嵌入式系统开发中五个设计驱动程序的方法

一个嵌入式应用软件都会在某些时候访问最底层的固件和进行一些硬件控制。驱动的设计和实施是确保一个系统能够满足其实时性要求的关键。以下 5 个窍门是每一个开发者在设计驱动程序时应该考虑的，下面就随我们一起来了解一下相关内容吧。

1.使用设计模式

设计模式是一个用来处理那些在软件中会重复出现的问题的解决方案。开发人员可以选择浪费宝贵的时间和预算从无到有地重新发明一个解决方案，也可以从他的解决方案工具箱中选择一个最适合解决这个问题的方案。在微处理器出现之初，底层驱动已经很成熟了，那么，为什么不利用现有的成熟的解决方案呢？

驱动程序设计模式大致分属以下 4 个类别：Bit bang、轮询、中断驱动和直接存储器访问(DMA)。

Bit bang 模式：当微控制器没有内外设去执行功能的时候，或者当所有的内外设都已经被使用了，而此时又有一个新的请求，那么开发者就应该选择 Bit bang 设计模式。Bit bang 模式的解决方案很有效率，但通常需要大量的软件开销来确保其实施的能力。Bit bang 模式可以让开发者手动完成通信协议或外部行为。

轮询模式用于简单地监视一个轮询调度方式中的事件。轮询模式适用于非常简单的系统，但许多现代应用程序都需要中断。

中断可以让开发者在事件发生时进行处理，而不用等代码手动检查。

DMA(直接存储器访问)模式允许其它外围设备来处理数据传输的需求，而不需要驱动的干预。

2.了解实时行为

一个实时系统是否能满足实时需求取决于它的驱动程序。写入能力差的驱动是低效的，并可能使不知情的开发者放弃系统的性能。设计者需要考虑驱动的两个特点：阻塞和非阻塞。一个阻塞的驱动程序在其完成工作之前会阻止其他任何软件执行操作。例如，一个 USART 驱动程序可以把一个字符装入传输缓冲区，然后一直等到接收到传输结束标志符才继续执行下一步操作。

另一方面，非阻塞驱动则是一般利用中断来实现它的功能。中断的使用可以防止驱动程序在等待一个事件发生时拦截其他软件的执行操作。USART 的驱动程序可以将一个字符装入传输缓冲区然后等主程序发布下一个指令。传输结束标志符的设置会导致中断结束，让驱动进行下一步操作。

无论哪种类型，为了保持实时性能，并防止系统中的故障，开发人员必须了解驱动的平均执行时间和最坏情况下的执行时间。一个完整的系统可能会因为一个潜在的风险而造成更大的安全问题。

3. 重用设计

在时间和预算都很紧张的情况下为什么还要再造轮子呢?在驱动程序开发中，重用、便携性和可维护性都是驱动设计的关键要求。这里面的许多特征可以通过硬件抽象层的设计和使用来说明。

硬件抽象层(HAL)为开发人员提供一种方式来创建一个标准接口去控制微控制器的外设。抽象隐藏实现细节，取而代之的是提供了可视化功能，如 Usart_Init 和 Usart_Transmit。这个方法就是让任何 USART、SPI、PWM 或其他外设具备所有微控制器都支持的共同特点。 使用 HAL 隐藏底层、特定设备的细节，让应用程序开发人员专注于应用的需求，而不是关注底层的硬件是如何工作的。同时 HAL 提供了一个重用的容器。

4.参考数据手册

微控制器在过去的几年里变得越来越复杂。以前想要完全了解一个微控制器需要掌握由一个大约包含 500 页组成的单一数据手册。而如今，一个 32 位微控制器通常包含由部分的数据手册、整个微控制器系列的资料表、每个外设数以百计的资料以及所有的勘误表组成的数据手册。 开发人员如果想要完全掌握这部分的内容需要了解几千页的文件。

不幸的是，所有这些数据手册都是一个驱动程序能真正合理实现所需要的。开发人员在一开始就要对每个数据手册中包含的信息进行收集和排序。通常它们中的每一个都需要被访问以使外设启动和运行。 关键信息被分散(或隐藏)在每种类型的数据手册中。

5.谨防外设故障

最近我刚好有机会把一系列的微控制器驱动移植到其他的微处理器上。制造商和数据手册都表明 PWM 外设在这两个系列的微控制器之间是相同的。 然而，实际情况却是在运行 PWM 驱动器的时候两者之间有很大的不同。该驱动程序只能在原来的微控制器工作，而在新系列的微控制器上却无效。

在反复翻看数据手册之后，我在数据手册中一个完全不相关的注脚里发现了 PWM 外设上电时会处于故障状态，需要将一个隐藏在寄存器中的标志位清零。在驱动程序实现的开始，确认外设可能出现的故障并查看其他看似无关的寄存器错误。