第 5 章“整装，待发”

设备内部多条指令的发送顺序

5.1 “整装待发 ”

一般 CAN 通讯协议只规范了多个设备在总线上同时发送指令时的竞争顺序，而对于单个设备内部产生的多条指令的发送顺序却很少涉及。

在纯主从模式下，设备内部产生多条指令的可能性并不大，相关问题很难暴露，而当系统发展演绎到全触发方式后情况就不同了，一个设备可能具有多个触发变量，它们可以是周期触发、变化触发、越界触发等等，而总线的发送速率是有限的，且同时还存在其他设备对总线的竞争，因此在同一设备中就有可能积累多条指令等待发送。

通常设备会默认以“先生优势”规则处理这些指令，即先产生的指令优先发送，这样处理的最大好处是指令动作先后逻辑规范，这对那些具有先后关联的系列指令尤为重要，但事物总是两方面的，这种“先生优势”规则，会延迟和阻碍后产生的高优先级指令的发送，如告警指令，尤其是当总线上存在竞争时，情况尤为严重，如：当设备指令发送端口被一条先生成的低优先级指令占据时，其在总线上的竞争处于劣势，从而使该设备中后产生的告警指令无法参与竞争而被推迟发送，这种延迟有时可能是致命的。

因此我们需要重新规范设备内部多条指令的发送规则，最粗暴简单的方法就是对这多条指令进行优先级排队，将 ID 字小的指令排在前面，每当设备产生一条新指令时都要重新进行一次指令排序（注意：这种重新排序应当包括那条已经处在发送端口的未发指令），这样设备便具有了最优的发送竞争力，但同时可能会破坏前面所说的某些指令的先后逻辑关联，可能使系统运转出现偏差。

那么，如何才能解决上述矛盾呢？TTCANopen 协议提供了相关的解决途径，这也是 TTCANopen 协议更为贴心之所在。

5.2 TC0061（设备内部多条指令的发送规则）

当一设备同时具有多条指令等待发送，其发送原则如下：

1）优先级位为 0 的指令优先于优先级位为 1 的指令发送；

2）优先级位相同的指令按其产生的先后顺序发送。

虽然 TC0061 只有短短的两条规定，但它却完成了其它协议很难规范完成的内容，而这对于 TTCANopen 协议来说却是如此的自然天成，它的第一条规定确保了低优先级（1）的指令不能阻碍高优先级（0）的发送，使设备中诸如告警指令得以顺利抢先通过总线发送；第二条规定保障了具有先后逻辑关系的系列指令的顺利完成，因为我们一般不会将这些相关指令定义在不同的优先级位上。

5.3 温柔一刀

正当我们暗自为 TC0061 子协议欣喜的时候，接下来的测试却给了我们当头一棒。

在设备内部我们分别为优先级位为 0 和 1 的指令建立了两个发送循环缓冲区，每个缓冲区都能容纳多个发送指令，缓冲区中的指令遵循先入先出的规则。在指令发送时，首先取优先级位为 0 的缓冲区中的指令，当该缓冲区中的全部指令顺利发送后，再取优先级位为 1 的缓冲区中的指令，如果优先级位为 0 的缓冲区中再次装载了高优先级的指令，则停止发送优先级位为 1 的缓冲区中的指令，转而发送优先级位为 0 的缓冲区中的指令。看似简单的发送逻辑过程，在测试中却出了问题，我们使用的某厂商的 ARM 芯片（带 CAN 接口）时，当我们将一个低优先级的指令从发送循环缓冲区取出放入芯片的发送寄存器中并使能发送后，此时出现高优先级指令，需停止或终止刚才那条低优先级指令的发送，当我们发出终止命令后，芯片端口产生了“死机”，低优先级指令的发送使能和终止命令相距越近死机的概率就越高。为此我们花费了整整几个月的时间，也上网求助过，终不能解决问题，我们甚至有放弃实施 TC0061 协议的想法。

也就在此时，邓先生提出了一个“曲线救国”的方法，邓先生认为那个先放入发送寄存器中的低优先级指令使能后，要么开始发送，要么因总线上有其它设备的指令而被堵塞，因此可以利用设备的 CAN 接收中断程序，在芯片的接收中断服务程序中读取发送端口的状态，并作标记，且在接收中断服务程序结束时，发出该终止发送命令。邓先生的方法虽然解决了芯片端口死机问题，但也使得在发生总线竞争堵塞时，高优先级指令获得发送的机会可能会延迟了一个 CAN 指令传输时间。

5.4 讨论与提示

5.4.1 更广泛的测试

目前，许多的单片机都集成有 CAN 接口，如 ST 公司的 TMS32 系列 ARM 单片机等，为了实现对 TC0061 子协议的支持，我们需要对更加广泛的 CAN 接口进行测试，尤其是对其终止发送命令的测试，俗话说众人拾柴火焰高，这个任务就交给广大感兴趣的读者来完成吧。让我们将更多的精力放在研发 TTCANopen 协议本身，谢谢大家！

5.4.2 USB 转 CAN 模块

由于 USB 端口通讯速率远大于 CAN 端口，因此在 USB 转 CAN 模块中就会有多条指令等待发送，正如本章所述，作为应用于 TTCANopen 协议的 USB 转 CAN 模块，理所应当支持 TC0061 子协议以获得最佳的系统性能，而目前市面上的通用转换模块是无法支持的，这也是我们为什么要研发 TTCANopen 专用转换模块的重要原因之一。在本书稍后章节我们将介绍 TTCANopen 专用转换模块的研发，其中不再累述对 TC0061 子协议支持的相关内容。

