# 计算机组成原理实验报告

计 22 班 201 组

王天一 袁源 吴鹏和

## 项目概述

### 实验目的

**实验工具**

实现以 16 位 THCO-MIPS 指令集为核心的基础计算机系统，CPU 采用流水线结构。实现 usb 串口、时钟中断、多道程序、VGA、小软件实现等拓展。

### 实验环境

操作系统：windows 7

编程环境：ISE Design Suite 14.7

编程语言：Verilog + VHDL

### 实验效果概述

计算机系统能在 12.5M 下稳定运行，能通过全部的性能测试。

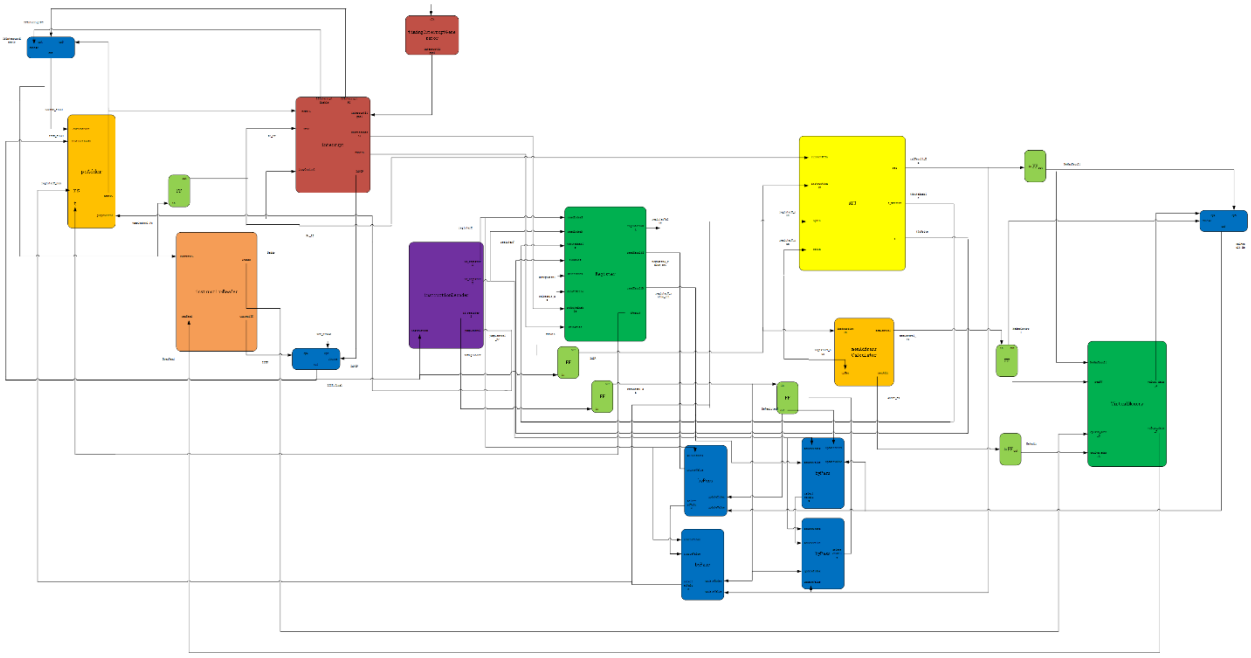
实验拓展了 USB 串口，并将两个串口同时与终端连接。通过时钟中断的处理，能以多进程的方式同时使两个 term 运行。实验通过了 6.25M 以上的性能测试以及自行设计的软件。

实验拓展了 VGA 接口，将终端所输入的 ASCII 字符串输出到显示器上。

我们还提供了八数码、打字机、刷屏游戏等小软件程度用于测试我们的系统。

# 主要模块设计

## 数据通路图



大图见附录

下面逐一每个模块进行逐一说明：

### pcAdder

该模块在每个下降沿读入指令和当前 PC。

而下一阶段的 PC 的运算是由组合逻辑实现的，它由 ID 模块解码出来的跳转信号控制，若为几个跳转指令之一的话，则一经 ID 阶段的解码就会改变 pcAdder 计算的下一条 PC，而控制信号产生时间是在上升沿，所以取址尚未发生。

