

同济大学
计算机科学与技术系

计算机组成原理课程实验报告



学 号 15xxxxx

姓 名 MHD

专 业 计算机科学与技术

授课老师 ZDD

日 期 2018.6.19

目录

一、实验目标.....	1
二、总体设计.....	1
1. 作品功能设计及原理说明	1
2. 硬件逻辑图	3
2.1. CPU 总体数据通路.....	3
2.2. 应用程序硬件设计	4
2.3. 控制器模块.....	5
2.4. ALU 模块	5
2.5. PC 寄存器模块	5
2.6. CP0 模块.....	6
2.7. 乘法除法控制器模块.....	6
2.8. 寄存器堆模块	6
2.9. 乘法器	7
2.10. 除法器	7
三、主要模块设计	8
1. CPU 内部模块	8
2. IO 接口模块.....	8
3. VGA 模块.....	9
四、应用程序.....	9
1. 应用程序流程图.....	9
1.1. 游戏总流程图	9
1.2. 游戏子流程图	11
2. 程序模块.....	13
五、测试/调试过程	19
1. 硬件模块调试	19
2. 汇编程序调试	19
3. 综合调试.....	20
六、实验结果分析	20
1. CPU 前仿真/在线测试	20
2. CPU 下板验证	21
3. MIPS 应用程序运行	22
七、结论.....	27
八、心得体会及建议.....	27
1. 心得体会.....	27
2. 建议.....	28
九、附录.....	28
1. 项目顶层 verilog 代码.....	28
2. CPU 封装顶层 verilog 代码	30

一、实验目标

- 深入了解 CPU 的原理。
- 画出实现 54 条指令的 CPU 的通路图。
- 学习使用 Verilog HDL 语言设计实现 54 条指令的 CPU。
- 学习使用 Mips 汇编指令编写应用程序并在 Mips Cpu 中运行。
- 掌握 CPU 与 IO 通信的方法并在应用程序中体现。

二、总体设计

1. 作品功能设计及原理说明

本项目实现的是单周期 54 条指令的 CPU。它能够执行编译成机器码的 MIPS32 中的 54 条指令。要完成设计实现，首先要充分了解 CPU 的构成和工作过程。

本项目的 CPU 由时钟、控制器、运算部件 ALU、乘除法控制器、两种乘法器、两种除法器、PC 寄存器、寄存器堆、协处理器 CP0 构成，CPU 外部还连接着指令存储器和数据存储器。

让 CPU 各部件协调工作，就是要正确地给出控制信号。首先要构建好数据通路，然后根据各指令的需求，设计控制信号。设计单周期数据通路的一般方法为：

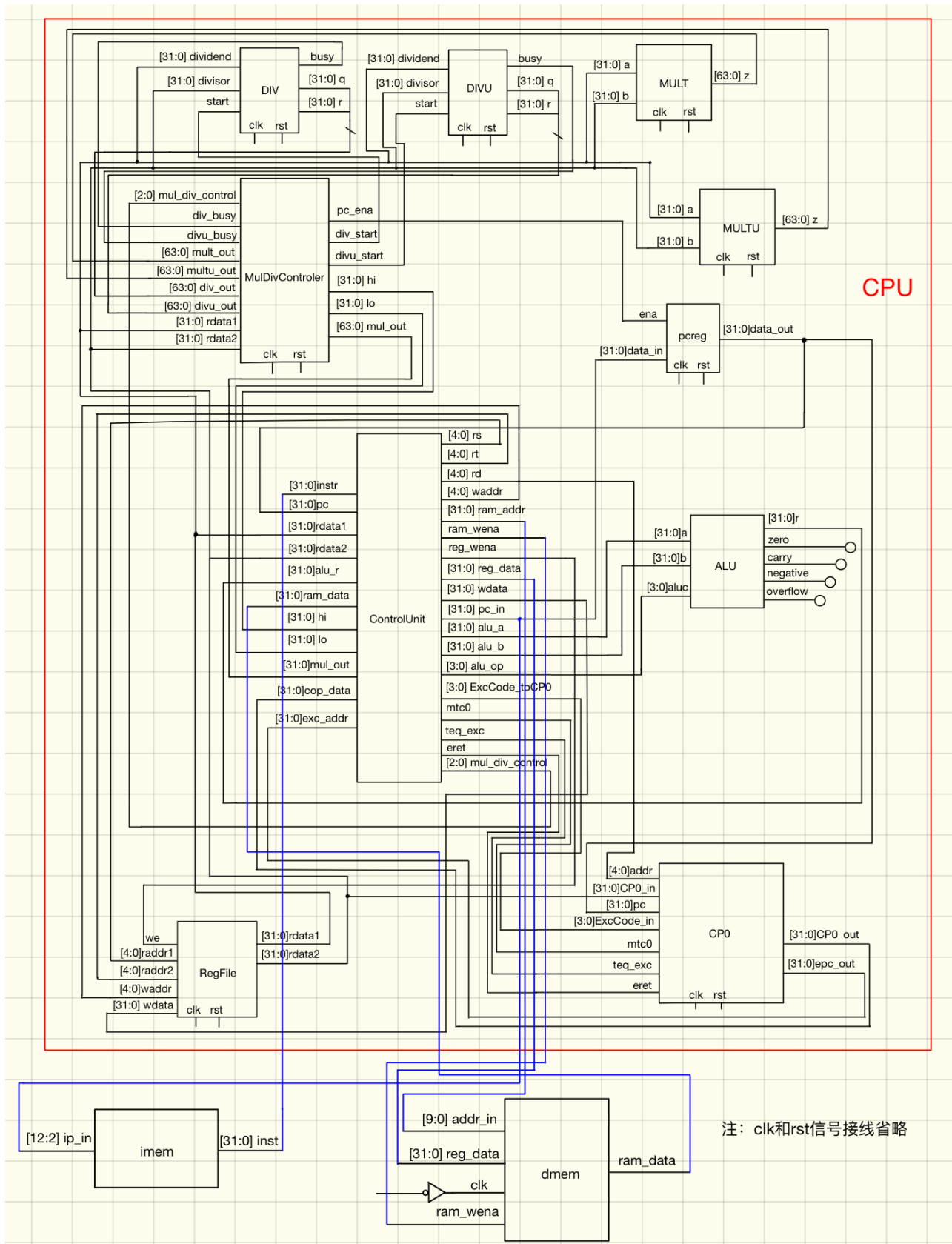
- 阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解
- 确定每条指令在执行过程中所用到的部件
- 使用表格列出指令所用部件，并在表格中填入每个部件的数据输入来源
- 根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

