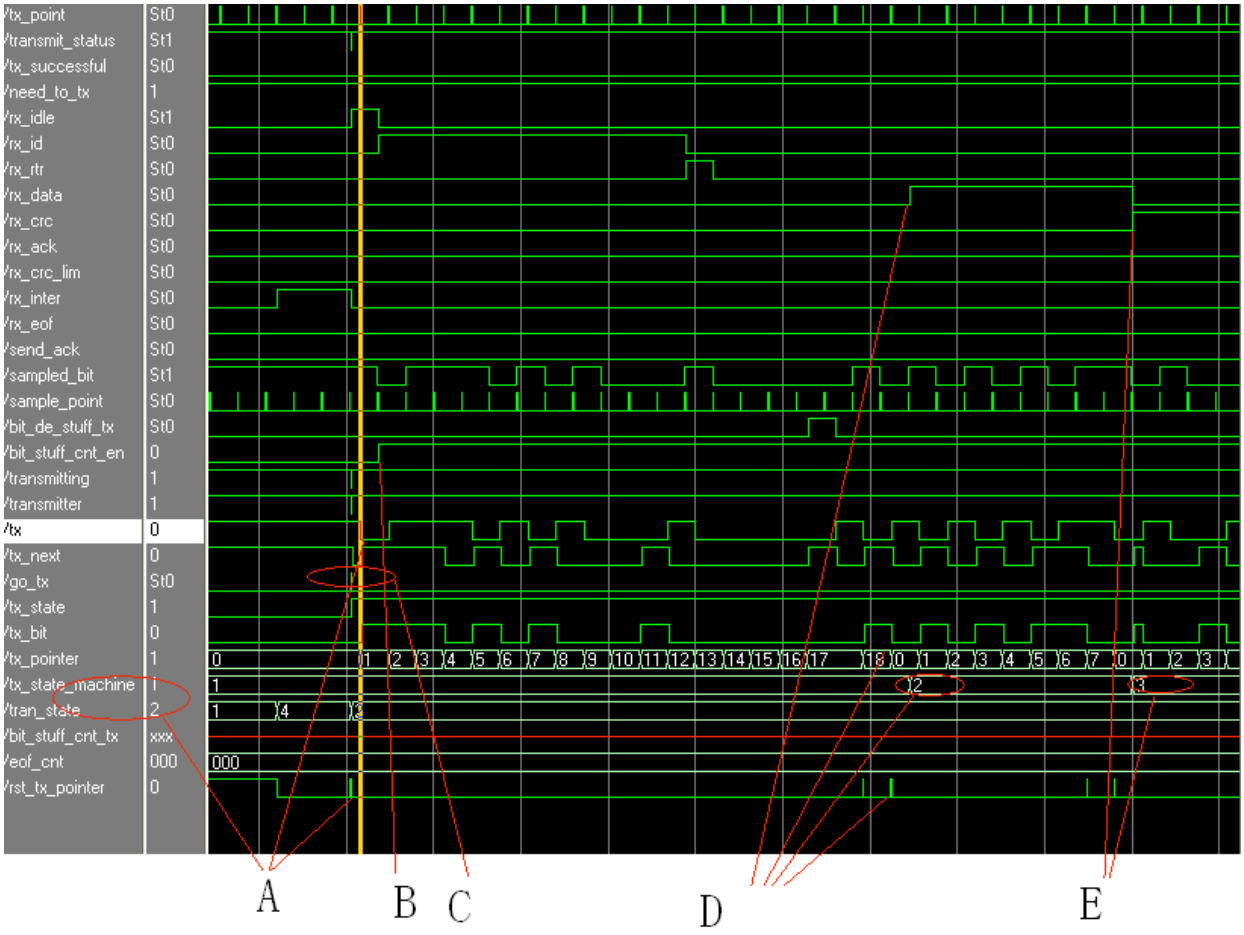


图3.28 发送一个数据帧的波形演示a

A、 开始发送帧起始位，大状态机进入发送数据状态，小状态机进入发送帧头状态。

B、 在发ID场时位填充信号有效(之前已经记录了发送的帧起始位)