### instructionReader

在上升沿读取存在 ram 中的指令，即使上一条是跳转在下次取址前改变 nextPC

### instructionDecoder

在每个上升沿进行解码，主要作用是产生跳转控制信号，解码有关寄存器号。

### registerFile

一方面，使用组合逻辑，直接从寄存器堆里读出 ID 解码的寄存器号相应的值。

另一方面，在下降沿处理 WB 阶段事宜，即赋值。

### alu

从 registerFile 读完操作数后，在下一个阶段的上升沿进行 alu 运算。

为了减少控制变量的数量，我们依然在此直接输入指令并在 alu 内部进行解码。

### addrCalculator

与 alu 是并列的，同在上升沿进行地址运算，只是该模块的输出信号与虚拟内存模块有联系。

### flipflop

触发器，由于我们两个阶段之间都有一些数据的交互，需要被暂存一段时间，所以需要 flipflop 暂存。

下降沿的时候读入，上升沿时输出。

### bypass

这里主要是为了把 alu 计算结果和 mem 取到的值快速写会寄存器堆。

由于数据冲突会发生在下两条指令以内，故旁路的主要工作则是在被写回的寄存器在下条指令被又被读的之前将值改变。

该模块处理两个被操作的寄存器，以及 t 寄存器。

### VirtualMemory

这个模块把物理内存和串口封装了起来，与 cpu 本身交互的部分是 instructionReader 和 addrCalculator。

为了解决结构冲突，我们采用的方案是 IF 和 MEM 阶段共用同一块内存，每一个阶段获得一半的访问时间。具体说明见内部存储设备一节。

### interrupt

中断处理部分请看下一章。

### vgaOutControl

用于产生控制 vga 的信号。

## 中断设计详解

我们实现了时钟中断，其职能为每过 1000 个周期触发中断，通知操作系统切换进程。

它主要包含两个模块：

1. 中断裁定模块，它负责进行中断的处理
2. 中断产生模块，它负责每过固定时间产生一个高电平

我们添加了一个新的寄存器:RN 寄存器，它负责保存中断返回地址

我们添加了三条指令，这三条指令不包含在本组要求的指令系统中，所以我们将这些指令作为中断处理指令使用：

1. JALR Rx 指令，这条指令的作用为将中断返回地址移入 Rx 通用寄存器。
2. CMPI Rx 指令，这条指令的作用为将 Rx 通用寄存器的值作为中断返回地址。
3. Int 指令，这条指令的作用为跳转到中断返回地址，注意到这条指令存在延时槽。

中断处理流程如下：

中断裁定模块探测到中断信号高电平之后，判定当前在 ID 阶段执行的指令是否为跳转指令。

如果 ID 阶段正在执行的指令为跳转指令，那么 IF 阶段正在执行的指令为一个延时槽，所以它需要被完全执行。中断仲裁模块此时命令 ID 模块下一个周期运行 IF 得到的指令，并且将 ID 解析出的跳转地址移入中断返回寄存器 RN。

如果 ID 执行的指令不是跳转指令，IF 阶段正在执行的指令不是延时槽，而他有可能是跳转指令，所以一定不能够被执行。此时中断仲裁模块命令 ID 模块下一个周期执行 NOP，并且将 IF 阶段的指令地址移入中断返回寄存器 RN。

以上两种情况的划分使得无论流水线中的情况如何，中断都能够被正确地处理，并且流水线中至多产生一个气泡。有兴趣的读者可以思考此设计正确的原因。

无论以上哪种情况，IF 在下一周期都会从固定的中断处理程序地址 0007 读取指令开始执行。

中断返回时，中断处理程序需要使用 CMPI 指令后使用 INT 指令，此时 CPU 会跳转到 Rn 寄存器指向的地址继续执行。

# 多进程操作系统

## 概述

通过普通串口和 usb 串口与两个 term 相连，以及在 cpu 中加入中断处理模块，并修改相应软件部分，再经过较为麻烦的单步调试，即可完成多进程操作系统。

## 进程切换机制

进程切换由时钟中断结合中断处理模块完成，主要分为现场保存和现场恢复两部分。

中断处理模块有两个数据保存区，分别保存被切换 kernel 和切换至 kernel 的通用寄存器 R0~R7、t 寄存器、栈顶寄存器 SP 和指令地址寄存器 PC。

发生时钟中断时，硬件系统会保存被切换 kernel 的 PC，并切换至中断处理模块。中断处理模块首先保存被切换 kernel 的 PC 和上述其它各寄存器至被切换数据保存区中，之后从切换至数据保存区中读出数据恢复各寄存器，并将被切换数据保存区内数据转移至切换至数据保存区，中间用栈进行辅助操作。最后跳转至切换至 kernel 的 PC，完成进程切换。