【数据来源表】

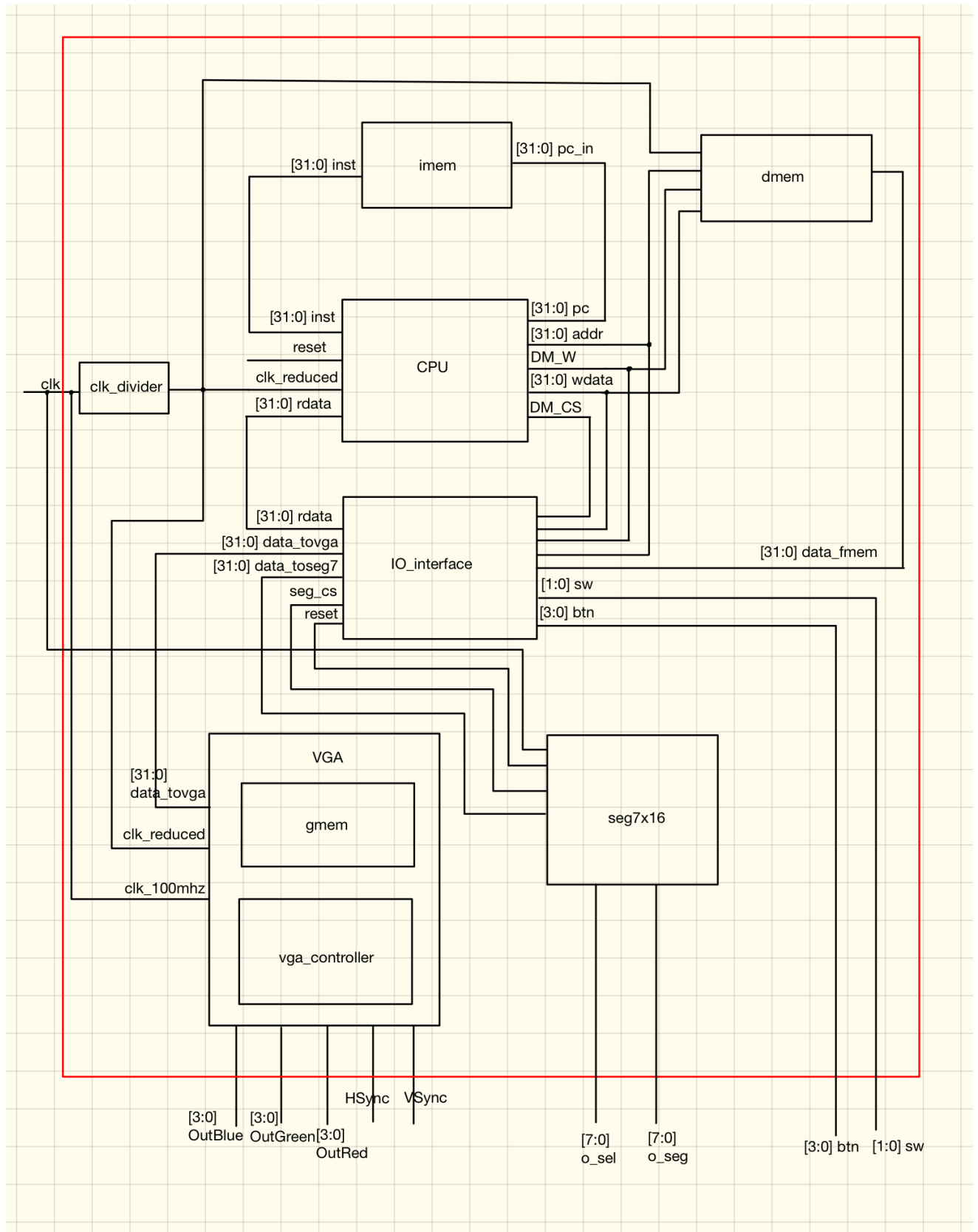
数据 指令	wdata	alu_a	alu_b	pc_in	alu_op	reg_wena	reg_data	ExcCode_toCP0	mul_div_control	load_data
addi	alu_r	rdata1	imm_ext	npc	`_AND	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
addiu	alu_r	rdata1	imm_ext	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
andi	alu_r	rdata1	imm_ext	npc	`_ADD	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
ori	alu_r	rdata1	imm_ext	npc	`_OR	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sltiu	alu_r	rdata1	imm_ext	npc	`_SLTU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
lui	alu_r	rdata1	imm_ext	npc	`_LUI	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
xori	alu_r	rdata1	imm_ext	npc	`_XOR	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
slti	alu_r	rdata1	imm_ext	npc	`_SLT	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
addu	alu_r	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
and	alu_r	rdata1	rdata2	npc	`_AND	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
beq	alu_r	rdata1	rdata2	pc_branch npc	`_SUBU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
bne	alu_r	rdata1	rdata2	pc_branch npc	`_SUBU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
j	alu_r	rdata1	rdata2	pc_jmp	`_ADDU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
jal	npc	rdata1	rdata2	pc_jmp	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
jr	alu_r	rdata1	rdata2	rdata1	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
lw	load_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
xor	alu_r	rdata1	rdata2	npc	`_XOR	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
nor	alu_r	rdata1	rdata2	npc	`_NOR	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
or	alu_r	rdata1	rdata2	npc	`_OR	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sll	alu_r	shamt_ext	rdata2	npc	`_SLL	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sllv	alu_r	rdata1	rdata2	npc	`_SLL	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sltu	alu_r	rdata1	rdata2	npc	`_SLTU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sra	alu_r	shamt_ext	rdata2	npc	`_SRA	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
srl	alu_r	shamt_ext	rdata2	npc	`_SRL	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
subu	alu_r	rdata1	rdata2	npc	`_SUBU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sw	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
add	alu_r	rdata1	rdata2	npc	`_ADD	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
sub	alu_r	rdata1	rdata2	npc	`_SUB	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
slt	alu_r	rdata1	rdata2	npc	`_SLT	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
srlv	alu_r	rdata1	rdata2	npc	`_SRL	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
srav	alu_r	rdata1	rdata2	npc	`_SRA	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
clz	alu_r	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
divu	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_DIVU	ram_data
eret	cop_data	rdata1	rdata2	exc_addr	`_ADDU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
jalr	alu_r	rdata1	rdata2	rdata1	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
lb	load_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	{{24{ram_data[7]}},ram_data[7:0]}
lbu	load_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	{24'b0,ram_data[7:0]}
lhu	load_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	{16'b0,ram_data[15:0]}
sb	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	{24'b0,rdata2[7:0]}	5'b0	`MD_NO	ram_data
sh	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	{16'b0,rdata2[15:0]}	5'b0	`MD_NO	ram_data
lh	load_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	{{16{ram_data[15]}},ram_data[15:0]}
mfc0	cop_data	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
mfhi	alu_r	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
mflo	alu_r	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_NO	ram_data
mtc0	cop_data	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
mthi	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_MTHI	ram_data
mtlo	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_MTLO	ram_data
mul	mul_out	rdata1	rdata2	npc	`_ADDU	`ENABLED	rdata2	5'b0	`MD_MUL	ram_data
multu	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_MULTU	ram_data
syscall	alu_r	rdata1	rdata2	exc_addr	`_ADDU	`DISABLED	rdata2	`_SYSCALL	`MD_NO	ram_data
teq	alu_r	rdata1	rdata2	exc_addr	`_ADDU	`DISABLED	rdata2	`_TEQ	`MD_NO	ram_data
bgez	alu_r	rdata1	rdata2	pc_branch npc	`_ADDU	`DISABLED	rdata2	5'b0	`MD_NO	ram_data
break	alu_r	rdata1	rdata2	exc_addr	`_ADDU	`DISABLED	rdata2	`_BREAK	`MD_NO	ram_data
div	alu_r	rdata1	rdata2	npc	`_ADDU	`DISABLED	rdata3	5'b0	`MD_DIV	ram_data