C、 作为第一个占据总线的发送方，go_tx即是进行CRC运算的使能信号。

D、 帧头计数到最后(0-18共19位)，rst_tx_pointer有效，发送指针清零，小状态机准备进入发送数据位。

E、 数据计数到最后(本次只发8位数据，0-7位)，发送指针清零，小状态机准备进入发送CRC状态。

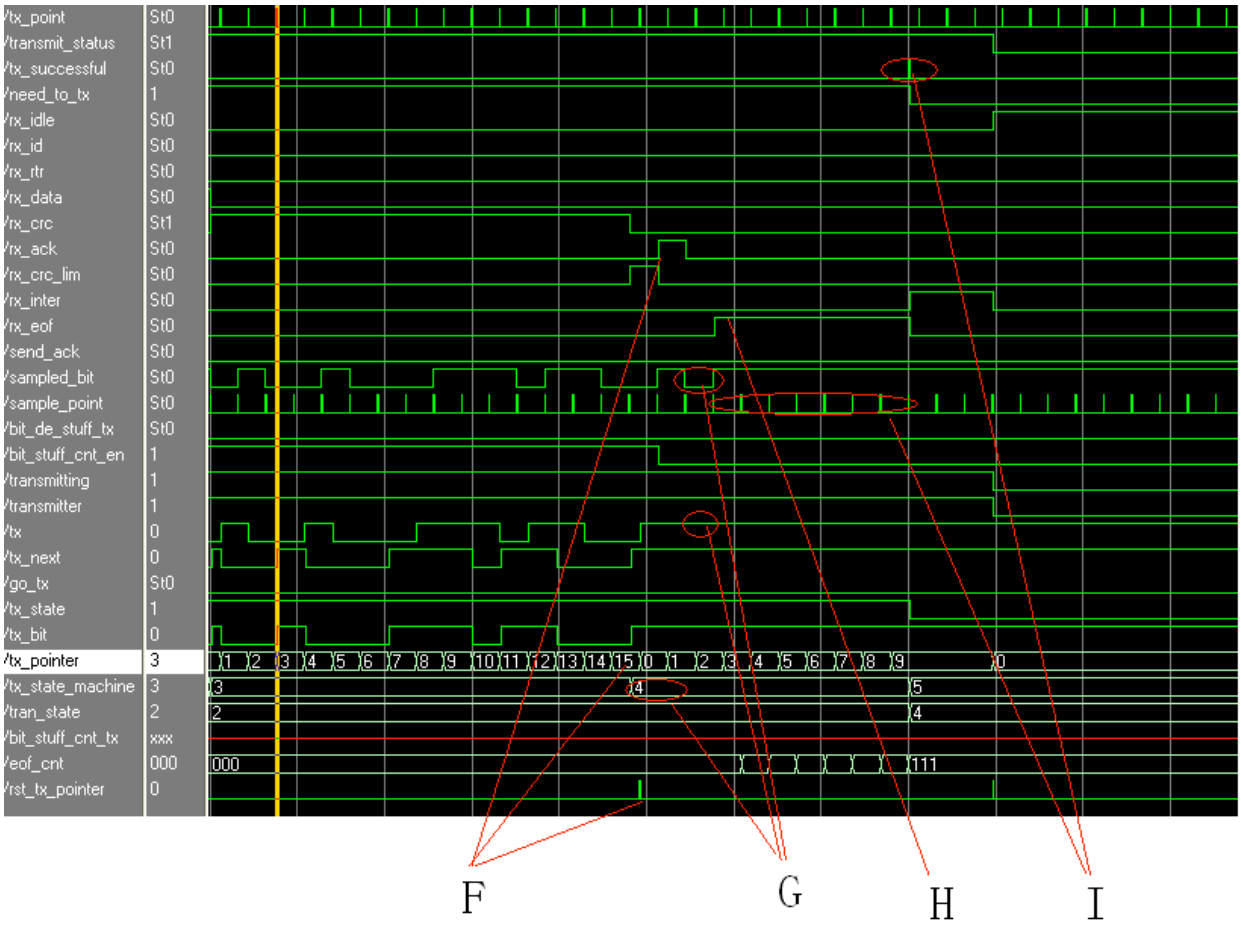


图3.29 发送一个数据帧的波形演示b

F、 CRC场发完(15位CRC值及一位CRC_DELIMITER),小状态机进入ACK状态，大状态机没有进入ACK状态因为其是发送节点。

G、 在ACK这位上，发送方发送“1”，而总线采样值为“0”，证明有接收方发送了“0”，这样就使应答成功。

H、 进入发送帧结束标志位状态。

I、 连续发送7位1(1个ACK_DELIMITER,6个帧结束位，虽然标准上是应该接收到7个“1”为完全结束，但协议也规定第7个“1”可以是“0”以保证快速通信)。此时发送成功。

3.7.2.3总线仲裁机制

CAN总线的仲裁机制规定ID位与RTR标志位共同组成仲裁场，由于现与逻辑，任何一个节点发出的显性位会湮灭其它节点发送的隐性位，也就是说在这个仲裁场内发生的发送“1”而接收“0”的事件不构成位错误事件而构成仲裁丢失事件。丢失仲裁的节点会在检测到总线空闲之后继续把这一帧发送完毕。

在具体实现中，只要先确定好仲裁场的位置，同时进行检测发生发送位为隐性位而采样值为显性位的事件即可。而仲裁场的位置就是接收ID场状态和接收RTR状态这两个信号的相或值。在丢失仲裁之后，由于记录发送请求的那一位信号未被清零所以这一节点在总线空闲时仍可发送。

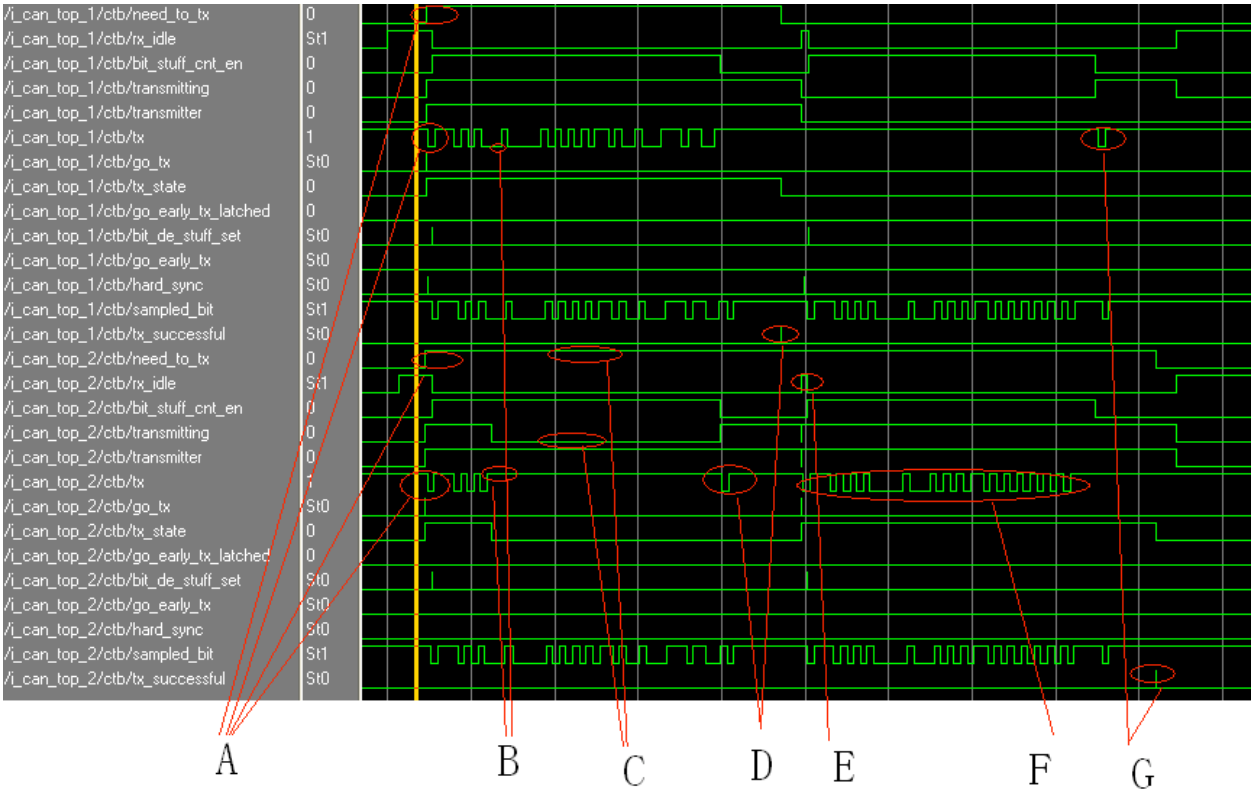


图3.30 总线仲裁的波形演示

注：

A、两个节点同时发送数据帧

B、节点2在此时丢失了仲裁，退出了发送

C、虽然退出了发送，但need_to_tx这个发送请求信号仍然有效。

D、在ACK位发送了一位应答，节点1在接到应答后又过了6位帧结束位，发送成功。

E、节点1发送成功后又过了一个间歇帧，总线空闲。

F、节点2检测到了总线空闲，开始发送帧

G、节点1作为接收方，发送应答位，之后节点2发送成功。

3.7.2.4 发送状态机可以检测到的错误

发送状态机可以检测到的和发送有关的是两类错误：应答错与位错误。**应答错误的检测**是很容易的，那就是在发送状态机接收应答场的这一状态时接没有检测到一位显性

位。**位错误的检测**则相对麻烦一些，因为位错误有六种例外(即在这六种情况下，即使发送的值与采样的值不同，也不发生位错误)。这六种例外是：(1)仲裁场时发送“1”；(2)发送方发送ACK位；(3)被动错误发送错误标志时；(4)发送错误帧标志位后且没有开始发送错误结束位(即被其它错误主动节点的标志位覆盖的位)；(5)错误结束位计数器为7时(虽然协议规定需发8位，但协议规定在发送7位后检测到1个“0”即其它节点的重发数据帧的信号并不是错误)；(6)接收方接收帧结束位时在已接到6个“0”的情况。

