计算机系统综合设计

设计报告

组长：鲁彬彬

成员：门帅秋

汪　磊

王　平

田志鹏

东南大学计算机科学与工程学院

二００九年一月

设计名称					
完成时间		验收时间		成绩	
本组成员情况					
姓　名	学　号	承　担　的　任　务			个人成绩
鲁彬彬	09005342	可视化界面、词法分析、打包整合、综合测试			
门帅秋	09005345	语法分析、代码生成、多错误、综合测试			
汪磊	09005314	LED、CTC、PWM、KeyPad、综合测试			
田志朋	09005313	UART、WTD、综合测试			
王平	09005341	CPU、中断、I/O 接口、综合测试			

注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明本组设计的成果和特色，能够反应小组中每个人的工作。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分之一，因此要在报告中明确标明每个模块的设计者。

设计报告最后一页是验收表和教师综合评价，请大家打印报告的时候将此页一并打印装订。

本组设计的功能描述（含所有实现的模块的功能）

硬件功能:

1. 整体描述

处理器采用哈佛结构，有独立的 2KB 指令存储器和 2KB 数据存储器。

实现的指令共 37 条:

- 算术指令－add, addu, addi, addiu, sub, subu, mul, mulu, mfhi, mflo，mthi, mtlo
- 逻辑指令－and, andi, or, ori, xor, xori, nor, sll, srl, sra, sllv, srlv, srav
- 数据传送指令－lw, sw, lui
- 比较、条件转移指令－beq, bne, slt, slti, sltu, sltiu
- 无条件转移指令－j, jr, jal

处理器采用流水线技术，共五级流水（取指，译码，执行，访存，寄存器写回），采用数据转发技术解决数据相关性，提高了流水线的效率。同时也加入了简单的分支预测功能，极大的提高了跳转指令执行的效率。

在中断处理方面，CPU 支持中断——中断 0 和中断 1，中断 0 优先级大于中断 1，不支持同级中断嵌套。

该系统共有 6 个外围设备，分别是：LED 数码管、4×4 键盘、定时/计数器、PWM 控制器、UART 串行通信控制器、看门狗控制。

2. 各模块介绍:

1） **sys ：** 整个系统的顶层文件，将各个子模块，组合起来。

2） **IFetch32**：取指单元，主要功能是选择正确的 PC 值（因为执行单元和分支预测单元以及它本身都会提供下一条指令的 PC 值，再考虑中断就更多了，所以 PC 值的选取是很重要的，直接关系到流水线执行的正确性），然后到指令存储器中取出指令，送到译码单元。

3） **OffshootPredict32:** 分支预测处理模块，向取指单元提供预测的 PC 值（如果在分支预测缓存中存在记录的话）。

4） **Idecode32:** 有两个重要的功能。首先，将 IFetch32 单元传来的指令进行译码，生成控制信号，并且从寄存器组中取得相应的数据。其次，负责寄

存器写回的操作（流水线的最后一步）。

5） **Executs32:** 执行单元模块。完成逻辑运算、算术运算（注意乘法运算是要 7 个周期完成的，因此要阻塞流水线）、移位运算、比较转移的 PC 值计算。

6） **Mult32:** 乘法器：利用 7 个周期完成一次乘法操作，首先，进行原码转换（如果必要的话），之后的四个周期完成此次累加计算，接着将前四次的结果进一步累计，得到结果。最后复位相关的信号。

7） AddressCheck32: 地址检查，检查预取的指令地址和计算出来的下一条指令地址是否相同，并对跳转指令进行地址检查，根据地址检测结果输出相应的控制信号（必要时阻塞流水）。

8） **DataTransmit32:** 数据转发单元，解决数据相关性（LW 指令要结合 ConflictCheck32 模块）。

9） **DataSelect32:** 配合 DataTransmit32 单元，选择正确的数据，完成数据转发。

10）**ConflictCheck32:** LW 指令的处理数据相关处理，下一条指令用到当前 LW 指令的结果，Block 置 2 阻塞一个时钟周期。等到数据从数据 Ram 或 IO 端口中取出后继续。

11）**dmemory32**：存储单元模块。实际完成对数据存储器(RAM)的读写操作。

12）**memorio32**：该模块功能作用是给 cpu 添加 io 接口，当读写 mem 时给 RAM 输出存储器读写信号，当读写 io 时给外设输出 io 读写信号，提供数据线和地址线与外设连接.

13）**ioread32**：I/O 设备数据选择器，根据控制信号，从不同的 IO 设备读取数据

14）**UART**：简单的串行通信模块。负责控制将 CPU 来的 8 位数据并转串，然后按照异步串行通信数据格式输出，将串口来的 8 位串行数据串转并，并在 CPU 请求的时候输入给 CPU。

15）**wtd16**：看门狗模块。内含一个 16 位定时器，系统复位后计数值为 FFFFH，之后每时钟计数值减 1，当减到 0 的时候，向 CPU 发 4 个时钟周

期的 RESET 信号，同时计数值恢复到 FFFFH 并继续计数。通过软件将看门狗计数器初始化为 FFFFH,然后不断地定期写看门狗端口来复位看门狗，使计数器重新从 FFFFH 开始计数。增加看门狗电路后，CPU 的 RESET 输入脚是系统复位信号和看门狗发出的复位信号的组合。

16）**LED**：一开始采用硬布线方式，将 4 位寄存器的输出端与 7 段共阳极数码管的输入端组合相连。但后来连上 CPU 以后发现拖了 CPU 的主频，故而放弃。这个部分对书上的代码稍做改动，加入了 CLK 信号，以确保不会在数据并未准备好时，向寄存器写数据。

17）**KEY**：使用一个 4 位移位计数器来处理键盘信号的频繁跳变。连续 8 个周期检测到列线有 0，才进行逐行扫描。其余部分均使用书上的代码。

18）**CTC**：这个部分书上的代码有问题。在计数的时候，执行读状态口的操作会出错。我把对状态口的所有操作都放在了 CLK 下降沿所触发的进程中。这样做节省了 4 个临时状态寄存器，也避免了错误。但需要添加一个寄存器来保存前一个周期时的计数值，用来维护循环计数时的计数到零标志位。

19）**PWM**：这个模块基本使用了书上的代码。我只是把计数的赋值操作改成了非阻塞的。消耗逻辑单元数奇迹般地从 92 个锐减到 72 个，频率也提高了一些。

软件功能:

1. 整体描述

汇编器采用 Java 作为主要开发语言，使用 Qt4、qtjambi 绘制图形界面。

开发环境：Eclipse

2. 功能模块

- 可视化编辑器 Visual Editor

Visual Editor 由代码编辑器 Code Editor、语法高亮 Highlighter、启动屏幕 Splash Screen 组成;

- 汇编模块 MIPS32 Assembler

MIPS32 Assembler 由 Token 管理器、词法分析器、语法分析器、异常控制、代码生成器组成。

本组设计的主要特色

硬件设计

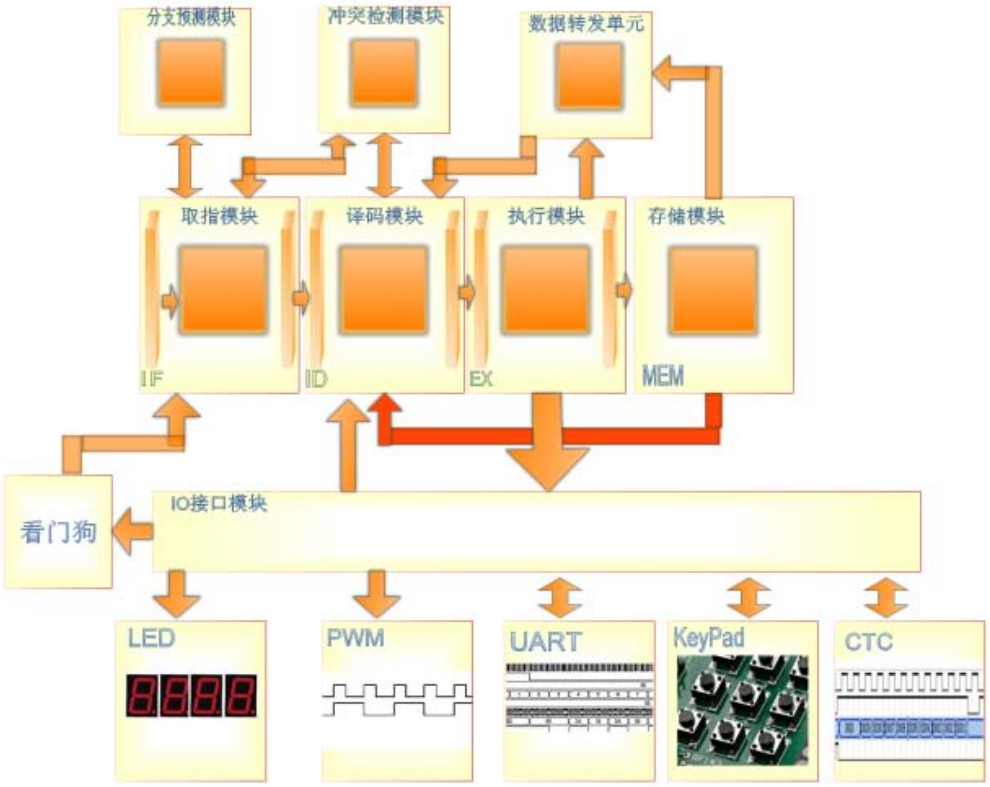
1. CPU 采用五级流水结构，提高资源利用率
2. 利用分支预测技术提高分支指令的定向准确率
3. 采用数据转发技术解决流水线中数据相关性问题
4. 增加了乘法指令和硬件实现的乘法器
5. 增加了键盘防抖，提高键盘稳定性

软件设计

1. 使用 JAVA 语言开发，方便移植，具有跨平台的可能性。
2. 遵循绿色软件思想，无需安装与卸载。
3. 可视化界面（语法高亮、行号、多文件）
4. 精确的错误定位（精确到行号、列号）
5. 支持多多错误检测