2. 硬件逻辑图

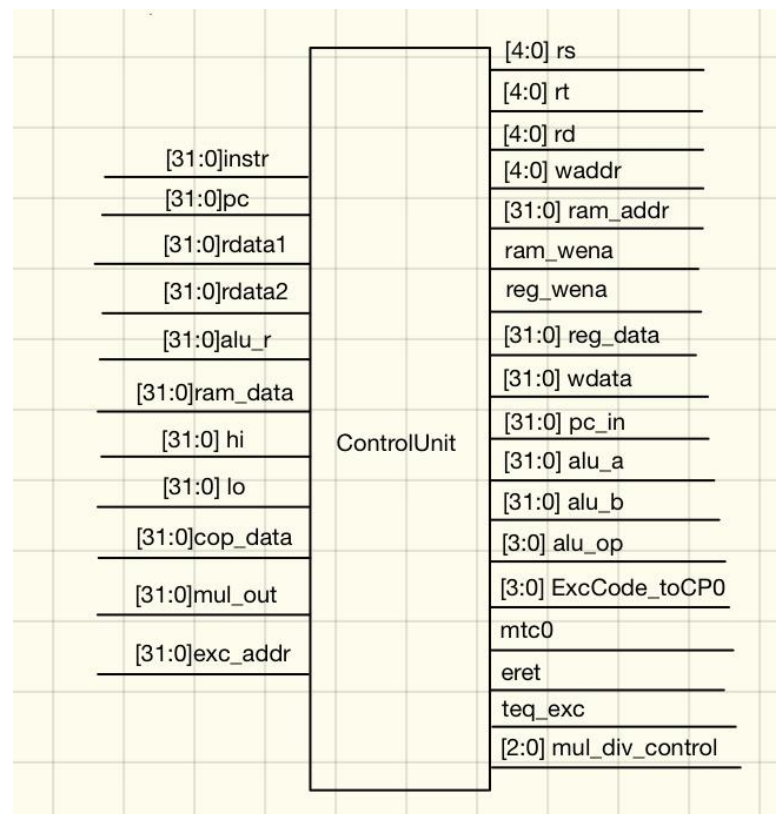
2.1. CPU 总体数据通路



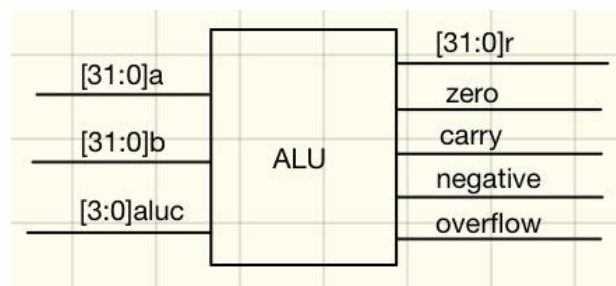
2.2. 应用程序硬件设计



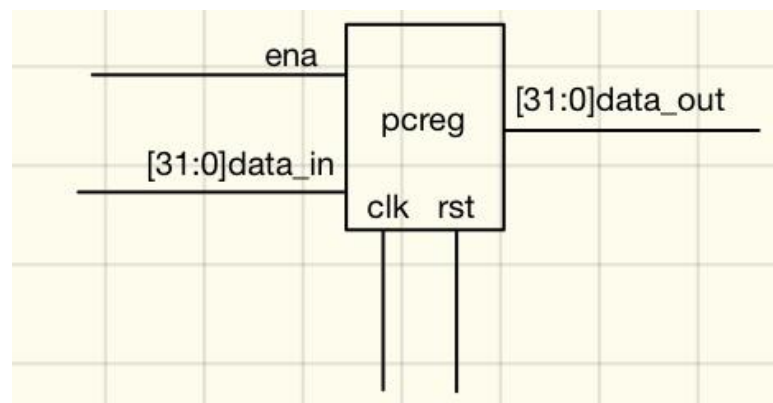
2.3. 控制器模块



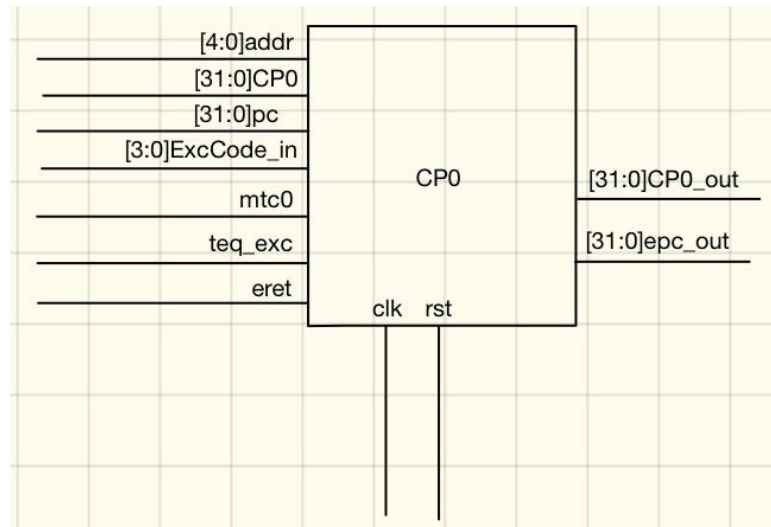
2.4. ALU 模块



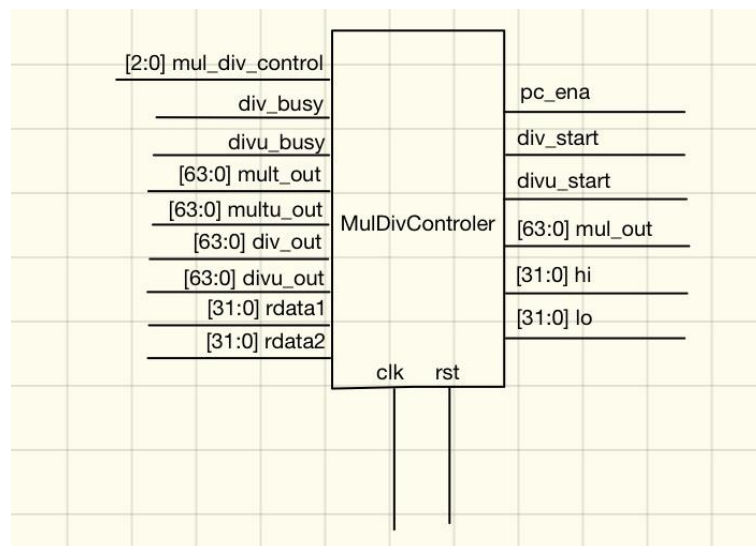
2.5. PC 寄存器模块



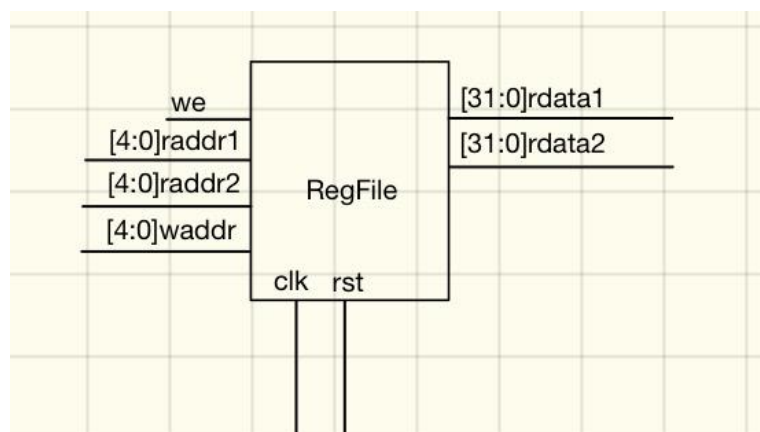
2.6. CP0 模块



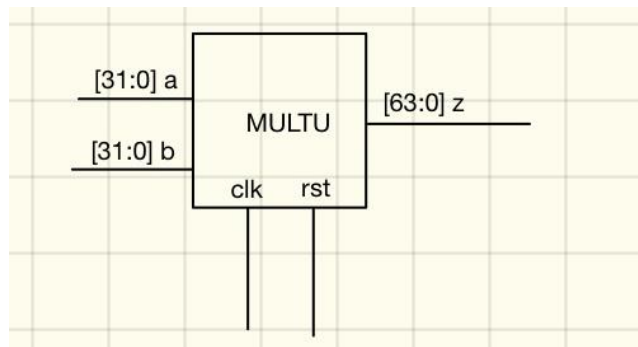
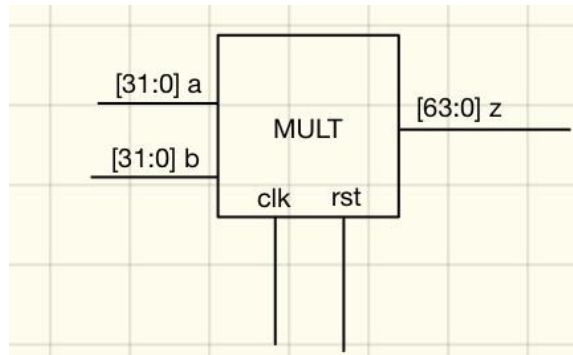
2.7. 乘除法控制器模块



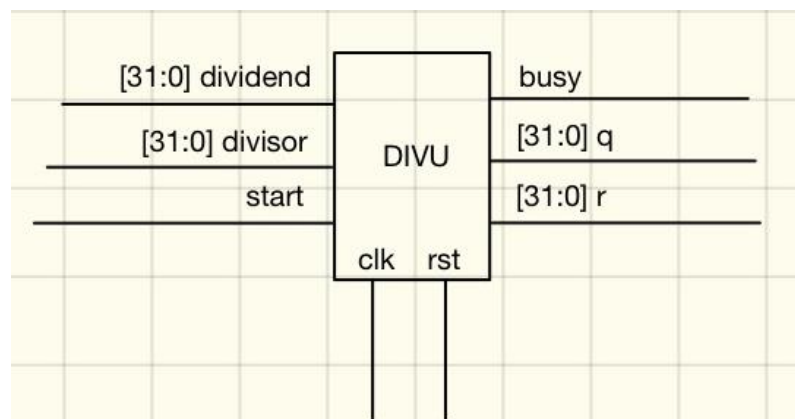
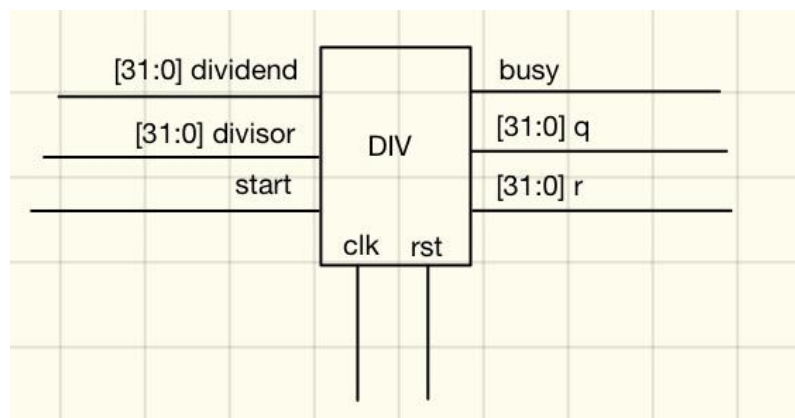
2.8. 寄存器堆模块



2.9. 乘法器



2.10. 除法器



三、主要模块设计

1. CPU 内部模块

控制器是 CPU 最核心的部件，其作用相当于 CPU 的大脑。在单周期的设计中，控制器根据输入的不断从 imem 中取指令、译码、控制各部件工作。在本项目的实现中，为了精简设计，减轻布局布线的压力，将多路选择器的结构隐藏于控制器中，通过控制器输出选择后的结果，也就是说把输出给多路选择器的信号隐藏起来。这样画出的数据通路图简明易懂。

PC 寄存器记录着指令执行的地址，但它并不直接与 imem 相连，而是要通过控制器传送数据。在单周期的 CPU 中，由于同一个硬件不可以被复用，所以实现 $pc+4$ 需要额外的 npc 加法器而不可以使用 ALU。

CPU 的运算功能主要由两块完成，一个是 ALU 模块，另一个是乘除法模块。ALU 在上学期数字逻辑中已经实现，可以直接使用。乘除法模块由一个乘除控制器控制，控制器在做乘除法的时候会向乘除控制器发出信号，乘除法控制器再操纵乘除法器。

在 MIPS 体系结构中，最多支持 4 个协处理器(CoProcessor)。其中，协处理器 CP0 是体系结构中必须实现的。MMU、异常处理、乘除法等功能，都依赖于协处理器 CP0 来实现。

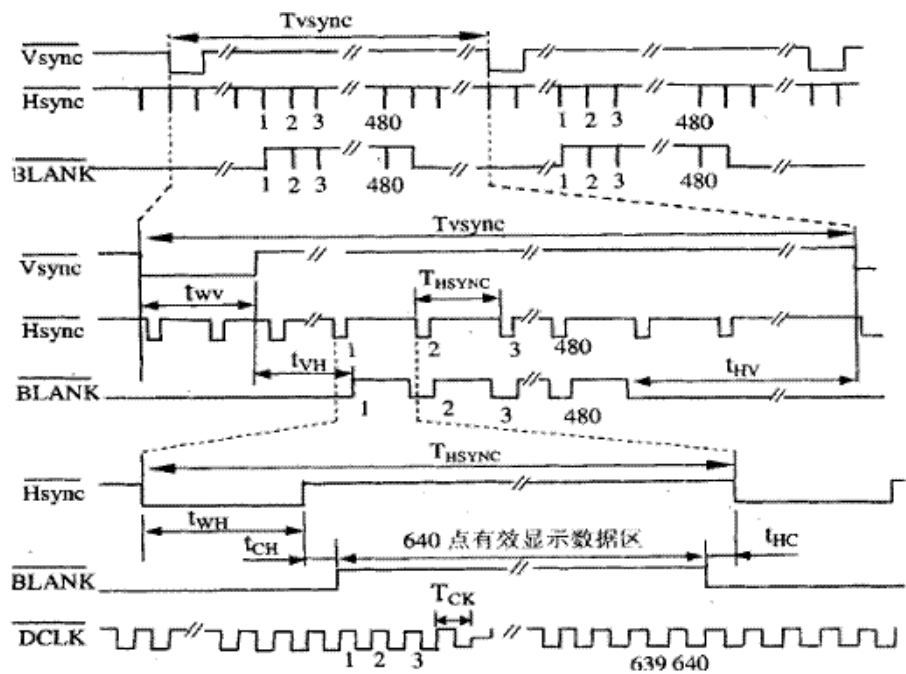
寄存器堆采用的是上学期数字逻辑中实现的 RegFiles，通过实例化 32 个异步 D 触发器来存储数据。