位错误的演示：

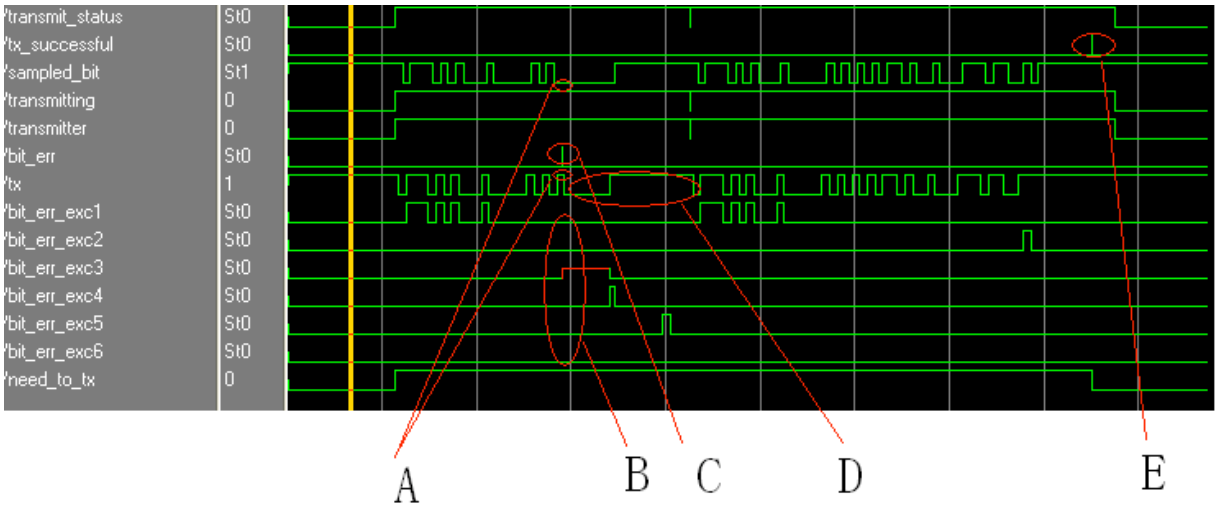


图3.31 位错误检测的波形演示

A. 发送的是“1”而采样的值是“0”。
B. 不是位错误的那6种例外。
C. 产生一个位错误。
D. 发送错误帧。
E. 重发并成功。

ACK错误的演示

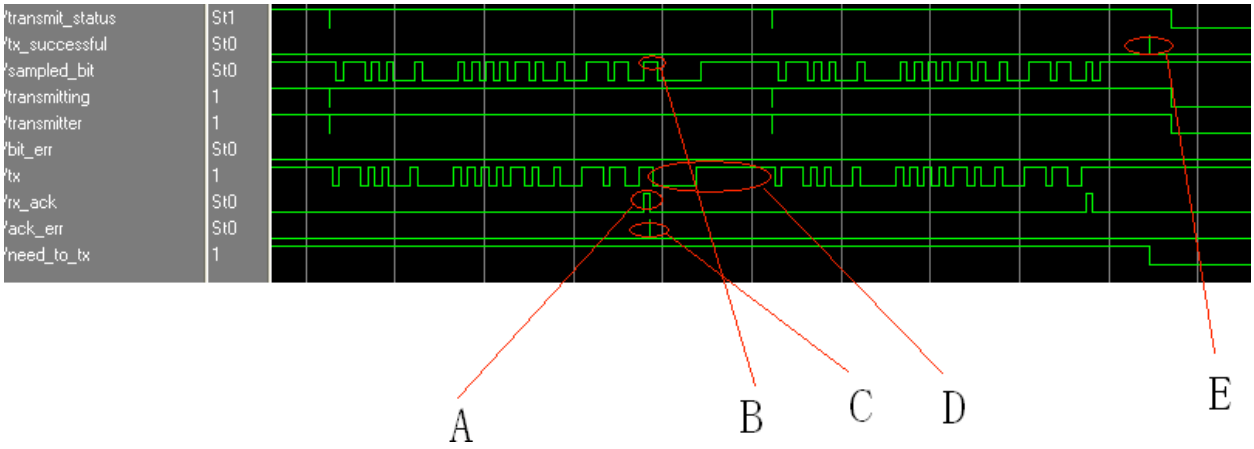


图3.32 ACK错误检测的波形演示

A、到了接收ACK位的状态。

B、总线采样为“1”而不是“0”。

C、产生ACK错。

D、发送错误帧。

E、重发并成功。

3.7.2.5错误重发机制

检测错误、发送错误帧并进行对数据帧的重发这也是发送状态机的一个重要功能。当一个节点发现错误时，立刻发送错误帧。由于这是广播总线，其它节点也会发现错误如位填充错误(因为错误帧标志位已经破坏了位填充的原则)或格式错(比如应该接收DELIMITER的位由于错误帧标志位发送的“0”被采样)。这样其它节点也会立即发送帧，这就像滚雪球一样，使这条总线上的所有节点都检测到错误，从而发送错误帧。因为错误帧并没有仲裁机制，所以错误帧里的某一位有可能被其它节点发送的位覆盖掉。**这样总线上的错误帧并不是任一个节点的错误帧，而是所有节点发送的错误帧共同叠加的结果。**因此，这个帧是总线本身的帧。而事前的发送方在检测到总线上这个错误帧之后，会与仲裁重发机制一样的方法，进行发送直到发送成功。

错误重发机制的演示

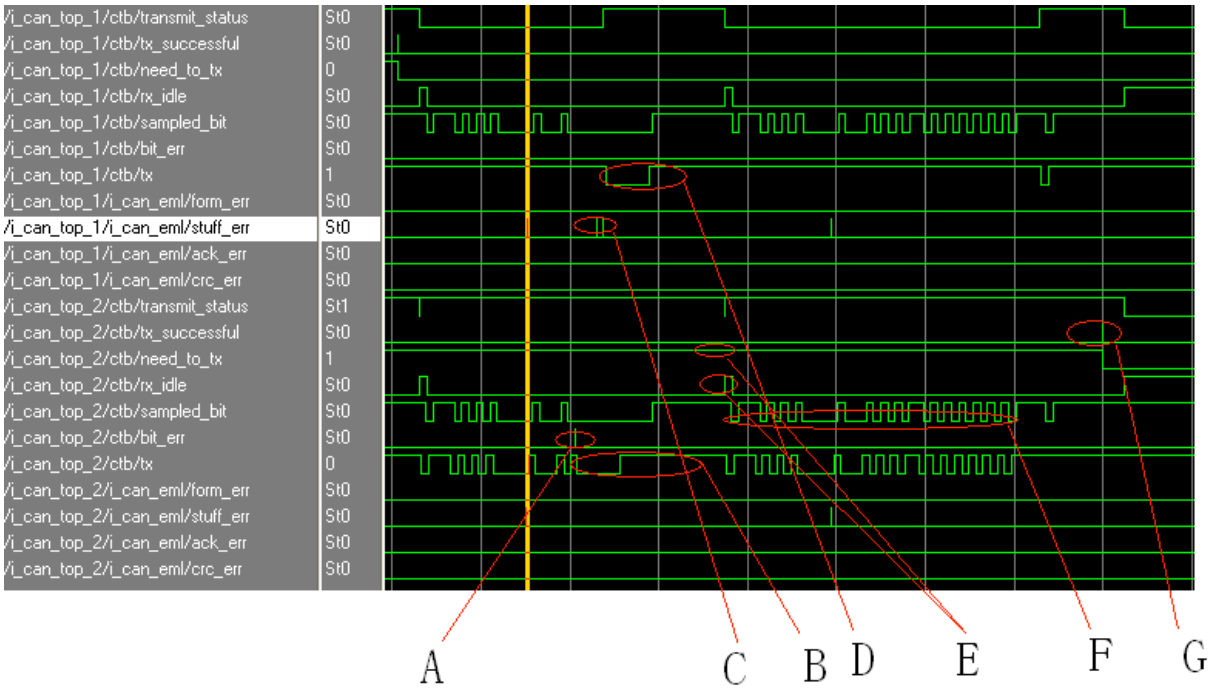


图3.33错误重发机制的波形演示

A、 节点2作为发送方在发送过程中检测到了一个位错误。

B、 节点2开始发送错误帧。

C、 节点1检测到了因节点2发送的错误帧引起的位填充错。

D、 节点1开始发送错误帧。

E、 节点2发送完错误帧后，等到了总线空闲，并且还有发送请求。

F、 节点2重发数据帧。

G、 节点2发送成功。

3.7.2.6中止发送的功能

上述无论是总线仲裁还是错误重发都会把未发完的数据帧继续发完，但是如何发送一个命令，使其中止发送呢？只要将重发机制中重要的信号need_to_tx在中止命令下清零即可。

