東南大學

计算机系统综合设计

设计报告

组长：苏　凯 09004113

成员：包　诚 09004111

李　挺 09004114

郑添隆 09004115

徐夙龙 09004118

朱晓林 09004122

东南大学计算机科学与工程学院

二 00 七年十一月

设计名称	miniMIPS16				
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
包　诚	09004111	BIOS 和 PROGRAM 应用程序的编写			
苏　凯	09004113	组长,CPU、中断及ＩＯ信号模块的设计，部分外围电路优化，CPU 与外围电路整合，全组工作分配与协调			
李　挺	09004114	汇编语言编译器和一些测试程序的编写			
郑添隆	09004115	外围电路键盘和看门狗			
徐夙龙	09004118	外围电路计数器，pwm，led，uart 等			
朱晓林	09004122	顶层文件 dbf 图，io 译码模块，中断			

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一个重要组成部分，因此要在报告中明确标明每个模块的设计者。

教师评语:

教师签名：_____

本组设计的功能描述（含所有实现的模块的功能）

整个设计分成个主要部分：CPU、外围电路、配套软件。

CPU:

一个可运行指定的 31 条 miniMIPS 指令的 RISC 型 MIPS16 微处理器，具有 32 位指令，16 位地址线和数据线。采用哈佛结构，有独立的 2KB 的程序处理器(ROM)和 2KB 的数据存储器(RAM)。IO 与 RAM 统一编址，片选 CS 及读写信号(IOR/IOW)都是低电平有效。有两个中断源入口，两级中断优先级，不允许同级嵌套，中断 0 优先级高于中断 1。

外围电路:

16 位定时/计数器

两个定时/计数器 CNT0 和 CNT1。

具有计数和定时两个功能。

计数方式下可以对输入的外部脉冲进行计数，当计数到初值寄存器的值的时候，设置状态寄存器的相应位。

定时方式下，在时钟作用下计时器做加 1，到定时初值的时候设置状态寄存器的相应位，并在相应的 COUT 脚输出一个时钟的低电平（平时 COUT 是高电平）。

状态寄存器在被读取后被清零。

PWM

内部一个 16 位计数器和一个 16 位对比值，计数器周而复始的加 1 计数，计数到计数器的最大值（默认为 FFFFH）的时候转为 0 再计数。当计数器的值大于对比值，输出端输出低电平，否则输出高电平。

对比值（默认为 7FFFH）的不同决定了输出脉冲的占空比。

计数器的最大值可以由软件设定，以便确定 PWM 的输出脉冲的调制频率。

4×4 键盘扫描电路

自动扫描 4×4 的键盘，当有键盘按下的时候扫描键值，将键值记录到键寄存器，并置位状态寄存器中的“有键”标志。当 CPU 读出键值后，将“有键”标志清除。

4 位 7 段 LED

通过向该控制电路写 16 位数据，经过译码控制共阳极的 7 段 LED 显示。16 位数每半个字节控制一位 7 段 LED，从高位到低位排列。

简单 UART

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。

看门狗

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。（下有注释）

通过软件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 始计数。

配套软件:

汇编语言编译器

用 C++语言编写的编译器(complier.exe)，在命令行窗口下，可以选择通过键盘输入一段程序，或者将已经编写好的汇编程序编译成相应的数据 RAM 和程序 ROM 的.mif 文件，供 Quartus 工程使用。键盘输入的程序写入 temp 文件夹中，并通过编译。编译后的 mif 文件位于应用程序所在目录。

BIOS 程序

位于 bios 文件夹中

bios1.asm: KEY16 功能调用实现: 扫描键盘输入，将输入的值写入 RAM 地址 37B 中

bios2.asm: LED16S 功能调用实现: 将 4 号寄存器的值单独显示在 LED 最低位上，高位为 F

bios3.asm: LED16A 功能调用实现: 将参数存入 4，5，6，7 号寄存器，由低到高依次显示在 LED 上

bios4.asm: UART16R 功能调用实现: 通过串口读入数据并将数据写入 RAM 地址 37C 中

bios5.asm: UART16W 功能调用实现: 通过串口将 4 号寄存器的值输出

应用程序

根据汇编语言的要求编写的应用程序，实现对硬件的测试或特定的应用功能，程序中可以调用 BIOS，也可以在最后定义 INT0 和 INT1 两个中断的中断服务程序。位于 programme 文件夹中。文件夹 test set 中的文件为实验过程中各部件的测试文件

本组设计的主要特色

CPU 采用多周期设计，大部分指令为 4 周期，此外有 2 周期、3 周期指令，缩短指令处理周期，优化 CPU 处理速度，最快 CLK 达 34MHz。如果遇到中断，增加 2 周期处理中断，并转入中断服务程序，中断结构时回到中断点继续执行。

设计中有多处模块为方便升级成流水线 CPU 及实现全 MIPS 做了考虑，如 ALUOP 编码使用 5 位，CPU 分成四个处理周期，及有限状态机的书写。

键盘模块中增加防抖动模块，并采用一次按键一次输入功能。

全部硬件用 VHDL 语言实现。

软件方面编译器增加了对操作堆栈的两条宏指令 push 和 pop 的支持。

对程序中编译出错的行进行输出显示。

对标号采用一遍扫描的方式，必要时向后扫描寻找是否有匹配的标号。

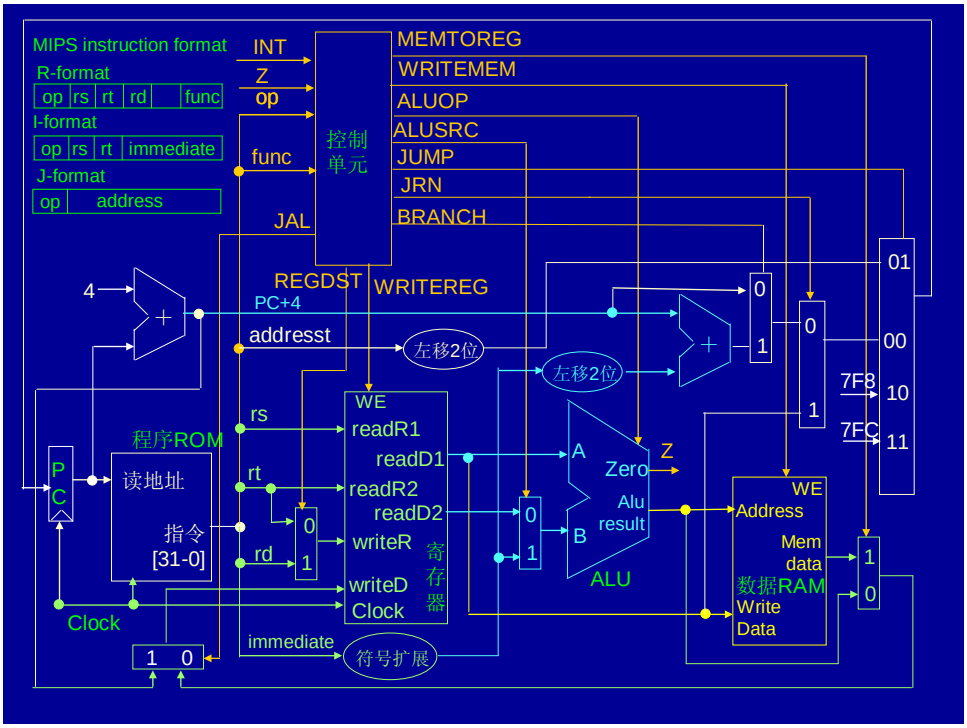
支持应用程序对 BIOS 的调用，编译时通过跳转指令（JAL 和 JR $15）在应用程序和 BIOS 程序之间实现切换，将 BIOS 程序写入固定的 ROM 区域（1B8—1FD）。

