“龙芯杯”系统能力培养大赛心得总结

作者 江学谦 朱威浦
北京理工大学 计算机学院 2015 级

目录

2018 年 10 月 8 日

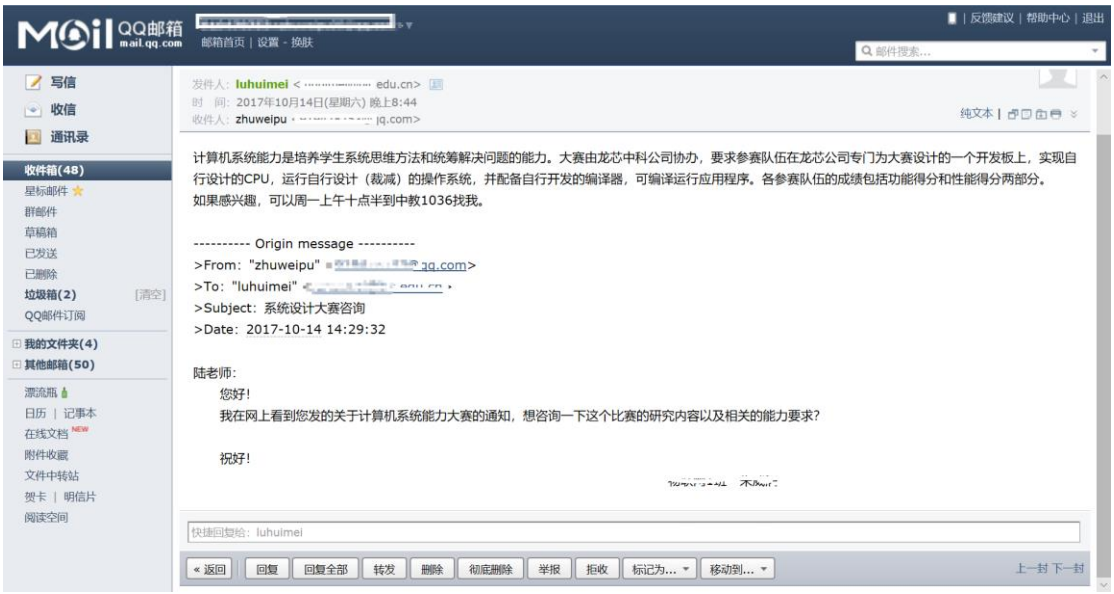
前言

第二届“龙芯杯”系统能力培养大赛已经结束，回想长达一年的准备，点滴心血皆历历在目。虽然最终没能取得很好的成绩，但无论如何，这个过程的收获颇丰，准备过程中也确实遇到过很多障碍，故仅能以此文记录参赛过程的琐碎杂事，权当以另一种形式展现自己的心路历程。

第一章 漫长的赛前准备（上）

第一节 吹响集结号

2017 年 10 月份，大三上学期的小学期课程刚结束不久，便看到导员帮陆老师在年级群里发布的系统能力培养大赛的通知。想来刚开学也闲来无事，好不容易搬回村里来了，应该多给自己找些事做做。抱着张张见识的心态，我联系了陆老师，想要了解一下比赛情况和内容。

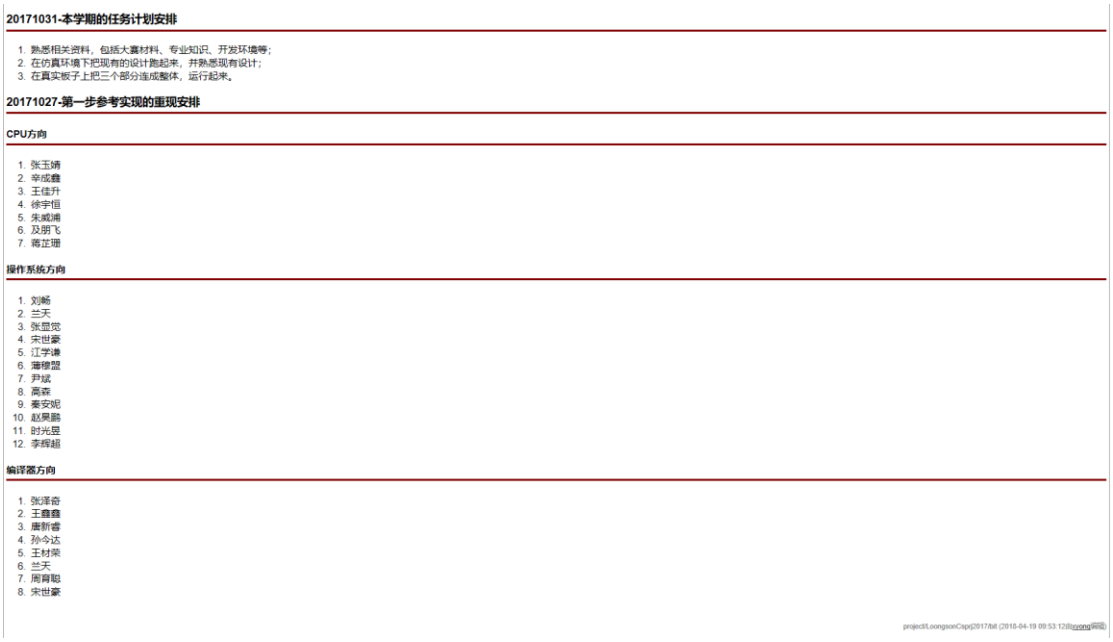


起初看到陆老师的回复，便隐隐感觉到这个比赛并不好做，因为对于当时刚上大三的我来说，对于陆老师所提到的 CPU、操作系统、编译器等专业方面的知识都一概不知。

周一早上我如约去到陆老师的实验室，等待了一会，陆续又有其他同学也来到了实验室，还有的同学干脆把第二节课翘掉急忙赶来。陆老师先是和我们寒暄了几句，询问一下个人的基本情况，想确认一下同学参加比赛的意愿和决心。陆老师是一个十分亲和的人，虽然第一次见面，但是聊起天来却没有压力。在她的介绍中，我得知这个比赛是第二次举办，我们学校还是第一次参加，没有前人的经验可以借鉴。听完后我们都表示还没有接触过相关方面的知识，不知道如何着手，陆老师便安慰我们说，现在手上已经有一个能够运行的作品——清华上一届获得比赛第一名的作品，而且清华大学的向老师也会过来指导我们的比赛。听到这样的话大家的心里才稍微有了点底子。

开始时报名参加比赛的同学很多，前前后后加入比赛微信群的人数一度高达二十几个。按照目标，我们需要涉及到 CPU、操作系统、编译器等三个方向的知识，所以老师一开始便

让我们先有所侧重，选择自己感兴趣的方向来复现实现——运行已有的实验作品。我个人比较喜欢做硬件，做 CPU 听起来也十分有趣，所以就选择了 CPU 方向作为参加比赛的侧重点。



20171031-本学期的任务计划安排

1. 熟悉相关资料，包括大赛材料、专业知识、开发环境等；
2. 在仿真环境下把现有的设计跑起来，并熟悉现有设计；
3. 在真实板子上把三个部分连成整体，运行起来。

20171027-第一步参考实现的重现安排

CPU方向

1. 张玉婧
2. 辛成鑫
3. 王佳升
4. 徐宇恒
5. 朱威浦
6. 及朋飞
7. 蒋芷珊

操作系统方向

1. 刘畅
2. 兰天
3. 张显宽
4. 宋世豪
5. 江学谦
6. 薄穆盟
7. 尹斌
8. 高森
9. 秦安妮
10. 赵昊鹏
11. 时光昱
12. 李辉超

编译器方向

1. 张泽奇
2. 王鑫鑫
3. 唐新睿
4. 孙今达
5. 王材荣
6. 兰天
7. 周育聪
8. 宋世豪

projectLoongsonCup2017/bit (2018-04-19 09:53:12)[liyong]

第二节 自学新知识

我觉得老师可能默认了我们学校会提供流水线设计方面的课程，但是其实并没有。计算机体系结构课程中会对流水线的概念做出解释，但是完全不涉及到如何实现。计算机组成原理讲解计算机的构成，操作系统介绍操作系统的构成和运行原理，汇编接口学的是 x86 体系，编译原理把课程重心完全放在了编译器前端，讲的都是词法分析和语法分析的原理，学完还是不知道 gcc 为何物。实现 MIPS CPU 是大四小学期的实验内容，单周期和多周期都有涉及，但是没有到达流水线的层面。于是要做比赛，就只能自学。确定了要参加比赛后，我便开始着手 CPU 设计方面的知识学习。网上看到好评还不错一些书——《自己动手写 CPU》、《CPU 自制入门》等，都是适合新手从零起步直接去做一个 CPU，查了一下图书馆有《自己动手写 CPU》的馆藏，所以就直接借了这本书回来作为入门学习的书籍。500 多页的一本书，摆在书架上比我其他任何一本书都要厚实得多，草草翻看了几页，觉得书本的知识还是挺通俗易懂的，麻烦的是内容实在是多，需要花很长时间才能看完。我干脆把它摆在了床边作为睡前读物，就这样，在一年的时间里，我从头到尾把这本书读了两遍，心里面对 OpenMips 这个开源的 CPU 熟的不能再熟。当然也通过这种方式认识到了 FPGA，Verilog，MIPS 指令架构，流水线设计等基本的知识。但是这样的学习方式却给我带来很多的局限性，比如对于 CPU 设计方式、MIPS 整体架构的认识、SoC 设计的狭隘等诸多问题，这个我会在往后再详细介绍。不管怎样，这本书的学习过程还是让自己产生了设计 CPU 的兴趣，增加了做比赛的信心。

第三节 定期开组会

比赛准备工作的安排都交由向老师负责，每两周固定开一次例会。每次都是向老师从清

华过来主持会议，原以为从清华来北理工也是一件容易事，直到我去清华参加暑期小学期，得每天 7 点就坐半个小时公交车兜兜转转才能去到清华东门时，才知道这是多么不容易的一件事。

第一次正式开会是一个周六的上午，第一个学期的例会都是在机房 103 举行的，毕竟开始时人多，其他地方都坐不下。当时我完全忘记了老师在微信群里发过的消息，闹钟都没定就错过了。幸好其他同学都过去了，会议也如期举行，会后有同学把我拉进了 CPU 小组的讨论群里，大概了解了一下会议的内容就是确定每个小组的工作内容，CPU 工作小组的内容是配置环境，由于大家都没有接触过 FPGA 设计，对于 Xilinx、Vivado 这些名词一点儿也不熟。同组的同学说隐约记得老师让我们装一个叫做 V 什么的软件。。。计算机专业的比赛对环境的要求就是简单，一台电脑、一个软件就能搞定所有事情。当然如你所料，连软件的名字都没有弄明白的人，怎么可能会配得好环境呢！

第二次开会，也是我第一次去参加例会，人还是比较多的，当然已经不是报名参加比赛时候的人数了。向老师询问到我的进展情况时我说自己仅了解和学习了一些 Verilog 语法知识。当时对比赛内容还是不太熟悉，其实并不知道接下来自己要干些什么事情，除了配置环境。

向老师一开始给我们定的目标是希望能够借着清华同学的实验成果运行一次实验内容，负责每个部分的同学先将自己所负责的任务弄通，然后写配置过程文档，作为其他组复现实验的依据，以减少复现实验所花费的时间，也降低一部分难度。无奈理想与现实都是有差距的，我记得在往后数次会议中，在向老师不厌其烦地向我们阐述这一个指导思想后，我才终于弄明白复现实验是什么意思——不是要我们马上照着别人的代码马上就写一份，而是先把前人的结果直接拿过来跑一次。

第四节 麻烦的软件安装

第一次和 Vivado 打交道，是在第二次会议结束后，我和其他几个小组的成员计划着先下载 Vivado 的安装包。本以为安装个软件没有什么问题，谁知上网一搜索，一个安装包整个 20G，摸摸我的钱包，这个月的网费怕是交不起了。当然这还是小意思，只是以这个校园网的速度，要把安装包下完，估计比赛都结束了吧。。。只好忙着上 pt 站搜索一下，几 M 每秒的速度还能接受，断断续续有人在做种，下完 2017.1 版本也用了好几天。下完后，我直接在我的笔记本 Windows 系统下安装了这个软件，花了整整一节课的时间，软件都装完了，打开时却和 VS 版本不兼容。这个软件需要一个低版本 VS 的库，但是我已经安装了一个更高版本的 VS。上网搜搜博客，刚好也有解决方案，动手试了两个方案还是失败了，一狠心卸载 VS，然而也并没有什么帮助，最后就只能转战 Linux。Vivado 在 Linux 下综合运行起来比较快，但是其实在我后来的实验中用了台式机 Windows 下装的 Vivado，并没有感受到太大的区别，因为项目本来就没多大。而且 Windows 下可以自动安装驱动，Linux 下要让系统识别实验板还得配权限（我的文档里面有写），很是麻烦。

由于之前我接触过的 Linux 系统太少，基本上只用过 Ubuntu 下的几个命令，只会用 gedit 的人还能指望他会装软件吗？开始时我在 Windows 下装了一个虚拟机然后又装了一个 Ubuntu，没搞清楚需要配置多大的空间重复来重复去，就这样两周时间过去了软件还是没有有安装好。第三次开会时只能照实交代没有安好软件，便出现了我们花了一个月都没把软件安好的梗，也表明了今后我们进展的缓慢。原本定为第一个学期期末就能够把实验复现的目标，硬生生到了第二个学期期末才算真正的搞定。好在后来还是把软件安装好了，也写了几篇文档放到仓库里压箱底。

第五节 按时更新 Wiki

向老师在微信群里给我们发了一个清华大学 Wiki 的注册链接，里面有许多清华计算机课程的资料，也专门开设了一个页面以维护参加比赛的资源和日记情况。也不知道自己在这一年里为这个页面贡献多少资源和日记，只知道它硬生生把我从一个喜欢用图形编辑界面的人变成一个习惯用文本方式编辑的人，因为网页的图像编辑界面实在不好用。

除了每两周一次的会议，向老师建议我们要把每次的进展情况更新到 Wiki 上面，至少每周一次。对于不爱记录东西的多数人来说还是难以养成习惯，时时要老师提醒以后才会去更新 Wiki 上的日志。不过到最后实践证明定时更新 Wiki，维护日志，写文档都是有好处的。人的忘性实在太大，今天的我也许就忘了昨天开会时布置给我的任务是什么了，所以开会时都得拿个小本子记录简明扼要地记录一下。还有之前做过的实验复现过程现在都已经忘记的差不多了，好在再看一眼当时写下的文档就能轻松地再做一遍。如果当时没有形成文档，现在估计又得重头来过，水过鸭背，什么也没留下。

主要包含：

1. 大赛实验箱校验工具与介绍、参赛可能用到的工具与介绍、SoC_demo与简单介绍
2. 功能测试源码和使用示例、soc_up的源码和介绍文档、func_test_v1.01（去除非延迟槽测试时延迟槽非nop情况）
3. 目录perf_test/（大赛提供的性能测试源码和使用说明）、目录ucore_thumips/（清华提供的小型操作系统ucore）
4. doc_v0.03/里，新增一个文档：预赛提交要求.pdf

大赛资源包

17.12.11 ~ 17.12.24

- 进展
  1. 完成实验箱文档的编写
  2. 将soc_lite中的project_1烧录进开发板中

17.12.11 ~ 17.12.24

- 进展
  1. 完成vivado新建工程和仿真过程文档的编写

17.12.3 ~ 17.12.10

- 进展
  1. 初步加载cpu指令部分代码，并运行成功。
  2. 遇到的问题： 运行过程中，软件直接闪退。还未找到原因
- 下周计划
  1. 对运行成功的代码进行仿真。
  2. 加载代码

