**实验五报告**

学号 2017K8009929044

姓名 李昊宸

箱子号 33

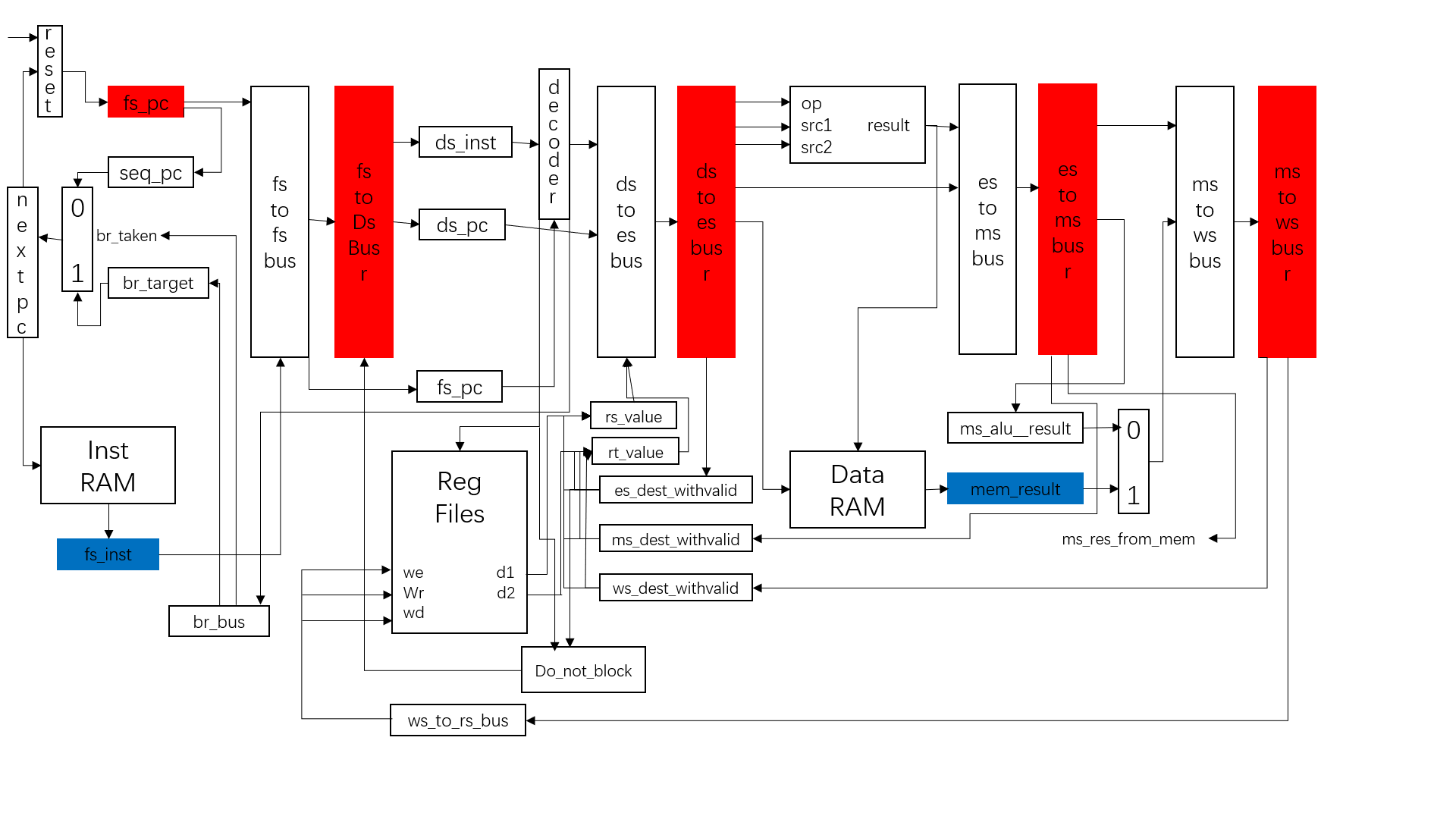
一、实验任务（10%）

1.. 加深对流水线结构的理解。

2. 学会使用阻塞的方式解决相关引发的流水线冲突的方法。

二、实验设计（40%）

（一）总体设计思路



图一 mycpu模块结构图

上周实验所实现的部分为使用阻塞的办法解决写后读相关的问题。当EXE,MEM,WB阶段出现向寄存器堆写入数据的目的寄存器号与ID阶段从寄存器堆中读出数据的源寄存器号相同时，部分情况下我们需要考虑将ID取值模块阻塞在这里，直到后续模块将数据写入之后再进行读取操作，否则会出现数据还未写入就已经被读取的错误。在这里，我们采取的方式是分别从EXE,MEM,WB模块中向ID模块引入三条组合逻辑总线，将当前的目标操作寄存器号送回ID模块，与ID模块源操作数进行比较，若符合阻塞条件就将ID模块的ds\_ready\_go修改为0。