支持应用程序对中断服务程序的编写，程序最后以 INT0: 和 INT1: 开始，以 END INT0 和 END INT1 结束。也可以不写，编译器将默认无中断服务程序。通过跳转指令（J 和 JR $i0/$i1）在应用程序和中断服务程序之间切换

编译器自动将中断服务程序的入口地址写入中断向量（ROM 中的 1FE，1FF），使用跳转 J 指令

本组设计的体系结构

CPU 体系结构基本框图



说明：实际的实现中，在增加了 IO 模块，限于图的大小无法画出，详细请看 bdf 图。在实现 Jal 指令时要写 15 号，所在增加了一个数据选择器。

数据通路

31 条指令共分成 8 类数据通路,与课件的 8 类数据类似，在此不一一画出，详细请看课件 1 每 41~48 页。

ALU 功能编码表

指令类型	ALUOP[4..0]	功能
逻辑指令	00000 01000 10000 11000	A 与 B A 或 B A 或非 B A 异或 B
加减运算指令	X0001 X1001	A 加 B A 减 B
比较指令	X1010	A<B
加载指令	XX011	把 B 的低 16 位加载到高 16 位上*
移位指令	00100 01100 10100 11100	B 逻辑右移 A 位 B 算术右移 A 位 B 逻辑左移 A 位 B 循环移位 A 位*
首 0、1 指令	X0101 X1101	对 A 首 1 计数* 对 A 首 0 计数*

乘法指令	00110 01110	A 乘 B，无符号乘* A 乘 B，有符号乘*
除法指令	00111 01111	A 除 B，无符号除* A 除 B，有符号除*

注：*号项为本次设计无需使用功能，待后升级完成。

CPU 结构设计

CPU 采用多周期设计，共分为 SIF、SID、SEXE、SWB 四个周期。PC 为下降沿触发，其余部件为上升沿触发。

SIF: 为取指周期，取出指令，跳入 SID 周期。

SID: 译码周期，完成指令的译码工作，并判断是否是跳转指令，若是，给 PC 赋值，并跳回 SIF 周期执行下条指令；不是，则跳入 SEXE 周期。

SEXE: 运算周期，跟据译码选择不同的运算码给 ALU。如果是 JRN 指令，保存 PC+4 到$15 寄存器，并给 PC 赋指令中的值，跳回 SIF 批令。

SWB: 写回周期。如果是寄存器写回，给出寄存器写回信号，如果是 SW 指令，给出 MEM 写信号，数据将在 CLK 的下一个上升沿写入。

中断处理设计

中断的检测在 SWB 周期，当中断有效时，跳入 SINTA,SINTB 两个周期来处理中断。

SINTA: 将 PC+4 的值存入$i0 或$i1。置位 IMASKR 相应位。

SINTB: 写 PC。PC 等于 7F8 或 7FC。

由于跳转指令(J,Jal,Jrn)后是不能有中断的(因为跳转指令的下一条指令的 PC 不是 PC+4)，所以当有中断请求时，如果是当前指令是跳转指令，要在下一条非跳转指令才能中断。由于 J 是两周期指令，Jal 是三周期指令，不会进入 SWB 周期，只有 Jrn 有可能遇到中断请求，这时应在 SWB 中特殊处理。

中断返回时在 SEXE 周期清零 IMASKR 相应位。

IO 模块设计

IO 端口地址与 DATA RAM 统一编址，IO 地址为 FF00H~FFFFH，将 8 根 IO 端口线（16 位地址线的低 8 位，高 8 位为全 1）的高 4 位用来译码得到最多 16 个接口电路的片选信号。低 4 位组成每个接口电路的 16 个字节端口地址，由于 MIPS16 只有 16 位数据处理能力。所以每个接口电路实际上是有 8 个字端口地址。如果有 CS 信号，则相应信号位在 SEXE 周期置低电平，在 SIF 周期置高；如果有 IOR 或 IOW，将在 SWB 周期置低电平，在 SIF 周期置高。

外围电路设计

主要功能在前面已经描述，下面讲述寄存器以及内部逻辑

address，cs，reset，clk，iow，ior 等信号统一描述

address 用于区分模块内部寄存器，cs 为片选，reset 为复位

iow 下降沿在 cs 有效时用于 CPU 写数据

ior 下降沿在 cs 有效时用于 CPU 读数据

clk 的具体用法视各模块定

16 位定时/计数器　　起始地址 FF20

CNT0 方式（状态）寄存器（FF20H）

CNT1 方式（状态）寄存器（FF22H）

作为方式寄存器时，0 位为 0 表示定时，为 1 表示计数；1 位为 0 表示非循环，为 1 表示循环。

作为状态寄存器时，0 位为 1 表示定时到，为 1 表示计数到；15 位为 0 表示定时/计数未开始，为 1 表示定时/计数开始。

两个寄存器均为只可写，读取后被清 0。

CNT0 初值寄存器(O)/当前值寄存器(I) (FF24H)

CNT1 初值寄存器(O)/当前值寄存器(I) (FF26H)

两个寄存器通过读/写线区别

主要输入输出

输出 rdata,输入 wdata 分别用于 CPU 读写数据

输入 pulse0, pulse1,为两个 CNT 的外部脉冲信号

输出 cout0, cout1 为定时/计数输出

CLK 用于定时，脉冲用于计数

进程描述:

