多通道打印平台使用说明与设计方案

目录

1、微流控液滴打印平台总体设计方案与使用说明 1

1.1 液滴打印平台总体设计方案 1

1.2 液滴打印平台使用操作说明 2

1.3 液滴打印工作模式 4

1.4 常见问题及解决办法 6

2、硬件电路设计 6

2.1 单片机控制电路设计 7

2.2 致动器的选取与驱动电路设计 9

2.3 步进电机电气连接说明 14

2.4 硬件触发电路 15

3、控制程序 17

3.1 单片机控制程序设计及说明 17

3.2 上位机程序设计及说明 22

4、打印头机械结构及装配方法 23

4.1 多通道打印头结构设计 23

4.2 多通道打印头装配方法 23

1、微流控液滴打印平台总体设计方案与使用说明

1.1 液滴打印平台总体设计方案

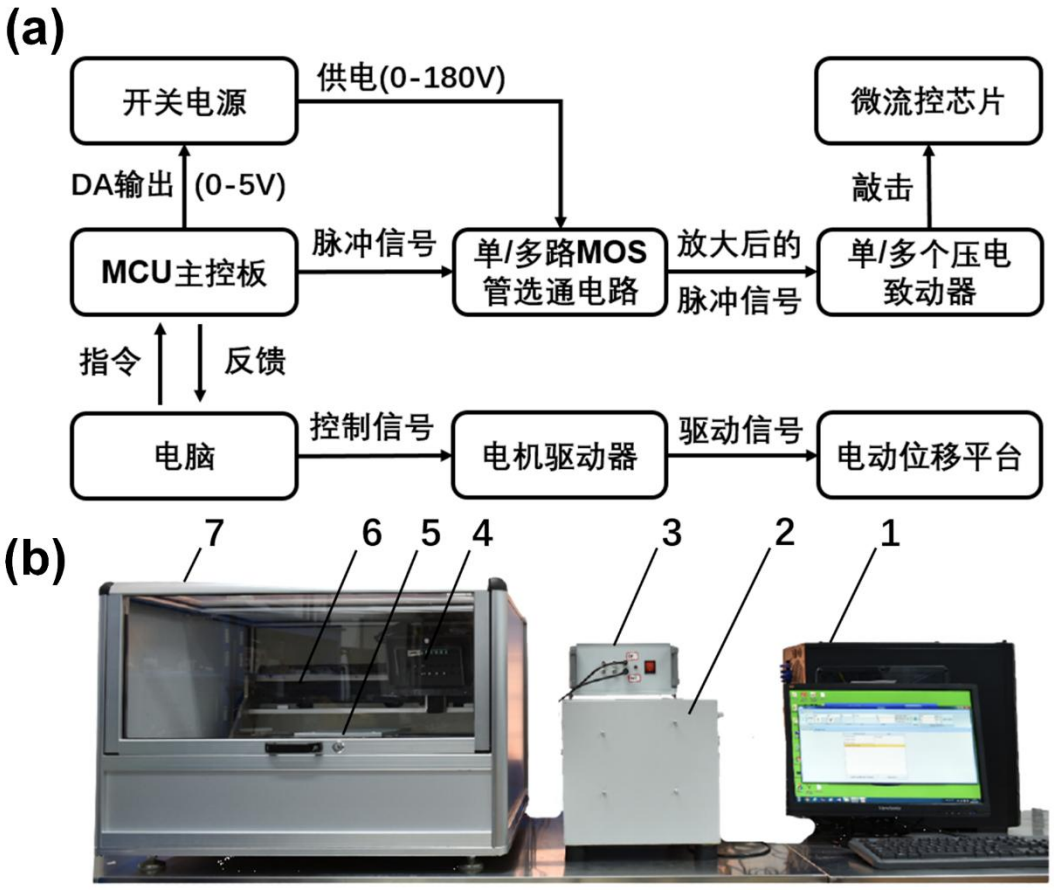


图 1.1 (a) 单/多通道微流控液滴打印系统原理框图。(b) 多通道微流控液滴打印平台实物照片，其中，1-电脑、2-电机驱动器控制箱、3-主控制箱 (包含 MCU 控制板与开关电源)、4-打印头 (包含致动器与微流控芯片)、5-工作台、6-位移轴、7-外壳。

本项目中设计的基于微流控冲击打原理的微液滴打印平台的系统结构如图 3.1a 所示。该系统主要由以下几部分组成：电脑、控制电路、驱动电路、打印头、微流控芯片及电动位移台。该系统的工作过程如下：首先，通过电脑发送指令到微控制器 (MicroControllerUnit, MCU)控制板，该控制板随后根据接收的指令输出对应的模拟电压调节开关电源的输出电压。同时，MCU 输出控制脉冲信号，

经过驱动电路放大，以驱动压电致动器。所使用的压电致动器末端安装有金属针(1381-1, Micro pins, KEYSTONE)，该金属针用于敲击微流控芯片的变形腔，从而产生微液滴。此外，该系统中还包含有一个三轴电动位移台。电脑端可以通过上位机软件发送控制信号给电机驱动器来精确控制该电动位移台的位移动作，从而与液滴打印头相配合实现液滴阵列或图案的打印。最终研制的微流控液滴打印平台样机实物如图 3.1b 所示。

为了实现上述液滴打印平台的研制，本章的工作主要有两个部分：一是致动器的选型与驱动方案设计；二是分液平台的控制电路、机械结构以及程序开发。以下将详细说明该微流控液滴打印平台这两部分工作进行详细的说明。

1.2 液滴打印平台使用操作说明

1、开机前检查。首先确认电脑 1、电机驱动器控制箱 2 供电正常。确认电机驱动器控制箱 2 与电脑机箱轴卡连线正常。连接好主控箱 3 与电脑 1 的 USB 串口通信线。

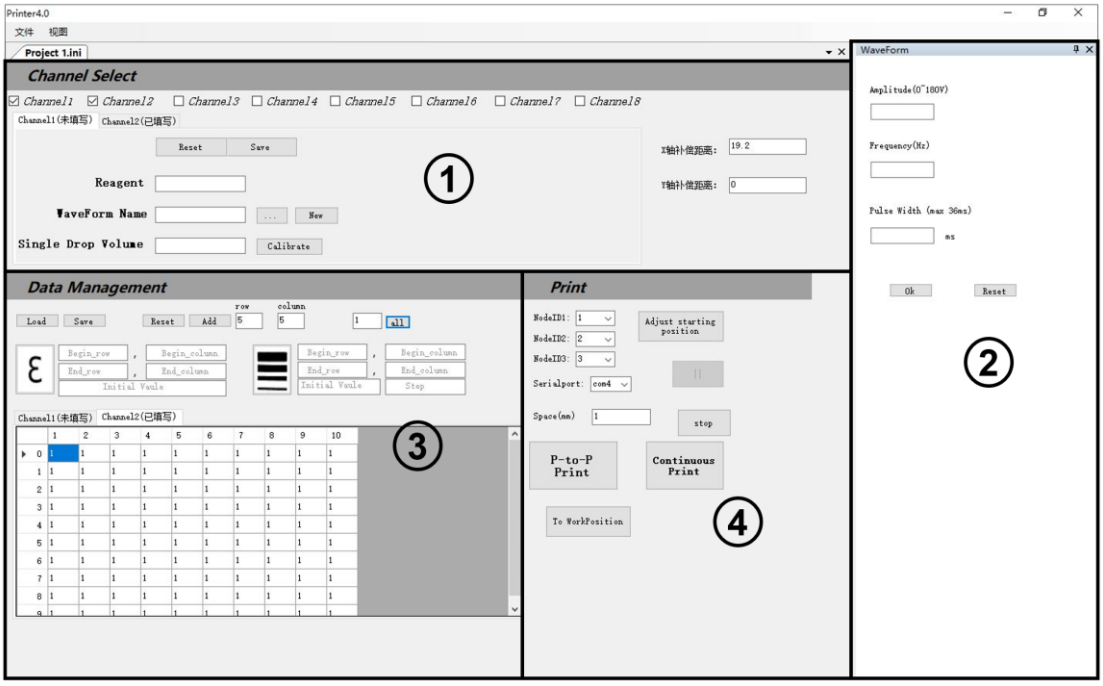


图 1.2 多通道微流控冲击打印系统控制软件界面。

2、开机。顺时针旋转电机驱动器控制箱 2 上的红色旋钮，为电机驱动器供

电。双击打开电脑端程序文件夹中的“printer4.0.sln”文件，在 Visual Studio 软件中单击启动按钮后，会弹出上图 1.2 所示的界面。该界面其主要由四部分组成：第一部分为通道选择区，其主要功能为选择所需要工作的通道并设置其试剂以及单液滴体积参数；第二部分为驱动波形设置区，其主要功能是设置所选择的每个通道对应的压电致动器驱动电压信号的频率、幅值和脉冲宽度时间参数；第三部分为液滴阵列设置区，其功能为设置所需要的液滴阵列参数，如液滴阵列行列数、每个位置点的液滴数；第四部分为打印起始控制区，其功能为控制液滴阵列打印的起始、选择液滴阵列打印模式和液滴阵列间距设置。

3、移动台初始化。首先点击图 1.2 中 Print 栏下方的“Adjust starting position”按钮，弹出下图 1.3 所示的界面。点击图 1.3 中的“ToOrigin”按钮，移动台移动到原点位置。随后通过点击左边的↑↓←→按钮调节打印头移动到达所需的位置，单次点击移动的步长在 distance 框中输入设置。