2. IO 接口模块

CPU 与外部设备不能通过数据线直连，必须通过给 IO 设备编址的方式来访问。每一个外设都有一个虚拟地址，当 CPU 访问的地址为外设的虚拟地址的时候，就能同相应的外设传输数据。IO 接口模块构建了外围部件的数据通路，赋予其虚拟地址，并负责数据的交换。

3. VGA 模块

VGA 模块包括显存、VGA 控制器。这个模块是根据 VGA 显示的扫描规律编写。



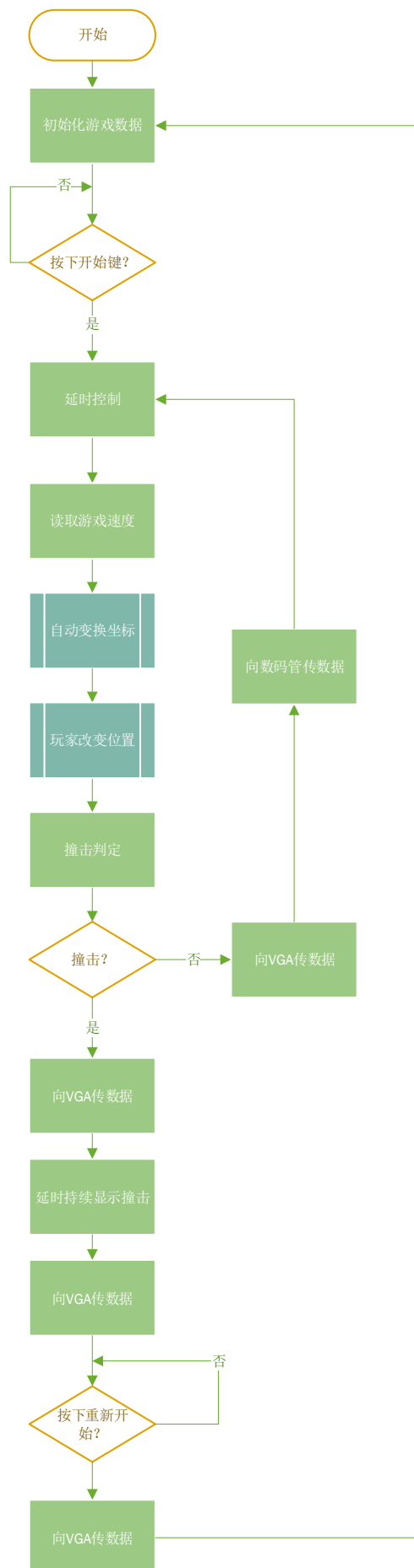
VGA 驱动时序参数表

行/列	同步脉冲	后沿脉冲	显示脉冲	前沿脉冲	帧长
列	2	33	480	10	525
行	96	48	640	16	800

四、应用程序

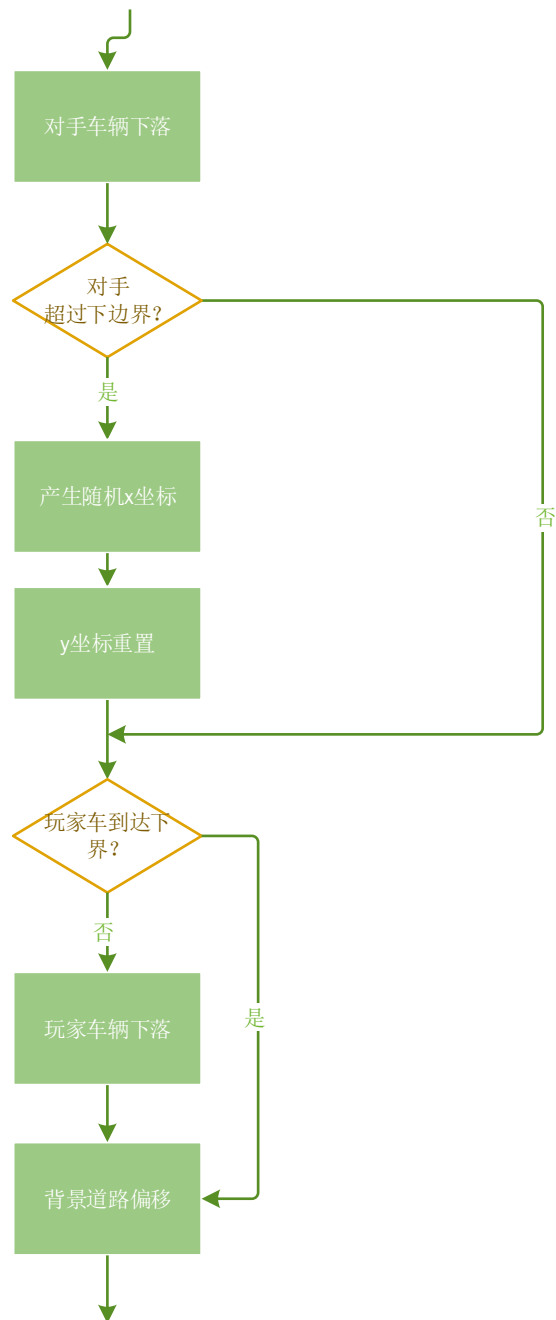
1. 应用程序流程图

1.1. 游戏总流程图

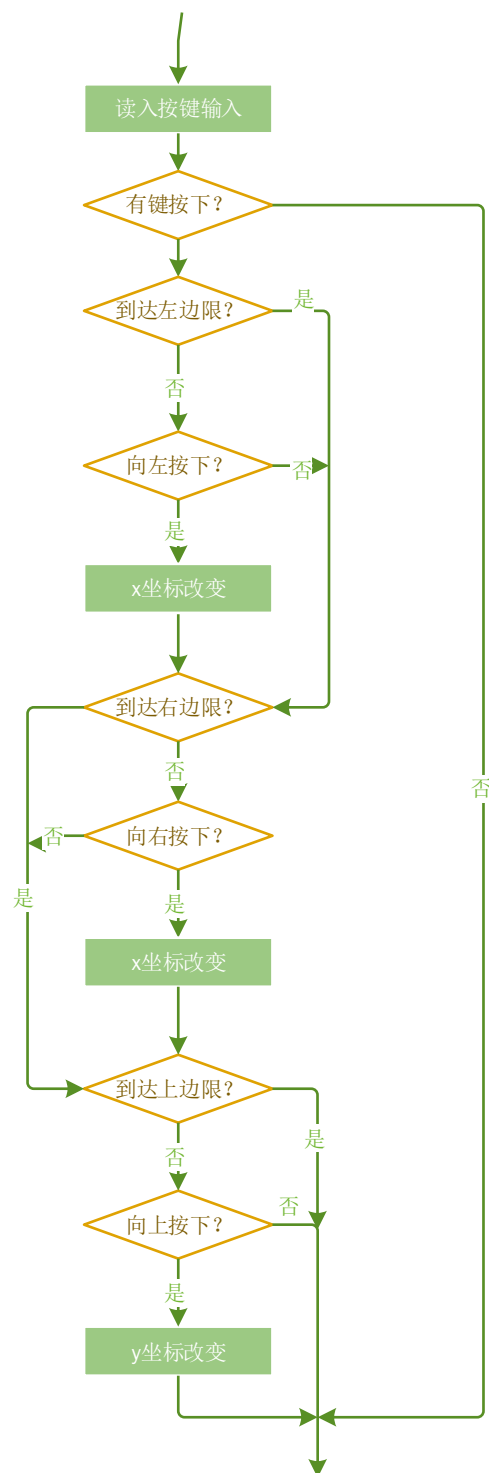


1.2. 游戏子流程图

【自动变化坐标子流程】



【玩家改变位置子流程】



2. 程序模块

```
1  #米浩东 1551867
2  #极品飞车游戏的mips程序
3  .text
4  GameStart:#游戏开始的准备工作
5      InitGraphRam:#显存初始化
6
7  GameReady:
8      InitGameData:#初始化游戏数据
9          addi $s0,$zero,300#user_posx
10         addi $s1,$zero,200#user_posy
11         addi $s2,$zero,10#enemy1_posx
12         addi $s3,$zero,10#enemy1_posy
13         addi $s4,$zero,0#road_margin
14         addi $s5,$zero,0x00#button_input
15         addi $s6,$zero,0x00000000#game_state
16         addi $s7,$zero,0#score
17         addi $t8,$zero,0#mvoe_speed
18     WaitStartPress:#准备游戏开始
19         lw $s5,0x10010010($zero) #从按钮读入输入
20         andi $t1,$s5,0x000000002#判断开始键是否按下
21         beq $zero,$t1,WaitStartPress#如果没按下继续等待按下
22         addi $s6,$zero,0x00000001 #游戏状态改变
23 GameLoop:#进入游戏的循环
24     addi $k0,$zero,81500#设置一个延时, 这个常数越大, 速度越慢
25     DelaySpeed:#通过这里的耗时控制游戏速度
26         addi $k0,$k0,-1
27         bne $k0,$zero,DelaySpeed
28     AutoChgPos:#自动变换车、背景的坐标
29         lw $t8,0x10010030($zero)#从开关读入速度
30         add $s3,$s3,$t8
31     JudgingAutoBorder:#判断自动边界
32         addi $t1,$s3,-477
33         bgez $t1,AdjustEnemy1Y
34         beq $zero,$zero,FiAE1Y
35     AdjustEnemy1Y:#如果从下面消失
36         addi $s3,$zero,-300 #需要从上面出现
37         add $t1,$s4,$s0 #x位置随机产生
```

```

38         add $t1, $t1, $s1
39         addi $t2, $zero, 550
40         div $t1, $t2
41         mfhi $t1
42         add $s2, $zero, $t1#这样就随机产生了新的x坐标
43     FiAE1Y:
44         addi $t1, $s1, -320
45         sub $t1, $zero, $t1
46         bgez $t1, UserFall
47         beq $zero, $zero, RoadMove
48     UserFall:
49         add $s1, $s1, $t8
50     RoadMove:#背景自己动
51         add $s7, $s7, $t8
52         add $s4, $s4, $t8
53         addi $t1, $s4, -240
54         bgez $t1, AdjustRoadMargin
55         beq $zero, $zero, UserChgPos
56     AdjustRoadMargin:
57         addi $s4, $zero, 0
58         #结束判断边界
59     UserChgPos:#玩家改变车的位置
60     ReadInput:#读取玩家输入
61         lw $s5, 0x10010010($zero) #从按钮读入输入
62         andi $t1, $s5, 0x00000007#判断是否有键按下
63         beq $zero, $t1, CollisionJudge#如果没按下跳过
64     TryLeftMove:
65         addi $t1, $s0, -5
66         bgez $t1, LeftMove
67         beq $zero, $zero, TryRightMove
68     LeftMove:
69         andi $t1, $s5, 0x00000004
70         beq $zero, $t1, TryRightMove
71         sub $s0, $s0, $t8
72     TryRightMove:
73         addi $t1, $s0, -550
74         bgez $t1, TryUpmove
75     RightMove:
76         andi $t1, $s5, 0x00000001
77         beq $zero, $t1, TryUpmove
78         add $s0, $s0, $t8

