**同济大学**

**计算机科学与技术系**

**计算机组成原理课程大作业实验报告**

****

**学 号 1652270**

**姓 名 冯舜**

**专 业 计算机科学与技术**

**授课老师 陈永生**

日 期 2018/6

目录

[一、 实验目标 6](#_Toc517253384)

[二、 总体设计 6](#_Toc517253385)

[1. 作品功能设计及原理说明 6](#_Toc517253386)

[三、 主要模块（指令）设计 9](#_Toc517253387)

[1. ADD 9](#_Toc517253388)

[1) 操作流程（不用流程图表示，用列表表示） 9](#_Toc517253389)

[2) 数据通路图 9](#_Toc517253390)

[2. ADDU 9](#_Toc517253391)

[1) 操作 9](#_Toc517253392)

[2) 数据通路图 10](#_Toc517253393)

[3. SUB 10](#_Toc517253394)

[1) 操作 10](#_Toc517253395)

[2) 数据通路图 10](#_Toc517253396)

[4. SUBU 10](#_Toc517253397)

[1) 操作 10](#_Toc517253398)

[2) 数据通路图 11](#_Toc517253399)

[5. AND 11](#_Toc517253400)

[1) 操作 11](#_Toc517253401)

[2) 数据通路图 11](#_Toc517253402)

[6. OR 11](#_Toc517253403)

[1) 操作 11](#_Toc517253404)

[2) 数据通路图 12](#_Toc517253405)

[7. XOR 12](#_Toc517253406)

[1) 操作 12](#_Toc517253407)

[2) 数据通路图 12](#_Toc517253408)

[8. NOR 12](#_Toc517253409)

[1) 操作 12](#_Toc517253410)

[2) 数据通路图 13](#_Toc517253411)

[9. SLT 13](#_Toc517253412)

[1) 操作 13](#_Toc517253413)

[2) 数据通路图 13](#_Toc517253414)

[10. SLTU 13](#_Toc517253415)

[1) 操作 13](#_Toc517253416)

[2) 数据通路图 14](#_Toc517253417)

[11. SLL 14](#_Toc517253418)

[1) 操作 14](#_Toc517253419)

[2) 数据通路图 14](#_Toc517253420)

[12. SRL 14](#_Toc517253421)

[1) 操作 14](#_Toc517253422)

[2) 数据通路图 15](#_Toc517253423)

[13. SRA 15](#_Toc517253424)

[1) 操作 15](#_Toc517253425)

[2) 数据通路图 15](#_Toc517253426)

[14. SLLV 15](#_Toc517253427)

[1) 操作 15](#_Toc517253428)

[2) 数据通路图 16](#_Toc517253429)

[15. SRLV 16](#_Toc517253430)

[1) 操作 16](#_Toc517253431)

[2) 数据通路图 16](#_Toc517253432)

[16. SRAV 16](#_Toc517253433)

[1) 操作 16](#_Toc517253434)

[2) 数据通路图 17](#_Toc517253435)

[17. JR 17](#_Toc517253436)

[1) 操作 17](#_Toc517253437)

[2) 数据通路图 17](#_Toc517253438)

[18. ADDI 17](#_Toc517253439)

[1) 操作 17](#_Toc517253440)

[2) 数据通路图 18](#_Toc517253441)

[19. ADDIU 18](#_Toc517253442)

[1) 操作 18](#_Toc517253443)

[2) 数据通路图 18](#_Toc517253444)

[20. ANDI 18](#_Toc517253445)

[1) 操作 18](#_Toc517253446)

[2) 数据通路图 19](#_Toc517253447)

[21. ORI 19](#_Toc517253448)

[1) 操作 19](#_Toc517253449)

[2) 数据通路图 19](#_Toc517253450)

[22. XORI 19](#_Toc517253451)

[1) 操作 19](#_Toc517253452)

[2) 数据通路图 20](#_Toc517253453)

[23. LW 20](#_Toc517253454)

[1) 操作 20](#_Toc517253455)

[2) 数据通路图 20](#_Toc517253456)

[24. SW 20](#_Toc517253457)

[1) 操作 20](#_Toc517253458)

[2) 数据通路图 21](#_Toc517253459)

[25. BEQ 21](#_Toc517253460)

[1) 操作 21](#_Toc517253461)

[2) 数据通路图 21](#_Toc517253462)

[26. BNE 21](#_Toc517253463)

[1) 操作 21](#_Toc517253464)

[2) 数据通路图 22](#_Toc517253465)

[27. SLTI 22](#_Toc517253466)

[1) 操作 22](#_Toc517253467)

[2) 数据通路图 22](#_Toc517253468)

[28. SLTIU 22](#_Toc517253469)

[1) 操作 22](#_Toc517253470)

[2) 数据通路图 23](#_Toc517253471)

[29. LUI 23](#_Toc517253472)

[1) 操作 23](#_Toc517253473)

[2) 数据通路图 23](#_Toc517253474)

[30. J 23](#_Toc517253475)

[1) 操作 23](#_Toc517253476)

[2) 数据通路图 24](#_Toc517253477)

[31. JAL 24](#_Toc517253478)

[1) 操作 24](#_Toc517253479)

[2) 数据通路图 24](#_Toc517253480)

[32. DIV 24](#_Toc517253481)

[1) 操作 24](#_Toc517253482)

[2) 数据通路图 25](#_Toc517253483)

[33. DIVU 25](#_Toc517253484)

[1) 操作 25](#_Toc517253485)

[2) 数据通路图 25](#_Toc517253486)

[34. MULT 25](#_Toc517253487)

[1) 操作 25](#_Toc517253488)

[2) 数据通路图 26](#_Toc517253489)

[35. MULTU 26](#_Toc517253490)

[1) 操作 26](#_Toc517253491)

[2) 数据通路图 26](#_Toc517253492)

[36. BGEZ 26](#_Toc517253493)

[1) 操作 26](#_Toc517253494)

[2) 数据通路图 27](#_Toc517253495)

[37. JALR 27](#_Toc517253496)

[1) 操作 27](#_Toc517253497)

[2) 数据通路图 27](#_Toc517253498)

[38. LBU 27](#_Toc517253499)

[1) 操作 27](#_Toc517253500)

[2) 数据通路图 28](#_Toc517253501)

[39. LHU 28](#_Toc517253502)

[1) 操作 28](#_Toc517253503)

[2) 数据通路图 29](#_Toc517253504)

[40. LB 29](#_Toc517253505)

[1) 操作 29](#_Toc517253506)

[2) 数据通路图 29](#_Toc517253507)

[41. LH 30](#_Toc517253508)

[1) 操作 30](#_Toc517253509)

[2) 数据通路图 30](#_Toc517253510)

[42. SB 30](#_Toc517253511)

[1) 操作 30](#_Toc517253512)

[2) 数据通路图 31](#_Toc517253513)

[43. SH 31](#_Toc517253514)

[1) 操作 31](#_Toc517253515)

[2) 数据通路图 31](#_Toc517253516)

[44. BREAK 32](#_Toc517253517)

[1) 操作 32](#_Toc517253518)

[2) 数据通路图 32](#_Toc517253519)

[45. SYSCALL 32](#_Toc517253520)

[1) 操作 32](#_Toc517253521)

[2) 数据通路图 32](#_Toc517253522)

[46. ERET 32](#_Toc517253523)

[1) 操作 32](#_Toc517253524)

[2) 数据通路图 33](#_Toc517253525)

[47. MFHI 33](#_Toc517253526)

[1) 操作 33](#_Toc517253527)

[2) 数据通路图 33](#_Toc517253528)

[48. MFLO 33](#_Toc517253529)

[1) 操作 33](#_Toc517253530)

[2) 数据通路图 34](#_Toc517253531)

[49. MTHI 34](#_Toc517253532)

[1) 操作 34](#_Toc517253533)

[2) 数据通路图 34](#_Toc517253534)

[50. MTLO 34](#_Toc517253535)

[1) 操作 34](#_Toc517253536)

[2) 数据通路图 35](#_Toc517253537)

[51. MFC0 35](#_Toc517253538)

[1) 操作 35](#_Toc517253539)

[2) 数据通路图 35](#_Toc517253540)

[52. MTC0 35](#_Toc517253541)

[1) 操作 35](#_Toc517253542)

[2) 数据通路图 36](#_Toc517253543)

[53. CLZ 36](#_Toc517253544)

[1) 操作 36](#_Toc517253545)

[2) 数据通路图 36](#_Toc517253546)

[54. TEQ 36](#_Toc517253547)

[1) 操作 36](#_Toc517253548)

[2) 数据通路图 37](#_Toc517253549)

[55. 总体控制器表和数据通路 37](#_Toc517253550)

[1) 列表 37](#_Toc517253551)

[2) 数据通路 40](#_Toc517253552)

[56. CoProcessor0（CP0）模块 40](#_Toc517253553)

[57. Verilog HDL实现CPU 40](#_Toc517253554)

[58. Verilog HDL实现CPU与外部设备通信 41](#_Toc517253555)

[59. “电子公告牌”的设计与运行流程 41](#_Toc517253556)

[60. 主要代码 43](#_Toc517253557)

[四、 测试/调试过程 62](#_Toc517253558)

[五、 实验结果分析 64](#_Toc517253559)

[六、 结论 66](#_Toc517253560)

[七、 心得体会及建议 66](#_Toc517253561)

# 实验目标

* 使用 Verilog HDL 语言实现 54 条 MIPS 指令的 CPU 的设计和仿真。
* 用Vivado 综合器和实现器将上述设计生成二进制流文件（.bit），加载到Nexys 4 DDR 开发板上运行该CPU，执行老师给定的指令序列，得到正确的结果。
* 拓展该CPU的输入/输出功能，并自己编写可在该CPU上运行的MIPS指令，实现CPU的一个具体应用。本人实现的应用为“电子公告牌”。

# 总体设计

## 作品功能设计及原理说明

MIPS的所有指令的格式如下：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Opcode | Name | Action | Opcode bitfields | | | | | |
| **Arithmetic Logic Unit** | | | | | | | | |
| ADD rd,rs,rt | Add | rd=rs+rt | 000000 | rs | rt | rd | 00000 | 100000 |
| ADDI rt,rs,imm | Add Immediate | rt=rs+imm | 001000 | rs | rt | imm | | |
| ADDIU rt,rs,imm | Add Immediate Unsigned | rt=rs+imm | 001001 | rs | rt | imm | | |
| ADDU rd,rs,rt | Add Unsigned | rd=rs+rt | 000000 | rs | rt | rd | 00000 | 100001 |
| AND rd,rs,rt | And | rd=rs&rt | 000000 | rs | rt | rd | 00000 | 100100 |
| ANDI rt,rs,imm | And Immediate | rt=rs&imm | 001100 | rs | rt | imm | | |
| LUI rt,imm | Load Upper Immediate | rt=imm<<16 | 001111 | rs | rt | imm | | |
| NOR rd,rs,rt | Nor | rd=~(rs|rt) | 000000 | rs | rt | rd | 00000 | 100111 |
| OR rd,rs,rt | Or | rd=rs|rt | 000000 | rs | rt | rd | 00000 | 100101 |
| ORI rt,rs,imm | Or Immediate | rt=rs|imm | 001101 | rs | rt | imm | | |
| SLT rd,rs,rt | Set On Less Than | rd=rs<rt | 000000 | rs | rt | rd | 00000 | 101010 |
| SLTI rt,rs,imm | Set On Less Than Immediate | rt=rs<imm | 001010 | rs | rt | imm | | |
| SLTIU rt,rs,imm | Set On | rt=rs<imm | 001011 | rs | rt | imm | | |
| SLTU rd,rs,rt | Set On Less Than Unsigned | rd=rs<rt | 000000 | rs | rt | rd | 00000 | 101011 |
| SUB rd,rs,rt | Subtract | rd=rs-rt | 000000 | rs | rt | rd | 00000 | 100010 |
| SUBU rd,rs,rt | Subtract Unsigned | rd=rs-rt | 000000 | rs | rt | rd | 00000 | 100011 |
| XOR rd,rs,rt | Exclusive Or | rd=rs^rt | 000000 | rs | rt | rd | 00000 | 100110 |
| XORI rt,rs,imm | Exclusive Or Immediate | rt=rs^imm | 001110 | rs | rt | imm | | |
| **Shifter** | | | | | | | | |
| SLL rd,rt,sa | Shift Left Logical | rd=rt<<sa | 000000 | rs | rt | rd | sa | 000000 |
| SLLV rd,rt,rs | Shift Left Logical Variable | rd=rt<<rs | 000000 | rs | rt | rd | 00000 | 000100 |
| SRA rd,rt,sa | Shift Right Arithmetic | rd=rt>>sa | 000000 | 00000 | rt | rd | sa | 000011 |
| SRAV rd,rt,rs | Shift Right Arithmetic Variable | rd=rt>>rs | 000000 | rs | rt | rd | 00000 | 000111 |
| SRL rd,rt,sa | Shift Right Logical | rd=rt>>sa | 000000 | rs | rt | rd | sa | 000010 |
| SRLV rd,rt,rs | Shift Right Logical Variable | rd=rt>>rs | 000000 | rs | rt | rd | 00000 | 000110 |
| **Multiply** | | | | | | | | |
| DIV rs,rt | Divide | HI=rs%rt; LO=rs/rt | 000000 | rs | rt | 0000000000 | | 011010 |
| DIVU rs,rt | Divide Unsigned | HI=rs%rt; LO=rs/rt | 000000 | rs | rt | 0000000000 | | 011011 |
| MFHI rd | Move From HI | rd=HI | 000000 | 0000000000 | | rd | 00000 | 010000 |
| MFLO rd | Move From LO | rd=LO | 000000 | 0000000000 | | rd | 00000 | 010010 |
| MTHI rs | Move To HI | HI=rs | 000000 | rs | 000000000000000 | | | 010001 |
| MTLO rs | Move To LO | LO=rs | 000000 | rs | 000000000000000 | | | 010011 |
| MULT rs,rt | Multiply | HI,LO=rs\*rt | 000000 | rs | rt | 0000000000 | | 011000 |
| MULTU rs,rt | Multiply Unsigned | HI,LO=rs\*rt | 000000 | rs | rt | 0000000000 | | 011001 |
| **Branch** | | | | | | | | |
| BEQ rs,rt,offset | Branch On Equal | if(rs==rt) pc+=offset\*4 | 000100 | rs | rt | offset | | |
| BGEZ rs,offset | Branch On >= 0 | if(rs>=0) pc+=offset\*4 | 000001 | rs | 00001 | offset | | |
| BGEZAL rs,offset | Branch On >= 0 And Link | r31=pc; if(rs>=0) pc+=offset\*4 | 000001 | rs | 10001 | offset | | |
| BGTZ rs,offset | Branch On > 0 | if(rs>0) pc+=offset\*4 | 000111 | rs | 00000 | offset | | |
| BLEZ rs,offset | Branch On | if(rs<=0) pc+=offset\*4 | 000110 | rs | 00000 | offset | | |
| BLTZ rs,offset | Branch On | if(rs<0) pc+=offset\*4 | 000001 | rs | 00000 | offset | | |
| BLTZAL rs,offset | Branch On | r31=pc; if(rs<0) pc+=offset\*4 | 000001 | rs | 10000 | offset | | |
| BNE rs,rt,offset | Branch On Not Equal | if(rs!=rt) pc+=offset\*4 | 000101 | rs | rt | offset | | |
| BREAK | Breakpoint | epc=pc; pc=0x3c | 000000 | code | | | | 001101 |
| J target | Jump | pc=pc\_upper|(target<<2) | 000010 | target | | | | |
| JAL target | Jump And Link | r31=pc; pc=target<<2 | 000011 | target | | | | |
| JALR rs | Jump And Link Register | rd=pc; pc=rs | 000000 | rs | 00000 | rd | 00000 | 001001 |
| JR rs | Jump Register | pc=rs | 000000 | rs | 000000000000000 | | | 001000 |
| MFC0 rt,rd | Move From Coprocessor | rt=CPR[0,rd] | 010000 | 00000 | rt | rd | 00000000000 | |
| MTC0 rt,rd | Move To Coprocessor | CPR[0,rd]=rt | 010000 | 00100 | rt | rd | 00000000000 | |
| SYSCALL | System Call | epc=pc; pc=0x3c | 000000 | 00000000000000000000 | | | | 001100 |
| **Memory Access** | | | | | | | | |
| LB rt,offset(rs) | Load Byte | rt=\*(char\*)(offset+rs) | 100000 | rs | rt | offset | | |
| LBU rt,offset(rs) | Load Byte Unsigned | rt=\*(Uchar\*)(offset+rs) | 100100 | rs | rt | offset | | |
| LH rt,offset(rs) | Load Halfword | rt=\*(short\*)(offset+rs) | 100001 | rs | rt | offset | | |
| LBU rt,offset(rs) | Load Halfword Unsigned | rt=\*(Ushort\*)(offset+rs) | 100101 | rs | rt | offset | | |
| LW rt,offset(rs) | Load Word | rt=\*(int\*)(offset+rs) | 100011 | rs | rt | offset | | |
| SB rt,offset(rs) | Store Byte | \*(char\*)(offset+rs)=rt | 101000 | rs | rt | offset | | |
| SH rt,offset(rs) | Store Halfword | \*(short\*)(offset+rs)=rt | 101001 | rs | rt | offset | | |
| SW rt,offset(rs) | Store Word | \*(int\*)(offset+rs)=rt | 101011 | rs | rt | offset | | |

这些指令皆为较简单的指令，只要结合正确的时序电路和组合逻辑电路，在合理的允许延迟时间下，可以在一个时钟周期内完成。CPU的设计思路如下：

* 设计统一的指令译码器，将CPU指令译码为一条使能的指令信号以及操作数。
* 根据每条指令设计不同的电路，以组合逻辑的方式，给下一步时序逻辑的所有输入端加载正确的信号。所有可能以组合逻辑方式实现的微操作，都用组合逻辑实现。
* 时序逻辑部分，当时钟上升沿到来时，根据上一步加载的正确信号，将寄存器堆、数据存储器、PC、HI、LO寄存器内的数据进行刷新。

而关于CPU的输入输出功能，主要用syscall引起的中断，以及执行系统调用的模块kernel进行：

* 指令译码器译出syscall指令，kernel使能信号加载高电平，同时组合逻辑电路调出 $v0寄存器（记录要调用的系统功能）送kernel模块。
* Kernel接受到使能信号和系统功能号，进入工作状态，其输出信号knWorking置1。knWorking信号回传到CPU，CPU进入暂停状态，不进行PC、寄存器堆、数据内存的刷新。
* Kernel向CPU的寄存器堆读取调用参数，主要存储在 $v1、$a0、$a1、$a2中。
* Kernel根据传来的参数执行这个系统功能的数字逻辑部分，如刷新显存、让数码管输出特定数字、接受用户输入（通过开关或按钮）等。
* 执行完毕后，knWorking置0，允许CPU继续运行。

