東南大學
SOUTHEAST UNIVERSITY

计算机系统综合课程设计

设计报告

组长：施嘉鸿

成员：刘翠翠

吕代超

孙鸿贤

孙芸

陈玮

二0一0 年 一 月

设计名称	基于 Minisys 的多种外围接口				
完成时间	2010-1-19	验收时间	2010-1-19	成绩	

本组成员情况

姓　名	学　号	承　担　的　任　务	个人成绩
吕代超	09006132	CPU 部分的设计	
孙芸	09006138	接口部分：TFT-LCD 彩色大屏接口设计	
孙鸿贤	09006218	编译器的是开发和优化	
陈玮	09006231	IO 口的设计与开发	
刘翠翠	09006304	接口部分：8×8 点阵、步进电机、点阵液晶显示器（黑白屏）的功能设计	
施嘉鸿	09006306	接口部分：标准小键盘、PS2 鼠标 的接口设计和挂接	

本组设计的功能描述（含所有实现的模块的功能）

CPU:

本 CPU 的功能是执行 31 条 MIPS 指令并包含有 rom 和 ram 部分。即，首先从 rom 当中取出预先存在其中的指令包括立即数，之后译码，从寄存器中取出数据并运算。结果根据指令的不同存取 ram 或 I/O 或者回写到寄存器当中。

接口:

1.8×8 点阵：显示 0—9、A—Z、两位的整数等；

2.步进电机、黑白屏：由 cpu 给定一个 10 位的二进制数据作为步进电机的旋转角度和方向，步进电机根据此数据朝着相应的方向（顺时针或逆时针）旋转相应的角度；同时黑白屏将所实现的动作显示出来，假设旋转度数为 θ°（0°≤θ≤360°）具体显示如下：

左边显示——“步进电机正转/反转 X°”

右边显示——笑脸图像

3.TFT－LCD 彩色大屏、PS2 鼠标：通过板子上的 PS2 鼠标接口外接一个标准 PS2 鼠标，通过鼠标的接口程序读到鼠标的各种动作，如：左键、右键、中键值和移动的坐标值，然后将鼠标的动作显示在彩屏上。

按下左键、右键、中键值分别对应着改变屏幕的红色、绿色、蓝色的信号值使屏幕变色。而鼠标也在屏幕上以 10×15 的箭头形式显示在屏幕上并跟着鼠标的移动而移动。

4.键盘、七段数码管：键盘读到一个键值，然后通过中断在七段数码管上显示出来。

编译器：将 C＋＋程序编译成汇编程序并做优化。

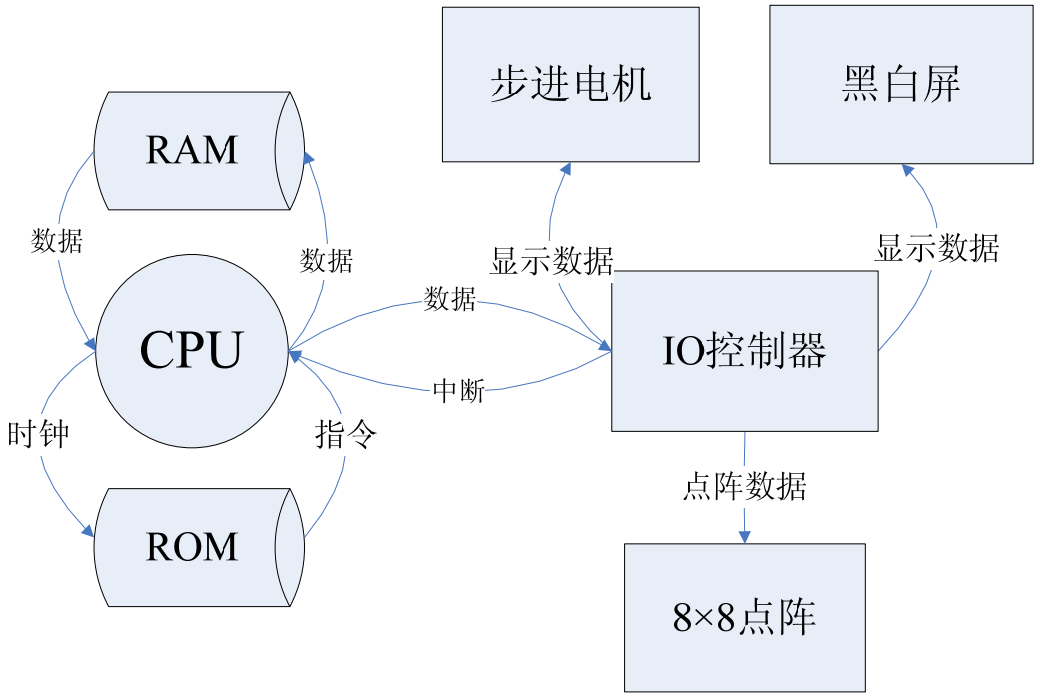
本组设计的主要特色

我们组在完成了 CPU 和编译器等基本功能设计的基础上还利用了板子上大大小小的很多接口使我们这个系统看上去更加的直观，也增强了系统的实用性。使整个系统更加的完整，可视性好。

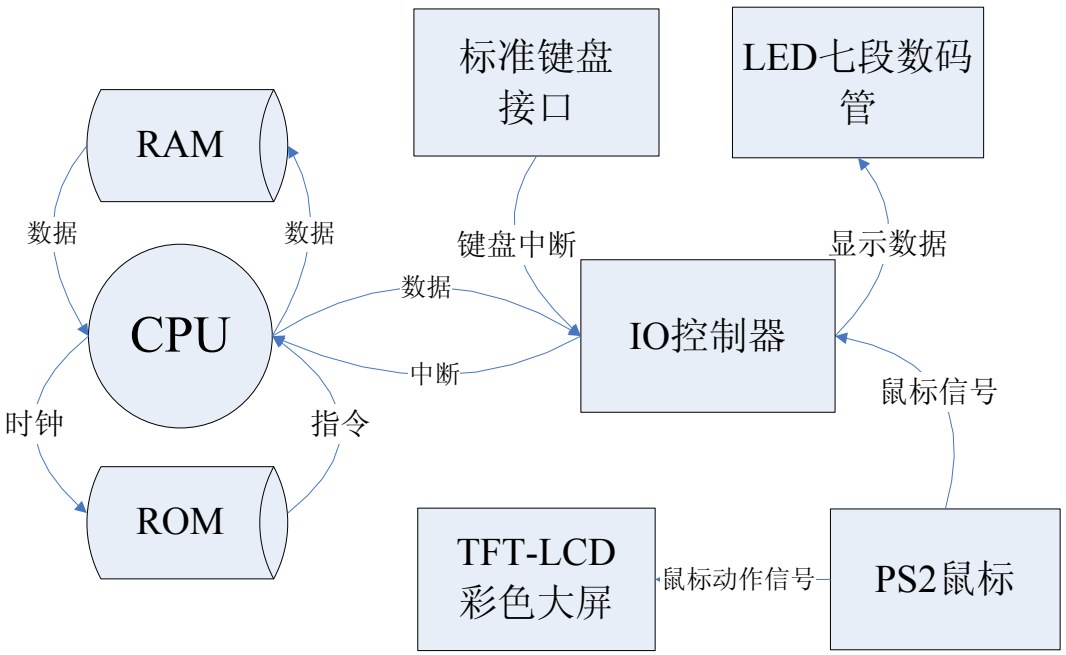
本组设计的体系结构

由于实验台跳线的限制，本组成员将做的接口分成两大组，组成两个体系。

1. CPU、8×8 点阵、步进电机、点阵液晶显示器（黑白屏）：



2. CPU、TFT－LCD 彩色大屏、PS2 鼠标、键盘、七段数码管



本组设计中各个部件的设计与特色概述（含关键代码）

1. CPU:

本 CPU 采用的是 minisys 的标准 5 级流水线设计。即，取指，译码，执行，存储和写回五级。流水线是 Intel 首次在 486 芯片中开始使用的。流水线的工作方式就象工业生产上的装配流水线。在 CPU 中由 n 个不同功能的电路单元组成一条指令处理流水线，然后将一条指令分成 n 步后再由这些电路单元分别执行，这样就能实现在一个 CPU 时钟周期完成一条指令，因此提高 CPU 的运算速度。如下图所示：

可以看到，在一个时钟周期内五个部件都在工作，这使得 CPU 的工作效率明显提高了，而理论上时钟周期可以减小 5 倍。然而，事实并非如此，5 部分的门延时并不相等，甚至可以多达数十倍的差别，所以流水线的主频取决于最慢的一个部件的延时大小，所以平均各个部件显然是提高流水线 CPU 的主要途径之一。然而，当遇到跳转指令的时候，整个流水线之前的部分都要被清空，而重新装填，这显然将会影响到 CPU 的效率，而如果一个应用程序的跳转指令非常多的话，那甚至有可能流水线的 CPU 的效率还不如单周期 CPU。解决这个问题的方案是提前对指令进行判断，判断其是否是跳转指令，而做出清空之前的流水线的操作。可是，这种 5 级流水线 CPU 的延时主要集中在前两个部件之间，如果将判断（特别是分支跳转的判断）的压力都集中在前两个部件的话，将使得前两个部件的延时更加长，从而影响整个 CPU 的性能。这里将是一个两难的选择。

另一个亟待解决的问题是各种相关的问题，首先是结构相关，存储器的同时存取会造成这类相关，但是本设计将 rom 和 ram 分开，所以不会出现同时存取数据的情况，所以在本设计当中这类相关被天然的排除掉了。其次是数据相关，当指令在流水型中重叠执行的时候，流水线有可能改变指令读写操作数的顺序，使得其顺序与程序本身的操作顺序不一致而造成结果失误。造成这一结果的原因是某条指令的操作数依赖前一条或者前几条指令的运行结果，这就是所谓的数据相关。

数据相关分为 3 种。写后读（RAW），如果后一条指令的操作数依赖前一指令的结果，那么在流水线中就会出现写后读相关性问题。例如：

```
addi    $1,$2,10
sub     $5,$1,$4
```

这两条指令在流水线中就会出现 RAW 相关。显然，第二条指令中$1 寄存器的值需要等待第一条指令的结果出来之后，但是在流水线中，之灵异的结果需要至少在第 4 个周期之后才会被赋值到$1，而第二条指令的读取在第 3 个周期之前就将完成，那么显然会出现相关性问题。

读后写（WAR）在 minisys5 级流水当中写操作必会晚于或等于读操作，所以不在这个问题。

写后写（WAW）在 minisys5 级流水当中只有一个部件可以写一种存储器（寄存器），所以本设计也不会存在这个问题。

控制相关，所谓控制相关就是遇到分支跳转指令的时候，条件分支会修改 PC 的值，那么就会出现两种情况，一个是 PC 值顺次到下一个地址而不做修改，另一种是重新赋值到分支目标地址。所以在没有分支预测机制的情况下，我们默认 PC 值将不作修改。因为不做修改的开销小于修改 PC 值的开销，因为修改 PC 值将至少清空一个流水级。我们没有必要为每一个分支语句承

担这个开销。

为了构造流水线并解决这些相关问题。我们需要在每一级之间定义 4 组寄存器：
IF/ID 寄存器：这个寄存器主要保存取指所得到的指令
定义如下：

```
if( RST == 1'b1 )
begin
    PC[17:0] = 18'h00000;
    Instruction = 32'h00000000;
end

else if( Jrn || Jal || Jmp || Wrong )
begin
    PC[17:0] = { (( Jmp == 1'b1 ) || ( Jal == 1'b1 )) ? Instruction[15:0] :
(( Wrong == 1 ) ? ( PC[15:2] + AddResult - 1 ) : (( Jrn == 1'b1 ) ? JrnPC : ( PC[17:2] + 1'b1 ))), 2'b00 };
    Instruction <= 32'h00000000;
end

else if( Pause )
begin
    PC[17:0] = PC;
end

else
begin
    Instruction = InstructionTemp[31:0];
    PC[17:0] = { ( PC[17:2] + 1'b1 ), 2'b00 };
end
```

ID/EXE 寄存器：用来存储译码后得到的各种控制信号和取到的操作数
定义如下：

```
if( !Pause && !Wrong )
begin
    Shamt <= Instruction[10:6];
    RegAdd <= ( Instruction[31:26] == 6'b101011 ) ? 5'b00000 : (( Jal == 1'b1 ) ?
        5'b11111 : (( Instruction[31:26] == 0 ) ? Instruction[15:11] : Instruction[20:16] ));
    ExeOpCode <= Instruction[31:26];
    FuncOpCode <= Instruction[5:0];
    SignExtend[31:16] <= (( Instruction[31:26] >= 6'b001011 )&&( Instruction[31:26] <= 6'b001110 ))
        ? 16'h0000
        : { sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign,sign };
    SignExtend[15:0] <= Instruction[15:0];
    AAdd <= Instruction[25:21];
    BAdd <= Instruction[20:16];
    SrcA <= A;
```

```
            SrcB <= B;
        end
        else
        begin
            RegAdd <= 5'b00000;
            ExeOpCode <= 6'b000000;
            FuncOpCode <= 6'b000000;
            SrcA <= 32'b00000000;
            SrcB <= 32'b00000000;
        end
```