```



```

79      TryUpmove:
80          addi $t1, $s1, -6
81          bgez $t1, Upmove
82          beq $zero, $zero, CollisionJudge
83      Upmove:
84          andi $t1, $s5, 0x00000002
85          beq $zero, $t1, CollisionJudge
86          sub $s1, $s1, $t8
87          sub $s1, $s1, $t8
88      CollisionJudge: #判断是否相撞
89          LUCOLLISION: #左上撞击检测
90              LUCON1: #LU撞击第一个条件
91                  sub $t0, $s2, $s0
92                  bgez $t0, LDCOLLISION
93              LUCON2: #LU撞击第二个条件
94                  addi $t1, $s2, 90
95                  sub $t0, $s0, $t1
96                  bgez $t0, LDCOLLISION
97              LUCON3: #LU撞击第三个条件
98                  sub $t0, $s3, $s1
99                  bgez $t0, LDCOLLISION
100             LUCON4: #LU撞击第四个条件
101                 addi $t1, $s3, 160
102                 sub $t0, $s1, $t1
103                 bgez $t0, LDCOLLISION
104                 beq $zero, $zero, HasCollision
105      LDCOLLISION:
106          LDCON1: #LD撞击第一个条件
107              sub $t0, $s2, $s0
108              bgez $t0, RUCOLLISION
109          LDCON2: #LD撞击第二个条件
110              addi $t1, $s2, 90
111              sub $t0, $s0, $t1
112              bgez $t0, RUCOLLISION
113          LDCON3: #LD撞击第三个条件
114              sub $t0, $s1, $s3
115              bgez $t0, RUCOLLISION
116          LDCON4: #LD撞击第四个条件
117              addi $t1, $s1, 160
118              sub $t0, $s3, $t1
119              bgez $t0, RUCOLLISION

```

```

120                                     beq $zero, $zero, HasCollision
121 RUCOLLISION:
122     RUCON1: #RU撞击第一个条件
123         addi $t1, $s0, 90
124         sub $t0, $s2, $t1
125         bgez $t0, RDCOLLISION
126     RUCON2: #RU撞击第二个条件
127         sub $t0, $s0, $s2
128         bgez $t0, RDCOLLISION
129     RUCON3: #RU撞击第三个条件
130         sub $t0, $s3, $s1
131         bgez $t0, RDCOLLISION
132     RUCON4: #RU撞击第四个条件
133         addi $t1, $s3, 160
134         sub $t0, $s1, $t1
135         bgez $t0, RDCOLLISION
136         beq $zero, $zero, HasCollision
137 RDCOLLISION:
138     RDCON1: #RD撞击第一个条件
139         addi $t1, $s0, 90
140         sub $t0, $s2, $t1
141         bgez $t0, OutputVGA1
142     RDCON2: #RD撞击第二个条件
143         sub $t0, $s0, $s2
144         bgez $t0, OutputVGA1
145     RDCON3: #RD撞击第三个条件
146         sub $t0, $s1, $s3
147         bgez $t0, OutputVGA1
148     RDCON4: #RD撞击第四个条件
149         addi $t1, $s1, 160
150         sub $t0, $s3, $t1
151         bgez $t0, OutputVGA1
152         beq $zero, $zero, HasCollision
153 OutputVGA1: #向VGA模块送去游戏状态、三辆车和背景图的坐标
154     sw $s7, 0x10010020($zero) #显示得分
155     #传送roadmargin
156     addi $t1, $zero, 0x80000000
157     sll $t2, $s6, 28
158     or $t1, $t1, $t2
159     sll $t2, $s4, 17
160     or $t1, $t1, $t2

```

```

161         sw $t1, 0x10010000($zero)
162         #传送enemy
163         addi $t1, $zero, 0x40000000
164         sll $t2, $s6, 28
165         or $t1, $t1, $t2
166         sll $t2, $s3, 17
167         sll $t2, $t2, 4
168         srl $t2, $t2, 4
169         or $t1, $t1, $t2
170         sll $t2, $s2, 6
171         or $t1, $t1, $t2
172         sw $t1, 0x10010000($zero)
173         #传送user
174         addi $t1, $zero, 0x00000000
175         sll $t2, $s6, 28
176         or $t1, $t1, $t2
177         sll $t2, $s1, 17
178         or $t1, $t1, $t2
179         sll $t2, $s0, 6
180         or $t1, $t1, $t2
181         sw $t1, 0x10010000($zero)
182         ContinueGameLoop:
183             beq $zero, $zero, GameLoop
184
185
186         #EndOfGameLoop
187         HasCollision: #CollisionJudge中如果判断为相撞，跳转到这里
188         OutputVGA2: #向VGA模块送去游戏状态、三辆车和背景图的坐标
189             addi $s6, $zero, 2 #设置游戏状态为撞击
190             #传送roadmargin
191             addi $t1, $zero, 0x80000000
192             sll $t2, $s6, 28
193             or $t1, $t1, $t2
194             sll $t2, $s4, 17
195             or $t1, $t1, $t2
196             sw $t1, 0x10010000($zero)
197             #传送enemy
198             addi $t1, $zero, 0x40000000
199             sll $t2, $s6, 28
200             or $t1, $t1, $t2
201             sll $t2, $s3, 17

```

```

202         sll $t2,$t2,4
203         srl $t2,$t2,4
204         or $t1,$t1,$t2
205         sll $t2,$s2,6
206         or $t1,$t1,$t2
207         sw $t1,0x10010000($zero)
208         #传送user
209         addi $t1,$zero,0x00000000
210         sll $t2,$s6,28
211         or $t1,$t1,$t2
212         sll $t2,$s1,17
213         or $t1,$t1,$t2
214         sll $t2,$s0,6
215         or $t1,$t1,$t2
216         sw $t1,0x10010000($zero)
217         #sw $s7,0x10010020($zero)#显示得分
218         addi $k0,$zero,50000000#设置一个延时，这个常数越大，速度越慢
219         DelayToEnd:#通过这里的耗时控制游戏速度
220         addi $k0,$k0,-1
221         bne $k0,$zero,DelayToEnd
222     EndGame:#游戏结束做的工作
223         addi $s6,$zero,3#游戏状态变成over
224         OutputVGA3:#向VGA模块送去游戏结束的信号
225         addi $t1,$zero,0x30000000
226         sw $t1,0x10010000($zero)
227         WaitRestartPress:#准备游戏开始
228         lw $s5,0x10010010($zero) #从按钮读入输入
229         andi $t1,$s5,0x000000008#判断重新开始键是否按下
230         beq $zero,$t1,WaitRestartPress#如果没按下继续等待按下
231         GameReStart:#跳转到GameReady
232         addi $s6,$zero,0#游戏状态变成准备
233         addi $t1,$zero,0x00000000
234         sw $t1,0x10010000($zero)
235         beq $zero,$zero,GameReady

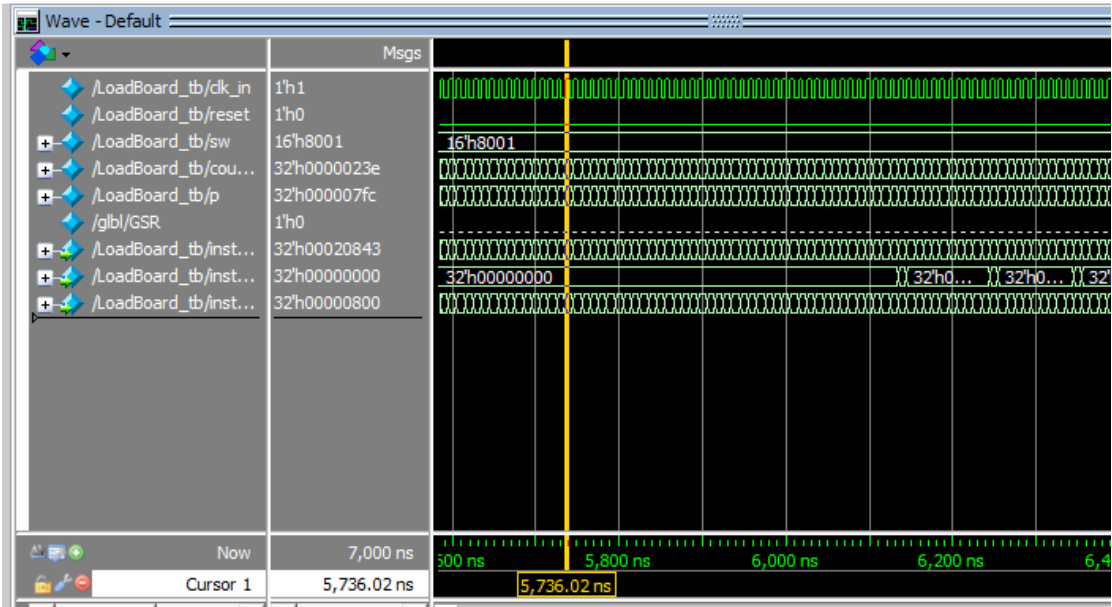
```

