一种避免饥饿的CAN总线优先级反转方案研究

摘要：在CAN总线中，当两个或更多节点同时开始发送时，存在基于优先级的仲裁方案来决定哪个节点将被授予继续发送的许可。在总线仲裁期间，每一个发送节点时刻监控总线状态，同时将接收到的位与发送的位进行比较，如果在发送隐性位1时接收到了显性位0，则该节点会停止发送。因此，在最高优先级节点上的所有消息都已发出之前，它将始终占用总线。在本研究中，提出了一种优先级反转方案，能够在一定程度上改善优先级高的节点持续占用总线的问题。运行结果表明,当有高优先级的报文持续不断发送时，未使用优先级反转方案的CAN总线低优先级在其周期内不能成功发送,而使用优先级反转方案之后,系统可正常运行,证明了算法有效性。

关键词：优先级反转；CAN总线；优先级队列；避免饥饿

0引言

CAN总线是用于连接ECU的多主机广播串行总线标准，每个节点都能够发送和接收消息，但不能同时发送和接收消息[1]。在CAN总线中，来自多个设备的较高优先级消息会阻塞低优先级消息占用CAN总线进行传输。近年来，已经有研究使用一些方案来解决CAN总线阻塞问题。文献[2]中，Murtaza和Khan提出在CAN总线上增加一个主节点，以防止其他低优先级节点出现阻塞问题。主节点必须尝试检测阻塞节点，并确保阻塞节点保持参与通信。主节点始终监视CAN网络，并在收到来自所有节点的消息时保持沉默。如果主节点发现来自某些节点的消息没有参与通信，则该主节点将变为活动状态，这确保了来自这些节点的消息可以在下一轮仲裁中进行访问以进行广播。虽然该方案可以防止启动问题，但由于需要额外的节点支持，因此该方案需要更高的负担。文献[3]中,提出基于使用优先级和FIFO队列的CAN总线消息的调度研究。文献[4]中，开发了一种使用无进程饥饿的方案，并使用了优先级反转来完成。文献[5]中，Haklin Kimm等人提出了一种具有优先级队列的避免进程饥饿CAN总线调度方案。但是这些工作大多过分重视将优先级低的任务在CAN总线仲裁中动态变为高优先级，而本文则是更多的考虑将优先级高的任务在一定条件下进行优先级反转。

第1节介绍了CAN总线帧格式和仲裁方法，第2节提出并实现了优先级反转避免阻塞的仲裁算法，第3节进行实验仿真验证，第4节为结语。

1.CAN总线帧格式和仲裁方式

自从德国Bosch公司在汽车工程师协会(SAE)大会上引入串行总线系统以来，CAN总线已经成为最成功的数据通信协议之一[]。在CAN总线当中有四种不同的报文帧，其中数据帧能够将数据从发生端广播至接收端。数据帧格式如图1所示

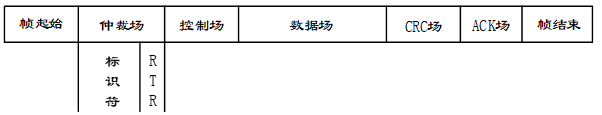


图1 数据帧格式

数据帧分为两种格式，标准格式以及扩展格式。其中标准格式的标识符长度为11位，而扩展格式标识符具有29位。本文以标准格式为进行论述，对数据帧中仲裁场里面的标识符进行进一步的分析研究。

CAN总线仲裁方式可以理解为，如果有两个或多个节点同时传输，且有显性位0和隐形位1同时发送，规定显性0状态的权重大于隐形1状态，故显性位0可以覆盖隐形位1，因此CAN总线将处于显性位0。该机制用于在比较标识符字段时控制对总线的访问。如果两个或多个节点同时开始传输，那么通过监控总线上的每个位，每个节点可以确定它当前正在传输的消息优先级高低，若在仲裁中某一节点胜出，那么该节点继续传输其消息的剩余部分，此时其他所有节点均退出，等待下一次仲裁开始。因为CAN总线没有全局时，所以需要节点在每次消息传输时重新同步，而且每个节点必须与首先开始传输的节点引起的帧起始上升沿同步，以便总线仲裁正常工作。

2 优先级反转算法设计

2.1 设计

在本文中，11位消息标识符由1位动态反转位、1位级别位和9位静态位组成，如图2消息标识符ID划分所示。当然这仅仅是在标准格式中的划分，除此之外也可以应用在扩展格式中。静态标识位（ID0-ID8）从最低的111111111开始，以最高的0位000000000结束。级别标识位ID9用来区分高优先级组和低优先级组，两组分别含有10个报文，其中高优先级组的级别标示位为0，低优先级组的级别标识位为1。动态反转位ID10默认为0未反转，其中动态反转位在本文中的用法如下：当高优先级组在一定的时间内出现某一报文连续发送的次数大于设置的固定值MAX，则在下一次其继续竞争总线之前，将其动态反转位进行反转由0变为1,从而使其优先级降低，进而能够使其不能持续占有总线，同时使被阻塞的报文得到机会发送。有了1位动态反转标识位，任何想要持续占用总线的消息，在持续占用一定次数后都会被中断。每个11位都以0开头，级别标识位可在0与1之间进行划分，静态标识位内数字任意。