本组设计的体系结构

硬件系统结构图



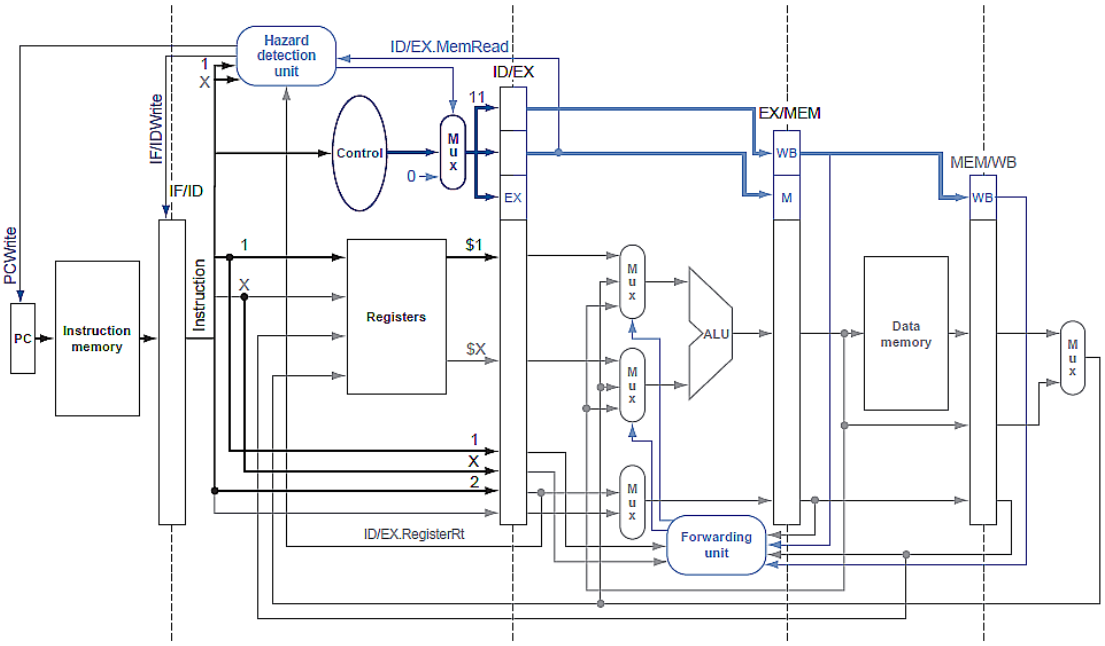
本组设计中各个部件的设计与特色概述

硬件部分

CPU:

由 10 大模块构成，分别是取指模块、译码模块、执行模块、存储模块、乘法器模块、分支预测模块、数据转发模块、数据选择模块（2 个）、地址检查模块。

流水 CPU 的基本工作原理图（省略了分支预测模块）:



WTD:

看门狗模块。内含一个 16 位定时器，系统复位后计数值为 FFFFH，之后每时钟计数值减 1，当减到 0 的时候，向 CPU 发 4 个时钟周期的 RESET 信号，同时计数值恢复到 FFFFH 并继续计数。通过软件将看门狗计数器初始化为 FFFFH,然后不断地定期写看门狗端口来复位看门狗，使计数器重新从 FFFFH 开始计数。增加看门狗电路后，CPU 的 RESET 输入脚是系统复位信号和看门狗发出的复位信号的组合。

UART:

串行通信模块。发送器要对外部时钟 XTAL 按照波特率要求进行分频，。接收器接收数据的采用率是波特率的 16 倍（也由 XTAL 分频得到）。采用中间值采用的原则，也就是一个数据位占 16 个接收时钟宽度，在第 8 个时钟的时候采样。数据格式固定为一位起始位 0,16 位数据位,暂无校验位,1 位停止位 1,若干空闲位 1 串行输出线空闲状态为 1。

软件部分

汇编器

汇编器的词法、语法分析程序基于 LL(1)文法

- 支持 MIPS32 的 37 条指令（31 条基本指令+6 条专用乘法指令）
- 所有立即数均支持十六进制与十进制表示方法
- 32 个寄存器同时支持序号表示方法和别名表示方法
- 支持空指令 NOP
- 支持标号、子程序调用
- 错误定位到行号、列号
- 对错误给出建议性修正
- 多个错误检测支持

可视化编辑器

可视化编辑器使用 Qt4 开发:

- 多文件视图
- 支持新建、打开、剪切、复制、粘贴、撤销、重做、保存、打印
- 代码语法高亮
- 行号显示
- 浮动窗口显示汇编信息（ROM、RAM）

设计与实现

词法解析：基于 LL(1)文法分析，词法规则示例如下：

```
< REG_0:      "$0"  | "$zero"   >
< REG_1:      "$1"  | "$at"     >
< REG_2:      "$2"  | "$v0"     >
< REG_3:      "$3"  | "$v1"     >
< ADD:        "add"             >
< ADDU:       "addu"            >
< SUB:        "sub"             >
< NUMBER:     ["0"-"9"] | ["1"-"9"](["0"-"9"])*  >
< HEX_NUM:    (["A"-"F","a"-"f","0"-"9"])+ "h"   >
< IDENTIFIER: ["A"-"Z","a"-"z"](["A"-"Z","a"-"z","0"-"9", "_"])* >
```

语法分析：词法规则同时包含动作，以 ADD 为例：

```
/* Operation : add */
Instruction op_add() throws Exception:
{
    Instruction inst = new Instruction();
    String machine_code = "000000";
    String shamt = "00000";
    String func = "100000";
    Register rs, rt, rd;
    Token t_op;
}
{
    t_op = <ADD> rd = register() <COMMA> rs = register()
    <COMMA> rt = register()
    {
        machine_code += rs.getRegisterCode();
        machine_code += rt.getRegisterCode();
        machine_code += rd.getRegisterCode();
        machine_code += shamt;
        machine_code += func;
        inst.setMachineCode(machine_code);
        inst.setCode(t_op.image + " " + rd.getRegisterName()
        + "," + rs.getRegisterName() + "," + rt.getRegisterName());
        return inst;
    }
}
```

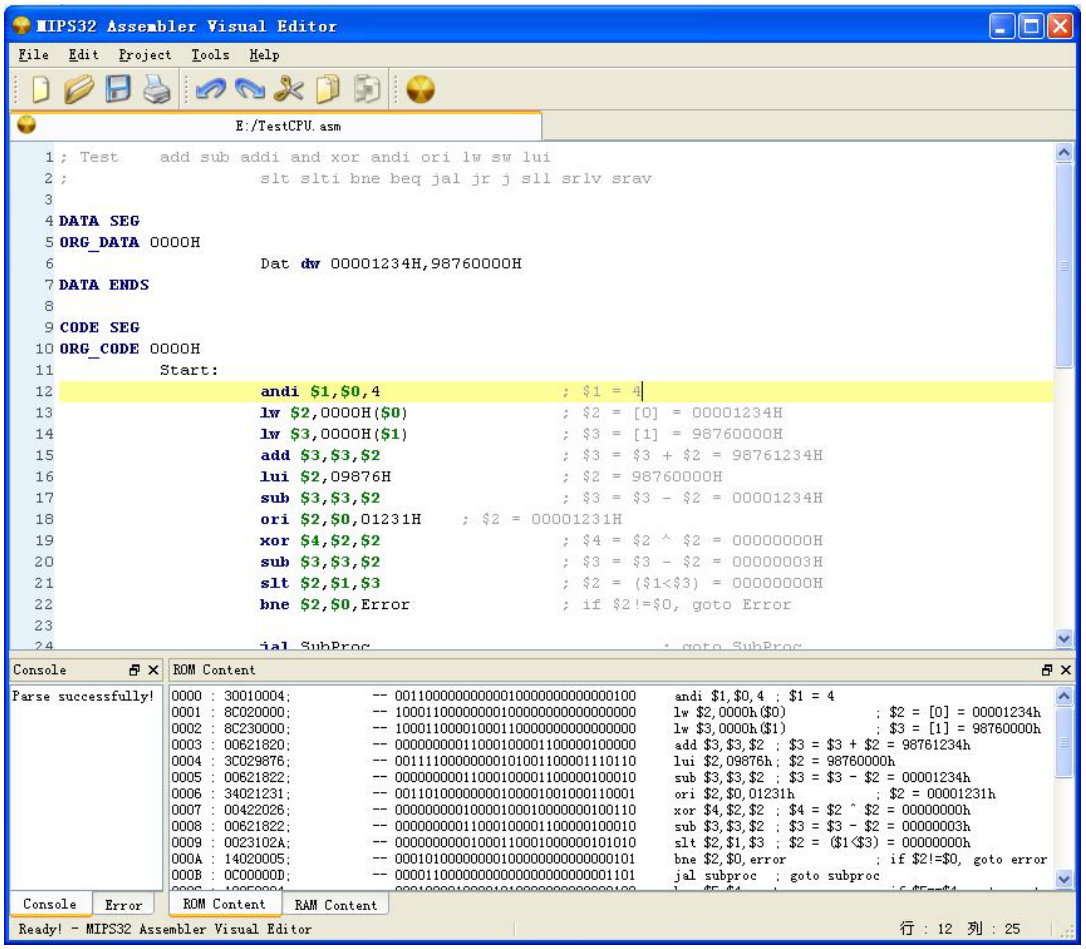
子程序调用：由于子程序的标号在程序最末端，因此需预先扫描代码段的所有标号，记录其对应的地址，在第二次扫描时，查找标号对应的地址并进行替换；

对错误给出建议性修正：处理指令行时，根据生成的状态机，当发生错误时，给出状态机所有可能的下一步转移，从而提出建议性的修改意见；

多错误检测：当某行指令解析到错误时，将该错误记录在容器中，跳过该指令行，解析下一段代码，当所有代码解析完毕时，依次显示所有的错误。可能产生的错误的种类有：词法错误、语法错误、代码生成阶段错误（标号不存在、地址越界等），因此容纳异常的容器单元类型为各类错误的基类；

代码语法高亮：通过 Qt4 的 QTextPlainEdit 控件，当键盘事件触发时，用正则表达式匹配所有语法关键字，达到关键字高亮的目的；

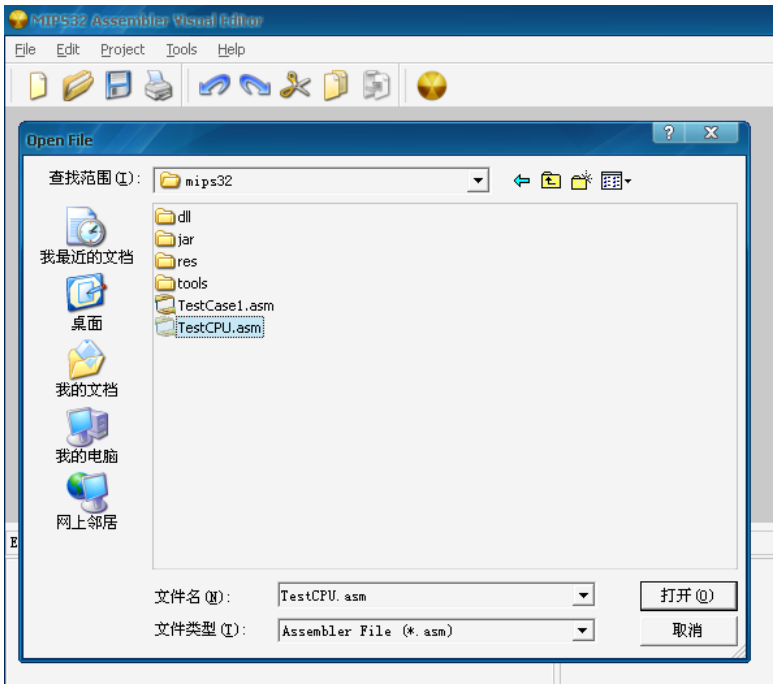
行号：获取用户视图，判断 QTextPlainEdit 控件的可见区域，根据行高计算可见行的开始值和终止值，在 QTextPlainEdit 左侧绘制矩形背景色，并绘制文本（数字行号）。