这一周的实验是在阻塞的基础上进行补充，采用前递的方法减少不必要的阻塞。思路就是将执行级、访存级、写回级出口端的将写回寄存器堆的数据传回译码级的寄存器堆读出口，在此处使用四选一数据选择器选出运行指令所需要的正确的数据，从而减少阻塞的时长。

（二）重要模块1设计：ID译码模块

1. 工作原理

从IF传入的信号，经过译码逻辑后，被分解成每个部分的信号后，和PC有关的信号被返回ID，其余和执行有关的信号被传递到EX模块。

1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟 |
| reset | IN | 1 | 复位信号，拉高为复位 |
| es\_allowin | IN | 1 | 允许数据流水进入EX |
| ds\_allowin | OUT | 1 | 允许数据流水进入DS |
| fs\_to\_ds\_valid | IN | 1 | 从IF向ID的数据传输是否可以进行 |
| fs\_to\_ds\_bus | IN | 64 | IF向ID的总线 |
| ds\_to\_es\_valid | OUT | 1 | 从ID向EX的数据传输是否可以进行 |
| ds\_to\_es\_bus | OUT | 136 | ID向ES的总线 |
| br\_bus | OUT | 33 | 从ID模块返回的下一步PC跳转信号 |
| ws\_to\_rf\_bus | IN | 38 | 从WB模块返回的存入regfile的数据 |
| es\_dest\_withvalid | IN | 39 | 从ES模块返回的目的寄存器号和阻塞信号和数据和选择信号 |
| ms\_dest\_withvalid | IN | 39 | 从MS模块返回的目的寄存器号和阻塞信号和数据和选择信号 |
| ws\_dest\_withvalid | IN | 38 | 从WS模块返回的目的寄存器号和阻塞信号和数据和选择信号 |

表一 ID模块接口定义

1. 功能描述

因为存在写后读相关的取值问题，在lab4中使用阻塞方法解决相关问题。引入三个目的寄存器号传递信号：es\_dest\_withvalid、ms\_dest\_withvalid和ws\_dest\_withvalid，作用为传递EXE,MEM和WB阶段正在运行中的寄存器号，随后与ID阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在lab5中，为了减少不必要的阻塞消耗，我们可以考虑把发生写后读冲突的寄存器的数据直接在流水中前递到译码级模块，也就是说，采用旁路（bypass）的方式，以取代堵，不必傻傻的等到数据存到寄存器内之后再取出来，而是将已经计算好的数据在还没存入之前就允许其他指令使用。具体的实现见实现部分。

（三）重要模块2设计：EX执行模块

1. 工作原理

从ID传入的信号，被一一取出，一部分进如alu中进行计算，另一部分作为数据RAM的使能和控制信号被输出到数据RAM处，等待下一拍上升沿被更新到MEM阶段。

1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟 |
| reset | IN | 1 | 复位信号，拉高为复位 |
| ms\_allowin | IN | 1 | 允许数据流水进入MEM |
| es\_allowin | OUT | 1 | 允许数据流水进入EX |
| ds\_to\_es\_valid | IN | 1 | 从ID向EX的数据传输是否可以进行 |
| ds\_to\_es\_bus | IN | 136 | ID向ES的总线 |
| es\_to\_ms\_valid | OUT | 1 | 从EX向MEM的数据传输是否可以进行 |
| es\_to\_ms\_bus | OUT | 71 | EX向MEM的总线 |
| data\_sram\_en | OUT | 1 | ram使能信号，高电平有效 |
| data\_sram\_wen | OUT | 4 | ram字节写使能信号，高电平有效 |
| data\_sram\_addr | OUT | 32 | ram读写地址，字节寻址 |
| data\_sram\_wdata | OUT | 32 | ram写数据 |
| es\_dest\_withvalid | OUT | 39 | 从ES模块返回的目的寄存器号和阻塞信号和数据和选择信号 |