# 主要模块（指令）设计

## ADD

### 操作流程（不用流程图表示，用列表表示）

* 取指令
* temp←GPR[rs]+GPR[rt]（判断上下溢出）
* 若溢出
  + 异常信号
* 否则
  + GPR[rd]←temp
* PC←PC+4

### 数据通路图



## ADDU

### 操作

* 取指令
* temp←GPR[rs]+GPR[rt]（判断上下溢出）
* GPR[rd]←temp
* PC←PC+4

### 数据通路图



## SUB

### 操作

* 取指令
* temp←GPR[rs]-GPR[rt]（判断上下溢出）
* 若溢出
  + 异常信号
* 否则
  + GPR[rd]←temp
* PC←PC+4

### 数据通路图



## SUBU

### 操作

* 取指令
* temp←GPR[rs]-GPR[rt]（不判断上下溢出）
* GPR[rd]←temp
* PC←PC+4

### 数据通路图



## AND

### 操作

* 取指令
* GPR[rd]←GPR[rs]|GPR[rt]
* PC←PC+4

### 数据通路图



## OR

### 操作

* 取指令
* GPR[rd]←GPR[rs]&GPR[rt]
* PC←PC+4

### 数据通路图



## XOR

### 操作

* 取指令
* GPR[rd]←GPR[rs]^GPR[rt]
* PC←PC+4

### 数据通路图



## NOR

### 操作

* 取指令
* GPR[rd]←GPR[rs]**↓**GPR[rt]
* PC←PC+4

### 数据通路图



## SLT

### 操作

* 取指令
* GPR[rd]←((signed)GPR[rs]<(signed)GPR[rt])?32'b1:32'b0
* PC←PC+4

### 数据通路图



## SLTU

### 操作

* 取指令
* GPR[rd]←((unsigned)GPR[rs]<(unsigned)GPR[rt])?32'b1:32'b0
* PC←PC+4

### 数据通路图



## SLL

### 操作

* 取指令
* s←sa
* GPR[rd]←{GPR[rt][31-s:0], (s){1'b0}}
* PC←PC+4

### 数据通路图



## SRL

### 操作

* 取指令
* s←sa
* GPR[rd]←{(s){1'b0}, GPR[rt][31:s]}
* PC←PC+4

### 数据通路图



## SRA

### 操作

* 取指令
* s←sa
* GPR[rd]←{(s){GPR[rt][31]}, GPR[rt][31:s]}
* PC←PC+4

### 数据通路图



## SLLV

### 操作

* 取指令
* s←GPR[rs][4:0]
* GPR[rd]←{GPR[rt][31-s:0], (s){1'b0}}
* PC←PC+4

### 数据通路图



## SRLV

### 操作

* 取指令
* s←GPR[rs][4:0]
* GPR[rd]←{(s){1'b0}, GPR[rt][31:s]}
* PC←PC+4

### 数据通路图



## SRAV

### 操作

* 取指令
* s←GPR[rs][4:0]
* GPR[rd]←{(s){GPR[rt][31]}, GPR[rt][31:s]}
* PC←PC+4

### 数据通路图



## JR

### 操作

* 取指令
* PC←GPR[rs]

### 数据通路图



## ADDI

### 操作

* 取指令
* signed\_imm←{{(16){immediate[15]}},immediate}
* temp←GPR[rs]+ signed\_imm（带符号，判断上下溢出）
* 若溢出
  + 异常信号
* 否则
  + GPR[rt]←temp
* PC←PC+4

### 数据通路图



## ADDIU

### 操作

* 取指令
* signed\_imm←{{(16){immediate[15]}},immediate}
* temp←GPR[rs]+ signed\_imm（无符号，不判断上下溢出）
* GPR[rt]←temp
* PC←PC+4

### 数据通路图



## ANDI

### 操作

* 取指令
* GPR[rt]←GPR[rs]&{16'b0, immediate}
* PC←PC+4

### 数据通路图



## ORI

### 操作

* 取指令
* GPR[rt]←GPR[rs]|{16'b0, immediate}
* PC←PC+4

### 数据通路图



## XORI

### 操作

* 取指令
* GPR[rt]←GPR[rs]^{16'b0, immediate}
* PC←PC+4

### 数据通路图



## LW

### 操作

* 取指令
* vAddr←{(16){offset[15]}, offset}+GPR[base] //虚拟地址
* pAddr←AddressTranslation(vAddr) //物理地址
* memword←LoadMemoryWord(pAddr)
* GPR[rt]←memword
* PC←PC+4

### 数据通路图



## SW

### 操作

* 取指令
* vAddr←{(16){offset[15]}, offset}+GPR[base] //虚拟地址
* pAddr←AddressTranslation(vAddr) //物理地址
* dataword←GPR[rt]
* StoreMemory(pAddr, dataword)
* PC←PC+4

### 数据通路图



## BEQ

### 操作

* 取指令
* target\_offset←{(14){offset[15],offset,2'b0}
* if(GPR[rs]==GPR[rt])
  + PC←PC+target\_offset
* else
  + PC←PC+4

### 数据通路图



## BNE

### 操作

* 取指令
* target\_offset←{(14){offset[15],offset,2'b0}
* if(!(GPR[rs]==GPR[rt]))
  + PC←PC+target\_offset
* else
  + PC←PC+4

### 数据通路图



## SLTI

### 操作

* 取指令
* GPR[rt]←((signed)GPR[rs]<(signed)({(16{immediate[15]}, immediate}))?32'b1:32'b0
* PC←PC+4

### 数据通路图



## SLTIU

### 操作

* 取指令
* GPR[rt]←((unsigned)GPR[rs]<(unsigned)({(16{immediate[15]}, immediate}))?32'b1:32'b0
* PC←PC+4

### 数据通路图



## LUI

### 操作

* 取指令
* GPR[rt]←{immediate,16'b0}
* PC←PC+4

### 数据通路图



## J

### 操作

* 取指令
* PC←{PC[31:28], instr\_index, 2'b0}

### 数据通路图



## JAL

### 操作

* 取指令
* GPR[31]←PC+4（没有延迟槽√）或PC+8（有延迟槽）
* PC←{PC[31:28], instr\_index, 2'b0}

### 数据通路图



## DIV

### 操作

* 取指令
* quotient←GPR[rs] / GPR[rt]（32位带符号）
* LO←quotient
* remainder←GPR[rs] % GPR[rt]（32位带符号）
* HI←remainder
* PC←PC+4

### 数据通路图



## DIVU

### 操作

* 取指令
* quotient←GPR[rs] / GPR[rt]（32位无符号）
* LO←quotient
* remainder←GPR[rs] % GPR[rt]（32位无符号）
* HI←remainder
* PC←PC+4

### 数据通路图



## MULT

### 操作

* 取指令
* product←GPR[rs] \* GPR[rt]（32位有符号）
* LO←product[31:0]
* HI←product[63:32]
* PC←PC+4

### 数据通路图



## MULTU

### 操作

* 取指令
* product←GPR[rs] \* GPR[rt]（32位无符号）
* LO←product[31:0]
* HI←product[63:32]
* PC←PC+4

### 数据通路图



## BGEZ

### 操作

* 取指令
* 如果 GPR[rs]>=0
  + PC←PC+{16{offset[15]}, offset}
* 否则
  + PC←PC+4

### 数据通路图



## JALR

### 操作

* 取指令
* temp←GPR[rs]
* GPR[rd]←PC+4（不考虑延迟槽）
* PC←temp

### 数据通路图



## LBU

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base]
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^2{ReverseEndian}}
* memword←LoadMemoryWord(pAddr)
* byte←vAddr[1:0]^{2{BigEndianCPU}}
* membyte←memword[8\*byte+:8]
* GPR[rt]←{24'b0,membyte}
* PC←PC+4

### 数据通路图



## LHU

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base] //(必须是2字节对齐的，即vAddr末位为0)
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^{ReverseEndian,0}}
* memword←LoadMemoryWord(pAddr)
* byte←vAddr[1:0]^{BigEndianCPU,0}
* membyte←memword[8\*byte+:16]
* GPR[rt]←{24'b0,membyte}
* PC←PC+4

### 数据通路图



## LB

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base]
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^2{ReverseEndian}}
* memword←LoadMemoryWord(pAddr)
* byte←vAddr[1:0]^{2{BigEndianCPU}}
* membyte←memword[8\*byte+:8]
* GPR[rt]←{24{membyte[8]},membyte}
* PC←PC+4

### 数据通路图



## LH

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base] //(必须是2字节对齐的，即vAddr末位为0)
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^{ReverseEndian,0}}
* memword←LoadMemoryWord(pAddr)
* byte←vAddr[1:0]^{BigEndianCPU,0}
* membyte←memword[8\*byte+:16]
* GPR[rt]←{16{membyte[15]},membyte}
* PC←PC+4

### 数据通路图



## SB

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base]
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^{ReverseEndian,0}}
* StoreMemoryByte(pAddr, GPR[rt][7:0]) //可能需要将GPR[rt]的低字节根据写入位置调整位置，补充0形成一个字再送内存
* PC←PC+4

### 数据通路图



## SH

### 操作

* 取指令
* ext\_offset←{16{offset[15]}, offset}
* vAddr←ext\_offset+GPR[base] //(必须是2字节对齐的，即vAddr末位为0)
* pAddr←AddressTranslation(vAddr)
* //pAddr←{pAddr[31:2], pAddr[1:0]^{ReverseEndian,0}}
* StoreMemoryHalfWord(pAddr, GPR[rt][15:0]) //可能需要将GPR[rt]的低半字根据写入位置调整位置，补充0形成一个字再送内存
* PC←PC+4

### 数据通路图



## BREAK

### 操作

* 取指令
* 触发BREAK异常

### 数据通路图



## SYSCALL

### 操作

* 取指令
* 触发SYSCALL异常

### 数据通路图



## ERET

### 操作

* 取指令
* 调用返回

### 数据通路图



## MFHI

### 操作

* 取指令
* GPR[rd]←HI
* PC←PC+4

### 数据通路图



## MFLO

### 操作

* 取指令
* GPR[rd]←LO
* PC←PC+4

### 数据通路图



## MTHI

### 操作

* 取指令
* HI←GPR[rs]
* PC←PC+4

### 数据通路图



## MTLO

### 操作

* 取指令
* LO←GPR[rs]
* PC←PC+4

### 数据通路图



## MFC0

### 操作

* 取指令
* GPR[rt]←CPR[0,rd]
* PC←PC+4

### 数据通路图



## MTC0

### 操作

* 取指令
* CPR[0,rd]←GPR[rt]
* PC←PC+4

### 数据通路图



## CLZ

### 操作

* 取指令
* GPR[rd]←count\_leading\_zeros GPR[rs]
* PC←PC+4

### 数据通路图



## TEQ

### 操作

* 取指令
* 如果 GPR[rt]==GPR[rs]
  + 触发TRAP异常
* 否则
  + PC←PC+4

### 数据通路图



## 总体控制器表和数据通路

### 列表

| **指令** | **PC** | **IM** | **RF和HI/LO** | **ALU** | | | **EXPT** | **EXTEND** | **EXTEND2** | **DMEM** | **ADDRMAP** | **MUX** | **UADD** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **WData(默认rd)** | **A** | **OP** | **B** | **A** | **B** |
| ADD | PC+4 | PC | ALU | GPR[rs] | S+ | GPR[rt] | ALUOVF? |  |  |  |  |  |  |  |
| ADDU | PC+4 | PC | ALU | GPR[rs] | U+ | GPR[rt] | 0 |  |  |  |  |  |  |  |
| SUB | PC+4 | PC | ALU | GPR[rs] | S- | GPR[rt] | ALUOVF? |  |  |  |  |  |  |  |
| SUBU | PC+4 | PC | ALU | GPR[rs] | U- | GPR[rt] | 0 |  |  |  |  |  |  |  |
| AND | PC+4 | PC | ALU | GPR[rs] | & | GPR[rt] | 0 |  |  |  |  |  |  |  |
| OR | PC+4 | PC | ALU | GPR[rs] | | | GPR[rt] | 0 |  |  |  |  |  |  |  |
| XOR | PC+4 | PC | ALU | GPR[rs] | ^ | GPR[rt] | 0 |  |  |  |  |  |  |  |
| NOR | PC+4 | PC | ALU | GPR[rs] | **↓** | GPR[rt] | 0 |  |  |  |  |  |  |  |
| SLT | PC+4 | PC | ALU | GPR[rs] | S< | GPR[rt] | 0 |  |  |  |  |  |  |  |
| SLTU | PC+4 | PC | ALU | GPR[rs] | U< | GPR[rt] | 0 |  |  |  |  |  |  |  |
| SLL | PC+4 | PC | ALU | GPR[rt] | << | sa | 0 |  |  |  |  |  |  |  |
| SRL | PC+4 | PC | ALU | GPR[rt] | >> | sa | 0 |  |  |  |  |  |  |  |
| SRA | PC+4 | PC | ALU | GPR[rt] | >>> | sa | 0 |  |  |  |  |  |  |  |
| SLLV | PC+4 | PC | ALU | GPR[rt] | << | GPR[rs] | 0 |  |  |  |  |  |  |  |
| SRLV | PC+4 | PC | ALU | GPR[rt] | >> | GPR[rs] | 0 |  |  |  |  |  |  |  |
| SRAV | PC+4 | PC | ALU | GPR[rt] | >>> | GPR[rs] | 0 |  |  |  |  |  |  |  |
| JR | GPR[rs] | PC | - | - | - | - | GPR[rs] [1:0]==2'b0 |  |  |  |  |  |  |  |
| ADDI | PC+4 | PC | ALU (→GPR[rt]) | GPR[rs] | S+ | EXTEND | ALUOVF? | IM[15:0] (SIGN) |  |  |  |  |  |  |
| ADDIU | PC+4 | PC | ALU (→GPR[rt]) | GPR[rs] | U+ | EXTEND | 0 | IM[15:0] (SIGN) |  |  |  |  |  |  |
| ANDI | PC+4 | PC | ALU (→GPR[rt]) | GPR[rs] | & | EXTEND | 0 | IM[15:0] (ZERO) |  |  |  |  |  |  |
| ORI | PC+4 | PC | ALU (→GPR[rt]) | GPR[rs] | | | EXTEND | 0 | IM[15:0] (ZERO) |  |  |  |  |  |  |
| XORI | PC+4 | PC | ALU (→GPR[rt]) | GPR[rs] | ^ | EXTEND | 0 | IM[15:0] (ZERO) |  |  |  |  |  |  |
| LW | PC+4 | PC | DMEM (→GPR[rt]) | GPR[base] | U+ | EXTEND | 0 | IM[15:0]  (SIGN) |  | ADDRMAP, R | ALU |  |  |  |
| SW | PC+4 | PC |  | GPR[base] | U+ | EXTEND | 0 | IM[15:0]  (SIGN) |  | ADDRMAP, GPR[rt], W | ALU |  |  |  |
| BEQ | MUX[ALU[0]] | PC |  | GPR[rs] | == | GPR[rt] | 0 | IM[15:0]  (SIGN) |  |  |  | PC+4,  UADD | PC | {EXTEND [29:0],2'b0} |
| BNE | MUX[~ALU[0]] | PC |  | GPR[rs] | == | GPR[rt] | 0 | IM[15:0]  (SIGN) |  |  |  | PC+4,  UADD | PC | {EXTEND [29:0],2'b0} |
| SLTI | PC+4 | PC | ALU  (→GPR[rt]) | GPR[rs] | S< | EXTEND | 0 | IM[15:0]  (SIGN) |  |  |  |  |  |  |
| SLTIU | PC+4 | PC | ALU  (→GPR[rt]) | GPR[rs] | S< | EXTEND | 0 | IM[15:0]  (SIGN) |  |  |  |  |  |  |
| LUI | PC+4 | PC | {immediate, 16'b0}  (→GPR[rt]) |  |  |  | 0 |  |  |  |  |  |  |  |
| J | {PC[31:28], instr\_index, 2'b0} | PC |  |  |  |  | 0 |  |  |  |  |  |  |  |
| JAL | {PC[31:28], instr\_index, 2'b0} | PC | PC+4  →GPR[31] |  |  |  | 0 |  |  |  |  |  |  |  |
| DIV | PC+4 | PC | ALU(除法)  LO←quotient  HI← remainder | GPR[rs] | S/ | GPR[rt] | 0 |  |  |  |  |  |  |  |
| DIVU | PC+4 | PC | ALU(除法)  LO←quotient  HI← remainder | GPR[rs] | U/ | GPR[rt] | 0 |  |  |  |  |  |  |  |
| MULT | PC+4 | PC | ALU(乘法)  {HI, LO}←remainder | GPR[rs] | S\* | GPR[rt] | 0 |  |  |  |  |  |  |  |
| MULTU | PC+4 | PC | ALU(乘法)  {HI, LO}←remainder | GPR[rs] | U\* | GPR[rt] | 0 |  |  |  |  |  |  |  |
| BGEZ | MUX[GPR[rs][31]] | PC |  |  |  |  | 0 | IM[15:0](offset)  (SIGN) |  |  |  | UADD,PC+4 | PC | EXTEND[29:0],2'b0 |
| JALR | GPR[rs] | PC | PC+8 |  |  |  | 0 |  |  |  |  |  |  |  |
| LBU | PC+4 | PC | EXTEND2→GRP[rt] |  |  |  | 0 | IM[15:0](offset)  (SIGN) | DMEM[byte\*8+:8]  (ZERO) | ADDRMAP,R | UADD |  | EXTEND | GPR[base] |
| LHU | PC+4 | PC | EXTEND2→GRP[rt] |  |  |  | 0 | IM[15:0](offset)  (SIGN) | DMEM[byte\*8+:16]  (ZERO) | ADDRMAP,R | UADD |  | EXTEND | GPR[base] |
| LB | PC+4 | PC | EXTEND2→GRP[rt] |  |  |  | 0 | IM[15:0](offset)  (SIGN) | DMEM[byte\*8+:8]  (SIGN) | ADDRMAP,R | UADD |  | EXTEND | GPR[base] |
| LH | PC+4 | PC | EXTEND2→GRP[rt] |  |  |  | 0 | IM[15:0](offset)  (SIGN) | DMEM[byte\*8+:16]  (SIGN) | ADDRMAP,R | UADD |  | EXTEND | GPR[base] |
| SB | PC+4 | PC |  |  |  |  | 0 | IM[15:0](offset)  (SIGN) |  | ADDRMAP,GPR[rt][7:0],W | UADD |  | EXTEND | GPR[base] |
| SH | PC+4 | PC |  |  |  |  | 0 | IM[15:0](offset)  (SIGN) |  | ADDRMAP,GPR[rt][15:0],W | UADD |  | EXTEND | GPR[base] |
| BREAK | BREAK←1，cause[6:2]←4'b1001  exec\_addr←PC | | | | | | | | | | | | | |
| SYSCALL | SYSCALL←1，cause[6:2]←4'b1000  exec\_addr←PC | | | | | | | | | | | | | |
| ERET | ERET←1, status←status>>5  PC←EPC\_out | | | | | | | | | | | | | |
| MFHI | PC+4 | PC | HI→GRP[rd] |  |  |  | 0 |  |  |  |  |  |  |  |
| MFLO | PC+4 | PC | LO→GRP[rd] |  |  |  | 0 |  |  |  |  |  |  |  |
| MTHI | PC+4 | PC | HI←GPR[rs] | | | | | | | | | | | |
| MTLO | PC+4 | PC | LO←GPR[rs] | | | | | | | | | | | |
| MFC0 | PC+4 | PC | cp0addr←rd  mfc0←1  GPR[rt]←cp0rdata | | | | | | | | | | | |
| MTC0 | PC+4 | PC | cp0addr←rd  mtc0←1  cp0data←GPR[rt] | | | | | | | | | | | |
| CLZ | PC+4 | PC | ALU→GRP[rd] | GPR[rs] | CLZ |  | 0 |  |  |  |  |  |  |  |
| TEQ | PC+4 | PC |  | GPR[rs] | == | GPR[rt] | TRAP←ALU[0]，cause[6:2]←ALU[0]?cause[6:2]  exec\_addr←ALU[0]?PC:exec\_addr | | | | | | | |