本组设计的 MiniSys 汇编程序使用手册

运行环境：JDK 1.6。

一. **文件操作：**扩展名提示。



1. File 菜单可提供文件新建，打开，保存的操作，如图 1。
2. Edit 菜单可提供撤销，重做，剪切，复制和粘贴的操作，如图 2。
3. Project 菜单提供建立操作，从.asm 文件到 rom 和 ram 的.mif 文件，如图 3。
4. Tool 菜单里的 options 选项提供修改字体的类型和大小，如图 4。

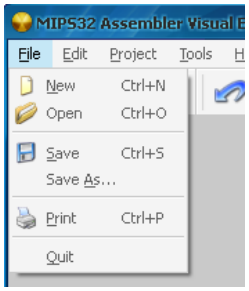


图 1

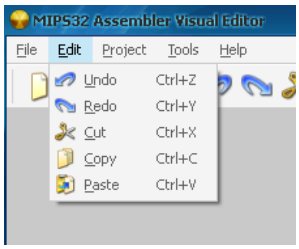


图 2

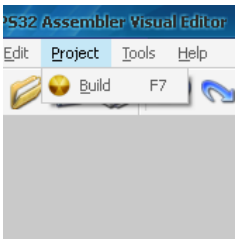


图 3

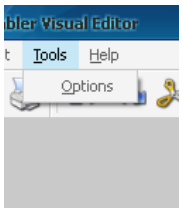


图 4

二. **编辑环境：**

1. 可同时编辑多个文件，如图 5 中的新建了“TestCPU.asm”和“TestCase1.asm”。
2. 能够显示当前光标的位置和行号，如图 5 中的“当前位置”和“行号”。
3. 进行汇编时显示相应的汇编语句和机器码，如图 5 中“ROM Content”里的

“机器码”和“汇编语句”；而“RAM Content”则显示相应的 Ram 的内容，如图 6。

4. 相关提示语，如果汇编语句都正确，则如图 5 中的“console”显示“Parse successfully!”。

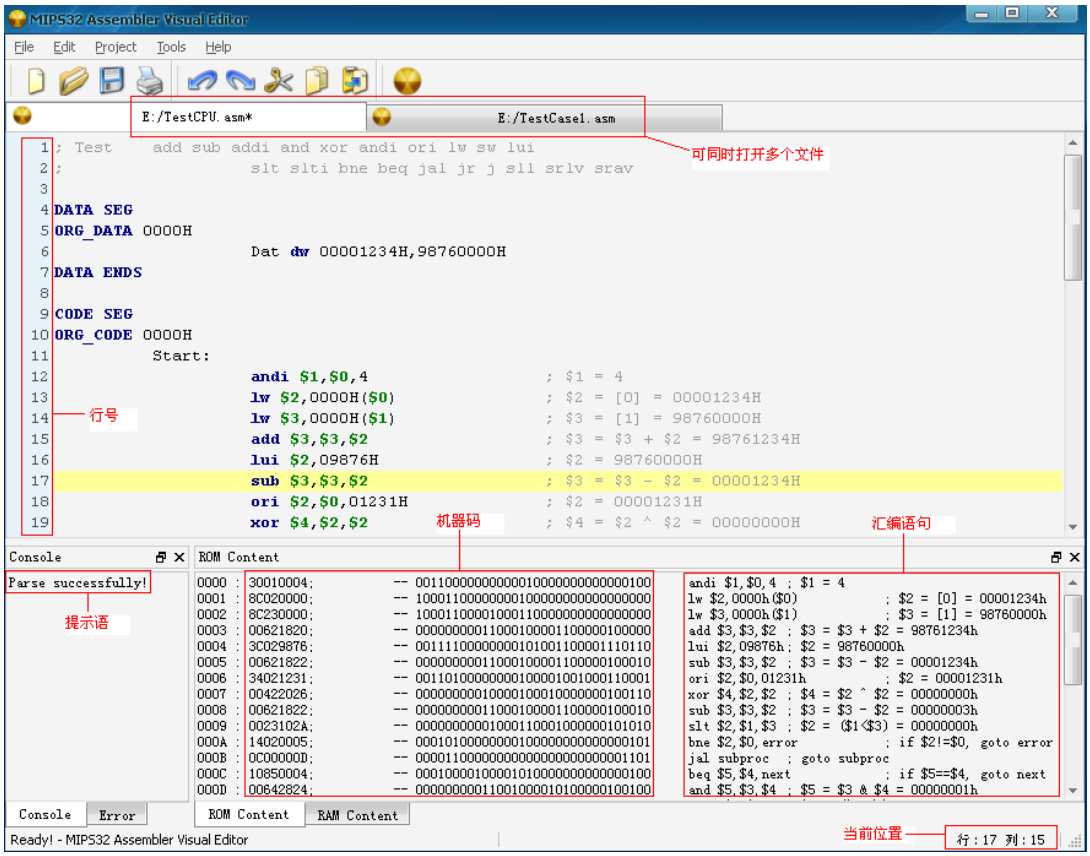


图 5

三. 多错误提示与窗口拖动:

1. Console，Error，ROM 和 RAM Content 均为活动窗口。

2. 如果汇编内容有错误，汇编器支持多错误提示。如图 6，汇编 TestCPU.asm 点击“Parse Exception at Row: 13, col:41”，相应的“Console”窗口会显示“Encountered " <COMMENT> "; $2 = [0] = 00001234h "" at line 13, column 41. Was expecting: ")" ...”，提示第 13 行 41 列缺少“)”；同样，如图 7，点击“Parse Exception at Row: 15, col:27”相应的“Console”窗口会显示“Encountered " <NUMBER> "2 "" at line 15, column 27. Was expecting one of: <REG_0><REG_1> ...<REG_31> ...”，提示第 15 行 17 列的“2”不正确，并且建议正确的应该是 32 个寄存器中的某一个。

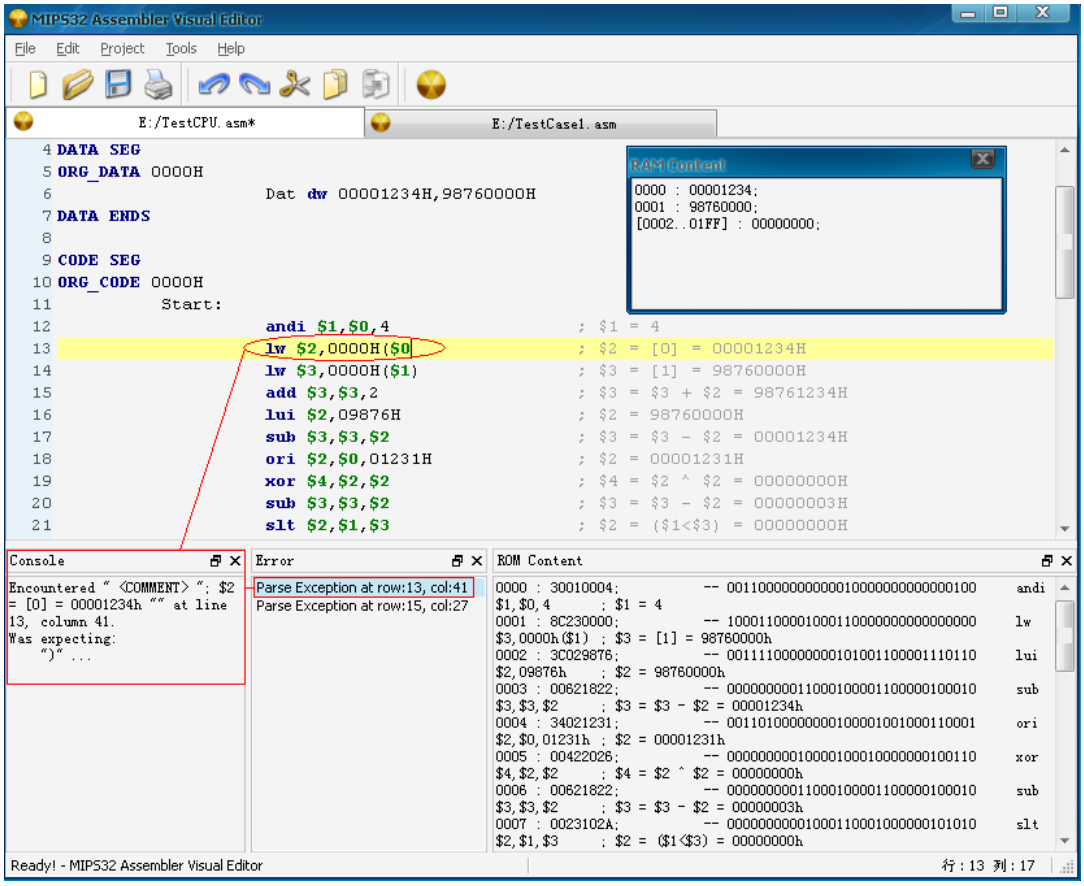

MIPS32 Assembler Visual Editor
File Edit Project Tools Help
E:/TestCPU.asm*
E:/TestCase1.asm
4 DATA SEG
5 ORG_DATA 0000H
6 Dat dw 00001234H,98760000H
7 DATA ENDS
8
9 CODE SEG
10 ORG_CODE 0000H
11 Start:
12 andi $1,$0,4 ; $1 = 4
13 lw $2,0000H($0 ; $2 = [0] = 00001234H
14 lw $3,0000H($1) ; $3 = [1] = 98760000H
15 add $3,$3,2 ; $3 = $3 + $2 = 98761234H
16 lui $2,09876H ; $2 = 98760000H
17 sub $3,$3,$2 ; $3 = $3 - $2 = 00001234H
18 ori $2,$0,01231H ; $2 = 00001231H
19 xor $4,$2,$2 ; $4 = $2 ^ $2 = 00000000H
20 sub $3,$3,$2 ; $3 = $3 - $2 = 00000003H
21 slt $2,$1,$3 ; $2 = ($1<$3) = 00000000H
RAM Content
0000 : 00001234;
0001 : 98760000;
[0002..01FF] : 00000000;
Console
Encountered " <COMMENT> ": $2 = [0] = 00001234h "" at line 13, column 41.
Was expecting:
")" ...
Error
Parse Exception at row:13, col:41
Parse Exception at row:15, col:27
ROM Content
0000 : 30010004; -- 00110000000000010000000000000100 andi $1,$0,4 ; $1 = 4
0001 : 8C230000; -- 10001100001000110000000000000000 lw $3,0000h($1) ; $3 = [1] = 98760000h
0002 : 3C029876; -- 00111100000000101001100001110110 lui $2,09876h ; $2 = 98760000h
0003 : 00621822; -- 00000000011000100001100000100010 sub $3,$3,$2 ; $3 = $3 - $2 = 00001234h
0004 : 34021231; -- 00110100000000100001001000110001 ori $2,$0,01231h ; $2 = 00001231h
0005 : 00422026; -- 00000000010000100010000000100110 xor $4,$2,$2 ; $4 = $2 ^ $2 = 00000000h
0006 : 00621822; -- 00000000011000100001100000100010 sub $3,$3,$2 ; $3 = $3 - $2 = 00000003h
0007 : 0023102A; -- 00000000001000110001000000101010 slt $2,$1,$3 ; $2 = ($1<$3) = 00000000h
Ready! - MIPS32 Assembler Visual Editor
行：13 列：17

