流水线CPU设计与实现

一. 实验目的

1. 了解流水线 CPU 基本功能部件的设计与实现方法，
2. 了解提高 CPU 性能的方法。
3. 掌握流水线 MIPS 微处理器的工作原理。
4. 理解数据冒险、控制冒险的概念以及流水线冲突的解决方法。
5. 掌握流水线 MIPS 微处理器的测试方法。

二. 实验内容

1.利用 HDL 语言，基于 Xilinx FPGA basys3 实验平台，用 Verilog HDL 语言或 VHDL 语言来编写，实现单周期CPU 的设计，这个流水线 CPU 能够完成MIPS 指令，至少包含以下指令：add、sub、and、or、addi、andi、ori、lw、sw、beq、bne。

2. **通过设计旁路解决非load-use数据冒险，通过设计硬件阻塞 (stall)解决load-use数据冒险和sw指令引发的数据冒险，通过分支预测（假设分支不发生）解决控制冒险问题。**

3. 掌握各个指令的相关功能并输出仿真结果进行验证，并最后在 FGPA 上实现，要求实现运行和单步调试两种状态，将其中的 运算结果在开发板数码管上显示出来。

三. 实验原理

流水线是数字系统中一种提高系统稳定性和工作速度的方法，广泛应用于现代CPU的架构中。根据MIPS处理器的特点，将整体的处理过程分为取指令（IF）、指令译码（ID）、执行（EX）、存储器访问（MEM）和寄存器会写（WB）五级，对应多周期的五个处理阶段。一个指令的执行需要5个时钟周期，每个时钟周期的上升沿来临时，此指令所代表的一系列数据和控制信息将转移到下一级处理。

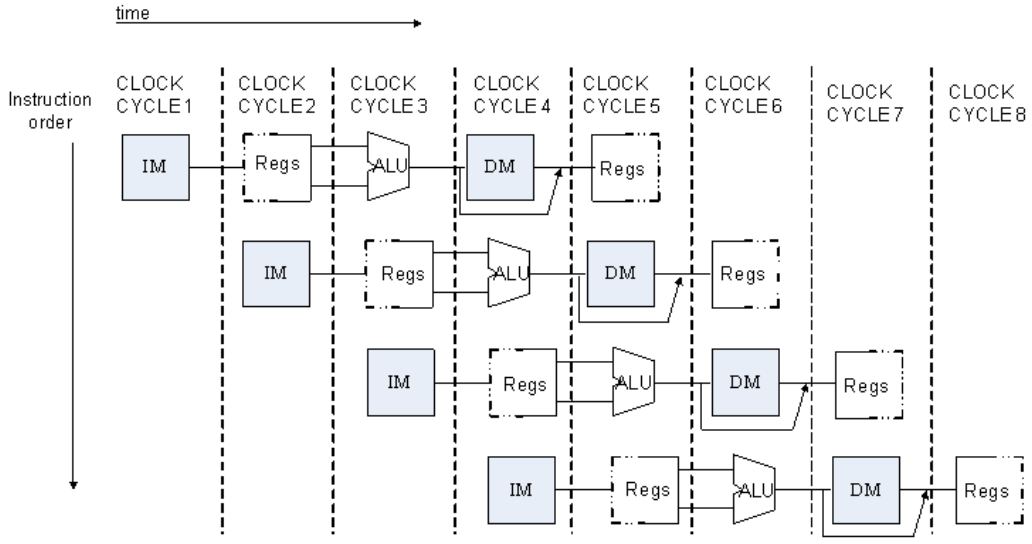
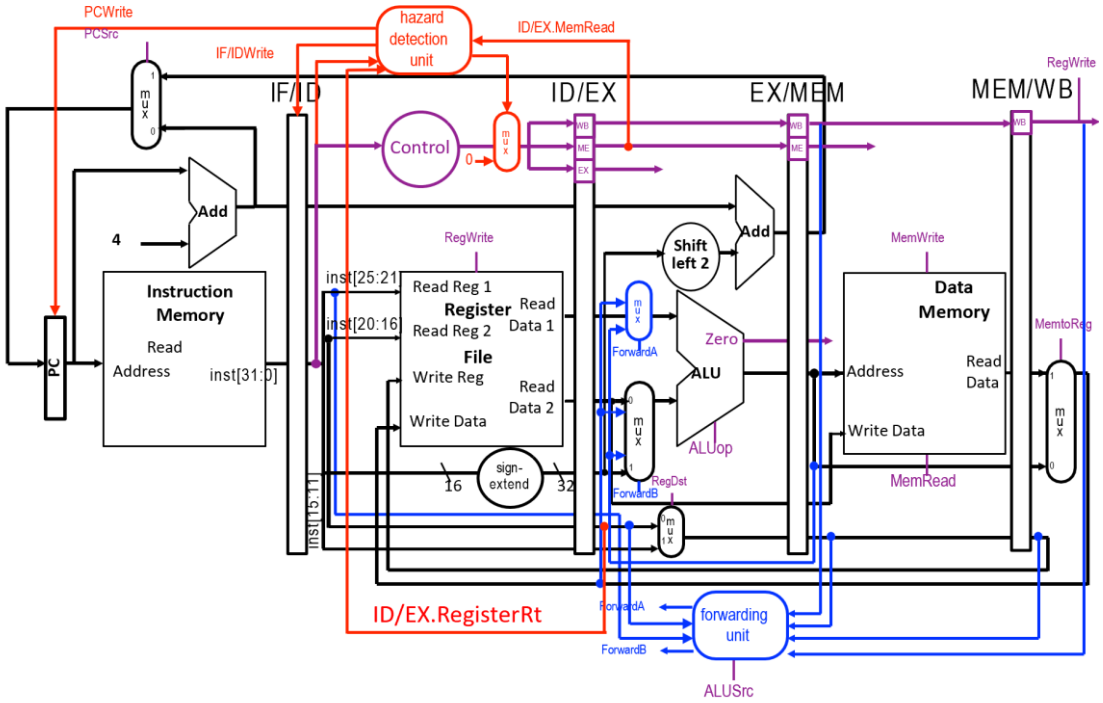