表二 EX模块接口定义

1. 功能描述

因为存在写后读相关的取值问题，在lab4中使用阻塞方法解决相关问题。引入EXE模块目的寄存器号传递信号：es\_dest\_withvalid，作用为传递EXE阶段正在运行中的寄存器号，随后与ID阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在lab5中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将es\_dest\_withvalid的功能扩大化，把待写入数据的值也一并传递，这样可以避免大部分的阻塞。另外，如果处于执行级的PC为lw或者其他的访存指令的话，需要等到执行到访存模块时才能拿到正确的值，所以如果此时目的寄存器号与译码模块的源寄存器号相同的话，需要将ID阻塞一拍。

（四）重要模块3设计：MEM访存模块

1. 工作原理

从EX传入的信号主要被保存之后，等待下一拍将其传递到下一个WB写回阶段。另外，最关键的部分是上个阶段从EX提出的访存申请被这一拍上升沿应答之后，返回到该模块中。

1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟 |
| reset | IN | 1 | 复位信号，拉高为复位 |
| ws\_allowin | IN | 1 | 允许数据流水进入WB |
| ms\_allowin | OUT | 1 | 允许数据流水进入MEM |
| es\_to\_ms\_valid | IN | 1 | 从EX向MEM的数据传输是否可以进行 |
| es\_to\_ms\_bus | IN | 64 | EX向MEM的总线 |
| ms\_to\_ws\_valid | OUT | 1 | 从MEM向WB的数据传输是否可以进行 |
| ms\_to\_ws\_bus | OUT | 70 | MEM向WB的总线 |
| data\_sram\_rdata | IN | 32 | ram读数据 |
| ms\_dest\_withvalid | OUT | 39 | 从MS模块返回的目的寄存器号和阻塞信号和数据和选择信号 |

表三 MEM模块接口定义

1. 功能描述

因为存在写后读相关的取值问题，在lab4中使用阻塞方法解决相关问题。引入MEM模块目的寄存器号传递信号：ms\_dest\_withvalid，作用为传递MEM阶段正在运行中的寄存器号，随后与ID阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在lab5中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将ms\_dest\_withvalid的功能扩大化，把待写入数据的值也一并传递，这样可以避免目前为止所有在访存级引起的写后读数据相关的阻塞。

（五）重要模块4设计：WB写回模块

1. 工作原理

从MEM传来的写回寄存器堆的数据在从ID一路传递过来的指令控制下，被写入到regfile中。

1. 接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟 |
| reset | IN | 1 | 复位信号，拉高为复位 |
| ws\_allowin | OUT | 1 | 允许数据流水进入WB |
| ms\_to\_ws\_valid | IN | 1 | 从MEM向WB的数据传输是否可以进行 |
| ms\_to\_ws\_bus | IN | 70 | MEM向WB的总线 |
| ws\_to\_rf\_bus | OUT | 38 | WB向ID中regfile的总线 |
| debug\_wb\_pc | OUT | 32 | 写回级（多周期最后一级）的 PC，需要 mycpu 里将 PC 一路带到 写回级 |
| debug\_wb\_rf\_wen | OUT | 4 | 写回级写寄存器堆(regfiles)的写使能，为字节写使能，如果 mycpu 写 regfiles为单字节写使能，则将写使能扩展成 4位即可 |
| debug\_wb\_rf\_wnum | OUT | 5 | 写回级写 regfiles的目的寄存器号 |
| debug\_wb\_rf\_wdata | OUT | 32 | 写回级写 regfiles的写数据 |
| ws\_dest\_withvalid | OUT | 38 | 从WS模块返回的目的寄存器号和阻塞信号和数据 |

表四 WB模块接口定义

1. 功能描述

因为存在写后读相关的取值问题，在lab4中使用阻塞方法解决相关问题。引入WB模块目的寄存器号传递信号：ws\_dest\_withvalid，作用为传递WB阶段正在运行中的寄存器号，随后与ID阶段的源寄存器号作比较，决定是否将其阻塞。具体实现方式见实验实现部分。

在lab5中，为了减少不必要的阻塞消耗，我们选择直接把已经计算出的数值直接通过旁路（bypass）传递回译码级模块。于是我们选择将ws\_dest\_withvalid的功能扩大化，把待写入数据的值也一并传递，这样可以避免目前为止所有在写回级引起的写后读数据相关的阻塞。

