基于 EDA 平台计算机组成原理实践教程

Practice Tutorial of Computer Organization Principle
Based on EDA Platform

易小琳　　朱文军　　鲁鹏程 编

北京航空航天大学出版社

第 3 章 基于 Verilog HDL 的主机系统设计

Verilog HDL 是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述，并可在相同描述中显式地进行时序建模。

Verilog HDL 语言具有下述描述能力：设计的行为特性、设计的数据流特性、设计的结构组成以及包含响应监控和设计验证方面的时延和波形产生机制。所有这些都使用同一种建模语言。此外，Verilog HDL 语言提供了编程语言接口，通过该接口可以在模拟、验证期间从设计外部访问设计，包括模拟的具体控制和运行。

Verilog HDL 语言不仅定义了语法，而且对每个语法结构都定义了清晰的模拟、仿真语义。因此，用这种语言编写的模型能够使用 Verilog 仿真器进行验证。语言从 C 编程语言中继承了多种操作符和结构。Verilog HDL 提供了扩展的建模能力，完整的硬件描述语言足以对从最复杂的芯片到完整的电子系统进行描述。

下面列出的是 Verilog 硬件描述语言的主要能力：

- 基本逻辑门，例如 and、or 和 nand 等都内置在语言中。
- 用户定义原语（UDP）创建的灵活性。用户定义的原语既可以是组合逻辑原语，也可以是时序逻辑原语。
- 开关级基本结构模型，例如 pmos 和 nmos 等也被内置在语言中。
- 提供显式语言结构指定设计中的端口到端口的时延及路径时延和设计的时序检查。
- 可采用三种不同方式或混合方式对设计建模。这些方式包括：行为描述方式—使用过程化结构建模；数据流方式—使用连续赋值语句方式建模；结构化方式—使用门和模块实例语句描述建模。
- Verilog HDL 中有两类数据类型：线网数据类型和寄存器数据类型。线网类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。
- 能够描述层次设计，可使用模块实例结构描述任何层次。
- 设计的规模可以是任意的；语言不对设计的规模（大小）施加任何限制。
- Verilog HDL 不再是某些公司的专有语言而是 IEEE 标准。
- 人和机器都可阅读 Verilog 语言，因此它可作为 EDA 的工具和设计者之间的交互语言。
- Verilog HDL 语言的描述能力能够通过使用编程语言接口（PLI）机制进一步扩展。PLI 是允许外部函数访问 Verilog 模块内信息、允许设计者与模拟器交互的例程集合。
- 设计能够在多个层次上加以描述，从开关级、门级、寄存器传送级（RTL）到算法级，包括进程和队列级。
- 能够使用内置开关级原语在开关级对设计完整建模。
- 同一语言可用于生成模拟激励和指定测试的验证约束条件，例如输入值的指定。
- Verilog HDL 能够监控模拟验证的执行，即模拟验证执行过程中设计的值能够被监控

和显示。这些值也能够用于与期望值比较，在不匹配的情况下，打印报告消息。

- 在行为级描述中， Verilog HDL 不仅能够在 RT L 级上进行设计描述，而且能够在体系结构级描述及其算法级行为上进行设计描述。
- 能够使用门和模块实例化语句在结构级进行结构描述。
- Verilog HDL 还具有内置逻辑函数，例如&（按位与）和|（按位或）。
- 对高级编程语言结构，例如条件语句、情况语句和循环语句，语言中都可以使用。
- 可以显式地对并发和定时进行建模。
- 提供强有力的文件读写能力。
- 语言在特定情况下是非确定性的，即在不同的模拟器上模型可以产生不同的结果；例如，事件队列上的事件顺序在标准中没有定义。

利用 Verilog HDL 语言，在 EDA 平台上可以轻松构建计算机主机系统的各主要部件，从而完成一台计算机主机的设计与实现。下面首先介绍 Verilog HDL 语言的基本语法规则，在此基础上，根据几个实例加深对构建计算机主机系统的理解。

3.1 Verilog HDL 基本架构

作为高级语言的一种，Verilog 语言以模块集合的形式来描述数字系统，其中每一个模块都有接口部分，用来描述与其它模块之间的连接。一般说来一个文件就是一个模块，但并不绝对如此。这些模块是并行运行的，但通常用一个高层模块来定义一个封闭的系统，包括测试数据和硬件描述。这一高层模块将调用其它模块的实例。