1.reset 信号有效时进行各信号以及寄存器的复位

2.否则进入工作状态

2.1 假如 cs 有效，即这个时候可以进行读写操作

iow 信号的下降沿进行写方式以及写初值操作

根据 address 判断

ff20: CNT0 的方式

ff22: CNT1 的方式

ff20: CNT0 的初值

ff20: CNT1 的初值

ior 信号灯下降沿进行读状态寄存器以及当前值的操作。

ff20: CNT0 的状态

ff22: CNT1 的状态

ff24: CNT0 的当前值

ff26: CNT2 的当前值

否则保持信号

2.2 假如 cs 无效，进入真正的定时或者计数操作

2.2.1 选择 CNT0，根据 ff20 判断是定时/计数，循环/非循环

2.2.1.1 定时：针对 clk

假如当前值等于初值

循环，当前值置零，继续

非循环，置状态值相应位，输出一个 clk 的低电平的 cout0

否则当前值加 1

2.2.1.2 计数：针对 pulse0

假如当前值等于初值

循环，当前值置零，继续

非循环，置状态值相应位

否则当前值加 1

2.2.2 选择 CNT1，根据 ff22 判断是定时/计数，循环/非循环

2.2.2.1 定时：针对 clk

假如当前值等于初值
　　循环，当前值置零，继续
　　非循环，置状态值相应位，输出一个 clk 的低电平的 cout1
否则当前值加 1
2.2.2.2 计数：针对 pulse1
假如当前值等于初值
　　循环，当前值置零，继续
　　非循环，置状态值相应位
否则当前值加 1

PWM　　起始地址 FF30
最大值寄存器（FF30H）(只写）
对比值寄存器（FF32H）(只写）
使能寄存器（FF34H）(只写）
0 位为 0 表示不允许输出脉冲，为 1 表示允许输出脉冲。
输入 d 用来写入最大值和对比值
输出 pwm
进程描述:
1reset 有效：复位各信号
2reset 无效：进入工作状态
2.1cs 有效，iow 的下降沿写各个寄存器
根据 address 判定寄存器
2.2 进行周而复始的计数工作
计数到最大值则转至 0
计数大于对比值输出端输出 0(根据使能寄存器决定是否输出 0)，否则为 1。

4×4 键盘扫描电路　　起始地址 FF30
键值寄存器（FF10H）(只读）
记录按键的键值
状态寄存器（FF12H）(只读）
有键按下时，置 0 位为 1，无键按下时置 0 位为 0。
输出 d 用于读出数据
col 用来表示键盘列输入线
line 用来表示键盘行输入线
进程 1:
产生键盘扫描时钟
进程 2:
1. reset 复位
2. 工作
2.1cs 有效，ior 下降沿根据 address 读键值寄存器 or 状态寄存器
2.2 分 5 个状态
num0 判断是否有键按下，有责转至 num1，否则 num0
num1 防抖动模块，有抖动，转至 num0，否则 num2
num2 产生行扫描信号，转至 num3

num3 根据 lednum 临时保存键值，转 num4
num4 将键值写入键值寄存器，并置状态寄存器

4 位 7 段 LED　　起始地址 FF00

数据锁存器（FF00H），用于写数据
输入 d 用来写数据
led3,led2,lde1,led0 分别用来表示由高到低的 4 位输出
进程描述:
根据 d 分出的四段数据，分别译码，得到各个 led 的输出线

简单 UART　　起始地址 FF40

数据格式固定为 1 位起始位，8 位数据位和 1 位停止位。起始位为 0，停止位为 1。串行输出线空闲状态为 1。

输出锁存器（FF40H）(只用 8 位)
用于串转并暂存输出数据
输入缓冲器（FF40H）(只用 8 位)
用于并转串暂存输入数据
状态寄存器（FF42H）
0 位为 1 表示输出完，1 位为 1 表示输入完
状态寄存器在读取后被清零
发送器，和接收器都需要按外部时钟分频
输出 rdata 用于 CPU 读数据
输入 wdata 用于 CPU 写数据
输入 xtal 外部时钟信号
输出 txd 串行输出
输入 rxd 外部串行输入
进程描述
串转并模块:
核心算法

```
recdata <= rxd&recdata(7 downto 1);
```

并转串模块:
核心算法

```
transdata(8 downto 1) <= ff40(7 downto 0);
 transdata(9) <= '1';
 transdata(0) <= '0';
transdata <= '1'&transdata(9 downto 1);
```

看门狗　　起始地址 FF50

只要 CS, IOW 同时有效，看门狗电路就被复位，看门狗电路访问的端口地址是 FF50H

输入 wdata 用于 CPU 写数据
输出 rst 输出给 CPU 复位信号
进程:
复位后，从 FFFF 进行减 1 计数，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。
写信好使看门狗复位。

本组设计中各个部件的设计与特色

CPU 控制模块采用有限状态机的方式实现多周期信号控制，其中程序地址寄存器 PC 采用下降沿触发方式，在各指令执行的最后一个周期完成 PC 的更新操作，节省了一个周期的取址时间。实现的 31 条指令中，J 为两周期指令，Jrn 是三周期指令，其余均为四周期指令。Load/Store 只用了四个周期即可完成，比一般 CPU 节省了一个周期。

在 16 个寄存器组，由于$12 和$14 为中断专用，用户不能修改，所以特设计了该两个寄存器的写保护。故寄存器组写信号共有 3 条线 WRITEMEM[2..0]，编码如下：(均为上升沿触发)

WRITEMEM[2..0]	功能
100	写$12 和$14 以外的寄存器，若是写$12 或$14 为无效操作
110	写$14
101	写$12
000	读
其他	无效

本组设计的 MIPS16 汇编程序使用手册

根据汇编语言编译器的要求，本实验中汇编程序设计需遵循以下规则：

1.汇编程序开始处需通过伪指令 ORG_DATA 定义所要使用的变量在数据 RAM 中的起始绝对地址，接着可定义任意个所需要的 DW 变量和初始值，也可以不定义

2.定义过变量后，通过伪指令 ORG_CODE 定义下一条语句在程序 ROM 中的绝对地址

3.程序开始处需定义起始标号，程序结束处要有 END 指令，并且后面跟的必须是起始标号

4.语句需遵循实验要求的 31 条指令的格式，编译器还支持 push 和 pop 两条操作堆栈的宏指令，格式为 push $1（把寄存器中的值压入堆栈），pop $2（将栈顶的值写入寄存器中）

5.标号名和语句之间，指令操作码和操作数之间都必须有间隔（空格或制表符），操作数之间要有逗号隔开，分号后面的为注释内容（仅限本行）。立即数如果是十六进制需要在最后加上字母 h 或 H

6.程序最后可以编写 INT0 和 INT1 中断服务程序，以 INT0: 和 INT1: 开始，以 END INT0 和 END INT1 结束

7.程序中可供调用的 BIOS 有五个

（1）格式：KEY16。功能：扫描键盘输入，将输入的值写入 RAM 地址 37B 中

（2）格式：LED16S。功能：将 4 号寄存器的值单独显示在 LED 最低位上，高位为 F

（3）格式：LED16A。功能：将参数存入 4，5，6，7 号寄存器，由低到高依次显示在 LED 上

（4）格式：UART16R。功能：通过串口读入数据并将数据写入 RAM 地址 37C 中

（5）格式：UART16W。功能：通过串口将 4 号寄存器的值输出

8.数据区 RAM 的 380—3FF 为堆栈区，37B—37F 保留给 BIOS 使用，程序员不能使用他们。程序区的 1B8—1FF 保留个 BIOS 程序和中断向量使用，所以用户程序地址不能超越 1B7