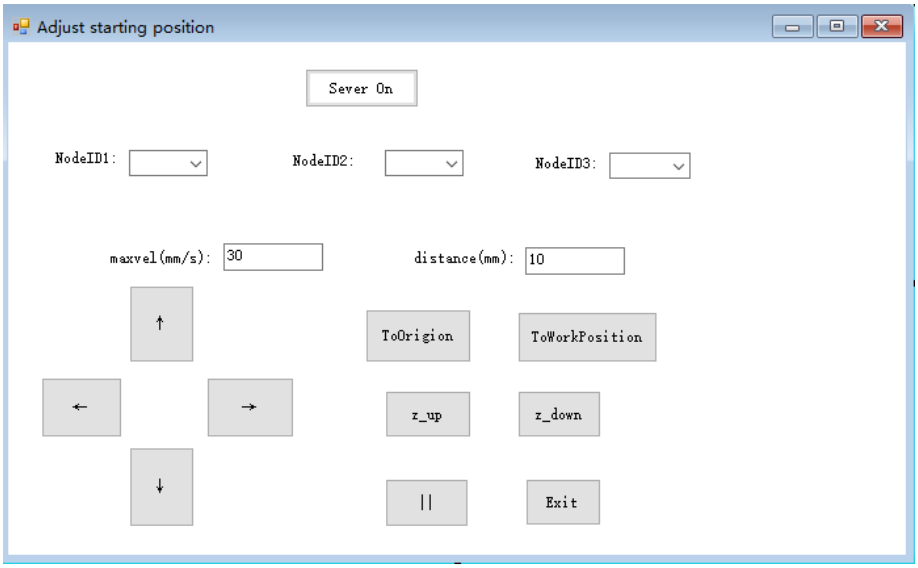


图 1.3 移动台位置调节界面。

4、液滴打印通道与致动器工作电压波形参数设置。在 Channel Select 栏下单击勾选要打印的通道。如果需要双通道混合打印时，则需要在右侧 X 轴与 Y 轴补偿距离添加，所使用的两个通道在 X 方向与 Y 方向的偏差值。压电致动的的电压波形可以通过点击“WaveForm Name”右边的按钮选择文件进行设置，也可通过在图 1.2 中区域②中，驱动电压波形的幅值、频率和脉宽时间进行设置。其中，驱动电压范围为 0 至 180 V，频率通常为 5 Hz 至 20 Hz，脉宽时间通常为 5 ms。

5、液滴阵列参数设置。在图 1.2 中区域③中进行所需打印的液滴阵列参数进行设置，该部分界面如下图 1.4 所示。通过“row”与“column”分别设置液滴阵列的行数与列数。如果是阵列的每个位置打印的液滴数目相同，直接在右侧框中输入液滴数后，点击“all”按钮即可。如果需要设置行列的梯度，则在下方框中输入起始行与列和结束行与列，并输入初始值与步长即可。液滴阵列的间距在 Print 栏下的“Space”右侧框中输入设置。

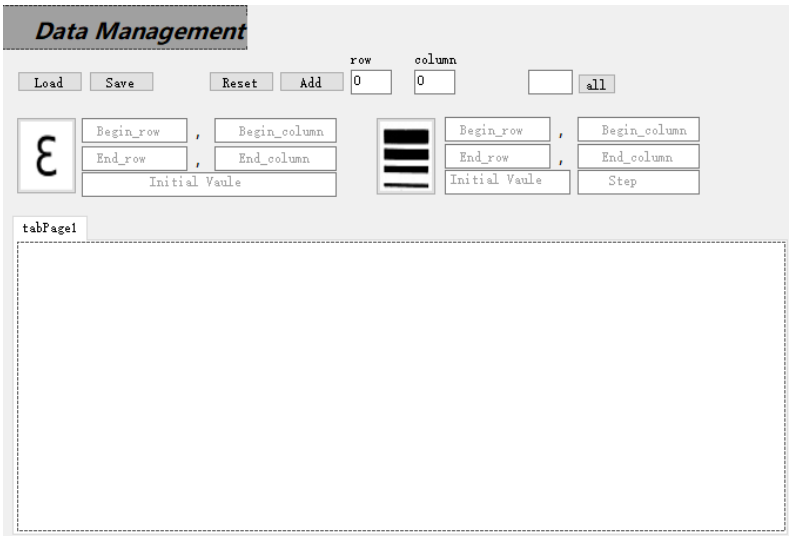


图 1.4 打印液滴阵列参数设置界面。

6、液滴打印。通过点击图 1.2 中区域④中的“P-to-P Print”或“Continuous”按钮后，仪器会开始液滴打印，直至液滴阵列打印完成。这两种液滴打印模式分别为点对点打印和连续打印，具体细节在下一小节“液滴工作模式说明”中详细说明。

1.3 液滴打印工作模式

液滴矩阵的打印方式有两种，分别是点对点打印和连续打印。简单来说，点对点打印就是在打印液滴矩阵时，移动台移动到一位置点停下来，打印头打印产生一定数量液滴，然后移动台将打印头移动到下一个位置点停下再进行液滴打印，如此循环。而连续模式是再矩阵同一行液滴生成时，一边移动打印头一边进行液滴打印，通过配合液滴打印频率和移动台移动速度来保证液滴间距。

这里首先说明一下点对点打印模式下，PC 端与 MCU 之间通信的工作过程。首先是在 PC 端软件界面设置好所需要的液滴矩阵参数矩阵(主要包括行列数、

每个点的液滴数)，PC 端通过电机驱动器控制移动台按照指定的矩阵参数进行运动，运动到矩阵第一个位置点后，PC 端软件通过串口通信发送该位置点液滴打印参数，指令格式为“1”+“XX”+“XXX”+“#”。第一个字符 1 为 MCU 模式判断标识位；后面的两位数字符“XX”代表液滴打印频率；再后面的三位数字符“XXX”代表该位置点所需要打印的液滴数量；最后为串口接收结束符“#”。MCU 接收到指令后，首先根据 5.2 节介绍的电路判断移动台运动状态是否为静止状态，若为静止状态，则按照接收的指令控制打印头以指定频率产生所需数量液滴。打印结束后 MCU 会发送字符“1”给 PC 端，作为打印结束标识符，然后 PC 端控制移动台移动到下一个位置点，如此循环直到打印结束。

点对点工作模式：

打印头移动到一个位置后，PC 端将该位置的液滴打印需求（打印频率、液滴数）发送给单片机。单片机接收到指令后，首先通过 5.2 节中介绍的电路判断移动台的运动状态，若为停止状态，则单片机控制打印头按照指令产生所需液滴。液滴打印结束后，单片机反馈一个结束符给 PC 端。PC 端接收到结束符后，控制打印头移动到下一个位置点，如此往复直到结束。点对点工作模式的有点在于液滴位置精度高，打印的液滴阵列行列之间对齐精度也较好。缺点是由于需要移动台移动到位置点后停止才打印液滴，因此其液滴阵列的生成速度较慢。

连续工作模式：

连续工作模式就是让打印头移动的同时打印液滴，这样就可以实现更快速的生成液滴阵列。这种工作模式的重点就在于如何保证液滴阵列位置/间距的精度以及液滴阵列的不同行液滴的对齐。

当 PC 控制移动台的移动一段距离时，移动台会经历加速阶段、匀速阶段和减速阶段三个阶段。因此为了保证产生液滴间距相同，就需要在移动台匀速运动阶段进行液滴打印。液滴间距大小根据移动台运动速度与打印头液滴生成频率确定。液滴阵列的位置根据加速阶段的运动距离以及液滴打印开始时间点确定。目前是通过给定一个延时时间，使得液滴打印开始的时间点在移动台匀速运动阶段。对于不同的液滴打印频率和液滴间距，所需的延时时间也是不同的。目前是通过实验测试不同频率所需的延时时间，基本能实现液滴阵列的对齐。后续可以考虑设计硬件电路的方法判断移动台是否在匀速运动阶段，进而触发液滴打印，来增加位置精度与可靠性。

1.4 常见问题及解决办法

1、打开打印软件界面后，无法识别到 COM 口。

解决方法：首先检查单片机主控板与电脑间的串口通信 USB 线连接是否正常，其次确认 CH340 驱动是否已安装。

2、软件启动后，显示连接不上移动台。

解决方法：可能的原因是电机驱动器未供电。在启动软件前，一定要先确保电机驱动器供电正常。

3、开机供电后，移动台无法移动。

解决方法：如果是 XYZ 轴均无法移动，可能的原因是电机与电机驱动器之间接线发生松动，重新插拔一下。如果某一个或两个轴无法移动，其它轴移动正常，可能是限位开关接线不良或损坏。该问题可以通过观察对应轴上的限位开关指示灯是否点亮，如下图 1.5 所示。如果指示灯不亮，则需要检查限位开关的接线与功能是否正常。