## 地址空间分配

本节参数操作系统的内存地址划分，如果要查看整个硬件空间地址映射，请参阅这一节。

# 内部存储设备

内存访问速度是没有 Cache 的 CPU 的瓶颈。同时内存的时序在高频下存在着很多不容易被注意到的问题。本节将阐述本项目内存访问结构的设计。

## 结构冲突

物理内存的访问存在结构冲突，由于 IF 阶段和 MEM 阶段要同时访问内存。为了解决这个问题可以采用 HAVARD 架构等解决方案。我们采用的方案是 IF 和 MEM 阶段共用同一块内存，每一个阶段获得一半的访问时间。我们在时钟的上沿进行 IF 阶段内存的使用，在时钟的下沿进行 MEM 阶段内存的使用。

## 地址映射

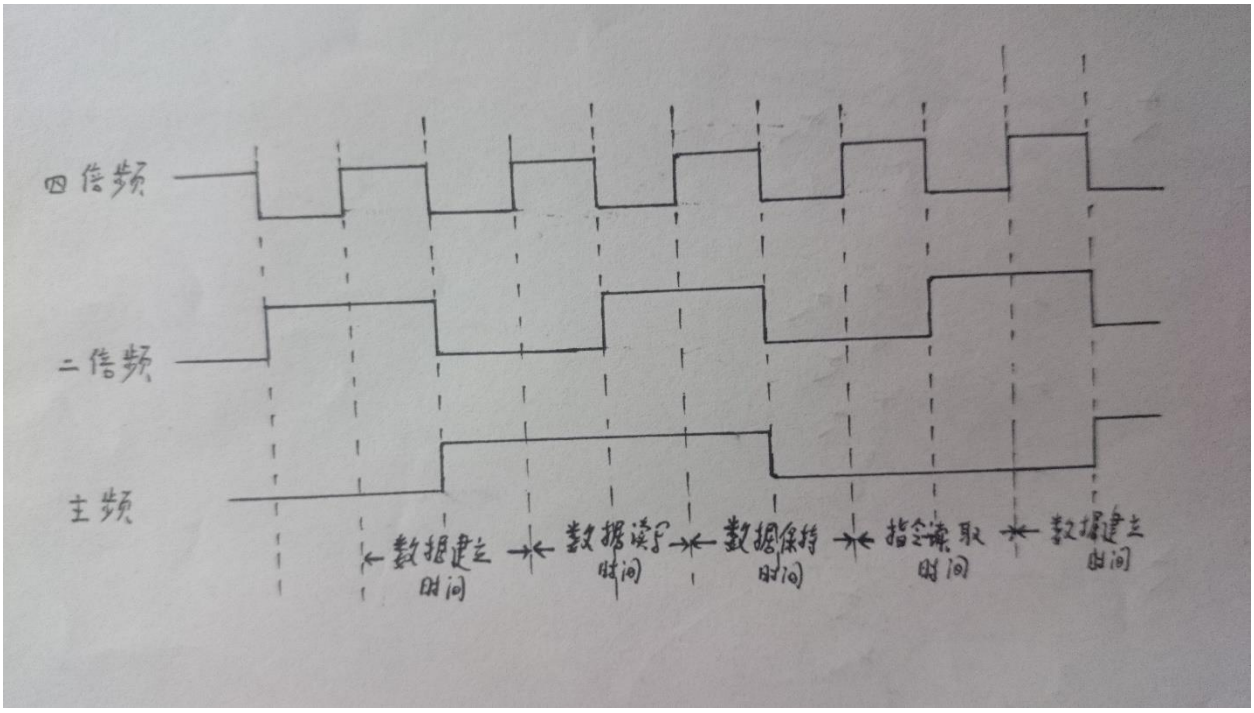
本节阐述硬件地址空间的地址映射关系，如果要查看操作系统的内存划分，参阅这一节。

| 地址 | 功能 |
| --- | --- |
| BF00 | 串口 1 数据总线 |
| BF01 | 串口 1 标志字节 |
| BF02 | 串口 2 数据总线 |
| BF03 | 串口 2 标志字节 |
| BF04 | 标准输出（VGA）地址。 |