3.1.1 Verilog HDL 模块的结构

模块代表硬件上的逻辑实体，其范围可以从简单的门到整个大的系统。比如一个计数器、一个存储子系统、一个微处理器等。模块可以根据描述方法的不同定义成行为型或结构型（或者是二者的组合）。行为型模块通过传统的编程语言结构定义数字系统（模块）的状态，如使用 if 条件语句、赋值语句等。结构型模块将数字系统（模块）的状态表达为具有层次概念的互相连接的子模块。其最底层的元件必须是基元或已定义过的行为型模块。Verilog 的基元包括门电路，如与非门，和传输二极管（开关）。

模块的结构如下：

module <模块名> （＜端口列表＞）；

<定义>

<模块条目>

endmodule

其中<模块名>是模块唯一性的标识符；<端口列表>是输入、输出和双向端口的列表，这些端口用来与其它模块进行连接；<定义>一段程序用来指定数据对象为寄存器型、存储器型、线型以及过程块，诸如函数块和任务块；而<模块条目>可以是 initial 结构、always 结构、连续赋值或模块实例。

下面是一个 NAND 与非模块的行为型描述，输出 out 是输入 in1 和 in2 相与后求反的结果。

```
//与非门的行为型描述
module NAND（in1，in2，out）；
        input in1，in2；
        output out；
//连续赋值语句
        assign out=~（in1 & in2）；
endmodule
```

in1 和 in2 和 out 端口指定为线型的。assign 连续赋值语句不间断地监视等式右端变量，一旦其发生变化，右端表达式被重新赋值后结果传给等式左端进行输出。

连续赋值语句用来描述组合电路，一旦其输入发生变动，输出也随之而改变。

调用模块实例的一般形式为：

<模块名><参数列表><实例名>（<端口列表>）；

在此，<参数列表>是传输到模块实例的参数值，参数传递的典型应用是定义门级延迟。

3.1.2 逻辑功能定义

模块中最重要的部分是逻辑功能定义。有 3 种方法可在模块中描述逻辑。

（1）用“assign”语句

如：assign F=~((A&B) | (C&D))；

这种方法的句法很简单，只须写一个“assign”，后面再加一个方程式即可。

“assign”语句一般适合于对组合逻辑进行赋值，称为连续赋值方式。

（2）用元件例化（instantiate）

如：and myand3（f,a,b,c）；

这个语句利用 Verilog HDL 提供的与门库，定义了一个三输入的与门。采用实例元件的方法同在电路图输入方式下调入库文件一样，键入元件的名字和引脚的名字即可。要求每个实例元件的名字必须是唯一的。

（3）用“always”块语句

如：always @ (posedge clk) //每当时钟上升沿到来时执行一遍块内语句

```
    begin
        if (load)
            out<=data；
        else
            out<=out+cin；
    end
```

“always”块可用于产生各种逻辑，常用于描述时序逻辑。这个例子用“always”块生成了一个带有同步置数的计数器。“always”块可用很多种描述手段来表达逻辑，如此例中就用了 if-else 语句来表达逻辑关系。

3.2 数据类型及运算符

Verilog HDL 中共有 19 种数据类型。数据类型是用来表示数字电路中的数据存储和传送单元的。在此介绍 4 个最基本的数据类型。它们是 integer 型、parameter 型、reg 型、wire 型。其他的类型有 large 型、medium 型、scalared 型、time 型、small 型、tri 型、trio 型、tril 型、triand 型、trior 型、trireg 型、vectored 型、wand 型和 wor 型等。

Verilog HDL 中也有常量和变量之分，它们分别属于以上这些类型。下面对最常用的几种常量和变量进行介绍。

3.2.1 常量

在程序运行过程中，其值不能被改变的量称为常量。

Verilog HDL 中有三类常量：整型、实数型、字符串型。

下划线符号（_）可以随意用在整数或实数中，它们就数量本身没有意义。它们能用来提高易读性；唯一的限制是下划线符号不能用作为首字符。

（1）整型数

整型数可以按如下两种方式书写：

1）简单的十进制数格式

2）基数格式

1. 简单的十进制格式

这种形式的整数定义为带有一个可选的“+”（一元）或“-”（一元）操作符的数字序列。下面是这种简易十进制形式整数的例子。

32　　十进制数 32

-15　　十进制数-15

这种形式的整数值代表一个有符号的数。负数可使用两种补码形式表示。因此 32 在 5 位的二进制形式中为 10000，在 6 位二进制形式中为 110001；-15 在 5 位二进制形式中为 10001，在 6 位二进制形式中为 110001。