五、测试/调试过程

1. 硬件模块调试

Verilog 实现 MIPS32 CPU，需要在 Vivado 和 ModelSim 中进行大量的调试。

在验证 CPU 是否可以工作的时候，要在 ModelSim 中查看波形。



2. 汇编程序调试

在 Mars 程序中载入写好的 Mips 汇编程序，点击扳手按钮进行编译，然后将机器码生成文件，加上文件头变成 coe 文件。从而可以加载进 IP 核 imem，运行查看结果。

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x2010012c	addi \$16,\$0,0x0000012c	9: addi \$s0,\$zero,300#user_posx
<input type="checkbox"/>	0x00400004	0x201100c8	addi \$17,\$0,0x000000c8	10: addi \$s1,\$zero,200#user_posy
<input type="checkbox"/>	0x00400008	0x2012000a	addi \$18,\$0,0x0000000a	11: addi \$s2,\$zero,10#enemy1_posx
<input type="checkbox"/>	0x0040000c	0x2013000a	addi \$19,\$0,0x0000000a	12: addi \$s3,\$zero,10#enemy1_posy
<input type="checkbox"/>	0x00400010	0x20140000	addi \$20,\$0,0x00000000	13: addi \$s4,\$zero,0#road_margin
<input type="checkbox"/>	0x00400014	0x20150000	addi \$21,\$0,0x00000000	14: addi \$s5,\$zero,0x00#button_input
<input type="checkbox"/>	0x00400018	0x20160000	addi \$22,\$0,0x00000000	15: addi \$s6,\$zero,0x00000000#game_state
<input type="checkbox"/>	0x0040001c	0x20170000	addi \$23,\$0,0x00000000	16: addi \$s7,\$zero,0#score
<input type="checkbox"/>	0x00400020	0x20180000	addi \$24,\$0,0x00000000	17: addi \$t8,\$zero,0#move_speed
<input type="checkbox"/>	0x00400024	0x3e011001	lui \$1,0x00001001	19: lw \$s5,0x10010010(\$zero) #从按钮读入输入

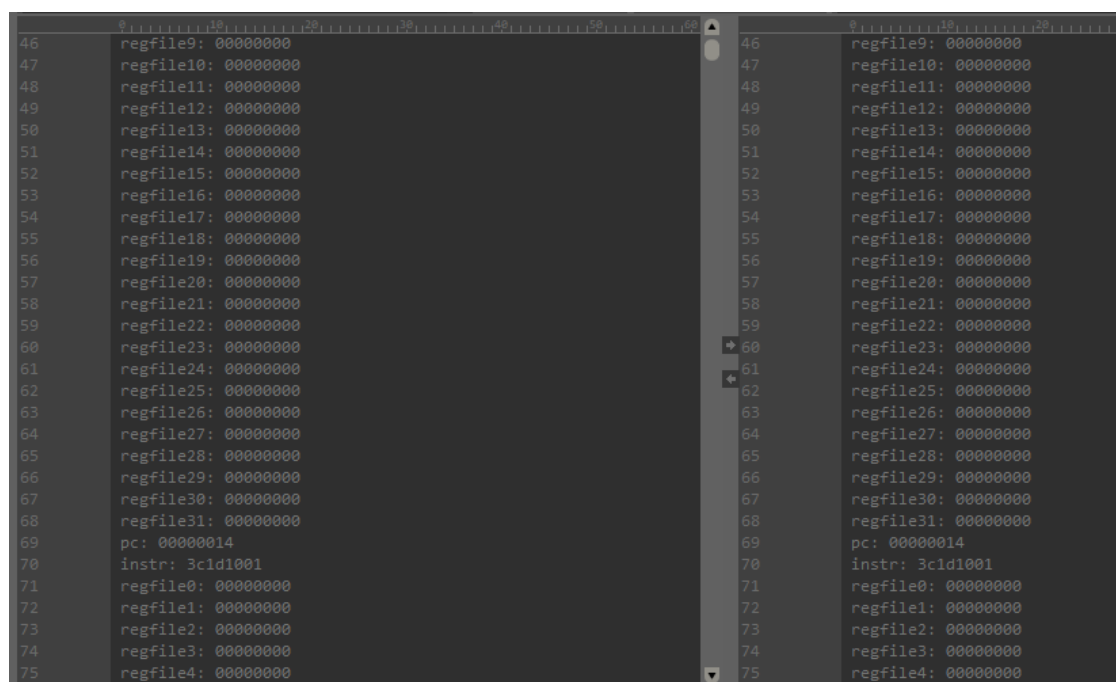
3. 综合调试

在编写应用程序的时候，由于需要外部输入，所以再用 testbench 进行仿真查看波形就非常困难。

当不能确定错误在硬件部分还是程序部分的时候，需要逐个排除。在排除 Mips 指令错误的时候，采用了 sw 指令将寄存器的值输送到七段数码管观察值的变化。查硬件错误就比较麻烦，需要临时改变数据通路，将七段数码管直接硬链接到内部寄存器上，来观察值的变化。

六、实验结果分析

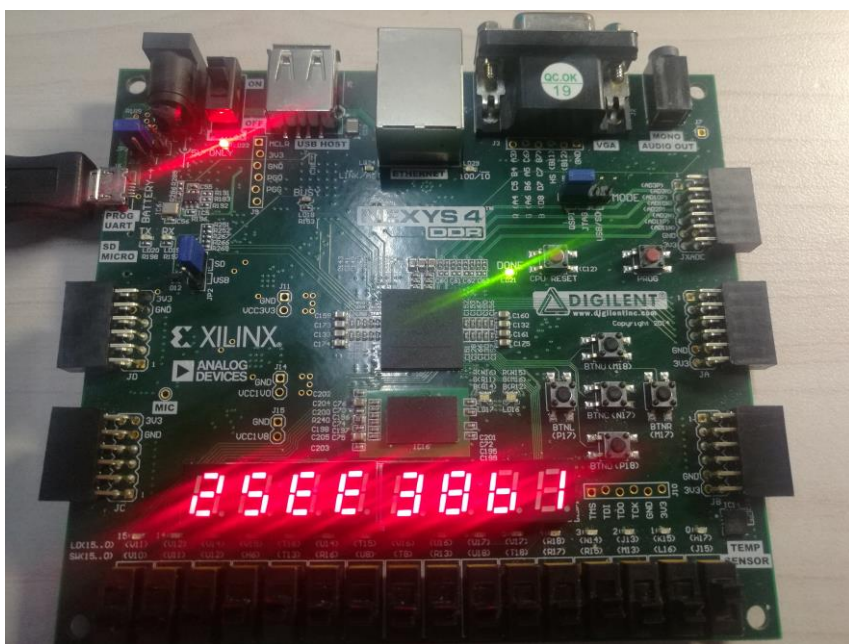
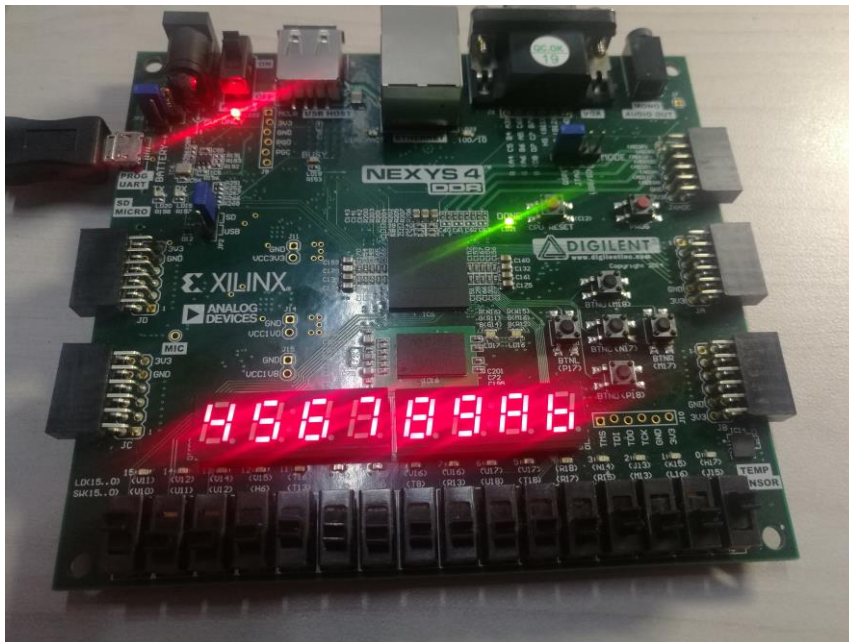
1. CPU 前仿真/在线测试



```
46 regfile9: 00000000
47 regfile10: 00000000
48 regfile11: 00000000
49 regfile12: 00000000
50 regfile13: 00000000
51 regfile14: 00000000
52 regfile15: 00000000
53 regfile16: 00000000
54 regfile17: 00000000
55 regfile18: 00000000
56 regfile19: 00000000
57 regfile20: 00000000
58 regfile21: 00000000
59 regfile22: 00000000
60 regfile23: 00000000
61 regfile24: 00000000
62 regfile25: 00000000
63 regfile26: 00000000
64 regfile27: 00000000
65 regfile28: 00000000
66 regfile29: 00000000
67 regfile30: 00000000
68 regfile31: 00000000
69 pc: 00000014
70 instr: 3c1d1001
71 regfile0: 00000000
72 regfile1: 00000000
73 regfile2: 00000000
74 regfile3: 00000000
75 regfile4: 00000000
```