## 物理内存时序

经过一些严重的内存读写问题之后我们认识到内存的访问需要严格遵守数据建立时间、数据保持时间等时序，所以我们对内存的时序做了非常保守的设计。

我们称 clk 为 CPU 的主时钟，clk2x 为其两倍频时钟，clk4x 为其四倍频时钟，每一个时钟由其二倍频时钟的下降沿生成。




我们使用每一个内存周期的中间二分之一时间来读写内存。在每一个内存周期的开始，内存控制器向内存给出地址和数据，在每一个内存周期的四分之一位置给出读写信号，在每一个周期的四分之三位置撤除读写信号。注意到这个过程可以通过在四倍频时钟的上升沿时判断 2 倍频和 1 倍频的组合来完成。

这种实现方式保证了内存读写的数据建立时间和数据保持时间，在频率容限内是完美的解决方案。此方案可以让 CPU 在 12.5M 下稳定运行。

# 外部设备

## 串口

除了串口一之外，我们利用 USB 接口实现了第二个串口并将其映射到地址 BFO2 和 BF03。此串口具有和串口一完全一致的性质和职能。

### VGA 模块

VGA 模块能够在屏幕上显示指定的字符串，能够在屏幕上显示 40*32 个字符。我们将其映射到内存地址 BF04，程序只需要向此地址写入数据即可在屏幕上显示字符串，并且可以连续地写入，无需等待。

## 软件设计

我们设计了若干个场景来展示本系统的强大职能。

### 交互式终端

我们修改了终端程序 Term 的源码，以实现与用户程序的实时交互。具体为：每当用户程序需要输入，则先向 Term 发送一个输入请求”0x06”，终端收到”0x06”后便从标准输入流中读入一个字符并发送到教学机。由此完成程序与用户的交互。

### 八数码游戏

经典的八数码游戏。程序会打印一个九宫格，由数字 1 到 8 和一个空格组成。用户可以输入’w’、’s’、’a’、’d’对空格附近的字符进行上下左右移动，经过若干次移动后，若九宫格内字符以目标次序排列则过关成功。程序使用到的全部指令为基本指令集+本组扩展指令 SLTI。

### 打字机

程序循环向终端发出输入请求并等待终端输入，每收到一个输入便将其送往 VGA 映射地址 0xBF04 进行输出。运行效果为：用户可以在任意时间向终端输入任意字符，且其输入的字符会实时显示在 VGA 屏幕上。

### 刷屏程序

程序会循环输出字母 a-z。两个终端同时运行刷屏程序，可以看到两边分别不停地输出字母 a-z，且进度略有差异，由此验证两个终端运行时互不干扰且具备视觉上的同步性。

## 效果演示

计算机系统能在 12.5M 下稳定运行，能通过全部的性能测试。

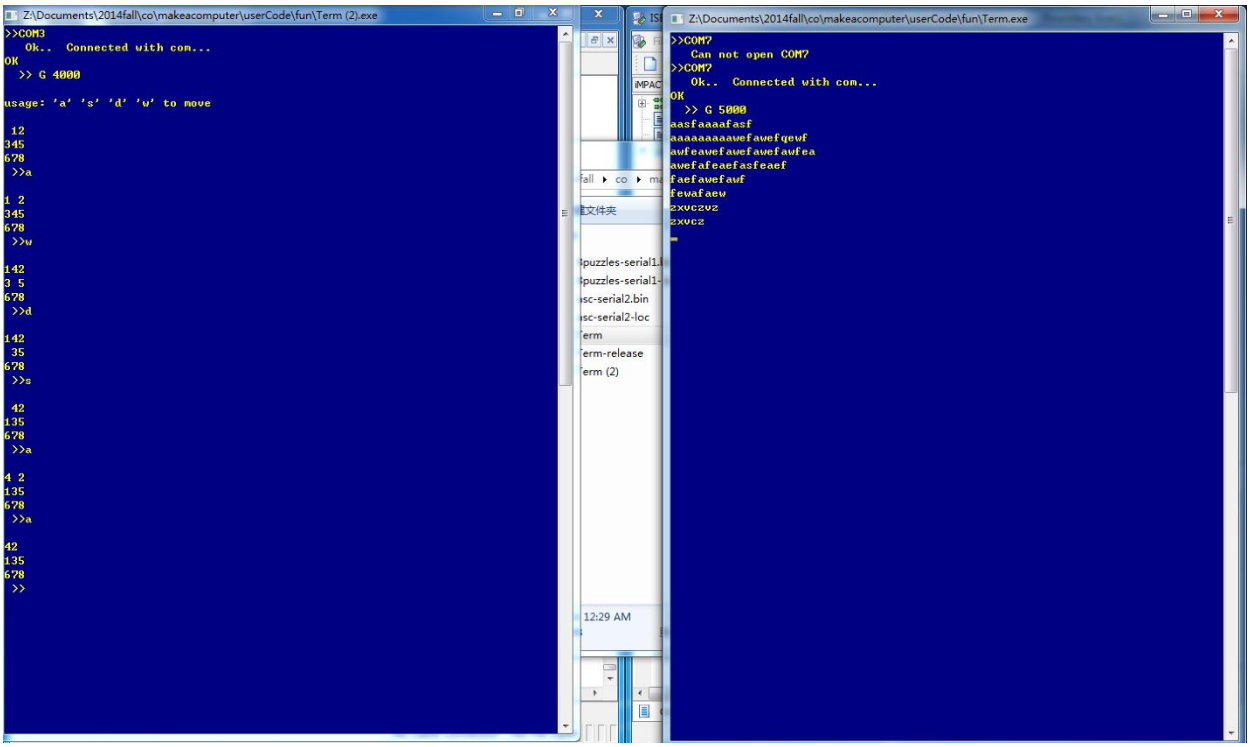
实验拓展了 USB 串口，并将两个串口同时与终端连接。通过时钟中断的处理，能以多进程的方式同时使两个 term 运行。实验通过了 6.25M 以上的性能测试以及自行设计的软件。