图 6

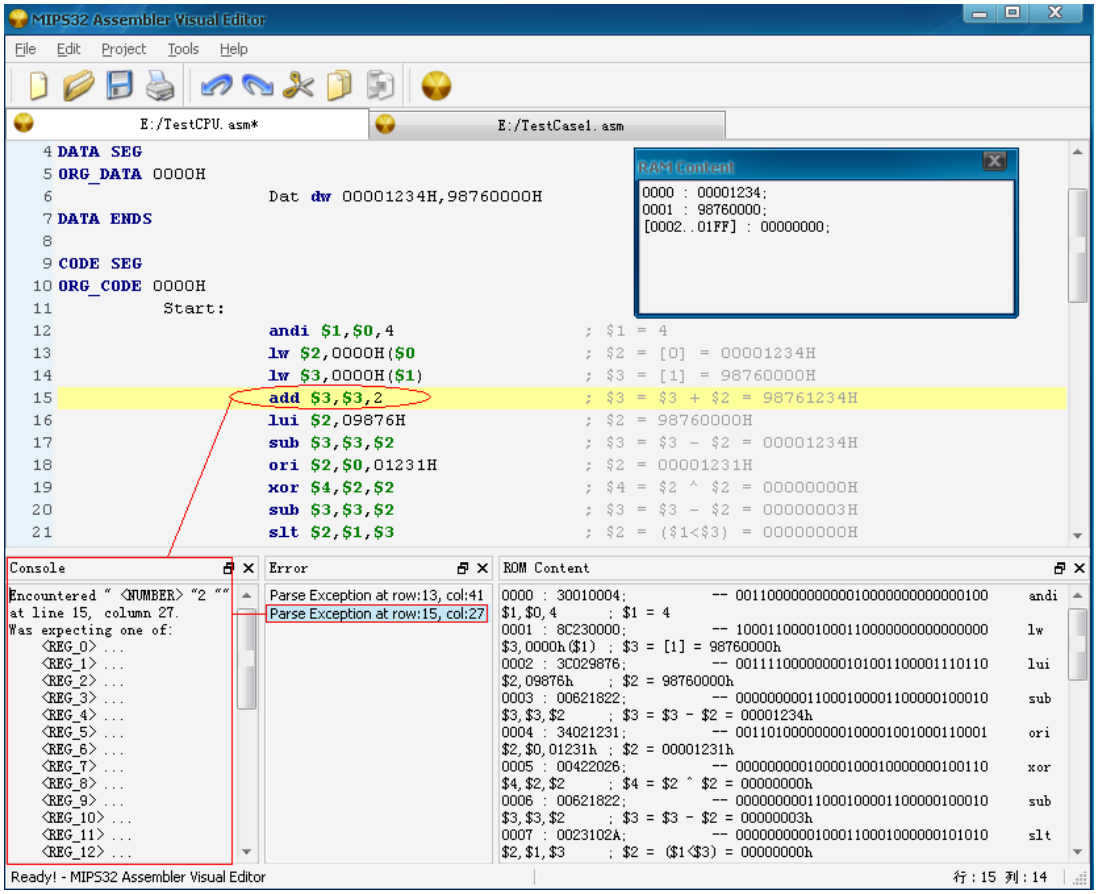
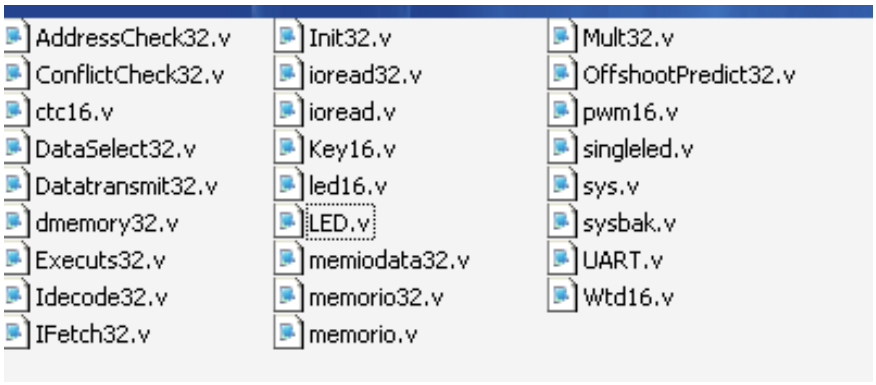

MIPS32 Assembler Visual Editor
File Edit Project Tools Help
E:/TestCPU.asm*
E:/TestCase1.asm
4 DATA SEG
5 ORG_DATA 0000H
6 Dat dw 00001234H,98760000H
7 DATA ENDS
8
9 CODE SEG
10 ORG_CODE 0000H
11 Start:
12 andi $1,$0,4 ; $1 = 4
13 lw $2,0000H($0 ; $2 = [0] = 00001234H
14 lw $3,0000H($1) ; $3 = [1] = 98760000H
15 add $3,$3,2 ; $3 = $3 + $2 = 98761234H
16 lui $2,09876H ; $2 = 98760000H
17 sub $3,$3,$2 ; $3 = $3 - $2 = 00001234H
18 ori $2,$0,01231H ; $2 = 00001231H
19 xor $4,$2,$2 ; $4 = $2 ^ $2 = 00000000H
20 sub $3,$3,$2 ; $3 = $3 - $2 = 00000003H
21 slt $2,$1,$3 ; $2 = ($1<$3) = 00000000H
RAM Content
0000 : 00001234;
0001 : 98760000;
[0002..01FF] : 00000000;
Console
Encountered " <NUMBER> "2 "" at line 15, column 27.
Was expecting one of:
<REG_0> ...
<REG_1> ...
<REG_2> ...
<REG_3> ...
<REG_4> ...
<REG_5> ...
<REG_6> ...
<REG_7> ...
<REG_8> ...
<REG_9> ...
<REG_10> ...
<REG_11> ...
<REG_12> ...
Error
Parse Exception at row:13, col:41
Parse Exception at row:15, col:27
ROM Content
0000 : 30010004; -- 00110000000000010000000000000100 andi $1,$0,4 ; $1 = 4
0001 : 8C230000; -- 10001100001000110000000000000000 lw $3,0000h($1) ; $3 = [1] = 98760000h
0002 : 3C029876; -- 00111100000000101001100001110110 lui $2,09876h ; $2 = 98760000h
0003 : 00621822; -- 00000000011000100001100000100010 sub $3,$3,$2 ; $3 = $3 - $2 = 00001234h
0004 : 34021231; -- 00110100000000100001001000110001 ori $2,$0,01231h ; $2 = 00001231h
0005 : 00422026; -- 00000000010000100010000000100110 xor $4,$2,$2 ; $4 = $2 ^ $2 = 00000000h
0006 : 00621822; -- 00000000011000100001100000100010 sub $3,$3,$2 ; $3 = $3 - $2 = 00000003h
0007 : 0023102A; -- 00000000001000110001000000101010 slt $2,$1,$3 ; $2 = ($1<$3) = 00000000h
Ready! - MIPS32 Assembler Visual Editor
行：15 列：14

图 7

本组设计中的 Verilog HDL **关键程序**清单及 GDT 图（如果有）

一、程序清单



二、关键程序:

1. PC 值的选择更新

```
always@(posedge CLK)//Get the next PC
begin
  intR1=0;
  intR0=0;
  if(InsEffctive_IN==1'b1)
  begin
    IntPC=AC_NextPC_IN;
  end
  if(RES==1'b1)
  begin
    PC=18'b000000000000000000;
  end
  else if(Int0==1&&Imask[0]==0)      //int0
  begin

    PC={16'h3fe,2'b00};
    Imask[0]=1;
    intR0=1;
  end
  else if(Imask[0]==0&&Imask[1]==0&&Int1==1) //int1
  begin

    PC={16'h3ff,2'b00};
    Imask[1]=1;
    intR1=1;
  end
  else if(NextAddrError_IN==1'b1&&MultBusy==1'b0)
       //whether blocked or not must jmp
  begin
    PC={AC_NextPC_IN[15:0],2'b00};
  end
  else if(JmpPredictt_IN==1'b1
          &&Block_IN==1'b0
          &&MultBusy==1'b0) //not blocked
  begin
    PC={PC_Predict_IN[15:0],2'b00};
  end
  else if(Block_IN==1'b0&&MultBusy==1'b0) //normally
  begin
    PC={next_PC,2'b00};
  end
    next_PC[15:0]=PC[17:2]+16'h0001;

 if(EXE_Jrn_IN==1) //interrupt resume
```

```
   begin
    if(EXE_ReadRegister1Address_IN==5'b11010)
    begin
             Imask[0]=0;
    end
    else if(EXE_ReadRegister1Address_IN==5'b11011)
    begin
             Imask[1]=0;
    end
   end
end
```

2. 分支预测，记录的加入和修改

```
always @(posedge CLK)
// update the  OffshootPredictCache, when the Exe
// modulecompleted an instraction of jmp like
begin
    PcIn=0;
    oPcIn=0;
    k=0;
    for(i=0;i<8;i=i+1) //to see whether the record is already
                       // in the  OffshootPredictCache
    begin
      if(current_address[i]==cPC[15:0])
      begin
       k=i;
        PcIn=1;
        if(object_address[i]==AC_JmpAddress_IN[15:0])
```

```
       begin
       oPcIn=1;
       end
      end
    end
    if(AC_DoJmp_IN==1'b1&&PcIn==0) //add a new record
     begin
     current_address[p]=cPC[15:0];
     object_address[p]=AC_JmpAddress_IN[15:0];
     p[2:0]=p[2:0]+3'b001;
     end
    else if(AC_DoJmp_IN==1'b1&&PcIn==1&&oPcIn==0)  //modify the record
    begin
      current_address[k]=cPC[15:0];
      object_address[k]=AC_JmpAddress_IN[15:0];
    end
end
```