本组设计中的 VHDL 程序清单及 GDT 图（如果有）

MIPS16.bdf　　miniMIPS16 总的顶层文件

CPU 模块:

程序名	模块功能
Cpu.bdf	CPU 顶层文件
ALU.vhd	ALU 模块,实现+/-/移位/跳转判断操作
CONTROL.vhd	控制模块，更周期信号控制
memreg.vhd	寄存器组
reg.vhd	PC
MEMorIO.vhd	IO 判断译码模块
MUX.vhd	数据选择器
MUXBUS_JUMP.vhd	JUMP 信号控制的数据选择器
IOBUS.vhd	外围设备输入选择器
shift_left.vhd	左移两位

外围电路:

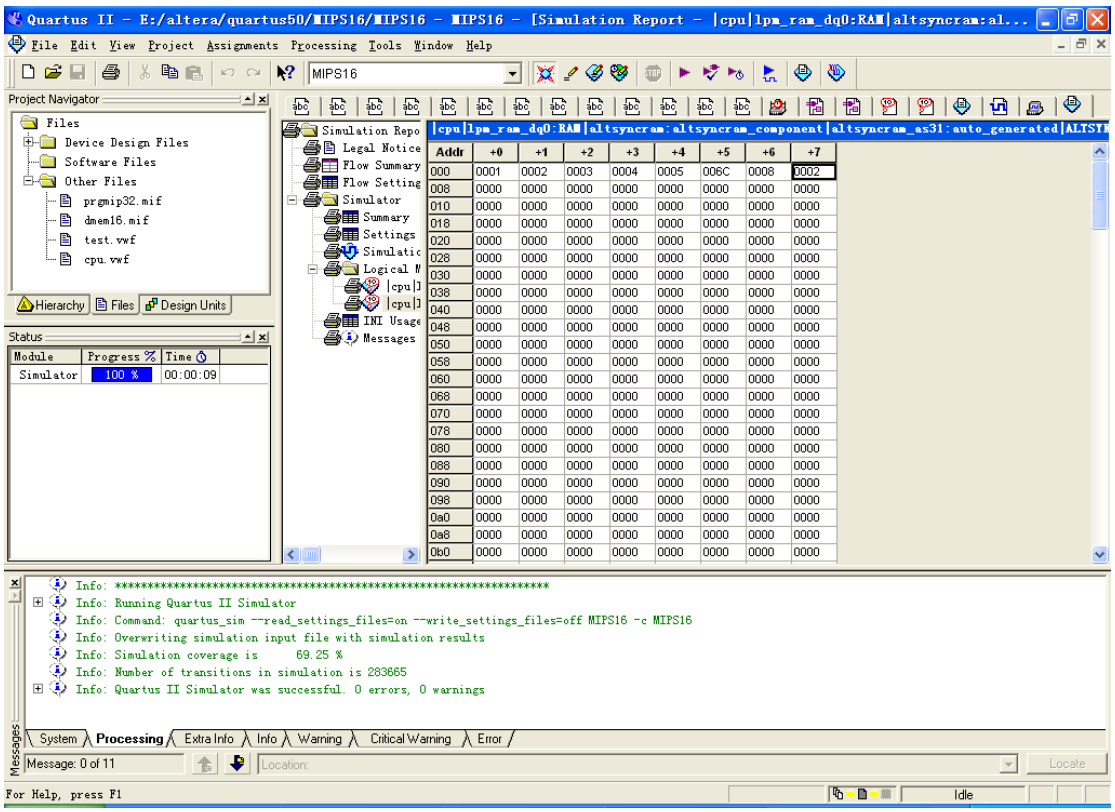
模块名	程序名
LED	led.vhd
Keyboard	keyborad.vhd
定时/计数器	count_time.vhd
PWM	pwm.vhd
串口	uart.vhd　　顶层文件 transmitter.vhd　　数据发送器 receiver.vhd　　数据接收器

prgmip32.mif　　程序存储文件
dmem16.mif　　数据存储文件
cpu.vwf　　CPU 仿真波形
test.vwf　　miniMIPS16 仿真波形

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）

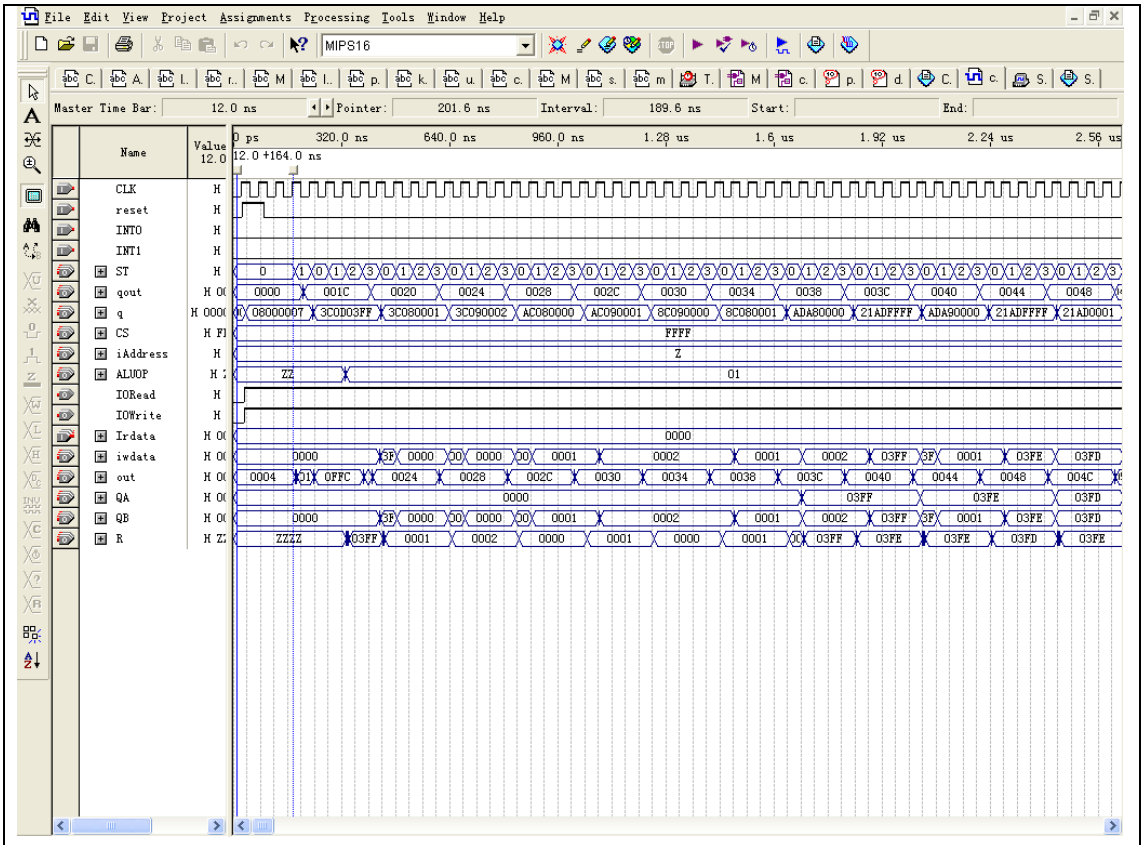
CPU 测试结果及分析:

测试程序详见附录 1。测试程序完成功能是在前 8 个内存中依次写入 1、2、3、4、5，6C,8,2。运行后内存结果如下图所示，与功能吻合。详细分析，见波形分析。

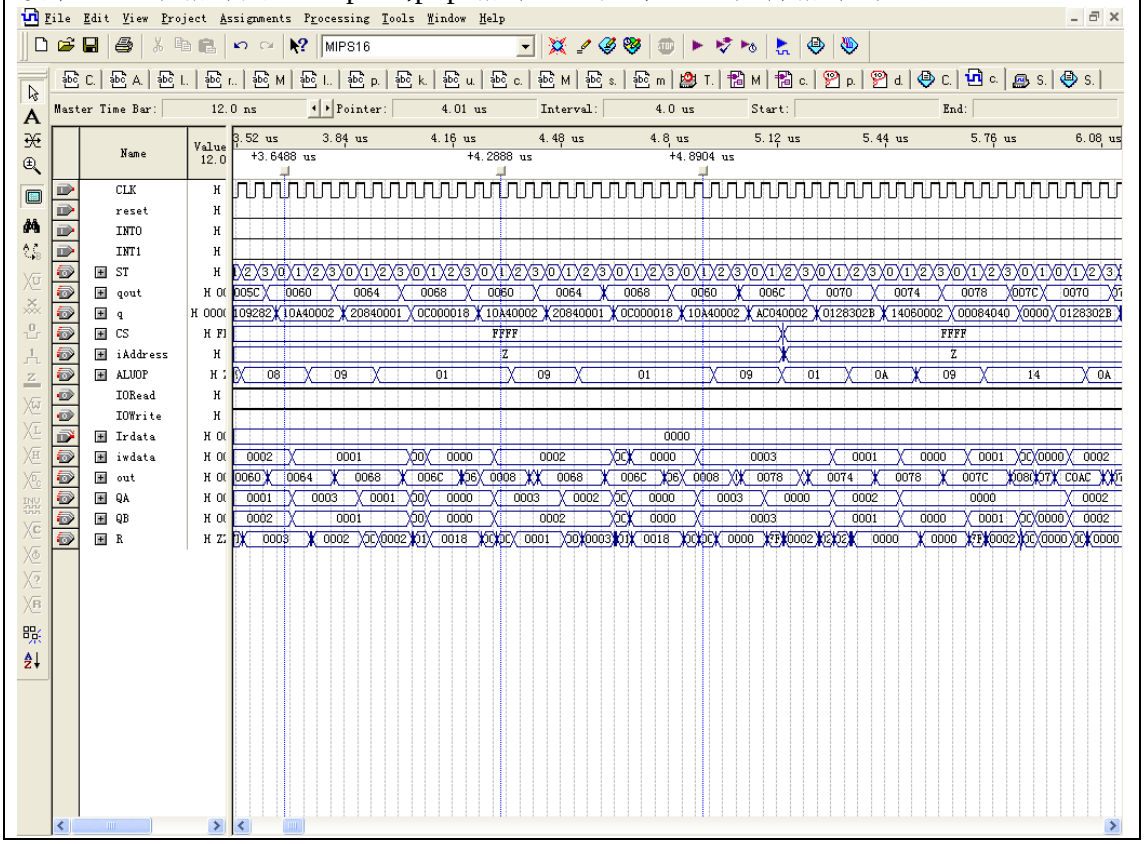


各信号意义:

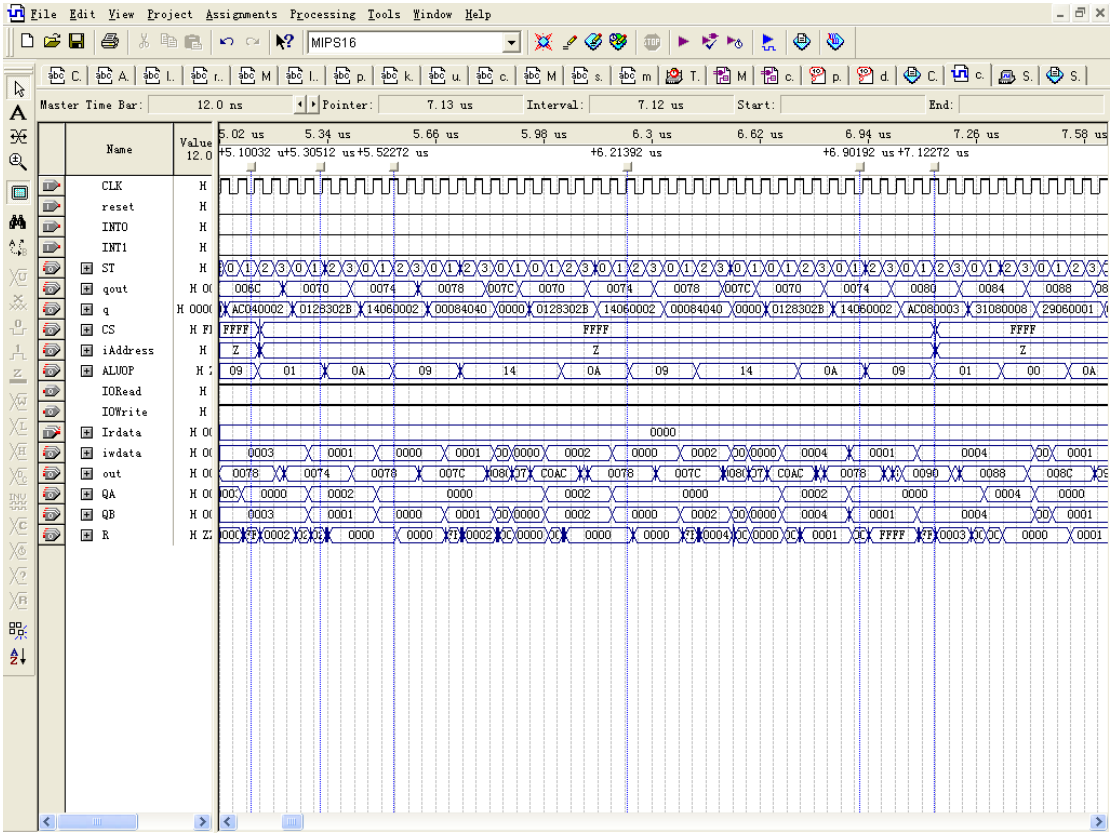
CLK 频率 20MHz， ST 控制模块状态，ALUOP ALU 的运算码，qout 当前 PC 地址，q 当前执行指令， R ALU 运算结果