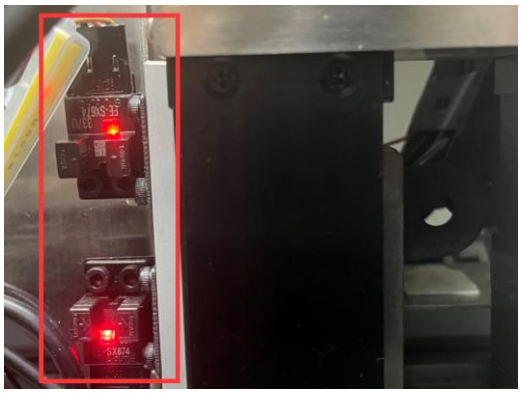


图 1.5 限位开关指示灯实物照片。

2、硬件电路设计

该平台中的硬件电路主要包含：单片机核心控制板、致动器驱动电路、运放电路、步进电机控制电路与硬件触发电路。接下来对各部分的硬件电路设计与选型进行详细说明。

2.1 单片机控制电路设计

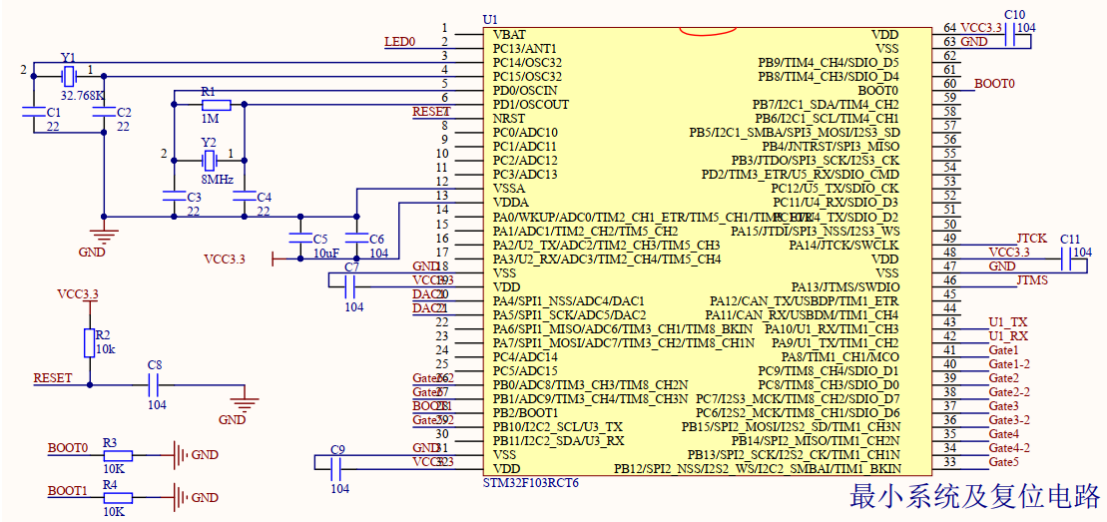


图 2.1 最小系统及复位电路。

控制核心采用的是意法半导体公司发布的 STM32F103RCT6。该微控制器(MCU)为工作频率 72MHz 的 32 位 ARM 控制器，闪存 256K，内存 48K，采用的是 LQFP64 封装。供电电压 3.3V，一般 0.8V 以下为低电平，2V 以上就是高电平，中间状态不确定。该芯片上集成 8 路 16 位定时器、16 路模数转换 (ADC) 、2 路数模转换 (DAC) 模块以及串口通信等外设功能，基本能满足常见的嵌入式开发需求。

控制电路包括单片机最小系统电路、复位电路、电源电路、下载电路、串口通信电路、运放电路以及接口等。

最小系统电路、复位电路和电源电路是 MCU 能正常工作的必需的部分。最小系统电路主要包括时钟电路和一些储能电容，如图 2.1 所示。采用的是 8MHz 的石英晶振，起振后经过 MCU 片内集成的的倍频电路后变成 72MHz，为 MCU 正常工作提高时钟基准。储能电容的作用是保持 MCU 供电稳定，防止电流脉冲引起异常保证其正常工作。复位电路是在 RESET 引脚上接一个上拉电阻和一个下拉电容，当 RESET 引脚为低电平时 MCU 会复位重启。上拉电阻与下拉电容的目的是防止电流脉冲引起一场复位，因此复位电路是必需的。

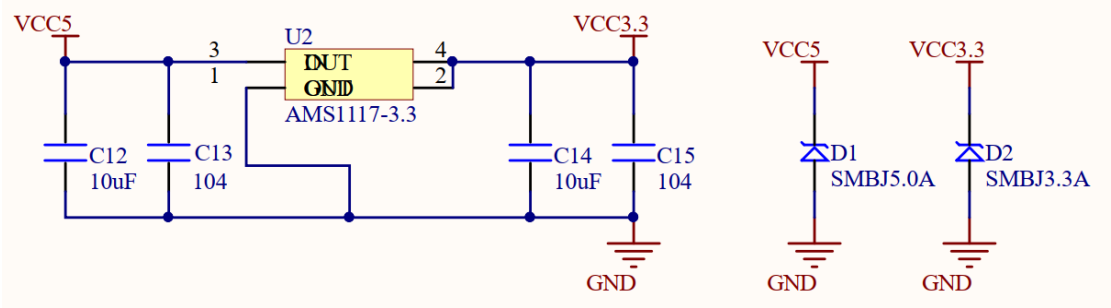


图 2.2 供电电路。

如图 2.2 所示，电源电路部分选取的是 AMS1117-3.3 稳压芯片将串口是 5V 电压稳压到 3.3 V。该芯片电压输入范围为 4.75 V~12 V，输出电流 0~1A。另外在电路上添加了型号为 SMBJ5.0A 与 SMBJ3.3A 的瞬态电压抑制二极管 (TVS)。当发生过压击穿时，TVS 管击穿将过高的能量对地泄放，可以有效地保护电子线路中的精密元器件，免受各种浪涌脉冲的损坏。

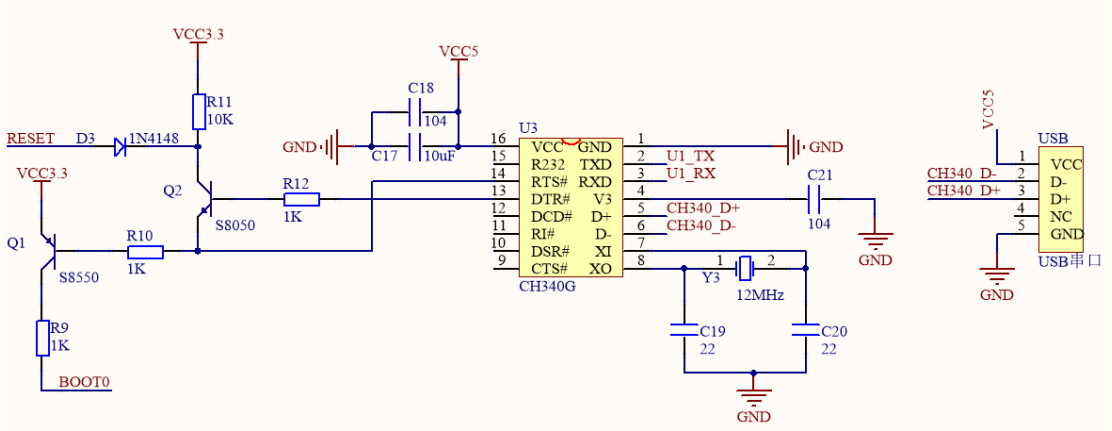


图 2.3 串口下载电路。

串口通信与下载电路如图 2.3 所示，电平转换芯片选择的是兼容 3.3 V 供电的 CH340G，可同时完成发送和接收转换的双重功能。在 STM32F103RCT6 中，提供了 BOOT0 和 BOOT1 两个管脚用于三种启动模式选择。本系统采用从系统存储器启动,也就是串口下载。BOOT0 和 BOOT1 两个管脚在芯片复位时的电平状态决定了芯片复位后从哪个区域开始启动，启动模式配置如表 2.1 所示。

表 2.1 STM32 启动配置

BOOT0	BOOT1	启动模式	说明
0	X	用户闪存存储器	用户闪存存储器，也就是 FLASH 启动
1	0	系统存储器	系统存储器启动，用于串口下载
1	1	SRAM 启动	SRAM 启动，用于 SARM 中调试代码

程序下载使用的是 FLYMCU (一键下载工具)，可以控制 CH340G 的 DTR#引脚和 RTS#引脚的输出的高低电平状态，通过如图 2.3 所示的电路进而控制 STM32 主控芯片 BOOT0 和 RESET 引脚的状态。DTR# 和 RTS#初始状态都是高电平，在启用下载的时，DTR#会维持高电平状态，RTS#先拉低，此时两个三极管 Q1 和 Q2 均导通，那么 BOOT0 为高电平，RESET 为低电平，MCU 进行复位。然后 RTS#变为低电平，此时 Q1 导通，Q2 不导通，复位结束，BOOT0 为高电平。由表 2.1 可知，STM32 启动模式变为从系统存储器启动，启用串口下载，STM32 芯片与 FLYMCU 下载软件进行通信下载代码，等待代码下载结束后，RST#先变为高电平，然后 DTR#变成高电平。如果 DTR#先变成高电平，则会产生一次复位。

2.2 致动器的选取与驱动电路设计

1. 致动器的选型

在本项目中选择压电致动器作为微流控冲击打印系统的致动器。压电致动器可分为压电梁、压电片和压电堆栈。其中压电梁相比于另外两种，具有行程较大、冲击力适中以及成本相对较低的特点。最终本项目选取了如图 2.4 所示的苏州攀特电陶科技股份有限公司的 QDTE52-7-0.82-3 型号的压电梁，作为微流控冲击打印系统的致动器，致动器的末端安装有金属针(1381-1, Micro pins, KEYSTONE)，用于冲击微流控芯片的变形腔，从而产生微液滴。其主要的规格参数如表 2.2 所示。