表 1 本实验所涉及的 11 条 MIPS 指令

R 型指令							
指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	功能
Add	000000	rs	rt	rd	000000	100000	寄存器加
Sub	000000	rs	rt	rd	000000	100010	寄存器减
And	000000	rs	rt	rd	000000	100100	寄存器与
Or	000000	rs	rt	rd	000000	100101	寄存器或
Xor	000000	rs	rt	rd	000000	100110	寄存器异或
I 型指令							
Addi	001000	rs	rt	immediate			立即数加
Andi	001100	rs	rt	immediate			立即数与
Ori	001101	rs	rt	immediate			立即数或
Lw	100011	rs	rt	offset			取数据
Sw	101011	rs	rt	offset			存数据
Beq	000100	rs	rt	offset			相等转移
Bne	000101	rs	rt	offset			不等转移

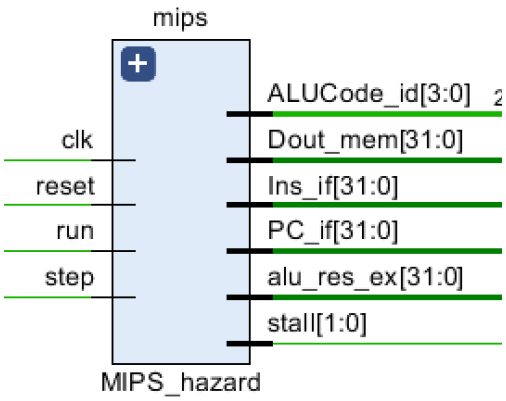
为了实现流水线CPU，需要在单周期CPU的基础上加入流水线寄存器保存每个周期的结果。为了解决数据冒险，我们需要增加旁路以减少阻塞，因此需要设计Forward Unit。为了解决load-use数据冒险，我们需要增加阻塞。在本项目中我们实现了使用硬件阻塞，因此需要设计Hazard Unit.由于不同指令的下一个PC值得的产生方式不同，产生的时间也不同，因此单周期CPU的next_PC模块不能再复用，必须重写PC更新逻辑。