第六节 clone 第一份代码

实验复现的代码都存储在 GitHub 上面，刚开始时并不怎么会用，看着页面上众多的链接，一会指向代码仓库，一会指向 Wiki，让我搞得很混乱。向老师真是手把手告诉我，页面里哪个链接里存放着哪些代码。然后让我回去下载复现。由于没有用过 git，我都是直接在网页上点击下载 zip 包，还会因为网速慢导致下载的代码不完整。直到后来被队友强烈安利了一波才慢慢学着去使用 git。于是我第一步就是把清华第一届比赛的作品——NaiveMIPS clone 下来。

第七节 尝试复现 CPU

复现 CPU 的实验并不困难，仅是将原本的项目下下来，用 Vivado 将工程文件打开即

可。由于一开始对于 NaiveMIPS 整个项目并不熟悉，下载下来的整个文件夹不仅有 Xilinx 的环境，还有 Altera 的环境，目录十分复杂，让我连工程文件都没有找到。于是第一次的尝试便是是自己新建一个工程，然后将源码一一添加进工程中去，然而这样是绝对行不通的，因为 NaiveMIPS 是用 block design 连接各个模块，并用到很多的 IP 核，所以直接加入源码并不能综合。后来向老师又给我们介绍了龙芯的开源代码 SoC_up，同样能够运行操作系统。这个项目的目录就简明易懂许多。于是我又转而尝试运行龙芯的开源 CPU 代码 SoC_up，直接打开工程文件就能够一路生成比特流，不会报 bug，非常顺利。我一直在自己笔记本上的虚拟机上运行 Vivado，而 Vivado 又是十分耗费内存的软件，一度导致系统运行到一半就会崩溃的情况，至少每次综合实现过后尝试 open implementation design 都会显示 run out of memory。

当龙芯的开源 CPU 代码运行成功后，2017 年也随之告了一个段落。《自己动手写 CPU》书中也有一份开源代码——OpenMIPS，由于 OpenMIPS 项目的开发环境是 Altera 的实验板和 quartus，不能直接拿到 Vivado 下运行，于是我便尝试着把源代码移植到 Vivado 的项目之中跑仿真，但是一开始便被 ROM 的使用情况给难住了。Vivado 中完全无法加入.data 格式的文件，所以我便不知道怎么用自己编写文件的 Verilog 文件代替 ROM 仿真。直到很久之后我才弄明白其实可以不把 data 文件加入到项目之中，直接在文件中使用 data 文件的绝对路径就能够解决。为了解决 Vivado 中使用 ROM 的问题，我转而找到了 Xilinx 的 IP 核 block ram generator 来生成 ROM，但是这个 IP 核需要 coe 这种专门格式的文件才能够初始化，当时我却找不到生成这种文件的方法，只能暂时搁置下来。

第八节 消散的参赛队员

CPU 小组的工作任务是将 SoC 综合下板。操作系统小组的任务是在实验板上运行操作系统，开始时需要做 ucore 的十个实验。编译器小组则是在操作系统上运行编译系统。可能是其他小组的难度比较大，需要学习的东西多，几次会议下来，似乎每次只有 CPU 小组有些些进展。每次开会，向老师都要照例一一询问每个同学的进展情况，每个人都要自行汇报个人进展，每两周一次的汇报着实吓退了不少人。向老师说话从来不拐弯抹角，做了什么工作就是什么，没做就是没做，不需要找太多的说辞和借口。只是若每次过来开会都说自己什么也没有做，怕是没有脸面继续参加下去，所以也是一种督促我们继续工作的极佳方式。但是人的惰性是很强大的，大部分人课下都很难坚持着花时间持续地投入到这场长期持久的比赛准备过程之中，即使一开始把每个人的工作任务都定了下来，也难以激励大家按时完成任务，日志更无从写起，所以最后坚持到第一个学期期末还能来开会的就剩下寥寥几人。

第九节 北理合伙人

第一次注意到江学谦是在第一学期的最后一次会议，来参加会议的人很少，气氛略显尴尬，却又在意料之内。江学谦是操作系统组的人员，但是似乎当时他并没有明白自己需要干什么？向老师听完我的进展汇报后，转头问江学谦。

向老师：你最近做了什么？

江：我最近在读 Linux 的代码。

向老师：那你知道操作系统小组的工作任务是什么吗？

江：不太清楚。

向老师：你先打开 GitHub 仓库里的链接页面，我来告诉你。

江：是那个链接页面？

向老师：（打开网页）你打算继续参加这个比赛吗？

江：本来我打算自己写一个文件管理系统或者搭建一个小型计算机，
刚好有这个比赛，我就参加了。

向老师：写文件系统是个大工程。但是和我们目前的任务不太吻合，
如果还想继续和我们一起做事参加比赛，就先按着我们的计划来进行。

江：好的。

刚听完江学谦的回答，我心想这哥们，都快一个学期了，怎么连我们在干嘛都还不知道？这样真的能行吗？不过听着他说又是看 Linux 源码，又是写文件系统的，估计是个深藏不露的大神。后来我和同学聊起现在比赛的现状，提起他时，同学说江学谦缺少是个很厉害的人。经过这次会议后，比赛的重担就交到了我们两个人手里，老师让我们多交流，多合作，但是大家都不熟，各自做着各自的事，谁知道今后我们却展开了如此长久的一场合作。

第十节 发现新大陆

大三上的期末考试过后，趁着离回家还有一段时间，我便窝在学校继续写大作业和做比赛。有一天我在信教上自习时无意看到了龙芯第一届比赛的培训视频，大概就是介绍比赛形式，讲解了一些基础知识，介绍实验板的使用方法。从培训视频中我发现比赛时有发布大赛资源包，里面有很多的指导资料，比如实验箱的文档介绍，FPGA 的引脚，烧录 flash 的工具，通过龙芯开源 CPU 代码 soc_up 运行 linux 的教程。后来费尽周折终于从清华大学参加第一届比赛的学长那里找到了这个资源包。之后便像打通了任督二脉，对比赛形式有了更充分的了解，终于在回家过年之前和江学谦一起用比赛发布的资源包运行起了 Linux 操作系统。

20180209 第一届比赛培训视屏

- 17.6.2 http://list.youku.com/albumlist/show/id_50027463
- 17.6.3 http://list.youku.com/albumlist/show/id_50036144
- 17.7.31 http://v.youku.com/v_show/id_XMjkzNTcyMzQ1Mg==.html?spm=a2h3j.8428770.3416059.1
- 如需要高清版本请到百度网盘下载： https://pan.baidu.com/s/1mikMykK 密码：whzh

18.1.22 ~ 18.2.2

- 进展
  1. 烧写测试程序 cpu_test 完成对实验箱的测试，包括LED灯、显示屏幕、矩阵按键
  2. 完成记忆小游戏的测试
  3. 将测试程序编译、链接成coe文件作为ram的初始化数据
  4. 通过烧录官方给定的bit文件实现对flash的控制，并能够通过串口工具将U-boot烧录至flash中
  5. 配置tftp服务器与实验箱通过网口将操作系统内核传输至ram中，并启动操作系统
- 计划
  1. 编写实验箱测试流程文档
  2. 编写程序编译链接文档，提供所需工具
  3. 编写操作系统移植过程文档
  4. 熟悉NaiveMIPS中cpu部分内容

20180117-第一届大赛资源包

主要包含：

1. 大赛实验箱校验工具与介绍、参赛可能用到的工具与介绍、SoC_demo与简单介绍
2. 功能测试源码和使用示例、soc_up的源码和介绍文档、func_test_v1.01（去除非延迟槽测试时延迟槽非nop情况）
3. 目录perf_test/（大赛提供的性能测试源码和使用说明）、目录ucore_thumips/（清华提供的小型操作系统ucore）
4. doc_v0.03/里，新增一个文档：预赛提交要求.pdf

大赛资源包

第二章 漫长的赛前准备（下）

第一节 重新召集队员

新学期开始了，大三下又是事情堆积的一个学期，有好几门挺难的专业课，于是我量力而行，把其他的比赛都推掉了，只留下这一个比赛还坚持准备。陆老师趁着学期开始，又找了一些同学来参加比赛，加上原来的同学还能凑两个参赛队伍。第一次开会时我给所有同学和老师们讲解了一下上学期我们的进展状况，然后让新加入的同学按照文档来复现我所走过的实验过程。

第二节 再次复现实验

按照向老师的指导思想，寒假时通过资源包进行的实验复现，并非是他所期望的。因为虽然 CPU 源码是自己编译的，但是操作系统和引导程序都是直接用编译好的映像文件，没有移植的可能，其次基本不可能照着 SoC_up 写一个 CPU，所以我们的任务便转向通过 NaiveMIPS 来复现实验，我负责 CPU 部分的综合，江学谦负责引导程序和 ucore 的编译。

开始时，我直接从 GitHub 上面将 NaiveMIPS 的源码下载下来，但是代码太多实在没有办法看懂，加上是直接从网页上下载的 zip 包后再解压，所以第一次打开项目直接就报错，提示我少文件，折腾了很久也没弄明白。然后向老师给我推荐了源码的作者，联系后才知道是代码下少了。于是我又重新下了一遍，事情也没有那么简单，即使少文件的错误解决了，但是项目同样综合不过去。

第二次开会时，向老师按照惯例问我们有什么问题，但是我真的不知道怎么描述遇到的问题，只能说做不出来，项目动不了。因为向老师的研究方向和 CPU 设计并不契合，当时我直接打开了 Vivado 中的项目，我们一干人等盯着屏幕看了很久，也无从下手。向老师还是让我转变研究思路——尝试替换项目中的 IP 核。从局部做起，弄清楚每个 IP 核的用法。后来从我自己实际搭建 SoC 的经验来看，发现这个方法也不太行的通，即使我能够把每个 IP 核的用法都弄懂，也还是很难了解整体的运行过程。但是转变方向的好处就是让我有了一些缓冲的时间去了解 SoC 的搭建过程，连接外围设备的连接和进一步加深对 Vivado 的使用。与此同时，南开大学的同学也一起加入了比赛的准备过程。

折腾到了 5 月份，向老师告诉我 NaiveMIPS 的作者已经修复好项目的问题，现在可以重新运行，于是我终于赶在 5 月底完成了对于 NaiveMIPS 的复现工作，成功运行起 U-boot 和 ucore。

第三节 报名参赛

在我尝试用 IP 核搭建 SoC 的同时，其他同学也各自分配到了自己的工作任务。让我觉得奇怪的是，明明我写了很多的实验指导，如果单纯复现实验的话，应该没有什么难度。可能大家都太忙了，对于硬件没有什么知识储备，而我们学习的课程对比赛的知识支撑帮助不大，所以久而久之便对比赛丧失了信心，在对分配的工作无从下手时，就坚持不下去了。记得原本开学的时候还能组两个队，可是到了五月份报名参赛时却只剩下 4 个人，将将能组成一个队伍，我给参赛队伍起了一个名字——BIT_MIPS。原本在 5 月 13 号在湖南大学有一个

赛前的培训工作，但是那段时间我们忙于准备 8 周结课的考试，就放弃了这次路途遥远的培训，后来又在王娟老师的帮助下把几十个 G 的培训视频下载了下来，但是一直到了比赛结束都没有看完。。。

第四节 一起移植软件

清华大学的 NaiveMIPS 虽然运行起来了，本以为能开始着手设计自己的 CPU，但准备工作似乎并没有完全完成，向老师要求我们达到的目标是用 NaiveMIPS 运行清华的 Supervisor 监控程序，并结合 Supervisor 来实现对于测试程序的自动化对比，以方便后期的调试工作。于是我便转向和江学谦一起去研究软件方向了。

Supervisor，测试程序，ucore 这些运行在系统上面的程序都是 MIPS 汇编编写的软件。刚开始时我看起 MIPS 汇编都有些吃力，更不用说整个大的项目。这里面还涉及到编译链接等各种内容，第一次看链接脚本，Makefile 文件时完全摸不着头脑，此时全靠江学谦一人 Carry 全场。我能做的就是把老师的要求和他讨论具体的实现方案，然后他来实现。这让我想起有一次我的任务是用 Verilog 来实现一个简单的 SPI 到 UART 的协议转换，可是我对协议，时序等知识的认识仅停留在课本上的概念，并不知道怎么实现。江学谦给我讲了很多遍我都没有弄明白，后来就差他没有手把手带着我写一遍，我才渐渐对协议实现有了一些浅显认识，算是为 CPU 设计打下一点基础。

回到软件的复现部分，CPU 虽然下载下来就能够运行，但是这些软件却不行，问题集中在对本地交叉编译环境的配置和对硬件的适配上。Supervisor 本身并没有适配 NaiveMIPS 的版本，所以还得先看懂然后去适配硬件。江学谦很快就完成了与硬件的适配工作，但是用 Supervisor 来进行自动化对比却并不是 Supervisor 本身就带有的功能，所以得自行实现。我和江学谦花了几个晚上的时间，讨论了很久，最后想出了实现的方式。虽然江学谦表示他能够做出来，但是其工程量和所耗费的时间在当时却是不可接受的，拖到了 6 月份最终只能放弃 Supervisor，转而想另外的自动化对比实现机制了。

在询问了 NaivMIPS 的作者之后，我才知道原来项目本身就自带自动化测试的对比程序，一个直接用 SystemVerilog 实现的仿真 top 文件，可以作为仿真环境下测试的参考。除了仿真环境下的测试，还要考虑到下板后的测试，在一次开会时我们找到了下板后的分析工具——ILA(逻辑分析仪)。其实就是 Xilinx 的一个 IP 核，但是在今后的调试工作中却帮了我们好大的忙。就这样，我们运行了 Supervisor，学习完 ILA 的使用后，准备工作也算是做的差不多了。6 月份将近过半，此时我们便放下手中的活计，专心准备期末考试了。

第三章 艰难的初赛设计

第一节 蹭一蹭清华的小学期

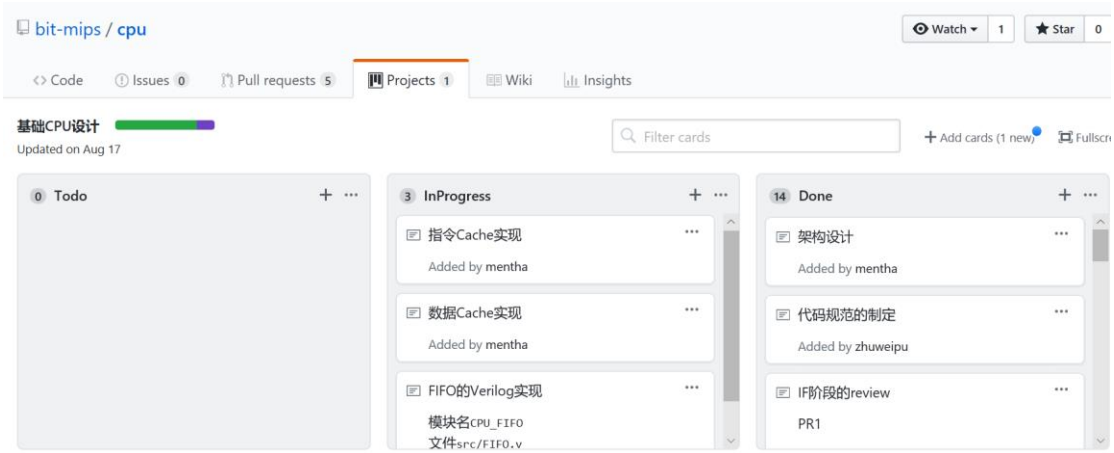
大三下学期结束了，虽然比赛的准备工作历时两个学期，但其实正事一点都还开始。因为 8 月中旬就要提交初赛作品，所以其实只有 1 个半月的时间让我们设计并实现参赛作品。向老师索性邀请我们参加清华大学的小学期课程，和清华的同学做完实验后将结课作业作为参赛的作品。这听起来是一个很好的机会，我们便欣然答应，而此时小组成员也只剩下三个人。