EXE/MEM 寄存器：用来存储运算结果和控制信号

定义如下：

```
    RMem <= ( ExeOpCode == 6'b100011 ) ? 1'b1 : 1'b0;
    WMem <= ( ExeOpCode == 6'b101011 ) ? 1'b1 : 1'b0;
    RegWrAdd <= RegAdd;
    WrMemData <= ReadDataB;
    WReg <= WRegTemp;
    ALUResult = ALUTemp;
```

MEM/WB 寄存器：用来存储访存结果和控制信号

定义如下：

```
    WBData <= ALUResult;
    ReadMem <= RMem;
    WBAdd <= RegWrAdd;
    WriteReg <= Wreg;
```

解决数据相关的问题时，本设计使用的是定向转发法，即：尽管一条指令的结果在回写阶段才会被写入寄存器，可是实际上对于非存储器指令来说，在 EXE 阶段就这个结果已经存在了，而对于存储器指令来说，这个结果也在 MEM 阶段中就产生了，所以我们不需要等待一条指令到回写阶段在获取其结果，这是定向转发法的基本原理。而对于本设计来说，我们作如下的判断：(1)当 A 或 B 操作数的地址与下一条指令的 ALU 输出的值的地址相同时，将 ALU 的临时结果赋值给 A 或 B。(2)当 A 或 B 操作数的地址与下一条指令的 MEM 输出结果的地址相同时，则等待一个周期一边进入(3)情况。(3)当 A 或 B 操作数与下下条指令的 MEM 输出结果相同时，则将其赋值给 A 或 B。(4)当 A 或 B 操作数与 WB 阶段中即将存入寄存器的值相同时，将其赋值给 A 或 B。代码如下：

```
assign Pause = RMem & (( RegWrAdd == AAdd ) | ( RegWrAdd == BAdd ));
assign A = (( RegWrAdd == AAdd ) && ( RegWrAdd != 5'b00000 ) && WReg ) ? ALUResult :
    ((( WBAdd == AAdd ) && ( WBAdd != 5'b00000 ) && ReadMem ) ? ReadData :
    ((( WBAdd == AAdd ) && ( WBAdd != 5'b00000 ) && !ReadMem ) ? WBData :
    ((( LastAdd == AAdd ) && ( LastAdd != 5'b00000 ) && LastWB ) ? LastData : SrcA )));
assign B = (( RegWrAdd == BAdd ) && ( RegWrAdd != 5'b00000 ) && WReg ) ? ALUResult :
    ((( WBAdd == BAdd ) && ( WBAdd != 5'b00000 ) && ReadMem ) ? ReadData :
    ((( WBAdd == BAdd ) && ( WBAdd != 5'b00000 ) && !ReadMem ) ? WBData :
    ((( LastAdd == BAdd ) && ( LastAdd != 5'b00000 ) && LastWB ) ? LastData : SrcB )));
```

```
assign ReadDataA = Pause ? 32'b00000000 : A;
assign ReadDataB = Pause ? 32'b00000000 : B;
```

而在解决控制相关问题的时候，本设计将判断分支的功能放在 ID 模块当中，这样会比普通的放到 EXE 中的情况对于跳转的代价要小 50%。而却导致 ID 模块延时增加使得主频下降了 10%。但是本设计只要是为了外设服务的，而外设的汇编程序大量使用分支判断。这样来说选择降低跳转的代价是更加明智的选择。

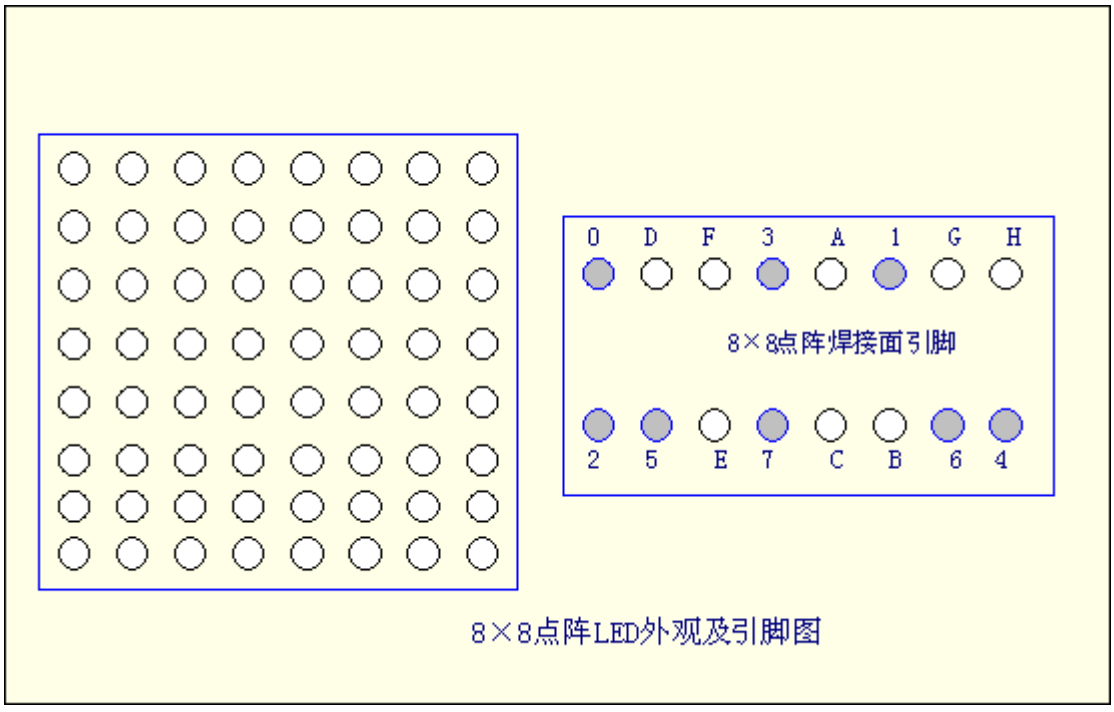
而其他各个模块的设计与标准的 minisys 系统区别不大，这里就不赘述了。

2. 接口:

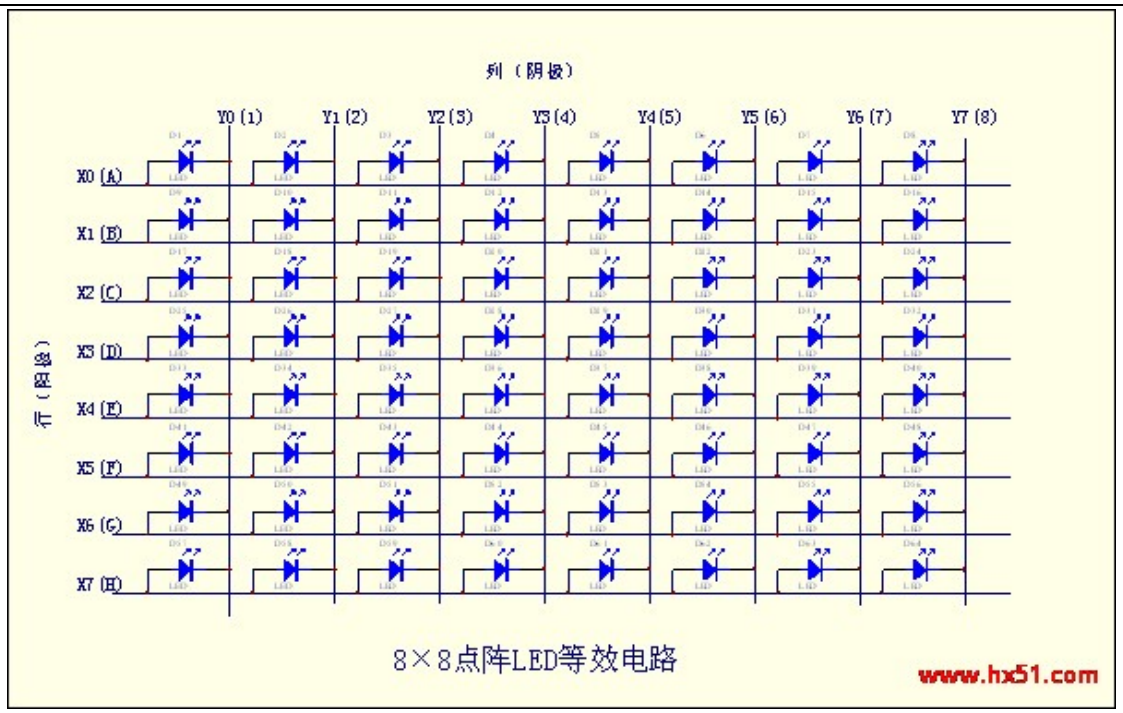
1. 8*8 点阵

工作原理

图（1）为 8×8 点阵 LED 外观及引脚图，其等效电路如图（2）所示，只要其对应的 X、Y 轴顺向偏压，即可使 LED 发亮。



图(1)



图(2)

点阵 LED 扫描法介绍

实验中采用行扫描方式，即先选择行，再对该行的 8 个点赋值。1 表示点灭，0 表示点亮。扫描频率大小要合适才能有很好的效果。如果太小，而每行点列开启的时间大于人眼的视觉暂停时间，那么会产生闪烁现象；而扫描频率太大，则会造成 LED 的频繁开启和关断，大大增加 LED 功耗（开启和关断的时刻功耗很大）。一般来说，扫描频率选在 50Hz 比较合适。

引脚设置

静态数码管、动态数码管和8×8点阵是共用16位IO，所以在某一个时刻只能使一个接口部件使能，即四位开关只能有一位在ON的位置（右边）。默认全部off，即在左边。1、表示选择点阵；2表示选择动态的数码管；3、表示选择静态的数码管。

row（8位） <--->行选择

data（8位）<--->对应选择行的8位数据值

实验内容

可以显示 0—9、A—Z、两位的整数等。

2. 步进电机

工作原理

1、步进电机概述

步进电机是一种能够将电脉冲信号转换成角位移或线位移的机电元件，它实际上是一种单相或多相同步电动机。单相步进电动机有单路电脉冲驱动，输出功率一般很小，其用途为微小功率驱动。多相步进电动机有多相方波脉冲驱动，用途很广。使用多相步进电动机时，单路电脉冲信号可先通过脉冲分配器转换为多相脉冲信号，在经功率放大后分别送入步进电动机各相绕组。每输入一个脉冲到脉冲分配器，电动机各相的通电状态就发生变化，转子会转过一定的角度（称为步距角）。正常情况下，步进电机转过的总角度和输入的脉冲数成正比；连续输入一定频率的脉冲时，电动机的转速与输入脉冲的频率保持严格的对应关系，不受电压波动和负载变化的影响。由于步进电动机能直接接收数字量的输入，所以特别适合于微机控制。

3、步进电机的工作原理

现以反应式三相步进电机为例说明其工作原理。定子铁心上有六个形状相同的大齿，相邻两个大齿之间的夹角为60度。每个大齿上都套有一个线圈，径向相对的两个线圈串联起来成为一相绕组。各个大齿的内表面上又有若干个均匀分布的小齿。转子是一个圆柱形铁心，外表面上圆周方向均匀的布满了小齿。转子小齿的齿距是和定子相同的。设计时应使转子齿数能被二整除。但某一相绕组通电，而转子可自由旋转时，该相两个大齿下的各个小齿将吸引相近的转子小齿，使电动机转动到转子小齿与该相定子小齿对齐的位置，而其它两相的各个大齿下的小齿必定和转子的小齿分别错开正负1/3的齿距，形成“齿错位”，从而形成电磁引力使电动机连续的转动下去。

和反应式步进电动机不同，永磁式步进电动机的绕组电流要求正，反向流动，故驱动电路一般要做成双极性驱动。混合式步进电动机的绕组电流也要求正，反向流动，故驱动电路通常也要做成双极性。

3、开发板中步进电机控制的实现

本开发板中使用的步进电机为四相步进电机。转子小齿数为64。

系统中采用四路I/O进行并行控制，FPGA直接发出多相脉冲信号，在通过功率放大后，进入步进电机的各相绕组。这样就不再需要脉冲分配器。脉冲分配器的功

能可以由纯软件的方法实现。

本系统中采用的是四相单、双八拍控制方法步距角的计算公式为：

$$\theta b = 360° / mCZ_k$$

其中：m为相数，控制方法是四相单四拍和四相双四拍时C为1，控制方法是四相单、双八拍时C为2，Zk为转子小齿数。 所以步距角为360°/512。但步进电机经过一个1/8的减速器引出，实际的步距角应为360°/512/8，即45°/512。

