打印机驱动设计构想

背景

第 1 次做打印机驱动到现在至少有 15 年了，完成后总觉得各方面都有不足，因此有过重构的想法，怕忘记就做了记录。但由于原公司迟迟没有这方面的规划，因此最终也未能付诸实施。此文至少也是十年前的设计了，这里分享出来，仅供参考。

代码分层及设计

分层简图

应用层
接口层
软件功能模块层
驱动逻辑层
硬件接口层

硬件接口层

1）异常检测

缺纸检测：一般是 IO 高低电平的检测

温度检测：使用 ADC 进行检测，要设定软件高温的阈值

过热检测（可选）：一般是 IO 高低电平检测

除以上 3 种硬件异常状态外，可能还存在一些软件异常状态，这种异常根据具体驱动的实现而定，比如：

内存异常：内存空间不足

指令异常：接收到无法识别的 ESC/POS 指令

……

2）硬件操作

电机步进：1-2 相和 2-2 相激励两种方式，按厂家的说法，2-2 相的驱动力更大（1-2 相更平滑，噪声低），因此最好两种都实现。这样可以根据硬件设计，选择不同的激励方式。

点行加热（STB 控制）：机芯提供 6 个 STB 控制，硬件设计一般 STB 都连在一起，即在 MCU 上是使用一个 IO 口来控制加热与否。分段加热是通过软件来实现，即软件上先将数据按最大同时加热点数的要求将一点行数据先分好段，然后按段送入打印机芯进行加热。

zhuoxietang

打印电源开关：打印的时候打开，不在打印时关闭。避免软件异常引起机芯烧毁或者长时间处于上电状态，影响机芯寿命。

数据通信：如果 MCU 有提供，则直接使用 MCU 提供的总线进行通信；如果没有，则通过 IO 模块总线。一次通信是 384 个 BIT。

数据锁存：将机芯数据缓存区的内容锁存到加热缓存区中。

定时：有电机步进和点行加热两个功能点需要用到定时功能。

驱动逻辑层

1） 启动

缓存两个字符行（或一个字符行+图片；或一个字符行+条码）的点阵数据；

扫描并确定各点行的分段数：算法待定；

送入第一段数据；

缓步启动：避免第一行字符压缩

2） 打印

锁存第一段数据，送入第二段数据；

同时启动电机步进和加热；

步进时间到，设置下一步的电机驱动时序；

加热时间到：锁存第 n 段数据，送入第 n+1 段数据；

同步的，在步进和加热的过程中，

准备下一字符行的点阵数据（也可能是图片或条码）；

扫描两个字符行并确定各点行的分段数；

3） 结束

缓步停止（可选）：尽量保证时序设置与电机转轴的位置一致，使得下次启动更顺畅。软件上要记录当前的步进值，下次打印的时候要从当前的步进值继续累加。

关闭电机电源。

软件功能模块

1） ESC/POS 指令解析

解析指令，转换为对应的设置值，或将字符转换成对应点阵，并启动打印。

2） 字符点阵读取

根据字体设置和字符值，获取不同的点阵数据，将其填入待打印缓冲区。

3） 图片点阵获取

将指令的图片数据格式转换成打印机要求的格式。

kongyuetang

4） 一二维码点阵获取

将一二维码的内容，转换成一二维码的图形点阵。

对外接口层

对外接口调用，在驱动层上实际只是将其转换成对应的 ESC/POS 指令，将缓存起来。直到用户调用“启动打印”的接口后，才真正启动打印。打印过程其实是边解析 ESC/POS 指令，边走纸加热。

对于旧设计中，没有“启动打印”的对外接口，要考虑怎么兼容（应用一般在打印完会调用一次查询打印状态，可以考虑以这条指令作为是否启动打印的依据，以实现对旧应用的兼容？）。

技术点说明

各点行分段算法

分段是根据硬件提供的硬件上能够承受的最大同时加热点数，对当前点行进行数据分段。假设点行黑点数是 n，同时加热点数最大为 m，则分段数为 n/m + （n%m != 0 ? 1 ： 0）。

但如果直接按这样的分段进行处理，则存在电机需要剧烈变速的情况，这样，1）电机噪声会很大，2）如果电机的驱动力不够，则会引起失步，使得打印的字迹模糊甚至压缩或插入白行的情况。因此，需要对各行的分段数要求，采用缓步加速或缓步降速的方案，对于打印过程进行速度平滑。