三、实验过程（50%）

（一）实验流水账

2019/9/25 10:00 – 10:30 对照代码研究多周期阻塞加前递流水cpu的结构，并画出框图

2019/9/25 18:00 –20:00 书写实验报告

（二）错误记录

1、错误1：多余阻塞时长

（1）错误现象

在实现前递后，非LW指令应该不引起阻塞，但是波形中依然存在该问题，总的用时并没有比仅采用阻塞解决冲突的方式更少。

（2）分析定位过程

决定是否进行阻塞的信号是ds\_ready\_go，问题应该出现在ds\_ready\_go信号。

（3）错误原因

在lab4的书写中，我引入了wire型变量do\_not\_block，其作用为当不存在写后读数据相关时为1，出现写后读数据相关时为0，并且直接把do\_not\_block的值赋用给ds\_ready\_go，这样导致无论数据是否前递成功，译码级模块都被阻塞，于是没有起到优化的效果。

（4）修正效果

对ID、EXE、MEM和WB全体进行修正：

对WB：

更改输出信号 *output [37 :0] ws\_dest\_withvalid*  ，并给他赋值：

*wire block\_valid,block;*

*assign block = (ws\_dest == 5'd0)?0:ws\_gr\_we;*

*assign block\_valid = block&ws\_valid;*

*assign ws\_dest\_withvalid = {ws\_final\_result,block\_valid,ws\_dest};*

block信号表示处于WB模块的指令是否是需要写回非0号寄存器的指令。blokc\_valid信号则为将block信号与ws\_valid信号做与操作，从而消除了在指令还未到达WB模块时，block信号为X，无法给ID模块做判断的情况。ws\_dest\_withvalid信号则为38位宽信号，第0至4位放置目的寄存器号，第5位ws\_dest\_withvalid信号，第6至37位放置将alu的计算结果。

对MEM：

更改输出信号 *output [38 :0] ms\_dest\_withvalid*  ，并给他赋值：

*wire block\_valid,block;*

*assign block = (ms\_dest == 5'd0)?0:ms\_gr\_we;*

*assign block\_valid = block&ms\_valid;*

*assign ms\_dest\_withvalid = {ms\_res\_from\_mem,ms\_final\_result,block\_valid,ms\_dest};*

block信号表示处于MEM模块的指令是否是需要写回非0号寄存器的指令。blokc\_valid信号则为将block信号与ms\_valid信号做与操作，从而消除了在指令还未到达MEM模块时，block信号为X，无法给ID模块做判断的情况。ms\_dest\_withvalid信号则为39位宽信号，第0至4位放置目的寄存器号，第5位ws\_dest\_withvalid信号，第6至37位放置将alu的计算结果，最高位放置最终数据来源是alu还是内存的信号。（其实最高位信号并没有用上，可以删去）

对EXE：

更改输出信号 *output [38 :0] es\_dest\_withvalid*  ，并给他赋值：

*wire block\_valid,block;*

*assign block = (es\_dest == 5'd0)?0:es\_gr\_we;*

*assign block\_valid = block&es\_valid;*

*assign es\_dest\_withvalid = {es\_load\_op&es\_valid,es\_alu\_result ,block\_valid,es\_dest};*

block信号表示处于EXE模块的指令是否是需要写回非0号寄存器的指令。blokc\_valid信号则为将block信号与es\_valid信号做与操作，从而消除了在指令还未到达EXE模块时，block信号为X，无法给ID模块做判断的情况。es\_dest\_withvalid信号则为39位宽信号，第0至4位放置目的寄存器号，第5位es\_dest\_withvalid信号，第6至37位放置将alu的计算结果，最高位放置最终数据来源是alu还是内存的信号。

对ID:

有很多信号可以删除，此处列出的仅为不用删除的部分和修改的部分：

*wire rs\_es, rs\_ms, rs\_ws, rt\_es, rt\_ms, rt\_ws;*

*assign rs\_es = (es\_dest\_withvalid[5] == 0)?1*  //没有写回操作或者目标寄存器为0，或者有写回操作但是目标寄存器不是ID源寄存器时为1