2. 基数表示法

这种形式的整数格式为：

[size]’base value

Size 定义以位计的常量的位长； base 为 o 或 O（表示八进制），b 或 B（表示二进制），d 或 D（表示十进制），h 或 H（表示十六进制）之一； value 是基于 base 的值的数字序列。值 x 和 z 以及十六进制中的 a 到 f 不区分大小写。

下面是一些具体实例：

5’O37	5 位八进制数
4’D2	4 位十进制数
4’B1x_01	4 位二进制数
7’Hx	7 位 x(扩展的 x)，即 xxxxxxx
4’hZ	4 位 z(扩展的 z)，即 zzzz
4’d-4	非法：数值不能为负

8'h 2 A　　在位长和字符之间,以及基数和数值之间允许出现空格

3 'b001　　非法: '和基数 b 之间不允许出现空格

(2+3)'b10　　非法:位长不能够为表达式

注意，x（或 z）在十六进制值中代表 4 位 x（或 z），在八进制中代表 3 位 x（或 z），在二进制中代表 1 位 x（或 z）。

基数格式计数形式的数通常为无符号数。这种形式的整型数的长度定义是可选的。如果没有定义一个整数型的长度，数的长度为相应值中定义的位数。下面是两个例子：

'o721　　9 位八进制数

'hAF　　8 位十六进制数

如果定义的长度比为常量指定的长度长，通常在左边填 0 补位。但是如果数最左边一位为 x 或 z，就相应地用 x 或 z 在左边补位。例如：

10'b10　　左边添 0 占位, 0000000010

10'bx0x1　　左边添 x 占位, xxxxxxx0x1

如果长度定义得更小，那么最左边的位相应地被截断。例如：

3'b 1 0 0 1 _ 0 0 1 1 与 3'b011 相等

5'H0FFF 与 5'H1F 相等

? 字符在数中可以代替值 z 在值 z 被解释为不分大小写的情况下提高可读性。

（2）实数

实数可以用下列两种形式定义：

1）十进制计数法；例如

2.0

5.678

11156.72

0.1

2.　　//非法：小数点两侧必须有 1 位数字

2）科学计数法； 这种形式的实数举例如下：

23_5.1e2　其值为 23510.0；忽略下划线

3.6E2　　360.0 （e 与 E 相同）

5E-4　　0.0005

Verilog HDL 语言定义了实数如何隐式地转换为整数。实数通过四舍五入被转换为最相近的整数。

42.446，42.45　　转换为整数 42

92.5, 92.699　　转换为整数 93

-15.62　　转换为整数-16

-26.22　　转换为整数-26

（3） 字符串

字符串是双引号内的字符序列。字符串不能分成多行书写。例如:

‘INTERNAL ERROR’

‘REACHED->HERE’

用 8 位 ASCII 值表示的字符可看作是无符号整数。因此字符串是 8 位 ASCII 值的序列。为存储字符串“INTERNAL ERROR”，变量需要 8*14 位。

```
reg [1:8*14] message;
. . .
message= ‘INTERNAL ERROR’
```

3.2.2 变量

变量是在程序运行过程中其值可以改变的量。变量分为两种：一种为网络型（nets type），另一种为寄存器型（register type）。下面分别对这两种变量进行说明。

（1）nets 型变量

Nets 型变量指输出始终根据输入的变化而更新其值得变量，它一般指的是硬件电路中的各种物理连接。Verilog HDL 中提供了多种 nets 型变量，这里着重介绍 wire 型变量。wire 是一种最常用的 nets 型变量，wire 型数据常用来表示以 assign 语句赋值的组合逻辑信号。Verilog HDL 模块中的输入/输出信号类型缺省时自动定义为 wire 型。wire 型信号可以用做任何方程式的输入，也可以用做“assign”语句和实例元件的输出。

wire 型变量的定义格式如下：

wire 数据名 1，数据名 2，……，数据名 n；

例如：

```
wire a，b;          //定义了两个 wire 型变量 a，b
```

上面两个变量 a，b 的宽度都是 1 位，若定义一个向量，可按以下方式：

wire[n-1：0] 数据名 1，数据名 2，……，数据名 n；

wire[n：1] 数据名 1，数据名 2，……，数据名 n；

它们定义了数据的宽度为 n 位。如下面定义了 8 位宽的数据总线，20 位宽的地址总线：

```
wire[7：0]  databus;
wire[7：0]  addrbus;
```