### 数据通路



## CoProcessor0（CP0）模块

CP0模块在本实验中辅助实现CPU的异常处理与返回的效能。其实现较为简单，只需要在模块内实现三个CP0寄存器cause\_reg、status\_reg、epc\_reg，并在有MFC0指令和MTC0指令时做适当的读写操作、在遇到异常命令SYSCALL、BREAK、TEQ且允许这个异常发生的情况下，操作三个内部寄存器写入异常信息；遇到返回命令ERET时，将epc\_reg的值输出，方便pc接受。

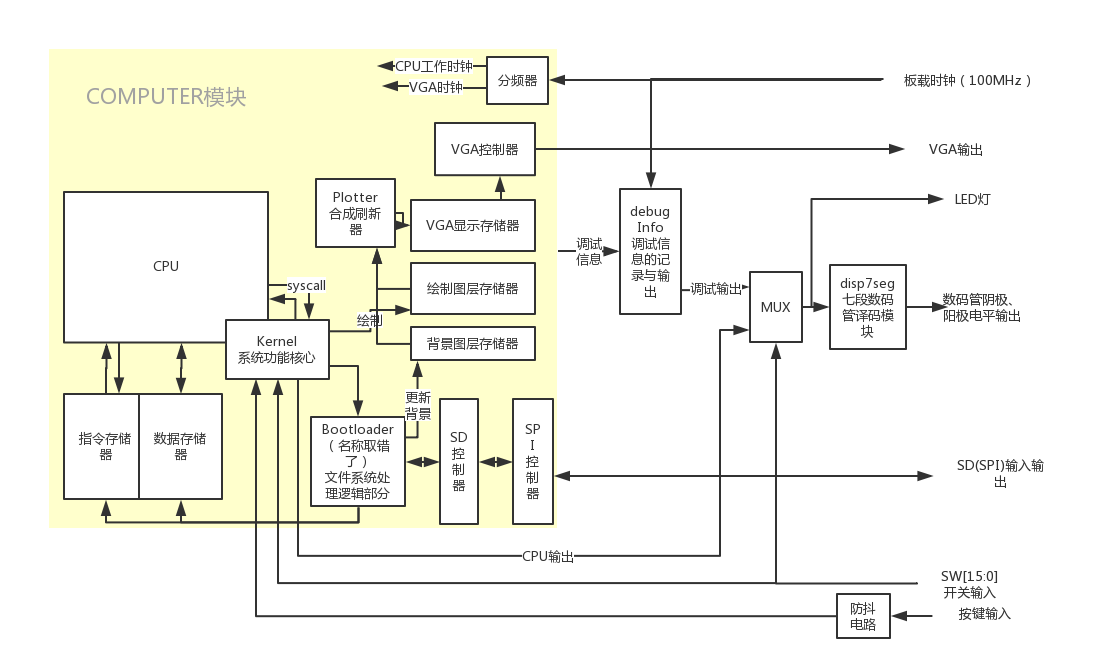
## Verilog HDL实现CPU

由于Verilog HDL是较为高级的硬件描述语言，可以使用行为级描述而让综合器生成正确的元器件，故实现CPU时：

* + - 指令译码器采用了always @\* 块，用组合逻辑的方式给每一个指令信号赋值；
    - 执行指令有关电路也采用了always @\* 组合逻辑块；
    - 刷新Regfile、PC、内存的电路用always @(posedge clk)的时序电路同步实现；
    - 数据内存和指令内存用IP核（Distributed Memory Generator和Block Memory Generator）实现。
    - 分频器用Clocking Wizard实现，生成CPU的工作时钟15MHz。

## Verilog HDL实现CPU与外部设备通信

具体逻辑层次如图所示。（其中部分数据通路未画出）



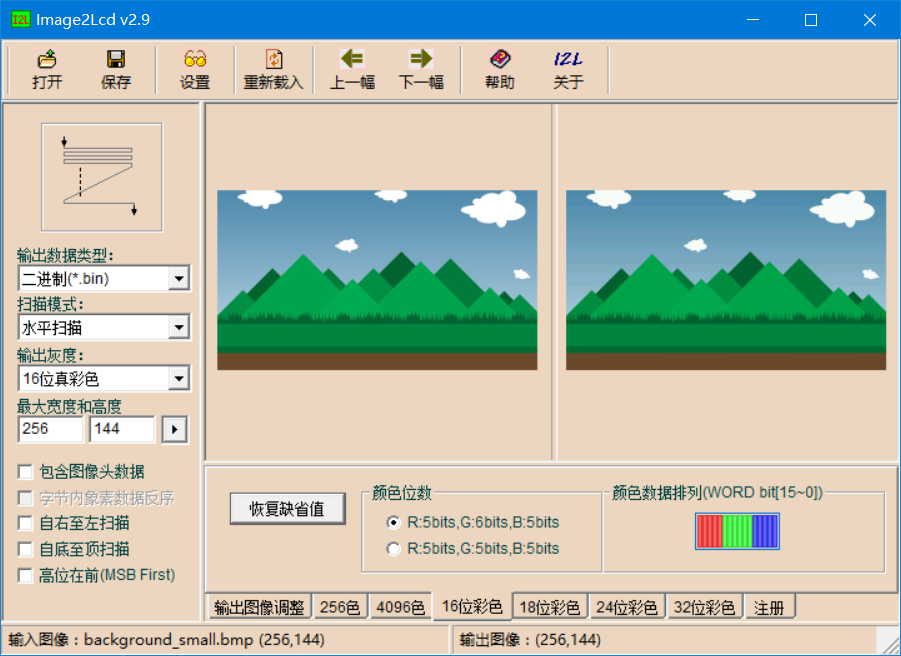
## “电子公告牌”的设计与运行流程

“电子公告牌”是一个基于FPGA实现的54 MIPS指令CPU、TF(Micro SD)卡控制器、VGA控制器的具体应用，可以读取用户存储在TF卡根目录下某一特定文件（NOTE0000.TXT）的内容并将其输出在VGA显示器上。

其设计思路是：CPU命令上述的Bootloader模块（FAT文件系统模块）读取SD卡根目录下的NOTE0000.TXT文件的内容，加载至数据存储器（Data MEMory），从而CPU可以对DMEM中的字符进行解析，计算GB2312字符的区位码。紧接着，CPU命令Bootloader模块读取SD卡根目录下的HZK16.DAT该区位码指定的部分，将8个32位字读入DMEM。这8个32位字中存储了这个GB2312字符的字形点阵。此后，CPU对这些32位字进行位操作，并适时通过Kernel向图层存储器传达“绘制像素”命令，从而将所有的GB2312字符绘制在图层中，经刷新到VGA显存，可以在显示器上显示该字符。

用户和系统运行该“电子公告牌”应用程序的流程如下：

1. 用户
   * + 1. 准备一张Micro SDHC卡，并将其以FDD或USB-HDD（务必保证要读取的FAT32分区是分区表内的第一个分区）方式格式化为FAT32文件系统格式。
       2. 用电脑在SD卡分区的根目录下放入本应用程序随附的apoclyps.bin（二进制的MIPS指令，可以查看apoclyps.asm内的源码）、HZK16.DAT（汉字字库文件）。
       3. 以GB2312编码将要写入的公告文本写入NOTEXXXX.TXT文件中，放入SD卡分区根目录。
       4. 如果需要切换公告牌背景，可以将256x144的BMP图像用Img2Lcd转换为BIN格式，重命名为BACKXXXX.IMG（将XXXX替换为十进制4位背景的编号，允许从0000到4095），放入SD卡分区根目录中。Img2Lcd的转换配置如图所示。本程序随附了两个背景图片（BACK0001.IMG和BACK0002.IMG）。
       5. 如果需要切换公告内容，可以将所需公告文件重命名为NOTEXXXX.IMG（将XXXX替换为十进制4位编号，允许从0000到4095），放入SD卡分区根目录中。



* + - 1. 将SD卡弹出，插入FPGA Nexys 4 DDR开发板。给开发板接上VGA显示器和电源。
      2. 将该应用的二进制比特流文件topmodule.bit下载到板上。
      3. 确保所有的开关是关闭的；按“中”键。
      4. 等待显示加载完毕。也可以拨开0号开关（最右），等待七段数码管显示的PC达到00400380。此时应该可以看到屏幕上显示背景图片和公告文字。0号开关关闭时，七段数码管显示F0000000，其中第2~4位000表示当前的文字颜色是红0绿0蓝0（黑色），第1、5~8位0000表示当前加载0号公告文件（File）。
      5. 如果需要切换显示文本的颜色，将颜色表示为12位二进制的红、绿、蓝颜色分量（各种原色各4位）。将红、绿、蓝分量的二进制在开关的右12位打出。紧接着，将从左数起的第3、4个开关打开，并确保左1、2开关关闭。例如，要更换为红0xf，绿0x8，蓝0x0的黄色，开关从左到右设置为：00,11,1111,1000,0000。然后按一下“中键”，并等待显示完毕。
      6. 如果需要切换背景图片，将图片编号表示为12位的二进制数码，并在开关的右12位输入。将左起第1、2个开关打开，3、4个开关关闭，按“中”键。背景图片应该立刻切换完成。如果没有切换并且15号LED灯（最左边）亮起，那么这一过程发生错误，很有可能SD卡中找不到以该编号命名的backxxxx.img。
      7. 如果需要切换公告文件，将文件编号表示为12位的二进制数码，并在开关的右12位输入。将左起第2、3个开关打开，1、4个开关关闭，按“中”键，应该会开始切换公告文件。如果没有切换并且15号LED灯（最左边）亮起，那么这一过程发生错误，很有可能SD卡中找不到以该编号命名的notexxxx.txt。
      8. 0号开关在任何情况下都是调试模式开启的开关。当0号开关开启时，按“上”键，即会执行复位操作。
      9. 15号LED灯在任何情况下都是执行错误的指示灯。当它亮起时，检查之前的操作是否有错，并按照k.中的方法复位程序后重新开始。
      10. 当调试模式关闭时，七段数码管正常输出颜色和文件编号的状态；当调试模式开启时，七段数码管会根据5~3号开关（选择输出哪一种调试信息）和15~8号开关（选择输出内容的索引）进行调试信息的输出。此处略去，请参考代码debugInfo.v。

1. 系统
   * + 1. 所有模块重置，PC指向指令存储器的第一条指令。
       2. 用户按“中”键。
       3. Bootloader接受到信号，自动从SD卡加载apoclyps.bin文件到指令存储器。
       4. CPU开始工作，以下为汇编指令的流程。
       5. 执行一syscall，让七段数码管正确显示颜色、背景图片编号状态。
       6. 初始化寄存器和记录文件名的内存数据等。
       7. 执行一syscall，让绘制图层清空绘制内容。
       8. 执行一syscall，从SD卡加载NOTE0000.TXT到数据存储器。
       9. 循环：每次从数据存储器加载一字节，分析其代表的字符。当凑齐可以当作一个中文字符的两个字节时，计算其区码和位码，并以此计算出该字字形点阵在HZK16.DAT中存储的字偏移。
       10. 执行一syscall，从HZK16.DAT的这一偏移地址开始，读取8个字（256bit）到数据存储器。
       11. 循环：每次从字符点阵中取出一bit，并根据bit为1或为0向绘制图层的指定位置描点，直到完成16\*16=256bit的字符点阵的绘制。
       12. 返回到分析字节的循环，读取下面的字符。
       13. 当所有字符输出完毕，等待输入。
       14. 接收到输入时，分析输入，并更新颜色（跳回f.重新绘制）或背景（只需重写背景图层）。
       15. 回到m.。