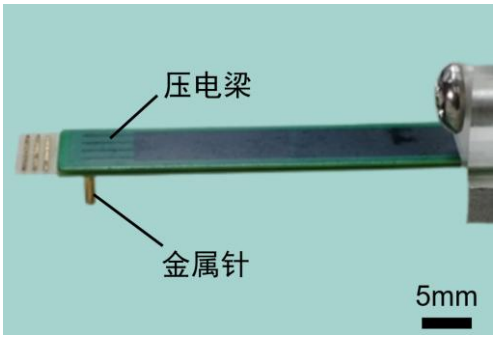


图 2.4 压电致动器实物图。

表 2.2 压电梁的规格参数

位移 (mm)	推力 (mN)	漏电流 (μA)	静电容量 (nF@1V,100Hz)	驱动电压 (V,DC)
≥0.3	≥100	≤0.5	40~62	0~180

2. 致动器驱动电路设计

由于选取的压电致动器的工作电压一般在 60 V-180 V，因此需要使用到 MCU 的 DAC 模块。STM32F103RCT6 芯片可以以 12 位精度输出 0-3.3 V 的模拟电压，因此其输出电压的分辨率为 $3.3V/(2^{12}-1)=8.06\times10^{-4}V$。驱动压电致动器的方案有两种，一种方案是将 MCU 输出的模拟电压经过运算功率放大器放到 0 到 180V 模拟电压。另一种方案是设计一个 MCU 通用输出引脚控制的开关选通电路，并使用开关电源供电以获得高电压方波来驱动压电致动器，如图 2.5 所示。下面分别介绍这两种方案的具体实施细节。

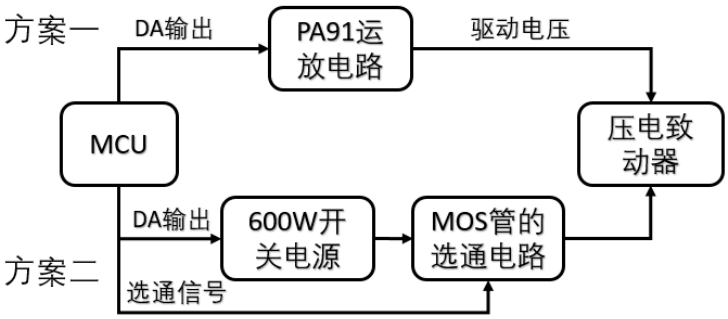


图 2.5 开关电源接线图。

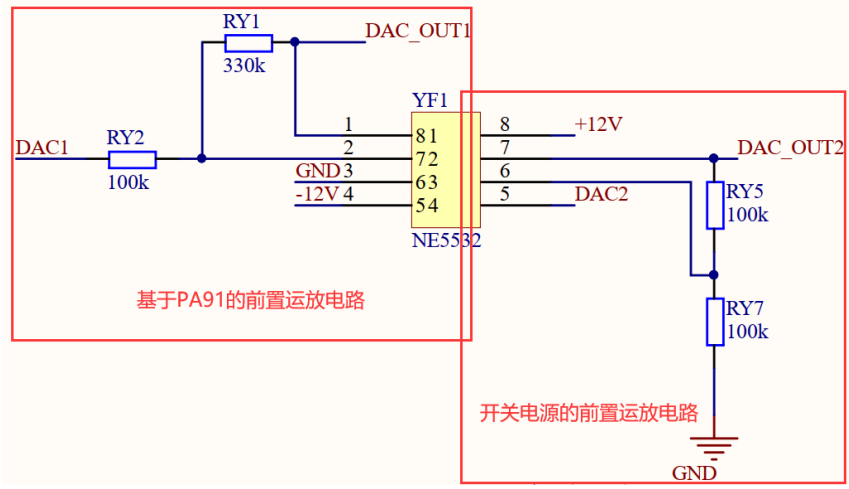


图 2.6 运放电路电路。

方案一：运放放大方案

由于实验室现有基于 PA91 设计的运算放大器，其放大倍数为-20 倍。因此为了得到 0-180 V 电压，MCU 输出的 0-3.3 V 还需要经过一个运放电路得到 0 到 -9V 的电压。选取了 NE5532 运放芯片设计了图 2.6 中左半部分的负相比例放大电路，放大倍数=(1+330k/100k)=-4.3。

方案二：基于 MOS 管的开关选通电路 + 开关电源供电

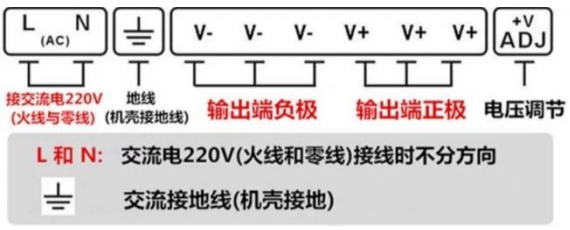


图 2.7 开关电源接线图。

开关电源选择的是拥有模拟电压可调节输出功能的 600 W 开关电源，开关电源的接线图如图 2.7 所示。通过 ADJ 接口接入的电压 0~5 V 同比例放大对应输出电压为 0~220 V，因此 MCU 输出的 0-3.3 V 还需要经过一个运放电路得到 0 到 5V 的电压。如图 2.6 所示，选取了 NE5532 运放芯片设计了图 2.6 中右半部分的同相比例放大电路，放大倍数=(100k/100k)=2。

基于 MOS 管的开关选通电路设计：

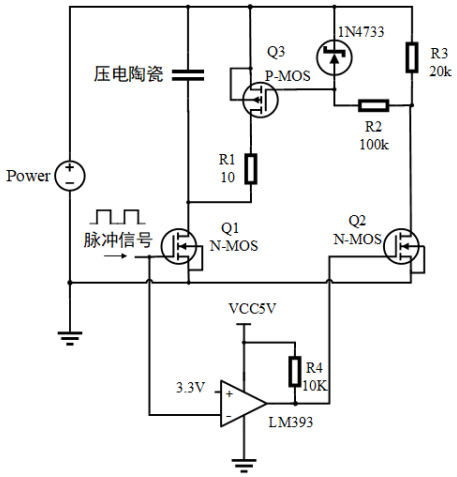


图 2.8 基于 MOS 管的开关选通电路。

微流控冲击打印的致动器选择的是商用的压电致动器（QDTE52-4，攀特）。压电致动器开关选通电路如图 2.8 所示，该驱动电路主要由两个 N 型 MOS 管、一个 P 型 MOS 管、一个比较器电路、一个稳压二极管与一些电阻组成。其中使用 LM393 器件组成的比较电路的功能是为了将 5V 的脉冲信号进行取反（即将高电平转换为低电平，低电平转换为高电平）。该比较电路的作用是为了实现 MOS 管 Q1 与 Q2 的导通状态是相反的（即 Q1 导通时 Q2 是未导通，Q1 未导通时 Q2 导通）。下面详细地说明一下该电路的工作过程：

在充电阶段，脉冲信号为高电平，则 Q1 导通，Q2 不导通。当 Q1 导通时，压电陶瓷直接连接在 Power 直流电源两端，此时压电陶瓷充电。由于 Q2 处于未导通状态，所以此时右边的回路处于断路状态，稳压管二极管 1N4733 不工作，MOS 管 Q3 的 V_{SG}=0V,所以 Q3 也未导通。当前 R1 所在的回路处于断路状态，R1 在充电阶段没有功耗。在放电阶段，脉冲信号为低电平，Q1 未导通，Q2 导通。由于 Q1 未导通，压电陶瓷充电回路断开。Q2 导通，则稳压二极管 1N4733 反向击穿，其反向击穿电压为 5.1 V，即此时 Q3 的 VSG=5.1 V，所以 Q3 导通，此时 R1 相当于直接连接在压电陶瓷的两端，从而实现压电陶瓷的放电。通过压电致动器的不断充放电来实现致动器末端的打印针不断重复撞击微流控芯片的变形腔。

由于选择的 MCU 有着许多空闲引脚，因此该电路也可以不需要比较器来获得反向电平信号，可以直接使用两个输出引脚来控制压电梁的充放电。改进之后

的电路如下图 2.9 所示，压电致动器的阳极接在 PZT-OUT1 上，阴极与电源地相接。通过单片机输出引脚 Gate1 与引脚 Gate1_2 来控制压电致动器的充放电。

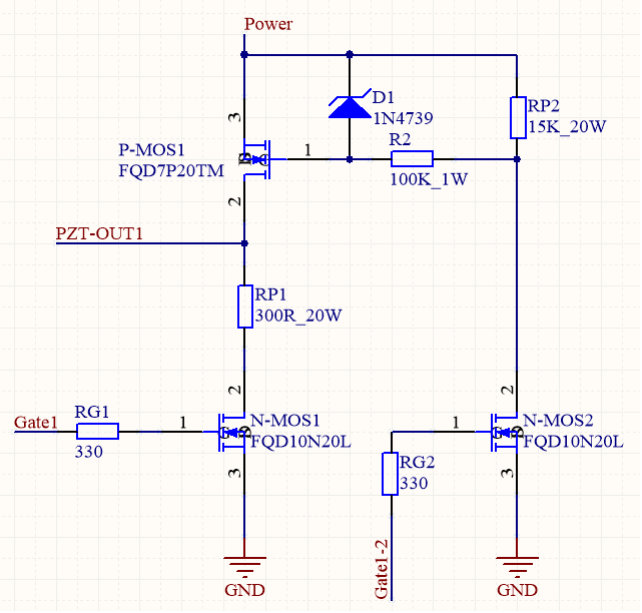


图 2.9 改进后的 MOS 管的开关选通电路。

元器件选择说明：

MOS 管选择时需要 MOS 管的耐压值高于 180 V，一般需要留有安全余量因此选择了耐压值 200V。在电路中稳压二极管的作用是控制 P-MOS 管的开关，其稳压值=-V_{GS},通过 P-MOS 管的数据手册可知-V_{GS}＞8 V 才能够保证 MOS 管通过电流大于 10A，因此选择了稳压值 9.1V 的稳压二极管 1N4739。电阻 RP1 是压电致动器放电的卸荷电阻，其阻值越小压电致动器放电速度越快，但是同时放电电流增加导致电阻 RP1 与 N-MOS1 发热严重，因此电阻 RP1 的阻值不能太小。具体元器件选型见表 2.3。