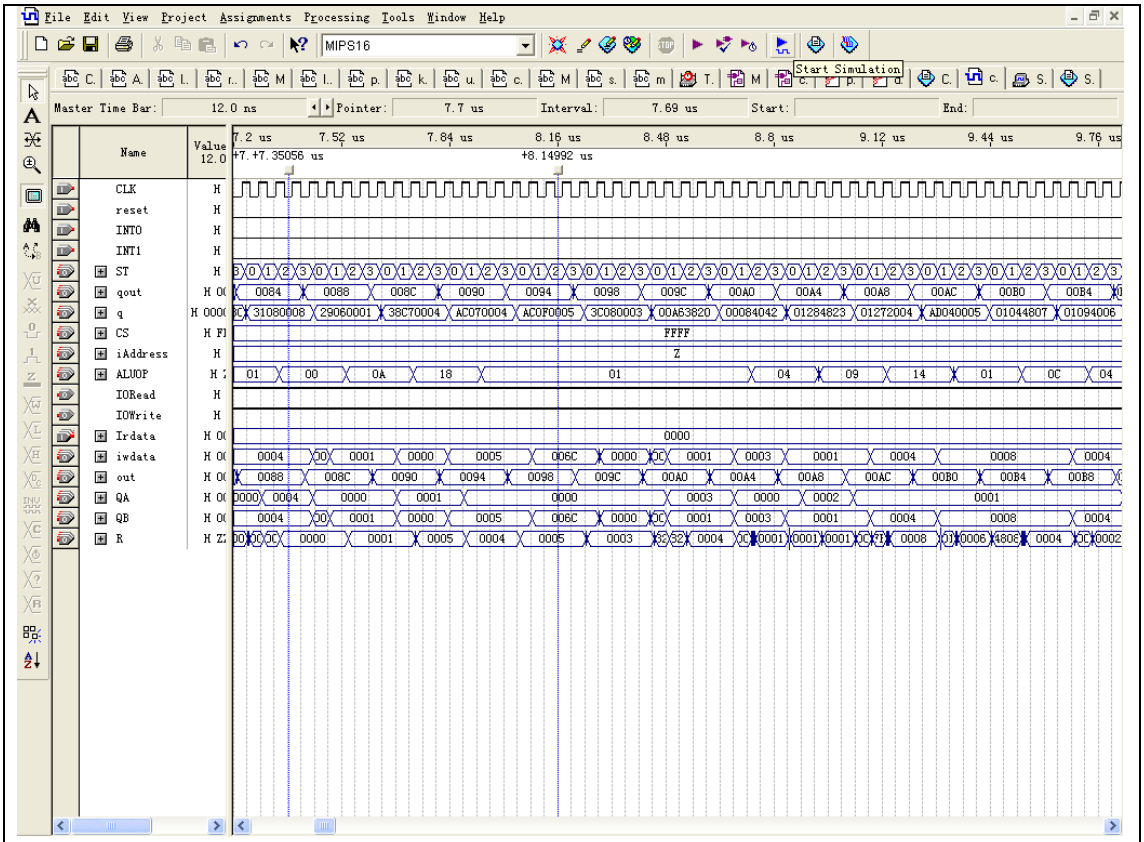
如上图，每一条指令为跳转指令 J，跳到地址 001C，第二条指令是执行的 001C，正确。0020 到 002C 是把 1,2 分别写入内存第 0,1 个单元，最后结果可知正确。接着后八条指令是测试 push,pop 指令，这关系后面跳转指令的正误。



上图，第 006A 指令是 beq $4,$5,2，第一次进来时，$4=1,$5=3,不等，故程序不跳转，下一条为$4=$4+1,循环两次可以跳出循环,如上图示，正确。也可知加法指令正确。



上图，第 006C 指令是在每 2 个内存单元写入 3,正确。下一条，判断$t0<$t1,第一次$t0=1,$t1=2，不跳转，执行 t0=t0<<1;执行两次后 t0=4,退出循环，正确。