由vivado生成的数据通路图，请见schematic.pdf

四. 实验过程与结果

1、 CPU设计的思想、方法。



流水线CPU由IF，IF_ID_Reg，Hazard_unit，ID，ID_EX_Reg，EX，EX_MEM_Reg，MEM，MEM_WB_Reg，Forward_Unit组成。该MIPS_hazrd可分功能模块设计，各个子模块分别设计其特定的功能，最终利用一个总的模块(top)进行子模块间连接，使得整个CPU能连贯执行指令，在仿真结果中观察设计结果，最终进行硬件下载，验证设计。其中各个模块功能如下：

（1） IF模块：包括Ins_ROM和PC寄存器。Ins_Rom会进行初始化，初始化ip核的

机器码见coe文件，用于测试的汇编指令见下方。PC寄存器会根据输入信号（跳转信号/PC+4/阻塞）判断下一个PC的取值。我们使用IP核构造指令存储器，并将project10.coe导入。

（2） IF_ID_Reg模块：是IF和ID之间的流水线寄存器。这个寄存器需要保留IF阶段输出的指令信号和PC+4信号。

（3） ID模块：包括寄存器堆，数字拓展模块和控制单元。值得注意的是，为了避免结构冒险，寄存器堆需要在下降沿写入；而读取可以全程读，只需要保证周期末尾读到的数是正确的即可。为了观察测试指令结果，寄存器堆初始化的值为寄存器编号，即rom[0]=32’h0.

（4） ID_EX_Reg模块：是ID和EX之间的流水线寄存器。这个寄存器需要保留ID阶段产生的控制信号，立即数，两个操作数，Rt/Rd的地址以产生正确的寄存器写入地址，Rs/Rt/Rd的地址以产生正确的旁路信号。

（5） EX模块：包含ALUSrcB的选择器，ALU，branch指令的地址计算以及寄存器写地址的选择器。注意这里需要引入旁路信号并选择正确的ALU操作数。同时该模块需要产生是否需要跳转的信号(Branch控制信号和ALU运算结果ZF相与)。

（6） EX_MEM_Reg模块：是EX和MEM之间的流水线寄存器，需要保存EX模块的ALU运算结果，branch跳转指令，以及来自ID_EX_Reg的关于MEM,WB的控制信号。同时需要保存Rs/Rt/Rd的地址以产生正确的旁路信号。

（7） MEM模块：包含数据存储器。为了观察测试指令结果，数据存储器初始化的值为ram[i]=i+32’h100.

（8） MEM_WB_Reg模块：是MEM和WB之间的流水线寄存器，需要保存ALU运算结果，存储器读数据，以及寄存器写数据信号，寄存器写地址信号。

（9） Forward_Unit模块：用于产生转发信号。

（10） Hazard_unit模块：用于产生stall信号。

详细的设计代码请见附件/source/mips_piepline。

2、 如何构造硬件阻塞stall：

a) 若要阻塞一个或以上周期，阻止IF获得新的指令，具体的方法是当stall等于1时保持PC值不变。

b) 若要阻塞两个或以上周期，具体的方法是当stall为1时，使得IF_ID_Reg模块的指令和PC+4信号不变（与上一个时钟周期相同）。

3、 如何阻止模块运行。

(1) 阻止ID运行。具体的方法是是当flush为1时，使得IF_ID_Reg模块的指令和PC+4信号清零。

(2) 阻止EX运行。具体的方法是当flush为1时，使得ID_EX_Reg模块的PC+4信号不变（与上一个时钟周期相同），让数据信号（操作数/立即数）和控制信号信号都清零。

(3) 阻止MEM运行。具体的方法是当flush为1时，使得EX_MEM_Reg模块的PC+4信号不变（与上一个时钟周期相同），让数据信号（操作数/立即数）和控制信号信号都清零。

要注意阻塞stall和清零flush的区别。阻塞stall表示进入流水线的指令是正确的，但译码的时机不对（比如load-use）；清零flush表示进入流水线的指令是错误的（比如beq）