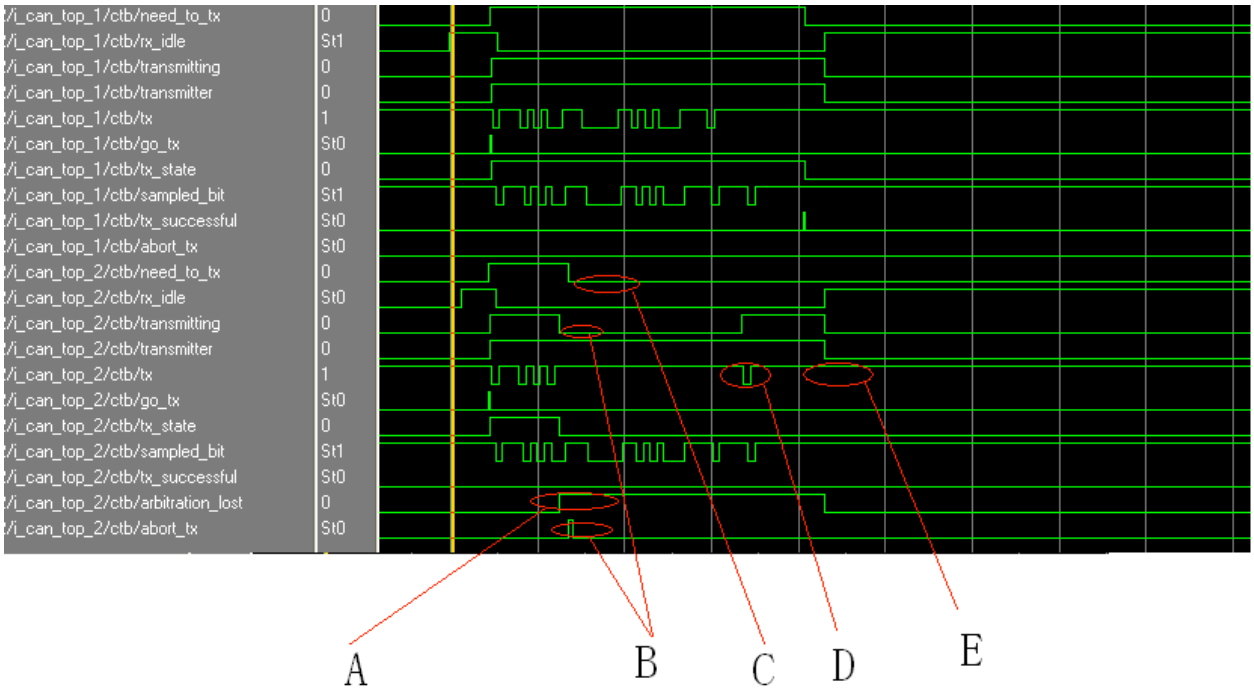


图3.34中止发送的波形演示

A、节点2丢失了总线仲裁

B、CPU对节点2发出了中止发送命令，且此时节点不在发送状态

C、对重发机制最重要的变量need_to_tx被清零

D、发送命令的取消不影响其发送ACK位

E、不重发数据帧

至此，关于发送状态机全部讨论完毕!

3.8 CAN软核内部寄存器组的具体实现

CAN软核的所有数据的输入输出，以及命令控制字都需要经过内部的寄存器组进行译

码。

3.8.1单元寄存器 can_register的具体实现

接口定义：

```
module can_register
( data_in,//输入数据
   data_out,//寄存器记录数据
   we,//写有效信号
   clk,//系统时钟
   rst,//异步清零信号
   rst_sync//同步清零信号);
```

单元寄存器是组成寄存器组的独立单位，每一个单元寄存器默认长度为8，默认清零值为零。可以实现同步、异步清零(即当同步或异步清零信号有效时，寄存器的值清零)。当清零信号无效时，寄存器的值等于输入值(默认长度)。

3.8.2 寄存器组can_regs的具体实现

3.8.2.1接口定义

```
module can_regs(
      clk,//系统时钟
      rst,//系统复位信号
      addr,//地址线
      data_in,//数据输入线
      data_out,//数据输出线
      wr,//写寄存器信号
      .
      .
      .
);
```

省略部分为有具体功能的寄存器(如命令寄存器、状态寄存器等)。地址译码的规则见“整体设计”部分给出的接口定义。

3.8.2.2与发送相关的寄存器的读写时序的配合以及相应的清零条件

(1) 发送缓冲区(tx_data_0—tx_data_9)：

写信号有效的条件：wr有效、处于复位状态、发送缓冲区有效(对应状态寄存器的相应位)、地址从8至17。

读信号有效的条件：此缓冲区不允许读。

异步清零：无。

同步清零：无。

(2) 命令寄存器中发送请求位

写信号有效的条件：wr有效、地址为1。

读信号有效的条件：无。

异步清零：采样点信号有效；复位模式，或未发送时发出的中止发送信号，或发送请求位有效。

同步清零：系统复位信号。

寄存器长度设为1。

(3) 命令寄存器中中止发送位

写信号有效的条件：wr有效、地址为1。

读信号有效的条件：无。

异步清零：采样点信号有效；复位模式，或未发送时发出的中止发送信号，或发送请求位有效。

同步清零：系统复位信号。

寄存器长度设为1。

(4) 状态寄存器“发送缓冲区可用”位：

写信号有效的条件：无。此位与can_tx_bsp的need_to_tx信号相关。

读信号有效的条件：地址线为1。

异步清零：无

同步清零：无

寄存器长度设为1。

(5)状态寄存器“发送完成”位：

写信号有效的条件：无。此位与can_tx_bsp的tx_successful和abort_tx信号相关。

读信号有效的条件：地址线为1。

异步清零：无。

同步清零：无。

寄存器长度设为1。

(6) 状态寄存器“发送完成”位：

写信号有效的条件：无。此位与can_tx_bsp的tx_successful和abort_tx信号相关。

读信号有效的条件：地址线为1。

异步清零：无。
同步清零：无。
寄存器长度设为1。
(7) 状态寄存器“发送状态位”位：
写信号有效的条件：无。此位与can_tx_bsp的tx_state信号相关。
读信号有效的条件：地址线为1。
异步清零：无。
同步清零：无。
寄存器长度设为1。
(8) 状态寄存器“节点是否为总线关闭”位：
写信号有效的条件：无。此位与can_eml的node_bus_off信号相关。
读信号有效的条件：地址线为1。
异步清零：无。
同步清零：无。
寄存器长度设为1。
(9)“发送错误计数器”
写信号有效的条件：无。此寄存器与can_eml的tx_err_cnt相关。
读信号有效的条件：地址线为6。
异步清零：无
同步清零：无
寄存器长度设为9。
(10)“接收错误计数器”
写信号有效的条件：无。此寄存器与can_eml的rx_err_cnt相关。
读信号有效的条件：地址线为5。
异步清零：无
同步清零：无
寄存器长度设为9。
(11)复位寄存器
写信号有效的条件：写信号有效，地址为0。
读信号有效的条件：地址线为0。
异步清零：系统复位信号。
同步清零：can_eml提供的set_reset_mode信号。
寄存器长度设为1。默认清零值为1。