3. 指令经过执行单元后进行地址检查

```
module AddressCheck32
(
CLK,
////from  EXE
EXE_Jmp_IN,
EXE_Jal_IN,
EXE_Branch_IN,
EXE_NBranch_IN,
EXE_Zero_IN,
EXE_Jrn_IN,
////
ID_NextPC_IN,
EXE_ALUResult_IN,
EXE_AddResult_IN,
EXE_ReadData1_IN,
InsEffective_IN,

NextPC_OUT,
NextAddrError_OUT,
Jmp_OUT
);
input CLK;
input EXE_Jmp_IN;
input EXE_Jal_IN;
input EXE_Branch_IN;
input EXE_NBranch_IN;
```

```
input EXE_Zero_IN;
input EXE_Jrn_IN;
input[15:0] ID_NextPC_IN;
input[15:0]  EXE_AddResult_IN;
input[15:0]  EXE_ReadData1_IN;
input[15:0]  EXE_ALUResult_IN;
input InsEffective_IN;

output[15:0]  NextPC_OUT;  //get next PC
output NextAddrError_OUT;  //set to '1' when PC was fecthed wrongly,else set
to '0'
output Jmp_OUT; //current Instruction is Jmp like Instruction ..send this
signle to Offshootpredict32
             // module  and updating OffshootPredictCache
wire [15:0] temp1;
wire[15:0] ID_PC;

assign temp1=(EXE_Jrn_IN==1)? EXE_ReadData1_IN:EXE_AddResult_IN;
assign NextPC_OUT=(EXE_Jmp_IN==1'b1) || (EXE_Jal_IN==1'b1)
                ? EXE_ALUResult_IN : temp1;
// ALUResult ---------- j. jal
// AddResult ---------- beq. bne
// ReadData ----------- Jr
assign
Jmp_OUT=((EXE_Jmp_IN==1)||(EXE_Jal_IN==1)||((EXE_Branch_IN==1)&&(EXE_Zero
_IN==1))||((EXE_NBranch_IN==1)&&(EXE_Zero_IN==0))||(EXE_Jrn_IN==1))?1'b1:
1'b0;

assign ID_PC[15:0]=ID_NextPC_IN[15:0]-16'h0001;
assign NextAddrError_OUT=(ID_PC==NextPC_OUT||InsEffective_IN==1'b0)
                        ? 1'b0 : 1'b1;
endmodule
```

4. 写寄存器之前的写入数据的选择

```
if(MEM_Mfhi_IN==1)               //selcete  data  to write
     write_data[31:0]=HI;
  else if(MEM_Mflo_IN==1)
     write_data[31:0]=LO;
  else if((MEM_MemtoReg_IN==0)&&(MEM_Jal_IN==0))
     write_data=MEM_ALUResult_IN[31:0];
  else if((MEM_MemtoReg_IN==0)&&(MEM_Jal_IN==1))
     write_data[31:0]={16'b0000000000000000,MEM_PCPlus4_IN[15:0]};
  else
     write_data=MEM_ReadData_IN;
```

5. 乘法相关指令寄存器操作

```
always@(posedge CLK)
begin
    if(MEM_Mthi_IN==1)    //Mthi instruction
    begin
      HI<=register[MEM_ReadRegister1Address_IN];
    end
    else if(MEM_Mtlo_IN==1)  //Mtlo instruction
    begin
      LO<=register[MEM_ReadRegister1Address_IN];
    end
    else if(EXE_Mult_IN==1)
    //write the multiplication results to HI and LO
    begin
      HI<=EXE_HI_IN;
      LO<=EXE_LO_IN;
    end
end
```

本组设计主要测试结果与性能分析（.vwf、.rpt 中的资源使用情况）

1. CPU 测试实例：

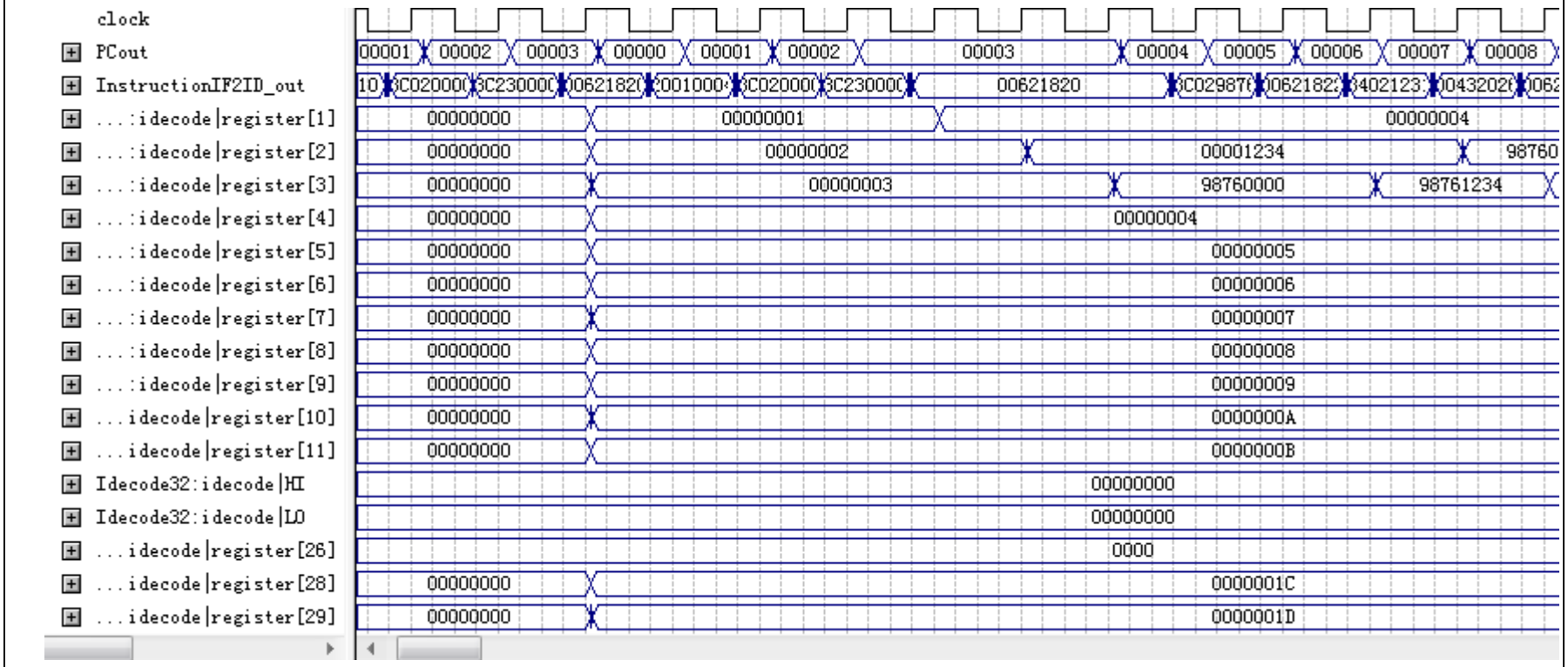
数据 RAM

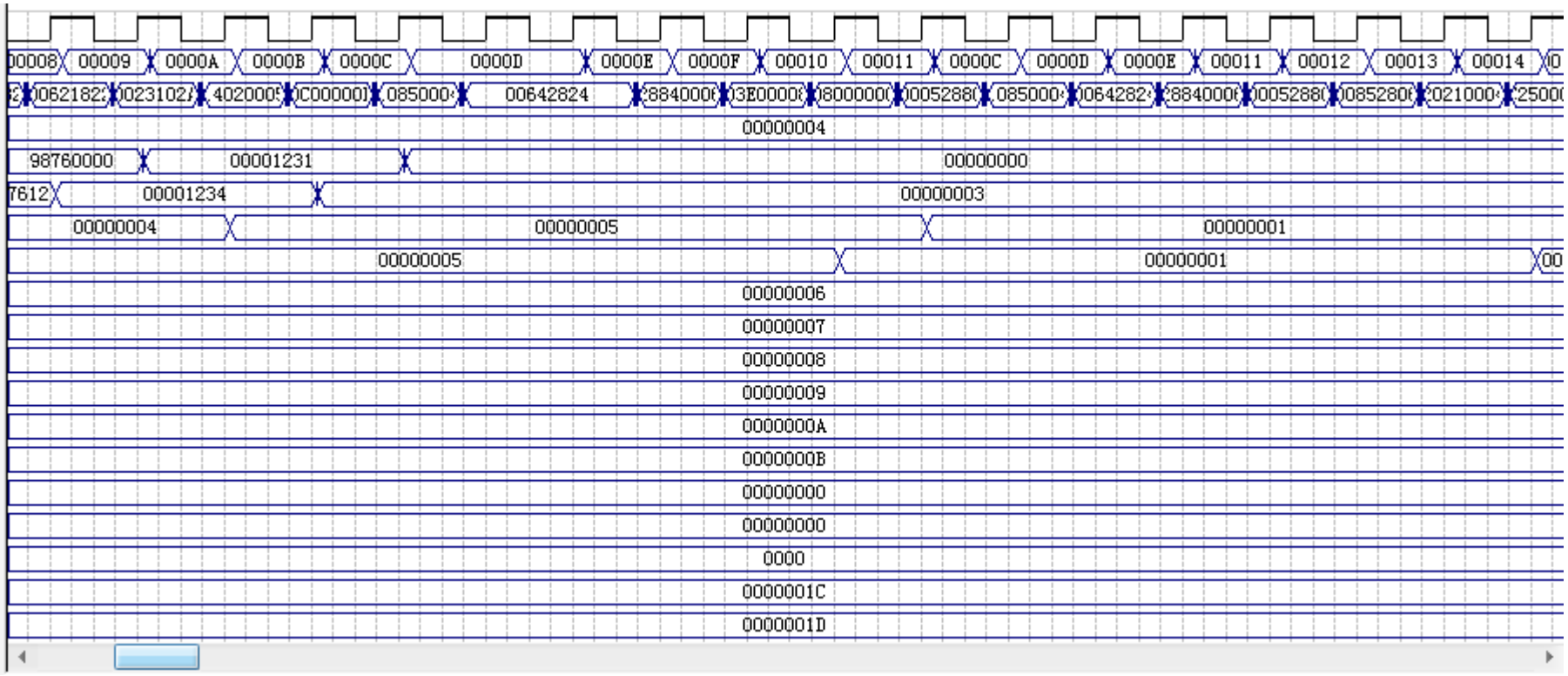
```
DEPTH = 1024;
WIDTH = 32;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
BEGIN
0000 : 00001234;
0001 : 98760000;
[0002..01FF] : 00000000;

END;
```