表 2.3 电路器件选型

器件	型号	封装	数量	参数
N-MOS 管	FQD10N20L	TO252	2	200 V，7.6 A
P-MOS 管	FQD7P20TM	TO252	1	-200 V，5.7 A
稳压二极管	1N4739	DIODE-0.4	1	1 W，9.1 V
电阻	330 Ω	0603	2	0.25 W
电阻	300 Ω	TO220-2	1	20 W

电阻	100 kΩ	AXIAL-0.5	1	1 W
电阻	15 kΩ	TO220-2	1	20 W

两种方案对比分析与测试：

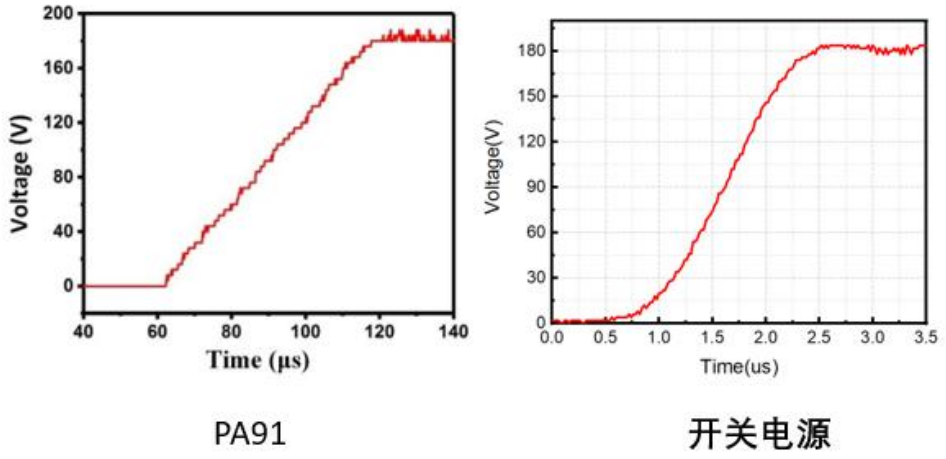


图 2.10 两种方案电压波形上升沿时间测试。

两种方案方案对压电制动器的充电速度进行了测试。测试方法是使用示波器测量两种方案输出 180 V 方波时压电致动器两端的波形变化，测得的上升沿结果如图 2.10 所示。从测试结果中可以看开关电源加选通电路的方案压电致动器充电速度更快，压摆率由方案一的 3.24 V/μs 提高到方案二的 90 V/μs。同时我们使用了激光位移传感器测试了压电致动器末端的位移速度从 0.42m/s 提高到 0.63m/s。由此可见方案二的优势在于可以以较低的成本实现对压电致动器的更快速的驱动。但是方案一的优势在于输出的电压波形是可调的，可以控制电压下降沿的波形与时间，输出如正弦波形等，这对于控制打印回吸以及增加液滴打印稳定性都是有帮助的。而方案二是基于 MOS 管开关电路设计的，因此只能输出方波。

2.3 步进电机电气连接说明

本平台的移动台运动部件方案采用的是现有的商业产品，具体所选择的电机驱动模组为中达电通公司的伺服电机 ECMA-C20604RS、泛用型伺服驱动器 ASDA-B2-400W 和轴卡 PCI-DMC-F02。伺服电机与其驱动器和轴卡的电气连接详见文件“JM20119-三轴移液工作台.pdf”。

2.4 硬件触发电路

为了使打印过程中，移动台的移动与致动器打印动作时序相匹配，本研究设计一个信号滤波处理电路的硬件触发方案。其原理是电机驱动器可以根据移动台移动状态的来输出高低电平变化的信号，该信号经滤波转化后接入到 MCU 引脚，从而使得单片机根据接入引脚电平信号的高低可以实时判断移动台的工作状态。

电平检测电路：

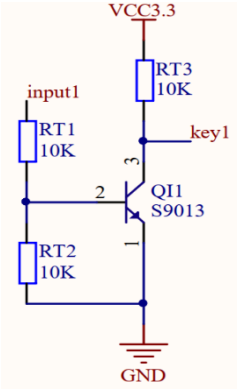


图 2.11 电平检测电路。

电平检测电路的功能是检测输入接口 input1 电平为低电平还是高电平，如图 2.11 所示的三极管选通电路，将输入的高低电平信号转换为 3.3 V 的低高电平信号，保护 MCU 引脚。

信号滤波电路：

本平台所选择的电机驱动模组为中达电通公司的伺服电机 ECMA-C20604RS、泛用型伺服驱动器 ASDA-B2-400W 和轴卡 PCI-DMC-F02。该套驱动方案的最高转速可达 5000 rpm，最大扭矩为 3.82N/m。此外，该驱动器可以输出代表电机当前工作状态的信号，即当电机处于运动状态时输出高电平(幅值为 5V)，静止时输出低电平(0 V)，如图 2.12a 所示。从中可以看出从驱动器直接输出的信号噪声较大，噪声的峰峰值($V_{p\text{-}p}$)高达 7.52 V，信噪比为-1.8 dB，容易出现误触发的情况，因此不能直接用于触发压电致动器的动作。从图 2.12a 的测量结果可以看出，信号噪声为高频噪声。因此，本研究设计了一个 RC 低通滤波器（电阻 10 kΩ，电容 0.1 μF，截止频率 f = 1/(RC) = 1k Hz）来过滤高频噪声，如图 2.12d 所示。如图 2.12b 所示，信号经过滤波处理后的波形图，噪声明显减低，

噪声的 $V_{p\text{-}p}$ 降低到 3.96 V。为了进一步使减少触发信号的信噪比，我们使用 LM393 比较器模块设计了如图 2.12d 所示的比较电路。如图 2.12c 所示，滤波后的信号经过比较电路后，信号噪声得到了进一步降低，噪声的 $V_{p\text{-}p}$ 降低到 0.43 V，此时信噪比提高到了 10.6 dB，基本可以避免误触发的情况。由于有 X 和 Y 两轴伺服电机运动的状态判断，因此在电路中加入一个或门电路，最终设计的电路原理图与 PCB 实物图如图 2.12e 所示。

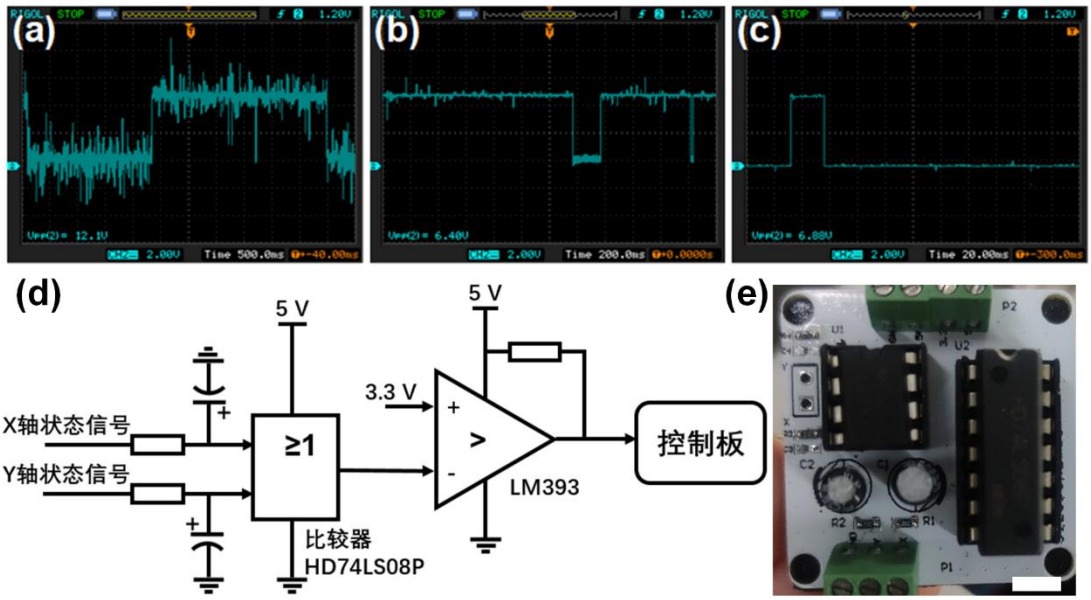


图 2.12 (a) 驱动器输入信号波形。(b) 滤波处理后的信号波形。(c) 比较电路处理后的信号波形。(d) 硬件触发电路原理图，包含：RC 低通滤波电路，比较电路和或门电路。(e) 触发电路板实物图，标尺：5 mm。

3、控制程序

3.1 单片机控制程序设计及说明

MCU 的程序是基于 C 语言语法进行设计。该程序设计的功能需求是能够实现对压电致动器的控制，主要包括：驱动电压幅值可调，脉宽时间可调，压电致动器的动作频率与次数可调节。具体实现方式是通过电脑端使用串口通信工具发送指令到单片机，然后单片机根据指令进行相应的动作控制。因此程序代码模块主要包含：串口通信模块、定时器模块、通用输出引脚控制模块以及 DAC 模块。下面是对各部分程序代码的详细说明。