4、 如何解决处理各类冒险

a) 数据冒险：可以通过增加stall来解决；也可以设计旁路，以减少阻塞，本工程使用后者方法。

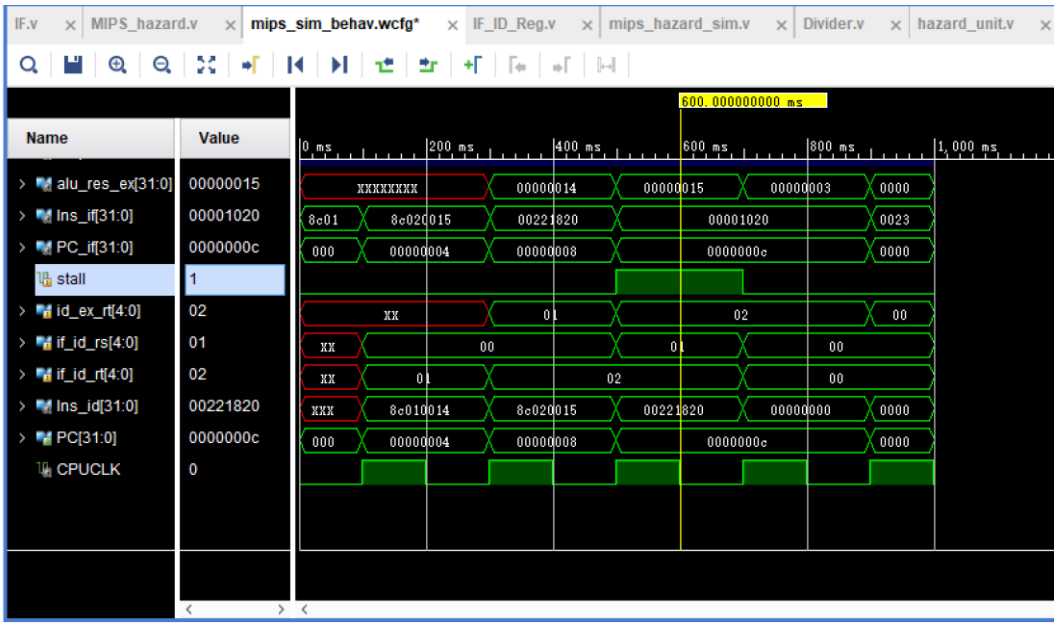
旁路信号由Forward_Unit生成。如果数据依赖的两条指令相邻，那么ALU操作数的源来自EX_MEM; 如果数据依赖的两条指令相隔一条指令，那么ALU操作数的源来自MEM_WB

多选器控制	源	解释
ForwardA = 00	ID_EX	第一个 ALU 操作数来自寄存器
ForwardA = 01	EX_MEM	第一个 ALU 操作数由上一个 ALU 运算结果旁路获得
ForwardA = 10	MEM_WB	第一个 ALU 操作数从数据存储器或前面的 ALU 结果中旁路获得
ForwardB = 00	ID_EX	第二个 ALU 操作数来自寄存器
ForwardB = 01	EM_MEM	第二个 ALU 操作数由上一个 ALU 运算结果旁路获得
ForwardB = 10	MEM_WB	第二个 ALU 操作数由数据存储器或前面的 ALU 结果旁路获得

至于load-use冒险和sw指令冒险。可以阻塞两个周期等待数据写入寄存器，或者在使用旁路的条件下阻塞一个周期，本工程使用前者方法。同时需要屏蔽EX的运行。

对于load-use冒险如何生成stall信号？当寄存器写使能且lw的rt寄存器与运算

指令的rs/rt寄存器相同时需要产生stall.值得注意的是，因为我们需要在第四个周期（下图黄线位置）等lw的rt寄存器和运算指令的rs/rt寄存器来到hazard_unit才能做判断，因此产生stall的时序逻辑应该是下降沿触发。



b) 结构冒险：寄存器堆和数据存储器遵循下降沿写（高电平的时间给予充分的setup时间）。同时对于读数据使用组合逻辑。

c) 控制冒险：一种方法是阻塞3个周期等待branch_confirm生成。另一种方法是预测分支不发生，这也是本工程使用的方法。我们假设分支不发生，并将beq后的两条指令导入流水线。若分支发生，将branch_addr信号和branch_confirm信号在EX阶段生成并将branch_confirm传到hazard unit模块，生成flush信号屏蔽下一个周期ID和EX的运行。同时IF模块会获取新的跳转指令。