5.4.3 高优先级指令的比例

一个系统高优先级（0）指令如果过多，其优先的特性就将荡然无存，我们仅仅只需将那些急迫的，极为重要的信息用高优先级传输，这一比例我们认为应当小于百分之一，即使在高优先级指令集中爆发时，这一比例也应当小于十分之一。当然读者可能会有不同的见解，认为我们给出的这一比例过于武断，这完全是可以仁者见仁，智者见智的，需根据具体应用而定夺。

5.4.4 画蛇添足

通常我们会认为设备中单片机对一条指令的处理速度远高于其在总线上的传输速度，这样在纯主从模式下，设备在 CAN 的接收和发送端口都不必设置接收和发送指令的缓冲区，如图 5－1 所示。

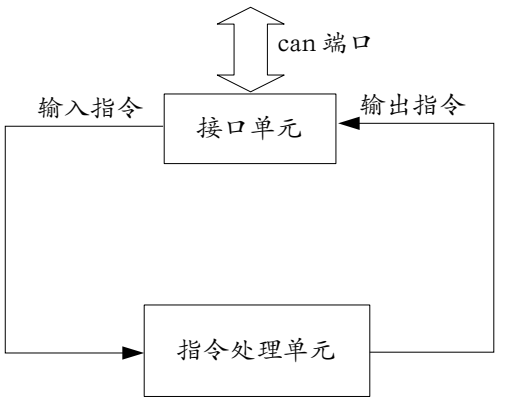

图 5－1 纯主从模式下设备 CAN 总线的输入输出端口

当设备演绎到全触发状态后，正如我们本章所述，设备的多个触发源可能同时触发多条指令等待发送，这样必须在设备的发送端口设置两个优先级不同的发送循环缓冲区，如图 5－2。

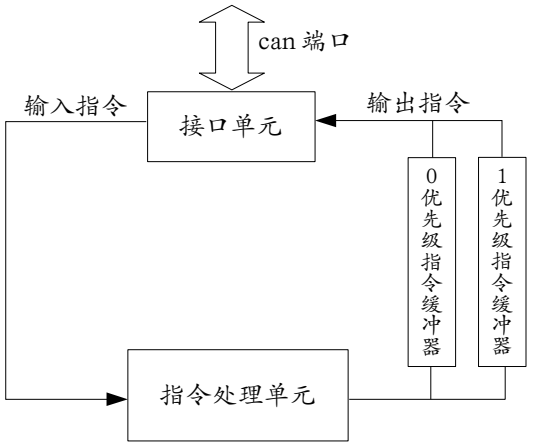

图 5－2 全触发模式下设备 CAN 总线的输入输出端口

当我把发送缓冲区画上的时候，总有一种莫名的冲动，想在接收端也画上两个缓冲区，这样看上去很对称，很美，见图 5－3。

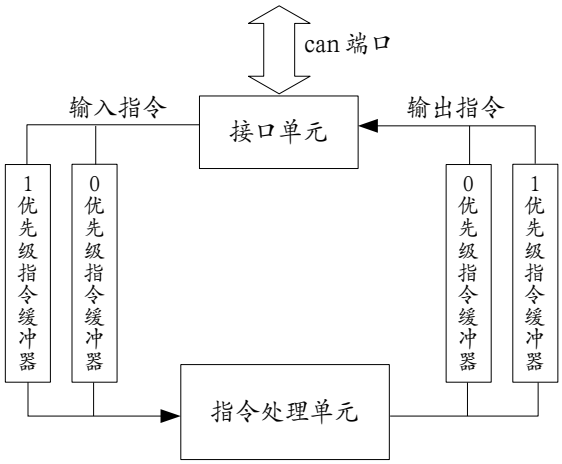

图 5－3 画蛇添足后设备 CAN 总线的输入输出端口

中国有一个成语叫做“画蛇添足”，为了这点美学上的感受，我几乎是在搜肠刮肚，为输入缓冲区的存在寻找理由。而这“理由”终于被我搜刮到了，我们知道设备自身触发的指令有两种，一种是以 0x0n 指令格式发送的，这些指令我们直接将它们送至发送缓冲区就好；而另一种指令是以 0x1n 指令格式发送的，对于这种指令我们可以将其看做是对主机 0x0n 指令的响应，如：我们需要使用 0x19 指令上传数据时，可以将其看做是对 0x09 数据读取指令的响应，这样我们就可以在设备上臆造一条 0x09 指令，并将它放入到输入缓冲区中，就好像是从总线上接收到的一样，由指令处理模块对其进行处理，并生成一个 0x19 应答指令发送到输出缓冲区。而对于指令处理模块而言，仿佛接收到的就是主机发送的 0x09 轮询数据采集指令，只是这条指令没有真正在总线上传输，而是设备自身产生的。由于这个自身产生的指令的存在，它和输入指令一起就可能会出现多条指令并存的情况，需要建立一个输入指令缓冲区。

其实，建立输入输出缓冲区可以有效的均衡设备处理速度与传输速度的差异，使设备的适用性更加广泛，让系统运转地更加稳定可靠，这才是本质的原因。