实验四报告

学号 2017K8009929044
姓名 李昊宸
箱子号 33

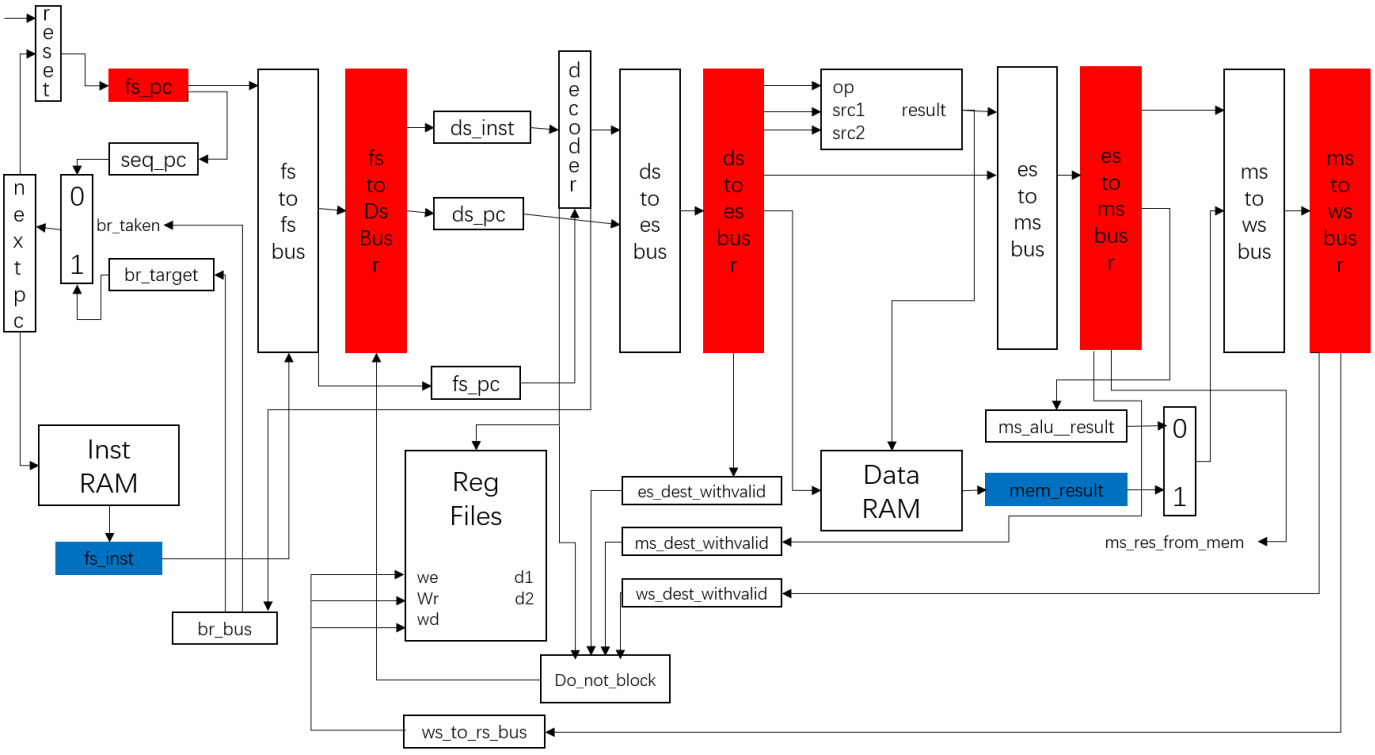
一、实验任务（10%）

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

mycpu 实验结构主要分为五个部分：IF 取值模块，ID 译码模块，EX 执行模块，MEM 访存模块和 WB 写回模块。

每一个模块流水线的控制核心均为一个触发器：IF 为 fs_pc，ID 为 fs_to_ds_bus_r，EX 为 ds_to_es_bus_r，MEM 为 es_to_ms_bus_r，WB 为 ms_to_ws_bus_r。四个触发器的值由流水条件中的 valid 与 allowin 控制，当握手信号握手成功时，数据被传输到下一个寄存器。