5、 验证的 CPU 正确性

首先需要设计仿真文件定义时钟周期。接下来进行仿真，并将仿真的结果（PC、ALU运算结果、数据存储器读数据、指令）与汇编代码进行对比。

下面的测试指令的冒险如下：（注意寄存器和数据存储器已经赋了初值，寄存器regfile[i]=i; 数据存储器ram[i]=i+0x100）

(1) 第1，2，3条指令发生load-use数据冒险。通过阻塞两个周期并阻止EX运行可以解决。

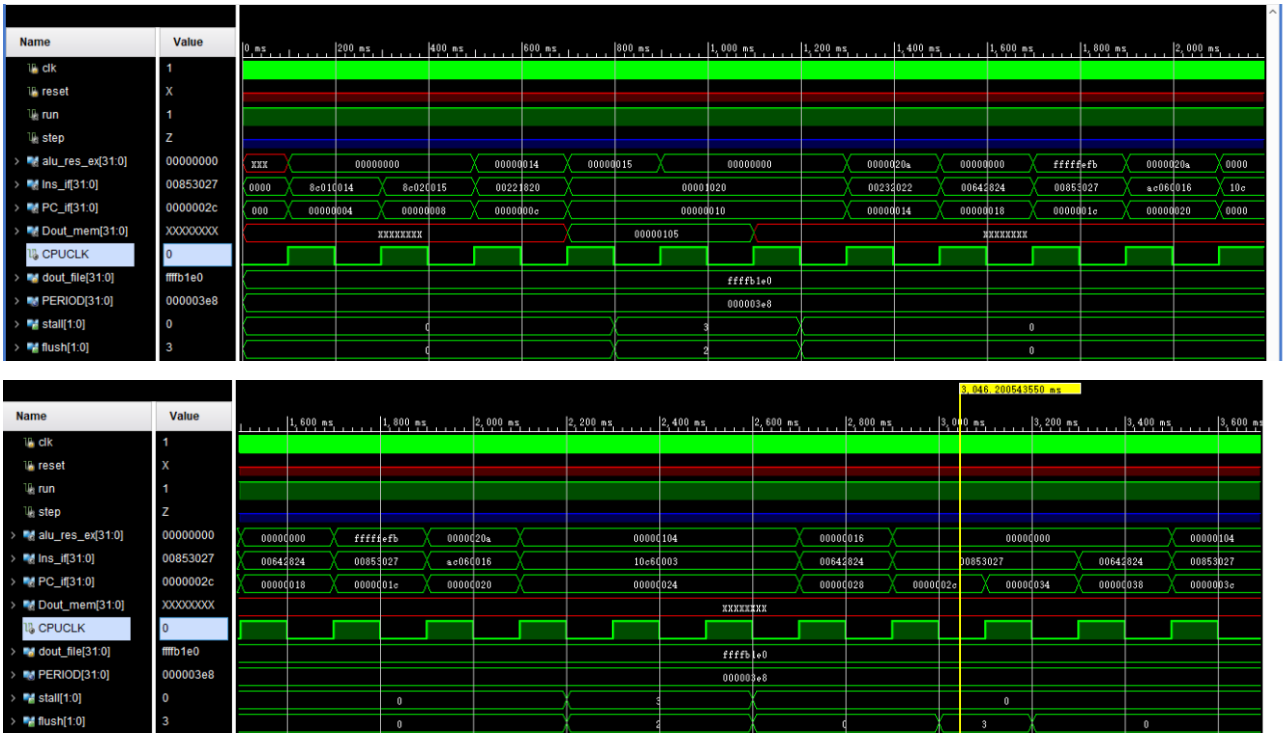
(2) 以下指令组发生数据依赖（数据冒险）：第3/5条指令（$3），第5/6条指令（$4），第5/6/7条指令（$4$5）。通过旁路可以不阻塞地解决该冒险问题。

(3) 第7，8条指令发生sw的数据冒险，通过阻塞两个周期阻止EX运行可以获得正确答案.