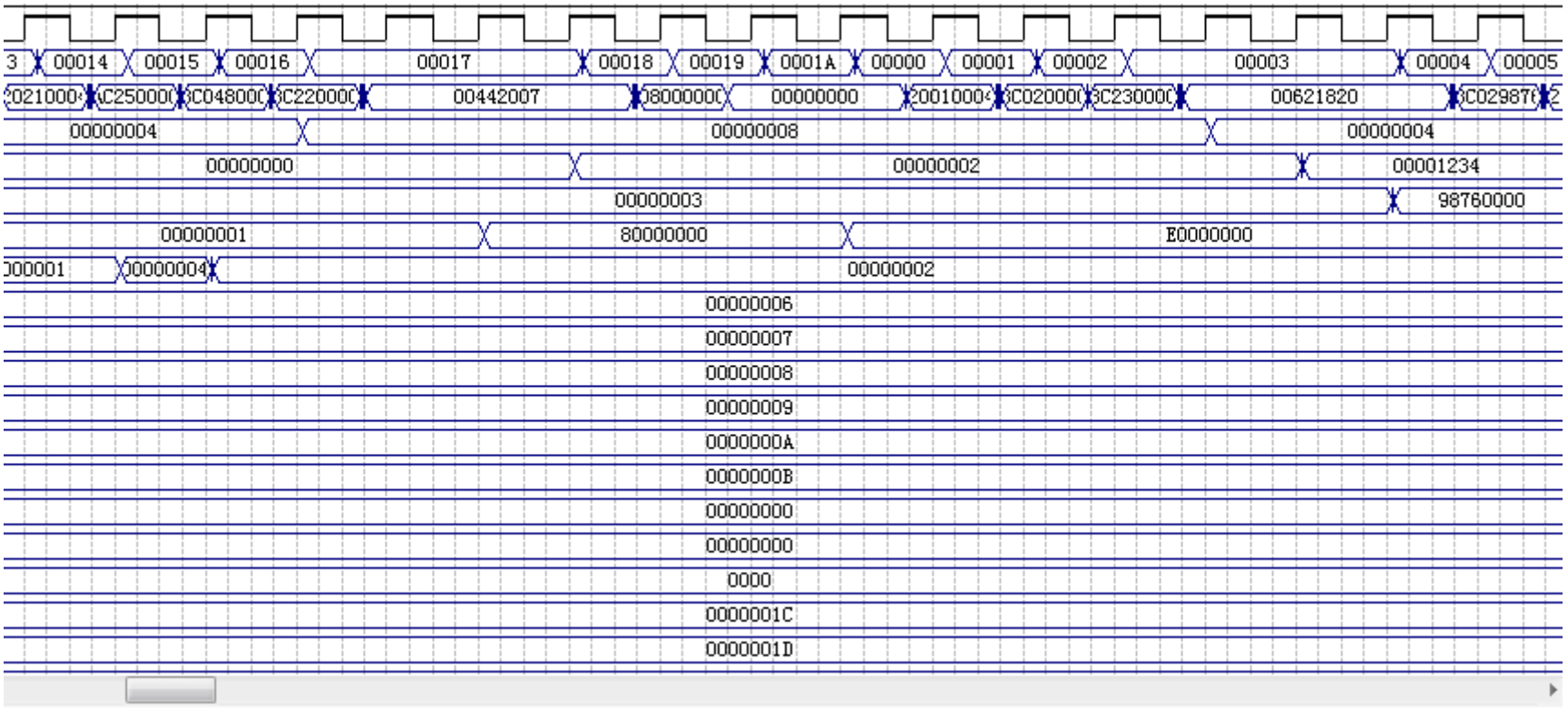
指令 ROM

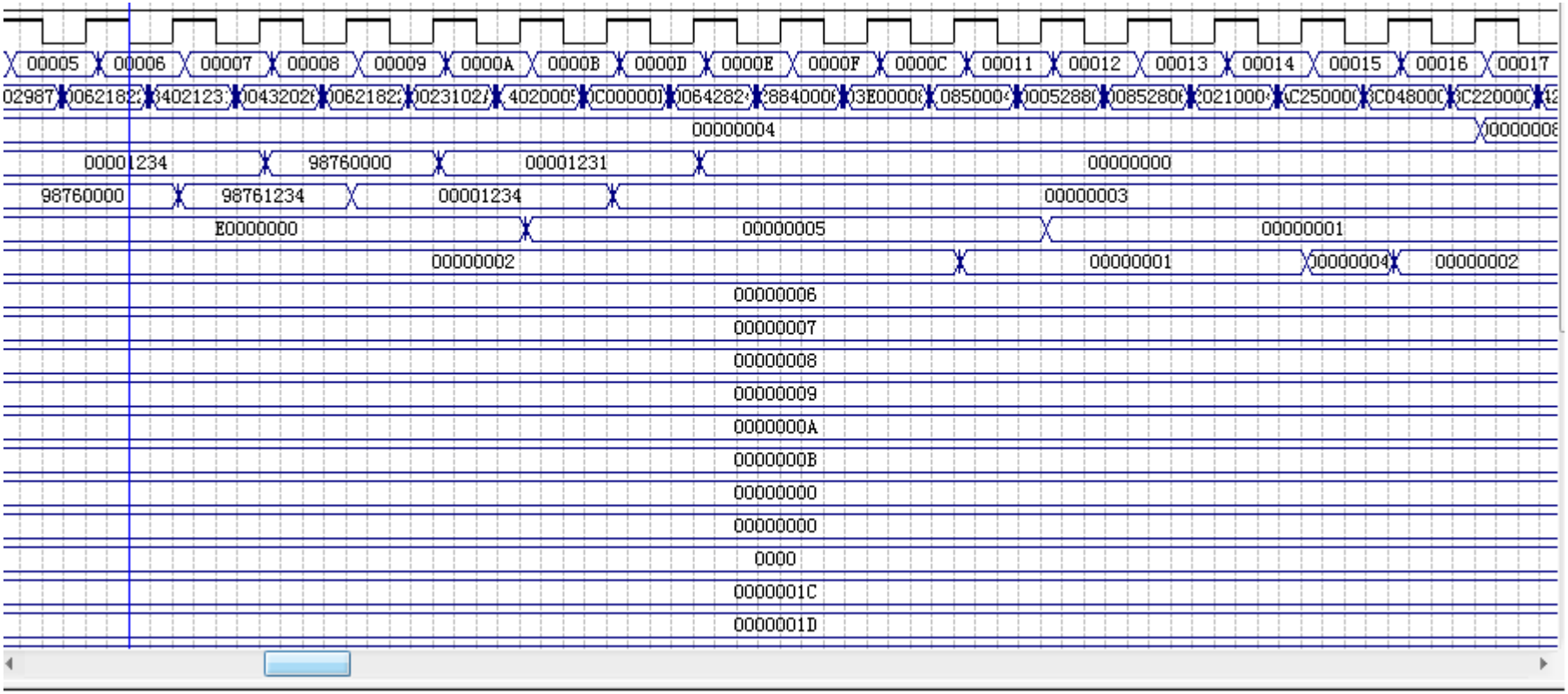
```
DEPTH = 1024;
WIDTH = 32;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
BEGIN
0000 : 20010004;        -- 00100000000000010000000000000100     addi $1,$0,4    ; $1 = 4
0001 : 8C020000;        -- 10001100000000100000000000000000     lw $2,0000h($0) ; $2 = [0] = 00001234h
0002 : 8C230000;        -- 10001100001000110000000000000000     lw $3,0000h($1) ; $3 = [1] = 98760000h
0003 : 00621820;        -- 00000000011000100001100000100000     add $3,$3,$2    ; $3 = $3 + $2 = 98761234h
0004 : 3C029876;        -- 00111100000000101001100001110110     lui $2,09876h   ; $2 = 98760000h
0005 : 00621822;        -- 00000000011000100001100000100010     sub $3,$3,$2    ; $3 = $3 - $2 = 00001234h
0006 : 34021231;        -- 00110100000000100001001000110001     ori $2,$0,01231h        ; $2 = 00001231h
0007 : 00432026;        -- 00000000010000110010000000100110     xor $4,$2,$3    ; $4 = $2 ^ $3 = 00000005h
0008 : 00621822;        -- 00000000011000100001100000100010     sub $3,$3,$2    ; $3 = $3 - $2 = 00000003h
0009 : 0023102A;        -- 00000000001000110001000000101010     slt $2,$1,$3    ; $2 = ($1<$3) = 00000000h
000A : 14020005;        -- 00010100000000100000000000000101     bne $2,$0,error ; if $2!=$0, goto error
000B : 0C00000D;        -- 00001100000000000000000000001101     jal subproc     ; goto subproc
000C : 10850004;        -- 00010000100001010000000000000100     beq $5,$4,next  ; if $5==$4, goto next
000D : 00642824;        -- 00000000011001000010100000100100     and $5,$3,$4    ; $5 = $3 & $4 = 00000001h
000E : 28840006;        -- 00101000100001000000000000000110     slti $4,$4,6    ; $4 = ($4 <6) = 00000001h
000F : 03E00008;        -- 00000011111100000000000000001000     jr $31  ; subproc return
0010 : 08000000;        -- 00001000000000000000000000000000     j start ; reset
0011 : 00052880;        -- 00000000000001010001010000010000     sll $5,$5,2     ; $5 = $5<<2  = 00000004h
0012 : 00852806;        -- 00000000100001010001010000000110     srlv $5,$4,$5   ; $5 = $5>>$4 = 00000002h
0013 : 20210004;        -- 00100000001000010000000000000100     addi $1,$1,4    ; $1 = $1 +4  = 8
0014 : AC250000;        -- 10101100001001010000000000000000     sw $5,0000h($1) ; [2] = $5 = 00000002h
0015 : 3C048000;        -- 00111100000001001000000000000000     lui $4,08000h   ; $4 = 80000000h
0016 : 8C220000;        -- 10001100001000010000000000000000     lw $2,0000h($1) ; $2 = [2] = 00000002h
0017 : 00442007;        -- 00000000010001000010000000000111     srav $4,$2,$4   ; $4 = $4>>2  = e0000000
0018 : 08000000;        -- 00001000000000000000000000000000     j start ; loop
[0019..03FD] : 00000000;
03FE : 03400008;        -- 00000011010000000000000000001000     jr $26  null
03FF : 03600008;        -- 00000011011000000000000000001000     jr $27  null

END;
```

仿真结果


clock
PCout
InstructionIF2ID_out
...:idecode|register[1]
...:idecode|register[2]
...:idecode|register[3]
...:idecode|register[4]
...:idecode|register[5]
...:idecode|register[6]
...:idecode|register[7]
...:idecode|register[8]
...:idecode|register[9]
...idecode|register[10]
...idecode|register[11]
Idecode32:idecode|HI
Idecode32:idecode|LO
...idecode|register[26]
...idecode|register[28]
...idecode|register[29]
00001
00002
00003
00000
00001
00002
00003
00004
00005
00006
00007
00008
00621820
00000000
00000001
00000004
00000002
00001234
98760
00000003
98760000
98761234
00000005
00000006
00000007
00000008
00000009
0000000A
0000000B
0000
0000001C
0000001D


00008
00009
0000A
0000B
0000C
0000D
0000E
0000F
00010
00011
0000C
0000D
0000E
00011
00012
00013
00014
00642824
00000004
98760000
00001231
00000000
7612
00001234
00000003
00000004
00000005
00000001
00000005
00000001
00000006
00000007
00000008
00000009
0000000A
0000000B
00000000
00000000
0000
0000001C
0000001D