另外，在模块之间被传输的数据主要有这些：因为目前的 cpu 为五级流水 cpu，在不同模块中被执行的指令都

是不同的指令，所以一般来讲在整个 cpu 中有五条指令同时在执行，所以每一个模块向下一个模块传输的数据都至少包括当前指令的 PC 值，当前指令的具体信息。此外，每一个模块也需要向下一个模块传递控制信号，比如 IF 向 ID 传输指令的编码，ID 向 EX 传输将指令译码之后得到的指令名和 alu 的操作码、操作数等等，EX 向 MEM 传输 alu 的计算结果，以及最终数据的来源是 alu 的计算结果还是访存的取出数据，MEM 向 WB 传输最终将写回 Regfiles 中的寄存器编号和写回数据。

除此之外，在 ID 到 IF，WB 到 ID 之间还有由组合逻辑搭成的反向总线，分别用于传输下一条指令地址和向寄存器写回的地址和数值。有人可能会问为什么不直接把寄存器堆放在 WB 模块中方便写回，这是因为在 ID 阶段涉及到一些指令需要从寄存器堆中取出数据，如果将其放在 WB 模块中运用起来并没有那么直观。

这些与上周实验所实现的部分都一样，这一次需要考虑写后读相关的问题，于是引入“阻塞”的思想。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。

（二）重要模块 1 设计：IF 取指模块

1、工作原理

以 fs_pc 作为操作核心，记录每个时钟周期处于 IF 操作模块内部正在执行的 PC，并伴随着同步 RAM（数据 RAM）不断取出对应 PC 地址处存放的指令，发送给下个模块 ID。同时，PC 的跳转逻辑是由与 ID 模块相连接的数据通路 br_bus 决定。

2、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
ds_allowin	IN	1	允许数据流水进入 ID
br_bus	IN	33	从 ID 模块返回的下一步 PC 跳转信号
fs_to_ds_valid	OUT	1	从 IF 向 ID 的数据传输是否可以进行
fs_to_ds_bus	OUT	64	IF 向 ID 的总线
inst_sram_en	OUT	1	ram 使能信号，高电平有效
inst_sram_wen	OUT	4	ram 字节写使能信号，高电平有效
inst_sram_addr	OUT	32	ram 读写地址，字节寻址
inst_sram_wdata	OUT	32	ram 写数据
isnt_sram_rdata	IN	32	ram 读数据

表一 IF 模块接口定义

3、功能描述

此模块的作用主要为 PC 的转换和指令的取出。首先，fs_pc 信号由 reset 信号置于 0xhbfbffffc，此时 nextpc 被赋值为起始地址 0xbfc00000，发送取值信号到指令 RAM 中，这一个阶段被称为 pre-IF 阶段。因为同步 RAM 从发起读命令后，需要经历一个上升沿之后才能取出指令，所以在 fs_pc 等于 0xbfbffffc 时，发出取 0xbfc00000 地址的指令，直到下一个上升沿，PC 更新到 0xbfc00000 时，位于 0xbfc00000 的指令恰好被返回到 fs_inst。随后，nextpc 被组合逻辑立即赋值到新的 PC 值，并被发送到数据 RAM 中，下一个上升沿重复以上循环。另外，PC 的下一步更新与从 ID 模块返回的 br_bus 信号有关。

（三）重要模块 2 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。

2、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	6	从 ES 模块返回的目的寄存器号和阻塞信号
ms_dest_withvalid	IN	6	从 MS 模块返回的目的寄存器号和阻塞信号
ws_dest_withvalid	IN	6	从 WS 模块返回的目的寄存器号和阻塞信号

表二 ID 模块接口定义

3、功能描述

从 fs_pc 传来的 PC 值经过一拍上升沿，被更新到 ds_pc 中。fs_inst 也被传递到 ds_inst 中。指令的跳转信号 br_taken 和将要跳转到的 PC 地址 br_target 被通过 br_bus 信号传递回 IF 模块。其他和执行有关的信号（如 ALU 控制信号，从 regfile 中取出的数据等）被放置于 ds_to_es_bus 中，传递到 EX 模块。此外，regfile 被放置在此模块中，由 rf_we,rf_waddr 和 rf_wdata 信号控制写使能，写地址和写数据。