7 月 4 号，暑期小学期课程正式开始，每天早上 9 点钟在清华大学三教开课，和我们一

起参加的还有南开大学的王同学。第一天，我们早早搭地铁又换乘小黄车到了清华。暑期的清华门口人山人海，各地过来旅游的小学生团纷纷在校门口换上学位服，等着排队拍照。因为游人过多，清华校门口的门卫会拦人，有证件才能出入，但是大多数时候只要背着书包长得像个学生就基本能够蒙混过关。过了校门口的一关，没想到教学楼还有门禁，我们只能给向老师打电话，然后让楼管给我们开门，费尽周折终于进到教室。第一天的课程是讲解课程目标，但是课程自始至终没有具体教流水线的设计知识。这门课程作为清华的一个挑战性选修课程，敢选课的人要么是真大佬，要么就是被迫选的，不过 20 来人。然而不管怎样，清华的同学在计算机组成原理实验已经做过一个 16 位的 MIPS CPU，操作系统的实验课也都做过 ucore 的 10 个实验，所以如果要上这门课简单的无非是把之前做过的大作业改一改就行，但是老师们深知每个同学相应的水平，针对每个小组提出不一样的目标要求。

第二节 开始我们的设计

幸好我和江学谦两个人都自学了流水线设计的知识。去清华上课的前一天晚上，我心里其实还没有一点底子，因为之前一直是纸上谈兵，没有设计 CPU 的实战经验。我们的作品到底要用什么样的架构，什么样的设计？是照着 OpenMIPS 直接写一个，还是参照 NaiveMIPS 写一个？还是自己从头到尾设计一个？于是我找到江学谦和他商量应该按照什么样的设计实现 CPU，当时的他和我完全不同，表现出一副胸有成竹，跃跃欲试的样子。显然我和他学习的路子不相同，讨论时听着他的设计方案总觉得比我的想法高出一个层次。原本我只是想简单的参考 OpenMIPS 的架构写一个 CPU，江学谦告诉我那叫做 Classical RISC Pipeline。但是江学谦却提出了更多很好的想法，听完他的介绍，让我信心一下子提高了不少，于是决定采用他的设计思路——FIFO 模式的流水线。我们确定了每个阶段的功能，接口和时序，想要完全按照自己的设计打造一款作品。为了做好代码的管理，当天晚上我们就在 GitHub 上建了一个仓库管理项目代码。



小学期第一天早上的课程结束以后，向老师请我们在清芬园三层吃了一顿午饭，类似于现在的北理工二食堂，连上陆老师和南开的同学，一共 6 个人。大家心疼向老师的饭卡，不好意思点太多，然后向老师开玩笑说我们吃那么少，像个小姑娘似的。吃饭时，向老师问我觉得做实验最大的障碍是什么？我回答说是缺乏一个全局的观念，每次老师问我遇到什么问题，我就只知道自己没做出来，但是不知道问题在哪里，又要去哪里寻求答案，连搜索引擎都不知道要输入什么关键字。向老师听后十分感慨，其他同学也发表了一些个人看法。

下午的实验安排在东主楼四楼的组原机房，其实并没有什么人指导，无非是提供了场地让我们做实验，大家各自为政。这时候因为时光昱还要补 Verilog，流水线设计的基础知识，并不能够投入到 CPU 的设计工作中。当时我也期望着他能够尽快跟上进度，实在不行还能

帮我们做做测试工作。我和江学谦则继续我们昨天的设计，进行各阶段接口的设计。一下午的时间很快就过去了，加上早上的奔波，中午也没有能够休息，所以很快就觉得疲倦了，趁着天色还早，我们便结束一天的工作，赶回北理工吃晚饭休息了。

第三节 转战 214

第二天早上是向老师的课，介绍操作系统，然而这个内容对于比赛来说没有什么帮助。可能是我习惯了自学，遇到不懂的知识点再问的学习方式，要我上课听讲反而记不住什么东西。下午我们又在组原机房继续设计 CPU，但还是没能把接口设计完。我觉得如果接下来还按照这样的进度，一共四周的实验课，第一周早上纯理论课，剩下的时间自由做实验，估计第一周过去了还没能完成设计。

七月份的天气异常炎热，每天奔波来去，清华的基础设施对于外来人口又特别不友好，各种限制，整个人都感觉不好了，于是我回去后就和向老师商量，希望不再过来清华做实验，改成在北理工，然后还按时参加每周一早上的工作汇报。向老师尊重我们的意见，只是劝说我们上一上理论课，我说想学的话可以看 Wiki 上的 ppt 和资料，向老师说现场听课的收益会大一些，但是因为彼此的学习方式不同，他便不再说什么了。于是我们把阵地转移到了机房 214，空旷整洁的大实验室，还有可控温度的空调，重点是不用每天跑来跑去，可以把精力更多的投入到比赛中。

第四节 失败的尝试

我们赶在小学期第三天将第一次 CPU 设计的接口、时序文档编写完毕，并且备份到 GitHub 中去。剩下的工作便是将开始实现，事实证明我们还是太年轻，因为我们之前并未实际写过一个 CPU，没有经验可言，所以无法预见实际做的过程中会遇到什么样的障碍。当我们两个人分着工，花了两天的时间，各自把每个阶段按照其功能和接口实现起来后，却发现代码已经不在我们的掌控之内。首先我们在单独写完每个阶段以后，并没有办法对每个阶段做相应的测试，其次如果直接将写好的各阶段简单粗暴的连接起来，完全不知道能否正常工作，状态不可预知，即使出错了也很难查找。

商量过后，我决定将已有代码交给江学谦继续维护，而我则转变方向参考 OpenMIPS 的经典五级流水结构来实现流水线。在第一版 CPU 接口和时序设计的基础上，我花了一天时间就大概确定了基础版本的流水线各阶段接口的设计，然后又花了一天时间就把一个流水线建立了起来。这次实现设计的方法是先通过实现一条指令将各个阶段串联起来，建立起一个初步的流水线，之后再在此基础上逐渐添加新的指令，添加完成后立马做测试，保证代码的可控性。这时候时光昱便参考 OpenMIPS 书上的测试用例来对流水线进行测试。

第一周的工作结束了，按照先前的约定，周一早上需要到清华进行实验进度的汇报。鉴于第一周我们的失败，只能硬着头皮过去做展示，陆老师和王老师也陪着我们一同到清华听报告。作报告时我事先并没有准备，所以自己都不知道自己在说什么，老师更是听懵了。总结起来就是上周做了一个版本的 CPU，发现做不出来，现在又推翻从头写起。这门课的任课老师刘卫东老师建议我们不要操之过急，先把接口时序，冲突冒险关系都想好，文档的设计必不可少，这也是对清华同学的基本要求。向老师说我们急于求成，觉得上个星期的理论课对做实验没有什么用途就干脆不过来上课了。我们站在上面不知道怎么回答。会后我们又在向老师的实验室里仔细地将第一次失败的作品设计说明了一次，王老师作为我们学校教这门课的老师也听懂了我们的问题，然后提出一些建议，让我们干脆写一个多周期的 CPU。当时

我却不明白王老师所说的建议要怎么去实现，直到大四上小学期的时候，看到小学期的实验资料时才明白原来我们对于 CPU 实现的理解并不是在同一个角度上，或者说是我入门的方式比较奇特，才导致我和王老师的之间的沟通屡屡不畅，往后多次讨论意见都不统一，最后还是江学谦听出来我们两个的矛盾点才解决了这一问题。

第五节 基础版本流水线新鲜出炉

第二周工作开始前，我们就按照自己的想法重新安排了工作内容和前中后期的目标。花了五天的时间，我就把基础版本的 CPU 整个写的差不多了，不过此时实现的仅有大赛要求的指令，基本上就是 MIPS1 指令集，去除中断异常后就很少了。时光昱照着《自己动手写 CPU》书中提供的用例做测试，检查出了不少的 bug，都是大意导致的问题，比如变量名没写对，数据宽度不对齐等。江学谦则继续在实现之前设计的 FIFO 结构流水线，不过后期还是难以测试，加之我这边把流水线搭建得差不多了，所以我们就彻底放弃了这个版本。最后两天江学谦则负责搭建 SoC 以运行 Supervisor，实现串口功能和软件。最后一天我们便搭建起一个能够正常综合，并在仿真环境下工作的 SoC。但是此时出现了一个很大的问题，项目综合完成后江学谦对我说发现综合出了很多的 Latches，我一脸懵。

我：什么是 Latches？

江：数电学过吗？

我：学过，但是忘得差不多了，项目有错吗？

江：没错。

我：没错就行，那就下板试试。

江：现在下板状态不可控，出现 Latches 以后 Vivado 就不能自动分析路径。

我：那这个东西是哪来的？

江：是你写的。

我：我没写啊。

江：你写的代码有问题，但是我现在也不确定哪里有问题，因为出现了几千个锁存器。现在只有一个个改，或者把代码重构一遍。

我：晕。。。。。。。。。。。。。。

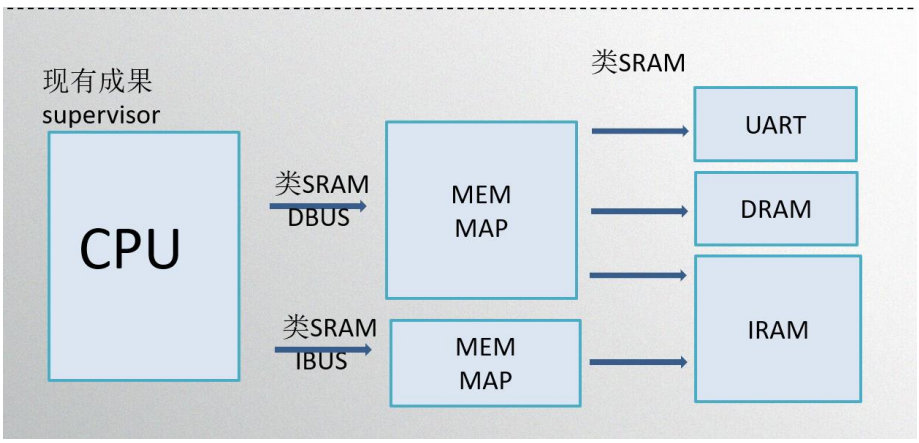
我发现要想做好这个比赛，得懂的东西还真是不少。`Inferred Latches'，即「推断出的锁存器」。Verilog 中的「寄存器」模型和计算模型与实际硬件中的锁存器(触发器，Flip-Flop)、查找表(LUT)、存储器(RAM)等行为不完全一致，因此在综合设计时，综合程序会根据推断规则以及 Verilog 所描述的行为尝试使用硬件模仿出相同的行为，如在时序逻辑中使用的「寄存器」会映射到 FF，逻辑函数映射至 LUT，「寄存器」数组往往映射至 RAM。在 Verilog 中，采用`assign'语句或`always @*'实现的组合逻辑一般为单稳态电路，用来实现逻辑函数。逻辑函数是数学函数的一种，其输出仅仅取决于输入，内部不能记忆状态。需要记忆状态时，必须采用多稳态电路，一般采用时序逻辑，综合时会用 FF 实现。如果在设计组合逻辑时不慎将某一个逻辑函数的输出通过其他逻辑函数接回至其输入，该组合电路就可能成为多稳态电路（如直接按照触发器的接法连接多个逻辑门），内部可以记忆状态。实际设计中，时序逻辑基本上可以实现所有需要记忆状态的功能，并且进行分析比较简单，因此，对于此类电路综合工具会给出警告。除了多稳态电路，用组合逻辑还可以实现无稳态电路。这种电路一般会构成不稳定的振荡电路，其输出频率受硬件和环境影响。若设计中的某一个部分不幸错误形成这种电路，其行为难以预测，可能给调试带来较大的困难。于是第二周的工作也只能就此收尾。

第三周周一的早上，我们照旧去参加报告。作报告时我们都是排在第二个，而第一个组

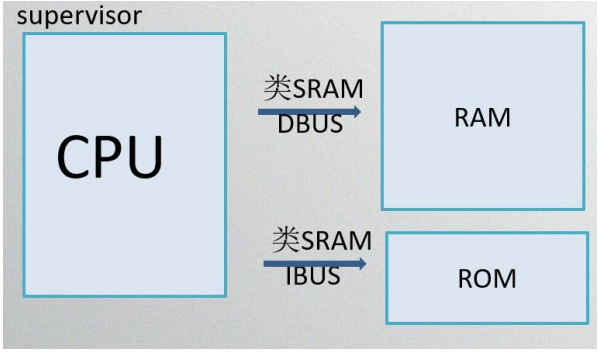
是清华做的最好的小组，每次的工作内容和进展都极为惊人，报告的内容得三四十分钟才能够讲完，和之后我们三分钟的即兴发挥形成强烈对比反差，一度让我感到压力山大，怀疑人生。参加完清华的报告，我又急匆匆赶去了夏令营报道，比赛中断了三天的时间，期间江学谦自己一个人完成了对 CP0 的设计实现，而时光昱最终还是退出了比赛，等我回来时小组就彻底只剩下两个人了。回来后，我们找到了之前犯错误的原因所在——由于我对于 Verilog 的不熟悉，在使用 if else，case 语句时会忘记写默认情况下的赋值，即赋值状态不完整，导致综合后的电路在默认情况下就会出现锁存器锁存先前值的情况。按照江学谦的建议，我花了两天的时间对代码进行了整体重构，通过 function 的形式写出每个功能部分，然后再用 assign 语句进行赋值，当整个比赛结束我回顾这一年来的工作时，我才发现这是我们做过的最正确的选择，没有之一。

第六节 移植 Supervisor

要运行 Supervisor，SoC 的一个很重要功能就是串口。由于一开始我们设计的流水线是类 SRAM 接口，所以并没有现成的 UART IP 核可以使用，需要自行编写。在我看来这还是挺难的一件事，然而江学谦三下五除二就做好了。运行 Supervisor 还需要用到片上的存储资源，所以还需要选择 RAM IP 的类型，一开始我还认为 block ram 的 IP 核时序都是当前周期给出地址，后一周期才会返回数据，（distributed ram 是用 LUT 实现的，当前周期即返回数据）所以访存接口与 OpenMIPS 并不相同，指令也直接返回至译码阶段，而不是取指和译码间的流水锁存器，这也是我为自己埋下的一个大坑。



这是我们搭建的运行 Supervisor 的 SoC。记得我开始时说通过看《自己动手写 CPU》来学习怎么写流水线具有一定的局限性，比如对 CPU 设计方式、MIPS 整体架构的认识狭隘。到了这个阶段问题就暴露得一览无余了。对比我和江学谦两个人的学习方式，最大的不同是他是通过看 MIPS 的手册来了解 MIPS 的设计原理，通过不同的途径也积累计算机系统的知识。我在学习写 CPU 时，就只学习了每条指令的具体实现方式，所以认为 CPU 都是像 OpenMIPS 一样从指令 ROM 里面取指令，从数据 RAM 里面存取数据，即标准哈佛结构。