(4) 第9，10，11条指令发生控制冒险，通过阻止ID和EX运行（生成flush信号重置流水线的信号）可以获得正确答案。

PC	Instruction	汇编指令	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00000004	8c010014	lw $1,20($0) ($1=00000105)	IF	ID	EXE	MEM	WB															
00000008	8c020015	lw $2,21($0) 注意这个21是字节位移，已经乘了4		IF	ID	EXE	MEM	WB														
		bubble			IF	ID(生成stall信号)	flush															
		bubble					flush															
0000000c	221820	add $3,$1,$2 ($3=0000020a)					IF	ID	EXE	MEM	WB											
00000010	1020	add $2,$0,$0						IF	ID	EXE	MEM	WB										
00000014	232022	sub $4,$1,$3 ($4= FFFF FEFB)							IF	ID	EXE	MEM	WB									
00000018	642824	and $5,$3,$4 ($5=20A)								IF	ID	EXE	MEM	WB								
0000001c	853027	nor $6,$4,$5 ($6=20A nor FFFF FEFB=0000 0104)									IF	ID	EXE	MEM	WB							
		bubble										IF	ID(生成stall信号)	flush								
		bubble												flush								
00000020	ac060016	sw $6,22($0)												IF	ID	EXE	MEM	WB				
00000024	10C60003	beq $6,$6,12 000100 00110 00110 0000000000000011													IF	ID	EXE产生branch信号	MEM	WB			
00000028	642824	and $5,$3,$4														IF	ID	flush				
0000002c	853027	nor $6,$4,$5															IF	flush				
00000030	642824	and $5,$3,$4																				
00000034	853027	nor $6,$4,$5																				
00000038	642824	and $5,$3,$4																IF	ID	EXE	MEM	WB
0000003c	853027	nor $6,$4,$5																	IF	ID	EXE	MEM
运算结果		alu_res_ex(h)	00000000	00000000	00000014	00000015	00000000	00000000	0000020a	00000000	ffffffefb	0000020a	00000104	00000104	00000104	00000016	00000000	00000000	00000000	00000104	0000020a	0000020a
		Ins_if(h)	8c010014	8c020015	00221820	00001020	00001020	00001020	00232022	00642824	00853027	ac060016	10c60003	10c60003	10c60003	00642824	00853027	00853027	00642824	00853027	00000000	00000000
		PC_if(h)	00000004	00000008	0000000c	00000010	00000010	00000010	00000014	00000018	0000001c	00000020	00000024	00000024	00000024	00000028	0000002c	00000034	00000038	0000003c	00000040	00000040
		Dout_mem(h)	xxxxxxxx	xxxxxxxx	xxxxxxxx	00000105	00000105	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx
各个模块在不同时间的运行情况，目的是展示解决load-use冒险和控制冒险		IF	lw4	lw8	add 0c	add 10	add 10	add 10	sub14							and28	nor2c	跳转PC为34 nor34	and38			
		ID		lw4	lw8	add 0c 检测到数据冒险，生成stall信号和flush信号	add 0c	add 0c	add 10							beq	and28	清空	nor34				
		EXE			lw4	lw8	清空		add 0c								beq-产生branch_confirm信号-flush信号	清空					
		MEM				lw4	lw8											beq				
		WB					lw4												beq			

data.txt 文件是由 vivado 生成的仿真结果文件，内含 alu_res_ex，Ins_if，PC_if Dout_mem;仿真结果也可以看下方波形图截图。



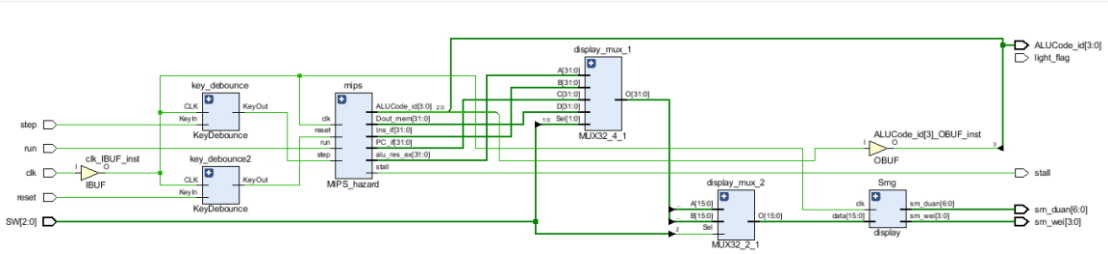
仿真文件如下：附件/source/mips_sim.v；附件/source/top_sim.v