3
00014
00015
00016
00017
00018
00019
0001A
00000
00001
00002
00003
00004
00005
021000
C25000
C04800
C22000
00442007
0800000
00000000
2001000
C02000
C23000
00621820
C02987
00000004
00000008
00000004
00000000
00000002
00001234
00000003
98760000
00000001
80000000
E0000000
000001
00000004
00000002
00000006
00000007
00000008
00000009
0000000A
0000000B
00000000
00000000
0000
0000001C
0000001D


00005
00006
00007
00008
00009
0000A
0000B
0000D
0000E
0000F
0000C
00011
00012
00013
00014
00015
00016
00017
00000004
00001234
98760000
00001231
00000000
98760000
98761234
00001234
00000003
E0000000
00000005
00000001
00000002
00000001
00000004
00000002
00000006
00000007
00000008
00000009
0000000A
0000000B
00000000
00000000
0000
0000001C
0000001D

2. CPU 与外设联合测试

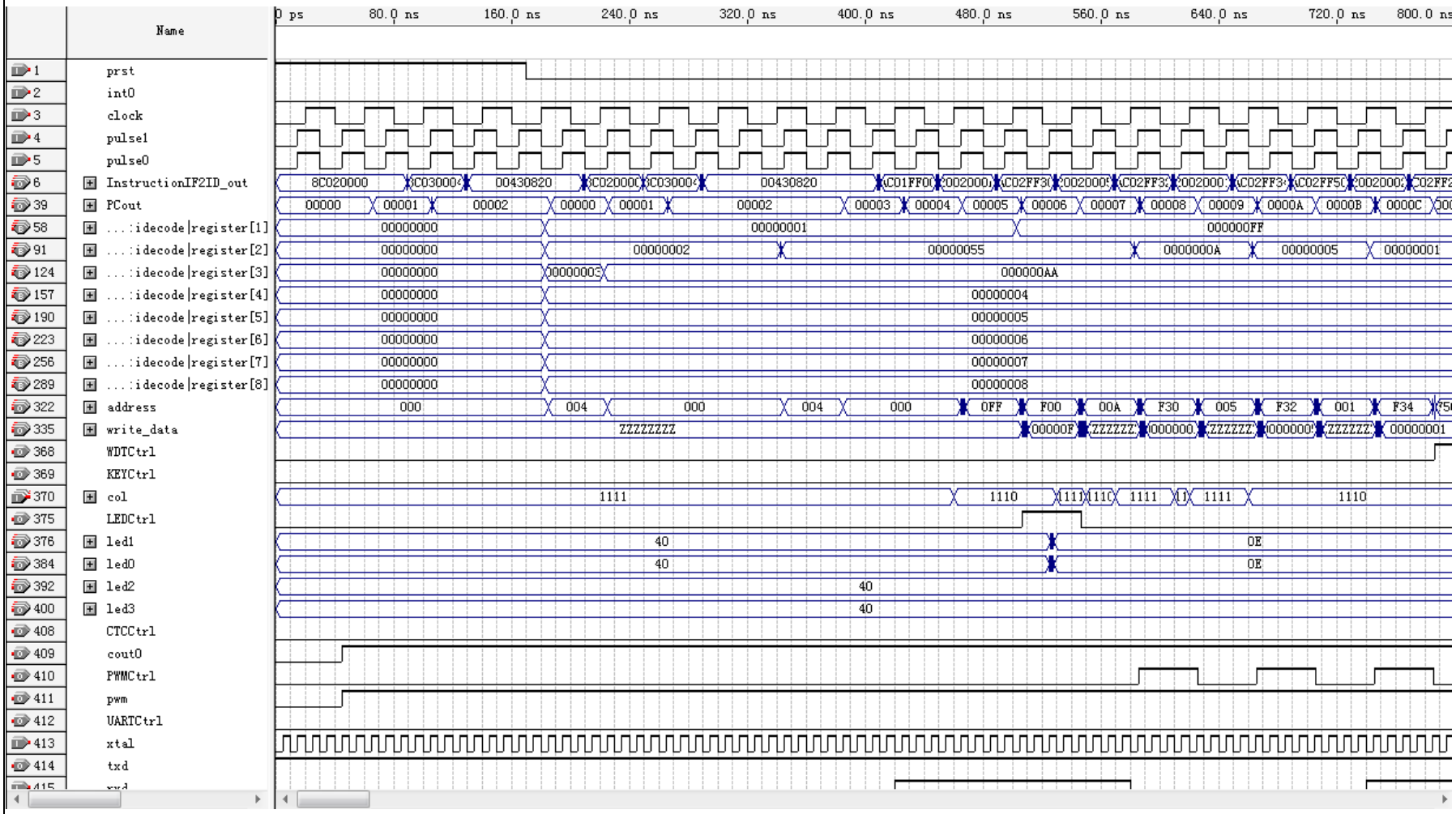
数据 RAM

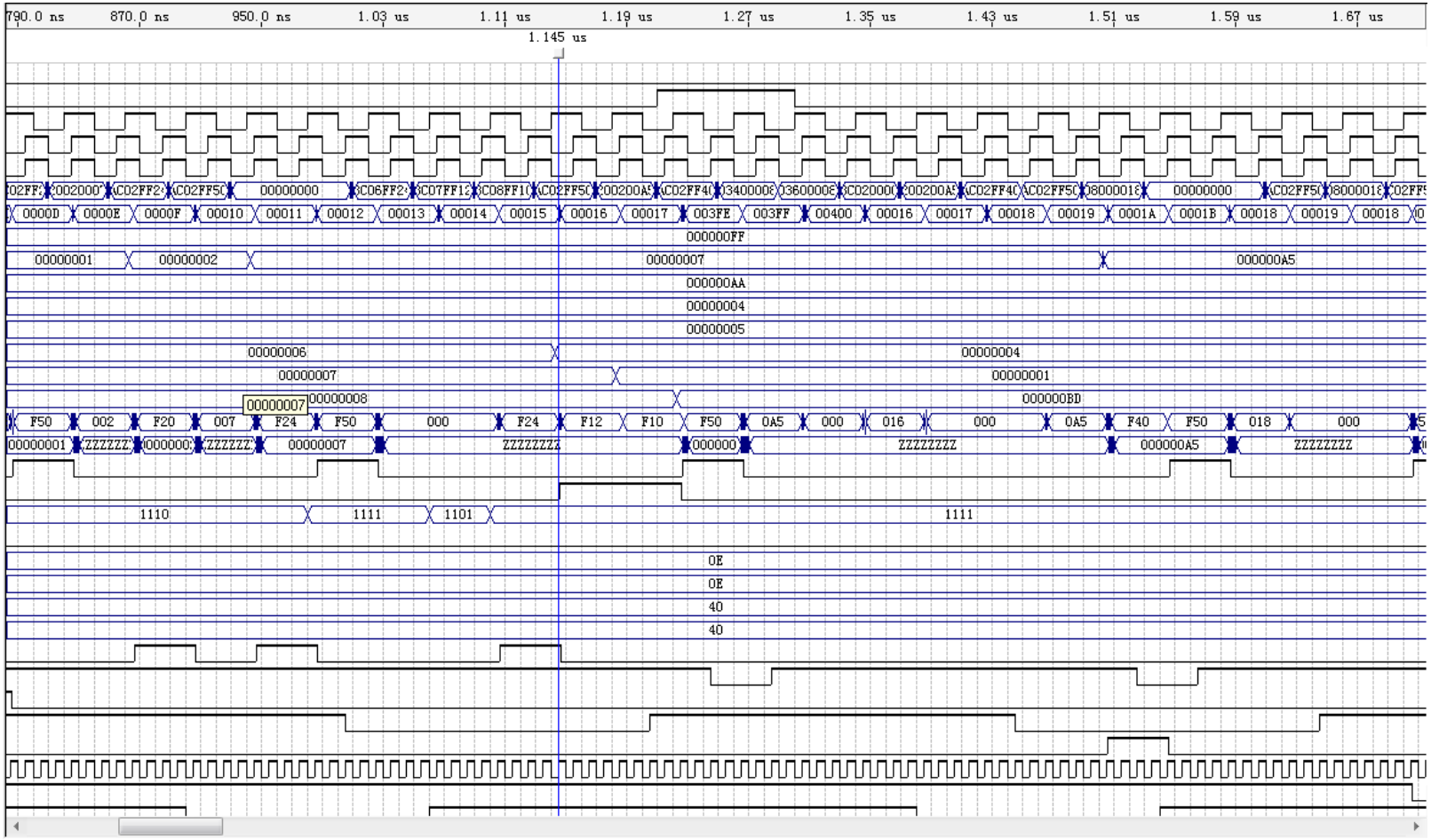
```
DEPTH = 512;
WIDTH = 32;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
BEGIN
        0 : 00000055;
        1 : 000000AA;
        [2..1FF] : 00000000;
END;
```