*:(rs == es\_dest\_withvalid[4:0])?0*

*:1;*

*assign rs\_ms = (ms\_dest\_withvalid[5] == 0)?1*

*:(rs == ms\_dest\_withvalid[4:0])?0*

*:1;*

*assign rs\_ws = (ws\_dest\_withvalid[5] == 0)?1*

*:(rs == ws\_dest\_withvalid[4:0])?0*

*:1;*

*assign rt\_es = (es\_dest\_withvalid[5] == 0)?1* //没有写回操作或者目标寄存器为0，或者有写回操作但是目标寄存器不是ID源寄存器时为1

*:(rt == es\_dest\_withvalid[4:0])?0*

*:1;*

*assign rt\_ms = (ms\_dest\_withvalid[5] == 0)?1*

*:(rt == ms\_dest\_withvalid[4:0])?0*

*:1;*

*assign rt\_ws = (ws\_dest\_withvalid[5] == 0)?1*

*:(rt == ws\_dest\_withvalid[4:0])?0*

*:1;*

*assign do\_not\_block = es\_dest\_withvalid[38]?0:1;* //如果exe阶段的是lw指令，那么就阻塞一拍，否则就不阻塞

*assign rs\_value = (~rs\_es)?es\_dest\_withvalid[37:6]*

*:(~rs\_ms)?ms\_dest\_withvalid[37:6]*

*:(~rs\_ws)?ws\_dest\_withvalid[37:6]*

*:rf\_rdata1;*

*assign rt\_value = (~rt\_es)?es\_dest\_withvalid[37:6]*

*:(~rt\_ms)?ms\_dest\_withvalid[37:6]*

*:(~rt\_ws)?ws\_dest\_withvalid[37:6]*

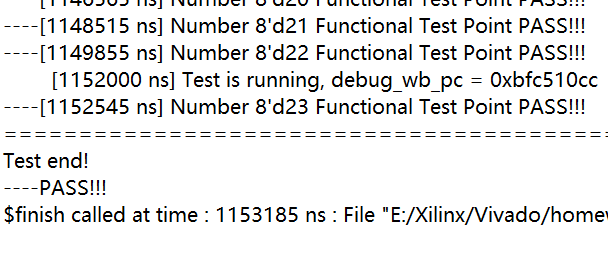
*:rf\_rdata2;*

最后，还需要对ds\_ready\_go信号做一点修改：

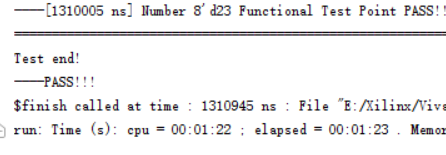
*assign ds\_ready\_go = do\_not\_block;*

（5）归纳总结（可选）

这次实验也很顺利，一次性通过。下附使用前递和不使用前递时间长短的对比图：

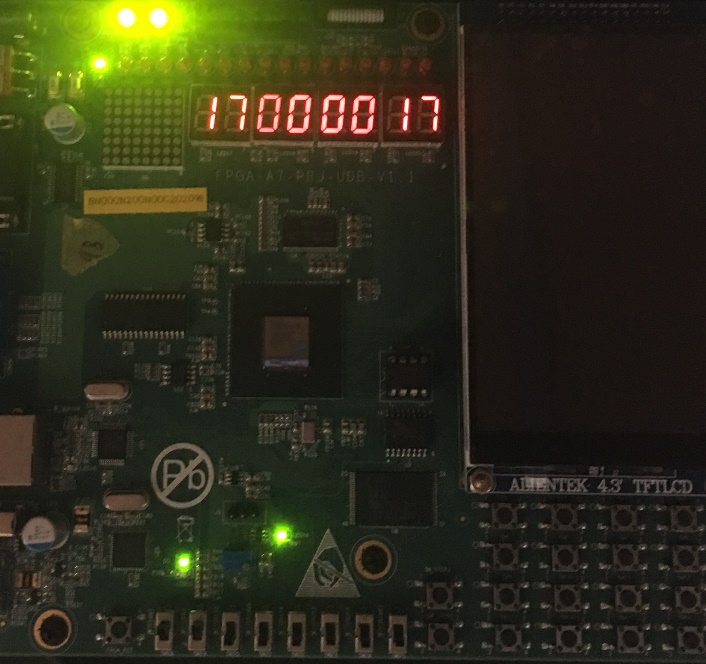


图一 使用前递方式的仿真运行时间



图二 不使用前递方式的仿真运行时间

四、实验总结（可选）



图三 上板成功拍照留念

终于完成了前递方式多周期五级流水cpu的设计！对上学期多周期cpu的设计进行了更系统的完善。