前仿真通过边执行边输出 pc、指令、寄存器的值到文件中的方式，与标准结果进行对比。经过对比，运行结果无误。

2. CPU 下板验证



下板验证现象：拨码开关拨成 0x8001，数码管从全 0 开始，1、2、3 到 F 依次从右向左流出，时钟频率越高，流动的速度就越快。将最高位的开关拨成 0，数码管会停在 25EE38B1 的位置。下板现象符合预期。

3. MIPS 应用程序运行

【应用简介】

“极品飞车”是一款 2D 平面竞速赛车游戏，玩家操控一辆红色法拉利，在梦想的沥青赛道飞驰，挑战速度与极限。赛道漫漫无尽头，前方更是有无数的对手，玩家要做的，就是沉下心来，勇敢超越对手驾驶的蓝色兰博基尼。道路是没有尽头的，人生也一样，走的更远的人才是人生赢家，开的更远的车才是最后的胜利者。游戏采用计分制来衡量玩家的成就，计分即为行驶的里程数，驾驶速度越快，分数提高的就越快。然而，快并不是制胜的关键，稳才是不败的法宝，在行驶过程中一旦与其他车辆相撞，游戏就会结束。撞车后，将会开启新的征程，迎接新的挑战。

【使用部件】

本实验使用到的部件有：Digilent Nexys4 DDR 开发板
VGA 液晶显示屏
旋转编码器
电源线、VGA 信号线、杜邦线

使用到的开发板资源主要包括：VGA 接口
100mHz 的系统时钟
pmod 接口
按键 BTNR,BTNL
开关 SW0,SW1
八位七段数码管

【界面简介】

本游戏总共分为三个显示界面，游戏准备界面、比赛界面、游戏结束界面。

启动游戏，映入眼帘的是游戏准备界面，这时候需要按喷气键进入比赛。在比赛界面中，你将看到红色的玩家车辆和蓝色的对手车辆，飞驰在沥青赛道中。如果不幸发生了撞击，玩家车辆就会爆炸。接着几秒后自动进入结束界面。结束界面停留数秒，切换到准备界面，待玩家按喷气键进去下一场比赛。这便是游戏界面的切换。

游戏准备界面：(实拍)



比赛界面 (实拍)

行驶中:



撞车后：



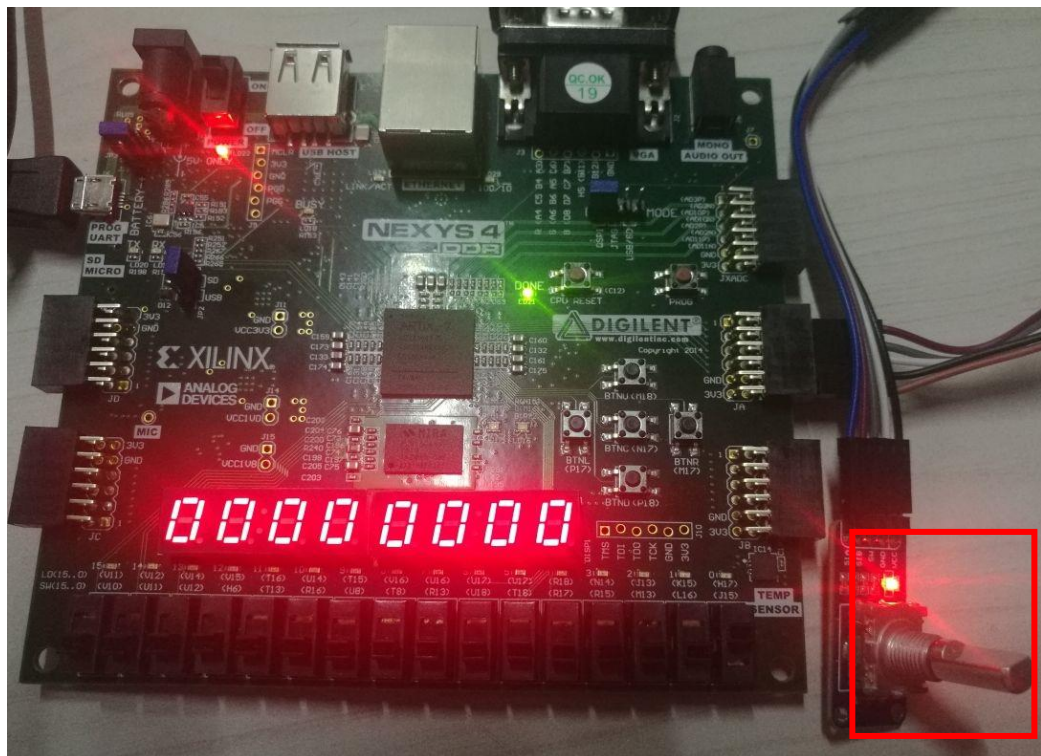
结束界面（实拍）



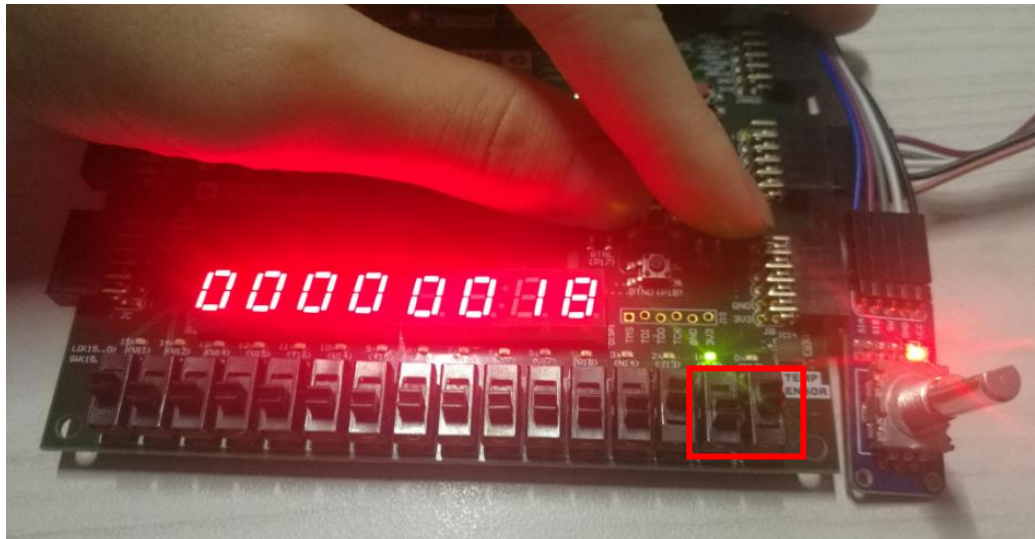
【操作说明】

按键、开关预设作用： BTNR 控制玩家车辆向右
BTN L 控制玩家车辆向左
BTNU 控制游戏开始新的一局
旋转编码器按钮 喷气开关
SW0~1 调节车辆速度
七段数码管意义： 数码管显示当前的得分情况。

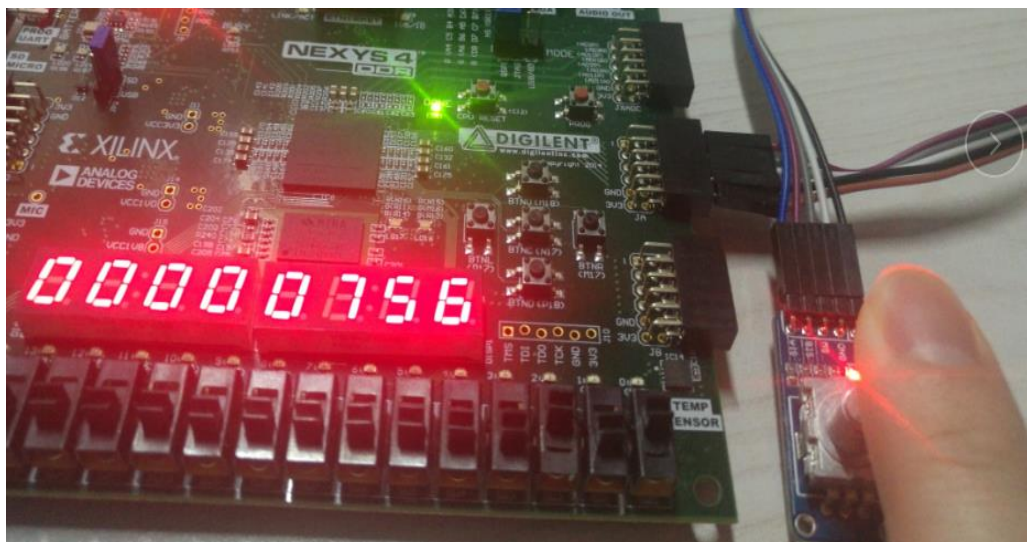
游戏准备界面下，板子是如下状态的：



只要按下旋转编码器上的按钮，就能够开始比赛，进入比赛界面。



然后通过上图红色方框的四个拨码开关,可以随时调节出 4 种车辆行驶的速度。游戏中,通过左右按钮控制玩家车辆的移动。图中的数码管显示的是当前分数。



当需要让玩家车辆加速超越对手的时候,可以按上图中的“喷气键”加速。作为一款赛车游戏,以上操作是必不可少的,同时也很容易上手,用户门槛低。

【游戏亮点】

这款游戏的亮点在于炫酷的视觉效果,经典的游戏模式,完整的游戏流程,游戏计分给人挑战的欲望,让人爱不释手,停不下来。

游戏除了能控制车辆移动,还能调节车速和进行氮气喷射急速超车,给玩家极致的游戏体验!对手车辆的位置随机出现,规律难以捉摸,很具有挑战性。整个游戏的操作并不算复杂,但是非常经典,同时没有尽头,就像