## 主要代码

Computer.v的内容：

|  |
| --- |
| `timescale 1ns/1ns  `include "sdHeader.vh"  module computer(  input clk\_in,  input reset,  input compStartEn,  output reg compStartOk,  input btnu,  input btnc,  input btnd,  input btnl,  input btnr,  input btnu\_orig,  input btnc\_orig,  input btnd\_orig,  input btnl\_orig,  input btnr\_orig,  input [15:0] SW,  input [12:0] debugDMEMAddr,  input [12:0] debugIMEMAddr,  input [4:0] debugRFAddr,  output clk\_cpu,  output [31:0] inst,  output [31:0] pc,  output [31:0] addr,  output [7:0] blState,  output [7:0] sdState,  output [4:0] spiState,  output [31:0] debugInfo,  output debugInfoAvailable,  output cpuRunning,  output cpuPaused,  output sdError,  output blError,  output [31:0] debugDMEMData,  output [31:0] debugIMEMData,  output [31:0] debugRFData,  output [3:0] VGA\_R,  output [3:0] VGA\_G,  output [3:0] VGA\_B,  output VGA\_HS,  output VGA\_VS,  output SPI\_CLK,  output SPI\_MOSI,  input SPI\_MISO,  output SPI\_CSn,  output SD\_RESET,  output [31:0] sevenSegOut  );  wire clk\_vga;  //wire clk\_cpu\_orig;  //assign clk\_cpu = SW[2] ? btnd\_orig : clk\_cpu\_orig;  assign clk\_cpu = clk\_cpu\_orig;  wire debugInfoAvailable\_bl;  wire debugInfoAvailable\_sd;  wire [31:0] debugInfo\_bl;  wire [63:0] debugInfo\_sd;  assign debugInfoAvailable = debugInfoAvailable\_sd | debugInfoAvailable\_bl;  assign debugInfo = debugInfoAvailable\_sd ? debugInfo\_sd[31:0] : debugInfoAvailable\_bl ? debugInfo\_bl : 32'hFFFFFFFF;  //////////////////  /// Clock generator    clk\_generator clkgen\_inst(  .clk\_100MHz(clk\_in),  .clk\_vga(clk\_vga),  .clk\_cpu(clk\_cpu\_orig)  );  //////////////////  /// VGA  wire [10:0] x;  wire [9:0] y;  wire [10:0] xNext;  wire [9:0] yNext;  wire [17:0] vgaMemXYAddr = {yNext[9:1], xNext[9:1]};  wire inplace;  reg [3:0] VGA\_R\_r;  reg [3:0] VGA\_G\_r;  reg [3:0] VGA\_B\_r;  wire [11:0] vgaMemWord;  wire vgaMemWe;  wire [17:0] vgaMemAddr;  wire [11:0] vgaMemDina;    vga vga\_inst  (  .clk(clk\_vga),  .rst(reset),  .hsync(VGA\_HS),  .vsync(VGA\_VS),  .x(x),  .y(y),  .xNext(xNext),  .yNext(yNext),  .inplace(inplace)  );  vga\_mem vga\_mem\_inst (  .clka(clk\_cpu), // input wire clka  .wea(vgaMemWe), // input wire [0 : 0] wea  .addra(vgaMemAddr), // input wire [17 : 0] addra  .dina(vgaMemDina), // input wire [11 : 0] dina  .clkb(clk\_vga), // input wire clkb  .addrb(vgaMemXYAddr), // input wire [17 : 0] addrb  .doutb(vgaMemWord) // output wire [11 : 0] doutb  );  wire backgroundMemWea;  wire [15:0] backgroundMemAddra;  wire [11:0] backgroundMemDina;  wire [15:0] backgroundMemAddrb;  wire [11:0] backgroundMemDoutb;  background\_mem background\_mem\_inst (  .clka(clk\_cpu), // input wire clka  .wea(backgroundMemWea), // input wire [0 : 0] wea  .addra(backgroundMemAddra), // input wire [15 : 0] addra  .dina(backgroundMemDina), // input wire [11 : 0] dina  .clkb(clk\_cpu), // input wire clkb  .addrb(backgroundMemAddrb), // input wire [15 : 0] addrb  .doutb(backgroundMemDoutb) // output wire [11 : 0] doutb  );  wire canvasMemWea;  wire [17:0] canvasMemAddra;  wire [12:0] canvasMemDina;  wire [17:0] canvasMemAddrb;  wire [12:0] canvasMemDoutb;  canvas\_mem canvas\_mem\_inst (  .clka(clk\_cpu), // input wire clka  .wea(canvasMemWea), // input wire [0 : 0] wea  .addra(canvasMemAddra), // input wire [17 : 0] addra  .dina(canvasMemDina), // input wire [12 : 0] dina  .clkb(clk\_cpu), // input wire clkb  .addrb(canvasMemAddrb), // input wire [17 : 0] addrb  .doutb(canvasMemDoutb) // output wire [12 : 0] doutb  );  plotter plotter\_inst(  .clk(clk\_cpu),  .reset(reset),  .backgroundMemAddrb(backgroundMemAddrb),  .backgroundMemDoutb(backgroundMemDoutb),  .canvasMemAddrb(canvasMemAddrb),  .canvasMemDoutb(canvasMemDoutb),  .vgaMemWe(vgaMemWe),  .vgaMemAddr(vgaMemAddr),  .vgaMemDina(vgaMemDina)  );  always @(\*)  begin  VGA\_R\_r = vgaMemWord[11:8];  VGA\_G\_r = vgaMemWord[7:4];  VGA\_B\_r = vgaMemWord[3:0];  end    assign VGA\_R = inplace ? VGA\_R\_r : 4'hz;  assign VGA\_G = inplace ? VGA\_G\_r : 4'hz;  assign VGA\_B = inplace ? VGA\_B\_r : 4'hz;  ////////////////  /// DMEM  /// Port A: Work at falling edge of clk.  /// Port B: Work at rising edge of clk.  wire knWorking;  wire knInitOk;  wire dmemAEn\_kn;  wire [3:0] dmemAWe\_kn;  wire [31:0] dmemAAddr\_kn;  wire [31:0] dmemAIn\_kn;  wire dmemAEn\_cpu;  wire dmemAEn\_bl;  wire dmemAEn = dmemAEn\_cpu | dmemAEn\_bl | dmemAEn\_kn;    wire [3:0] dmemAWe\_cpu;  wire [3:0] dmemAWe\_bl;  wire [3:0] dmemAWe = dmemAEn\_bl ? dmemAWe\_bl : knWorking ? dmemAWe\_kn : dmemAEn\_cpu ? dmemAWe\_cpu : 4'h0;  wire [31:0] dmemAAddr\_cpu;  wire [31:0] dmemAAddr\_bl;  wire [31:0] dmemAAddr = dmemAEn\_bl ? dmemAAddr\_bl : knWorking ? dmemAAddr\_kn : dmemAEn\_cpu ? dmemAAddr\_cpu : 0;  wire [31:0] dmemAIn\_cpu;  wire [31:0] dmemAIn\_bl;  wire [31:0] dmemAIn = dmemAEn\_bl ? dmemAIn\_bl : knWorking ? dmemAIn\_kn : dmemAEn\_cpu ? dmemAIn\_cpu : 0;  wire [31:0] dmemAOut;  /////////  /// DMEM Address Mapper  wire [31:0] dmemARealAddr = dmemAAddr - 32'h10010000;  // DMEM  assign addr = dmemAEn ? dmemAAddr : 32'hFFFFFFFF;  wire clk\_cpu\_n = ~clk\_cpu;  DMEM dmem (  .clka(clk\_cpu\_n), // input wire clka  .ena(dmemAEn), // input wire ena  .wea(dmemAWe), // input wire [3 : 0] wea  .addra(dmemARealAddr[14:2]),  .dina(dmemAIn), // input wire [31 : 0] dina  .douta(dmemAOut), // output wire [31 : 0] douta  .clkb(clk\_in),  .web(0),  .addrb(debugDMEMAddr),  .dinb('hx),  .doutb(debugDMEMData)  );  //////////////  /// IMEM  ///  wire imemWe\_cpu;  wire imemWe\_bl;  wire imemWe = imemWe\_bl | imemWe\_cpu;  wire [31:0] imemRAddr;  wire [31:0] imemOut;  wire [31:0] imemWAddr\_cpu;  wire [31:0] imemWAddr\_bl;  wire [31:0] imemWAddr = imemWe\_cpu ? imemWAddr\_cpu : imemWe\_bl ? imemWAddr\_bl : 32'hFFFFFFFF;  wire [31:0] imemWData\_cpu;  wire [31:0] imemWData\_bl;  wire [31:0] imemWData = imemWe\_cpu ? imemWData\_cpu : imemWe\_bl ? imemWData\_bl : 32'hFEFEFEFE;  wire [31:0] imemSelectedAddr = imemWe ? imemWAddr : imemRAddr;  wire [31:0] imemRealAddr = imemSelectedAddr - 32'h00400000;    IMEM imem (  .a(imemRealAddr[14:2]),  .d(imemWData),  .dpra(debugIMEMAddr),  .clk(clk\_cpu),  .we(imemWe),  .spo(imemOut),  .dpo(debugIMEMData)  );  //////////////  /// CPU Instantiation  reg cpuEna;  assign pc = imemRAddr;  assign inst = imemOut;  wire [7:0] cpuSyscallFuncCode;  wire [4:0] rfRAddr1\_kn;  wire [31:0] rfRData1;  nyarlathotep sccpu(  .clk(clk\_cpu),  .reset(reset),  .ena(cpuEna),  .dmemAEn(dmemAEn\_cpu),  .dmemAWe(dmemAWe\_cpu),  .dmemAAddr(dmemAAddr\_cpu),  .dmemAIn(dmemAIn\_cpu),  .dmemAOut(dmemAOut),  .inst(imemOut),  .cpuRunning(cpuRunning),  .cpuPaused(cpuPaused),  .pc(imemRAddr),  .imemWAddr(imemWAddr\_cpu),  .imemWData(imemWData\_cpu),  .imemWe(imemWe\_cpu),  .debugRFAddr(debugRFAddr),  .debugRFData(debugRFData),  .syscallFuncCode(cpuSyscallFuncCode),  .rfRAddr1\_kn(rfRAddr1\_kn),  .rfRData1(rfRData1),  .knWorking(knWorking)  );  `include "sdStates.vh"  ////////////////  /// Bootloader and File System Logic  reg blEn;  wire blOk;  wire sdStartEn;  wire sdReadEn;  wire [31:0] sdReadAddr;  wire [31:0] sdReadSectorNum;  wire sdStartOk;  wire sdReadOk;  wire [7:0] sdReadData;  wire sdReadDataValid;  wire sdReadDataASectorDone;    wire blLoadExecutableEn;  wire [87:0] blLoadExecutableName;  wire [31:0] blLoadExecutableIMEMAddr;  wire blLoadExecutableOk;  wire [31:0] blWordLimit;  wire [31:0] blWordOffset;  wire blLoadFileEn;  wire [87:0] blLoadFileName;  wire [31:0] blLoadFileDMEMAddr;  wire blLoadFileOk;  wire blLoadBackgroundFileEn;  wire blLoadBackgroundFileOk;  wire blWorking;  bootloader bootloader\_inst(  .clk(clk\_cpu),  .reset(reset),  .iEn(blEn),  .oBootLoadOk(blOk),  .blWorking(blWorking),  .sdIdle(sdState == S\_SD\_IDLE),  .sdStartEn(sdStartEn),  .sdReadEn(sdReadEn),  .sdReadAddr(sdReadAddr),  .sdReadSectorNum(sdReadSectorNum),  .sdStartOk(sdStartOk),  .sdReadOk(sdReadOk),  .sdReadData(sdReadData),  .sdReadDataValid(sdReadDataValid),  .sdReadDataASectorDone(sdReadDataASectorDone),  .imemWe(imemWe\_bl),  .imemWAddr(imemWAddr\_bl),  .imemWData(imemWData\_bl),  .dmemAWe(dmemAWe\_bl),  .dmemAEn(dmemAEn\_bl),  .dmemAAddr(dmemAAddr\_bl),  .dmemAIn(dmemAIn\_bl),  .iLoadExecutableEn(blLoadExecutableEn),  .iLoadExecutableName(blLoadExecutableName),  .iLoadExecutableIMEMAddr(blLoadExecutableIMEMAddr),  .oLoadExecutableOk(blLoadExecutableOk),  .iWordLimit(blWordLimit),  .iWordOffset(blWordOffset),  .iLoadFileEn(blLoadFileEn),  .iLoadFileName(blLoadFileName),  .iLoadFileDMEMAddr(blLoadFileDMEMAddr),  .oLoadFileOk(blLoadFileOk),  .iLoadBackgroundFileEn(blLoadBackgroundFileEn),  .oLoadBackgroundFileOk(blLoadBackgroundFileOk),  .backgroundMemWea(backgroundMemWea),  .backgroundMemAddra(backgroundMemAddra),  .backgroundMemDina(backgroundMemDina),  .blState(blState),  .blError(blError),  .debugInfo(debugInfo\_bl),  .debugInfoAvailable(debugInfoAvailable\_bl)  );  //////////////////////  /// Kernel - Handles all syscalls.  kernel kernel\_inst(  .clk(clk\_cpu\_n),  .reset(reset),  .funcCode(cpuSyscallFuncCode),  .SW(SW),  .btnu(btnu),  .btnc(btnc\_orig), // 确认按键，可以不受 SW[0] 影响  .btnd(btnd),  .btnl(btnl),  .btnr(btnr),  .rfRAddr1(rfRAddr1\_kn),  .rfRData1(rfRData1),  .working(knWorking),  .initOk(knInitOk),  .dmemAOut(dmemAOut),  .dmemAEn(dmemAEn\_kn),  .dmemAWe(dmemAWe\_kn),  .dmemAAddr(dmemAAddr\_kn),  .dmemAIn(dmemAIn\_kn),  .blLoadExecutableEn(blLoadExecutableEn),  .blLoadExecutableName(blLoadExecutableName),  .blLoadExecutableIMEMAddr(blLoadExecutableIMEMAddr),  .blLoadExecutableOk(blLoadExecutableOk),  .blLoadFileEn(blLoadFileEn),  .blLoadFileName(blLoadFileName),  .blLoadFileDMEMAddr(blLoadFileDMEMAddr),  .blLoadFileOk(blLoadFileOk),  .blWordLimit(blWordLimit),  .blWordOffset(blWordOffset),  .blLoadBackgroundFileEn(blLoadBackgroundFileEn),  .blLoadBackgroundFileOk(blLoadBackgroundFileOk),  .canvasMemWea(canvasMemWea),  .canvasMemAddra(canvasMemAddra),  .canvasMemDina(canvasMemDina),    .sevenSegOut(sevenSegOut)  );  //////////////////////  /// SD Controller and SPI Controller.  wire spiEn;  wire spiClk74En;  wire spiClk74Ok;  wire spiTxEn;  wire spiTxOk;  wire [7:0] spiTxData;  wire spiRxEn;  wire spiRxOk;  wire [7:0] spiRxData;  wire spiClk8En;  wire spiClk8Ok;  sd\_controller sdcon\_inst(  .clk(clk\_cpu),  .rst(reset),  .iStartEn(sdStartEn),  .oStartOk(sdStartOk),  .iReadEn(sdReadEn),  .oReadOk(sdReadOk),  .iReadAddr(sdReadAddr),  .iReadSectorNum(sdReadSectorNum),    .oReadData(sdReadData),  .oReadDataValid(sdReadDataValid),  .oReadDataASectorDone(sdReadDataASectorDone),  .oSpiEn(spiEn),  .oSpiClk74En(spiClk74En),  .iSpiClk74Ok(spiClk74Ok),  .oSpiTxEn(spiTxEn),  .iSpiTxOk(spiTxOk),  .oSpiTxData(spiTxData),  .oSpiRxEn(spiRxEn),  .iSpiRxOk(spiRxOk),  .iSpiRxData(spiRxData),  .oSpiClk8En(spiClk8En),  .iSpiClk8Ok(spiClk8Ok),  .sdState(sdState),  .error(sdError),  .oDebugInfo(debugInfo\_sd),  .oDebugInfoAvailable(debugInfoAvailable\_sd)  );  spi\_controller spicon\_inst(  .clk(clk\_cpu),  .rst(reset),  .en(spiEn),  .iClk74En(spiClk74En),  .oClk74Ok(spiClk74Ok),  .iTxEn(spiTxEn),  .oTxOk(spiTxOk),  .iTxData(spiTxData),  .oRxOk(spiRxOk),  .iRxEn(spiRxEn),  .oRxData(spiRxData),  .oClk8Ok(spiClk8Ok),  .iClk8En(spiClk8En),  .SPI\_CLK(SPI\_CLK),  .SPI\_MOSI(SPI\_MOSI),  .SPI\_MISO(SPI\_MISO),  .SPI\_CSn(SPI\_CSn),  .SD\_RESET(SD\_RESET),  .spiState(spiState),  .speedChoice(3)  );  //////////////  /// Startup Logic  always @(posedge clk\_in)  begin  if(reset) begin  cpuEna <= `False;  blEn <= `False;  compStartOk <= `False;  end else begin  if(blOk) begin  blEn <= `False;  if(knInitOk) begin  cpuEna <= `True;  compStartOk <= `True;  end  end else begin  if(compStartEn) begin  blEn <= `True;  end  end  end  end  endmodule |