3.9顶层模块can_top的具体设计

模块can_top的功能根据Avalon总线的时序，把CPU相应的值及地址打入CAN软核，向

其中写入或读出数据；并把CAN软核里最关键的两个值“tx”、“rx”两个值引到CAN总线(一对双绞线)上。这个模块还要把上述讨论过的所有模块和与接收有关的所有模块用“wire”型变量连接起来。

3.9.1 接口定义

```
module can_top
(
        tx,//发送到总线上的值
        rx,//从总线接收的值
        rst,//系统复位信号
        clk,//系统时钟
        cs_n,//片选信号，低有效
        wr_n,//写入数据信号，低有效
        rd_n,//读出数据信号，低有效
        addr,//地址信号
        data_inout,//数据输入输出的双向总线
        outputenable_n//Avalon总线对从设备输出的控制信号，低有效
);
```

其中，~wr_n信号与寄存器组模块里的写信号相连；addr与寄存器组里的地址信号相连；data_inout作为输入时与寄存器组里的输入信号相连，作为输出时与寄存器组里的输出信号相连。在默认情况下(即outputenable_n低有效时)，data_inout作为输出，因为Avalon总线默认是读数据，但此时也须有一个读信号(即rd_n)信号，否则无法保障其能读出有效信号。另外一个需注意的问题是，Avalon总线的地址宽度是32位，而CAN软核的地址宽度是8位，这就涉及到了一个地址匹配的问题：若CPU方面未对地址进行除四处理的话，CAN软核内部就要对输入的地址进行除四处理然后再进行寻址。经过这样的连接和处理后，CAN软核就可以与使用Avalon总线时序的CPU相连接并正常工作了。

3.9.2 CAN_TOP的最终模块图

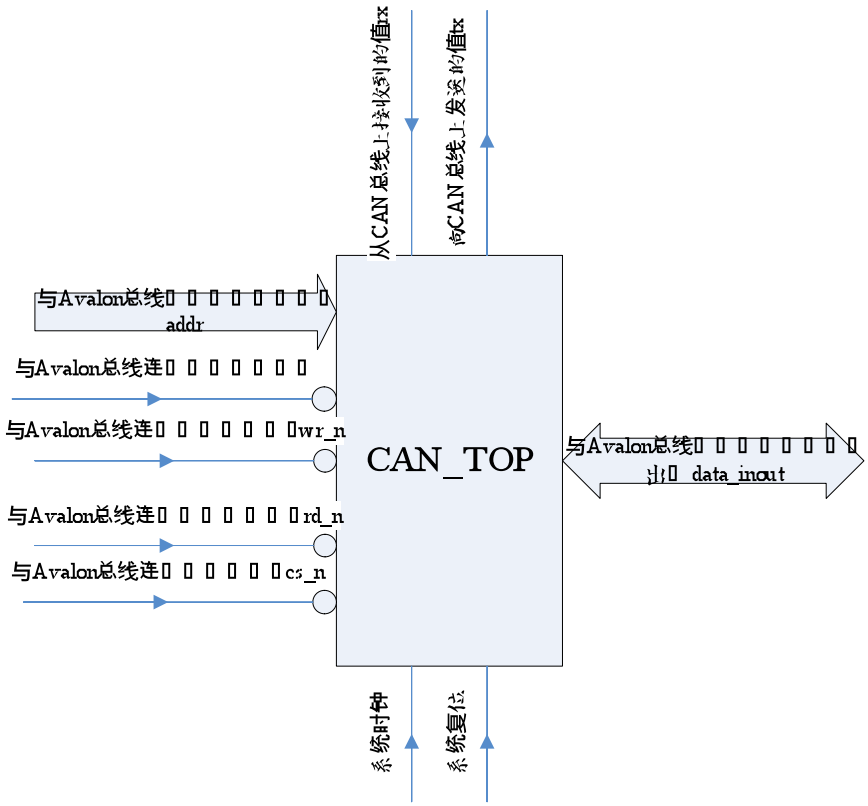


图3.39 CAN_TOP的模块图

3.10模拟仿真

硬件描述语言编程的精华在于模拟仿真，因为据经验数据表明当软件模拟成功时，90%的硬件下载都没有问题。本人使用的是ModelSim仿真软件。在仿真过程中，笔者使用了如下技术：

接收方Rx的反馈。因为CAN的发送方与接收方是一个闭环反馈系统，这需要对发送的数据进行一次延时，使用wait语句检测到发送信号的变化，并使用repeat语句进行一个时间份额的延迟，反馈到输入端，这样可以使其在位定时的同步段，就可以将值得到，并进行正确的采样。

单节点发送时，ACK信号的产生。因为在模拟一个节点发送的时候，没有节点在相应位发出ACK信号，这样会使发送方检测到ACK错。加入一个信号与接收值进行相与构成新的接收值，使用wait语句得到发送ACK的状态，然后，对这个信号值得0，从而反馈到接收方的值也为0，产生ACK位的效果。

多节点发送时，模拟多个节点，以及多个读、写寄存器的函数，同时使用fork…join

语句对这几个节点进行并发测试。另外，要接出一个信号，对各自延迟的发送信号进行与运算，来模拟“线与逻辑”，然后将这个值反馈到所有节点的接收端。这是测试总线仲裁、错误重发及位定时、同步的基础，也是模拟中的难点所在。

综上，再采用适当的延时信号以及初始值，就可以模拟能出现的情况。以笔者的经验，只有想不到的模拟条件，而绝不会有模拟不出来的情况。

4. 硬件下载与测试

4.1 下载与连线

本组采用的综合器是Quartus4.1下带的综合器，下载元件是Altera公司的EP1K30QC208C-3型号的FPGA。使用的实验台是革新公司的实验台。首先将这个工程在Quartus下进行编译、综合，生成了器件使用的报告以及下载到FPGA上的网表。然后根据测试的需要在Quartus软件里对FPGA进行管脚分配。管脚分配的准则是从实验手册上找到相应FPGA板上的空闲管脚，然后将顶层模块的输入输出信号线一位一位的对应到空闲管脚上。对一些已有内部连线的管脚(如产生上升沿的按键、点阵、七段显示灯)要按照输入输出信号的功能进行恰当分配。其中有四个七段显示灯已实现了内部译码，将其共阳极直接接到电源，就可以将四位信号以十六进制显示，十分方便。最后通过JTAG口把程序下载到FPGA上。