串口通信模块：

```
if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)  //接收中断(接收到的数据必须是0x23结尾)
  {
  Res =USART_ReceiveData(USART1); //读取接收到的数据

  if((USART_RX_STA&0x8000)==0)//接收未完成
    {

      if(Res==0x23)USART_RX_STA|=0x8000;//接受结束标志字符，0X23代表#
      else
        {
        USART_RX_BUF[USART_RX_STA&0X3FFF]=Res ;
        USART_RX_STA++;
        //  printf("  m=%c   \r\n",USART_RX_BUF[t]);
        if(USART_RX_STA>(USART_REC_LEN-1))USART_RX_STA=0;//接收数据错误,重新开始接收
        }

    }
  }
```

图 3.1 串口中断函数。

串口通信程序的核心部分是串口中断函数，如图 3.1 所示。使用官方库函数 USART_ReceiveDate()读取 USART1 接收缓冲区的数据 (串口通信发送的数据格式为字符)，并将读取的字符赋值给定义变量 Res。定义了变量 USART_RX_STA 作为接收状态标记，初始赋值为 0。通过 Res 变量作为中间传递变量，将接收的字符依次赋值给数组 USART_RX_BUF[]中，从 USART_RX_BUF[0]开始依次赋值。直到接收到字符为# (其 ASCII 码值为 0x0d)时，则判断接收完成并退出循环。

```
void ReceiveDate(void)
{
    if(USART_RX_STA&0x8000)
        {
            len=USART_RX_STA&0x3fff;//得到此次接收到的数据长度

            for(t=0;t<len;t++)  //将USART_RX_BUF[t]的高4位清零
            {
                USART_RX_BUF[t]=USART_RX_BUF[t]&0x0f;

            }

            m_Voltage=USART_RX_BUF[0]*100+USART_RX_BUF[1]*10+USART_RX_BUF[2];
            m_Frequency=USART_RX_BUF[4]*100+USART_RX_BUF[5]*10+USART_RX_BUF[6];
            m_PulseTime=USART_RX_BUF[8]*10+USART_RX_BUF[9];
            m_Channel=USART_RX_BUF[11];
            START=USART_RX_BUF[13];
            mode=USART_RX_BUF[15];
            Print_Times=USART_RX_BUF[17]*100+USART_RX_BUF[18]*10+USART_RX_BUF[19];
            printf("v=%d,f=%d,p=%d,c=%d,START=%d,len=%d,Mode=%d,Print_Times=%d \n",m_Voltage,m_Frequency,m_PulseTime,m_Channel,START,len,mode,Print_Times);
            pin=(1000/m_Frequency);
            vol=(float)m_Voltage*11.183-10.427;
            if(vol>vol_max)vol=vol_max;
            if(vol<vol_min)vol=vol_min;
            Dac2_Set_Vol(vol);
            ADC1_sample();
            USART_RX_STA=0;
    }
}
```

图 3.2 串口接收数据处理函数。

如图 3.2 所示，由于接收的数据均为字符格式，因此需要将字符转化为数字。只需要将获得的字符与 16 进制数 0x0f 做&运算即可。如字符 1 对应的 ASCII 码值是 0x31，与 0x0f 做&运算后的 0x01。接着，将转化后的数据赋值给相应的变量。并用 Printf()函数将获得的数据反馈到电脑串口工具，以校正接收数据是否有误。数据处理的最后调用 Dac2_Set_Vol()函数根据接收的数据输出相应的模拟电压。这里对数据发送格式与变量名含义做一下简要的说明。

数据发送格式：XXX XXX XX X X X XXX#

含义说明：第一个三位数 XXX 代表压电致动器的电压值，单位为 V；第二个三位数 XXX 代表压电致动器的动作频率，单位为 Hz；第三个两位数 XX 代表高电压的脉宽时间，单位为 ms；第四个一位数 X 代表选择的通道序号；第五个一位数 X 代表启示位，0 代表接收到指令后压电致动器不动作，1 代表接收到指令后压电致动器立刻动作；第六个一位数 X 代表工作模式位，0 代表压电致动器会一直按照前几位数据的指令进行动作，1 代表压电致动器会进行有限次数的动作，其动作的次数等于最后三位数字 XXX 的大小。#为发送数据的结束标志位。

例如：090 010 05 1 1 0 100# 单片机接收到该指令后，通道 1 对应的压电致动器就会以 10Hz，电压幅值为 90V，脉宽为 5ms 持续敲击。

例如：090 010 05 2 1 1 100# 单片机接收到该指令后，通道 2 对应的压电致动器就会以 10Hz，电压幅值为 90V，脉宽为 5ms 敲击 100 次后停止动作。

注意点：数字位数不能改变，不够的需要在前面添加 0；数字之间用空格隔开；一定要以#结尾。

定时器模块：

```
void TIM3_Int_Init(u16 arr,u16 psc)
{
    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE); //时钟使能

  TIM_TimeBaseStructure.TIM_Period = arr; //设置在下一个更新事件装入活动的自动重装载寄存器周期的值    计数到5000为500ms
  TIM_TimeBaseStructure.TIM_Prescaler =psc; //设置用来作为TIMx时钟频率除数的预分频值   10Khz的计数频率
  TIM_TimeBaseStructure.TIM_ClockDivision = 0; //设置时钟分割:TDTS = Tck_tim
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM向上计数模式
  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure); //根据TIM_TimeBaseInitStruct中指定的参数初始化TIMx的时间基数单位

  TIM_ITConfig(  //使能或者失能指定的TIM中断
    TIM3, //TIM2
    TIM_IT_Update ,
    ENABLE  //使能
    );
  NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;  //TIM3中断
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;  //先占优先级0级
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;  //从优先级3级
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //IRQ通道被使能
  NVIC_Init(&NVIC_InitStructure);  //根据NVIC_InitStruct中指定的参数初始化外设NVIC寄存器

  TIM_Cmd(TIM3, ENABLE);  //使能TIMx外设

}
```

图 3.3 定时器初始化函数。

定时器模块主要有两个函数，一是定时器初始化函数，功能主要是设置定时器的参数；二是定时器中断函数。功能主要是对当定时器寄存器计数满时触发中断，来处理的一些程序指令。

定时器初始化函数如图 3.3 所示，首先使能定时器时钟，计数模式设置为向上技术模式，设置定时器中断抢占优先级 0 级，从优先级 3 级。定时器初始化函数 TIM3_Int_Init()有两个形参，arr 用于设置定时器周期，psc 用于设置定时器的计数频率。

```
void TIM3_IRQHandler(void)   //TIM3中断
{

  if (TIM_GetITStatus(TIM3, TIM_IT_Update) != RESET) //检查指定的TIM中断发生与否:TIM 中断源
    {
    TIM_ClearITPendingBit(TIM3, TIM_IT_Update  );  //清除TIMx的中断待处理位:TIM 中断源

       switch (m_Channel)
        {
        case 1:  if(time_flg<m_PulseTime) {GATE1=0;GATE1_2=1;}
                 else {GATE1_2=0;GATE1=1;} break;
        case 2:  if(time_flg<m_PulseTime) {GATE2=0; GATE2_2=1;}
                 else {GATE2_2=0; GATE2=1;} break;
        case 3:  if(time_flg<m_PulseTime) {GATE3=0; GATE3_2=1;}
                 else {GATE3_2=0; GATE3=1;} break;
        case 4: if(time_flg<m_PulseTime) {GATE4=0;GATE4_2=1;}
                else {GATE4_2=0;GATE4=1;} break;
        case 5: if(time_flg<m_PulseTime) {GATE5=0;GATE5_2=1;}
                else {GATE5_2=0;GATE5=1;} break;
        case 6: if(time_flg<m_PulseTime) {GATE6=0;GATE6_2=1;}
                else {GATE6_2=0;GATE6=1;} break;
        default: break;
        }
        if(time_flg==0)
        {
            LED0=!LED0;
        }
        if(time_flg==m_PulseTime)
        {
            LED0=!LED0;
        }
        time_flg++;
        time_flg%=pin;
        if(mode==1 && time_flg==0 && START==1)
        {
          Print_Counter++;
          if(Print_Counter>=Print_Times)
          {
            TIM_Cmd(TIM3, DISABLE);
            START=0;
            Print_Counter=0;
          }
        }
    }
}
```

图 3.4 定时器中断函数。

定时器中断函数如图 3.4 所示，当定时器寄存器计数满时会触发定时器中断。进入中断函数后，首先使用库函数 TIM_ClearITPendingBit()函数清空计数寄存器，清除中断标志位，对定时器进行复位操作。接着是一个 switch 语句，根据串口接收的通道标志位 m_channel，例如 m_channel=1，进入 1 对应的语句，在 time_flg＜m_PulseTime (方波高电压脉宽时间)时，GATE1=0;GATE1_2=1；此时压电致动器处于充电弯曲变形状态。当 time_flg ≧ m_PulseTime 时，GATE1_2=0;GATE1=1;此时压电致动器恢复变形的状态。这里值得注意的是，一定要把引脚置 0 的语句放在引脚置 1 的语句前面，这是为了防止存在短暂瞬间两个引脚均为置 1 的状态，根据图 2.9 的 MOS 管的开关电路可知，这会导致短路的现象，可能会引起 N-MOS 电流过大而烧坏。Switch 语句后是一系列的判断处理程序，主要用于工作模式、起始位、打印次数以及打印周期的判断处理。