Bootloader.v内容：

|  |
| --- |
| `timescale 1ns/1ns  `include "sdHeader.vh"  `define Root 2'h0  `define File 2'h1  `define Executable 2'h2  `define Background 2'h3  module bootloader(  input clk,  input reset,  input iEn,  output reg oBootLoadOk,  output reg blWorking,  input sdIdle,  output reg sdStartEn,  output reg sdReadEn,  output reg [31:0] sdReadAddr,  output reg [31:0] sdReadSectorNum,  input sdStartOk,  input sdReadOk,  input [7:0] sdReadData,  input sdReadDataValid,  input sdReadDataASectorDone,  output reg imemWe,  output reg [31:0] imemWAddr,  output reg [31:0] imemWData,  output reg [3:0] dmemAWe,  output reg dmemAEn,  output reg [31:0] dmemAAddr,  output reg [31:0] dmemAIn,    input iLoadExecutableEn,  input [87:0] iLoadExecutableName,  input [31:0] iLoadExecutableIMEMAddr,  output reg oLoadExecutableOk,  input [31:0] iWordLimit,  input [31:0] iWordOffset,  input iLoadFileEn,  input [87:0] iLoadFileName,  input [31:0] iLoadFileDMEMAddr,  output reg oLoadFileOk,  input iLoadBackgroundFileEn,  output reg oLoadBackgroundFileOk,  output reg backgroundMemWea,  output reg [15:0] backgroundMemAddra,  output reg [11:0] backgroundMemDina,  output reg [7:0] blState,  output reg blError,  output reg [31:0] debugInfo,  output reg debugInfoAvailable  );  localparam initInstAddr = 32'h00400000;  localparam initDataAddr = 32'h10010000;  localparam exceptionEntry = 32'h00400004;  `include "blStates.vh"  localparam N\_BL\_INITSCRIPT = "APOCLYPSBIN";      reg tmpError;    reg [10:0] readByteCounter;  reg [31:0] readSectorCounter;  reg [23:0] mbrHeader;  reg [31:0] dbrAddr;  reg isFdd;  reg [87:0] loadExecutableName;  reg [31:0] loadExecutableIMEMAddr;  reg [87:0] loadFileName;  reg [31:0] loadFileDMEMAddr;  reg [31:0] wordLimit;  reg [31:0] wordOffset;  reg [1:0] executableOrFile;  reg rootClusMemWe;  reg [7:0] rootClusMemDina;  reg [14:0] rootClusMemAddra;  wire [7:0] rootClusMemDoutb;  reg [14:0] rootClusMemAddrb;  reg [5:0] BPB\_SecPerClus\_log2;  reg [15:0] BPB\_RsvdSecCnt;  reg [7:0] BPB\_NumFATs;  reg [31:0] BPB\_FATSz32;  reg [31:0] BPB\_RootClusNum;  reg [31:0] currRootFATSecNum;  reg [31:0] currFileFATSecNum;  reg [31:0] currRootClusNum;  wire [31:0] fatSecNum = dbrAddr + BPB\_RsvdSecCnt;  wire [31:0] firstClusSec = fatSecNum + BPB\_NumFATs \* BPB\_FATSz32;  wire [5:0] clusSize\_log2 = BPB\_SecPerClus\_log2 + 9;  wire [5:0] dirItemsPerClus\_log2 = clusSize\_log2 - 5;  reg [10:0] dirItemIndex;  reg [7:0] scanRootState;  reg [31:0] currFileClusNum;  reg [31:0] currFileSectorCounter;  reg [87:0] currFileName;  reg [31:0] currFileSize;  reg [1:0] clusNumUse;  reg [31:0] nextClusNum;  reg [7:0] nextState;  reg [7:0] nextNextState;  reg [31:0] fatClusAddr;  reg [7:0] rootFATSector [0:511];  reg [7:0] fileFATSector [0:511];  reg [31:0] memWord;  reg [1:0] memByteCount;  reg [15:0] memWordCount;  root\_cluster root\_cluster\_inst (  .clka(clk),  .wea(rootClusMemWe),  .addra(rootClusMemAddra),  .dina(rootClusMemDina),  .clkb(clk),  .addrb(rootClusMemAddrb),  .doutb(rootClusMemDoutb)  );  always @(posedge clk)  begin  if (debugInfoAvailable)  debugInfoAvailable <= `False;  if (rootClusMemWe)  rootClusMemWe <= `Disabled;  if (imemWe)  imemWe <= `Disabled;    if (dmemAEn)  dmemAEn <= `Disabled;  if (dmemAWe)  dmemAWe <= 4'h0;  if (backgroundMemWea)  backgroundMemWea <= `Disabled;  if (oLoadExecutableOk)  oLoadExecutableOk <= `False;  if (oLoadFileOk)  oLoadFileOk <= `False;  if (oLoadBackgroundFileOk)  oLoadBackgroundFileOk <= `False;  if(reset || blState == S\_BL\_RESET) begin  blState <= S\_BL\_INIT;  sdStartEn <= `Disabled;  sdReadEn <= `Disabled;  imemWe <= `Disabled;  oBootLoadOk <= `False;  sdReadAddr <= 0;  sdReadSectorNum <= 0;  imemWAddr <= 0;  imemWData <= 0;  dmemAWe <= 0;  dmemAEn <= `Disabled;  dmemAAddr <= 0;  dmemAIn <= 0;  oLoadExecutableOk <= `False;  oLoadFileOk <= `False;  oLoadBackgroundFileOk <= `False;  blError <= `False;  debugInfoAvailable <= `False;  tmpError <= `False;  readByteCounter <= 0;  readSectorCounter <= 0;  mbrHeader <= 0;  dbrAddr <= 0;  isFdd <= `False;  loadExecutableName <= 0;  loadExecutableIMEMAddr <= 0;  loadFileName <= 0;  loadFileDMEMAddr <= 0;  wordLimit <= 0;  wordOffset <= 0;  executableOrFile <= 0;  rootClusMemWe <= `Disabled;  rootClusMemDina <= 0;  rootClusMemAddra <= 0;  rootClusMemAddrb <= 0;  BPB\_SecPerClus\_log2 <= 0;  BPB\_RsvdSecCnt <= 0;  BPB\_NumFATs <= 0;  BPB\_FATSz32 <= 0;  BPB\_RootClusNum <= 0;  currRootFATSecNum <= 0;  currFileFATSecNum <= 0;  currRootClusNum <= 0;  dirItemIndex <= 0;  scanRootState <= 0;  currFileClusNum <= 0;  currFileName <= 0;  clusNumUse <= 0;  nextClusNum <= 0;  nextState <= 0;  nextNextState <= 0;  fatClusAddr <= 0;  memWord <= 0;  memByteCount <= 0;  memWordCount <= 0;  blWorking <= `False;  end else case(blState)  S\_BL\_INIT:  begin  if (iEn) begin  blState <= S\_BL\_SDSTART;  blWorking <= `True;  end  end  S\_BL\_SDSTART:  begin  if(sdStartOk) begin  blState <= S\_BL\_READSECTOR0\_PRE;  end else begin  sdStartEn <= `Enabled;  end  end  S\_BL\_READSECTOR0\_PRE:  begin  if (sdStartOk && sdIdle)  begin  readByteCounter <= 0;  readSectorCounter <= 0;  sdReadSectorNum <= 1;  sdReadAddr <= 0;  sdReadEn <= `Enabled;  blState <= S\_BL\_READSECTOR0;  tmpError <= `False;  isFdd <= `False;  end else if (~sdStartOk)  blState <= S\_BL\_RESET;  end  S\_BL\_READSECTOR0:  if (sdReadOk) begin  sdReadEn <= `Disabled;    if (tmpError) begin  blState <= S\_BL\_ERROR;  end else begin  blState <= S\_BL\_READDBR\_PRE;  end  end else if (sdReadDataValid) begin  readByteCounter <= readByteCounter + 1;    if(!tmpError) begin  if(readByteCounter < 2) begin  mbrHeader[(readByteCounter << 3) +: 8] <= sdReadData;  end else if (readByteCounter == 2) begin  if(mbrHeader[7:0] == 8'hEB && sdReadData == 8'h90 || mbrHeader[7:0] == 8'hE9) begin  isFdd <= `True;  dbrAddr <= 0;  end  end else if (!isFdd && readByteCounter == 'h1C2) begin  if (sdReadData == 'h0B || sdReadData == 'h0C)  begin  // 第一个分区是 FAT32  ;  end else begin  debugInfo <= {24'hC0, sdReadData};  debugInfoAvailable <= `True;  tmpError <= `True;  end  end else if (!isFdd && readByteCounter >= 'h1C6 && readByteCounter <= 'h1C9)  begin  // 分区的 0 扇区位置  dbrAddr[((readByteCounter - 'h1C6) << 3 ) +: 8] <= sdReadData;  end  end  end else if(sdReadDataASectorDone) begin  readByteCounter <= 0;  readSectorCounter <= readSectorCounter + 1;  end  S\_BL\_READDBR\_PRE:  begin  readByteCounter <= 0;  readSectorCounter <= 0;  sdReadSectorNum <= 1;  sdReadAddr <= dbrAddr;  sdReadEn <= `Enabled;  blState <= S\_BL\_READDBR;  tmpError <= `False;  end  S\_BL\_READDBR:  if (sdReadOk) begin  sdReadEn <= `Disabled;  if(!tmpError) begin  loadExecutableName <= N\_BL\_INITSCRIPT;  loadExecutableIMEMAddr <= initInstAddr;  wordLimit <= 8192;  wordOffset <= 0;  executableOrFile <= `Executable;  blState <= S\_BL\_READROOTCLUS\_PRE;  debugInfo <= {8'hEC, BPB\_FATSz32[23:0]};  debugInfoAvailable <= `True;  end else begin  blState <= S\_BL\_ERROR;  end  end else if (sdReadDataValid) begin  readByteCounter <= readByteCounter + 1;    if(!tmpError) begin  if(readByteCounter == 13) begin  case (sdReadData)  1:BPB\_SecPerClus\_log2 <= 0;  2:BPB\_SecPerClus\_log2 <= 1;  4:BPB\_SecPerClus\_log2 <= 2;  8:BPB\_SecPerClus\_log2 <= 3;  16:BPB\_SecPerClus\_log2 <= 4;  32:BPB\_SecPerClus\_log2 <= 5;  64:BPB\_SecPerClus\_log2 <= 6;  default:begin  debugInfo <= {24'hDC, sdReadData};  debugInfoAvailable <= `True;  tmpError <= `True;  end  endcase  end else if (readByteCounter == 14 || readByteCounter == 15) begin  BPB\_RsvdSecCnt[((readByteCounter - 14) << 3) +: 8] <= sdReadData;  end else if (readByteCounter == 16) begin  BPB\_NumFATs <= sdReadData;  end else if (readByteCounter >= 36 && readByteCounter < 40)  begin  BPB\_FATSz32[((readByteCounter - 36) << 3) +: 8] <= sdReadData;  end else if (readByteCounter >= 44 && readByteCounter < 48)  begin  BPB\_RootClusNum[((readByteCounter - 44) << 3) +: 8] <= sdReadData;  end  end  end else if(sdReadDataASectorDone) begin  readByteCounter <= 0;  readSectorCounter <= readSectorCounter + 1;  end  S\_BL\_READROOTCLUS\_PRE, S\_BL\_LOAD:  begin  currRootFATSecNum <= 32'hFFFFFFFF;  currFileFATSecNum <= 32'hFFFFFFFF;  currRootClusNum <= BPB\_RootClusNum;  debugInfo <= {2'b10, BPB\_SecPerClus\_log2, BPB\_RsvdSecCnt, BPB\_NumFATs};  debugInfoAvailable <= `True;    blState <= S\_BL\_READROOTCLUS\_INVOKE;  end    S\_BL\_READROOTCLUS\_INVOKE:  begin  readByteCounter <= 0;  readSectorCounter <= 0;  tmpError <= `False;  sdReadEn <= `True;  sdReadSectorNum <= (1 << BPB\_SecPerClus\_log2);  sdReadAddr <= ((currRootClusNum - 2) << BPB\_SecPerClus\_log2) + firstClusSec;  blState <= S\_BL\_READROOTCLUS;  end  S\_BL\_READROOTCLUS:  if (sdReadOk) begin  rootClusMemWe <= `Disabled;  sdReadEn <= `Disabled;  rootClusMemAddrb <= 'h10;  blState <= S\_BL\_SCANROOT\_PRE;  end else if (sdReadDataValid) begin  readByteCounter <= readByteCounter + 1;    rootClusMemWe <= `Enabled;  rootClusMemAddra <= (readSectorCounter << 9) | readByteCounter;  rootClusMemDina <= sdReadData;  if (readSectorCounter <= 1 && readByteCounter < 8) begin  debugInfo <= {16'hC7C7, readByteCounter[7:0],sdReadData};  debugInfoAvailable <= `True;  end  end else if (sdReadDataASectorDone) begin  readByteCounter <= 0;  readSectorCounter <= readSectorCounter + 1;  end  S\_BL\_SCANROOT\_PRE:  begin  dirItemIndex <= 0;  scanRootState <= 0;  debugInfo <= {8'hED, sdReadAddr[23:0]};  debugInfoAvailable <= `True;  blState <= S\_BL\_SCANROOT\_DO;  end  S\_BL\_SCANROOT\_INCREMENT:  begin  if (dirItemIndex == (1 << dirItemsPerClus\_log2) - 1) begin  // 读下一簇  blState <= S\_BL\_GETNEXTROOTCLUS;  end else begin  dirItemIndex <= dirItemIndex + 1;  scanRootState <= 0;  blState <= S\_BL\_SCANROOT\_DO;  end  end  S\_BL\_SCANROOT\_DO:  begin  scanRootState <= scanRootState + 1;  case(scanRootState)  0:  begin  rootClusMemAddrb <= (dirItemIndex << 5) + 26;  if(dirItemIndex == 0)begin  debugInfo <= {24'hE7, rootClusMemDoutb};  debugInfoAvailable <= `True;  end  end  1,2,3,4,5:  begin  if (scanRootState == 2) begin  rootClusMemAddrb <= rootClusMemAddrb - 7;  end else begin  rootClusMemAddrb <= rootClusMemAddrb + 1;  end  if (scanRootState == 1) begin  ;  end else begin  currFileClusNum[((scanRootState - 2) << 3) +: 8] <= rootClusMemDoutb;  end  end  6:  begin  if (currFileClusNum == 0) begin  scanRootState <= 0;  blState <= S\_BL\_SCANROOT\_INCREMENT;  end else begin  rootClusMemAddrb <= (dirItemIndex << 5) + 0;  end  end  7,8,9,10,11,12,13,14,15,16,17,18:  begin  rootClusMemAddrb <= rootClusMemAddrb + 1;  if(scanRootState == 7) begin  ;  end else begin  // Big Endian  // Upper Case  currFileName[((18 - scanRootState) << 3) +: 8] <= ((rootClusMemDoutb >= 'h61 && rootClusMemDoutb <= 'h7a) ? (rootClusMemDoutb - 'h20) : rootClusMemDoutb);  end  end    19:  begin  if (currFileName == ((executableOrFile == `Executable) ? loadExecutableName : loadFileName)) begin  debugInfo <= currFileName[31:0];  debugInfoAvailable <= `True;  rootClusMemAddrb <= (dirItemIndex << 5) + 11;  end else begin  debugInfo <= currFileName[31:0];  debugInfoAvailable <= `True;  blState <= S\_BL\_SCANROOT\_INCREMENT;  end  end    20:  ;  21:  begin  if((rootClusMemDoutb & 8'h0F) == 8'h0F) begin  debugInfo <= {{3{8'hA2}}, rootClusMemDoutb};  debugInfoAvailable <= `True;  blState <= S\_BL\_SCANROOT\_INCREMENT;  end else begin  rootClusMemAddrb <= (dirItemIndex << 5) + 28;  end  end  22:  rootClusMemAddrb <= rootClusMemAddrb + 1;  23,24,25,26:  begin  rootClusMemAddrb <= rootClusMemAddrb + 1;  currFileSize[((scanRootState - 23) << 3) +: 8] <= rootClusMemDoutb;  end  27:  if (currFileSize == 0) begin  debugInfo <= currFileSize;  debugInfoAvailable <= `True;  blState <= S\_BL\_SCANROOT\_INCREMENT;  end else begin  if (executableOrFile == `Executable) begin  ;  end else if (executableOrFile == `File) begin  dmemAEn <= `Enabled;  dmemAWe <= (loadFileDMEMAddr != 32'h10010000) ? (4'hf) : (4'h0);  dmemAAddr <= loadFileDMEMAddr - 1;  dmemAIn <= currFileSize;  end  blState <= S\_BL\_READFILECLUS\_PRE\_JUMPCLUS;  end  endcase  end  S\_BL\_GETNEXTROOTCLUS:  begin  nextState <= S\_BL\_GETNEXTROOTCLUS\_POST;  clusNumUse <= `Root;  blState <= S\_BL\_GETNEXTCLUSNUM;  end  S\_BL\_GETNEXTROOTCLUS\_POST:  begin  //debugInfo <= {8'hB9, nextClusNum[23:0]};  //debugInfoAvailable <= `True;  if (nextClusNum[30:0] != 31'hfffffff) begin  currRootClusNum <= nextClusNum;  blState <= S\_BL\_READROOTCLUS\_INVOKE;  end else begin  debugInfo <= {8'hBA, nextClusNum[23:0]};  debugInfoAvailable <= `True;  blState <= S\_BL\_ERROR;  end  end  S\_BL\_GETNEXTCLUSNUM:  begin  nextNextState <= nextState;  if( (clusNumUse == `Root && ((currRootClusNum << 2) >> 9) + fatSecNum == currRootFATSecNum ) || (clusNumUse == `File && ((currFileClusNum << 2) >> 9) + fatSecNum == currFileFATSecNum )) begin  // 不需要读 FAT 表，之前已经读入过  blState <= S\_BL\_GETNEXTCLUSNUM\_POST;  end else begin  fatClusAddr <= (clusNumUse == `Root) ? (((currRootClusNum << 2) >> 9) + fatSecNum) : ((currFileClusNum << 2) >> 9) + fatSecNum;  nextState <= S\_BL\_GETNEXTCLUSNUM\_POST;  blState <= S\_BL\_READFATSEC\_PRE;  end  end  S\_BL\_GETNEXTCLUSNUM\_POST:  begin  //debugInfo <= {8'hB5, fatClusAddr[23:0]};  //debugInfoAvailable <= `True;    if(clusNumUse == `File) begin  nextClusNum <= {fileFATSector[((currFileClusNum << 2) & 9'b111111111) + 3], fileFATSector[((currFileClusNum << 2) & 9'b111111111) + 2], fileFATSector[((currFileClusNum << 2) & 9'b111111111) + 1], fileFATSector[(currFileClusNum << 2) & 9'b111111111]};  currFileFATSecNum <= fatClusAddr;  end else begin  nextClusNum <= {rootFATSector[((currRootClusNum << 2) & 9'b111111111) + 3], rootFATSector[((currRootClusNum << 2) & 9'b111111111) + 2], rootFATSector[((currRootClusNum << 2) & 9'b111111111) + 1], rootFATSector[(currRootClusNum << 2) & 9'b111111111]};  currRootFATSecNum <= fatClusAddr;  end  blState <= nextNextState;  end  S\_BL\_READFATSEC\_PRE:  begin  sdReadEn <= `Enabled;  readByteCounter <= 0;  readSectorCounter <= 0;  sdReadSectorNum <= 1;  sdReadAddr <= fatClusAddr;  blState <= S\_BL\_READFATSEC;  end  S\_BL\_READFATSEC:  if(sdReadOk) begin  blState <= nextState;  sdReadEn <= `Disabled;  end else if (sdReadDataValid) begin  readByteCounter <= readByteCounter + 1;    if (clusNumUse == `File)  fileFATSector[readByteCounter] <= sdReadData;  else  rootFATSector[readByteCounter] <= sdReadData;  end else if (sdReadDataASectorDone) begin  readByteCounter <= 0;  readSectorCounter <= readSectorCounter + 1;  end  S\_BL\_READFILECLUS\_PRE\_JUMPCLUS:  begin  readSectorCounter <= 0;  currFileSectorCounter <= 0;  memWordCount <= 0;  blState <= S\_BL\_READFILECLUS\_PRE\_JUMPCLUS\_CHECK;  end  S\_BL\_READFILECLUS\_PRE\_JUMPCLUS\_CHECK:  begin  if (((wordOffset >> 7) >> BPB\_SecPerClus\_log2) != (currFileSectorCounter >> BPB\_SecPerClus\_log2)) begin  nextState <= S\_BL\_READFILECLUS\_PRE\_JUMPCLUS\_POST;  clusNumUse <= `File;  blState <= S\_BL\_GETNEXTCLUSNUM;  end else begin  blState <= S\_BL\_READFILECLUS\_PRE;  end  end  S\_BL\_READFILECLUS\_PRE\_JUMPCLUS\_POST:  if (nextClusNum[30:0] != 31'hfffffff) begin  debugInfo <= {8'h6E, currFileSectorCounter[23:0]};  debugInfoAvailable <= `True;  currFileClusNum <= nextClusNum;  currFileSectorCounter <= currFileSectorCounter + (1 << BPB\_SecPerClus\_log2);  blState <= S\_BL\_READFILECLUS\_PRE\_JUMPCLUS\_CHECK;  end else begin  debugInfo <= {8'h6D, currFileSectorCounter[23:0]};  debugInfoAvailable <= `True;  blState <= S\_BL\_ERROR;  end  // currFileSectorCounter: 备份 readSectorCounter  S\_BL\_READFILECLUS\_PRE:  begin  sdReadEn <= `Enabled;  readByteCounter <= 0;  readSectorCounter <= currFileSectorCounter;  sdReadSectorNum <= (1 << BPB\_SecPerClus\_log2);  sdReadAddr <= ((currFileClusNum - 2) << BPB\_SecPerClus\_log2) + firstClusSec;  memWord <= 0;  memByteCount <= 0;  blState <= S\_BL\_READFILECLUS;  end  S\_BL\_READFILECLUS:  if(sdReadOk) begin  sdReadEn <= `Disabled;  blState <= S\_BL\_READFILECLUS\_CHECKNEXTCLUS;  debugInfo <= {8'hB6, sdReadAddr[23:0]};  debugInfoAvailable <= `True;  end else if (sdReadDataValid) begin  readByteCounter <= readByteCounter + 1;  memByteCount <= memByteCount + 1;  memWord[{memByteCount, 3'h0} +: 8] <= sdReadData;  if (readSectorCounter <= 1 && readByteCounter > 0 && readByteCounter < 14 && memByteCount == 0) begin  debugInfo <= {8'hB7, imemWAddr[23:0]};  debugInfoAvailable <= `True;  end  if (memByteCount == 3)  begin  if (readSectorCounter <= 1 && readByteCounter < 14) begin  debugInfo <= {8'hB5, memWord[23:0]};  debugInfoAvailable <= `True;  end    if (memWordCount < wordLimit && (((readSectorCounter << 9) | readByteCounter) >> 2) <= ((currFileSize - 1) >> 2) && (((readSectorCounter << 9) | readByteCounter) >> 2) >= wordOffset)  begin  memWordCount <= memWordCount + 1;  if (executableOrFile == `Executable) begin  imemWe <= `Enabled;  imemWAddr <= loadExecutableIMEMAddr + (((readSectorCounter << 9) | readByteCounter)) - (wordOffset << 2) - 3;  imemWData <= {sdReadData, memWord[23:0]};  end else if (executableOrFile == `File) begin  dmemAEn <= `Enabled;  dmemAWe <= 4'hf;  dmemAAddr <= loadFileDMEMAddr + (((readSectorCounter << 9) | readByteCounter)) - (wordOffset << 2) - 3;  dmemAIn <= {sdReadData, memWord[23:0]};  end else if (executableOrFile == `Background) begin  backgroundMemWea <= `Enabled;  backgroundMemAddra <= (((readSectorCounter << 9) | readByteCounter)) >> 1;  backgroundMemDina <= {sdReadData[7:4], sdReadData[2:0], memWord[23], memWord[20:17]};  end  end  end else if (memByteCount == 1)  begin  if (executableOrFile == `Background) begin  backgroundMemWea <= `Enabled;  backgroundMemAddra <= (((readSectorCounter << 9) | readByteCounter)) >> 1;  backgroundMemDina <= {sdReadData[7:4], sdReadData[2:0], memWord[7], memWord[4:1]};  end  end  end else if (sdReadDataASectorDone) begin  readByteCounter <= 0;  readSectorCounter <= readSectorCounter + 1;  end  S\_BL\_READFILECLUS\_CHECKNEXTCLUS:  begin  if(memWordCount < wordLimit) begin  currFileSectorCounter <= readSectorCounter;  nextState <= S\_BL\_READFILECLUS\_CHECKNEXTCLUS\_POST;  clusNumUse <= `File;  blState <= S\_BL\_GETNEXTCLUSNUM;  end else begin  debugInfo <= {8'hBf, currFileSectorCounter[23:0]};  debugInfoAvailable <= `True;  blState <= S\_BL\_READFILECLUS\_POST;  end  end  S\_BL\_READFILECLUS\_CHECKNEXTCLUS\_POST:  begin  if (nextClusNum[30:0] != 31'hfffffff) begin  debugInfo <= {8'hBE, currFileSectorCounter[23:0]};  debugInfoAvailable <= `True;  currFileClusNum <= nextClusNum;  sdReadEn <= `Enabled;  readByteCounter <= 0;  readSectorCounter <= currFileSectorCounter;  sdReadSectorNum <= (1 << BPB\_SecPerClus\_log2);  sdReadAddr <= ((nextClusNum - 2) << BPB\_SecPerClus\_log2) + firstClusSec;  memWord <= 0;  memByteCount <= 0;  blState <= S\_BL\_READFILECLUS;  end else begin  debugInfo <= {8'hBD, currFileSectorCounter[23:0]};  debugInfoAvailable <= `True;  blState <= S\_BL\_READFILECLUS\_POST;  end  end  S\_BL\_READFILECLUS\_POST:  begin  if(!oBootLoadOk) begin  oBootLoadOk <= `True;  end else begin  if (executableOrFile == `Executable)  oLoadExecutableOk <= `True;  else if (executableOrFile == `File)  oLoadFileOk <= `True;  else if (executableOrFile == `Background)  oLoadBackgroundFileOk <= `True;  end  blState <= S\_BL\_IDLE;  end    S\_BL\_IDLE:  begin  blWorking <= `False;  if (iLoadExecutableEn) begin  loadExecutableIMEMAddr <= iLoadExecutableIMEMAddr;  loadExecutableName <= iLoadExecutableName;  wordLimit <= iWordLimit;  wordOffset <= iWordOffset;  executableOrFile <= `Executable;  blState <= S\_BL\_LOAD;  blWorking <= `True;  end else if (iLoadFileEn) begin  loadFileDMEMAddr <= iLoadFileDMEMAddr;  loadFileName <= iLoadFileName;  wordLimit <= iWordLimit;  wordOffset <= iWordOffset;  executableOrFile <= `File;  blState <= S\_BL\_LOAD;  blWorking <= `True;  end else if (iLoadBackgroundFileEn) begin  loadFileName <= iLoadFileName;  wordLimit <= 32'hFFFFFFFF;  wordOffset <= 0;  executableOrFile <= `Background;  blState <= S\_BL\_LOAD;  blWorking <= `True;  end  end  S\_BL\_ERROR:  blError <= `True;  default:  blState <= S\_BL\_RESET;  endcase  end  endmodule |