当时江学谦反问了我一个问题，如果你要实现 Supervisor 里面的写一段程序到指定地址然后执行的功能怎么办？如果 CPU 只能从 ROM 中取指令就意味着不能够改写程序，就实现不了这个功能。我设计的 SoC 思想完全只适用于嵌入式操作系统，所以指令 ROM 必须改用为双口的 RAM，让 CPU 能够从一个接口读取指令，另一个接口作为访存阶段写入新指令的接口。此外各种的外部设备和存储器要统一的按照 MIPS 规范进行地址映射。我在专业书中不止一次的看到过一个概念——Address Space，但是我对于它的理解一直是 DDR 一类的内存，联系不到外设统一编址的概念，所以做地址映射 MEMMAP 模块就更不会写了。因为是自己写的串口，搭建完成 SoC 后就一直在调 UART 模块的 bug，好不容易终于下板让串口输出了正确信息，但是 Supervisor 的还是没能正常工作。

第二天去清华做完第三周的报告后，我们又接着昨天的工作继续调试 Supervisor，最后只有部分功能可以正常使用。江学谦没有心情继续调试 Supervisor，因为他认为这个软件本身还有许多问题，便转而写 AXI Cache 接口的模块，而我为了在清华的结课报告上有些展示的内容，便继续调试这个软件。

后来我花了一个早上的时间按照 Supervisor 的出错功能编写了相应的测试文件，之后在仿真环境下运行程序，成功定位到了软件的问题。后来发现其实是 Supervisor 的源码出现问题，而不是硬件的问题，为了赶比赛进度便放弃了本次调试，转而实现比赛要求的所有剩余指令。

第七节 适配初赛框架

在我实现剩余指令的期间，江学谦抽空把 TLB 模块写完了，所以整个 CPU 最难的部分，CP0 和 TLB 都是他写的，我基本上把这两个模块当成黑盒子直接用。实现完比赛要求的指令后，我本来打算直接将 TLB 模块加入到流水线中，但是通过一早上的改动，我的心态出现了一些问题。算起来，我们来来回回对流水线改动很多次，但是如今我们只有两个劳动力，测试一直跟不上，我怕最后如果出现问题，都不知道该回退到哪个版本，有种差点就要控制不住局面的感觉，所以我干脆停下继续做新功能的脚步，转而去适配龙芯发布的比赛框架，并以此作为 CPU 的测试用例。

比赛的框架很早就发布出来了，分为三个部分 SRAM 接口的功能测试，AXI 接口的功能测试，AXI 接口的性能测试。我们最早适配的是 SRAM 接口的功能测试框架，所谓适配，就是将 SoC 的 CPU 部分换成我们自己写的 CPU，外设和存储不需要自己动手搭建。虽然我们的 CPU 设计是类 SRAM 接口，但是适配起来并不麻烦，一上午便能够开始综合下板。

功能测试框架一共 89 个功能测试点，第一次运行时无论是仿真还是下板，都才运行了几条指令就卡住不动，显然系统已经跑飞了。等着我们的就是艰难的调试环节，在这里真的不得不夸奖江学谦调 bug 的能力。记得在准备比赛的过程中，向老师早早就和我们提起过自动化测试的问题，这个工作任务交到过至少三个人的手里面，但是他们都一一退出了，所以

我把这项工作称为——测试劝退。其实龙芯所给的框架里面就自带这一项功能，叫做 trace 机制，能够把错误的指令和地址信息打印出来，参赛的同学大部分都靠着这个功能完成了调错，但是我们别树一帜，从头到尾愣是没用，完全靠着观察波形把 bug 都找出来了。

江学谦找 bug 的方式是这样的，首先将程序的运行方式看懂。程序每个部分的动作都在脑海里面有个清楚的认识。测试框架程序会在数码管上显示当前已经测试的指令数和通过的指令数，在仿真环境便以变量此作为观察程序是否仍正常运行的依据。当出现未通过的指令时，则添加信号追踪出错的指令。通常错误不止一个，但是一个小问题往往在运行过程中就会导致很大的错误，比如系统进入死循环，跳转至错误处理部分等，所以追踪时应该先确定系统当前状态与预期效果最明显不一样的地方，然后一一回溯，这样便能找到最初导致错误的地方，也就能够找到执行出错的测试用例，并以此为依据修正代码。

当时出现的错误真是花样百出，包括代码变量写错，在 function 中使用全局变量，有符号数和无符号数的判断等问题，多属于粗心写错，而不是设计上的问题。

记得有一次仿真和下板环境下运行出错的指令不在同一个地方，这就让我感到很困惑了。只能分别在每种环境下单独调试，仿真环境下则观察波形，下板时则需要借助 ILA 来查错。因为当流水线接口采用类 SRAM 接口时，指令会在下一个时钟周期返回至译码阶段，但是此时如果流水线出现暂停，block ram 并不会接收到暂停信号并保存此次数据，从而导致指令和地址无法对齐。每次江学谦定位到了错误，我们便分析这个问题是谁导致的，谁写的谁就负责修复。每次调试完一个 bug，又重新开始运行，渐渐地数码管上通过的指令从刚开始的个位数变成十几，二十几，三十几，直到最后全部通过。整个过程就像是一步步通关打怪，不断的升级进阶，满满的成就感。到了第四周结束的时候，我们便能够通过全部的功能测试。龙芯的测试用例真的很完善，从这以后一直到比赛结束，流水线已实现的指令代码便再也没有出现过任何问题。

第八节 再见小学期

适配完 SRAM 接口的功能测试后，离清华小学期结课还剩下三天的时间，为了有个好的成果展示，我建议江学谦和我再调试一次 Supervisor。于是我们又回到上次错误的地方，花了一些时间找到Supervisor源码中的问题，终于赶在报告前一天运行起带中断的Supervisor。虽然之后便再也没有用过这个程序，为此搭建的 SoC 如今也不见了踪迹，但是也算让我们的报告展示显得没那么窘迫和寒酸。当时我打电话给向老师汇报进展，他说想给江学谦一点锻炼表达能力的机会，让江学谦做结课的报告。第二天作报告的时候我按照规定介绍了流水线部分的内容，因为流水线架构是按照我的设计进行，所以交由我讲，虽然只有一页 ppt，但是我讲的比江学谦所有 ppt 加起来还要长，这就让人产生了一种江学谦讲不下去了所以然后我才接着讲的错觉。

总结在清华上小学期的这段经历，虽然表面上对于我们做比赛并没有知识层面上的促进作用，也没有达到和清华同学相互交流学习的目的，但是却以另一种形式促进了我们的比赛，给出了一个时间上面的规划。每周的定时汇报也让我们能够及时审视自己的进展，比如通过第一周的失败快速迭代作品，更关键的是受到一群优秀同龄人的撼动，知道别人可以优秀到什么地步，才不会轻易的自我满足。

第九节 错综复杂的 AXI

课程结束后，我们便全身心投入到初赛的工作任务，适配 AXI 版本的功能测试和性能

测试。由于设计的作品是类 SRAM 接口，需要写一个转换桥将接口转换为 AXI。起初这项工作交由江学谦完成。此时遇到的最大障碍就是流水线的暂停问题解决不好。设计按照 OpenMIPS 使用统一的 ctrl 模块控制流水线的暂停，但是这样的解决方案被江学谦诟病了很多次，终于在取指和访存阶段间的暂停请求互锁问题出现后，让我决定重构流水线的暂停控制模块，将每个阶段所发出的暂停请求集中到一条总线上，然后将暂停信号引入转换桥。在重构的时候，两个人都进入了一种疲倦状态，在我们设计把信号分离出来后，脑子就一团混乱，既不知道如何修改流水线，也不知道该将哪些信号组合起来作为转换桥的输入。我试着拿起马克笔在白板上引导要讨论的内容，但是两个人都默不出声，像是经历了长途的跋涉后再也迈不出去的大腿。虽然剩下的工作就只剩下类 SRAM 转换为 AXI 接口的实现，但却觉得像一道巨大的鸿沟阻碍着我们继续前进，又无法跨越。能够做出来就有性能分数，做不出来就没性能分数，决赛也无从谈起。况且整个 Xilinx 体系的 IP 核基本都是 AXI 接口，要搭建 SoC 更绕不开 AXI 协议。后来王老师过来和我们讨论了许久，期间还和向老师通了一个电话会议，但是似乎他们没有弄懂当前迫在眉睫需要解决的问题是什么。按照老师的建议，我还是写下了几条准则作为实现转换桥的标准。

关于增加外部总线的设计原则

1. 延长访存周期，将单周期延时改为多周期延时
2. 确定访存优先级，以MEM阶段的访存优先
3. 确定访存时流水线中断异常的处理方式

工作安排

1. 朱威浦实现转换状态较少的AHB接口，之后用IP核将其转换为AXI
2. 江学谦实现AXI接口Cache

由于写转换桥的工作一直交予江学谦，我对 AXI 协议的理解并不深刻，只能在旁边干着急。后来迫不得已便转换思路，虽然直接复用 NaiveMIPS 的代码并不实际，但是我可以参照 NaiveMIPS 将类 SRAM 接口先转换为 AHB 接口，之后再通过 IP 核转换为 AXI 接口。AHB 协议比 AXI 协议涉及到的信号量少，花了一下午时间我便把状态机和时序都设计完成了。虽然绕了很大弯子，还是把类 SRAM 转换成了 AXI 接口。在我刚刚实现完成转换模块后，便看到龙芯在比赛官网里发布了直接将类 SRAM 转换为 AXI 的转换桥，敢情费了那么大劲儿，全给白费了。我还是珍惜自己的劳动成果，硬要拿自己写的模块先把类 SRAM 转换成 AHB 接口，然后再通过 IP 核转换成 AXI。适配完 AXI 功能测试 SoC 框架后，又出现了一个问题，因为框架的 AXI 协议版本和 IP 核不一致，通过 AXI 版本转换 IP 核后，我的转换桥模块便不能正常工作。调试了半天，发现问题出现在 IP 核上，没有办法做改动。此时，我又不得不转换方向，直接用比赛提供的类 SRAM 转 AXI 模块实现转换，放弃原先写过的转 AHB 模块代码。

当我真正开始看龙芯的开源转换模块代码时又愣住了，因为我原先理解的类 SRAM 接口与龙芯的模块中提到的实际并不是一回事。龙芯的转接桥里提到的类 SRAM 接口是指在 SRAM 接口的基础上添加握手信号实现同步，而我在流水线里面写的类 SRAM 接口只是与 SRAM 接口的时序不相同。因此，为了减少对于流水线的改动，我就不得不先把流水线的接口改回 SRAM 接口。

经过如此多的周折，我才摸清了设计的门道，才知道在选择时什么时候要选 SRAM 接口，什么时候要做类 SRAM 接口，怎么走会带来什么样的后果，产生什么样的影响，这些都是自己去尝试过一遍以后才能够积累下来的经验。因此，我对取指访存的时序做了更改，

指令将直接返回取值和译码阶段间的流水锁存器。当初辛苦想出来的解决暂停信号导致指令地址不对齐问题的机制都可以去掉了，而原本设计在执行阶段发出访存请求，也调整到了访存阶段。幸好 AHB 和类 SRAM 也有些相似，于是我在原本 AHB 模块的代码基础上做出一点改动，就实现了为 SRAM 添加握手信号，转换为类 SRAM 接口的功能。这一次运气很好，AXI 接口功能实现后，再也没有出设计上的问题，顺利的适配了 AXI 功能测试与性能测试框架。

第十节 难以提高的性能

初赛提交截至前的一周左右，我们便把性能测试运行了一遍。第一次跑完性能测试时看到仅有可怜的 0.9 几分（与龙芯 ls232 比较的性能倍数），那时我们俩眼角一斜，觉得这性能差的出奇。然后趁着还有时间，我们便想要着手提高性能，主要从提高主频和加入 Cache 两个方面入手。一开始我们的主频只有 50Mhz，于是我们干脆把 pll 输出时钟调整至 100Mhz，Vivado 综合分析后出现了负数的 wns，还记得我在写第一版 CPU 的时候综合出现的锁存器吗?如果有这些锁存器存在，Vivado 便无法自行分析出哪些路径是过长超时而无法正常运行的。通过江学谦的分析，我们看到了 Vivado 综合过后的电路图，找出了限制频率的最长组合路径，发现这是一条从译码阶段延申到访存阶段又返回至取指阶段的路径，我们惊讶于为什么 Vivado 会综合处那么长的一条路径，几乎贯穿了五级的流水，最后发现居然是对于转移指令的实现与数据前推导致的问题。首先我们会在译码阶段取操作数，涉及到数据相关，采用数据前推的方式解决，然后执行和访存阶段的数据便会前推至译码阶段，为了避免转移指令跳转时对指令的浪费，把转移指令的实现放至译码阶段处理，这样有好处，就是方便，无需做转预测。但是到了后期，便会出现转移指令在执行时，首先要等待执行和访存阶段的前推的数据作为是否转移的判断条件，最后再把跳转的地址送至取指阶段的超长路径。由于各阶段联系得十分紧密，无法做出大的改动，只能做些小改动把主频提升到 80Mhz 就已经到达极限了，和其他学校的 100Mhz 以上的主频完全无法媲美。其次是增加 Cache，江学谦尝试直接写 AXI 接口的指令和数据 Cache，当时我问他能不能看懂 NaveMIPS 然后照搬 Cache 过来用呢？江学谦表示不行，因为 Cache 需要保证数据完整性。两个 Cache 如何保证同步，如何安排暂停信号避免死锁，如何在另一个 Cache 从内存读取旧数据前更新内存。如果通过软件解决这些问题，那么就要考虑 Cache 要以怎样的接口接入流水线，访存指令是否需要增加等待。我虽然不理解，但是后来我尝试着按我的想法直接移植过来使用时，还是发现我想的太简单，因为接口时序并不一致，而且涉及到指令和数据 Cache 的同步问题，还要看懂他的实现，然后在我的流水线上增加相应功能才有可能复用。尝试了很久，Cache 想做很简单，但是很难做好。如果做不好不如直接用 IP 核，至少不会出现 Bug。Cache 中包含大量的存储单元，如果用 FF 实现，仅 16KB 就需要 128K 个 FF；若采用 Distributed RAM，一个 LUT 能存储 64b，16KB 需要 2K 个，所有的端口都必须连上，能不能成功布线？怎么布线不影响性能？NaiveMIPS 中就出现这个问题，容量加大后布线失败；Block RAM 差不多是实现高性能 Cache 的唯一选择。GitHub 上的 Cache 很多，但搜索结果前三页上的实现没有一个综合出 Block RAM，还有不少连 Distributed RAM 都用不到。我们几次都在 Cache 这个领域败下阵来，这也意味着直接做 AXI 接口对我们来说已经不可能了。之后我们找到一个 Xilinx IP 核 System Cache 作为系统的 Cache 使用。经过这样的提升后才把性能分数提高到了 5.9 几分。当我看到第一届比赛的性能分数最高分还不到 3 分时，我觉得还有希望。