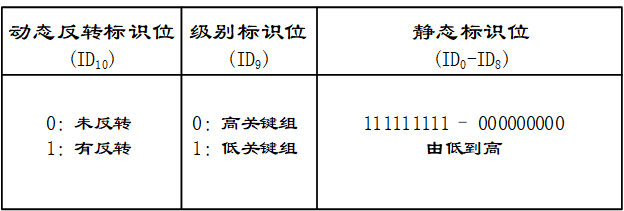


图2 消息标识符ID划分

2.2 流程图

图3是优先级反转算法的流程图，算法运行后，初始默认全都发送。整个流程需要进行三次判断分别是：

（1）判断高优先级组和低优先级组是否有报文到达发送时间；

（2）如果是高优先级组有报文到达，还需要判断是否触发了反转条件；

（3）如果触发了反转条件，还需要在判断是否反转后的ID能够发送。

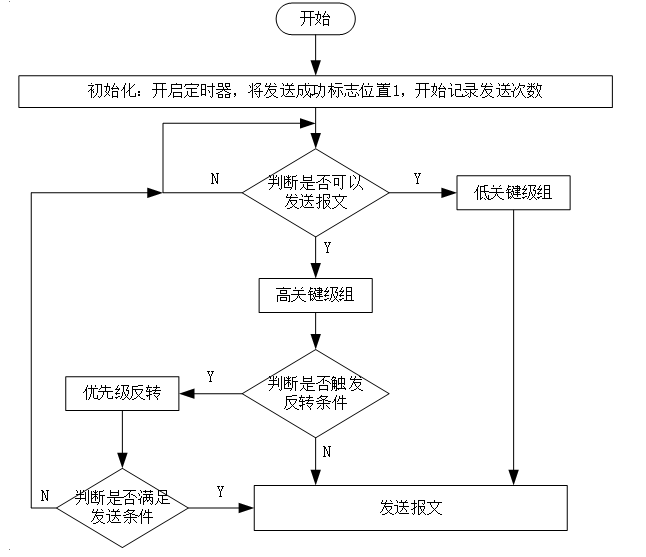


图3 优先级反转算法流程

2.3 模型

模型如图4所示，在本模型中，进行了如下定义：

定义1：CAN总线利用率是Ki/Ti的和，其中K为传输时间，T为传输周期，i = 1，2，… n。

定义2：报文的响应时间定义为报文发送完成时间和报文激活时间之差，即当报文在时间t1被激活并在时间t2完成时，那么t2-t1称为报文的响应时间。

定义3：平均响应时间是指所有报文响应时间之和与发送不同ID报文数量N之比。

定义4：到达率指的是一定时间内实际发送某一报文的数量R与该报文预计发送的数量P之比。

定义5：平均到达率是Ri/Pi的和与发送不同ID报文数量N之比，i = 1，2，… n。

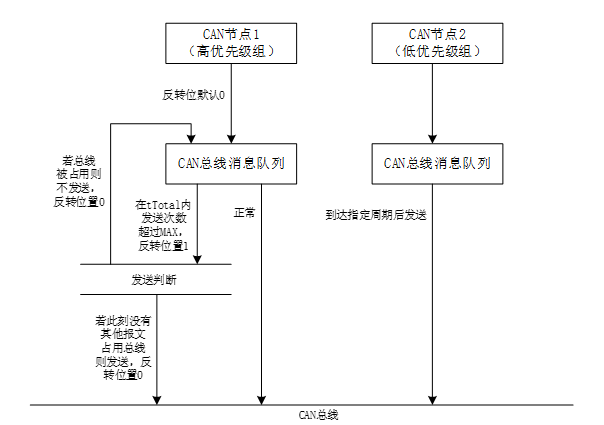


图4 优先级反转模型

3 实验仿真

在本节中，我们使用CANoe平台来评估优先级反转方案的性能，其中本实验所需的一些默认参数如表1所示，我们设置了两组周期消息队列，分别为高优先级组和低优先级组，每组均有10个报文。

表1 优先级反转方案中默认参数

|  |  |  |
| --- | --- | --- |
| 参数 | 说明 | 值 |
| tTotal | 在此时间内检测发送次数 | 4ms |
| tArrive | 所有报文在程序开始时第一次发送时间 | 0 |
| tSend | 计算低优先级组预计发送数量的时间 | 1s |
| n | 在tSend时间下低优先级组预计发送的数量 | 404 |
| Max | 在Total\_time里面发送超过MAX次数则反转 | 3 |
| N | 不同ID的报文数量 | 20 |

默认在0时刻发送所有的报文到总线上，当在tTotal时间内，若高优先级组中某一报文发送的次数大于等于MAX则会触发动态反转标志位，使其反转到0。具体算法如图4所示。

如图5所示，当CAN总线利用率变化时，低优先级组的平均响应时间也随之变化，但是有反转的平均响应时间总是小于未反转的平均响应时间。

在本实验中，若CAN总线的利用率小于100时，有无反转对低优先级组的到达时间并无显著影响，但CAN总线的利用率大于等于100时，如图6所示，此时CAN总线处于阻塞状态，必然存在着一定数量的消息一直处于饥饿。当CAN总线的利用率逐渐增加时，未进行反转的低优先级组的平均到达率持续减少，直到为0，此时低优先级组没有一个消息能够发送到CAN总线中，而有反转的低优先级组的平均到达率则很稳定，低优先级组能够正常得到发送。

4 结论

本文针对CAN总线中存在的饥饿问题，建立了一个优先级反转模型，与以往不同的是，文中的模型在满足优先级反转条件下，会使高优先级反转为低优先级。通过进行试验仿真，在不同的CAN总线利用率下，该模型能够通过改善CAN总线中周期任务的平均响应时间来减少饥饿。当CAN总线利用率超过100时，有反转的低关键级组平均到达率也能够保持正常。故本文提出的算法在避免饥饿方面是有效的。