Kernel.v 内容：

|  |
| --- |
| `timescale 1ns/1ns  `include "sdHeader.vh"  module kernel(  input clk,  input reset,  input [7:0] funcCode,  input [15:0] SW,  input btnu,  input btnc,  input btnd,  input btnl,  input btnr,    output reg [4:0] rfRAddr1,  input [31:0] rfRData1,  output reg working,  output reg initOk,  input [31:0] dmemAOut,  output reg dmemAEn,  output reg [3:0] dmemAWe,  output reg [31:0] dmemAAddr,  output reg [31:0] dmemAIn,  output reg blLoadExecutableEn,  output reg [87:0] blLoadExecutableName,  output reg [31:0] blLoadExecutableIMEMAddr,  input blLoadExecutableOk,  output reg blLoadFileEn,  output reg [87:0] blLoadFileName,  output reg [31:0] blLoadFileDMEMAddr,  input blLoadFileOk,  output reg [31:0] blWordLimit,  output reg [31:0] blWordOffset,  output reg blLoadBackgroundFileEn,  input blLoadBackgroundFileOk,  output reg canvasMemWea,  output reg [17:0] canvasMemAddra,  output reg [12:0] canvasMemDina,  output reg [31:0] sevenSegOut  );  reg [31:0] v0;  reg [31:0] v1;  reg [31:0] a0;  reg [31:0] a1;  reg [31:0] a2;  reg [7:0] knState;  reg [7:0] nextState;  reg [7:0] funcCode\_stored;  `include "knStates.vh"  localparam canvasSize = 512\*288;  reg [17:0] canvasPixelCount;  always @(posedge clk) begin  if (reset) begin  working <= `False;  dmemAEn <= `Disabled;  dmemAWe <= 0;  dmemAAddr <= 0;  dmemAIn <= 0;    blLoadExecutableEn <= `Disabled;  blLoadFileEn <= `Disabled;  blLoadBackgroundFileEn <= `Disabled;  canvasMemWea <= `Disabled;  sevenSegOut <= 32'h12ABCDEF;  initOk <= `False;  canvasPixelCount <= 0;  end else if(~working && ~initOk) begin  working <= `True;  canvasPixelCount <= 0;  knState <= S\_KN\_CLEARCANVAS;  dmemAWe <= 0;  end else if(working) begin  case (knState)  S\_KN\_READV0:  begin  knState <= S\_KN\_READV1;  v0 <= rfRData1;  end  S\_KN\_READV1:  begin  knState <= S\_KN\_READA0;  v1 <= rfRData1;  end  S\_KN\_READA0:  begin  knState <= S\_KN\_READA1;  a0 <= rfRData1;  end  S\_KN\_READA1:  begin  knState <= S\_KN\_READA2;  a1 <= rfRData1;  end  S\_KN\_READA2:  begin  knState <= S\_KN\_FUNCBRANCH;  a2 <= rfRData1;  end  S\_KN\_FUNCBRANCH:  begin  if(funcCode\_stored == 'h11) begin  knState <= S\_KN\_READSDTODMEM;  end else if(funcCode\_stored == 'h12) begin  knState <= S\_KN\_READSDTOIMEM;  end else if(funcCode\_stored == 'h13) begin  knState <= S\_KN\_READSDTOBACKGROUND;  end else if(funcCode\_stored == 'h1) begin  knState <= S\_KN\_CHANGE7SEG;  end else if(funcCode\_stored == 'h2) begin  knState <= S\_KN\_DRAWPIXEL;  end else if(funcCode\_stored == 'h3) begin  canvasPixelCount <= 0;  knState <= S\_KN\_CLEARCANVAS;  end else if(funcCode\_stored == 'h8) begin  knState <= S\_KN\_WAITFORBUTTON;  end else if(funcCode\_stored == 'h9) begin  knState <= S\_KN\_READSW;  end else begin  knState <= S\_KN\_DONE;  end  end  S\_KN\_DONE:  begin  working <= `False;  canvasMemWea <= `False;  dmemAWe <= 0;  dmemAEn <= `Disabled;  end  S\_KN\_READSDTODMEM:  begin  nextState <= S\_KN\_READSDTODMEM\_DO;  knState <= S\_KN\_READFILENAME;  end  S\_KN\_READSDTOIMEM:  begin  nextState <= S\_KN\_READSDTOIMEM\_DO\_PRE;  knState <= S\_KN\_READFILENAME;  end  S\_KN\_READSDTOBACKGROUND:  begin  nextState <= S\_KN\_READSDTOBACKGROUND\_DO;  knState <= S\_KN\_READFILENAME;  end  S\_KN\_READFILENAME:  begin  // 从内存读取文件名  dmemAEn <= `True;  dmemAAddr <= v1;  knState <= S\_KN\_READFILENAME\_0;  end  S\_KN\_READFILENAME\_0:  begin  dmemAEn <= `True;  dmemAAddr <= dmemAAddr + 4;  knState <= S\_KN\_READFILENAME\_1;  end  S\_KN\_READFILENAME\_1:  begin  blLoadFileName[87:56] <= dmemAOut;  dmemAEn <= `True;  dmemAAddr <= dmemAAddr + 4;  knState <= S\_KN\_READFILENAME\_2;  end  S\_KN\_READFILENAME\_2:  begin  blLoadFileName[55:24] <= dmemAOut;  knState <= S\_KN\_READFILENAME\_3;  end  S\_KN\_READFILENAME\_3:  begin  blLoadFileName[23:0] <= dmemAOut[31:8];  knState <= nextState;  end  S\_KN\_READSDTODMEM\_DO:  begin  if(blLoadFileOk) begin  knState <= S\_KN\_DONE;  blLoadFileEn <= `False;  end else begin  blLoadFileDMEMAddr <= a0;  blWordOffset <= a1;  blWordLimit <= a2;  blLoadFileEn <= `True;  end  end  S\_KN\_READSDTOIMEM\_DO\_PRE:  begin  blLoadExecutableName <= blLoadFileName;  knState <= S\_KN\_READSDTOIMEM\_DO;  end  S\_KN\_READSDTOIMEM\_DO:  begin  if(blLoadExecutableOk) begin  knState <= S\_KN\_DONE;  blLoadExecutableEn <= `False;  end else begin  blLoadExecutableIMEMAddr <= a0;  blWordOffset <= a1;  blWordLimit <= a2;  blLoadExecutableEn <= `True;  end  end  S\_KN\_READSDTOBACKGROUND\_DO:  begin  if(blLoadBackgroundFileOk) begin  knState <= S\_KN\_DONE;  blLoadBackgroundFileEn <= `False;  end else begin  blLoadBackgroundFileEn <= `True;  end  end  S\_KN\_CHANGE7SEG:  begin  sevenSegOut <= v1;  knState <= S\_KN\_DONE;  end  S\_KN\_DRAWPIXEL:  begin  canvasMemWea <= `Enabled;  canvasMemAddra <= {v1[24:16], v1[8:0]};  canvasMemDina <= {1'b1, a0[15:12], a0[10:7], a0[4:1]};  knState <= S\_KN\_DONE;  end  S\_KN\_CLEARCANVAS:  begin  if (canvasPixelCount < canvasSize) begin  canvasMemWea <= `Enabled;  canvasMemAddra <= canvasPixelCount;  canvasMemDina <= 0;  canvasPixelCount <= canvasPixelCount + 1;  end else begin  canvasMemWea <= `Disabled;  initOk <= `True;  knState <= S\_KN\_DONE;  end  end  S\_KN\_WAITFORBUTTON:  begin  knState <= S\_KN\_DONE;  if(btnu) begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= 32'h1;  end else if(btnc) begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= 32'h2;  end else if(btnd) begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= 32'h3;  end else if(btnl) begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= 32'h4;  end else if(btnr) begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= 32'h5;  end else begin  knState <= S\_KN\_WAITFORBUTTON;  end  end  S\_KN\_READSW:  begin  dmemAEn <= `True;  dmemAWe <= 'hf;  dmemAAddr <= a0;  dmemAIn <= {16'h0, SW};  knState <= S\_KN\_DONE;  end  default:  begin  knState <= S\_KN\_DONE;  end  endcase  end else if(funcCode != 0) begin  working <= `True;  knState <= S\_KN\_READV0;  funcCode\_stored <= funcCode;  dmemAWe <= 0;  end  end  always @\* begin  case (knState)  S\_KN\_READV0:  begin  rfRAddr1 = 2;  end  S\_KN\_READV1:  begin  rfRAddr1 = 3;  end  S\_KN\_READA0:  begin  rfRAddr1 = 4;  end  S\_KN\_READA1:  begin  rfRAddr1 = 5;  end  S\_KN\_READA2:  begin  rfRAddr1 = 6;  end  default:  rfRAddr1 = 0;  endcase  end  endmodule |