开发平台中使用EXI/O的高四位控制四相步进电机的四个相。按照四相单、双八拍控制方法，电机正转时的控制顺序为A→AB→B→BC→C→CD→D→DA。EXI/O的高四位的值参见下表。

电机正转时，FPGA四位IO口的值 十六进制	二进制	通电状态
1H	0001	A
3H	0011	AB
2H	0010	B
6H	0110	BC
4H	0100	C
CH	1100	CD
8H	1000	D
9H	1001	DA

反转时，只要将控制信号按相反的顺序给出即可。

引脚设置

步进电机共有 4 个输出引脚 StepDriver[0]- StepDriver[3],对应着四位 IO 口的值

实验设计

对于 cpu 给定的 10 位输入数据：

最高位表示旋转方向：0——正转（顺时针旋转），1——反转（逆时针旋转）。4 位 IO 口共有 8 种状态，若是正转，则按照 A→AB→B→BC→C→CD→D→DA 的顺序循环给 StepDriver 赋值；若是反转，则按照 A→AD→D→DC→C→CB→B→BA 的顺序循环给 StepDriver 赋值。

低 9 位表示旋转角度：旋转度数通过计数实现，假设旋转角度为 θ，又有步距角为 45°/512，所以需要计数 512θ/45 次，当计数完毕，步进电机停止旋转。

3. 点阵液晶显示器

工作原理

1. LCD 的显示原理

液晶显示器件的显示原理为：在外加电场的作用下，具有偶极距的液晶棒状分子在排列状态上发生变化，使得通过液晶显示器件的光被调制，从而呈现明或暗或透过与不透过的显示效果。液晶显示驱动器的功能就是建立这种电场

2. C12864-1 128×64DOTS

在我们实验中用到的 C12864-1 是一种图形点阵液晶显示器，它主要由行驱动器 / 列驱动器及 128×64 全点阵液晶显示器组成。可完成图形显示，也可以显示 8×4 个（16×16 点阵）汉字。

主要技术参数和性能：

(1). 显示内容：128（列）×64（行）个点

(2).C12864-1 自带七种指令，完成各种操作，以实现图形或汉字的显示。

(3). C12864-1 与 CPU 接口采用 8 位数据总线 data 并行输入输出和 3 条控制线：R/W（读写控制信号）——R/W = 1，读信号；R/W = 0，写信号。

D/I（指令数据控制信号）——D/I = 1，数据操作；D/I = 0，指令操作。

E（使能信号）——E 信号下降沿各种指令有效

(4). IC 主要有以下功能：

①. 指令寄存器 IR

IR是用于寄存指令码，当R/W＝0时，在E信号下降沿的作用下，指令码写入IR。

②. 数据寄存器 DR

DR是用于寄存数据，当R/W＝1时，在E信号下降沿作用下，图形显示数据写入DR，或在E信号高电平作用下由DR读到DB7～DBO数据总线。DR和DDRAM之间的数据传输是模块内部自动执行的。

③. 忙标志BF

BF 标志提供内部工作情况。BF＝1表示模块在内部正在进行操作，此时模块不接受外部指令和数据。BF＝0时，模块为准备状态，随时可接受外部指令和数据。

利用指令表中的“读状态”指令，可以将BF读到DB7总线，可以检验到模

块的工作状态。

④． 显示控制触发器DFF

此触发器是控制模块屏幕显示开和关。DFF＝1为开显示，DDRAM的内容就显示在屏幕上，DFF=0为关显示。DDF的状态是指令“显示开关控制”指令和RST信号控制的。

⑤． Y地址计数器

XY地址计数器是一个9位计数器。高3位是X地址计数器，低6位为Y地址计数器，XY地址计数器实际上是作为DDRAM的地址指针，X地址计数器为DDRAM的页指针，Y地址计数器为DDRAM的Y地址指针。

X地址计数器是没有记数功能的，只能用指令设置。

Y地址计数器具有循环记数功能，和显示数据写入后，Y地址自动加1，Y地址指针从0到63。

⑥． 显示数据RAM（DDRAM）

DDRAM是存储图形显示数据的。数据为1表示显示选择，数据为0表示显示非选择。DDRAM与地址和显示位置的关系见DDRAM地址表示。

	CS1＝1					CS2＝1					
Y＝	0	1	‥	62	63	0	1	‥	62	63	行号
	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	0～7
X＝0～X＝7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	8～55
	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	DB0～DB7	56～63

⑦． 地址计数器

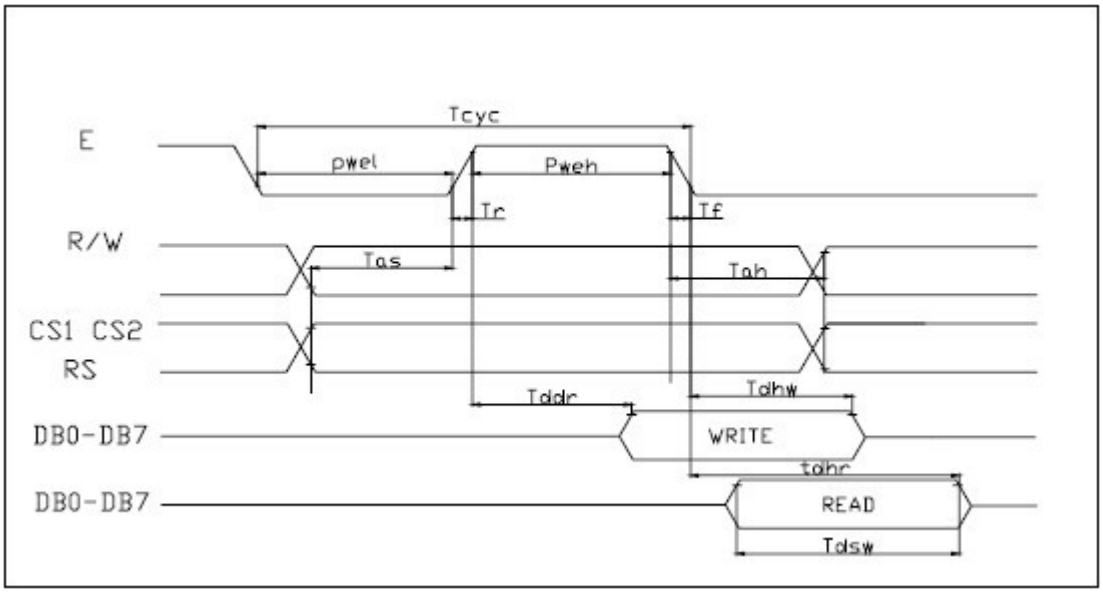
Z地址计数器是一个6位计数器，此计器具备循环记数功能，它是用于显示行扫描同步。当一行描完成，此地址计数器自动加1，指向下一行扫描数据，RST复位后Z地址计数器为0。

Z地址计数器可以用“置显示起始行”指令 预置。因此，显示屏幕的起始行就由此指令控制，即DDRAM的数据从哪一行开始显示在屏幕的第一行。此模块的DDRAM共64行，屏幕可以循环滚动显示64行。

(5). 指令格式及指令表

指令	指令码										功能
	R / W	D / I	D7	D6	D5	D4	D3	D2	D1	D0	
显示 ON/OFF	0	0	0	0	1	1	1	1	1	1 / 0	控制显示器的开关，不影响 DDRAM 数据和内部状态
显示起始行	0	0	1	1	显示起始行（0～63）						指定显示屏从 DDRAM 中哪一行开始显示数据
设置 X 地址	0	0	1	0	1	1	1	X：0…7			设置 DDRAM 中的页地址（X 地址）
设置 Y 地址	0	0	0	1	Y 地址（0～63）						设置地址（Y 地址）
0 读状态	1	0	BUSY	0	ON / OFF	RST	0	0	0	0	读取状态 RST 1：复位 0：正常 ON / OFF 1：显示开 0：显示关 BUSY 0：READY 1：INOPERATION
写显示数据	0	1	显示数据								将数据线上的数据 DB7～DB0 写入 DDRAM
读显示数据	1	1	显示数据								将数据线上的数据 DDRAM 写入 DB7～DB0

3. 读写操作时序



名　称	符号	最小值	典型值	最大值	单位
E 周期时间	Tcyc	1000	---	---	ns
E 高电平宽度	Pweh	450	---	---	ns
E 低电平宽度	Pwel	450	---	---	ns
E 上升时间	Tr	---	---	25	ns
E 下降时间	Tf	---	---	25	ns
地址建立时间	Tas	140	---	---	ns
地址保持时间	Tah	10	---	---	ns
数据建立时间	Tdsw	200	---	---	ns
数据延迟时间	Tddr	---	---	320	ns
写数据保持时间	Tdhw	10	---	---	ns
读数据保持时间	Tdhw	20	---	---	ns

4. TFT－LCD 彩色大屏

摘要：参照 SHARP 公司 LQ080V3DG01 TFT-LCD 的逻辑要求和控制时序要求，设计了并制作了 FPGA 控制驱动的显示系统，通过鼠标点击来点亮要显示的小块区域。

关键字：FPGA；夏普；TFT-LCD；鼠标；

1、引言

FPGA 是作为专用集成电路(ASIC)领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。因具备接口、控制、功能 IP、内嵌 CPU 等特点可以用于实现一个构造简单，固化程度高，功能全面的系统产品。随着科技的发展，FPGA 在社会各个方面的实际应用越发广泛。FPGA 在通信、数据处理、网络、仪器、工业控制、军事和航空航天等众多领域得到了广泛应用。TFT-LCD 是数字终端显示最理想的显示器件之一，随着 TFT 彩色液晶显示技术日益成熟，因其功耗、体积、重量、可靠性等性能指标均优于传统的 CRT 显示器，现已被广泛应用于各类图形显示系统。

2、开发环境：

处理器 CycloneⅡ EP2C35F672C8N；鼠标：惠普 M-SBF96；

TFT-LCD：夏普 LQ080V3DG01:；电源： 5V ；

3、器件介绍

LQ080V3DG01 是一种用非晶硅 TFT 作为开关器件的有源矩阵液晶显示器，该模块包括 TFT-LCD 显示屏，驱动电路和背光板，其接口使用 TTL 电平。显示分辨率为 640×480 像素，RGB 数据为均为 6 位，18bit 数据显示颜色为 262144 色，4 个时序驱动信号，液晶屏可以使用 5v 或 3.3v 电压驱动，采用开关电源作为背光驱动电源，其性能很适合 FPGA 的应用场合。

4、功能描述

本实验开发了 LQ080V3DG01 的显示驱动程序，即当鼠标点击液晶屏上任一位置时，将所指定的那一小块区域点亮。

5、部件设计

• 液晶模块

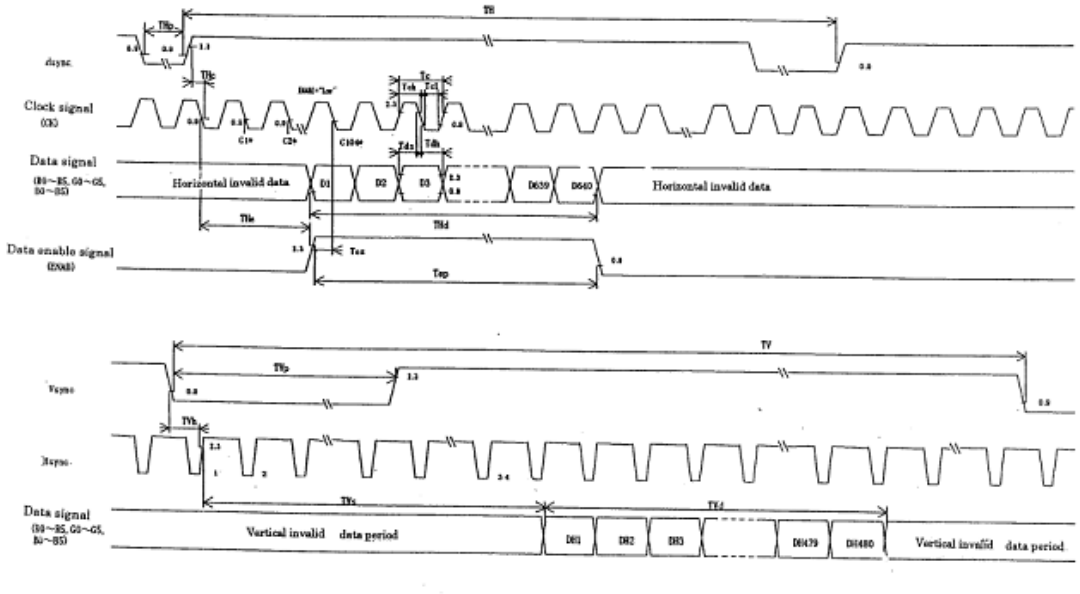
（1）TFT-LCD 面板驱动部分：

引脚号	标识	功能	引脚号	标志	功能
1	GND	地	17	G4	绿色数据信号
2	CK	时钟信号	18	G5	绿色数据信号（高位）
3	Hsync	水平同步信号	19	GND	
4	Vsync	垂直同步信号	20	B0	蓝色数据信号（低位）
5	GND	地	21	B1	蓝色数据信号
6	R0	红色数据信号（低位）	22	B2	蓝色数据信号
7	R1	红色数据信号	23	B3	蓝色数据信号
8	R2	红色数据信号	24	B4	蓝色数据信号
9	R3	红色数据信号	25	B5	蓝色数据信号（高位）
10	R4	红色数据信号	26	GND	地
11	R5	红色数据信号（高位）	27	ENAB	数据始能信号
12	GND	地	28	Vcc	电源供应
13	G0	绿色数据信号（低位）	29	Vcc	电源供应
14	G1	绿色数据信号	30	R/L	水平扫描方向控制信号
15	G2	绿色数据信号	31	U/D	垂直扫描方向控制信号
16	G3	绿色数据信号	32	GND	地