第十一节 提交初赛作品

这中间还有一个小插曲，Vivado 这个软件是相当耗内存的，起初一直是用我自己的笔记本电脑就能够综合跑完功能测试，但是当我们加入 Cache 之后，笔记本的内存就承受不了性能测试的仿真了。于是我找到了王老师，申请使用学校的服务器，谁知道学校却没有能够跑硬件的服务器，只能从机房抱了一台配置比较好的台式机到 214 使用。这台电脑只有 8G 的内存，跑起来还是崩溃了。后来王老师又给我们介绍了一位电脑配置了 30G 内存的研究生学长，于是我们把项目交给了学长，让他帮忙跑一下测试。第二天早上我本以为可以运行完，谁知学长才把功能测试运行完，正准备跑性能测试。我只能耐心等待，下午我去到 214 的时候，江学谦告诉我他已经解决 Vivado 运行崩溃的问题，我问他是怎么做到的，他说花了一早上写了一个程序，调用系统函数限制 Vivado 的内存使用，始终给系统留下一部分内存做调度。等我们到下午跑完整个性能测试的时候，研究生学长那边才刚跑完了一半性能测试。

到这里，我们便完成了初赛作品的设计，小心地按照规定打包，然后在不同的机器上测试了好几回，最后还提前了几天提交，剩下的便是等待初赛结果和决赛通知。

第四章 迷离曲折的决赛设计

第一节 ucore 还是 Linux

老师们看到我们的成果都感到很开心，于是便趁着时机召开一次会议总结成果和做后期的工作安排，我们也表示有信心能够进入决赛。按照比赛的组织形式，决赛需要拿出一定的展示，可以从提高性能和搭建系统两个方面入手。按照向老师当初的目标，希望我们能够继续搭建系统以运行操作系统。江学谦终于可以回归自己的老本行了。

我一直对操作系统的内容不是很了解，所以此时江学谦说什么就是什么。当时向老师让我们移植 ucore，但是江学谦并不中意 ucore 的代码风格，认为改动起来不是一件容易事，然后便主张直接移植 Linux 内核。

向老师：如果 ucore 都移植不起来，那么 Linux 就更不用想了，应该有个循序渐进的过程。

江：其实 ucore 改起来就只需要改一改串口地址

向老师（疑惑脸）：为什么？（有那么简单吗）

江：其实移植 Linux 也是移，移植 ucore 也是移。

我：嗯，就是下板如果真的有问题，那么我觉得调试的手段也是一样的，倒不如直接上手 Linux。

但是当时我也有些担忧，以我们 CPU 的性能，移植 Linux 内核会不会太冒险。江学谦安慰我说以这样的性能运行 Linux 足够了。我问他移植 ucore 真的只需要改代码里的串口地址就行？他说是的，但是要真正下板还得和硬件做适配，这才是麻烦的事情，所以我不想弄。既然如此，老师们也就不再多说什么，任由我们继续尝试了。

第二节 赶上暑假的小尾巴

初赛完成后，离开学只剩下一周的时间，向老师怕我们时间紧张，所以一直催我们尽快开始搭建系统做移植工作。但是我刚好需要和同学去广州比赛，所以便借此休息了一个星期的时间。期间，组委会公布了初赛的成绩。我们能够如愿进入决赛是我意想之内的事，但是今年的参赛队伍如此强劲却是我意想之外的事。

比赛回来时同学们都已经开始了小学期的课程。大四上学期的小学期刚好就是写 CPU 的实验课。王老师担任这门的任课老师，便让我俩继续进行此次比赛的设计，作为自主实践课程的任务同其他同学一起参加小学期的课程，也省去了一些不必要的重复工作。

第三节 改动流水线

江学谦的任务是移植操作系统，我的任务是对流水线做出相应改动。要移植操作系统，首先是增加 TLB 模块，包括实现 TLB 指令，用 TLB 做地址转换。其次是搭建 SoC，难点在于各种外设控制器。TLB 模块江学谦很早以前就完成了，只是我一直没有把这个模块加入到流水线中去，因为怕增加以后影响主频，性能有所下降。一早上时间，我便增加了需要用到的几条 TLB 访问指令，但是如何将 TLB 加入流水线做地址转换却让我没有了头绪。江学谦给我讲明白 TLB 模块的用法后，我对流水线的接口进行了一系列改动，修改部分的粒度非常大，尤其是访存接口与原来已经完全不同，TLB 将代替原来的流水线访存接口与转接桥模块进行连接。其次是新添加的指令会修改原有异常类型，并带来新的异常情况，最后还有转接桥的地址，Cache 属性也得做出改动。

增加完 TLB 访问指令，还需实现非对齐的访存指令，为此也修改了转换桥和龙芯开源代码中的字节使能端口。非对齐访存的指令实现起来比较困难，光是想明白字节使能信号的值就费了很大的劲，既要考虑大端小端，也要考虑左对齐右对齐，截高位截低位的问题，查了一早上手册才彻底把指令的功能弄清楚。

修改过后，江学谦编写了一段小程序对改动模块进行测试（会写测试程序很重要，反正我不会）。但是 TLB 模块还是出现了问题，在于我把旧的由流水线提供的访存使能信号和新的由 TLB 提供的访存使能信号混淆，而且 TLB 会引起异常的情况也没有考虑完整。总之又调整了两天，这部分内容才真正开始工作，实现过程比我们想象中的要麻烦很多。

2018.8.28 尝试添加 tlb 访问指令

1. 添加 TLB 指令

1.1 在 id 阶段识别 tlb 指令，不至于发出无效指令异常

1.2 在 ex 阶段识别是 tlb 指令，然后通过 cmd 操作发送至 CP0。

通过函数 get_cmdvalid 判断 mtc0、mfc0 以及 tlb 指令然后使访问有效

通过函数 get_cmdop 给出访问操作.

cmd_no、cmd_sel、cmd_data 暂且保存为 mtc0 和 mfc0 的赋值

1.3 在 mem 阶段接收 CP0 的反馈，判断 tlb 是否触发异常。

更改了 mfc0 写入寄存器值的判断情况

通过函数 get_exrvalid 得到异常有效信息主要有 tlbwi、tlbwr、访存不对齐

通过函数 get_exrtype 得到异常类型

2. 添加 tlb 至流水线，实现访存地址转换

2.1 例化全部 CP0 接口，添加 bit_mips 中 tlb 与外界接口，原接口保持不变

2.2 IF 阶段增加 tlb_miss 、tlb_v、tlb_kern、iskernel 接口。判断是否引发异常

首先通过函数 get_exr_valid 判断是否产生了异常

通过函数 get_exrtype 判断异常类型

同时更改发出访存信号的时序逻辑，其访存信号与 exrvalid 相关联

2.3 ctrl 模块加入!tlbi_ready 的暂停信号和!tlbd_ready 的暂停信号

2.4 mem 阶段增加 tlb_miss 、tlb_v、tlb_d、tlb_kern 接口

2.5 更改 get_exrvalid 函数，加入 tlb 异常的判断

更改 get_exrtype 函数，加入 get_exra0 函数

3. 更改类 sram 接口，添加 cache 接口，其赋值规律同 addr

4. 更改 axi 转换口， 添加 cache 接口， 更改其 axi_cache 属性

5. debug id 阶段识别 tlb 指令时 func 的宏定义出错

6. 修改转接桥中的 byteenble 选项

6.1 在 dbus_sramlike 中增加 byteenable 的输出接口，直接将输入的值赋给输出

6.2 在 axi 转接桥中增加输入 byteenable 的输入接口，将其值赋给 axi_strb

7. 增加 lwl，lwr

7.1 id 阶段识别 lwl, lwr，因为读取的是 rs、rt 寄存器的值，所以不需要特意更改

7.2 ex 阶段将 rt value 赋给需要写回的寄存器的值

7.3 mem 阶段 get_mem_read 函数需要加入 lwl lwr

get_mem_byteenable 需要加入 lwl lwr

7.4 mem 阶段修改 get_extend_data 加入参数 write_data 为原先寄存器的值对于 lwl 和 lwr 直接用 byteenable、读取内存的值、原寄存器的值来计算 extend_data

8. 增加 swl, swr

8.1 id 识别 swl swr

8.2 ex 修改 mem_wdata 函数加入 swl swlr 需要写入内存的数据同 sw

8.3 mem 修改 get_mem_byteenable 同 lwl lwr

9. 调试 bugs，对于查询 tlb 指令，判断其异常类型的方式若发出 read 或者 write 指令，则开始查询 tlb，此时需要等待 tlb 返回的 tlb_miss tlb_v tlb_kern 等等参数。根据情况判断，但前提是发出了 read 或者 write 信号

10. 修改 TLB 查找结果，若查到完成则复位查找寄存器

11. 修改 cpu_axi 转换桥 araddr 和 awaddr 地址应该直接赋为 do_addr，不能直接将输入的地址给他，否则地址只能保持一个时钟周期

2018.8.30

1 修改 ibus_read dbus_read dbus_write 会受到异常影响的情况，

因为这些信号并不是真正意义上直接用来访存的信号。

故其应该只受到地址错误的影响，即此时不去查 tlb

1.1 if 阶段修改时序逻辑里面的 s_ibus_read 值的判断条件，

仅当地址不对齐时不应该去查 tlb

1.2 mem 阶段 mem_read mem_write 影响其的判断条件

为 input_exr_valid || is_AdEL || is_AdES ，即之前若产生异常或者此时地址错误，

则不再查询 tlb

2 tlb 指令会产生 stall，故需要将 CP0 的 stall 请求接入 ctrl 模块

修改 tlb 异常的判断情况，tlb_v 应该等到 tlb_ready 给出时再判断。

所以将 tlb_ready 传入 if 和 mem 阶段，作为判断异常条件之一

3 修改 mem 阶段查询 tlb 的时候其暂停信号应该为 mem_stall_req，让整个流水线暂停

4 原设计中 mem 阶段的类 sram 未做异常的应对，

原先真正发出访存信号的组合逻辑并未加入异常判断。

增加 exr_valid 到发出访存信号的组合逻辑模块中作为判断条件之一

表格中是对流水线改动的日志。我习惯了把这些改动记录下来，因为这样有助于查错，如果出了 bug，可以一个个核对，知道自己对哪些部分做出哪些改动，否则真的有可能出现最后查不出 bug，然后只能直接把这段新增的功能全部去掉的情况。

第四节 搭建新的 SoC

流水线改动完成后，江学谦只完成了内核态 Linux 的移植，能够在 qemu 中运行到一定的状态。我让他先放下这个工作，帮忙搭建新的 SoC 一运行操作系统。首先我们决定采用从 flash 处启动系统的启动方式，于是江学谦便自己写了一个 flash 控制器模块。

我：为什么不用现成的 IP 核。

江：现成的 IP 核找不到可以直接读 flash 的。

我：我看 NaiveMIPS 的 SoC 里面有 flash 控制器的 IP。

江：能找到的都是需要执行一段程序来读写 flash，这样就需要先执行引导程序，但是我们的引导程序本身就是要放在 flash 中，所以需要直接读写 flash 控制器，将 flash 映射到地址空间，让 CPU 可以像读内存一样直接读 flash。

我：哦哦（虽然不太懂，但是觉得好厉害的样子）

添加完 flash 控制器，就到了 DDR 控制器。原本我们都以为只要参考 NaiveMIPS，配置好 IP 核的参数就能够使用，但事实并非如此。江学谦在开学前一周里看过 NaiveMIPS 的参数配置，发现其实不太对，但竟然也正常工作了，于是他拿出一本 Samsung DDR 的手册让我照着配置。配完参数后，其实我们也不知道是不是正确的，因为可以找到的参考文档不全。其次还有外部引脚和时序需要连接，这部分也没有任何参考，因为每个 SoC 的设计不同。生成 DDR 控制器的 IP 核叫做 MIG(memory interface generator)，有内部时钟，外部时钟，自己生成的时钟等等各种时钟，因为文档不全所以不知道这些时钟的具体作用，最后还是通过参考已有的设计才慢慢摸索出这些信号的意义。

UART 直接用的 IP 核，是最简单的外设，直接连上 AXI 接口就能使用。我们暂时只需要那么多功能，搭建完成后，第一步是测试 flash 控制器。毕竟是自己写的程序，本身还是存在有许多 bug。刚开始时发现 flash 控制器不起作用，一点反应都没有，但是复查了很多遍代码，还是没能发现问题。江学谦怀疑是不是实验板上面的 flash 本身出现了问题，于是我们拿起实验室里示波器，直接从 flash 的物理引脚采集信号，在我举了一早上的示波器探针后，终于发现是因为对于 SPI flash 引脚 DI 、DO 理解反了导致的问题，于是通过调整引脚，终于看到了从 flash 中读出的数据。后来又陆续找到了字节序错误和寄存器忘记清零的 bug 后才算真正能把 flash 控制器用起来。

接着开始测试 DDR 控制器，还是通过自己写的程序来测试（临时起意写的，当时忘了整理，现在不见了），但是 DDR 并没有正常工作。就在这时，向老师给我发微信，问我能不能在决赛前一个星期，也就是 9 月 13 号做一遍展示。经过两天不断的调试打击，我实在没有信心预测 10 天以内能不能把系统搭建好，所以只能回复说不可预知，如果有好的结果的会及时通报。

继续调试 DDR 时，我们发现 MIG IP 核有一个 Init_calibre_done 的信号，便猜测启动过程需要初始化，启动完成后会输出相应信号，观察 ILA 波形后我们发现 MIG 确实有输出这个信号，说明 DDR 正常启动并初始化了。开始时我们并没有注意到初始化这个问题，估计程序都已经执行完了，DDR 还没有启动起来，所以必然不能正常访存。后来我们将 MIG IP 核输出的初始化完成信号连接至系统的 reset，然后 DDR 启动起来后再撤销对系统 reset 就解决了这个问题。DDR 控制器正常工作后，整个 SoC 系统也就搭建完毕。

第六节 被性能耽误的 Linux

系统搭建完成后，江学谦立马写了一个引导程序，把移植了一半的 Linux 一起烧录到 flash 中，想看看 Linux 是否能够运行到 qemu 中运行到的状态，以此作为系统的一个测试。此时我们用上了官方提供的 flash 烧录工具，非常的好用，所有就干脆把烧录 flash 的 bit 流固化到实验板上。Linux 的启动信息迟迟没有从串口显示出来，于是我们查看 ILA 抓取到的信息，才发现系统如今还在执行引导程序。于是，我们把实验板放在桌上，趁着系统加载的时间出去吃晚饭，希望回来时能够看到 Linux 运行启动的信息，然后就结束今天的工作回去休息。回来时系统如愿启动到在 qemu 中运行的状态，但是结局却出乎我们的想象，因为整个过程维持了将近半个小时，也就是仅把 Linux 加载至内存中就花了半个小时。在 qemu 中运行整个程序不过是几秒钟的事，然后江学谦又在引导程序中加入了测试信息，计算了一下实际的加载速度。添加完 TLB 和外设后，如今系统的效率低得超乎想象，我们不知道是怎么回事，系统能够正常工作，但是效率很低，而且 Cache 好像并不起作用，当时我们没有再深究下去，觉得就是流水线的设计问题。按照这样的情况，即使再花时间去移植 Linux，也很可能会因为 Linux 本身的时钟中断处理程序就耗尽所有系统资源，导致 Linux 崩溃。所以此时我们就不得不放弃继续移植 Linux 这一条路。

第七节 兜兜转转原地转圈

我：既然移植 Linux 不行，那么不如还是退步回去移植 ucore 吧？

江：但是 ucore 本身也有时钟中断处理，需要的性能也不低。你这个估计只能跑嵌入式操作系统了。

我：（我这个？）好吧，那你打算怎么做。

江：移植嵌入式系统很容易，可以不急，所以我还是先来改善系统的性能。

于是，江学谦便重新回到设计 Cache 的道路，这是目前唯一不用改动流水线而提高性能的方法。我没有意见，但是表示我对实现 Cache 帮不上忙。

趁着这段时间，我曾想试着移植 ucore，但是当我拿着这份代码，却不知道该如何下手。后来我硬着头皮，改动了串口地址，改了 Makefile 中的编译目标指令集，编译完准备下板时。江学谦告诉我还是先在 qemu 中调试后再下板，于是他又把静态编译好的 qemu 拷给我，让我配合 gdb 调试。但是我真的不太会用，看着我手残的样子，江学谦又放下手中的活计，帮我调 ucore，找到了一个像是引导程序和操作系统的代码覆盖问题。就和我们当初想的一样，调试 ucore 很复杂，所以我们两个都放弃挣扎了。