4.1.1编译报告

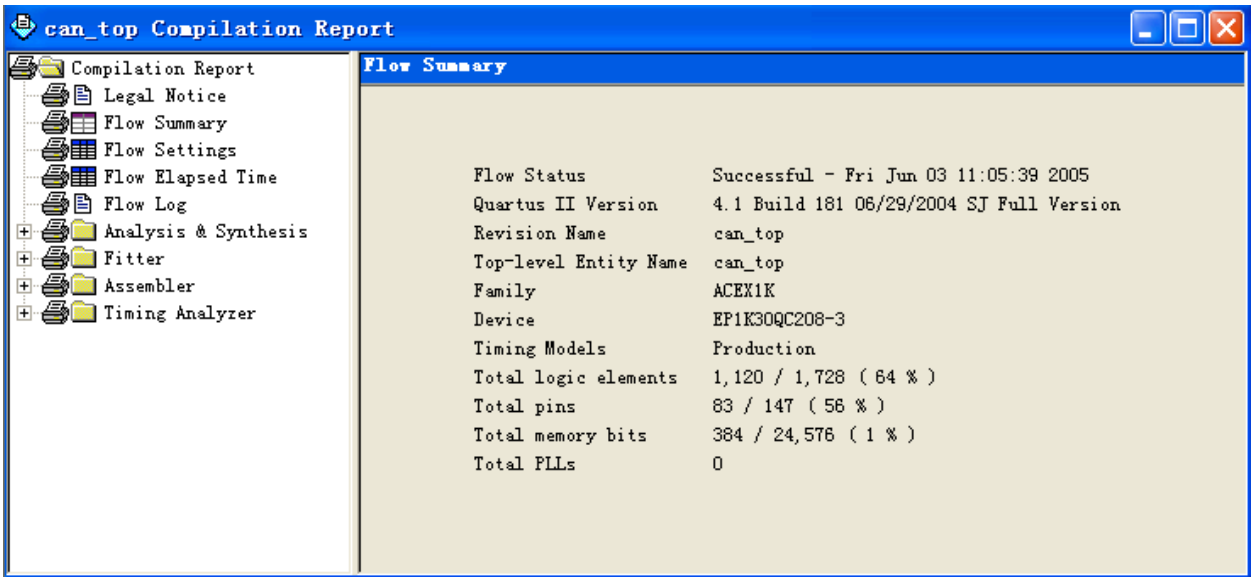


图4.1编译报告图

在这份报告中可以看出使用的FPGA是ACEX1K系列，使用了1120个逻辑单元、83个引脚和384位存储单元。

4.1.2测试的顶层模块图(接口说明见下表)

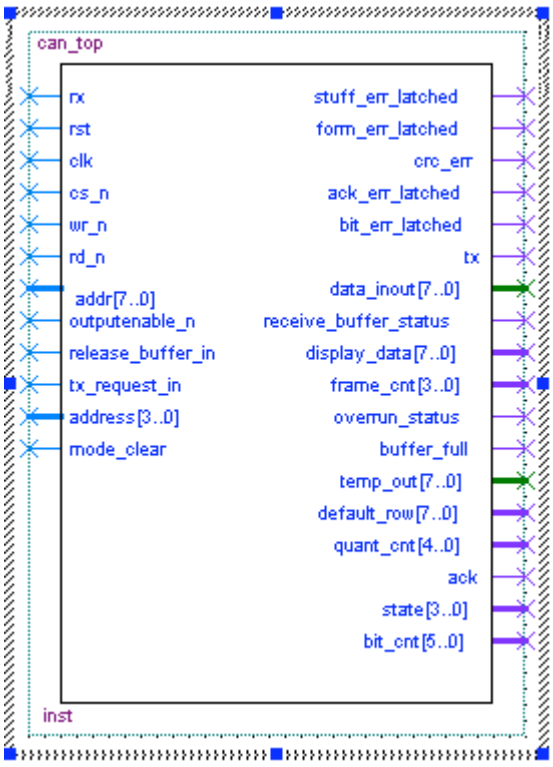


图4.2测试时的对外模块图

4.1.3实验台各管脚所代表的信号如下所示

表4.1实验台管脚的连接方案表

管脚号及其与实验台的连接	信号及其功能
12 接一LED	Receive_buffer_status 表示是否接收到一帧
13接一LED	Overrun_status 接收缓冲区是否过载
17接一LED	State[0]与另同组的另三个信号共同显示接收、发送状态
18接一LED	State[1]
19接一LED	State[2]
20接一LED	State[3]
39 与点阵下方的开关默认连接	Address[3] 另同组的另三个信号共同给出读取发送、接收缓冲区的地址值

40与点阵下方的开关默认连接	Address[2]
41与点阵下方的开关默认连接	Address[1]
44与点阵下方的开关默认连接	Address[0]
68 与点阵默认连接	Display_data[7] 另同组的另七个信号共同给出读取发送、接收缓冲区的数值
69与点阵默认连接	Display_data[6]
70与点阵默认连接	Display_data[5]
71与点阵默认连接	Display_data[4]
73与点阵默认连接	Display_data[3]
74与点阵默认连接	Display_data[2]
75与点阵默认连接	Display_data[1]
83与点阵默认连接	Display_data[0]
79 接一晶振输出	Clk 系统时钟
85 接一上升沿触发的按键	Rst 复位信号
86接一上升沿触发的按键	Tx_request_in 发送请求
87 接面包板上的rx端	Rx 从总线读取的数值
88接面包板上的tx端	Tx 打到总线上的数值
89接一上升沿触发的按键	Mode_clear 退出复位状态
90 接一LED	Ack 接收方需发送ACK
94	
95	
96	
97	
127 与七段灯默认连接	Quant_cnt[0] 与同组另三个信号给出位周期状态
128与七段灯默认连接	Quant_cnt[1]
131与七段灯默认连接	Quant_cnt[2]
132与七段灯默认连接	Quant_cnt[3]
133	Frame_cnt[0]
134	Frame_cnt[1]
135	Frame_cnt[2]
136	Frame_cnt[3]
139	Bit_cnt[0]
140	Bit_cnt[1]
141	Bit_cnt[2]
142	Bit_cnt[3]
143	Bit_cnt[4]

144	Bit_cnt[5]
170	Default_row[0]与同组另七个信号给出 点阵上显示行的列坐标
172	Default_row[1]
173	Default_row[2]
174	Default_row[3]
175	Default_row[4]
176	Default_row[5]
177	Default_row[6]
179	Default_row[7]
187接一LED灯	Stuff_err 位填充错误
189接一LED灯	Crc_err CRC错误
190接一LED灯	Bit_err 位错误
191接一LED灯	Ack_err 应答错误
192接一LED灯	Form_err 格式错误
193接一上升沿触发的按键	Release_buffer 释放接收缓冲区

4.1.4 CAN核驱动器82C251的连接方法

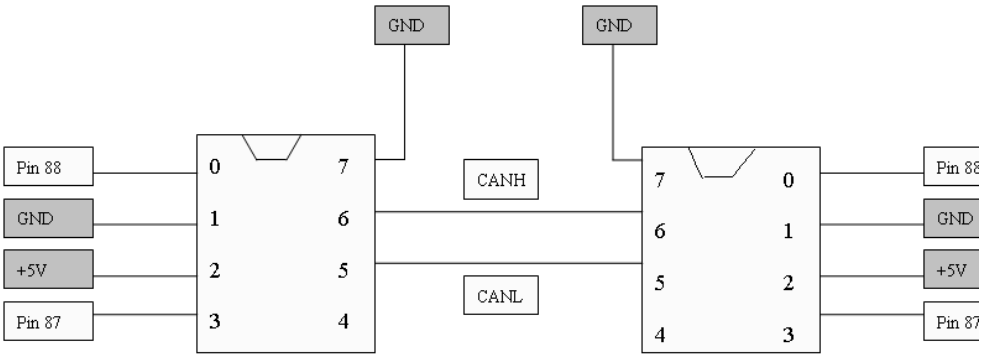


图4.3CAN核驱动器82C251的连接方法

CAN软核需要与一个驱动芯片连接进行逻辑电路到模拟电路的转换，85C251这枚芯片就是起到这样的功能。如图所示，每一个芯片的0端接发送信号tx,1端接地，2端接电源，3端接rx,4端空闲，5端、6端分别接双绞线的低压、高压线，7端接地。在双绞线两端和接一个124欧姆的电阻，并联构成64欧姆的阻抗匹配。