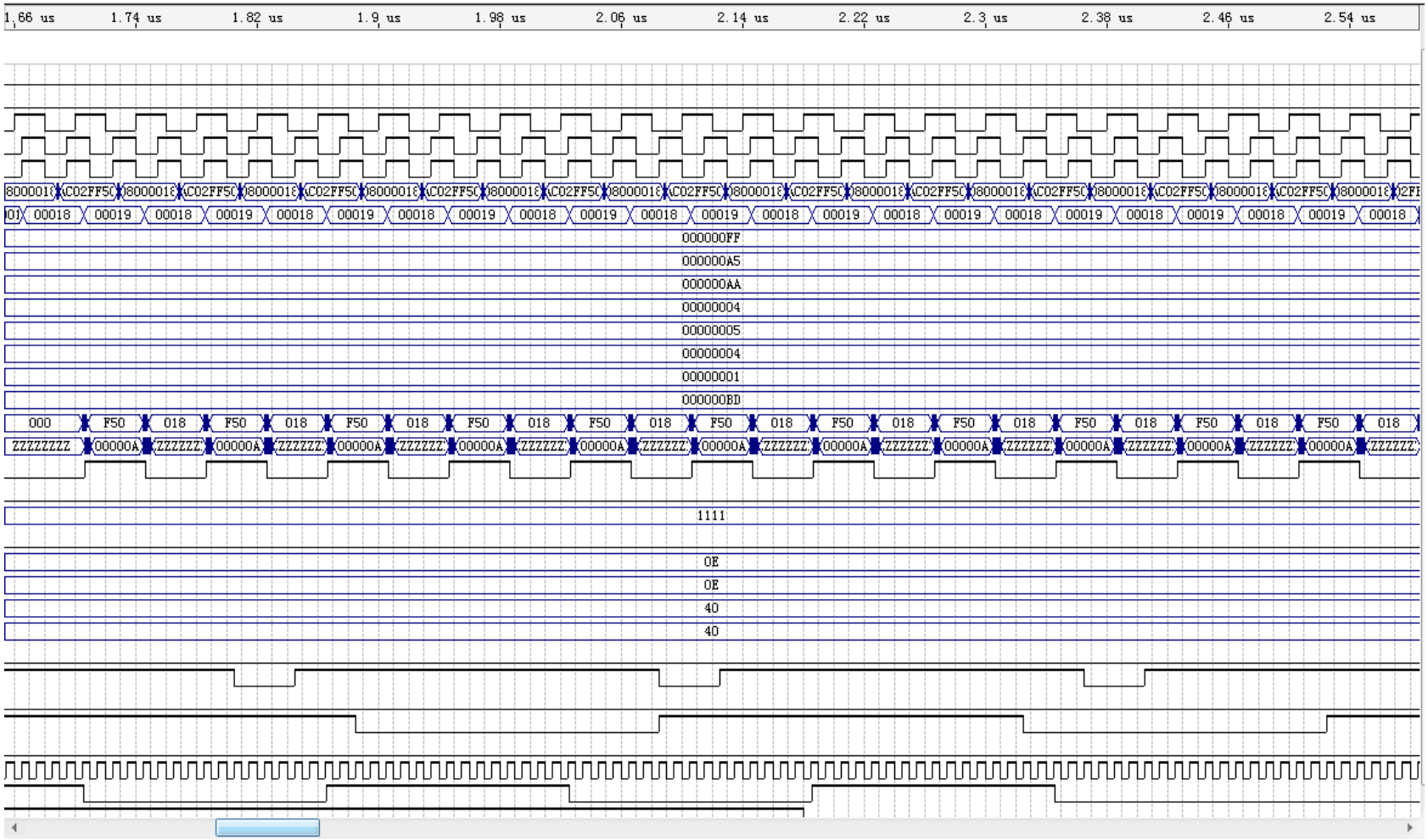
指令 ROM

```
DEPTH = 1024;
WIDTH = 32;

ADDRESS_RADIX = HEX;
DATA_RADIX = HEX;

CONTENT
BEGIN

        0 : 8c020000; -- lw $2 0($0)                100011(LW)   00000(RS) 00010(RT) 0000000000000000
        1 : 8c030004; -- lw $3 2($0)                100011(LW)   00000(RS) 00011(RT) 0000000000000001
        2 : 00430820; -- add $1,$2,$3               000000       00010(RS) 00011(RT) 00001(RD) 00000
        3 : AC01FF00; -- sw $1 FF00($0)             101011(SW)   00000(RS) 00001(RT) 1111111100000000
        4 : 2002000A; -- addi $2,$0,10              001000(ADDI) 00000(RS) 00010(RT) 0000000000001000
        5 : AC02FF30; -- sw $2 FF30($0)             101011(SW)   00000(RS) 00010(RT) 1111111100110000
        6 : 20020005; -- addi $2,$0,5               001000(ADDI) 00000(RS) 00010(RT) 0000000000000101
        7 : AC02FF32; -- sw $2 FF32($0)             101011(SW)   00000(RS) 00010(RT) 1111111100110010
        8 : 20020001; -- addi $2,$0,1               001000(ADDI) 00000(RS) 00010(RT) 0000000000000001
        9 : AC02FF34; -- sw $2 FF34($0)             101011(SW)   00000(RS) 00010(RT) 1111111100110100
        A : AC02FF50; -- sw $2 FF50($0)             101011(SW)   00000(RS) 00010(RT) 1111111101010000
        B : 20020002; -- addi $2,$0,2               001000(ADDI) 00000(RS) 00010(RT) 0000000000000010
        C : AC02FF20; -- sw $2 FF20($0)             101011(SW)   00000(RS) 00010(RT) 1111111100100000
        D : 20020007; -- addi $2,$0,7               001000(ADDI) 00000(RS) 00010(RT) 0000000000000111
        E : AC02FF24; -- sw $2 FF24($0)             101011(SW)   00000(RS) 00010(RT) 1111111100100100
        F : AC02FF50; -- sw $2 FF50($0)             101011(SW)   00000(RS) 00010(RT) 1111111101010000
        10: 00000000; -- nop
        11: 00000000; -- nop
        12: 8C06FF24; -- lw $6,FF24($0)             100011(LW)   00000(RS) 00110(RT) 1111111100100100
        13: 8C07FF12; -- lw $7,FF12($0)             100011(LW)   00000(RS) 00111(RT) 1111111100010010
        14: 8C08FF10; -- lw $8,FF10($0)             100011(LW)   00000(RS) 00111(RT) 1111111100010000
        15: AC02FF50; -- sw $2 FF50($0)             101011(SW)   00000(RS) 00010(RT) 1111111101010000
        16: 200200A5; -- addi $2,$0,A5              001000(ADDI) 00000(RS) 00010(RT) 0000000010100101
        17: AC02FF40; -- sw $2 FF40($0)             101011(SW)   00000(RS) 00010(RT) 1111111101000000
        18: AC02FF50; -- SW $2,FF50($0)
        19: 08000018; -- j 0060H                     000010(J)    00000000000000000000000000(0000H)
        [1A..3FD] : 00000000;
        3FE: 03400008; -- jr $1A                     000000       01100(RS) 00000 00000 00000 001000
        3FF: 03600008; -- jr $1B                     000000       01110(RS) 00000 00000 00000 001000
END;
```

仿真结果:


Name
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
800.0 ns
1 prst
2 int0
3 clock
4 pulse1
5 pulse0
6 InstructionIF2ID_out
39 PCout
58 ...:idecode|register[1]
91 ...:idecode|register[2]
124 ...:idecode|register[3]
157 ...:idecode|register[4]
190 ...:idecode|register[5]
223 ...:idecode|register[6]
256 ...:idecode|register[7]
289 ...:idecode|register[8]
322 address
335 write_data
368 WDTCtrl
369 KEYCtrl
370 col
375 LEDCtrl
376 led1
384 led0
392 led2
400 led3
408 CTCCtrl
409 cout0
410 PWMCtrl
411 pwm
412 UARTCtrl
413 xtal
414 txd


790.0 ns
870.0 ns
950.0 ns
1.03 us
1.11 us
1.19 us
1.27 us
1.35 us
1.43 us
1.51 us
1.59 us
1.67 us
1.145 us
00000007
1110
1111
1101
1111
0E
0E
40
40


1.66 us
1.74 us
1.82 us
1.9 us
1.98 us
2.06 us
2.14 us
2.22 us
2.3 us
2.38 us
2.46 us
2.54 us
8000018
C02FF50
00018
00019
000000FF
000000A5
000000AA
00000004
00000005
00000004
00000001
000000BD
000
F50
018
ZZZZZZZZ
00000A
ZZZZZZ
1111
0E
0E
40
40

性能分析:

1. 主频:

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock
1	Total number of failed paths						
2	Worst-case tsu	N/A	None	11.193 ns	col[0]	Key16:keyIloread_data[0]	--
3	Worst-case tco	N/A	None	14.080 ns	memorio32:memorioIMemWriteIn_r	write_data[26]	clock
4	Worst-case th	N/A	None	-0.729 ns	rxd	UART:uartIrcnt3.00001	--
5	Clock Setup: 'clock'	N/A	None	44.35 MHz (period = 22.550 ns)	Executs32:executeIPCplus4_OUT_r[0]	OlfshootPredict32:olfshootPredictIcurrent_address[2][4]	clock
6	Clock Setup: 'xtal'	N/A	None	102.92 MHz (period = 9.716 ns)	UART:uartIstatus[0]	UART:uartIstat3[0]	xtal
7	Clock Hold: 'clock'	Not operational: Clock Skew > Data Delay	None	N/A	UART:uartIstat1[5]	UART:uartIstatus[5]	clock
8	Clock Hold: 'xtal'	Not operational: Clock Skew > Data Delay	None	N/A	UART:uartIstat3[3]	UART:uartIstatus[3]	xtal

2. 逻辑单元数目

Fitter Summary

Fitter Status	Successful - Sat Dec 27 15:39:30 2008
Quartus II Version	7.2 Build 151 09/26/2007 SJ Full Version
Revision Name	sys
Top-level Entity Name	sys
Family	Cyclone
Device	EP1C6Q240C6
Timing Models	Final
Total logic elements	5,809 / 5,980 (97 %)
Total pins	148 / 185 (80 %)
Total virtual pins	0
Total memory bits	65,536 / 92,160 (71 %)
Total PLLs	0 / 2 (0 %)

3. 功耗:

owerPlay Power Analyzer Summary

PowerPlay Power Analyzer Status	Successful - Sun Dec 28 13:32:30
Quartus II Version	7.2 Build 151 09/26/2007 SJ Full
Revision Name	sys
Top-level Entity Name	sys
Family	Cyclone
Device	EP1C6Q240C6
Power Models	Final
Total Thermal Power Dissipation	60.11 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	60.00 mW
I/O Thermal Power Dissipation	0.11 mW
Power Estimation Confidence	Low: user provided insufficient

课程设计总结（包括设计的总结和还需改进的内容）

这次综合课程设计集合了硬件和软件的设计，硬件部分的设计使得我们对片上系统的前端设计以及时序仿真和分析有了基本的了解。在硬件方面，引入了经典的流水线思想，很大程度上提高了主频。同时加入分支预测和数据转发功能，进一步提高了流水效率。 当然还有很多需要改进的地方，这里的分支预测对条件跳转的预测效果有时不是很好，因此可以增加动态判断，提高预测的正确率。在主频方面，由于没有做到门级优化，很难进一步提高主频。由于芯片面积的限制，很难再增加除法器，因此该系统有待于进一步优化以减小芯片面积。

通过这次课程设计，我们不仅重新复习了过去上过的很多硬件和软件课程的内容，锻炼了动手能力，更为重要的是我们学会了全面综合地去分析问题，从硬件和软件两个方面协同设计，完成任务。还有一点让我们感到高兴的是，我们通过这次设计，感觉 CPU 设计以及一个实际编译器的设计并不是那么高不可攀，这为我们今后走上工作岗位，在遇到新的挑战的时候，提高了自信。

验 收 报 告

主频	44.35MHz	逻辑单元数	(%)	功 耗	mW
CPU 类型		☐单周期 ☐多周期 ☐流水线 ☐超标量			
CPU 设计	☐通过 ☐未通过		数码管控制	☐通过 ☐未通过	
定时/计数器	☐通过 ☐未通过		键盘控制器	☐通过 ☐未通过	
PWM 控制	☐通过 ☐未通过		UART 控制	☐通过 ☐未通过	
看门狗控制	☐通过 ☐未通过		合 成	☐通过 ☐未通过	
汇编器设计	☐通过 ☐未通过		BIOS	☐有 ☐无 ☐未通过	
中断	☐有 ☐无 ☐未通过		验收答辩	☐通过 ☐未通过	
加分项目					
验收结论	☐优秀 ☐良好 ☐中等 ☐及格 ☐不及格				

教师综合评价:

教师签名: _____