（2）输入信号的时序特征：

参数	符号	最小值	典型值	最大值	单位
时钟频率	CLK		25.18	28.33	MHz
行同步周期	TH	700	800	900	CLK
列同步周期	TV	512	525	560	TH
水平显示宽度	THd	640	640	640	CLK
垂直显示宽度	TVd	480	480	480	TH
使能信号宽度	Tep	2	640	640	CLK
行负脉冲宽度	THp	2	96	200	CLK
列负脉冲宽度	TVp	1		34	TH
使能信号与行信号距离	The	44			CLK
列扫描与场周期信号距离	TVs		34		TH

根据 LQ080V3DG01 的时序信号特征表可以描绘出 TFT-LCD 的时序图，如下：



（3）驱动程序设计

根据是上述时序图，可以为 FPGA 设计出驱动程序模块，LQ080V3DG01 是 18 位彩色的信号接口液晶模块，根据实际需要，我们只取了其中的 16 位 RGB 信号线。除此之外，还需要以更行同步信号线，一根列同步信号线，一根使能信号线，一根时钟信号线，这样一共 20 根线号线。

①行同步程序设计：

根据时序特性，本实验选取的 LCD 时钟是 2 个系统时钟，用 verilog HDL 写的部分行同步周期时序程序如下：

生成 LCD 时钟信号：

```
always @(posedge CLK or negedge RST)
begin
    if(!RST)
        TLcdClk <= 1'b0;
    else
        TLcdClk <= !TLcdClk;
end
```

生成 HSync 信号：

```
always @(posedge TLcdClk or negedge RST)
begin
    if(!RST )
        HCount <= 0;
    else
        begin
        HCount <= HCount+1;
        if(HCount == HDCount -1)
            HSync <= #1 1'b1;
        else if(HCount == HTCount -1)
            begin
```

```
                HSync <= #1 1'b0;
                HCount <=#1 'b0;
                end
            end

end

生成 VSync 信号
always @(posedge HSync or negedge RST)
begin
    if(!RST)
        VCount <= 0;
    else
        begin
        VCount <= VCount + 1;
        if(VCount == VDCount -1)
            begin
            VSync <= #1 1'b1;
            end
        else if(VCount == VTCount - 1)
            begin
            VSync <= #1 1'b0;
            VCount <= #1 'b0;
            end
        end
end

生成 Enab 信号
always @( posedge TLcdClk or negedge RST    )
begin
if(!RST)
    EnCount <= 0;
else if (!HSYNC)
        EnCount <= 0;
     else
        begin
        EnCount <= EnCount+1;
        if(EnCount == EnDCount - 1)
            Enab <= #1 1'b1;
        else
            if(EnCount == EnTCount -1)
            begin
            Enab <= #1 1'b0;
            EnCount <= #1 'b0;
```

```
                end
            end
        end

    endmodule
```

②列同步程序设计：

列同步信号是控制每一屏幕的刷新起始位置，保持同步，防止图像扭曲。列同步周期的时序设计部分程序如下：

③鼠标与屏的交互

当鼠标点击屏幕时，将鼠标所点击的那一部分小区与点亮，部分程序如下：

```
assign MxReach = ((HCount>xMouse)&&(HCount<xMouse+MxNum+1))?1'b1:1'b0;
assign MyReach = ((VCount>yMouse)&&(VCount<yMouse+MyNum+1))?1'b1:1'b0;
assign MReach_t = (MxReach&&MyReach)?1'b1:1'b0;
dff M1(.CLK(CLK),.CLRN(RST),.PRN(1'b1),.D(MReach_t),.Q(MReach));
assign R = (MReach)?MData[4:0]:5'b11111;
assign G = (MReach)?MData[11:6]:6'b111111;
assign B = (MReach)?MData[16:12]:5'b11111;
```

5、参考文献

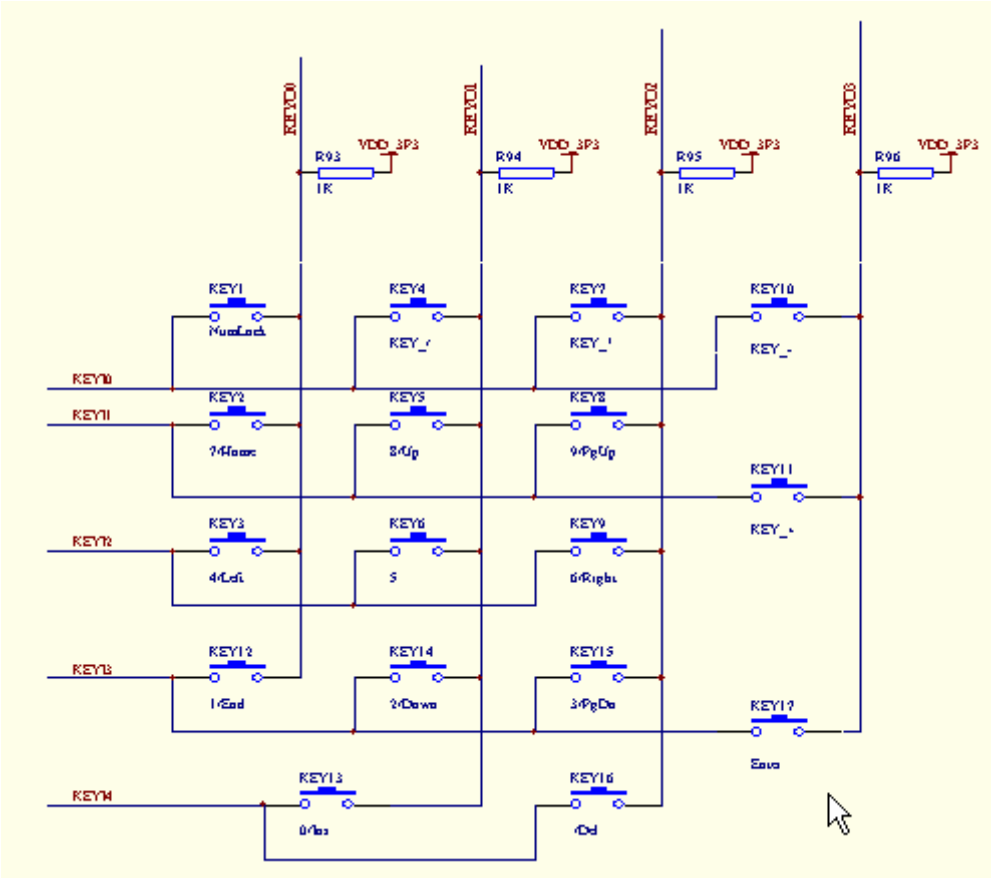
• 计算机综合课程设计　杨全胜　清华大学出版社

5. 标准键盘

1、矩阵键盘的原理

矩阵键盘又叫行列式键盘。用带 IO 口的线组成行列结构，按键设置在行列的交点上。例如用 4×4 的行列式结构可以构成 16 个键的键盘。这样，当按键数量平方增长时，I/O 口只是线性增长，这样就可以节省 I/O 口。

矩阵键盘原理图：



按键设置在行列线交叉点，行列线分别连接到按键开关的两端。列线通过上拉电阻接3.3V电压，即列线的输出被钳位到高电平状态。

判断键盘中有无按键按下式通过行线送入扫描线好然后从列线读取状态得到的。其方法是依次给行线送低电平，检查列线的输入。如果列线全是高电平，则代表低电平信号所在的行中无按键按下；如果列线有输入为低电平，则代表低电平信号所在的行和出现低电平的列的交点处有按键按下。

2、键盘接口设计：

① 扫描频率选择：

② 键盘扫描：

由键盘的工作原理可知，要正确地完成按键输入工作必须有按键扫描电路产生行扫描信号LINE，同时必须有按键译码电路从keydrv中和keyin中读出按键的键值。

键盘扫描电路是用于产生LINE[4]~LINE[0]信号，其变化顺序是：11110→11101→11011→10111→01111→11110…周而复始地扫描。其停留时间大概在10ms。更短的时间没有必要，因为人为按键的时间大概为10ms，不可能产生有更快的动作；另外，更短的停留时间还容易采集到抖动信号，会干扰判断。而太长的时间容易丢失某些较快的按键动作。

③ 键值译码：

通过KeyDecode 模块将每次得到的 LINE 和 COL 的值进行解码，输出键盘上的键值，把17个键对应如下的键值：

按键	键值(二进制)
0	00000
1	00001
2	00010
3	00011
4	00100
5	00101
6	00110
7	00111
8	01000
9	01001
＋	01010
－	01011
×	01100
/	01101
.	01110
Del	01111
NumLock	10000

3、键盘接口特色：

完成了锁键盘这一性质，当按下 NumLock 后键盘会上锁，再按就不会产生中断了，当再次按下 NumLock 后键盘会解锁继续正常工作。

下面是 NumLock 在解码模块中读取的代码 ：

```
output LOCK;
wire ABLE;
assign ABLE = !LOCK;

dffe DeL(.CLK(NEWKEY),.CLRN(!RST),.PRN(1'b1),.ENA(K[4]),.D(ABLE),.Q(LOCK));
```

由于 NumLock 是唯一以为解码后译码第四位为高的，所以以此位为使能端将 LOCK 值取反。再在外层模块中以 LOCK 值和按键值的与作为始能，代码如下图所示：

```
wire KEYABLE;
assign KEYABLE = NEWKEY && !LOCK;
dff dffe_Int0(.CLK(!CLK),.CLRN(!RST),.PRN(1'b1),.D(KEYABLE),.Q(KEY_INT));
```

PS2 鼠标

1、 PS2 接口简介：

物理上的PS/2 端口是两类连接器中的一种5 脚的DIN 或6 脚的mini-DIN 这两种连接器在电气特性上是十分类似的实际上两者只有一点不同那就是管脚的排列。接口如下图所示：

Male 公的	Female 母的	5-pin DIN (AT/XT):	5 脚 DIN(AT/XT)
		1 - Clock	1—时钟
		2 - Data	2—数据
		3 - Not Implemented	3—未实现，保留
		4 - Ground	4—电源地
(Plug) 插头	(Socket) 插座	5 - +5v	5—电源+5V

Male 公的	Female 母的	6-pin Mini-DIN (PS/2):	6 脚 Mini-DIN(PS/2)
		1 - Data	1—数据
		2 - Not Implemented	2—未实现，保留
		3 - Ground	3—电源地
		4 - +5v	4—电源+5V
(Plug) 插头	(Socket) 插座	5 - Clock	5—时钟
		6 - Not Implemented	6—未实现，保留

PS/2 鼠标和键盘履行一种双向同步串行协议，换句话说每次数据线上发送一位数据并且每在时钟线上发一个脉冲就被读入。键盘/鼠标可以发送数据到主机而主机也可以发送数据到设备，但主机总是在总线上有优先权，它可以在任何时候抑制来自于键盘/鼠标的通讯只要把时钟拉低即可。

从键盘/鼠标发送到主机的数据在时钟信号的下降沿当时钟从高变到低的时候被读取。

从主机发送到键盘/鼠标的数据在上升沿当时钟从低变到高的时候被读取。

不管通讯的方向怎样键盘/鼠标总是产生时钟信号，如果主机要发送数据它必须首先告诉设备开始产生时钟信号。大多数设备工作在10、20kHz 如果你要制作一个PS/2 设备很多资料推荐把频率控制在15kHz 左右这就意味着时钟应该是高40 微秒和低40 微秒。