缓步加速：对于步进电机的加速算法，网络上有大量的资料，可以参考。

缓步降速：可以考虑直接使用缓步加速的逆算法，这方面还没有具体了解过。

具体使用的时候，建议是在扫描阶段先把分段数和步进时间都计算好放在一个表格里头，驱动在需要的时候直接从表格里头获取值进行步进定时以及分段数据的获取。

字符行间的打印速度过渡考虑

早期的设计都是准备一个字符行打印一个字符行，因此打印变速设计只能解决一个字符行内速度变化的过渡。如果字符行之间的变速很剧烈，就无法兼顾到了。因此，本设计要求是缓冲两个字符行后再进行各点行的分段计算，以设置合理的分段，平滑字符行间的加速或减速过程。那缓冲两个字符行，打印一个字符行的同时，补充一个字符行数据，使得缓冲数据始终保持两个字符行。

减少对于硬件定时器的需求

从软件设计简单的角度考虑，使用两个定时器是最佳的。但有的 MCU 定时器资源会比较紧张，这个时候可能就只能使用一个定时器，甚至一个物理定时器，多个模块复用的情况。

benjuetang

因此，需要引入定时器复用的算法。大体的思路是：

建立一个定时器模块，模块可以接收定时请求。定时器在遍历多个定时请求后，建立定时链表，定时时间到了之后，再根据情况调用链表上对应模块的处理。具体根据软件平台及硬件方案进行详细设计，对于两个请求之间的间隔时间太短或者单个请求时间太短的情况可能就不适用于复用方案。定时器管理模块可以根据情况，将其指派给一个独立的定时器。

打印与数据准备并行

如果数据发送采用 IO 模拟的方式，那么发送的时间可能会比较长，此时将数据发送放在中断中进行就不太合适。因此，这种情况下就需要使用中断延迟处理。

对于 C8 或 C9，都已经有 DPC（即中断延迟处理）机制，在 Linux 系统上，也有类似的机制。对于电机步进可以直接在中断中完成，因为只是 IO 操作，处理很快，因此不在这里说明。而加热定时中断到了之后，具体处理则放在中断延迟处理中完成。即，在延迟处理函数中，锁存第 n 段数据，计算得到并发送第 n+1 段数据。

加热时间控制

加热时间的控制主要考虑两点，一个是当前的机芯温度，一个加热点密度。对于机芯温度，打印机模块都会提供机制供 CPU 检测当前温度，软件上可以根据机芯的当前温度对加热时间做调整。由于传导延迟，采样精度等的影响，热敏头的实际温度与采集温度可能会有比较大的偏差，因此这部分能够实现的调整精度并不高。对于加热点密度，在加热点比较集中的情况下，加热效率会很高，看起来的效果就是打印很黑，在加热点比较分散的情况下，加热效率就会很低，看起来的效果就是打印出来的内容很淡，因此针对加热点的分布，对打印进行控制也是有必要的。

1、机芯温度

这个软件上实现起来比较简单，就是根据采集到的温度对加热时间进行调节。但，如何建立这种对应关系会比较麻烦。

1） 由于实际温度传导到采集模块，可能存在滞后性，因此对应表做的太细意义不大；

2） 温度与加热时间的确定，依赖于人工判断。可以通过打印相同内容（如条纹），在不同温度下，打印出来的灰度基本一致为合适加热时间的标准。

通过调整加热时间，最主要是解决，刚开始打印（冷机状态）的时候内容比较淡，打印一段时间后内容又太浓。

2、加热点密度

这里提到的密度分纵向与横向。纵向是指同样的一个点，在时间维度上一直处于加热状态。横向是指，加热点在空间上，相对集中的分布。密度的不均会带来灰度的不均匀，严重的情况下，甚至还会有很明显的拖尾现象。改善思路：

1） 纵向的密度

对于单段打印，可以适当降低加热时间，对于多段打印，可以将与上一点行有交集的部分单独分成一段来加热，即人为降低密度。

2） 横向的密度

类似的，对于单段打印，可以适当降低加热时间，对于多段打印，可以进行拆分，即集中在一起的点分到不同段去加热，也是人为降低密度。

以上，解决思路是相似的，但只能改善，不能彻底解决。

PS：从改善打印显示的角度，选择字印字体也很重要，尽量选择圆润，等宽，并且尽量大（尽可能占满，比如 24*24 点阵的空间）的字体。

hehuoxuetang