4.2 实验步骤及测试现象

因为本组只是对软核进行测试，所以发送的数据都是事先打入好的，并且略去了对寄存器读入读出的测试，命令寄存器的值直接由实验台上按键给出。

4.2.1 正常传输状态下实验现象及步骤

(1)将程序分别下载到两个实验台上。确定一方为发送方，一方为接收方。
(2)两台分别按下“系统复位键”、 “退出复位状态键”。此时双方同时进入操作状态，双方的状态机都开始运行。可通过七段显示灯看出双方的位周期状态，此时由于没有进行同步，两个状态完全不一致。此时可通过发送方的点阵显示并配合开关读出发送缓冲区中的数据。双方的发送灯和接收灯持续亮(表示发送接收都为1)。
(3)发送方按下“发送请求”命令，进入发送状态。发送方的发送灯与接收灯根据发送的值亮、灭。可根据发送方的位周期状态灯清楚观察到当位周期每进入最后一位时(发送点位置)，发送灯会发生变化。接收方的发送灯一直亮，而接收灯则随发送方的发送值变化。当接收方检测到硬同步时，向发送方的位周期状态机同步，此时可观察到两者已经同步。
(4)发送方与接收方的发送/接收状态机同步变化，表明双方同时进入相应的状态(如ID场、DLC场、DATA域、CRC场等)
(5)当双方同时进入ACK场时，接收方的发送灯变(发出ACK应答)，发送方发送灯亮，双方的接收灯同时灭，这表明应答位接收成功。
(6)经过8个位周期后(1位ACK_DELIMITER,7位帧结尾)，接收方的接收缓冲区指示灯亮，表明接收到数据。
(7)使用开关与点阵配合，对应发送方点阵(发送数据)与接收方点阵(接收数据)按位对照，发现一致。至此，发送、接收数据成功。
换用不同组的频率从1Hz至300KHz，重复上述步骤，都获成功。

4.2.2 错误状态下的实验步骤与现象

由于实验台上提供的晶振有限，且不同级别晶振之间相差倍数过大，使得在软件测试中，模拟两个有一定误差但误差不大的时钟来产生错误的这种方法在硬件中无法采用。在摸索了使用电位信号干扰等方法后，发现251芯片的抗干扰能力非常强，这使得常规方法无法造成错误传输，所以只好采取在传输过程中拔线的方法。为了能观测到五种错误，特别设计了如下一套方案(要将双方的系统时钟调慢，否则有一定的实验难度)：
(1)在发送数据场时，拔掉发送方的接收线，由于此时发送方发送的数据与接收的数据不一致，产生了位错误(可观察到发送方的位错误灯亮)，此时发送方开始发送错误帧(在此过程中需将发送方的接收线重新插上)，可观察到发送方的发送灯持续灭一段时间(此时正在发送错误帧的错误标志位，由于其是主动错误型，此时发送0)。接收方由于接收到超过五个0，产生位填充错误(可观察到接收方的位填充错误灯亮)。此时接收方也开始发送错误帧，可见接收方的发送灯持续灭一段时间。当双方停止发送错误帧后，发送方检测到总线空闲后，又继续发送。
(2)在这次发送过程中，等到进入CRC场后，拔掉接收方的接收线，过两个位周期后，再将

其插上。这样到CRC场结束后，接收方会检测到CRC错误(可观察接收方的CRC错误指示灯)，但根据协议规定此时不发送错误帧。当进入ACK场时，拔掉发送方的接收线，此时发送方因未检测到总线上的“0”，而产生ACK错(可观察发送方的ACK错误指示灯)，此时发送方开始发送错误帧。接收方此时进入接收ACK_DELIMITER状态，但由于接收到的是发送方发的错误帧，所以接收到的是“0”而不是规定的“1”，于是发生格式错(可观察接收方的格式错误指示灯)，于是也开始发送错误帧。至此，五种错误全部产生。

(3)当错误帧发送完成后，双方的错误指示灯均熄灭。发送方待检测到总线空闲后，又进入发送状态。

在整体测试中，并没有遇见太大的麻烦，这是因为之前在ModelSim下进行了一定量的模拟，所以下载调试过程比较顺利，唯一有麻烦的地方是当把系统频率调慢后(降到1Hz以下)，按键速度不能过快，否则实验台将因还未进入到一个时钟周期，而检测不到输入量的变化。

4.3 本组另一同学测试的结果

本组专门负责测试的同学在测试过程中发现，当发送方发送过程中出现错误，接收方此时接收到一发送命令，当总线上错误帧结束后，这个发送命令不能正确进行。后经排查，发现是在发送方的小状态机里当发生错误时没有将状态机回归帧头状态等待发送这一错误造成的。**这是个严重的错误。**将此错误排除后，发送方的功能到目前为止未发现有异常。

4.4 其它组使用本组软核的情况

上述都是测试模块，当测试完成后，需要向使用本组模块的同学提供顶层模块。即符合Avalon总线的模块。模块编译后产生的框图如下：

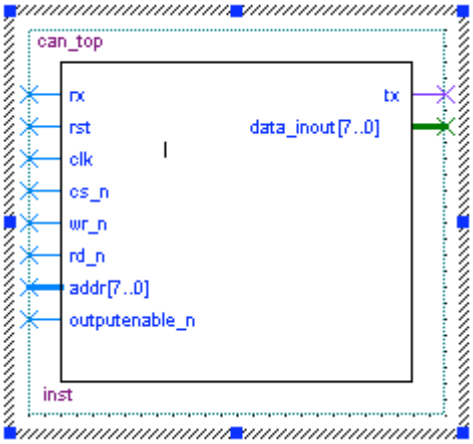


图4.4对外提供的CAN核模块图

需要本组提供此模块的组共有四组，分别是编码组、ECU数据传输组、行驶仪记录组

和中央CPU组。其中，行驶仪记录组在进行测试时发现数据总无法正常发送，但是能正常发错误帧，经排查，发现是因为使用的是两个不同的实验台及FPGA板，在跨平台实验时，需设定相同的参考点，即需要把两块板子共地。经上述操作后，行驶仪记录组可以正常使用CAN_TOP模块。另外，中央CPU组发现的问题是Avalon总线无法正确地向CAN核内选址，后经翻阅Avalon总线相关书籍，发现这里涉及到Avalon总线32位地址线与CAN核8位地址线的地址对齐问题，需要针对Avalon总线在CAN核内部对地址线除以四，但对零的情况下不能做相应的右移四位的操作，否则会出错。而其他组使用PIO总线对CAN核操作时未发现相同问题。到目前为止，本组提供的CAN核经受住了考验，没有发现错误。

5. 系统存在的不足与改进方向

由于时间有限，对一些计数器的清零的使能设计得并不严格，虽然并不影响功能的正常进行，但是也需要今后注意。发送错误计数器由于宽度是9位而CAN核的数据线是8位，每次只能读出低8位，同时还需将节点是否为“总线关闭”状态读出，来判断发送错误计数器的最高位是否为“1”。

6. 收获与致谢

6.1收获

虽然我的研究生方向是网络安全，但在上学期就选择做这个毕设课题的原因是本人觉得在工大学习计算机科学这三年多的时间里，对编程、算法之类的东西学习了不少，也实践了不少；但对硬件方面涉足不深。而一个真正的计算机人才应该是对计算方面软硬兼修。学习好硬件知识，了解计算机内部是如何工作的，这对更深入了解编程是会有好处的。除了这个原因，本人还想好好补习一下通信与网络的知识，因为本人觉得大三学习的这两门课程并没有学得很明白，主要是没有实际动手的机会。经过这几个月的学习与实践，现在想想，这两个主要目的都实现了。特在此将收获总结如下：