DAC 模块：

```
void Dac2_Set_Vol(u16 vol)
{
  float temp=vol;
  temp/=1000;
  temp=temp*4095/3.3;
  DAC_SetChannel2Data(DAC_Align_12b_R,temp);//12位右对齐数据格式设置DAC值
}
```

图 3.5 DAC 函数。

DAC 函数如图 3.5 所示。STM32 的 DAC 模块是 12 位数字输入，电压输出型的 DAC。我们使用的 STM32F103RCT6 的 DAC 模块的参考电压直接来自其 VDDA 引脚的供电，也就是稳压芯片输出的 3.3V。DAC 的输出是受 DORx 寄存器直接控制。DAC 的输出电压是线性的从 0~3.3V，12 位模式下 DAC 输出电压与 3.3V 以及 DORx 的计算公式如下：

$$DAC\ 输出电压=3.3V*(DORx/(2^{12}-1))$$

由于稳压芯片实际输出的电压值与 3.3 V 会存在细微的偏差，因此还需要通过示波器测试不同 DORx 值与实际开关电源输出电压的关系，然后使用最小二乘法进行线性拟合得到以下关系式：

$$DORx = 实际输出电压\ m_Voltage*11.183-10.427;$$

主函数部分：

```
int main(void)
{
   delay_init();        //延时函数初始化
   NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);// 设置中断优先级分组2
   uart_init(9600);   //串口初始化为9600
   TIM3_Int_Init(9,7199);//10Khz的计数频率，计数到5000为500ms
   LED_Init();
   KEY_Init();
   Dac2_Init();
   Adc_Init();
   TIM_Cmd(TIM3, DISABLE);//关闭定时器
   pin=(1000/m_Frequency);
   v=m_Voltage/88.0;
   vol=(v/3.3)*3300-5;
   Dac2_Set_Vol(vol_min);//DAC输出函数
   GATE_Init();
   while(1)
   {
       ReceiveDate();//数据接受函数
       if(START==1)//fly模式触发
         {
           while(START==1)
             {

               TIM_Cmd(TIM3, ENABLE);
               ReceiveDate();   //读取串口接受数据
               if(START==0){TIM_Cmd(TIM3, DISABLE);break;}
             }
         }
    }
}
```

图 3.6 主函数。

主函数是 MCU 启动后进入的执行函数，其程序如图 3.6 所示。首先是系统

中断优先级、串口函数、定时器以及其它函数的初始化。然后进入 while 循环，不断执行串口接收函数，判断起始位 START 是否为 1。当 START=1 时，使能定时器进行计数来触发定时器中断以执行输出所需的电压波形。

3.2 上位机程序设计及说明

开发的多通道微流控冲击打印系统上位机控制软件界面如图 3.7 所示。其主要由四部分组成：第一部分为通道选择区，其主要功能为选择所需要工作的通道并设置其试剂以及单液滴体积参数；第二部分为驱动波形设置区，其主要功能是设置所选择的每个通道对应的压电致动器驱动电压信号的频率、幅值和脉冲宽度时间参数；第三部分为液滴阵列设置区，其功能为设置所需要的液滴阵列参数，如液滴阵列行列数、每个位置点的液滴数；第四部分为打印起始控制区，其功能为控制液滴阵列打印的起始、选择液滴阵列打印模式和液滴阵列间距设置。该控制软件是基于 Visual Studio 软件平台使用 C#语言编写而成，可视化用户交互界面如图 3.7 所示。

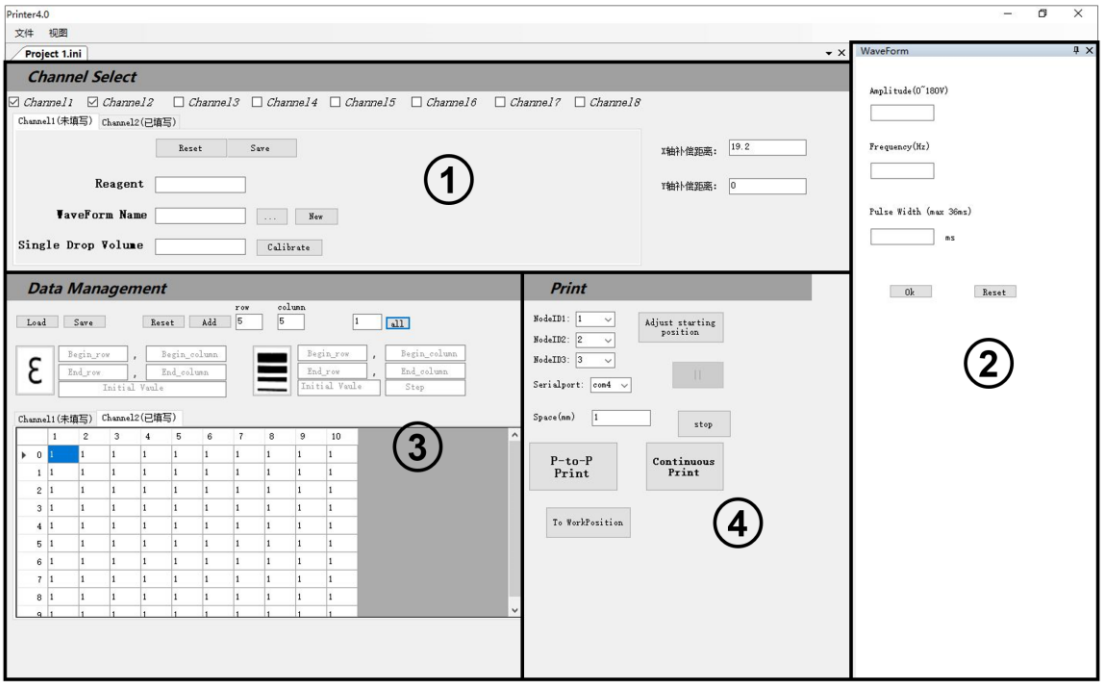


图 3.7　多通道微流控冲击打印系统控制软件界面。

4、打印头机械结构及装配方法

4.1 多通道打印头结构设计

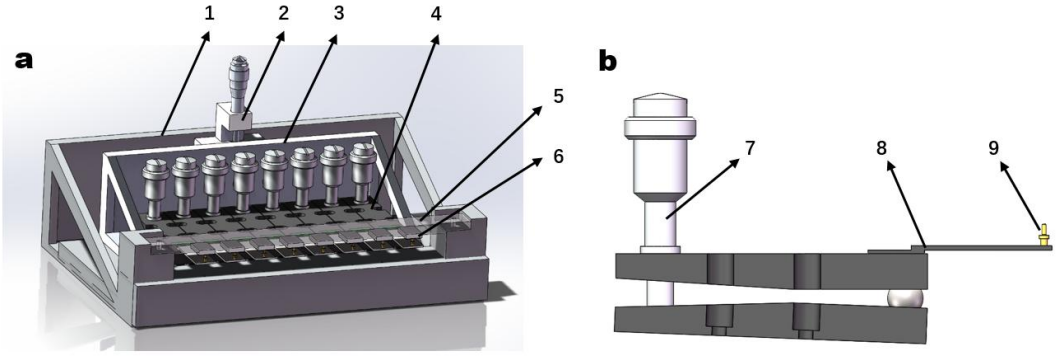


图 4.1 (a) 8 通道打印头装配示意图；(b) 单通道装配示意图。(1-打印头底座；2-移动台；3-俯仰台固定底座；4-单通道转配体；5-微流控芯片支架；6-微流控芯片；7-旋转螺杆；8-压电致动器；9-打印针。)

我们设计了如图 4.1a 所示的 8 通道打印头，底座 1 是用来装夹微流控芯片与连接到移动台的；移动台 2 是用来调节 8 个通道打印针与芯片之间间距的，顺时针旋转螺杆打印针与芯片之间间距会减小。如图 4.1b 所示，俯仰台 4 可以通过旋转螺杆 7 来调节末端打印针 9 的高度。由于压电致动器之间装配在高度方向有误差，因此设计了俯仰台来进行高度方向误差补偿，以保证 8 个通道之间打印针高度的一致性。下面对打印头的装配流程详细说明一下。

4.2 多通道打印头装配方法

俯仰台的夹紧与定位：

如图 4.2 安装示意图所示，先用底面凹槽里的螺纹孔用于固定俯仰台；随后，再用凹槽前面的平面用于俯仰台的该零件用于螺纹固定的底面存在 2.25 度的倾斜角，用于补偿俯仰台螺旋千分头位于中间刻度位置时俯仰台的倾斜角。最后将压电梁用螺钉安装在俯仰台上，拧紧固定好。