实验拓展了 VGA 接口，将终端所输入的 ASCII 字符串输出到显示器上。

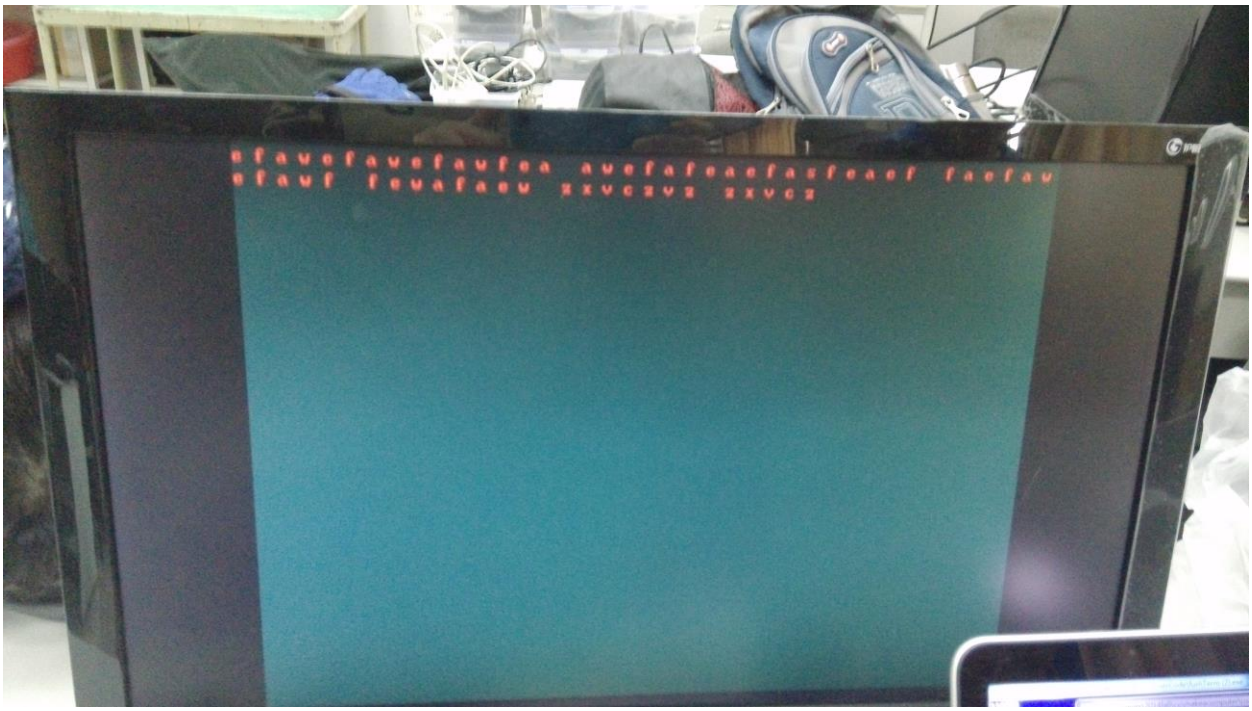
我们还提供了八数码、打字机、刷屏游戏等小软件程度用于测试我们的系统。

下面是我们几个综合性的演示：

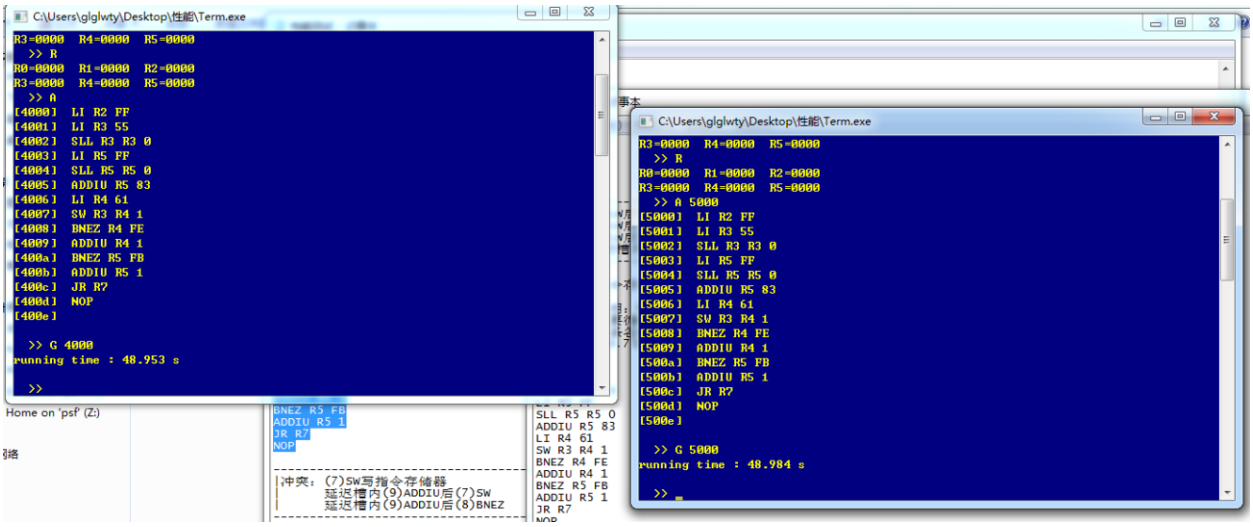
## 两终端分别运行八数码和打字机（VGA 拓展）




此时显示器显示



## 两终端同时运行性能测试




可以看到，在 6.25M 的频率能够稳定运行。

# 实验心得体会

本次大实验让我们收获颇多。

1. 我们对组成原理课程知识的理解大大加深。起初我们觉得有了课上刘老师所教的理论知识业已足够让我们清晰地设计我们的数据通路。然而，事实并非如此。我们组员讨论了一整个晚上，反复地讨论、修改我们的数据通路。几经大改，我们才完成了我们的最终版本。
2. 我们硬件语言编写能力有所增强。起初我们组员达成一致，就是实验主体用 Verilog 硬件描述语言（usb-urat 部分由于是直接复制的网络学堂提供的版本，故用 VHDL 描写）。虽然 Verilog 代码比较简洁，但是却带来的另外的麻烦：其容错性过强，甚至会出现变量名打错之后编译器直接生成新变量而不报错的情况。
3. 我们的硬件调试能力大大增强。起初我们的访存控制信号设计不够完善，这导致了单步调试中断处理十分麻烦——因为会出现很多随机的错误。经过夜以继日的劳动，我们终于能有幸在大作业检查日期之前将中断部分的数十个 bug 逐一调出并且还发现了访存的问题。

4. 硬件查错能力有得到了极强的锻炼。经过起初的教训，我们养成了查错必先查硬件错误的习惯，没发生一些异常情况必先检查两个串口以及导线是否有问题。
5. 我们的团队协作、沟通能力大大提高。我们一共三位组员，而实验设计本身较难以划分职责，所以实验过程中沟通极其重要。我们使用 github 进行版本控制，这为我们的沟通提供了很大便利。

最后我们要感谢刘卫东老师、李山山老师以及各位助教对我们悉心的帮助和指导。