PS2 端口把数据按照字节传送。字节的格式如下图所示：

1 个起始位总是为0
8 个数据位低位在前
1 个校验位奇校验
1 个停止位总是为1
1 个应答位仅在主机对设备的通讯中

2、 PS2 鼠标工作原理：

• 标准PS2 鼠标包由三个字节组成（带滚轮的鼠标多一个字节的Z Movement组成）三个字节如图所示：

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Y overflow	X overflow	Y sign bit	X sign bit	Always 1	Middle Btn	Right Btn	Left Btn
Byte 2	X Movement							
Byte 3	Y Movement							

其中第一个字节中的Bit0、Bit1、Bit2分别表示左、右、中键状态。0表示放下，1表示按下；Bit4、Bit5 分别表示朝 X、Y轴的方向；Bit6、Bit7表示X、Y运动的溢出。

第二、三个字节为以二进制补码表示的X、Y轴的移动向量值。

• 标准PS2 鼠标工作模式：

1. Reset模式：

   鼠标在上电或收到Reset 0xFF 命令后进入Reset 模式

2. Stream模式：

   这是缺省模式在Reset 执行完成后也是多数软件使用鼠标的模式如果主机先前吧鼠标设置到了Remote模式： 它可以发送Set Stream Mode 0xEA 命令给鼠标让鼠标重新进入Stream模式。

3. Remote模式：

   在某些情况下Remote 模式很有用可以通过发送Set Remote Mode 0xF0 命令进入。

4. Wrap模式：

   除了为测试鼠标和它的主机之间的连接外这个模式不是特别地有用。Wrap 模式可以通过发送Set Wrap Mode 0xEE 命令给鼠标来进入。要退出Wrap 模式主机必须发布Reset 0xFF命令或Reset Wrap Mode 0xEC 命令。如果Reset 0xFF 命令收到了鼠标将进入Reset 模式。如果收到的是Reset Wrap Mode 0xEC 命令鼠标将进入Wrap 模式前的那个模式。

• 标准PS2 鼠标的命令集（发送给鼠标的命令）：

命令	作用
0xFF（Reset）	鼠标用应答0xFA 回应这条命令并进入Reset 模式
0xFE (Resend)	只要从鼠标收到无效数据主机就发送这条命令鼠标的回应是重新发送它最后发给主机的数据包
0xF6（Set Defaults）	鼠标用应答0xFA 来回应然后载入如下的值采样率＝100、 分辨率＝4 个值/毫米，比例＝1: 1， 禁止数据报告接着鼠标清空它所有的位移计数器并进入stream 模式。
0xF5（Disable Data Reporting）	鼠标用“应答”（0xFA） 回应命令，然后禁止数据报告并复位它的位移计数器。这仅对Stream 模式下的数据报告有效并且它不能禁止采样。禁止的stream 模式功能与remote 模式相同。
0xF4（Enable Data Reporting）	鼠标用“应答”（0xFA） 回应命令，然后使能数据报告并复位它的位移计数器。这条命令可以对在Remote 模式（或Stream 模式）下的鼠标发布，但只对Stream 模式下的数据报告有效。
0xF3（ Set Sample Rate）	鼠标用“应答”（0xFA） 回应命令，然后从主机读入一个或更多字节。鼠标保留这个字节作为新的采样速率。在收到采样速率后，鼠标再次用“应答”（0xFA） 回应并复位它的位移计数器。有效的采样速率是10、

	20、40、60、80、100 和200 采样点/秒。
0xF2（Get Device ID）	鼠标用“应答”（0xFA） 回应命令后面跟着它的设备ID（对标准PS/2 鼠标来说是0x00）。鼠标同样会复位它的位移计数器。
0xF0 (Set Remote Mode)	鼠标用“应答”（0xFA） 回应，然后复位它的位移计数器并进入Remote 模式。
0xEE（Set Wrap Mode）	鼠标用“应答”（0xFA）回应，然后复位它的位移计数器并进入wrap 模式
0xEC（Reset Wrap Mode）	鼠标用“应答”（0xFA） 回应，然后复位它的位移计数器并进入wrap 模式之前的那个模式（stream 模式或remote 模式）
0xEB（ Read Data）	鼠标用“应答”（0xFA） 回应，然后发送位移数据包。这是在remote 模式中读数据的位移方法。在数据包成功地被发送后，鼠标将复位它的位移计数器。
0xEA（ Set Stream Mode）	鼠标用“应答”（0xFA）回应，然后复位它的位移计数器并进入stream 模式
0xE9 (Status Request)	鼠标用“应答”（0xFA）回应，然后发送3 个字节的状态包（在此不列举了）
0xE8 (Set Resolution)	鼠标用“应答”（0xFA）回应，然后从主机读取一个字节，并再次用“应答”（0xFA）回应，然后复位它的位移计数器。从主机读入的字节决定了分辨率。
0xE7 (Set Scaling 2:1)	鼠标用“应答”（0xFA）回应，然后使能2 :1 比例
0xE6 (Set Scaling 1:1)	鼠标用“应答”（0xFA）回应，然后使能1 :1 比例

• 标准PS2 鼠标的初始化：

正常情况下，PS2 鼠标仅在计算机刚启动的时候被检测和初始化。因此鼠标不能热插拔。

初始化时需先将时钟线拉低并保持400us，而后发送初始化命令0XF4,发送完毕后将数据线拉高等待鼠标返回应答信号。若当时钟信号下降沿来临时数据线未表示此次握手失败，否则握手成功。

3、 实验设备介绍：

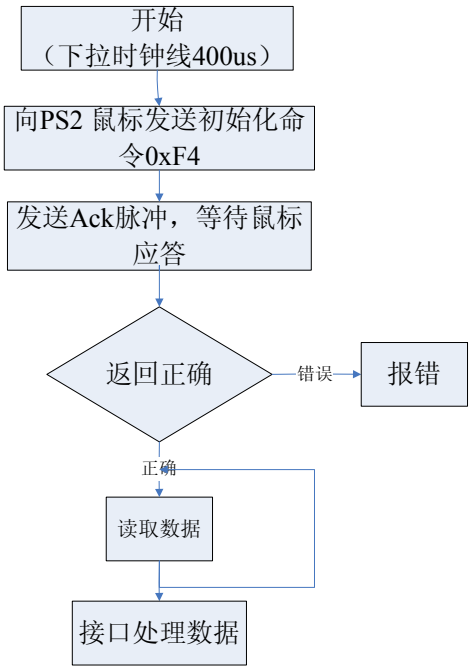
FPGA 芯片：ALTERA，CycloneⅡ EP2C35F672C8N。

PS2 接口： 六脚。

PS2 鼠标型号： 惠普，M-SBF96

4、PS2 鼠标接口设计：

流程图如图所示：



5、PS2 鼠标代码：

（1）自动机部分

```
always @(m2_state or fall or watchdog_timer_done)
 begin
 gather_start <= 1'b0;
 case(m2_state)
      m2_reset:
           begin
           m2_next_state <= m2_hold_clk_l;
           end
      m2_wait:
           begin
           ps2_clk_hi_z <= 1'b1;
           output_strobe <= 1'b0;
           if(fall == 1'b1)
                begin
                m2_next_state <= m2_gather;
                gather_start <= 1'b1;
                end
           else
                begin
                m2_next_state <= m2_wait;
                end
           end
      m2_gather:
           begin
           ps2_data_hi_z <= 1'b1;
```

```
            if(bitcount == TOTAL_BITS)    //watchdog delete
                begin
                m2_next_state <= m2_verify;
                end
            else
                begin
                m2_next_state <= m2_gather;
                end
            end
        m2_verify:
            begin
            ps2_data_hi_z <= 1'b1;
            m2_next_state <= m2_use;
            end
        m2_use:
            begin
            ps2_data_hi_z <= 1'b1;
            output_strobe <= 1'b1;
            m2_next_state <= m2_wait;
            end
        m2_hold_clk_l:
            begin
            ps2_clk_hi_z <= 1'b0;
            ps2_clk_reg <= 1'b0;
            if(watchdog_timer_done == 1'b1)
                begin
                m2_next_state <= m2_data_low_1;
                end
            else
                begin
                m2_next_state <= m2_hold_clk_l;
                end
            end
        m2_data_low_1:
            begin
            ps2_clk_hi_z <= 1'b1;
            ps2_data_hi_z <= 1'b0;
            ps2_data_reg <= 1'b0;
            if(fall == 1'b1 && (bitcount == 2) )
                begin
                m2_next_state <= m2_data_high_1;
                end
            else
                begin
```

```
                m2_next_state <= m2_data_low_1;
                end
        end
    m2_data_high_1:
        begin
        ps2_data_hi_z <= 1'b0;
        ps2_data_reg <= 1'b1;
        if(fall == 1'b1 && (bitcount == 3))
            begin
            m2_next_state <= m2_data_low_2;
            end
        else
            begin
            m2_next_state <= m2_data_high_1;
            end
        end
    m2_data_low_2:
        begin
        ps2_data_hi_z <= 1'b0;
        ps2_data_reg <= 1'b0;
        if(fall == 1'b1 && (bitcount == 4))
            begin
            m2_next_state <= m2_data_high_2;
            end
        else
            begin
            m2_next_state <= m2_data_low_2;
            end
        end
    m2_data_high_2:
        begin
        ps2_data_hi_z <= 1'b0;
        ps2_data_reg <= 1'b1;
        if(fall == 1'b1 && (bitcount == 8))
            begin
            m2_next_state <= m2_data_low_3;
            end
        else
            begin
            m2_next_state <= m2_data_high_2;
            end
        end
    m2_data_low_3:
        begin
```

```
            ps2_data_hi_z <= 1'b0;
            ps2_data_reg <= 1'b0;
            if(fall == 1'b1)
                begin
                m2_next_state <= m2_data_high_3;
                end
            else
                begin
                m2_next_state <= m2_data_low_3;
                end
            end
        m2_data_high_3:
            begin
            ps2_data_hi_z <= 1'b1;
            if(fall == 1'b1 && (PS2_DATA==1'b1))
                begin
                m2_next_state <= m2_error_no_ack;
                end
            else if(fall == 1'b1 &&(PS2_DATA == 1'b0))
                begin
                m2_next_state <= m2_await_response;
                end
            else
                begin
                m2_next_state <= m2_data_high_3;
                end
            end
        m2_error_no_ack:
            begin
            error_no_ack <= 1'b1;
            m2_next_state <= m2_error_no_ack;
            end
        m2_await_response:
            begin
            ps2_data_hi_z <= 1'b1;
            m2_next_state <= m2_verify;
            end
        default:
            begin
            m2_next_state <= m2_wait;
            end
    endcase
    end
```

(2) 判断时钟下降沿：

```
        always @(posedge CLK or negedge RST)
    begin
        if(!RST)
            begin
            fallsig <= 3'b000;
            end
        else
            begin
            fallsig[2] = fallsig[1];
            fallsig[1] = fallsig[0];
            fallsig[0] = PS2_CLK;
            end
    end
assign fall = (fallsig == 3'b110)?1'b1:  1'b0;
```

(3) 读取屏幕各值：

```
assign MOUSEX = MOUSEX_T[11:2];
assign MOUSEY = MOUSEY_T[11:2];
always @(posedge output_strobe or negedge RST)
    begin
    if(!RST)
        begin
        LBUTTON <= 1'b0;
        RBUTTON <= 1'b0;
        MBUTTON <= 1'b0;
        MOUSEX_T <= 12'h3ff;//400
        MOUSEY_T <= 12'h3ff;//300
        end
    else
        begin
        LBUTTON <= q[1];
        RBUTTON <= q[2];
        MBUTTON <= q[3];
        //100+640     100
        if((MOUSEX>872 && q[5]==1'b0 )||(MOUSEX<232 && q[5]==1'b1))
            begin
            MOUSEX_T <= MOUSEX_T;
            end
        else
            begin
            if(q[5])
                begin
                MOUSEX_T <= MOUSEX_T-4;
```

```
                end
            else
                begin
                MOUSEX_T <= MOUSEX_T+4;
                end
            end
        if((MOUSEY>525 && q[6]==1'b0)||(MOUSEY<59 && q[6]==1'b1))
            begin
            MOUSEY_T <= MOUSEY_T;
            end
        else
            begin
            if(q[6])
                begin
                MOUSEY_T <= MOUSEY_T-4;
                end
            else
                begin
                MOUSEY_T <= MOUSEY_T+4;
                end
            end
        end
    end
```

6、参考文献：

- 《计算机综合课程设计》　杨全胜编著　清华大学出版社
- 《基于 FPGA 的工程设计与应用》王彦主编　西安电子科技大学出版社
- 《PS/2 技术参考》　著 Adam Chapweske 译 Roy Show
- 《FPGA 嵌入式项目开发实战》　刘福奇编著　电子工业出版社

编译器：

㈠ MiniSys 系统的优化编译器 – 中级中间语言上的初步尝试

Compilers are the Queen of Computing Science and Technology.