另外，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。

（四）重要模块 3 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。

2、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

名称	方向	位宽	功能描述
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	6	从 ES 模块返回的目的寄存器号和阻塞信号

表三 EX 模块接口定义

3、功能描述

从 fs_pc 传来的 PC 值经过一拍上升沿，被更新到 ds_pc 中。fs_inst 也被传递到 ds_inst 中。指令的跳转信号 br_taken 和将要跳转到的 PC 地址 br_target 被通过 br_bus 信号传递回 IF 模块。其他和执行有关的信号（如 ALU 控制信号，从 regfile 中取出的数据等）被放置于 ds_to_es_bus 中，传递到 EX 模块。此外，regfile 被放置在此模块中，由 rf_we,rf_waddr 和 rf_wdata 信号控制写使能，写地址和写数据。

另外，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。

（五）重要模块 4 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。

2、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	6	从 MS 模块返回的目的寄存器号和阻塞信号

表四 MEM 模块接口定义

3、功能描述

从 EX 阶段传来的 PC,指令和寄存器地址，被保存后通过 ms_to_ws_bus 传递到 WB 写回模块。在这个模块，

上个周期向数据 RAM 提交的请求将返回对应的内存数据到 data_sram_rdata，并且从上个模块传过来的 alu_result 在从 ID 传递的信号下，二者被选择出对应指令的那个后传递到 WB 模块。

另外，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。

（六）重要模块 5 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。

2、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	6	从 WS 模块返回的目的寄存器号和阻塞信号

表五 WB 模块接口定义

3、功能描述

从 MEM 传递来的写回寄存器的数据和写回的地址被传递到 ws_to_rf_bus 总线中，被返回到 ID 模块中执行写回操作。

另外，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。

三、实验过程（50%）

（一）实验流水账

2019/9/19　17:00 – 20:00 对照代码研究多周期阻塞流水 cpu 的结构，并画出框图

2019/9/20　13:00 –15:00　书写实验报告

（二）错误记录

1、错误 1：写后读相关

（1）错误现象

```
lui   a0, ref_base;
lui   t0, in_a;
NOP4;
addu  a0, a0, t1;
addu  t1, t1, t2;
NOP4;
bne   t0, a0, inst_error;
```

测试程序在执行到 bne 语句时，发生了不应该发生的跳转。

（2）分析定位过程

按照正常运行结果，执行到 bne 指令处，t0 应该与 a0 相等，从而 bne 不会发生跳转。但是对比波形图，发现此处 t0 与 a0 不相等。看汇编代码，在 bne 前面有 addu 指令，故问题可能是出现了写后读相关。

（3）错误原因

实际运行中，*addu a0, a0, t1;*这条指令导致了问题的发生。在 ID 模块执行到 *bne t0, a0, inst_error;*时，WB 模块还在执行 *addu a0, a0, t1;*，写回寄存器堆的值需要等到下一个时钟上升沿才会被读取出来。而 bne 的比对在上升沿之前就将被进行完毕，导致取出了还未更新的操作数，出现跳转错误。

（4）修正效果

对 ID、EXE、MEM 和 WB 全体进行修正：

对 WB：

加入新的输出信号 *output [5 :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 = {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 6 位宽信号，低 5 位放置目的寄存器号，最高位放置 ws_dest_withvalid 信号。

对 MEM：

加入新的输出信号 *output [5 :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 = {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 6 位宽信号，低 5 位放置目的寄存器号，最高位放置 ms_dest_withvalid 信号。

对 EXE：

加入新的输出信号 *output [5 :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 = {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 6 位宽信号，低 5 位放置目的寄存器号，最高位放置 es_dest_withvalid 信号。