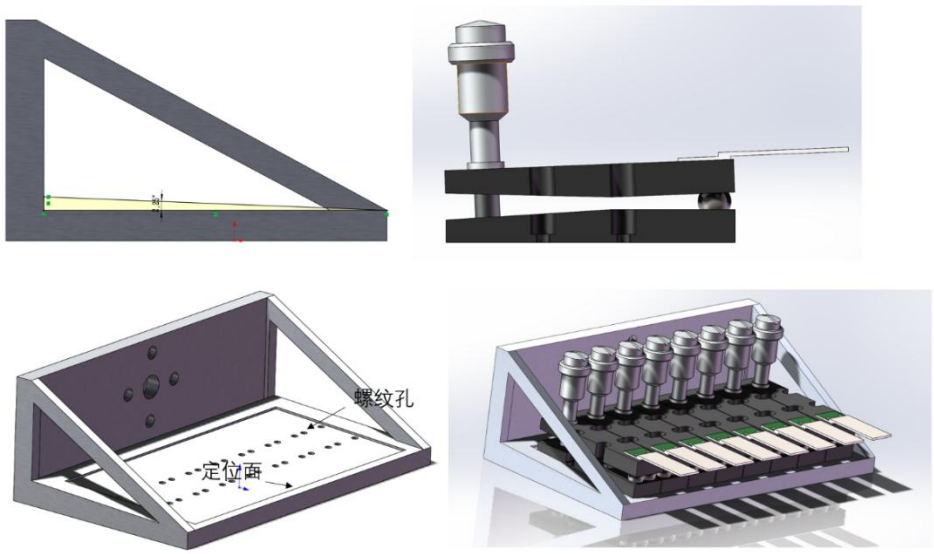


图 4.2 俯仰台安装示意图。

打印针的安装：

为了方便微流控芯片装载液体，因此需要将针安装在梁的上表面。打印针安装方法如图 4.3 所示。首先，将金属针用双面胶粘在夹具的定位孔中。随后，在压电梁的末端上表面贴上 3M 双面胶。接着，将安装好的 8 个俯仰台的装配体与安装针的夹具配合连接在一起。最后，通过调节俯仰台，使得梁的上表面刚好与针底部相接触，使针粘接在梁的末端。

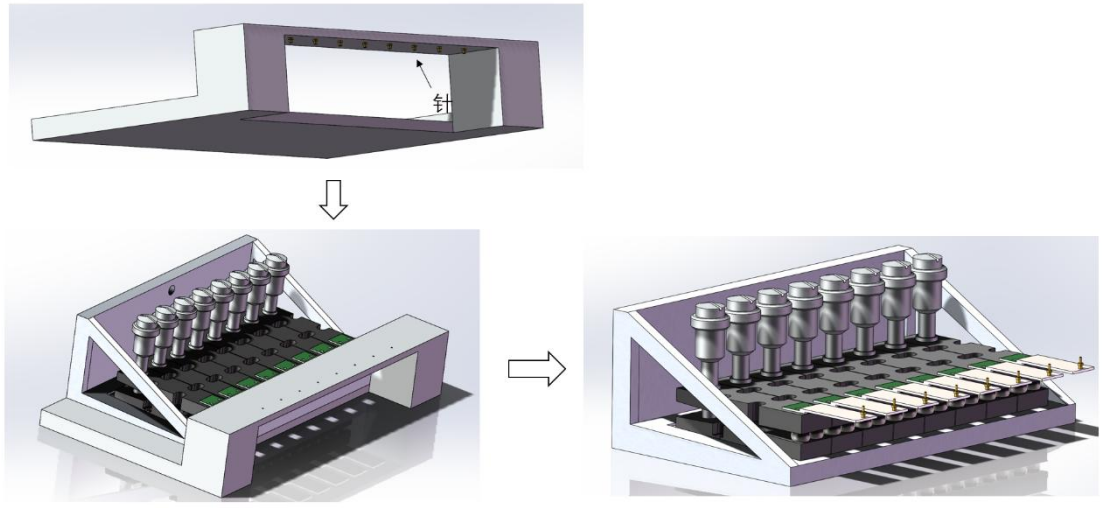


图 4.3 打印针安装示意图。

打印头组装：

先将上一步装配好的装配体与移动台连接固定在一起，再把移动台与大底座固定连接在一起。俯仰台的底座与大底座安装移动台的侧面上都有一个凸起

的长方体，用于移动台的定位，其目的是为了保证安装好后的两个底座底面平行。

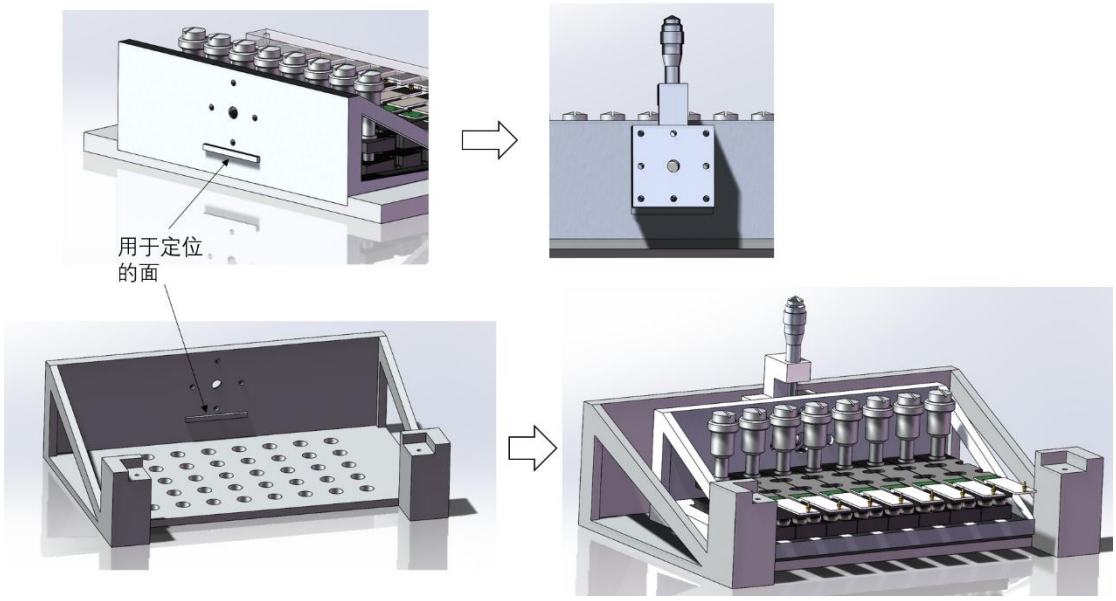


图 4.4 打印头组装示意图。

芯片安装方法：

芯片安装主要分两步，一是芯片的注液与粘接在亚克力基板上，二是将粘接好芯片的亚克力基板用螺钉安装上去。具体操作如下：

1、首先将待打印液体注入微流控芯片，将液体充分填充微管道，如图 4.5 左上所示。亚克力芯片基板如图 4.5 右上所示，表明使用激光刻有芯片轮廓图，用于芯片粘接时位置对准。在亚克力芯片基板粘贴一层 3M 双面胶后，将注液后的芯片粘接到亚克力基板对应位置，如图 4.5 下方所示。粘贴过程中要确保芯片的敲击腔位置与基板刻线的敲击腔轮廓线对准。

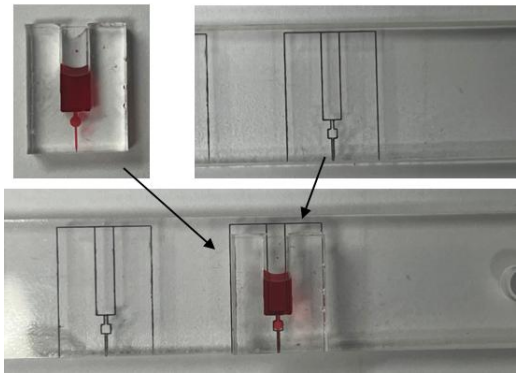


图 4.5 微流控芯片安装。

2、将粘接好芯片的亚克力基板，使用螺钉安装到打印头对应位置，如图 4.6 所示。

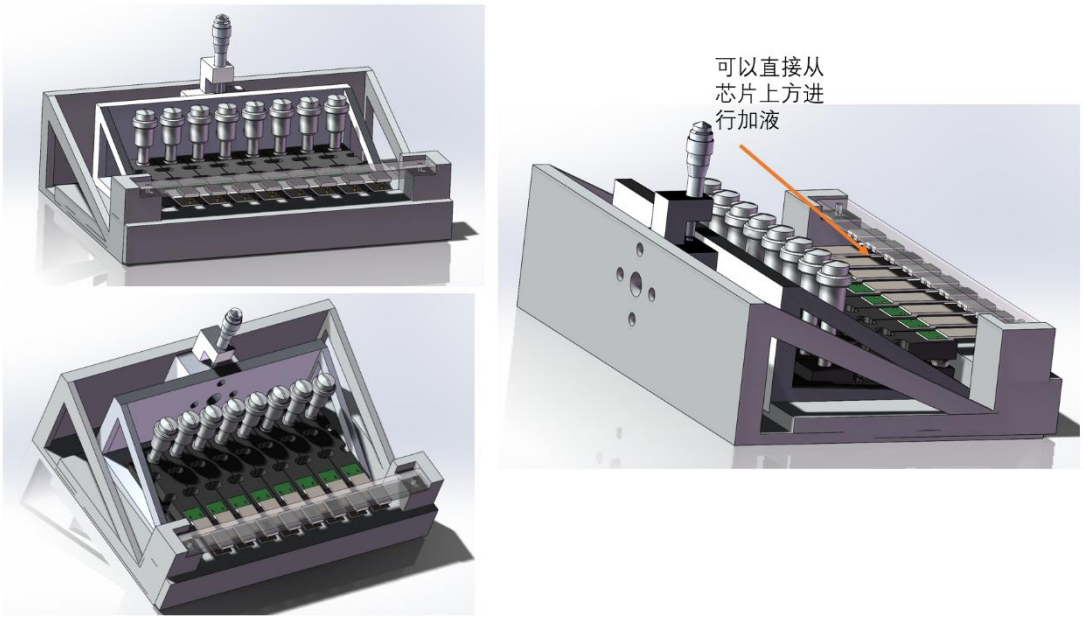


图 4.6 芯片安装到打印头示意图。