此时我们的目标是如果江学谦能够把 Cache 写出来，提高系统性能，那么我们就继续跑 Linux，不然就跑嵌入式操作系统，总之要有一个可以展示的作品。那段时间压力很大，时间又紧迫，感觉整个项目又脱离了自己的掌控，到处乱串，最后能否做出来得靠运气。陆老师和王老师来到实验室看我们的进展情况，我们也如实说了如今所面临的问题。老师们给了我们很多鼓励，让我们放宽心，毕竟是第一次比赛，能够有这样的成果已经实属不易。后来陆老师和向老师说起我们的情况，向老师听后希望我们去清华给他展示一遍，因为他觉得可能是我们的硬件上出现了一定的问题，才导致操作系统没有启动起来。但是我还是打电话告诉他是性能太差的原因。当然硬件上也还有问题，但是我们还远没有到达那一步，如今在可预见的范围内都知道即使移植过去也不会有好的效果。当说起我们可能只能运行一个嵌入式操作系统的时候，向老师还是感到一些失望。

第八节 柳暗花明又一村

江学谦着手设计 Cache 时，我闲着没事做，便试着把《自己动手写 CPU》书里面介绍的 ucos-II 移植到我们的系统中来。嵌入式系统移植起来工作量并不算多，主要是中断这部分需要做出修改，但是后来我发现如果要复用一部分已经移植好的 ucos-II 的代码，则需要把指令集拓展成 MIPS32 R1 指令集。和江学谦商量过后，他表示很赞成我的想法，因为如今可以借鉴到的移植到 MIPS 上的操作系统基本上都是 MIPS32 指令集。当初按照比赛的要求，只做了 MIPS 1 的指令集，所以移植操作系统就比较麻烦，如果实现的是 MIPS32 指令集，至少移植 ucore 就会变得简单许多。

于是我又花了一天时间将剩下的 MIPS32 指令补全，也是因为这次机会，我形成了一套添加指令的流程规范，为后来决赛实现自定义指令剩下不少时间。

2018.9.5
1. 实现 TEQ 指令
1.1 defines.h 增加 teq
1.2 id 阶段判断为有效指令
1.3 EX 阶段识别并发出异常
2018.9.7
实现指令集拓展，为 mips r1
1. sync、pref 指令不做实现
2. 实现移动操作指令 movn、movz
2.1 define.h 加入指令码
2.2 ID 阶段识别指令,special 类的 wreg
2.3 EXE 阶段将 rs 的值赋给 wdata，根据 rs 的值改变 wreg
3. 实现 clo、clz
3.1 define.h 加入指令码
3.2 ID 识别指令，给出 wreg
3.3 EXE 阶段计算 rs_value 相应 01 的个数，方法同 openmips
4. 实现 mul
4.1 define.h 加入指令码
4.2 ID 识别指令，special2 类的 wreg
4.3 EXE 阶段计算乘法结果，将乘法结果作为 get_wdata 函数的输入，低 32 位 wdata
5. 实现乘累加、乘累减
5.1define.h 加入指令码
5.2 ID 阶段识别指令
5.3 EXE 阶段，修改读取 hilo 模块，每次均需读出 hilo 的数据，接收 hilo 数据接口宽度变为 64 位
5.4 EXE 阶段新增 get_r_hilo 函数给出是否要读 hilo
5.5 EXE 阶段为 mfhi, mflo 准备的变量以及解决数据相关的 hilo_data 不变，给其赋值的函数中将变量 input_hilo_data 的宽度改变
5.6 EXE 阶段 get_mul_result 函数增加 input_hilo_data 参数，并为乘累加累减计算
5.7 EXE 阶段 get_w_hilo 需要给出写入 hilo 地址
5.8 EXE 阶段 get_hilo_wdata 给出写入 hilo 的数据
5.9 hilo 模块修改读取方式，每次均把 hilo 同时读出

5.10 修改 bit_mips.v top 模块的连线和接口名
6. ll 当作 lb、sc 当作 sb，暂时不做 LLbit
6.1 define.h 加入指令码
6.2 ID 阶段识别指令,ll 需要给出 get_wreg
6.3 EX 阶段 ll 需要计入 is_load，sc 需要给出 mem_wdata
6.4 MEM 阶段 get_memread 计入 ll, get_memwrite 计入 sc, get_membyteenbale 计入 ll sc，get_extendeddata 计入 ll
7. 修改 load 相关和 mfcp0 相关，判断其写入的地址与需要读的地址是否相关
8.实现不包含立即数的 trap 指令
8.1 define.h 加入指令码
8.2 ID 阶段识别指令
8.3 EXE 阶段加入 get_trap 函数判断是否满足 trap 要求
9. 实现包含立即数的 trap 指令
9.1 define.h 加入指令码
9.2 ID 阶段识别指令,符号拓展立即数
9.3 EXE 阶段 get_trap 函数判断是否满足 trap 要求

添加完新的指令后，我又用修改过的流水线代码适配龙芯的功能测试和性能测试框架。此时功能测试下板后出现了原先没有遇到过的问题，对于不同的延时，系统通过的指令条数竟然不一样，而且对于同样的外部激励，其结果也不一样。这时我有些慌了神，因为我新添加的指令并不会影响到原有的指令，那么问题会不会出现在 TLB 模块？江学谦一口否认了 TLB 出错的可能，因为下板后对于同样的激励，就算有错，那也是同样的结果，为什么会出现不一样的结果？应该是流水线时钟约束没有弄对导致的，于是我们把矛头指向时钟，通过查看 implementation 后的结果，却没有发现任何时钟上的问题。

对于江学谦提出的这种猜想，我不知道该如何验证，又如何解决，因为当时在仿真环境下功能测试是全部通过的。第二天，想不出办法的我只能抱着试试的心态，把比赛提供的调错方法环节都走一遍，要是真的调不出来，就只能把初赛提交的作品再提交一次，反正即使没有 TLB，没有 MIPS32 的指令，一个嵌入式操作系统也还是能够跑起来，当时的要求就是降的那么低。按照比赛提供的方法，我将仿真参数调整为与下板环境下相同的延时，然后运行仿真。江学谦让我不要做无谓的尝试，但是我还是想试试，这一次运气很好，也没有像江学谦想象的那样，出错的指令很快的就找到了，是乘除法指令，我马上就想到了前几天在加新指令的时候更改过 hilo 模块的接口和设计，定位后发现是一个数据相关问题导致的错误，第一次写的时候考虑了，但是加新指令的时候忘记考虑，然后在不同的情况下有时刚好冲突能取对数据，有时又不对。更改错误以后，整个功能测试和性能测试又能正常工作了，但是性能下降了好几倍，我和江学谦感到很疑惑，明明主频没有变，TLB 又是组合逻辑实现的，性能应该不会受到影响才对。

添加完 MIPS32 R1 的指令后，我把上次移植的 ucore 的 Makefile 又改了回来，所以 ucoe 实际上就只改了串口地址和中断号，然后直接编译出来，烧录至 flash 中。江学谦也给面子，停下手中工作，帮忙调 ucore。调试过程中，江学谦无意间发现了当初启动 Linux 时 Cache 没有工作的原因——因为添加 TLB 模块时新添加 Cache 属性端口，可以直接连接到龙芯开源的转换桥上，于是我就很暴力的把值赋给转接桥的的输出，但是 Cache 属性只能维持一个周期，之后便不会正常工作，所以 Cache 便失去了原本的功能，纯粹变成一个通路。可笑的是 Cache 没有启动，并不会对系统的功能产生任何错误影响，顶多影响一下性能，所以我们便一直查不到这个问题。修复后性能分数又回到了初赛时候的分数。

Cache 正常工作后，引导程序加载启动过程瞬间变快了很多，也让我们觉得启动操作

系统有了希望。但是这次系统还是没有正常运行起来，庆幸的是上次没有解决的问题这一次却没有出现了，转而出现的是另一个问题——有一段指令和地址没有对上，追踪发现指令能够正常读入 Cache，但是出来时却又对不上了。似乎是 Cache 这个 IP 核内部出现了问题。

无论如何，现在我们的信心提升了许多。第二天早上王老师问到我们的情况，我说有希望了。于是下午陆老师又和王老师过来看我们的工作进展，但是还是没有什么内容可以展示。开会时我在一旁和老师讨论，提出 Cache 导致的指令地址不对齐的问题，王老师给出了很多可能的意见，但是都不太吻合。后来话题又转到去南京比赛的食宿行程问题，和陆老师商议着定了住宿的酒店，会议便结束了。然后我转头过来看项目的时候，江学谦告诉我他已经把问题解决了。。。

这次 Cache 的问题在对于 burst 类型的不了解。System Cache IP 核使用的是 WRAP burst，但是我们自己写的 flash 控制器支持的是 INCR burst。两种 burst 传输的方式不同，所以导致从 flash 中取出来的指令经过 Cache 后产生地址不对齐的情况。我好奇江学谦是怎么发现这种 bug 的，因为之前我连这两种 burst 都没有听说过，他说我查了 System Cache IP 核的手册后发现的。

Cache 的问题总算解决了，但是操作系统没有按照预想情况在串口输出信息，于是我们便回去看 ucore 的串口部分代码，然后发现 ucore 的串口并不适用于 16550。可是我们明明看到 NaiveMIPS 里面就是用的 16550，然后江学谦就问我是不是把代码的分支下错了，后来去到 GitHub 一看果然是把分支下错了，本该下载的是 for-NSCSCC，但是我却下成了 ucore-fix，我果然是坑队友的一把好手。代码分支调整后，按照之前的改动，将串口地址和中断号一改便可以用了。但是这次在我电脑上却怎么也编译不过去，我看着 Makefile 里面的编译选项不知道怎么调整，江学谦和我试着把编译参数改了好几遍依然宣告失败，完全不知道作者用的是什么版本的编译器。最后我对江学谦说不如移到你的电脑上试试，运气真好，一遍就编译过了，改都不用改，我问他你用的是什么编译器，他说是我自己配置的。

再次启动的时候抱着这次还是不会成功，还要继续调 bug 的心态，没想到屏幕上刷刷刷打印出来很多东西，ucore 终于运行到了 shell 界面，心情那叫一个激动。当我准备输入几个命令测试 ucore 的时候，串口却没有任何的反应信息，ucore 像是就此卡住了。这时候其实我们挺犯难的，到底是硬件的问题，还是软件的问题？江学谦把问题定位到了一条跳转指令并未及时跳转上。

第二天一早我们又重新把昨天的情况复现了一遍。江学谦用 ILA 把板子的运行状态抓取出来，现在系统运行在一个死循环里出不来，不到一会，他的耐心显然就有些磨损了。我则把昨天的那条跳转指令的实现复核了一遍，却没有发现任何问题，后来只能试着把循环的地址一一抄写出来，然后对着反编译出来的代码找寻进入死循环的原因。江学谦照着我写的地址和 ILA 抓取的波形做了对比，发现又有一条指令和地址对不上，这才是跳转指令没有及时跳转的原因。为什么又是指令和地址对不上，上次调这种 bug 都调的快要吐血了。但是至少现在排除了是软件的问题，但是到底是流水线问题，还是 SoC 的问题？又该从哪里下手，以什么样的方式定位呢？

按照习惯，我们先把 AXI 通道的信号抓取出来，于是 ILA 的第一个窗口便布满了 AXI 的各种信号。看完后江学谦一口笃定 SoC 外部 AXI 读取回来的那条指令与地址是完全正确的。那么就再往前抓取信号，发现送入流水线内部的指令和地址已经是错误的。当时我还是觉得问题并没有出在我这边，因为流水线经过了这么多轮的测试，指令的实现上不会再出 bug 了才对，于是目光只能转到之前写的 SRAM 转类 SRAM 的转换桥上。刚开始加入的信号不完整，我还是认为这个模块没有问题，但是排除过 AXI 的问题和流水线的问题，

就只剩下这个部分了。于是我从头到尾的又把这个模块的信号量全部加进来，终于确定了是这个模块出现的问题。思来想去，我还是想不明白问题出在了哪里，于是对着代码一遍遍模拟运行状态，想当前状态机的运行情况，终于把导致错误的情况给找出来了。

真的想不到之前能够运行通过所有功能测试，但是却偏偏遇不到类 SRAM 接口同时返回 addr_ok 和 flush 的情况。其实这种情况我当时在写转换桥的时候，认真考虑过 addr_ok 和 flush 的优先关系，把 addr_ok 在 flush 前回来和把 addr_ok 在 flush 之后回来的情况都考虑进来了，就是没有考虑同时回来的情况。借江学谦的话说，所有 bug 都是因为懒导致的，如果当初能够设计清楚，后来就不会有那么多的 bug 调了。修复完这个 bug，我们又下板跑了一次，还是运行到 shell 界面卡住了，江学谦按照套路迅速定位到了转换桥的问题，然后又是情况考虑不周，但是这次前后花了不到半个小时就修复好了。终于在 9 月 12 号的晚上我们把 ucore 运行起来了。

第九节 作品的收尾工作

比赛作品的实现到了收尾阶段，剩下的时间需要把改动过的部分再加入到性能测试与功能测试框架中，然后编写报告和答辩 ppt，打包提交。陆老师和向老师让我们准备准备，把最后的工作重心放在展示上面，14 号时做一次报告，然后给我们提些意见。第二天我花了一下午的时间弄了个 ppt。刚开始做 ppt 时我便发现整个项目其实并没有太多的亮点，首先流水线写得相当普通，性能不高，也没有抠细节，所以便一笔带过，于是便决定让报告围绕着运行操作系统的 SoC 来展开，但是做完以后还是发现展示效果非常普通，而且主线很不清楚。

第二天见面时，向老师对我们说：“原来你们说要跑 Linux，结果后来又说要跑嵌入式操作系统，最后跌跌撞撞饶了一个大弯又回到了 ucore。本来我对这事都不抱希望了，但是前天你们又突然告诉我成了。要是你们当初一早就开始弄 ucore，这事是不是能够早一个星期就成了呢？” 我只能说：“还真不是您想的那样，那样的话说不定更成不了。”然后向老师告诉我们现在听说只有华科跑起了 ucore，国科大跑起了 Linux，反正北航没有跑起来，这对我们来说是个好消息。至少如今在老师心目中只要我们不掉链子，至少是个二等奖。接着我花了不到 5 分钟时间磕磕绊绊地给老师们演示了一遍，效果显然不好，然后向老师给我们提出了很多意见，包括 ppt 字体配色，增加内容和动画等等。听完我们的报告，向老师又赶着去听清华同学的报告去了，我们下午刚好有小学期的网教考试，怕赶回北理工来不及，便待在向老师的实验室里等考试开始。陆老师便和我们聊起整个学习的过程，希望我们能够把做比赛过程的一些方法总结总结让后人有所借鉴。后来向老师听完报告回来，告诉我们清华参赛队跑起了双核的 Linux，让我再次感叹了一番。回去时我和江学谦又说起了这事。