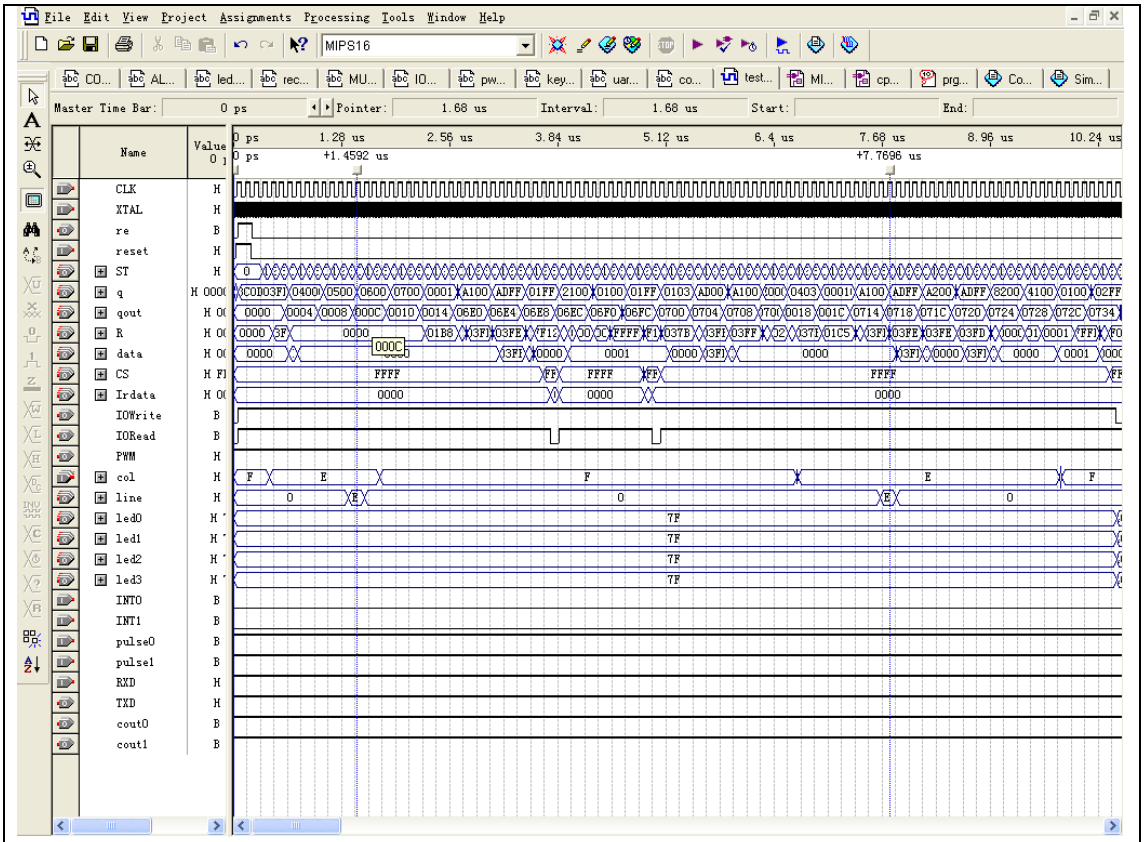


上图中，第 0084 条指令是 andi $t0,$t0,8,由于$t0=4,故结果是 0 正确。第 0094 是把前面 Jal 指令保存下来的地址取出来存入第 6 个内存单元,最后结果可知正确。

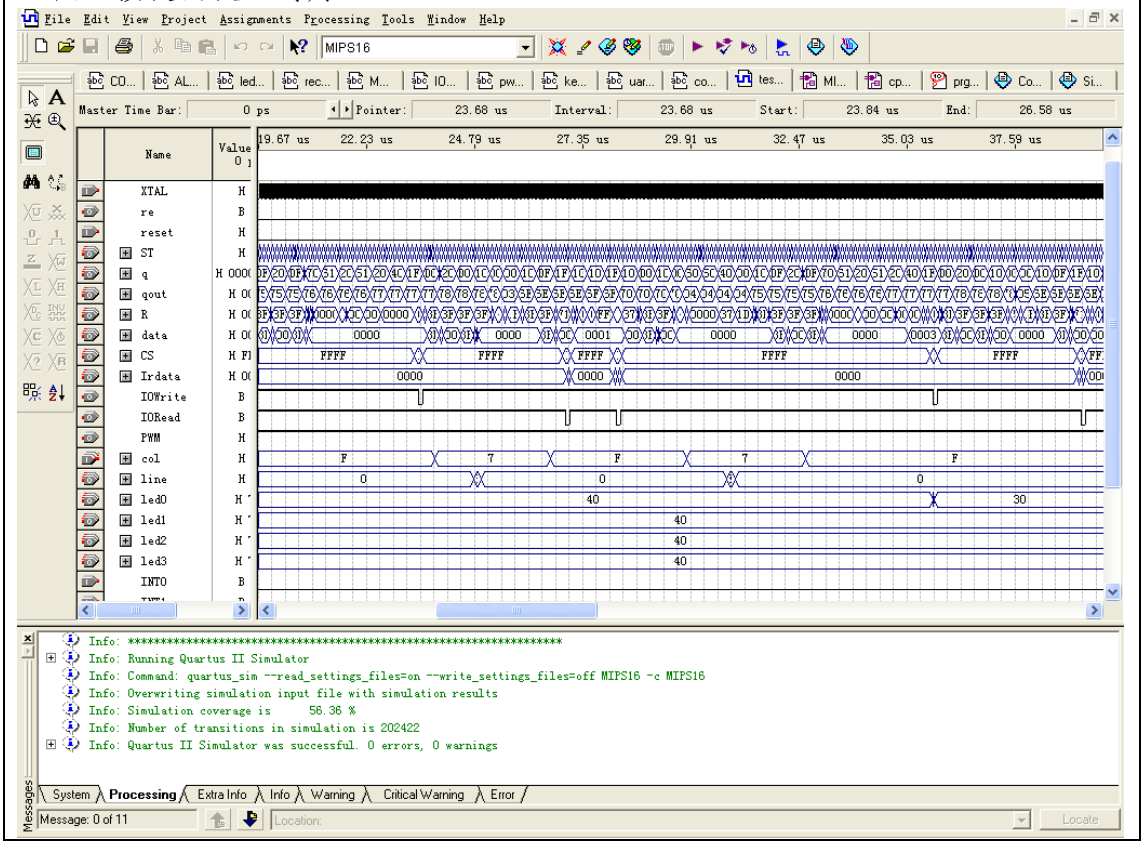
从上面几个测试图可以得到，8 类指令均各自运行正确，总的结果也运行正确。CPU 最高频率可达 35MHz。大部分指令是四周期，速度比较可观。

miniMIPS16 运行测试及分析:

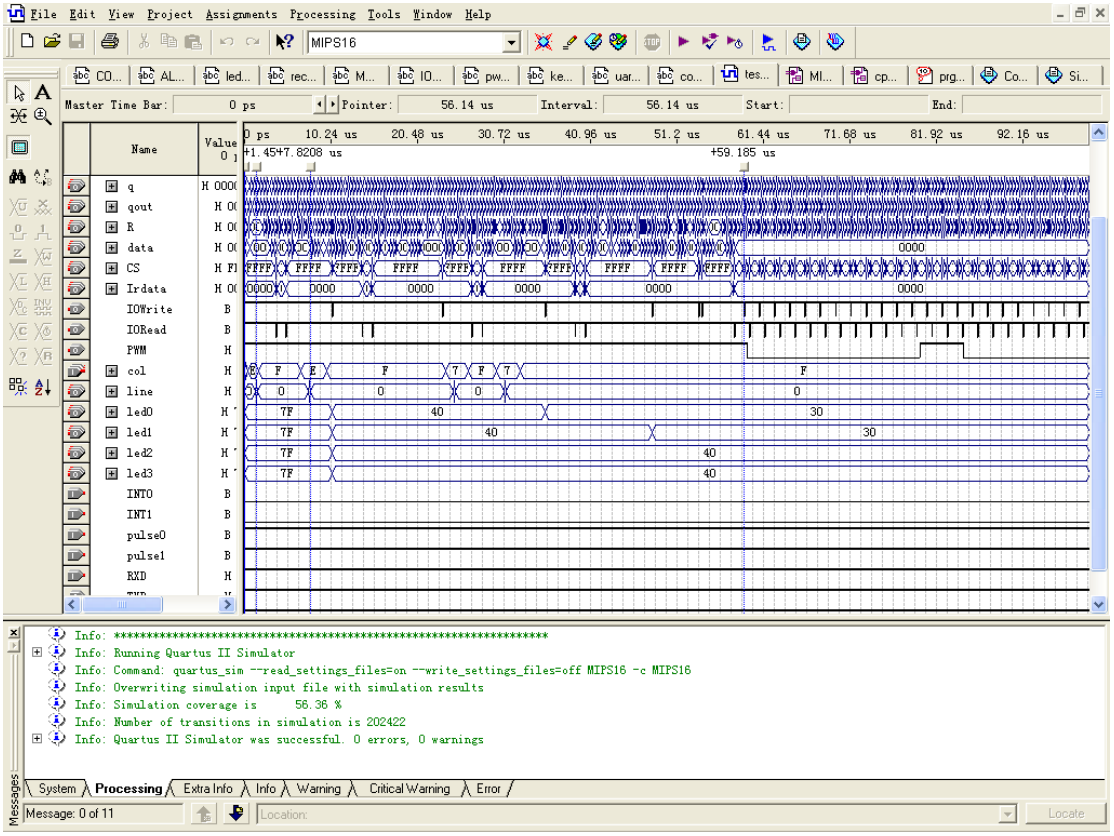
测试程序实现功能：从键盘输入一个 4 位的 16 进制数，将它送给 PWM 的对比值寄存器，并打开使能，输出 PWM 波。默认最大值计数器值是 0FFFFH,测试时为的方便观察，临时改成 00FFH。