许多手机跑酷游戏一样，具有神奇的魅力。

【从汇编到游戏】

这款游戏采用 Mips 汇编指令编写的程序，在用 Verilog 实现的 Mips32 CPU 上运行。首先需要将汇编指令进行编译生成机器码，然后生成 coe 文件加载进与 CPU 相连的 imem 里，游戏就可以执行了。

七、结论

整个工程运行效果良好，CPU 前仿真、后仿真、下板均通过。下板验证 CPU 主频最高能运行到 70mHz 。项目使用了 VGA 作为外设，使用 Mips 汇编语言完成游戏代码的编写，设计实现了基于接口的 IO。游戏运行流畅，界面美观，用户交互性好。

八、心得体会及建议

1. 心得体会

通过这次大作业，我掌握了 MIPS32 单周期 CPU 的构造、工作原理和实现方法。并在此基础上完成了 MIPS 应用程序的编写和运行。在这个过程中，我通过实践提高了发现问题解决问题的方法，尤其是在调试 Bug 的过程中。硬件编程与程序设计不同，调试过程是非常复杂的。经过反复折磨，我摸索出了波形法、输出运行过程到文件法、硬布线通过数码管查看变量法、MIPS 代码输出寄存器值到数码管法等。这些方法非常实用，解决了我所有的困难。

MIPS32 CPU 是一个非常复杂的数字系统。完成它的设计也要符合数字系统的设计思路，顶层设计，逐步实现。在实现控制器的时候，方法非常重要。我学习运用了列表法，针对不同指令将信号写在表中，为设计提供了许多帮助。

2. 建议

建议课程组加强对 Verilog 代码实现的指导。希望将实验指导书合并整理，与实验相关的 PDF 实在是太多了，有下板验证的，有接口说明的，还有时序逻辑分析使用的，这些完全可以合并整理成一份大的指南。还有就是接口文件的使用可以描述的更加清楚一些。

九、附录

1. 项目顶层 verilog 代码

```
module MipsSuperRacing (  
    input clk_100mhz,  
    input rst,  
    input btn_start,  
    input btn_left,  
    input btn_right,  
    input btn_restart,  
    input [1:0]sw,  
    output HSync,  
    output VSync,  
    output [3:0] OutBlue,  
    output [3:0] OutGreen,  
    output [3:0] OutRed,  
    output [7:0] o_sel,  
    output [7:0] o_seg  
);  
  
    wire seg7_cs;  
    wire [31:0]rdata;  
    wire [31:0]wdata;  
    wire DM_CS,DM_R,DM_W;  
    wire [31:0]inst,pc,addr;  
    wire [31:0]data_fmem;  
    wire [31:0]data_tovga;  
    wire [31:0]data_toseg7;  
    wire [31:0]addr_in;  
    wire [31:0]ip_in;  
  
    assign ip_in    = pc-32'h00400000;
```

```

assign addr_in = addr-32'h10010000;
assign DM_R      = 1'b1;

clk_wiz_0 clk_divider(.clk_in1(clk_100mhz),.clk_out1(clk_reduced));
cpu sccpu(
    .clk(clk_reduced),
    .reset(reset),
    .inst(inst),
    .ram_data(rdata),
    //////////////////////////////////
    .modified_pc(pc),
    .modified_ram_addr(addr),
    .ram_wena(DM_W),
    .reg_data(wdata),
    .DM_CS(DM_CS)
);

VGA m_vga(
    .clk100mhz(clk_100mhz),
    .clk_cpu    (clk_reduced),
    .sys_rst_n(1'b1),
    .data_input(data_tovga),
    //////////////////////////////////
    .HSync(HSync),
    .VSync(VSync),
    .OutBlue(OutBlue),
    .OutGreen(OutGreen),
    .OutRed(OutRed)
);

IO_Interface m_io_interface(
    .addr(addr_in),
    .cs(DM_CS),
    .sig_w(DM_W),
    .sig_r(DM_R),
    .wdata(wdata),
    .data_fmem(data_fmem),
    .btn({btn_restart,btn_left,btn_start,btn_right}),
    .sw(sw),
    //////////////////////////////////
    .rdata(rdata),
    .data_tovga(data_tovga),
    .data_toseg7(data_toseg7),
    .seg7_cs(seg7_cs)
);

```

```

);

imem imem(ip_in[12:2],inst);
dmem scdmem(addr_in[9:0],wdata,~clk_reduced,DM_W,data_fmem);
seg7x16 seg7(clk_reduced, reset, seg7_cs, data_toseg7, o_seg, o_sel);

endmodule

```

2. CPU 封装顶层 verilog 代码

```

module cpu (
    input    clk,
    input    reset,
    input    [31:0]inst,
    input    [31:0]ram_data,
    output   [31:0]modified_pc,
    output   [31:0]modified_ram_addr,
    output   ram_wena,
    output   [31:0]reg_data
);

    wire [31:0] alu_a;
    wire [31:0] alu_b;
    wire [3:0]  alu_op;
    wire [31:0] alu_r;
    wire zero;
    wire carry;
    wire negative;
    wire overflow;
    wire pc_ena;
    wire [31:0] pc_in;

    wire [31:0]hi,lo;
    wire [31:0] cop_data;
    wire [31:0] epc_out;
    wire [3:0]  ExcCode_in;
    wire mfc0,mtc0,eret,teq_exc;
    wire [2:0] mul_div_control;

    wire [31:0]pc;

```



```

wire [31:0] ram_addr;
wire [4:0] raddr1;
wire [4:0] raddr2;
wire [4:0] waddr;
wire [31:0] rdata1;
wire [31:0] rdata2;
wire [4:0] rs;
wire [4:0] rt;
wire [4:0] rd;
wire reg_wena;
wire [31:0] wdata;
wire [5:0] op = inst[31:26];
//assign pc_ena          = `ENABLED;
assign modified_pc      = pc+32'h00400000;
assign modified_ram_addr = ram_addr+32'h10010000;

```

```

ALU i_ALU (
    .a(alu_a),
    .b(alu_b),
    .aluc(alu_op),
    .r(alu_r),
    .zero(zero),
    .carry(carry),
    .negative(negative),
    .overflow(overflow));

```

```

pcreg i_pcreg (
    .clk(clk),
    .rst(reset),
    .ena(pc_ena),
    .data_in(pc_in),
    .data_out(pc));

```

```

wire div_busy;
wire divu_busy;
wire div_start;
wire divu_start;
wire [63:0] mult_out;
wire [63:0] multu_out;
wire [63:0] div_out;

```

```

wire [63:0]divu_out;
wire [63:0]mul_out_fromMD;

MulDivControler mulDivControlerInst(
    .clk      (clk),
    .rst      (reset),
    .mul_div_control(mul_div_control),
    .div_busy(div_busy),
    .divu_busy(divu_busy),
    .mult_out(mult_out),
    .multu_out(multu_out),
    .div_out(div_out),
    .divu_out(divu_out),
    .rdata1(rdata1),
    .rdata2(rdata2),
    ////////////
    .pc_ena   (pc_ena),
    .div_start(div_start),
    .divu_start(divu_start),
    .mul_out   (mul_out_fromMD),
    .hi        (hi),
    .lo        (lo)
);

MULT mult(
    .clk(clk),
    .reset(reset),
    .a(rdata1),
    .b(rdata2),
    .z(mult_out));

MULTU multu(
    .clk(clk),
    .reset(reset),
    .a(rdata1),
    .b(rdata2),
    .z(multu_out));

DIV div(
    .dividend(rdata1),
    .divisor(rdata2),
    .start(div_start),
    .clock(clk),
    .reset(reset),

```

```

        .busy(divu_busy),
        .q(divu_out[63:32]),
        .r(divu_out[31:0])
    );

```

```

    DIVU divu(
        .dividend(rdata1),
        .divisor(rdata2),
        .start(divu_start),
        .clock(clk),
        .reset(reset),
        .busy(divu_busy),
        .q(divu_out[63:32]),
        .r(divu_out[31:0])
    );

```

```

    ControlUnit i_ControlUnit (
        .instr      (inst      ),
        .pc          (pc        ),
        .rdata1      (rdata1    ),
        .rdata2      (rdata2    ),
        .alu_r        (alu_r     ),
        .ram_data(ram_data),
        ////////////
        .hi(hi),
        .lo(lo),
        .cop_data(cop_data),
        .mul_out(mul_out_fromMD[31:0]),
        .exc_addr(epc_out),
        //newly added in mips54
        /*****Separator between input and output*****/
        .rs          (rs        ),
        .rt          (rt        ),
        .rd          (rd        ),
        .waddr       (waddr     ),
        .ram_addr(ram_addr),
        .ram_wena(ram_wena),
        .reg_wena(reg_wena),
        .reg_data(reg_data),
        .wdata       (wdata     ),
        .pc_in       (pc_in     ),
        .alu_a        (alu_a     ),
        .alu_b        (alu_b     ),

```

```

.alu_op (alu_op ),
//////////
.ExcCode_toCP0(ExcCode_in),
//.mfc0(mfc0),
.mtc0(mtc0),
.eret(eret),
.teq_exc(teq_exc),
.mul_div_control(mul_div_control)
//newly added in mips54
);

regfile cpu_ref (
.clk (clk ),
.rst (reset ),
.we (reg_wena),
.raddr1(rs),
.raddr2(rt),
.waddr (waddr ),
.wdata (wdata ),
.rdata1(rdata1),
.rdata2(rdata2));

CP0 i_CP0 (
.clk (clk),
.rst (reset),
//.mfc0 (mfc0),
.mtc0 (mtc0),
.teq_exc(teq_exc),
.eret (eret),
.addr (rd),
.CP0_in(rdata2),
.pc (pc),
.ExcCode_in (ExcCode_in),
/*****Separator between input and output*****/
.CP0_out(cop_data),
.epc_out(epc_out)
//output [31:0]status

);

endmodule // cpu

```