我：今年看来大家都好厉害的样子，既然双核都出来了，那么多发射什么的应该也会有吧。

江：那不一定，和多发射比起来，多核是最容易的。

我：为什么？

江：你觉得多核很难吗？其实就是先启动一个核，然后用一个核带动另一个核启动，只要多实现几条同步指令，你写的 CPU 就能开始做多核了。

我：那你能做吗？

江：给我时间应该没有什么问题。

我：。。。（大佬惹不起）

回去以后，我的任务就是继续负责写文档，打包项目。江学谦则继续做他的尝试，实

现设计好的一个流水线 Cache，实现了一半，他问我有没有兴趣一起写，因为他不太擅长写流水线，我急忙否认，表示没有兴趣，后来还是硬着头皮听完他的设计，表示不太理解，于是他也第 n 次放弃了写 Cache。后来我和他说到能不能趁着最后两天时间再给展示的 SoC 添加一点炫酷的效果，比如 VGA 显示图片等等，江学谦听后表示 so easy 啊，然后开始一通设计，各种想法喷涌而出，听得我都觉得高大上，但是我还是止不住泼了冷水说：“大哥，当初我们移植操作系统的时候也觉得没什么，但是真正做起来的时候却问题不断，何况现在 17 号就要提交作品，只剩下两天的时间了，所以还是做一点简单实际的吧。”于是江学谦把剩下的一点时间放在了 GPIO 和 VGA 上，写了一天，调了一天，VGA 最终也只是以半成品的方式提交了上去。

提交完作品后离决赛还有几天时间，不可以改硬件但却还可以改软件。原本我们打算试着移植一下 Linux，但是小学期已经结束了，其他课程都陆续开始，如果要移植也只是江学谦一个人干，我也不能催着他干活。从前到后忙碌了近三个月，我们怕都是不想再动了，于是便心照不宣，默认放弃 Linux 了。

第五章 哨声吹响

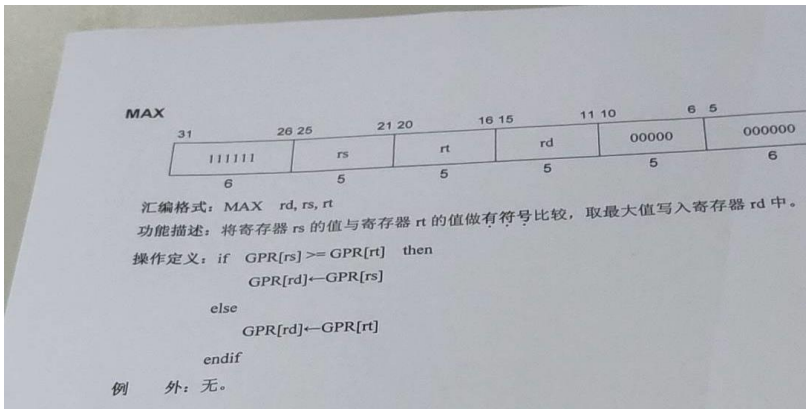
第一节 省事的新指令答题

9 月 20 号，我们踏上了南下的高铁，前往南京大学仙林校区。

时间	地点	内容	备注
9月21日 12:00-14:00	南大仙林校区	各参赛队报道	12:00 开始签到 13:59前未签到视为放弃比赛。
9月21日 14:00-17:00	南大仙林校区	25支决赛队伍现场指令集答题	答题时间为14:00-17:00，期间上交手机，不允许上网
9月21日 17:00-18:00	南大仙林校区	25支决赛队伍演示	需提前搭建好演示环境
9月22日 8:30-12:00 13:30-18:00	南大仙林校区	25支决赛队伍答辩、演示	**8:00到达实验室** 8：15-8：30 宣布评审规则、解释打分表等 每队15分钟（ppt讲解7分钟，专家问答8分钟）。答辩时进行系统演示。 **答辩结束后，答辩室门口归还实验箱。**
9月23日 8：30开始	南大仙林校区	颁奖典礼	

天气的突变加上近来的操劳紧张，让我身体有些欠佳，刚去到南京就开始发烧。第二天中午我还是打起精神去到比赛现场，准备指令集答题，因为流水线指令的实现从头到尾都是我负责的，怕江学谦一个人搞不定。报到时，我们领了参赛的服装胸牌，还抽到了 3 号的答辩顺序，就进到比赛场地开始准备。等待比赛开始期间，我头痛得趴在桌子上休

息，江学谦劝我去趟医务室买药，我说人生地不熟的，比赛也马上开始了，还是能撑过去的。后来江学谦帮我找了组委会的工作人员，在比赛开始前的半个小时，有个工作人员跑过来说要带我去南大的校医院，我便匆匆和他赶去了，花了半个小时才把整个流程走完一遍。幸好有南大的袁老师开车送我赶回比赛现场，才没有错过太多内容。回来时各比赛小组已经拿到了新的测试框架和需要实现的新指令。今年添加的新指令是 MAX 指令，R 型指令。看到这个指令时我们就笑了，因为真的很简单。前十分钟龙芯负责比赛的邢老师简单解释了一下新的框架和指令，然后让我们先把新的测试文件初始化到 RAM IP 核中，一边加新指令，一边综合 RAM IP，这样会省时间。之后，参赛队便各自着手添加指令。比赛用的机器是自己自带的笔记本电脑，我们队用的是江学谦的笔记本，于是动手写代码的人便是他。刚开始时江学谦一上来就往执行阶段加代码，我说你这样不行，首先译码阶段就会报无效指令异常，应该从头一个阶段一个阶段的来。于是按照我的要求，先将新指令的指令码在头文件中做宏定义，然后依次在译码和执行阶段识别指令并作相应操作，前后改动代码不超过 20 行，花了将近 10 分钟就全部弄好了，于是我们就一边综合一边仿真。



新的测试用例是在原有的 89 条指令基础上将第一条指令替换成 MAX，于是还是 89 条指令。仿真环境下很快就通过了第一条指令的测试，意味着我们添加的指令没有任何问题。按照要求验收时需要查看的是仿真环境下某一特定地址处的写回结果，趁着综合还没结束，我们查看了对应该地址的写回结果，也完全正确。不过奇怪的是，输送给执行阶段的比较值是错的，但是结果竟然是对的。我们俩百思不得其解，陆续又添加了其他信号进来做比对，但是还是相同的结果。为什么显示两个比较的数是 0x12340000 和 0x12340000（具体的数我也忘了，编了个类似的数说明），按理由相等输出的也是 0x12340000 才对，但是写回结果却是 0x1234abcd，而且还是正确的。我只能说不管了，掩盖住事实的真相，然后只让他看结果就是了。不一会综合的结果也出来了，我便把实验板拿出来准备下板测试，看着我们那么早就拿出实验箱，周围几个队都有些坐不住了。下板测试后依然是正确的，于是我们就再也没有想过那个问题，也许是我们查看的方式不对，也许是软件有问题？总之这件事成为了一桩千古悬案。比赛两点钟开始，五点才结束，组委会为了不让大家有压力，三点才开始验收，验收完也不能走，但可以在在座位搭建系统，因为五点钟以后还有专家组过来看演示。我只能坐在位置上发呆，让实验板一遍遍跑动测试用例，等着开始验收。后来邢老师走到我们旁边，我小声问他能不能验收，他过来一一查看了仿真和下板的情况，2 点 50 分的时候终于把指令集测试都验收完毕了。

第二节 为 ucore 添加新功能

由于 ucore 的演示功能太少，只有简单的几个命令可以在 shell 界面输出些信息，比如 ls，cat 之类的，于是趁着仅剩的半天时间，我让江学谦在往里面添加一些新的功能，他也很快就加了一个计算斐波那契数列的命令。我说这样也太不炫了，能不能把 LCD 弄起来，比赛之前我按照 NaiveMIPS，尝试过一遍，还差一点就能完成。于是江学谦将 NaiveMIPS 的 LCD 功能 IP 核和代码复制出来，准备往我们搭建的 SoC 中移植。因为我们整体 SoC 的时钟和外部的接口与 NaiveMIPS 并不相同，所有只能按照自己的想法搭建，但是之前在我看来也并不是难事，无非是把模块代码拖进去连接时钟，配置引脚的事。到了比赛结束时，江学谦也已经按照和我相同的想法完成搭建，开始综合项目了。本来想等待会专家组过来视察时就可以展示给他们看，但是综合过后项目却报错了。此时我们的电脑又开始变卡，综合一次少说一个小时，所以展示 LCD 的计划便泡汤了。专家组按时来到现场，然后在工作人员的指引下依次查看了各组的演示，其实并没有多少组有展示的项目，最厉害的就是清华的双核 Linux，国科大的 Linux 加触摸板小游戏，还有南大的超级玛丽，其他的便是跑引导程序，或者推箱子，贪吃蛇之类的小游戏，实在没有的就是记忆小游戏。我们组的地理位置刚好被排到了最后一个，展示的时候我大概介绍了一下操作系统的启动过程，然后 ucore 启动花了太长时间，第一波老师直接就走了，走了。。。后来清华的刘老师过来看到我们启动了 ucore，问我们花了多长的时间弄好的。当时我能看得出他脸上的惊喜和诧异，然后我说 9 月分才开始弄的，前后两个星期左右吧。他止不住向周围老师介绍这是北理工的，又对我说：“当时看你们在清华的时候好像做得挺艰难的样子，怎么感觉一下子又做出来了呢？”我笑了笑不语，是的，做的确实挺艰难，但却不是一下子就做出来的。

等我们展示完成后，江学谦告诉我他找的了 LCD 的问题所在。因为系统的复位信号一直采用的是 Processor System Reset IP 核产生的复位信号，而且这个复位信号指定采用一种内部特定的 flip flop 才能驱动，不能够直接接到 IO 设备上，直到那时我才知道为什么 NaiveMIPS 放着好好的能直接产生 reset 信号的 IP 核不用，非得自己实现。我们的解决方案是在 IP 核产生的复位信号和 IO 引脚间再加入一个用 Verilog 实现的寄存器。经过改动，我们将原本能够在 NaiveMIPS 上的 ucore 运行我们搭建的 SoC 上，并且在 LCD 显示出实现设定好的图片。为了进一步做出改动，我说能不能把图片换一换，甚至能做成现场拍摄

一张照片然后显示在 LCD 上，江学谦回答说可以。我找了张北理工校徽的图片，然后按照他的要求裁成相应的分辨率大小。江学谦找出他在图像处理课上做的大作业，然后把图象转成了相应数据，准备移植到 ucore 中去。原本打算放在用户态然后通过串口加载，尝试了一次后发现空间太小又放弃了。饥肠辘辘的我们决定还是先吃晚饭再接着干活。吃晚饭的时候，陆老师和我们透露了她收集的情报信息。陆老师和西工大的老师同学们聊过一会，西工大的老师推荐了一本书，但是当时记不住名字。西工大在两届比赛中都获得性能第一名的成绩，今年更是包揽了前两名。吃过晚饭后，回到住宿的地方我便休息下了，也没有再对换图片的事情抱太多希望。江学谦则在在办公桌上挑灯夜战。记得很多次我醒来看见窗旁的台灯一直亮着，以为天色还不算晚。然后半夜我起来喝水时江学谦指了指 LCD 屏幕上的图片，一个北理的校徽赫然显示在上面，我欣喜的问他怎么做到的，他说把图片换到内核态了。

第三节 开始我们的表演

因为是第三个答辩，所以第二天早上我们便早早的起了床。去到吃早餐的地方时，便看到陆老师和西工大的老师相谈甚欢，看来这趟行程我们取了不少经呢。吃早餐时我问江学谦他昨晚几点睡的，他说一直做到四点多才睡的。去到比赛现场时，答辩已经开始了。第一个答辩的队伍是清华，我说好险没有抽到第二个，否则评委老师心里面肯定会有落差的。我们在等候室等待工作人员的通知，期间将答辩的过程有又走了一遍。很快，就轮到我们答辩。按照先前的分工，我负责讲 PPT 内的所有内容，然后江学谦做展示和回答问题。PPT 的内容我已经看过无数遍，演讲稿也都烂熟于心。整个答辩是 15 分钟，讲 ppt7 分钟，专家提问 8 分钟。原本我以为展示系统的时间不计入 7 分钟的讲 ppt 时间，然后答辩前一天才得知这一消息。当时也觉得没有问题，因为平时正常讲完 ppt 是 6 分半，再加快点速度就行，而且 ucore 本来命令就少，如果提前加载好，半分钟足够展示了。因为我们留给展示的时间很短，所以旁边的计时老师可能真的觉得我会讲不完还是咋的，隔三岔五的就给我提醒一次时间，搞得我原本可以很流利的讲完，还是不小心停顿了几下。在我展示 ppt 的时候，江学谦在一旁启动系统，最后还有一分钟的时候，我赶忙把串口线插到我的电脑上，按照计划就应该开始输命令展示，但是却看到 ucore 还在启动过程，当时江学谦指了指他电脑屏上的信息，我便知道发生了什么事。ucore 启动以后还是会时不时崩一下，然后又得重启。当时我有点冒冷汗，因为平时至少 2 分钟才能完全加载出整个系统。幸好，系统是在几分钟前崩溃的，等了 10 多秒钟，就进入了 shell 界面，江学谦赶忙输入了 ucore 的几个指令，但是遗憾还是剩下了 LCD 的功能没能展示出来。但是至少证明了我们的 SoC 是能够运行操作系统的。

展示完接着便是提问环节，当时现场坐着几十个老师，每排座位四个人，整整一个小教室。第一个问题老师便就着我在 ppt 中讲到的流水线的缺憾展开。我在 ppt 中提到由于阶段功能过多，导致流水线的主频难以提升。原本我对这个问题的了解限于当时主频卡在 80Mhz 的那条长长的路径上。

评委老师甲：是那个阶段导致的？

江：是执行阶段的逻辑过多。

我：（不是译码阶段？）

评委老师乙：你们是怎么知道这个阶段的逻辑多的呢？

江：通过分析 Vivado 综合过后的电路图发现执行阶段所经过的路径最多。

我：（我以前怎么没有发现？）

评委老师丙：那你们在执行阶段里面做了什么呢？

江：部分译码，计算结果，乘除法等。

评委老师甲：你们在执行阶段还做译码？

江：对。

其他老师：op code， func code 这些的。

评委老师乙：那你们乘法模块怎么做的？

江：乘法是自己做的，用的 DSP 综合出来。

评委老师乙：那除法模块呢？

我：除法模块用的开源代码，期间会阻塞流水线。

其他评委老师：所以其实流水线并没有流起来，做成多周期的了。

江：是这样的。

我：（是这样的？）

评委老师丁：刚才看到你们那个流水线出来又是转换成类 SRAM 又是转换成 AXI，中间还接 Cache，为什么要转来转去呢？

我：（急忙再打开 ppt，转到那一页）。

评委老师乙：那个 Cache 是什么 Cache？（丁的问题已被淹没）。

我：直接用的 Xilinx 的 IP 核（敢情老师问的东西没有一个是自己写的）。

评委老师丁：那是接在流水线里还是系统的 Cache 呢？

江：系统 Cache。

评委老师甲：你们刚才说的用 ILA 调试，不用可以查出来吗？

我：不行，因为下板后系统的状态是未知的，必须通过 ILA 采集数据后才能够知道系统的状态。

评委老师甲：那么这个东西是怎么找到的呢？

我：是 Vivado 自带的一个功能。

刘老师：那你们觉得跑起 ucore 来有什么可以给大家借鉴的地方呢？

计时老师：最后一个问题。

江：首先是测试足够的充足，我们不仅用了龙芯的测试用例，而且用到了向老师给的一些测试用例。其次是操作系统不能直接下板，要配置 qemu，然后在仿真环境下调通再下板继续调试。