上图，按键两次，均为 0。

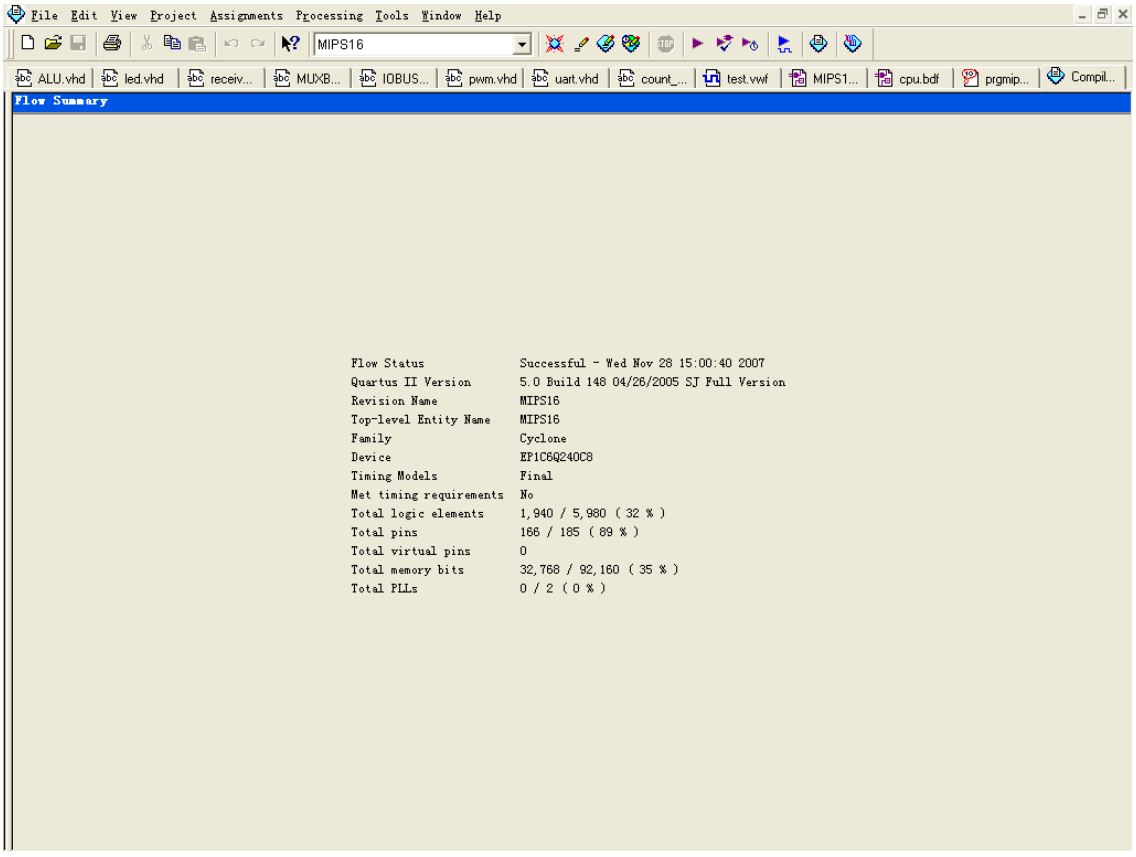


上图，按键两次，均为 3，这样输入的数为 0033H,即在 LED 上显示 0033，且输出占空比为 0033H/00FFH 的 PWM 波。



上图波形显示结果正确，见 led0~3,PWM 信号。

资源使用情况:



课程设计总结（包括设计的总结和还需改进的内容）

通过本次 miniMIPS16 的设计工作，当我们初步了解了设计一台高性能的计算机系统所需采用的方法，更加熟练的掌握了 VHDL 语言，深刻体会了计算机的组织结构与各部件配合协同工作。尽管我们只是实现了 31 条指令，但整个设计过程，让我们学会掌握设计通用 CPU 的步骤，也能体会到不同架构体系及不同的部件性能对 CPU 整体性能的影响。由于是第一次进行 SOC 设计，设计过程中遇到了许多困难，如 CPU 控制信号时序的合理分配，定时／计数器及串口状态寄存器的异步清零问题、IO 片选及读写信号时序设计等，我们通过不断的查阅资料及用多种不同的方法进行实验验证，最终成功解决了绝大部分难题。

在这特别提出的是定时器的状态寄存器清零问题，我们花费了很大精力，但最终未能完全解决。该难题如下：定时／计数器由片选 CS（低电平有效）、写 IOW(下降沿触发)、读 IOR(下降沿触发)来写方式和初值寄存器、读状态寄存器，状态寄存器要在读后清 0。当定时或计数到时，要在状态寄存器相应位置 1。这样如果不加时序控制，有可能会出现清 0 和置 1 同时操作，无法实现功能。我们想到的解决办法是 CS 有效时才能清 0、无效时才能置 1，但这样又影响到定时／计数器的正常工作，在读或写操作时定时／计数器将停止工作。显然，对寄存器的读写应该不影响它的正常工作，所以这是我们设计的定时／计数器的存在的问题。

整个设计过程涉及到《计算机组成原理》、《计算机体系结构》、《微机接口》、《编译原理》等课程的知识，通过这次设计过程，使我们对以前学到的留于理论上的知识有了实实在在的认识，我们所有参与的人都收获良多。

通过这次综合课程设计，也使我们每个人都认识到了团队合作的重要性，特别是软件开发和硬件开发之间的所需要的紧密联系，对我们参与项目的能力的提高很有帮助。

可以改进的地方：

CPU 可以改进成流水线处理提高速度，IO 读写信号时序可以改进得更加合理，顶层文件可以改用 vdhl 文件代替 bdf 文件，更有可移植性，修改也更加方便。

软件编译器可以通过 MFC 包装成可视化界面，增强用户易用性

16 位定时/计数器

在进行读取的时候不可以进行计数，有待改进