David Kuck

现代体系结构的优化编译器是最为复杂的软件系统之一。编译程序的性能很大程度上决定了计算机软件系统的性能。对于硬件资源本来就很稀缺的Minisys嵌入式系统，能拥有一个较高性能的编译器是非常重要的。这也是我们决定为原始编译器提供优化层的原因。

MiniC的编译器主要的问题包括不支持作用域规则，不支持显式指针。由于对变量填充符号表细节不甚清楚，我们把优化局限于过程内，并且不对指针做特殊处理。

参考[1],我们决定提供如下几种重要的优化：

1. 常数折叠；
2. 代数化简和重结合；

3. 全局值编号；
4. 稀有条件常数传播；
5. 由公共子表达式删除和循环不变代码外提组成一对优化；
6. 强度削弱；
7. 归纳变量删除和线性函数测试替换；
8. 死代码删除；
9. 不可达到代码删除（控制流优化）；
10. 图着色寄存器分配；

其中，全局值编号(Global Value Numbering)由于时间问题没有完成。图着色寄存器(Graph Coloring Register Allocation)其实是上述所有优化中最重要的，对于生成的汇编程序的效率有非常大的影响。但由于时间有限，我们无法仔细分析旧版编译器的符号表管理部分，对原编译器如何进行存储分配，过程间如何传参数并不清楚。另外，寄存器分配本身代码量相当大。基于这两个原因，我们最后没有实现它。[1]为分配器设计的顶层结构是错误的，如果要实现这个算法，推荐参考文献[2].我们对循环优化也作了相当程度的简化。主要原因是无法知道原编译器中对While循环的具体翻译方式。特别的，我们认为所有要处理的循环都是规则循环。
我们首先介绍代码优化的主要分析技术：数据流分析和控制流分析.
1. Control-Flow Analysis

控制流分析用来从中间代码中提取特征，生成程序的控制流图。由此确定基本块间的必经（直接必经）关系，识别出循环。数据流分析提供了构造程序的SSA形式的基本信息。数据流分析的基础是节点间必经关系和迭代法。
2. Data – Flow Analysis
数据流分析的目的在于提供程序如何操纵数据的全局信息。在优化中数据流分析被用来做：
A. Define – Reach Analysis
即确定一个定值能否到达程序中的某个点。这是构造Du, Ud链以及确定程序的SSA形式的基础，在优化中具有根本性的重要作用。
B. Live-Variable Analysis
确定从某个变量的定值位置有没有一条路径通往程序结束点。这是Graph Coloring Register Allocation的基础之一。
数据流分析的数学基础是格函数的不动点迭代。
下面介绍优化中最重要两个数据结构：Du/Ud链和过程的SSA形式(Static Single Assignment Form)。
Du链(定值-使用链)给出了某变量的一个定值所能够到达的所有使用的位置。Ud链则相反，对于变量的每个使用，给出所有可能提供该值的定义的位置。
过程的SSA形式是一种较新形式的中间代码。在SSA形式中，每个变量都只有一个定值。这样，运算中的值就可以与其存储的位置分开，我们不需要关心“某定值会不会到达某使用”这类问题。SSA Form尽可能暴露了变量的所有使用，进而可使一些优化更为有效的进行，如稀有条件常数传播和全局值编号。
我们设计的优化顺序为：
1. 不可达代码删除
2. 代数化简
3. 稀有条件常数传播

4. 常数表达式计算

5. 循环不变代码外提

6. 局部公共子表达式删除

7. 全局公共字表达式删除

8. 循环本身的优化，包括强度削弱，归纳变量删除和线性函数测试替换

9. 代数化简

10. 稀有条件常数传播

11. 常数表达式计算

12. 死代码删除

这一顺序保证了优化整体的效果，并显著简化了把SSA代码复原成普通中间代码的过程。下面逐一介绍这些优化，因为他们大都涉及复杂的算法，其原理无法简单地讲明白，所以这里只简单介绍一下他们的效果。

1. 不可达代码删除通过分析控制流图，发现那些不可能到达的基本块，并将其删除。

2. 代数化简分析每条指令，并进行可能的化简。如:

- i+ 0 = 0 + i = i-0 = i
- i* 1 = 1*i = i/1 = i
- 移位偏移为0转化为MOV, 超过字长限制直接置0
- 加减和移位操作代替乘法

3. 稀有条件常数传播观察常量定值：

x <- c

通过分析控制流，确定在某个用到x的指令处，如果没有其他不同的对x的定值的干扰，就把使用的x替换成常数c。

稀有条件常数传播基于SSA Form和常数传播格(Const Propagation Lattice),其指导思想是程序的符号执行。它为后面常量表达式计算带来更多的机会。

4. 常量表达式计算

如果当前指令是一个算术指令，并且操作数都是常数，则可以直接计算其值。获得的结果在本块内作有限的传播。

5. 循环不变代码外提

其作用很显然，识别出循环代码中与循环变量无关的部分并外提。这可以显著提高程序效率。一般地，循环占据程序执行时间相当大的比例，所以循环优化力度大些是可取的。

6. 局部和全局公共子表达式删除

这一优化的作用也很显然，尤其是子表达式计算代价很大时。但是如果子表达式很简单，这一优化并不一定很值得。

7. 强度削弱

强度削弱是循环中的一种强有力的优化，把依赖于循环变量的其他变量的计算尽可能简化。如循环变量i的初值为 0增量方式是

i= i +2

而循环中依赖于i的变量每次都被定值为

j = i * 4

则强度削弱通过收集归纳变量进行处理：

在循环前置块中分配 tj, 置初值为0，j的定值语句变为:

j = tj

在i = i +2后添加 tj = tj + 8;

由此j的计算的强度被大大削弱

8. 归纳变量删除和线性函数测试替换

9. 死代码删除

死代码删除也是一种作用较强的优化。如果某指令对程序的输出无贡献，就会被整个删除。这往往可以配合其他优化删除大量无用代码。

作为优化的例子，我们给出测试程序，一个二重循环：

MOV i 0

MOV j 0

MOV a 2

LABEL L4

SLT B1 i 5

BEQ L1 B1 0

MLT t0 2 i

ADD m t0 a

LABEL L3

SLT B2 j 5

BNE L2 B2 1

MLT t1 4 j

ADD c t1 3

MLT t2 c 3

SUB d t2 4

ADD e i d

ADD j j 1

GOTO L3

LABEL L2

ADD f m e

ADD i 2 i

GOTO L4

LABEL L1

RET f

经过优化，结果的中间表示为：

Entry

Block 1: MOV i 0

MOV j 0

LABEL L4

SLT B1 i 5

BEQ L1 B1 0

Block 3:

Block 4:LABEL L3

SLT B2 j 5

BNE L2 B2 1

Block 5:

Block 6:LABEL L2

ADD f _temp_25 e

ADD i 2 i

ADD _temp_25 _temp_25 _temp_26

Block 7:LABEL L1

RET f

Exit

参考文献：

[1] *Advanced Compiler Design & Implementation, 2/e*, Steven S. Muchnick

[2] *Engineering a Compiler*, Keith D. Cooper & Linda Torczon

[3] *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*, Ron Cytron et al., *ACM TOPLAS*, VOL 13, NO. 4, Oct 1991, P451-490

㈡编译器特色

编译程序的性能很大程度上决定了计算机软件系统的性能。然而编译优化本身是非常困难的，我们在中级中间语言级别为Minisys
系统的编译器优化做了初步尝试。由于各方面的限制，导致优化无法深入到机器底层，所完成的可以说只是一个演示程序。我们比较系统的理解了如下几种优化：

1. 常数折叠
2. 代数化简和重结合
3. 局部值编号和全局值编号
4. 稀有条件常数传播
5. 公共子表达式删除和循环不变代码外提
6. 强度削弱
7. 归纳变量删除和线性函数测试替换
8. 死代码删除
9. 不可达代码删除
10. 图着色寄存器分配

最后完成了3,10之外的8个部分。我们的工作的主要特色是通过实现这些主要的优化算法，演示了编译器后端对前段生成的原始代码(Yacc生成的原始编译器所能生成的代码)的显著优化作用。资料表明，优化器能数倍地提高程序运行效率。编译器优化的两个支撑技术是数据流分析和控制流分析。我们实现的这8个优化也不例外，它们大多涉及到复杂的算法和深刻的数学背景。通过实际动手，增强了我们理解算法，实现算法的能力。我们的数学功底也得到了加强，特别是有关格的一些知识。

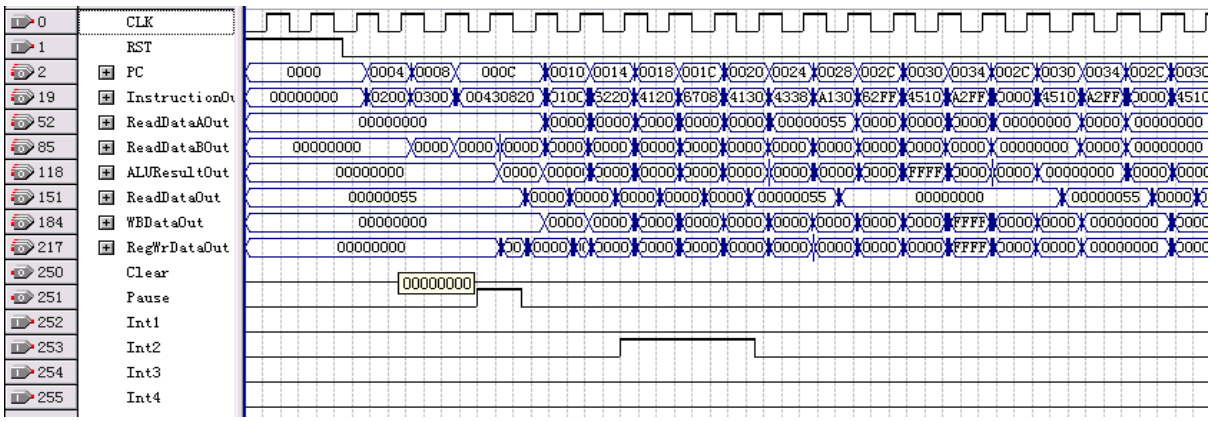
总结：

我们经过长时间的学习，对主要的中间层机器无关优化有了比较深刻的理解。这近 4000 行的代码，没有一处是在简单枚举大量可能情形的。整个程序，就是一个接一个复杂算法整合而来的。实现这个程序，我们的逻辑和算法思维得到了很大锻炼。由于是两个人合作，也碰到了一些不可避免的代码风格问题。整合代码是非常繁琐的工作，对我们 debug 的能力也是极大的考验。然而，为实现一个实际可用，效果显著的 MiniSys 系统上的编译器，仍然有不少工作要做。包括重新设计更为合理的中间语言，搞清运行时内存分配并支持静态作用域规则，以及一些更高级的优化方法。或许直接移植 MIPS 平台上的 lcc 是一个不错的选择。