对 ID:

```
wire        do_not_block;
wire        source_is_rs, source_is_rt;
assign      source_is_rs = ~inst_sll&~inst_srl&~inst_sra&~inst_lui&~inst_jal;
assign      source_is_rt = ~inst_addiu&~inst_lui&~inst_lw&~inst_jal&~inst_jr;  //错误 把&打成^
wire        rs_is_not_zero,rt_is_not_zero;
assign      rs_is_not_zero = (rs == 5'd0)?0:1;
assign      rt_is_not_zero = (rt == 5'd0)?0:1;
wire        rs_check,rt_check;
assign      rs_check = (~source_is_rs)||(~rs_is_not_zero);          //rs 无效或者为 0 时为 1，有效时为 0
assign      rt_check = (~source_is_rt)||(~rt_is_not_zero);          //rt 无效或者为 0 时为 1，有效时为 0

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;
wire        rs_rt_compare;
assign      rs_rt_compare = rs_es&rs_ms&rs_ws&rt_es&rt_ms&rt_ws;     //若为 1 说明没有相等的操作号

assign   do_not_block = rs_rt_compare || ((rs_check||(rs_es&rs_ms&rs_ws))&(rt_check||(rt_es&rt_ms&rt_ws))) ;
```

//没有相等的操作号　或者　rs 无效或 rs 与后面不相等且 rt 无效或 rt 与后面不相等

上面信号的赋用的具体解释见注释。最后，还需要对 ds_ready_go 信号做一点修改：

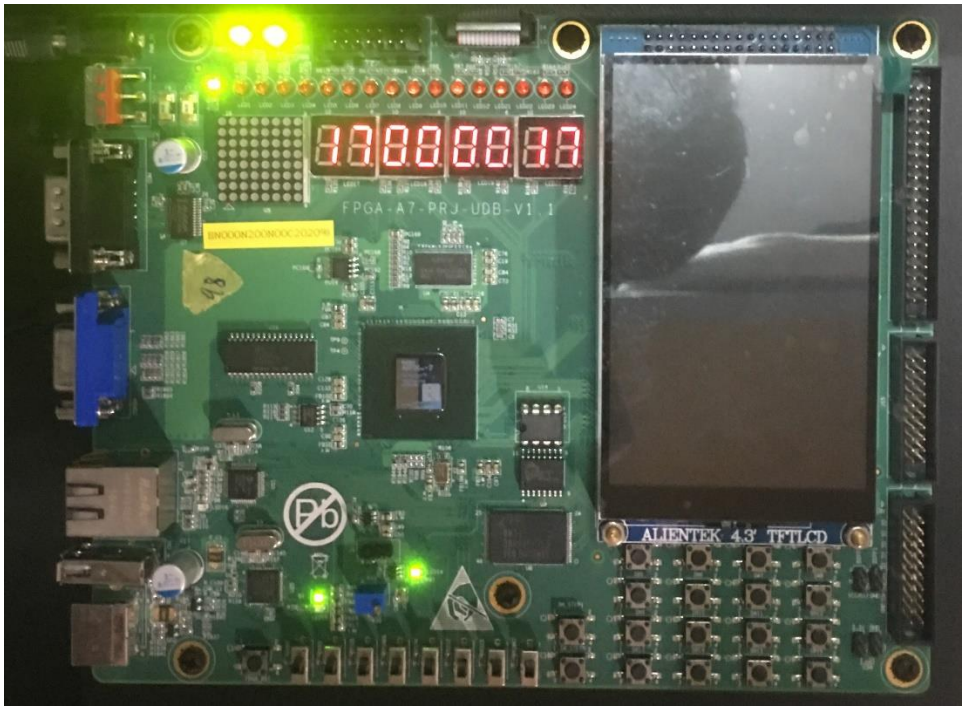
assign ds_ready_go = do_not_block;

（5）归纳总结（可选）

虽然这样未经修改一遍就成功了，但我认为这里面有一个很大的坑：SW 指令。

这一条指令看起来只有一个源操作数，但其实有两个，rs 和 rt 都是需要判断的。另外，尤其要注意笔误的情况。我在写判断条件的时候把&写成了^,还好一眼看到了，不然不知道调试得花多久……

四、实验总结（可选）



图一　上板成功拍照留念

这次进行的出奇的顺利，提前一天构思修改思路，代码一遍过。