(1) 消除了对硬件的恐惧感。由于之前没有很多的机会接触与硬件相关的东西，心中对硬件方面有一种恐惧感，虽然在相关的课程取得的成绩不比在软件编程取得的成绩差，但是仍然认为自己对这方面了解很有限。但是，当本人在实验室里亲手做了这些实验，天天使用着这些JTAG口、不同的FPGA板、网卡驱动芯片；甚至是在帮助老师捡拾东西时，第一次亲眼看见那些不同规格的芯片、CPU、E^2PROM这些以前只闻其名未见其物的东西,本人对硬件的感觉立刻觉得拉近了。尤其是当我们的程序在实验台上跑起来了，突然觉得这就是我们自己制造的一块网卡，只不过它是符合CAN协议的而不是TCP/IP

协议的，心中非常高兴。记得大三时做过一个用Maxplus实现的小型CPU的实验，当时对这些东西还有距离感，现在想想发现那其实是很简单的设计时序的问题。本人将在研究生第一学期学习CPU设计和数字系统设计，相信这次毕设为那时的学习会打下很好的基础。

(2) 加深了对通信的了解，现在本人对CAN协议各种帧的格式定义、位时序定义、出错处理、纠错、位填充这些细节理解的比较深入。其实这和其它通信协议也有很多类似的东西，这样就使我对通信的底层机制有了更多的了解。同时也明白了通信协议以时序为驱动这一原理。深深感受到了许老师之前说过的“通信里的时序控制是最复杂的”这句话的道理。并且本人发现，通信原理的影子在很多地方都有体现—比如操作系统里的同步互斥问题。同步对应着通信协议里的同步，比如CAN协议中的硬同步，重同步。互斥则对应着CAN协议中的总线仲裁，所谓的临界资源就是对总线的控制权。而通信与网络更是不分家，比如CAN协议中的ACK应答位这种应答的技术就对应着TCP/IP里三次握手里的应答帧。只不过CAN里的ACK是用总线上的一位表示，让接收方来添补这一位，而TCP/IP里是通过发送一帧来实现的。并且本人觉得计算机内部所有的信号也都是靠一个主频信号控制的，这与协议里的系统时钟的概念是非常一致的。所有的硬件设计(除了一些非常简单的译码电路等组合逻辑)在本质上都是如果对控制信号进行分频，然后用这些子信号来控制相应的事件发生，产生相应的状态转换。而这本身就是图灵机的思想—对一组输入，根据状态转换函数，进行相应的运算。这同时也是编译的原理。Dennis Riche曾经说过，C语言的编程实际上就是对输入的字符串的解析。这同样与硬件设计在本质上是相同的东西。本人觉得自从学习了硬件设计这些东西之后，对计算机科学又有了新的认识。

(3) 深刻地体会到了硬件设计的简洁美以及强大的功能。8根地址线、8根数据线、1根读信号线、1根写信号线、1根片选线、1根发送线、1根接收线，1根CLK，1根RST，如此而已，但是通过我们的设计，对这些输入信号进行配合完成了一个复杂的通信功能。我们设计的通信芯片“麻雀虽小，五脏俱全”。这是非常令人称奇的。

(4) 了解了IP复用技术，使用了OpenCore资源。当本人看到我们的软核被其它组用作通信基础时，立刻感觉到IP利用技术的威力—随时编写，随用随下载，非常方便。同时，本人还发现了一个非常棒的网站—www.opencores.org，上面有各种各样的用VHDL/Verilog HDL编写的软核。其中就有CAN协议的软核、USB协议的软核、小型RISC-CPU的软核。研究软核是一件非常有乐趣的事情，可以帮助大家对这些技术底层细节有所理解，就像读Linux源码一样。特别是被本人视为“圣经”的Open CAN软核，它给本人带来的影响是巨大的。很难想象没有这个软核，本组会顺利完成这次毕设。它不仅让我们对协议的了解大大加深，因为只读协议对我们这些丝毫没有这方面经验的学生来说是绝不可能用硬件描述方言，这门从未学过的语言，完成其硬件实现的。更重要的是，它为我们提供了活生生的教科书，上面简明的设计思想、标准的设计格式、严谨的语言风格无一不对我们起到了巨大的教育作用。

(5) 加深了对硬件领域的了解。在上网查资料的时候，本人惊奇地发现我国自主研发的“龙芯”就是使用Verilog语言编写并下载到FPGA上。当然他们做的项目很大，我们做的项

目很小；但本人觉得至少自己学习的技术是大有用处的。在自己的芯片上跑自己的操作系统，一直是我国广大计算机科研人员的梦想，我非常希望我能在这方面做出自己的一点贡献。而这次毕设也使我在自主研发芯片这方面迈出了第一步。

(6) 这次毕设的工作量是很大的，本人第一次因为做课设、写论文熬夜。这次毕设也帮助我加强了自身对生物钟的调节。但是，和使自己的程序能运行，自己的软核能被别人正常地使用所获得的快乐相比，这些就都不算什么了。这次毕设中，本人还更好地学会了团队合作，以更快的速度和更高的效率完成了毕设。

6.2致谢

在这次毕设中，有许多人提供了帮助，在这里表示感谢：

首先，感谢我的父母，为我的生活提供了如此多的便利与支持，使我能够安心学习。特别是从三月中到四月底这个“攻坚”阶段，我一个多月没有回家，他们虽然很想念我，但依然给我极大的支持和鼓励，使我能够完成毕设。

其次，感谢OpenCAN的作者Igor Mohor，他编写的程序对我无论在语言学习、在对CAN协议的理解还是对系统设计都起到了极其重要的启蒙作用。更重要的是，他无私地与大家分享了这一成果。

然后，要感谢指导老师许向众老师。许老师是我数字逻辑的启蒙老师，他丰富的教学经验及任劳任怨的教学作风给我打下了良好的数字逻辑的基础，使我能较快地掌握数字设计的方法。在毕设环节中，许老师丰富的工程经验、精确的大局观，为我指明了道路，纠正了设计上的错误，解决了实际问题；尤其是许老师和蔼亲切的作风使大学能够团结得很好，顺利地完成任务。

再次，要感谢本组的彭建朝老师。彭老师以老黄牛的工作态度给大家树立了榜样，并同大家一起解决了许多实际上的工程问题。还要感谢孙丽君老师当年教授我微机系统接口这门课，使我在进行设计时能很快调通软核与Avalon总线的时序接口；以及感谢贾熹滨老师认真帮我修改英文翻译。

最后，但绝不是最少的，要特别感谢我的合作伙伴任哲同学，任哲同学用他踏实肯干、一丝不苟的学风给我树立了非常好的榜样。我到现在耳边似乎还回荡着他“快点、再快点，咱们可不能拖别人组进度”的催促声。同时，还要感谢为我们做硬件测试的李择一同学，以及在工作中及时给我们反馈的李毅、宋洋两位同学。

附：参考文献

1. CAN协议 Bosche 公司
2. OpenCAN Igor Mohor www.opencores.org
3. Verilog HDL高级数字设计 Michael D.Clietti Pearson Prentice Hall&电子工业出版社
4. Verilog 数字系统设计教程 夏宇闻 北京航空航天大学出版社
5. 现场总线CAN原理与应用技术 饶运涛等北京航空航天大学出版社
6. 数字通信—传输原理及应用 欧阳长月北京航空航天大学出版社
7. Computer Networks Fourth Edition A.S.Tanenbaum Prentice Hall&清华大学出版社
8. 挑战SOC—基于NIOS的SOPC设计与实践 周博等 清华大学出版社