本组设计主要测试结果（.vwf）

1. CPU

CPU 测试截图：

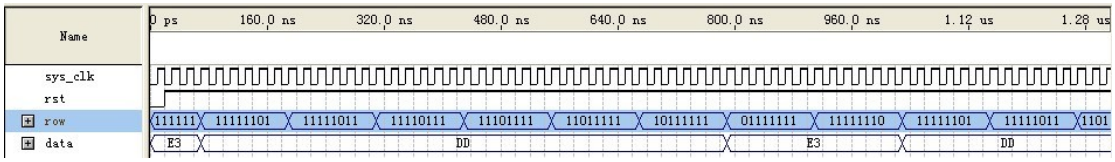


CPU 测试结果：没用流水的主频大约是 15M。用了流水的主频大约是 30M

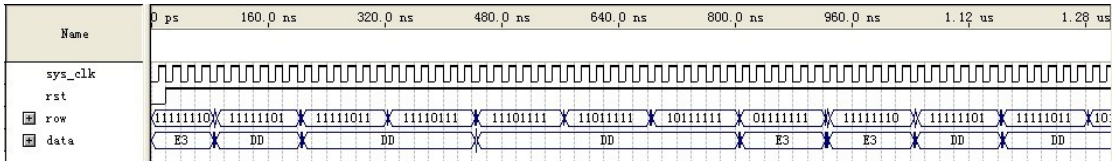
2. 8×8 点阵

8×8 点阵连续显示数字 0---9，字母 A---Z

功能仿真图

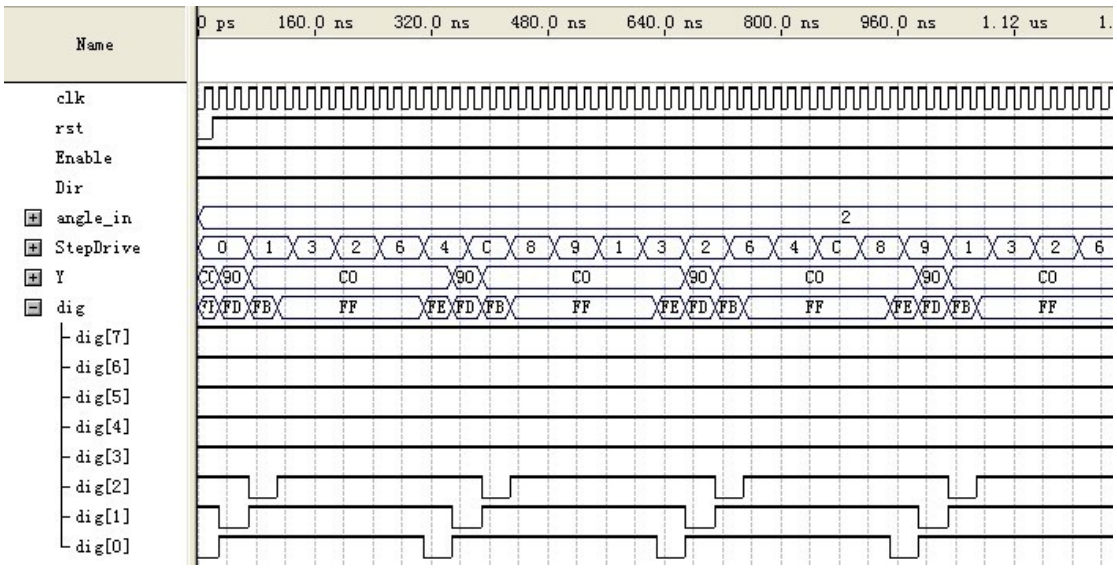


时序仿真图

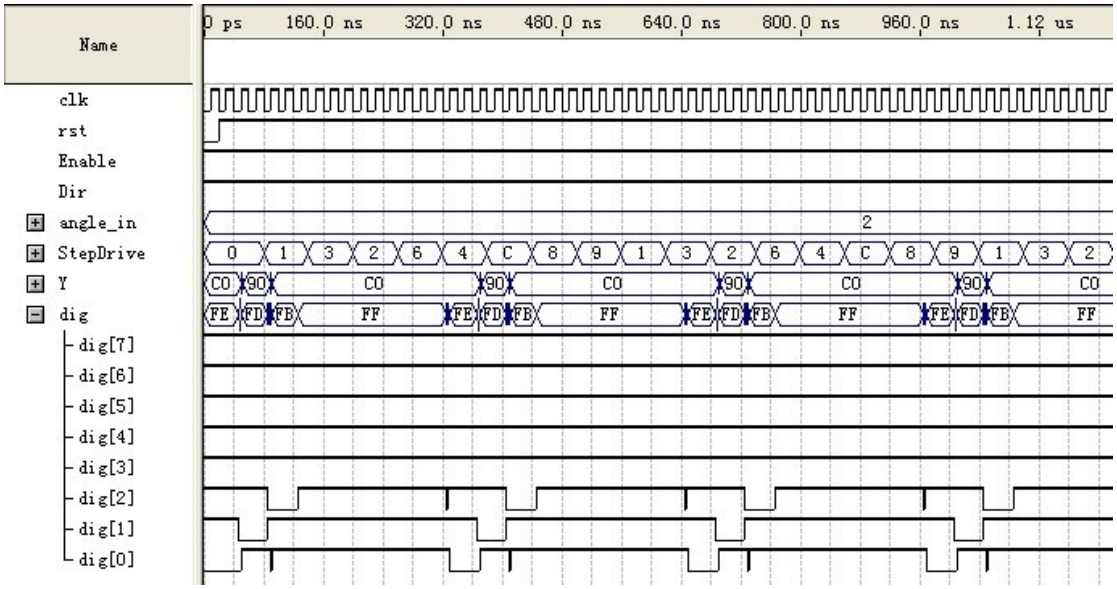


3. 步进电机

时序仿真图



功能仿真图

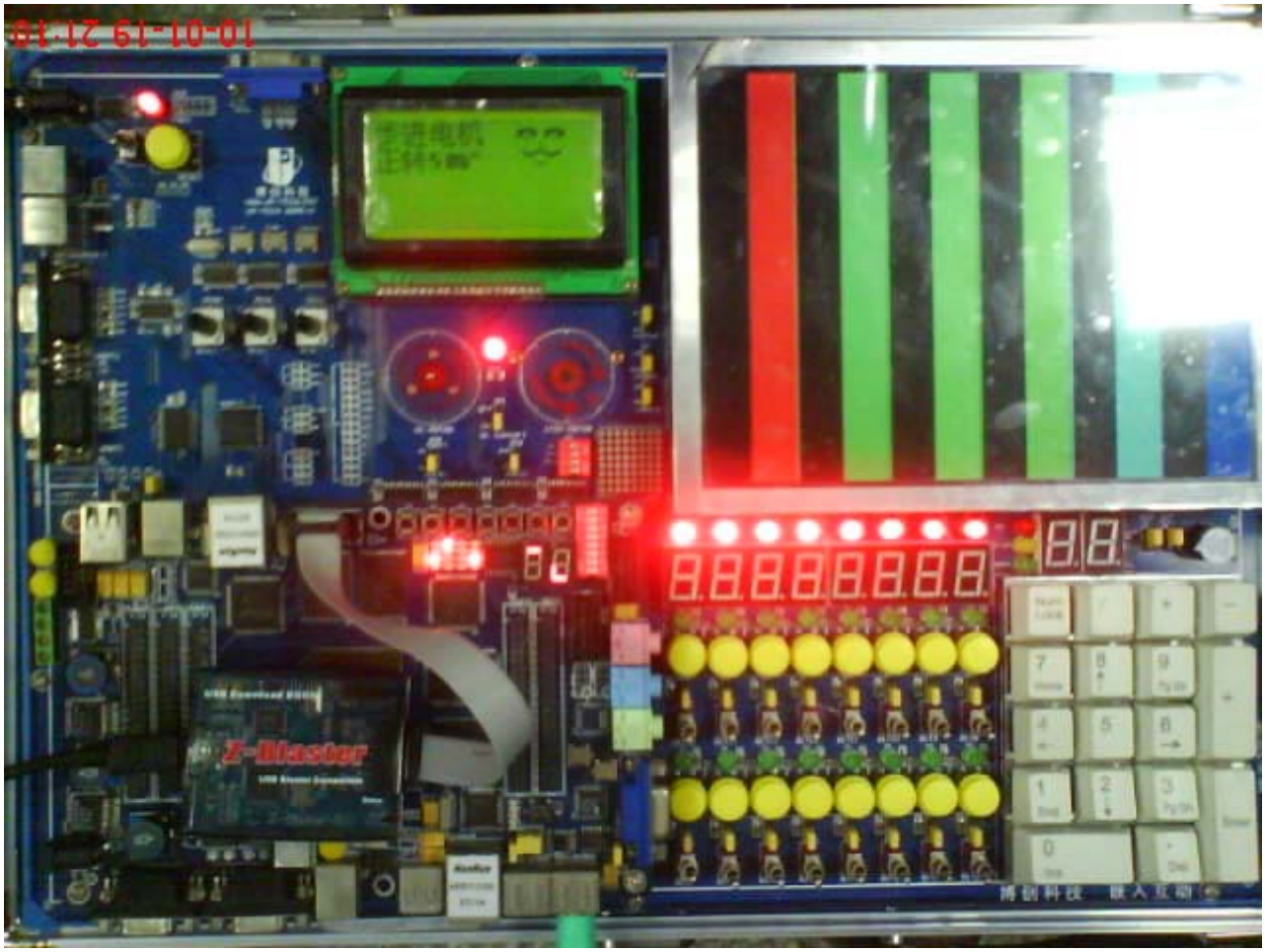


4. 点阵液晶显示器（黑白屏）

附照片：

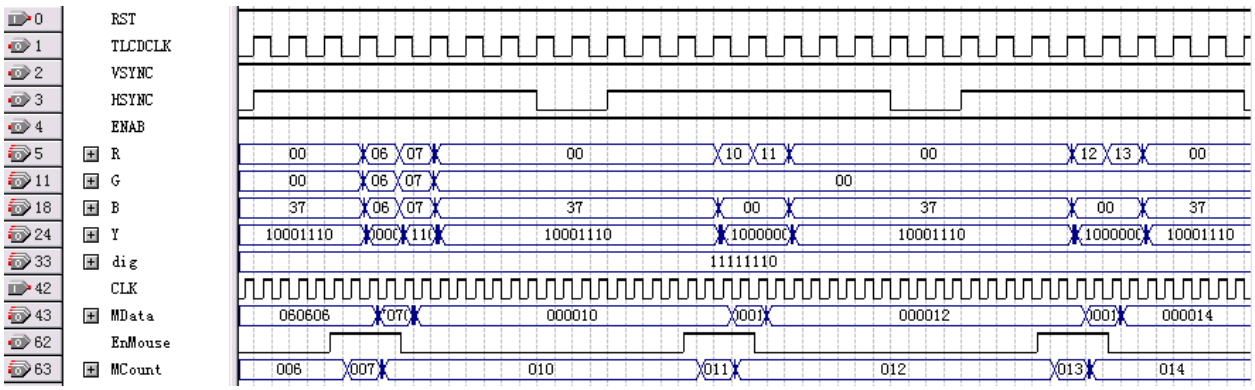
本组设计的性能分析（资源使用情况、主频、功耗数据和自我分析）

整体照片：黑白屏 + 步进电机 + 点阵

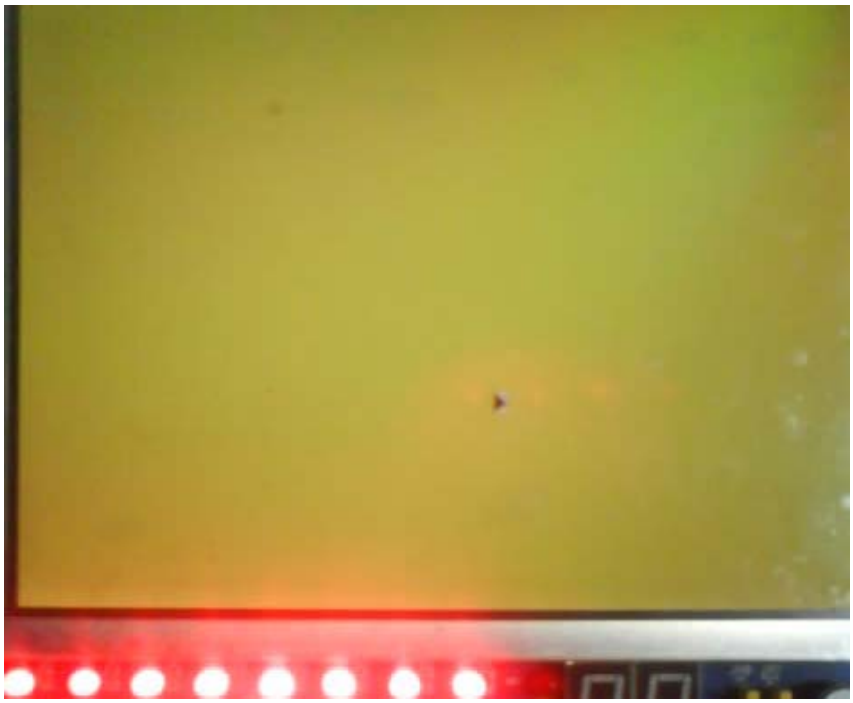


5、TFT-LCD 彩色大屏

时序仿真：

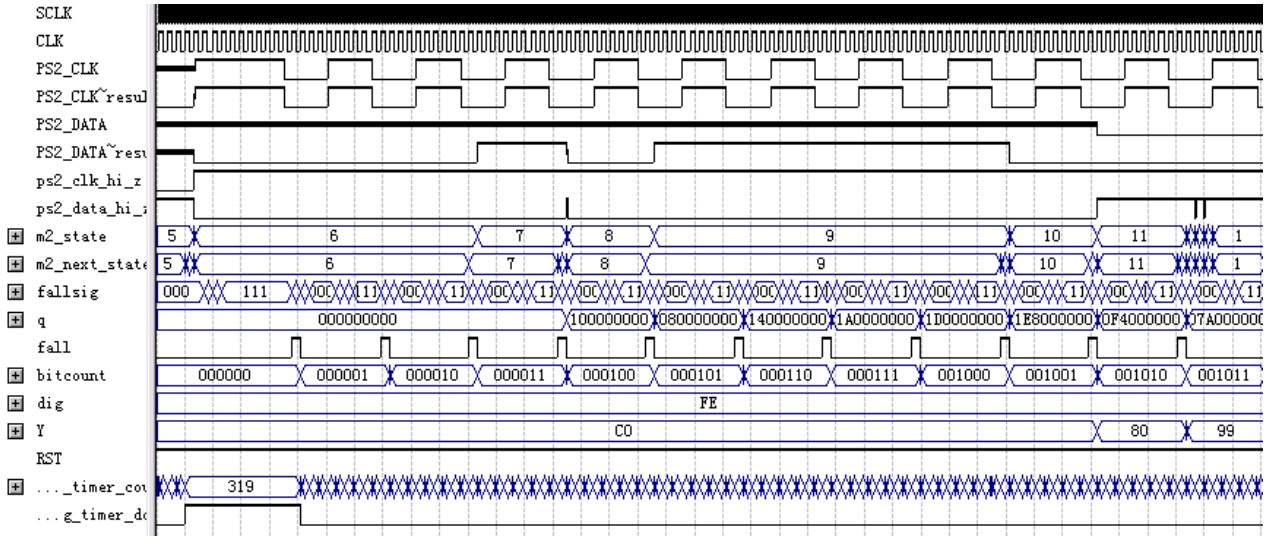


成果图片：

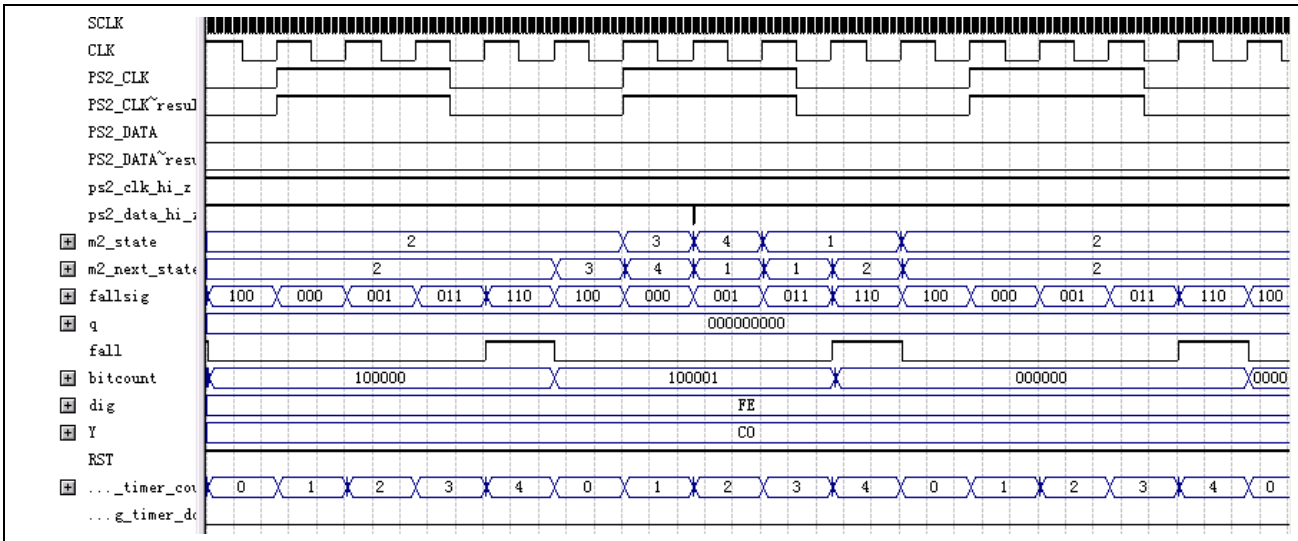


6、PS2 鼠标:

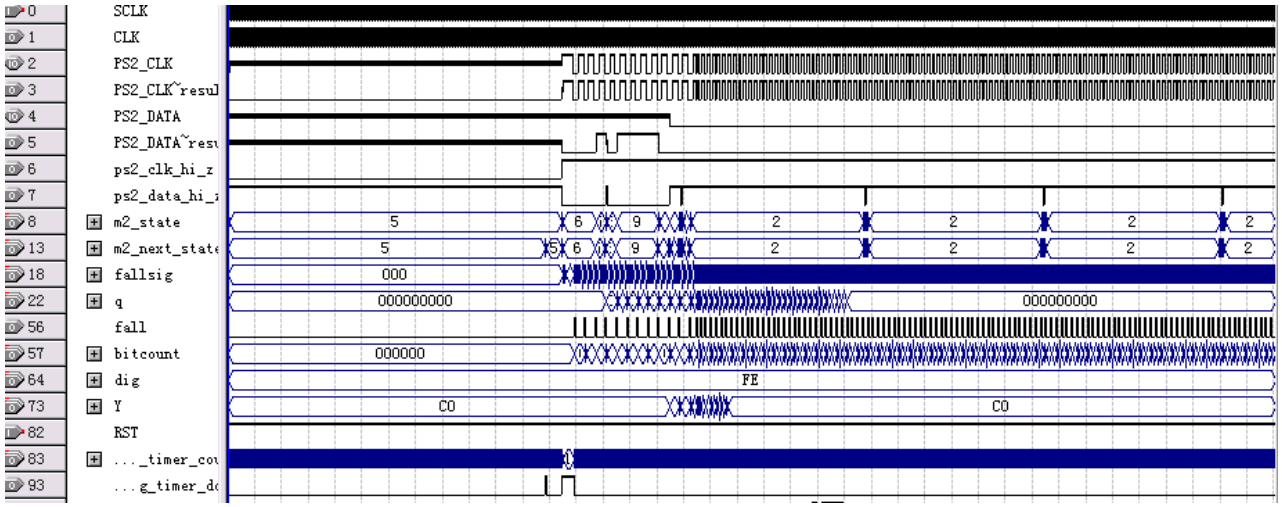
部分时序仿真图 1（传送开始指令 0xF4 时）:



部分时序仿真图 2（Stream 状态时）:

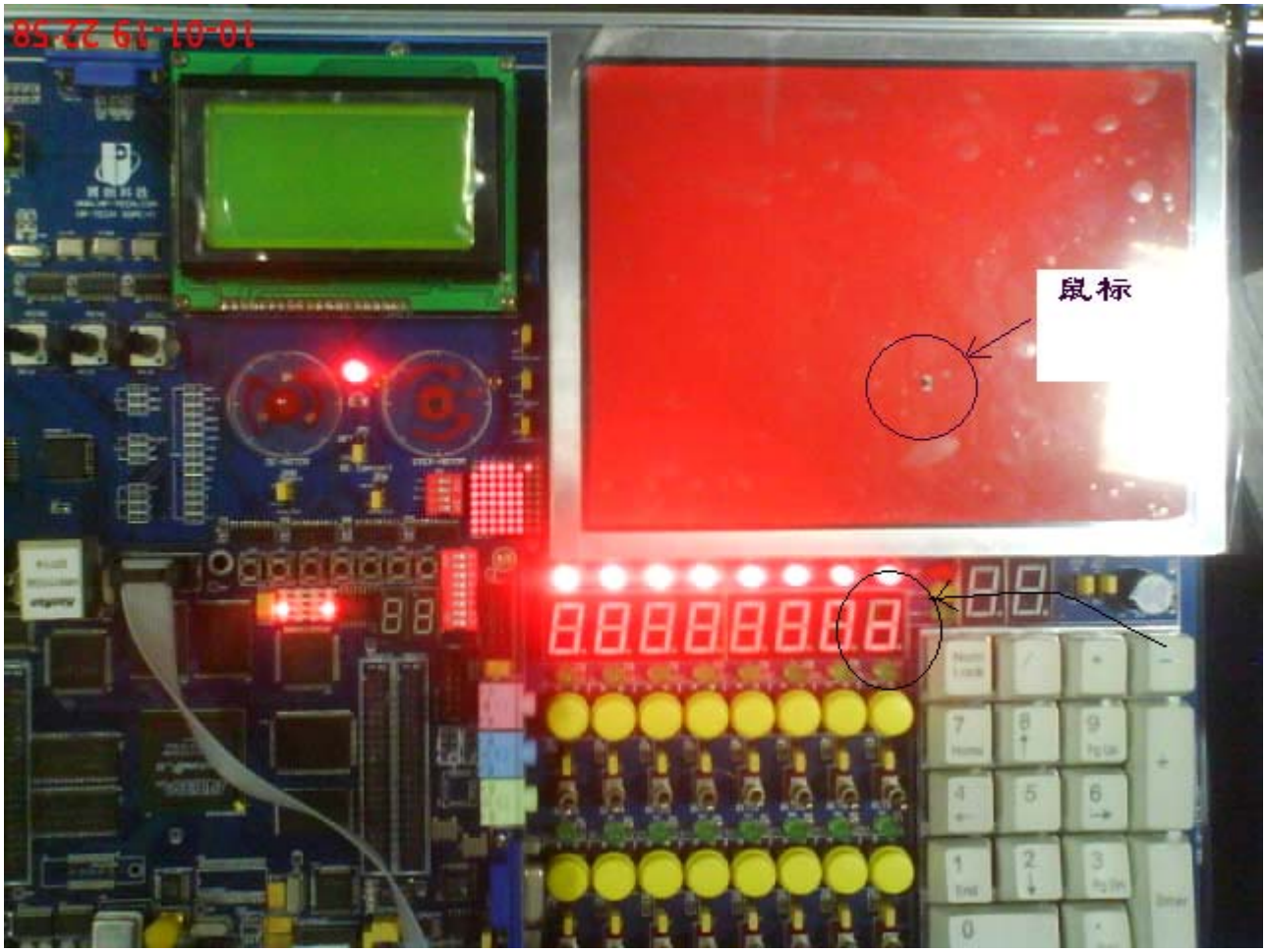


整体时序仿真图:



7、TFT－LCD 彩色大屏 + PS2 鼠标 + LED 灯 + 键盘 整体:

照片:


鼠标


鼠标
10-01-19 22:58

课程设计总结（包括设计的总结和还需改进的内容）

总结