走出答辩的教室，陆老师迎面走过来问我们结果如何，我说有些遗憾，系统崩溃了，江学谦花了一晚上才做好的功能，最后还是没能演示。陆老师安慰我说幸好不是在演示的时候崩的。回到等待答辩的教室时，其他学校的参赛队员凑过来询问评委老师都问了些什么问题，我说老师都是具体围绕着作品来提问，都很专业和深入。然后我和江学谦谈起刚才评委老师们问的问题。

我：评委老师们问的问题我有好几个都答不上来，这次多亏了你。那个执行阶段逻辑过多和流水线没有流起来是怎么回事，

江：（惊讶）你居然不知道。

我：我在实现的时候只关注了功能，其他的完全没有关注到啊。

（于是我们又再次讨论起流水线的性能问题。）

我：如何在原本设计的基础提高性能，是不是做一个流水线内的 Cache 会好很多呢？

江：我们现在的设计，从一开始就注定不会有一个好的性能，比如用一个 ctrl 模块就控制了整个流水线的暂停。

我：没有办法，因为这是我唯一学过的解决办法，智商又不够，想不出更好的解决办法，就只能参照别人的设计。如果你能够想出来，我也同意用你的想法。

后来陆老师把话题转到了如何构造一个性能更优的流水线，如何更好地去学习相关理论知

识上。印象最深的是江学谦说要系统地学习，并对知识有个全局的把握。但是我却反问他这个系统的定义什么，怎么知道自己学习的知识是不是系统的，有没有什么书籍，资料推荐。江学谦却说没有。陆老师问他：“那你是怎么学的？”江学谦说靠平时的积累。后来陆老师又和我们说起西工大的设计，据介绍他们的设计全都是参考《计算机组成与设计》这本书，我也好奇西工大的流水线架构是什么样的。于是我找到上一届比赛答辩的 ppt，上届比赛两个队在 ppt 上展示的流水线架构和书中的确实一模一样。

第四节 成绩单下达

比赛第三天早上是颁奖典礼，各参赛队伍的成绩都是未知的，一切都还保留着悬念。陆老师前些天和我说这届比赛的第一名应该又是清华了吧，跑起来双核 Linux，我却说不一定，可能是清华也可能是国科大，因为性能分国科大差不多是清华的两倍，而且国科大的队伍也跑起了 Linux，无论如何，我觉得评委在打分环节也不至于让两个队伍的差距有那么大。决赛共有 24 支队伍，初赛时我们的成绩是第 17 名，特等奖一个，一等奖三个，二等奖六个，也就是说能进入前十名就有一个二等奖。我一直对我们拿二等奖抱有一定的希望，毕竟能跑起操作系统的队伍并不多。颁奖典礼上听取了了专家对于此次比赛的点评，今年的参数作品质量明显比去年提高了很多。听到我们获得的是三等奖时，我转过头和陆老师相视一笑，眼神里透露着些失落，但是其实这结果和我心里面预期的差不多，因为我知道性能确实很重要。整个决赛的评分标准是这样的，新添加指令 20 分，其实基本上大家都拿到了，性能分 30 分，答辩演示 50 分，其中展示的新颖独特性 20 分，答辩讲解 30 分。虽然看起来性能的 30 分并不占大头，但是按照我们和第一名 10 几倍的差距，估计只拿到了不到 3 分。虽然其他队伍没有运行起操作系统，也并不影响答辩环节讲自己如何构造的流水线，而且展示环节只要有一定的应用能够展示，也不会拉开什么巨大差距，况且 ucore 本身展示起来也就是一个串口工具，输入几个命令，并没有让人觉得有多么了不起。况且别的参赛队应该老早就开始入手写 CPU 了吧，哪像我们就花了两个月，都没来得及迭代产品。

第一届全国大学生计算机系统能力培养大赛（龙芯杯）
决赛A组入围名单

	参赛队			预赛提交		预赛复核				
序号	参赛队	参赛学生	指导老师	功能	性能	周期	频率比值	检查性能	功能分	性能分
1	西北工业大学计算机学院2队	卢柏岑、付豪、耿毓羚、龚宇航	安建峰、张萌	100	2.7569	14.526	2.295	1.201	100	2.757
2	清华大学计算机科学与技术系1队	张宇翔、王邋、刘家昌	刘卫东	100	2.69	14.75	2.260	1.211	100	2.737
3	南京大学计算机科学与技术系1队	丁浩然、陈嘉鹏、陈锦赐、郑彬	李俊、蒋炎岩	100	2.272	10.25	3.252	0.706	100	2.296
4	北京航空航天大学2队	方科栋、安万贺、李一睿、李明轩	栾钟智	100	2.0907	16.368	2.036	1.027	100	2.091
5	武汉大学计算机学院	韦业鑫、唐宇奕、李赛南、张子晗	龚奕利、肖忠付	100	3.63/1.8	10	3.333	0.607	100	2.023
6	西北工业大学计算机学院1队	许东泽、刘仕怡、刘佩、师鹏飞	王党辉、王继禾	100	1.746	23	1.449	1.205	100	1.746
7	北京航空航天大学1队	刘康旭、陈鸿超、林子义、刘恒睿	万寒、杨建磊	100	1.721	23.81	1.400	1.229	100	1.721
8	中国科学院大学1队	谭弘泽、何博、彭少辉、张北辰	黄博文	100	2.029	16.5	2.020	0.831	100	1.679
9	南京大学计算机科学与技术系2队	刘博、李洲、田永畅、欧先飞	李俊、蒋炎岩	100	1.653	17.03	1.957	0.853	100	1.670
10	南京航空航天大学计算机科学与技术学院1队	高襄、宗华 张靖棠	冯爱民	100	1.4049	20.2	1.650	0.850	100	1.403

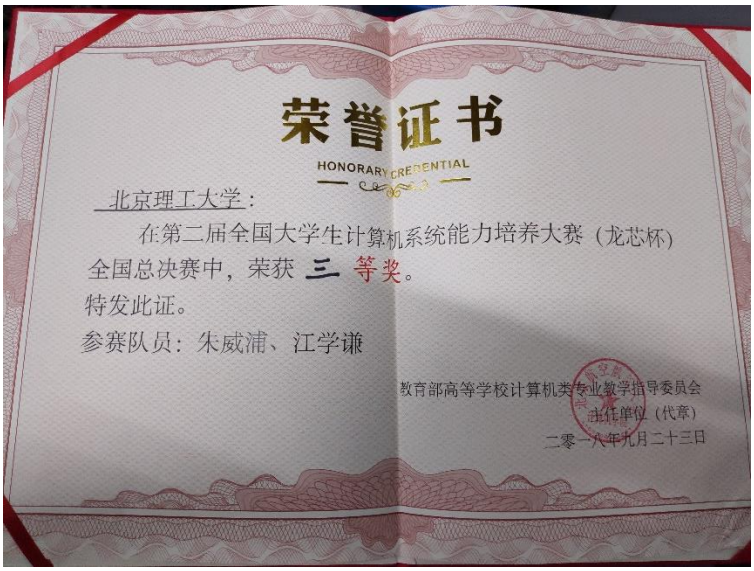
第一届全国大学生计算机系统能力培养大赛（龙芯杯）
决赛B组入围名单

参赛队			预赛复核				
参赛队	参赛学生	指导老师	周期	频率比值	检查性能	功能分	性能分
重庆大学计算机学院	赵小强、赵丰年、索鑫、吕昱峰	钟将、黄仁	25	1.333	0.420	100	0.561
中山大学物理学院1队	农姗珊、萧嘉乐、苏梓培、梁东宝	粟涛	24	1.389	0.100	90	0.139
天津大学计算机学院	肖健、徐文富、薛臻	魏继增	14	2.381	0.323	99	0.769
金陵科技学院计算机工程学院	吴官东、刁睿成、戈明月、黄潇	朱勇、王君	41.435	0.804	0.100	100	0.080
济南大学信息科学与工程学院2队	朱宇辉、李小鹏、于建勋、李玉亮	王凯、张玉璘	-	-	-	39	-
济南大学信息科学与工程学院1队	张建洪、房体育、刘志浩、韩笑	孔祥玉、刘伟峰	44	0.758	1.088	100	0.824
华中科技大学计算机科学与技术学院2队	余文梦、王锦程、王浩波、万鑫晨	谭志虎、邵志远	80	0.417	0.248	100	0.103
河北地质大学信息工程学院1队	赵颖琪、庄寅、曾学香、程谦谦	左瑞欣、董建彬	-	-	-	44	-
东北大学计算机科学与工程学院2队	唐牧天、温冬、赵庆宇、董鑫欣	于亚新、郭军	20.53	1.624	0.785	100	1.275
东北大学计算机科学与工程学院1队	丰玉霖、张旭、金敏鹏、宋凯	张高原、聂铁铮	57.5	0.580	0.906	100	0.525

第二届决赛入围名单如下：

	入围决赛的参赛队		预赛分数	
序号	队伍名称	参赛学生	功能分	性能分
1	西北工业大学 2 队	蔺嘉炜、张克、李冬	100.000	71.173
2	西北工业大学 1 队	蔡金洋、杨赟、董奕兰、王杰	100.000	70.773
3	中国科学院大学 1 队	俞子舒、杨丰远、徐易难、周盈坤	100.000	69.289
4	北京航空航天大学 1 队	程星洲、齐英杰、刘琼澳、于洋	100.000	49.124
5	南京大学 2 队	刘志刚、欧先飞、沈明杰、邱子豪	100.000	41.142
6	华中科技大学 1 队	张鑫、马翔、林力韬、梁瑾	100.000	40.671
7	河北大学 1 队	李凯、黄玉彪、朱艳慧、汪林	100.000	40.150
8	清华大学 1 队	唐适之、蔡子熙、刘熙航	100.000	36.760
9	济南大学 1 队	胡洧、许亦文、张建洪、张文艳	100.000	36.312
10	河北大学 2 队	候苑博、李亚凯、张洁、吴雨桐	100.000	35.258
11	北京航空航天大学 2 队	段逸骁、胡俊崧、王科翔、王潜	100.000	34.705
12	华中科技大学 2 队	张洋铭、雷宇、胡皓胜、周元辉	100.000	29.215
13	重庆大学 2 队	吕昱峰、张启航、葛胡军、蔡林晋	100.000	19.162
14	重庆大学 1 队	王千里、冉浙江、刘朕、李之尧	100.000	16.425
15	天津大学队	薛臻、储旭、施思雨	100.000	12.196
16	中国科学院大学 2 队	张旭、符岗、王苑铮、袁峥	100.000	10.288
17	北京理工大学 1 队	朱威浦、江学谦	100.000	5.909
18	东北大学 1 队	吕振峰、沈希乐、廖子墨、齐洪雨	100.000	4.755
19	东北大学 2 队	张家逢、周泽帆、刘涛、高小萌	100.000	4.692
20	南京理工大学 2 队	杜元民、杨飞明、孙维华	100.000	1.389
21	北京科技大学 1 队	邢其正、王硕、姚广宇、陈搏	100.000	1.121
22	武汉大学队	贺进年、唐宇奕 、王浩宇、黄睿	100.000	0.972
23	南开大学 1 队	王理治	100.000	0.966
24	哈尔滨工业大学（威海）1 队	柴可、李佳欣、陈宣合、王宣哲	100.000	0.899
25	南京航空航天大学	余浩岳、孙峰、李悦欣、宗华	100.000	0.885

这样的结果让我心服口服，因为我们确实存在很大的短板，比赛的比较本身就可以从多个方面进行，既可以从提高性能入手，也可以从搭建 SoC，运行系统和各种应用程序入手。三等奖的奖品是两本书，一本是胡伟武老师编著的《计算机体系结构》，一本是 Patterson 和 Hennessy 写的《计算机组成与设计》，正好就是西工大老师推荐参考教材。


荣誉证书

HONORARY CREDENTIAL

北京理工大学：

在第二届全国大学生计算机系统能力培养大赛（龙芯杯）全国总决赛中，荣获 三 等奖。

特发此证。

参赛队员：朱威浦、江学谦

教育部高等学校计算机类专业教学指导委员会

主任单位（代章）

二零一八年九月二十三日

下午，我们坐上回程的高铁，告别为期三天的南京比赛之旅。

第六章 路漫漫其修远兮

到这里，维持了一年的比赛工作也全部结束。这是我所参加过的耗时最长，难度最大的一场比赛。向老师经常问我们参加这个比赛值不值得，能够坚持下来的原因是什么？江学谦回答是兴趣爱好，因为感兴趣，是自己想做的事所以坚持了下来。我的回答和他一样，但是除此之外，我觉得能坚持下来的原因还在于有一个好的队友，从队友的身上我学到了许许多多新的东西，不同的思想方法冲撞出不一样火花，对知识产生不一样的理解，还在于有一群认真负责，志同道合的老师，一直很 push，督促我们不断向前迈进，帮助我们解决困难。无论如何，我们从头到尾都没有抱着功利的心态参加这个比赛。记得有一次我和向老师说拿不拿奖无所谓，向老师说奖无所谓，但是你所做的事被不被别人认可有所谓。最后虽然没能实现被人认可这一目标，但是在我看来这不过是个包装精美的盒子，而我所在意的是自身的能力又没得到提高，知识有没有得到增长这枚珠子。通过这次比赛，我知道自己懂的东西很多，但是不懂的东西却更多，愈发的发现还有更多更宽广的领域值得自己去学习去探索。总而言之，路漫漫其修远兮，吾将上下而求索。

番外篇

最近一直在思考一个问题，教育方式的本质是什么？是为一个本来就懂很多，不用教也会的人定一个目标，培养到能够完成这样一个有难度的比赛，还是把一个原本什么都不会的人培养到能够顺利完成这样的比赛呢？参加完比赛后，老师让我们总结经验，给后人做参考，虽然我也长篇大论写了三万字把整个比赛的过程记录了下来，但是还是有很多的东西没能够记录下来，因为很多东西默认了接下来参加比赛的人已经会了，可是实际上真的能够保证这一点？虽然如今我作为参赛队员之一完成了整个比赛，但是如今让我自己一人完整复现一次我认为有些环节我还是无法做到，比如看懂整个 ucore 的代码移植，编写编译链接脚本，迅速看懂 IP 核文档搜寻想要的信息，编写一个能够映射至内存的 flash 控制器，快速定位问题等等。其实我也很希望自己能够什么都懂，但不是每个人都能像江学谦一样，有一个很好的基础，然后站得高，看的远。我希望他们不要再重蹈我的后辙，按

照原本的设计和学习线路来做，因为我发现第一次学习的内容真的会深刻影响短时间内的作品，这点从西工大和北航的经历也可以看出来。西工大的队员都是大二的学生，为什么我们这批大三的老鸟还比不过，因为人家的课程设计如此，手边的学习参考资料就是这样，照着这样的设计来做就会有这样的结果。虽然不知道北航的课程和学习资料是什么，但是从他们两届四支队伍相同的七级流水线加神经网络设计也可知一二。我在做流水线方面给不了更多的意见，因为我的设计与主流设计不同，专业的老师们甚至会有些不了解。至于搭建 SoC 和移植操作系统，那多是江学谦的功劳，若要他给什么意见，我倒可以照着他的口吻回答一番——自己去学。无论如何，我相信只要拥有良好的自学能力，一颗坚持的决心，不放弃的恒心，不气馁的信心，终究会获得一个好的结果。

完结