仿真结果见 附件/source/data.txt：

6、 在Basys3板上运行所设计的CPU

我们需要设计display文件以在数码管上显示；需要KeyDebounce模块消除单步调试时的抖动。七段译码显示的内容是16位的，而ALU的运算结果是32位的，将运算结果分为高十六位和低十六位，分别传进七段译码模块；在顶层模块可用一个开关，来选择是显示高16位还是低16位；同时我们还要选择数码管显示对象（PC、指令、ALURes、memreaddata）我们增加消抖模块。按键消抖的原理是：在按下实验板上的键时，会在极短时间内产生很多次不稳定的电平波动，从而造成不可预料的后果，我们可以统计电平脉冲的数量，认为存在3次就是一次按键，这样基本上就可以避免上述情况的发生。

最后我们用top文件将display、MIPS、KeyDebounce三个模块连接起来。



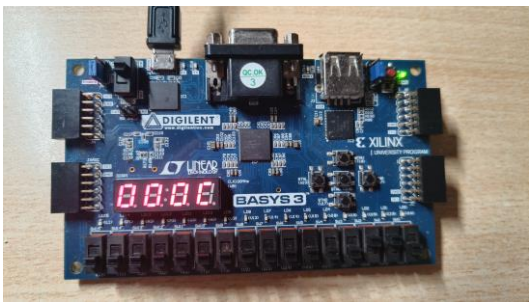
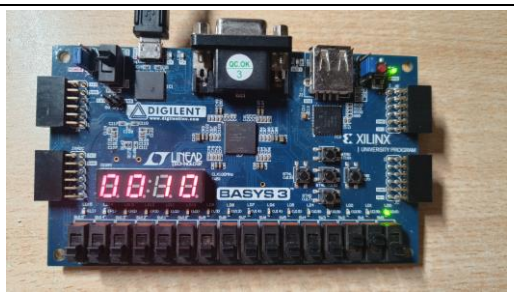
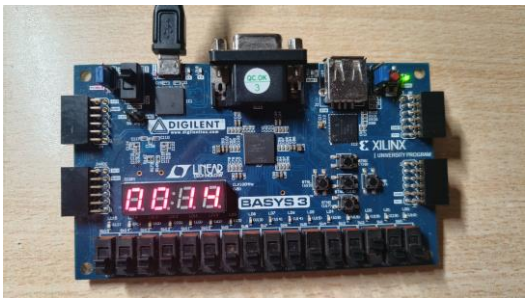
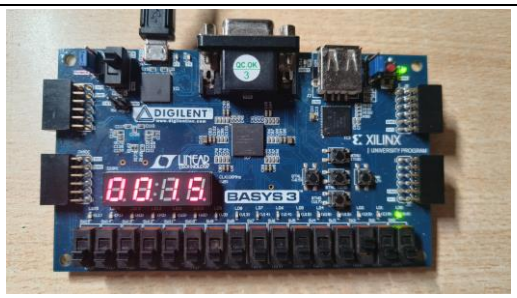
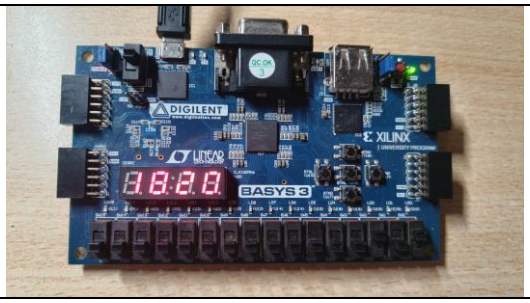
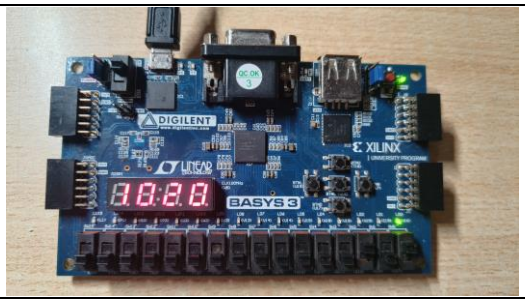
我们需要约束文件以确定引脚位置。最后生成比特流，将比特流写入开发板。