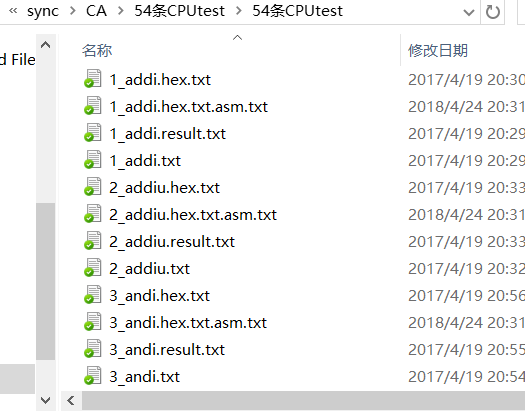
Nyalarthotep(CPU).v 内容如下：

|  |
| --- |
| `timescale 1ns / 1ns  `define ENABLE 1'b1  `define DISABLE 1'b0  `define SIGNED 1'b1  `define UNSIGNED 1'b0    module nyarlathotep(  input clk,  input reset,  input ena,  input [31:0] inst, //imemOut  output cpuRunning,  output cpuPaused,  output reg [31:0] pc, //imemRAddr  output reg dmemAEn,  output [3:0] dmemAWe,  output reg [31:0] dmemAAddr,  output reg [31:0] dmemAIn,  input [31:0] dmemAOut,  output [31:0] imemWAddr,  output [31:0] imemWData,  output imemWe,  input [4:0] debugRFAddr,  output [31:0] debugRFData,  output reg [7:0] syscallFuncCode,  input [4:0] rfRAddr1\_kn,  output [31:0] rfRData1,  input knWorking  );  assign imemWAddr = 32'h0;  assign imemWData = 32'h0;  assign imemWe = `DISABLE;    reg cpuStarted;  assign cpuRunning = (ena & cpuStarted);  `include "aluHeader.vh"  ////////////////////  /// Parts Instantiating  /// Register File  reg rfWe;  reg [4:0] rfRAddr1;  reg [4:0] rfRAddr2;  reg [4:0] rfWAddr;  reg [31:0] rfWData;  wire [31:0] rfRData2;  regfile cpu\_ref(  .clk(clk),  .rst(reset),  .we(rfWe),  .cpuPaused(cpuPaused),  .raddr1(knWorking ? rfRAddr1\_kn : rfRAddr1),  .raddr2(rfRAddr2),  .waddr(rfWAddr),  .rdata1(rfRData1),  .rdata2(rfRData2),  .wdata(rfWData),  .debugRFAddr(debugRFAddr),  .debugRFData(debugRFData)  );  /// ALU  reg [31:0] aluA;  reg [31:0] aluB;  reg [4:0] aluModeSel;  wire [31:0] aluR;  wire [31:0] aluRX;  wire aluBusy;  wire aluZero, aluCarry, aluNegative, aluOverflow;  ALU alu(  .clk(~clk),  .A(aluA),  .B(aluB),  .modeSel(aluModeSel),  .R(aluR),  .RX(aluRX),  .isRZero(aluZero),  .isCarry(aluCarry),  .isRNegative(aluNegative),  .isOverflow(aluOverflow),  .busy(aluBusy)  );  ///////////////  /// Extender  ///  reg [31:0] extend16S\_1In;  wire [31:0] extend16S\_1Out;  reg [31:0] extend16S\_2In;  wire [31:0] extend16S\_2Out;  reg [31:0] extend16UIn;  wire [31:0] extend16UOut;  reg [31:0] extend8SIn;  wire [31:0] extend8SOut;  reg [31:0] extend8UIn;  wire [31:0] extend8UOut;  extender #(16, `SIGNED) extend16S\_1(  .in(extend16S\_1In),  .out(extend16S\_1Out)  );  extender #(16, `SIGNED) extend16S\_2(  .in(extend16S\_2In),  .out(extend16S\_2Out)  );  extender #(16, `UNSIGNED) extend16U(  .in(extend16UIn),  .out(extend16UOut)  );  extender #(8, `SIGNED) extend8S(  .in(extend8SIn),  .out(extend8SOut)  );  extender #(8, `UNSIGNED) extend8U(  .in(extend8UIn),  .out(extend8UOut)  );  /////////////////////////////////////  /// Special structures  /// Instruction Decoder  wire [5:0] op = inst[31:26];  wire [5:0] func = inst[5:0];  wire [4:0] rs = inst[25:21];  wire [4:0] base = inst[25:21];  wire [4:0] rt = inst[20:16];  wire [4:0] rd = inst[15:11];  wire [4:0] shamt = inst[10:6];  wire [15:0] imm = inst[15:0];  wire [25:0] index = inst[25:0];  reg iMul, iAdd, iAddu, iSub, iSubu, iAnd, iOr, iXor, iNor, iSlt, iSltu, iSll, iSrl, iSra, iSllv, iSrlv, iSrav, iJr, iAddi, iAddiu, iAndi, iOri, iXori, iLw, iSw, iBeq, iBne, iSlti, iSltiu, iLui, iJ, iJal, iDiv, iDivu, iMult, iMultu, iBgez, iJalr, iLbu, iLhu, iLb, iLh, iSb, iSh, iBreak, iSyscall, iEret, iMfhi, iMflo, iMthi, iMtlo, iMfc0, iMtc0, iClz, iTeq;  always @ (\*)  begin  if(op == 6'b000000 && func == 6'b100000) iAdd = 1; else iAdd = 0;  if(op == 6'b000000 && func == 6'b100001) iAddu = 1; else iAddu = 0;  if(op == 6'b000000 && func == 6'b100010) iSub = 1; else iSub = 0;  if(op == 6'b000000 && func == 6'b100011) iSubu = 1; else iSubu = 0;  if(op == 6'b000000 && func == 6'b100100) iAnd = 1; else iAnd = 0;  if(op == 6'b000000 && func == 6'b100101) iOr = 1; else iOr = 0;  if(op == 6'b000000 && func == 6'b100110) iXor = 1; else iXor = 0;  if(op == 6'b000000 && func == 6'b100111) iNor = 1; else iNor = 0;  if(op == 6'b000000 && func == 6'b101010) iSlt = 1; else iSlt = 0;  if(op == 6'b000000 && func == 6'b101011) iSltu = 1; else iSltu = 0;  if(op == 6'b000000 && func == 6'b000000) iSll = 1; else iSll = 0;  if(op == 6'b000000 && func == 6'b000010) iSrl = 1; else iSrl = 0;  if(op == 6'b000000 && func == 6'b000011) iSra = 1; else iSra = 0;  if(op == 6'b000000 && func == 6'b000100) iSllv = 1; else iSllv = 0;  if(op == 6'b000000 && func == 6'b000110) iSrlv = 1; else iSrlv = 0;  if(op == 6'b000000 && func == 6'b000111) iSrav = 1; else iSrav = 0;  if(op == 6'b000000 && func == 6'b001000 && rt == 5'h0 && rd == 5'h0) iJr = 1; else iJr = 0;  if(op == 6'b001000) iAddi = 1; else iAddi = 0;  if(op == 6'b001001) iAddiu = 1; else iAddiu = 0;  if(op == 6'b001100) iAndi = 1; else iAndi = 0;  if(op == 6'b001101) iOri = 1; else iOri = 0;  if(op == 6'b001110) iXori = 1; else iXori = 0;  if(op == 6'b100011) iLw = 1; else iLw = 0;  if(op == 6'b101011) iSw = 1; else iSw = 0;  if(op == 6'b000100) iBeq = 1; else iBeq = 0;  if(op == 6'b000101) iBne = 1; else iBne = 0;  if(op == 6'b001010) iSlti = 1; else iSlti = 0;  if(op == 6'b001011) iSltiu = 1; else iSltiu = 0;  if(op == 6'b001111) iLui = 1; else iLui = 0;  if(op == 6'b000010) iJ = 1; else iJ = 0;  if(op == 6'b000011) iJal = 1; else iJal = 0;    if(op == 6'b000000 && func == 6'b011010 && rd == 5'h0) iDiv = 1; else iDiv = 0;  if(op == 6'b000000 && func == 6'b011011 && rd == 5'h0) iDivu = 1; else iDivu = 0;  if(op == 6'b011100 && func == 6'b000010) iMul = 1; else iMul = 0;  if(op == 6'b000000 && func == 6'b011000 && rd == 5'h0) iMult = 1; else iMult = 0;  if(op == 6'b000000 && func == 6'b011001 && rd == 5'h0) iMultu = 1; else iMultu = 0;  if(op == 6'b000001 && rt == 5'b00001) iBgez = 1; else iBgez = 0;  if(op == 6'b000000 && rt == 5'b00000 && shamt == 5'b00000 && func == 6'b001001) iJalr = 1; else iJalr = 0;  if(op == 6'b100000) iLb = 1; else iLb = 0;  if(op == 6'b100001) iLh = 1; else iLh = 0;  if(op == 6'b100100) iLbu = 1; else iLbu = 0;  if(op == 6'b100101) iLhu = 1; else iLhu = 0;  if(op == 6'b101000) iSb = 1; else iSb = 0;  if(op == 6'b101001) iSh = 1; else iSh = 0;  if(op == 6'b000000 && func == 6'b001101) iBreak = 1; else iBreak = 0;  if(inst == 32'b000000\_00000\_00000\_00000\_00000\_001100) iSyscall = 1; else iSyscall = 0;  if(inst == 32'h42000018) iEret = 1; else iEret = 0;  if(op == 6'b000000 && rt == 5'h0 && func == 6'b010000) iMfhi = 1; else iMfhi = 0;  if(op == 6'b000000 && rt == 5'h0 && func == 6'b010010) iMflo = 1; else iMflo = 0;  if(op == 6'b000000 && rd == 5'h0 && rt == 5'h0 && func == 6'b010001) iMthi = 1; else iMthi = 0;  if(op == 6'b000000 && rd == 5'h0 && rt == 5'h0 && func == 6'b010011) iMtlo = 1; else iMtlo = 0;  if(op == 6'b010000 && rs == 5'b00000 && inst[10:3] == 8'h00) iMfc0 = 1; else iMfc0 = 0;  if(op == 6'b010000 && rs == 5'b00100 && inst[10:3] == 8'h00) iMtc0 = 1; else iMtc0 = 0;  if(op == 6'b011100 && func == 6'b100000) iClz = 1; else iClz = 0;  if(op == 6'b000000 && func == 6'b110100) iTeq = 1; else iTeq = 0;  end  /////////////////////  /// UADD - unsigned adder  ///  reg [31:0] uaddA;  reg [31:0] uaddB;  wire [31:0] uaddR = uaddA + uaddB;  ///////////////////////  /// Next PC  wire [31:0] pcPlus4 = pc + 4;  reg [31:0] nextPC;  wire [31:0] nextPC\_cp0;  wire [31:0] cp0ExecAddr;  ///////////////////////  /// PC, HI, LO  /// PC is defined at I/O ports.  reg [31:0] hi;  reg [31:0] lo;  reg [31:0] nextHi;  reg [31:0] nextLo;  /////////////////////////  /// Main Blocks  reg [3:0] startCounter;  localparam startNo = 10;  localparam initInstAddr = 32'h00400000;  localparam initDataAddr = 32'h10010000;  assign cpuPaused = aluBusy | knWorking;  always @(posedge clk) begin  if(reset == `ENABLE) begin  startCounter <= 0;  pc <= initInstAddr;  hi <= 0;  lo <= 0;  end  else  begin  if(cpuStarted == `DISABLE) begin  if(startCounter < startNo - 1) begin  startCounter <= startCounter + 1;  end  else if (startCounter >= startNo - 1) begin  startCounter <= startNo - 1;  end  end else if(cpuRunning & ~cpuPaused) begin  pc <= nextPC; //update PC at rising edge  lo <= nextLo;  hi <= nextHi;  end  end  end  always @(negedge clk) begin  if(reset == `ENABLE) begin  cpuStarted <= `DISABLE;  end  else  begin  if(cpuStarted == `DISABLE && ena) begin  if(startCounter == startNo - 1) begin  cpuStarted <= `ENABLE;  end  end  end  end      `define SYSCALLCAUSE 5'b01000  `define BREAKCAUSE 5'b01001  `define TEQCAUSE 5'b01101  //////////////////////////////////  /// Instruction-specific datapath  reg [4:0] bytePos;    reg [4:0] cp0Addr;  reg [31:0] cp0WData;  wire [31:0] cp0RData;  reg cp0Exception;  wire cp0Intr = `DISABLE;  wire [31:0] cp0Status;  reg [31:0] cp0Cause;  reg trap;  reg [3:0] dmemAWe\_orig;  assign dmemAWe = dmemAWe\_orig & {4{cpuRunning & ~cpuPaused}};  localparam BigEndianCPU = 1'b0;  always @(\*) begin  trap = 0;  cp0Exception = `DISABLE;  cp0Addr = 32'h0;  cp0WData = 32'h0;  cp0Cause = 32'h0;  extend16S\_1In = 32'hxxxxxxxx;  extend16S\_2In = 32'hxxxxxxxx;  extend16UIn = 32'hxxxxxxxx;  extend8SIn = 32'hxxxxxxxx;  extend8UIn = 32'hxxxxxxxx;    dmemAWe\_orig = 4'h0;  dmemAEn = 1'b0;  dmemAIn = 32'haaaaaaaa;  dmemAAddr = initDataAddr;  rfRAddr1 = 32'hxxxxxxxx;  rfRAddr2 = 32'hxxxxxxxx;  bytePos = 5'bxxxxx;  aluA = 32'hxxxxxxxx;  aluB = 32'hxxxxxxxx;  aluModeSel = ALU\_AND;  rfWe = 1'b0;  rfWAddr = 32'hxxxxxxxx;  rfWData = 32'hxxxxxxxx;  nextHi = hi;  nextLo = lo;    uaddA = 32'hxxxxxxxx;  uaddB = 32'hxxxxxxxx;  nextPC = pcPlus4;  syscallFuncCode = 0;    if(cpuRunning) begin  if(iAdd) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SADD;  rfWe = aluOverflow ? 1'b0 : 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iAddu) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_UADD;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSub) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SSUB;  rfWe = aluOverflow ? 1'b0 : 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSubu) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_USUB;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iAnd) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_AND;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iOr) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_OR;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iXor) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_XOR;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iNor) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_NOR;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSlt) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SLES;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSltu) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_ULES;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSll) begin  rfRAddr1 = rt;  rfRAddr2 = 0;  aluA = rfRData1;  aluB = shamt;  aluModeSel = ALU\_SL;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSrl) begin  rfRAddr1 = rt;  rfRAddr2 = 0;  aluA = rfRData1;  aluB = shamt;  aluModeSel = ALU\_SRL;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSra) begin  rfRAddr1 = rt;  rfRAddr2 = 0;  aluA = rfRData1;  aluB = shamt;  aluModeSel = ALU\_SRA;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSllv) begin  rfRAddr1 = rt;  rfRAddr2 = rs;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SL;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSrlv) begin  rfRAddr1 = rt;  rfRAddr2 = rs;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SRL;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iSrav) begin  rfRAddr1 = rt;  rfRAddr2 = rs;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SRA;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = aluR;  end else if (iJr) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  aluA = 0;  aluB = 0;  aluModeSel = ALU\_AND;  nextPC = rfRData1;  end else if (iAddi) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_SADD;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iAddiu) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_UADD;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iAndi) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16UIn = imm;  aluA = rfRData1;  aluB = extend16UOut;  aluModeSel = ALU\_AND;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iOri) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16UIn = imm;  aluA = rfRData1;  aluB = extend16UOut;  aluModeSel = ALU\_OR;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iXori) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16UIn = imm;  aluA = rfRData1;  aluB = extend16UOut;  aluModeSel = ALU\_XOR;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iLw) begin  rfRAddr1 = base;  rfRAddr2 = 0;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_UADD;  dmemAEn = 1'b1;  dmemAAddr = aluR;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = dmemAOut;  end else if (iSw) begin  rfRAddr1 = base;  rfRAddr2 = rt;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_UADD;  dmemAEn = 1'b1;  dmemAWe\_orig = 4'hf;  dmemAAddr = aluR;  dmemAIn = rfRData2;  end else if (iBeq || iBne) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_EQU;  uaddA = pcPlus4;  uaddB = extend16S\_1Out << 2;  if (iBeq)  nextPC = aluR[0] ? uaddR : pcPlus4;  else if (iBne)  nextPC = aluR[0] ? pcPlus4 : uaddR;  end else if (iSlti) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_SLES;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iSltiu) begin  rfRAddr1 = rs;  rfRAddr2 = 0;  extend16S\_1In = imm;  aluA = rfRData1;  aluB = extend16S\_1Out;  aluModeSel = ALU\_ULES;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = aluR;  end else if (iLui) begin  rfRAddr1 = 0;  rfRAddr2 = 0;  rfWe = 1'b1;  rfWAddr = rt;  rfWData = {imm, 16'h0};  end else if (iJ) begin  rfWe = 1'b0; // Do nothing  nextPC = {pc[31:28], index, 2'b0};  end else if (iJal) begin  rfWe = 1'b1;  rfWAddr = 31;  rfWData = pcPlus4;  nextPC = {pc[31:28], index, 2'b0};  end else if (iDiv) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SDIV;    nextHi = aluRX;  nextLo = aluR;  end else if (iDivu) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_UDIV;    nextHi = aluRX;  nextLo = aluR;  end else if (iMult) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SMUL;    nextHi = aluRX;  nextLo = aluR;  end else if (iMul) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_SMUL;    rfWData = aluR;  rfWAddr = rd;  rfWe = 1;    end else if (iMultu) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_UMUL;    nextHi = aluRX;  nextLo = aluR;  end else if (iBgez) begin  rfRAddr1 = rs;  extend16S\_1In = imm;    uaddA = pcPlus4;  uaddB = extend16S\_1Out << 2;  nextPC = rfRData1[31] ? pcPlus4 : uaddR;  end else if (iJalr) begin  rfRAddr1 = rs;  rfWe = 1'b1;  rfWAddr = rd;  rfWData = pcPlus4;  nextPC = rfRData1;  end else if (iLbu) begin  rfRAddr1 = base;  rfWAddr = rt;  rfWe = 1'b1;  extend16S\_1In = imm;    uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAAddr = uaddR;  dmemAEn = 1'b1;  bytePos = {3'b000, dmemAAddr[1:0] ^ {2{BigEndianCPU}}};  extend8UIn = dmemAOut[8 \* bytePos +: 8];  rfWData = extend8UOut;  end else if (iLhu) begin  rfRAddr1 = base;  rfWAddr = rt;  rfWe = 1'b1;  extend16S\_1In = imm;  uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAAddr = uaddR;  dmemAEn = 1'b1;  //bytePos = {3'b000, dmemAAddr[1:0] ^ {BigEndianCPU, 1'b0}};  bytePos = {3'b000, dmemAAddr[1] ^ BigEndianCPU, 1'b0};  extend16UIn = dmemAOut[8 \* bytePos +: 16];  rfWData = extend16UOut;  end else if (iLb) begin  rfRAddr1 = base;  rfWAddr = rt;  rfWe = 1'b1;  extend16S\_1In = imm;    uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAAddr = uaddR;  dmemAEn = 1'b1;  bytePos = {3'b000, dmemAAddr[1:0] ^ {2{BigEndianCPU}}};  extend8SIn = dmemAOut[8 \* bytePos +: 8];  rfWData = extend8SOut;  end else if (iLh) begin  rfRAddr1 = base;  rfWAddr = rt;  rfWe = 1'b1;  extend16S\_1In = imm;    uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAAddr = uaddR;  dmemAEn = 1'b1;  //bytePos = {3'b000, dmemAAddr[1:0] ^ {BigEndianCPU, 1'b0}};  bytePos = {3'b000, dmemAAddr[1] ^ BigEndianCPU, 1'b0};  extend16S\_2In = dmemAOut[8 \* bytePos +: 16];  rfWData = extend16S\_2Out;  end else if (iSb) begin  rfRAddr1 = base;  rfRAddr2 = rt;  extend16S\_1In = imm;    uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAEn = 1'b1;  dmemAAddr = uaddR;  bytePos = {3'b000, dmemAAddr[1:0] ^ {2{BigEndianCPU}} };  dmemAWe\_orig[0] = (bytePos[1:0] == 2'h0);  dmemAWe\_orig[1] = (bytePos[1:0] == 2'h1);  dmemAWe\_orig[2] = (bytePos[1:0] == 2'h2);  dmemAWe\_orig[3] = (bytePos[1:0] == 2'h3);  aluA = rfRData2;  aluB = bytePos[1:0] << 3;  aluModeSel = ALU\_SL;    dmemAIn = aluR;  end else if (iSh) begin  rfRAddr1 = base;  rfRAddr2 = rt;  extend16S\_1In = imm;    uaddA = extend16S\_1Out;  uaddB = rfRData1;  dmemAAddr = uaddR;  //bytePos = {3'b000, dmemAAddr[1:0] ^ {BigEndianCPU, 1'b0}};  bytePos = {3'b000, dmemAAddr[1] ^ BigEndianCPU, 1'b0};  dmemAEn = 1'b1;  dmemAWe\_orig[0] = (bytePos[1] == 2'h0);  dmemAWe\_orig[1] = (bytePos[1] == 2'h0);  dmemAWe\_orig[2] = (bytePos[1] == 2'h1);  dmemAWe\_orig[3] = (bytePos[1] == 2'h1);  aluA = rfRData2;  aluB = bytePos[1] ? 16 : 0;  aluModeSel = ALU\_SL;    dmemAIn = aluR;  end else if (iBreak) begin  cp0Exception = `ENABLE;  cp0Cause = {25'h0, `BREAKCAUSE, 2'h0};  nextPC = nextPC\_cp0;  end else if (iSyscall) begin  rfRAddr2 = 2;  cp0Exception = `ENABLE;  cp0Cause = {25'h0, `SYSCALLCAUSE, 2'h0};  nextPC = nextPC\_cp0;  syscallFuncCode = rfRData2[7:0];  end else if (iEret) begin  nextPC = cp0ExecAddr;  end else if (iMfhi) begin  rfWAddr = rd;  rfWe = 1'b1;  rfWData = hi;  end else if (iMflo) begin  rfWAddr = rd;  rfWe = 1'b1;  rfWData = lo;  end else if (iMthi) begin  rfRAddr1 = rs;  nextHi = rfRData1;  end else if (iMtlo) begin  rfRAddr1 = rs;  nextLo = rfRData1;  end else if (iMfc0) begin  cp0Addr = rd;  rfWe = `ENABLE;  rfWAddr = rt;  rfWData = cp0RData;  end else if (iMtc0) begin  rfRAddr1 = rt;  cp0Addr = rd;  cp0WData = rfRData1;  end else if (iClz) begin  rfRAddr1 = rs;  aluA = rfRData1;  aluB = 0;  aluModeSel = ALU\_CLZ;  rfWe = 1;  rfWAddr = rd;  rfWData = aluR;  end else if (iTeq) begin  rfRAddr1 = rs;  rfRAddr2 = rt;  aluA = rfRData1;  aluB = rfRData2;  aluModeSel = ALU\_EQU;  if(aluR[0]) begin  cp0Cause = {25'h0, `TEQCAUSE, 2'h0};  cp0Exception = `ENABLE;  end  trap = aluR[0];  nextPC = nextPC\_cp0;  end  end  end  ///////////////  /// Trapezohedron - CP0    Trapezohedron cp0(  .clk(clk),  .rst(reset),  .cpuPaused(cpuPaused),  .mfc0(iMfc0),  .mtc0(iMtc0),  .pc(pc),  .nextPC(nextPC\_cp0),  .addr(cp0Addr),  .data(cp0WData),  .exception(cp0Exception),  .eret(iEret),  .cause(cp0Cause),  .intr(cp0Intr),  .PC0\_out(cp0RData),  .status(cp0Status),  .epc\_out(cp0ExecAddr)  );  endmodule |