通过组里同学的共同努力，计算机综合课程设计终于完成了。虽然不是十分完美，但其中凝聚的心血也让大家有收获的喜悦。合作的过程就是发现问题、解决问题的不断循环。这次实验我们收获很多。首先，动手能力得到了很大的提高。在这次之前，我们接触的一般是软件实验，有少部分硬件实验，而且都是分开的。但这次实验要求我们软硬结合，，遇到很多和软件实验完全不同的问题，如时序。理论设计的正确并不代表实验的成功，因为调试过程中可能会出现各种意想不到的问题，稍有偏差，整个调试结果可能完全错误，这让我们明白了细节决定成败的道理。此外，牵一发而动全身，对实验稍做修改，可能测试结果会完全不一样，这也导致实验进度慢。其次，自学能力有所提高。在实验过程中，接触到很多新知识，除了老师的指导，同学间的讨论，自己慢慢摸索也是必须的，这样既可以掌握知识，又能够锻炼自己的学习能力，一举两得。再者，加强了团队合作的意识。刚开始实验时，觉得实验很复杂，毫无思路。后来，经过大家多次的讨论，终于理清思路，开始实验；在实验各部分合起来的时候，更需要多个人一起完成，往往一个人想很久想不通的问题，在大家你一言我一语的讨论过程中就迎刃而解了。

通过这次实验，我们也发现了自身存在的一些缺点和不足。如时间安排不恰当，做事拖拉，总是把事情拖到最后才做，大大影响了做事效率。此外，分工不够明确，有时会出现重复劳动的现象。在今后的学习过程中，我们会吸取教训，改正不足，争取做得更好！

还需修改的内容

这些接口其实每个都可以扩展应用很多，如键盘和 LED 灯可以通过 BIOS 写成简单的计算器，比如大小屏幕甚至可以用来放动画，而点阵甚至可以用来看 CPU 的利用率……由于时间关系这些设想都没有实现。而鼠标只是简单的做成了一个自动机的模样，如果拓展开来可以将每次的收发信息写成一个模块进行调用，这样就可以适应 PS2 鼠标的很多应用，可以调节鼠标的灵活性并加强纠错能力。

此致

敬礼

小组成员签名：

验 收 报 告

主频	MHz	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		接口电路设计	☐通过 ☐未通过	
汇编器设计	☐通过 ☐未通过		合 成	☐通过 ☐未通过	
中断	☐有 ☐无 ☐未通过		BIOS	☐有 ☐无 ☐未通过	
应用软件	☐有 ☐无 ☐未通过		验收答辩	☐通过 ☐未通过	
加分项目					
存在问题					
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: _____