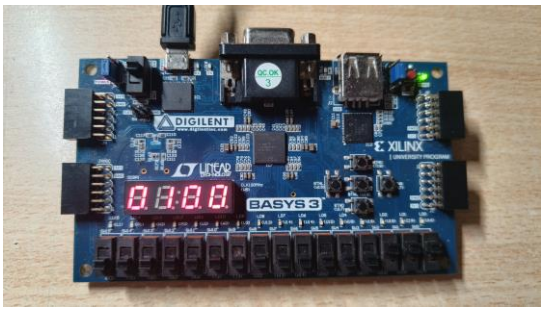
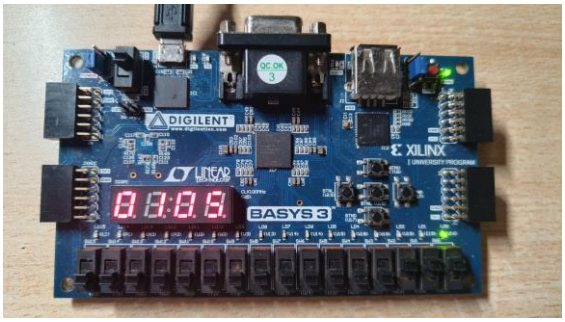
各个模块的代码，约束文件位于附件：

top_hazard.v, KeyDebounce.v, display.v

constrain.xdc

在板上运行的结果，拍照，贴图，来说明。由于空间有限，只展示第3，4个周期的的PC_if、ALU运算结果、指令_if.

	周期3	周期4
PC		
ALU		
Instruction 低16位		

Memreaddata		

五. 实验心得

1、 在实验中遇到多个报错问题并成功解决。以下是报错总结。

a) 仿真的时候所有结果都是正确的，但在写板时出现alures无法正常显示的问题（PC和指令都能正常显示），而且我设计了stall，PC=101时本来应该显示的时间长一些，但是没有。这是因为我直接复制了实验PPT上的流水线寄存器代码，但PPT代码的if(~reset) ，导致出错。修改为if(reset)即可。

b) 区别阻塞赋值和非阻塞赋值：阻塞赋值是按序执行，计算等号右边的值并同时赋给左边变量，当前语句完成才能执行下一句。非阻塞赋值是并行执行，各语句同时执行。

c) initial语句可以在仿真中为寄存器赋初值，但是在综合时，initial语句是不可综合的语法，因为它不能被转换为硬件电路。在硬件电路中，寄存器的初始值是不确定的，它们只能通过时钟信号或复位信号来改变。所以，如果你想在综合时为寄存器赋初值，你需要使用一个复位信号，并在always语句中根据复位信号的状态来给寄存器赋值。

d) 生成BitStream时遇到Vivado Error问题[DRC LUTLP-1] Combinatorial Loop Alert: 2 LUT cells form a combinatorial loop. /仿真时出现错误Error: (vsim-3601) Iteration limit reached at time 55445 ns.

在代码里面出现了回环，通常是组合电路的问题。比方说在一个组合逻辑块里面，对敏感变量进行赋值。作为敏感变量，只要变化，就会触发组合逻辑块的赋值，而赋值又会立马让敏感变量变化，然后再触发组合逻辑块赋值。这样循环往复，每次触发变化的时间，是几乎可以忽略不计的。一旦敏感变量触发组合电路的赋值，便会不断地，触发--赋值--触发---赋值，变成死循环。如何解决？修改敏感信号列表“sensitivity list”

e) vivado：mixed level sensitive and edge triggered event controls are not supported for synthesis。因为一个触发器不能同时是边缘触发和电平触发的，因此不可被综合

2、 实验可以继续改进的地方

a) 可以继续完善控制冒险，增加分支预测部分等模块。

b) 可以增加针对load-use的数据旁路，减少一个时钟周期。

c) 增加其他指令譬如j指令以完善实验