# 测试/调试过程

* 首先，编写CPU的各个模块。主模块命名为Nyarlathotep，存在nyarlathotep.v文件下；CP0模块命名为Trapezohedron，存在Trapezohedron.v文件下。使用MIPS246网站提供的“54指令CPU测试文件”进行测试，Testbench代码如下：

|  |
| --- |
| module sccomp\_dataflow\_oj\_tb(  );  reg clk\_in;  wire clk\_div;  reg rst;  wire [31:0] inst;  wire [31:0] pc;  wire [31:0] addr;    reg [31:0] instPrev;  reg [31:0] pcPrev;  wire cpuRunning;  integer file\_output;  reg cpuEna\_n = 1'b0;  sccomp\_dataflow uut(  .clk\_in(clk\_in),  .clk\_afterDiv(clk\_div),  .cpuEna\_n(cpuEna\_n),  .reset(rst),  .inst(inst),  .pc(pc),  .addr(addr),  .cpuRunning(cpuRunning)  );    initial begin  clk\_in = 0;  rst = 1;  file\_output = $fopen("results.txt");  #0  $readmemh("../../../Test/54\_div.hex.txt", uut.imem.array\_reg);  #5  rst = 0;  end        always #5 clk\_in = ~clk\_in;    reg cpuDonotRecordFirst = 0;    always @(negedge clk\_div) begin  instPrev <= inst;  pcPrev <= pc;  if (cpuRunning) begin  if(inst[0] === 1'bx)  cpuEna\_n <= 1'b1;  if(cpuDonotRecordFirst) begin  $fdisplay(file\_output,"pc: %h", pcPrev - 32'h00400000);  $fdisplay(file\_output,"instr: %h", instPrev);      $fdisplay(file\_output,"regfile0: %h",uut.sccpu.cpu\_ref.array\_reg[0]);  $fdisplay(file\_output,"regfile1: %h",uut.sccpu.cpu\_ref.array\_reg[1]);  $fdisplay(file\_output,"regfile2: %h",uut.sccpu.cpu\_ref.array\_reg[2]);  $fdisplay(file\_output,"regfile3: %h",uut.sccpu.cpu\_ref.array\_reg[3]);  $fdisplay(file\_output,"regfile4: %h",uut.sccpu.cpu\_ref.array\_reg[4]);  $fdisplay(file\_output,"regfile5: %h",uut.sccpu.cpu\_ref.array\_reg[5]);  $fdisplay(file\_output,"regfile6: %h",uut.sccpu.cpu\_ref.array\_reg[6]);  $fdisplay(file\_output,"regfile7: %h",uut.sccpu.cpu\_ref.array\_reg[7]);  $fdisplay(file\_output,"regfile8: %h",uut.sccpu.cpu\_ref.array\_reg[8]);  $fdisplay(file\_output,"regfile9: %h",uut.sccpu.cpu\_ref.array\_reg[9]);    $fdisplay(file\_output,"regfile10: %h",uut.sccpu.cpu\_ref.array\_reg[10]);  $fdisplay(file\_output,"regfile11: %h",uut.sccpu.cpu\_ref.array\_reg[11]);  $fdisplay(file\_output,"regfile12: %h",uut.sccpu.cpu\_ref.array\_reg[12]);  $fdisplay(file\_output,"regfile13: %h",uut.sccpu.cpu\_ref.array\_reg[13]);  $fdisplay(file\_output,"regfile14: %h",uut.sccpu.cpu\_ref.array\_reg[14]);  $fdisplay(file\_output,"regfile15: %h",uut.sccpu.cpu\_ref.array\_reg[15]);  $fdisplay(file\_output,"regfile16: %h",uut.sccpu.cpu\_ref.array\_reg[16]);  $fdisplay(file\_output,"regfile17: %h",uut.sccpu.cpu\_ref.array\_reg[17]);  $fdisplay(file\_output,"regfile18: %h",uut.sccpu.cpu\_ref.array\_reg[18]);  $fdisplay(file\_output,"regfile19: %h",uut.sccpu.cpu\_ref.array\_reg[19]);    $fdisplay(file\_output,"regfile20: %h",uut.sccpu.cpu\_ref.array\_reg[20]);  $fdisplay(file\_output,"regfile21: %h",uut.sccpu.cpu\_ref.array\_reg[21]);  $fdisplay(file\_output,"regfile22: %h",uut.sccpu.cpu\_ref.array\_reg[22]);  $fdisplay(file\_output,"regfile23: %h",uut.sccpu.cpu\_ref.array\_reg[23]);  $fdisplay(file\_output,"regfile24: %h",uut.sccpu.cpu\_ref.array\_reg[24]);  $fdisplay(file\_output,"regfile25: %h",uut.sccpu.cpu\_ref.array\_reg[25]);  $fdisplay(file\_output,"regfile26: %h",uut.sccpu.cpu\_ref.array\_reg[26]);  $fdisplay(file\_output,"regfile27: %h",uut.sccpu.cpu\_ref.array\_reg[27]);  $fdisplay(file\_output,"regfile28: %h",uut.sccpu.cpu\_ref.array\_reg[28]);  $fdisplay(file\_output,"regfile29: %h",uut.sccpu.cpu\_ref.array\_reg[29]);    $fdisplay(file\_output,"regfile30: %h",uut.sccpu.cpu\_ref.array\_reg[30]);  $fdisplay(file\_output,"regfile31: %h",uut.sccpu.cpu\_ref.array\_reg[31]);  end else  cpuDonotRecordFirst <= 1;  end else  cpuDonotRecordFirst <= 0;  end  endmodule |

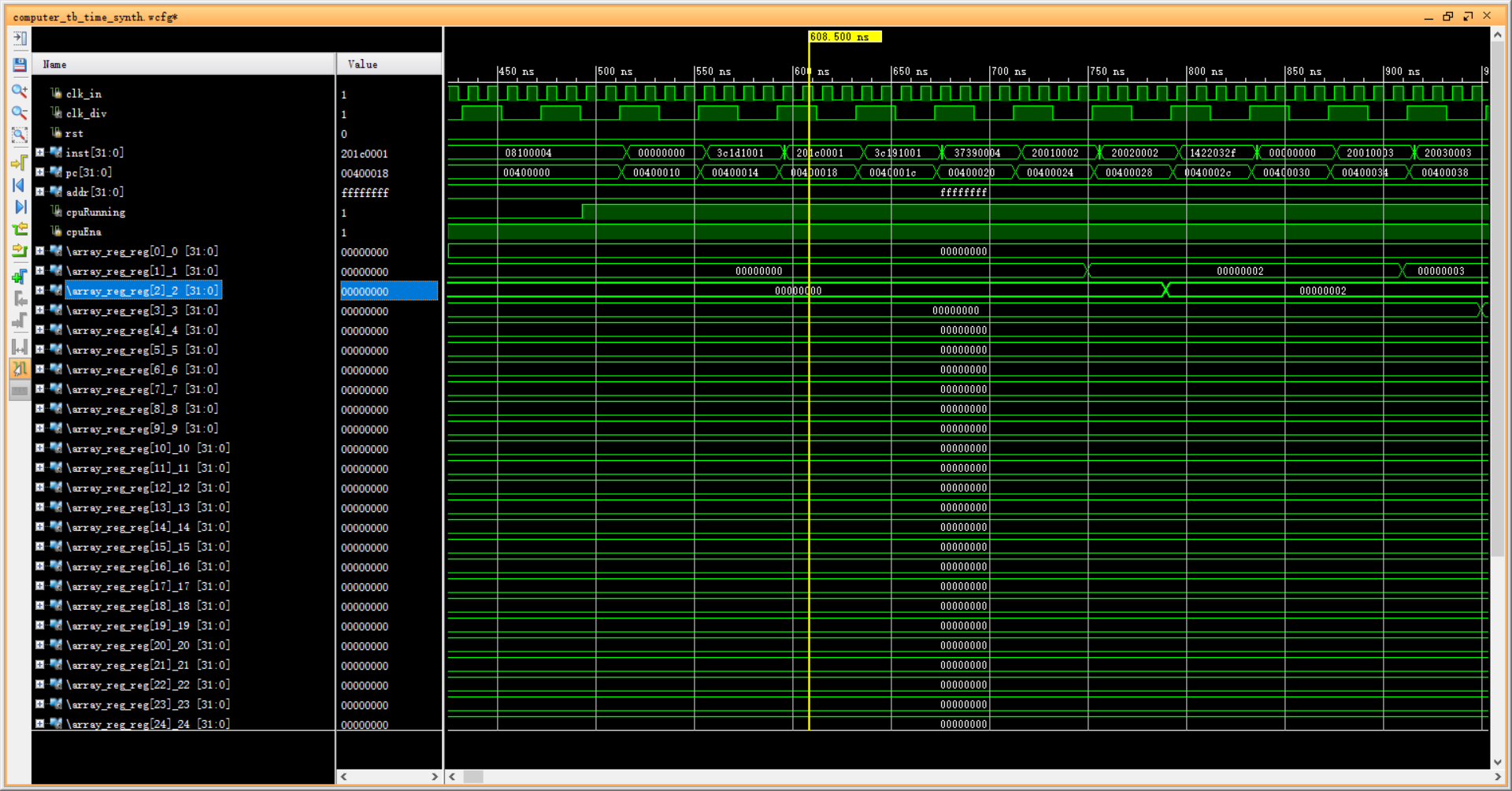


采用上述文件中的\*.hex.txt加载入IMEM进行测试；

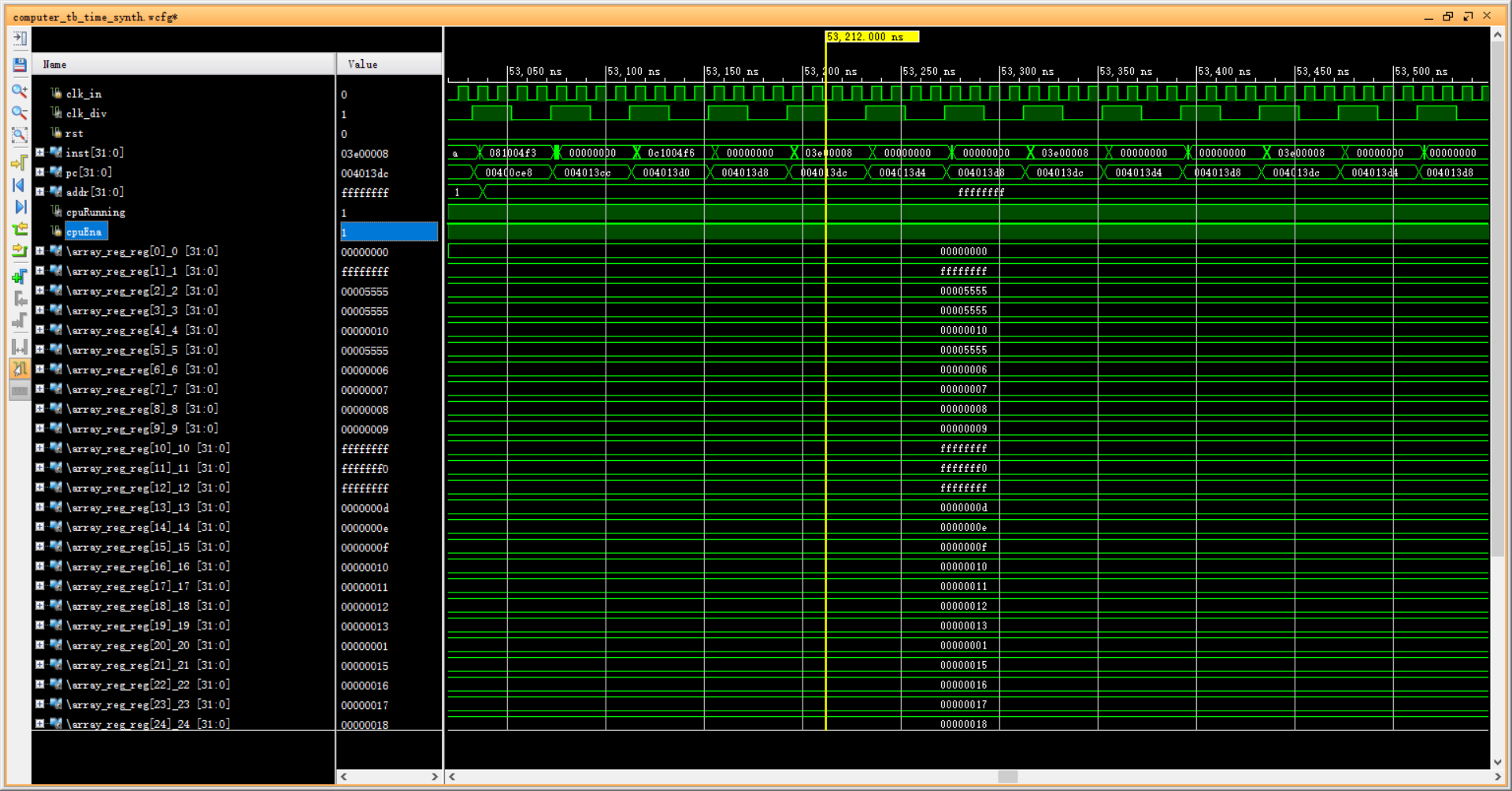
* + - 加载mips\_54\_mars\_simulate\_student\_ForWeb.coe，使用其中的汇编指令，综合后仿真运行设计好的CPU；同时将它反汇编为mips\_54\_mars\_simulate\_student\_ForWeb.asm，加载入MARS（MIPS模拟器）。同时运行两者，观察两个CPU的状态，看是否相同。若相同，则这个CPU的设计达到了目标。
    - 在MIPS246网站上进行在线检测。
    - 将CPU下板，运行mips\_54\_mars\_board\_switch\_student.coe。看运行结果是否与老师给出的结果一致。
    - 将CPU与其他部件结合下板，运行应用程序的汇编二进制文件，看是否完成应用功能。

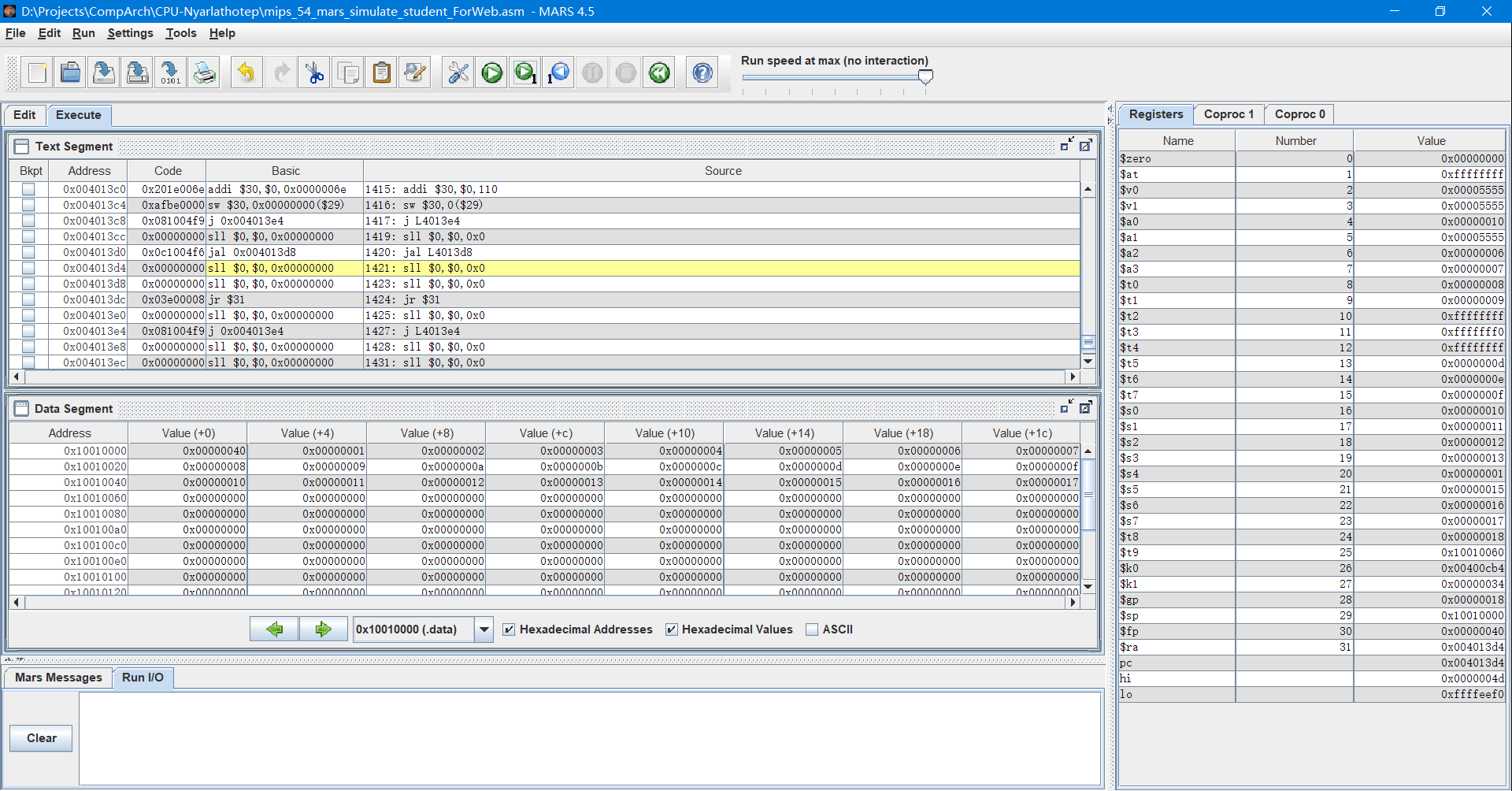
# 实验结果分析

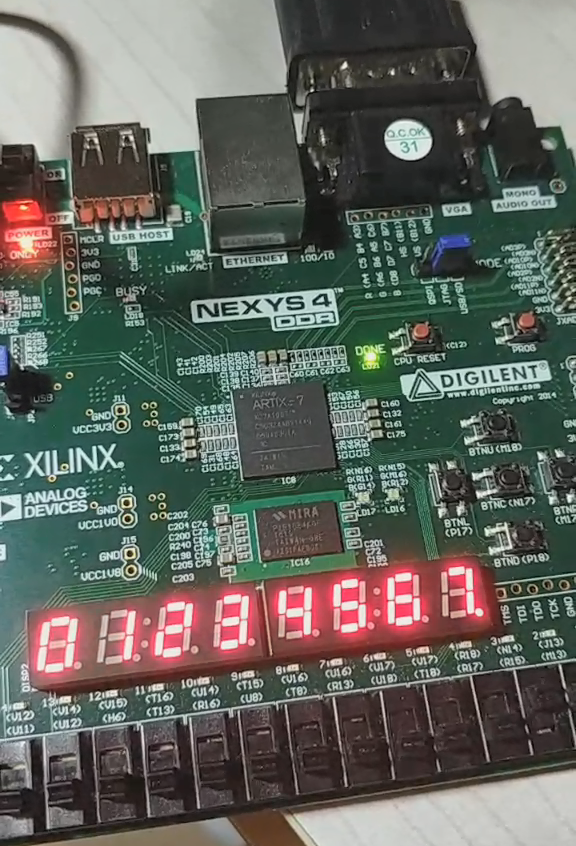
* + - 本地测试，测试输出结果均与\*.result.txt符合。
    - 综合后仿真测试：CPU在25MHz的仿真时钟信号下正常运行。



* + - MARS在运行反汇编后的代码时，运行完成后代码在0x004013d4和0x004013dc之间进入循环；而综合后仿真的结果也达到了这个状态。据观察，该汇编程序可以很好地检测CPU各条指令的运行情况，若有不正常的结果产生，会将程序流程导向一个表示错误的死循环。而本CPU没有进入任何一个错误死循环，说明CPU很好的完成了任务。
    - 在线检测达到AC。
    - 下板正常显示滚动的1~F和指定的数字25EE38B1。
    - 带部件和应用下板后，工作正常。参见随附的视频。







# 结论

可以使用Verilog HDL和IP核完成一个单周期54条指令MIPS CPU的设计，以及完成一个电子公告牌应用的设计。

# 心得体会及建议

本次大作业的制作过程耗时极长，过程复杂，锻炼了我的硬件思维和硬件设计能力，也让我充分了解到了CPU的运行机制。