（2）register 型变量

register 型变量对应的是具有状态保持作用的电路元件，如触发器、寄存器等。register 型变量与 nets 型变量的根本区别在于：register 型变量需要被明确地赋值，并且在被重新赋值前一直保持原值。在设计中必须将寄存器型变量放在过程块语句中，通过过程赋值语句赋值。

Verilog HDL 中，有 4 种寄存器型变量：reg、integer、real、time，reg 型变量是最常用的一种寄存器型变量，下面着重对其进行介绍。

reg 型变量的定义格式类似于 wire 型，具体格式为：

reg 数据名 1，数据名 2，……，数据名 n；

例如：

reg a，b；　//定义了两个 reg 型变量 a，b

上面两个变量 a，b 的宽度都是 1 位，若定义一个向量，可按以下方式：

reg[n-1：0] 数据名 1，数据名 2，……，数据名 n；

reg[n：1] 数据名 1，数据名 2，……，数据名 n；

它们定义了数据的宽度为 n 位。如下面定义了 8 位宽的数据：

reg[7：0]　data；

reg[8：1]　data；

（3）数组

若干个相同宽度的向量构成数组，reg 型数组变量即为 memory 型变量可，即可定义存储器型数据。如：

reg[7：0] mymem[1023：0]；

上面的语句定义了一个 1024 个字节、每个字节宽度为 8 位的存储器。通常，存储器采用如下方式定义：

parameter wordwidth=8，memsize=1024；

reg[wordwidth-1：0] mymem[memsize-1：0]；

上面的语句定义了一个宽度为 8 位、1024 个存储单元的存储器，该存储器的名字是 mymem，若对该存储器中的某一单元赋值，采用如下方式：

mymem[8]=1；　　//mymem 存储器中的第 8 个单元赋值为 1

3.2.3 运算符

Verilog HDL 的运算符范围很广，按功能份包括以下几类：算术运算符、逻辑运算符、关系运算符、等式运算符、缩减运算符、条件运算符、位运算符、移位运算符和拼接运算符共 9 类。下面对常用的一些运算符加以说明。

（1）算术运算符

常用的算术运算符包括：+（加）、-（减）、*（乘）、/（除）、%（求模），都属于双目运算符，即每个运算符可带两个操作数。

（2）逻辑运算符

常用的逻辑运算符包括：&&（逻辑与）、||（逻辑或）、！（逻辑非）。

（3）位运算符

位运算是将两个操作数按对应位进行逻辑运算。位运算符包括：~（按位取反）、&（按位与）、|（按位或）、^（按位异或）、^~或~^（按位同或）。

（4）关系运算符

关系运算符包括：<（小于）、<=（小于或等于）、>（大于）、>=（大于或等于），其中“<=”操作符也用于表示信号的一种赋值操作。

在进行关系运算时，如果声明的关系是假，则返回值是 0；如果声明的关系是真，则返回值是 1；如果某个操作书的值不定，则其结果是模糊的，返回值是不定值。

（5）移位运算符

移位运算符包括：>>（右移）、<<（左移），Verilog HDL 的移位运算符只有左移和右移两个。其用法为：

A>>n 或 A<<n

表示把操作数 A 右移或左移 n 位，同时用 0 填充移出的位。

3.3 常用语句

Verilog HDL 支持许多语句，从而成为结构化和过程性的语言。Verilog HDL 的语句包括：赋值语句、条件语句、循环语句、结构说明语句和编译预处理语句等。每一类又包括几种不同的语句。下面分别介绍上述语句，重点介绍编程中常用的可综合的语句，并对语句的顺序执行与并行执行进行说明。

3.3.1 赋值语句

在 Verilog HDL 中，信号有以下赋值方式和赋值语句。

（1）连续赋值语句

assign 为连续赋值语句，它用于对 wire 型变量进行赋值。如：

assign c=a&b;

在上面的赋值中，a、b、c 三个变量皆为 wire 型变量，a 和 b 信号的任何变化，都将随时反映到 c 上来，因此称为连续赋值方式。

（2）过程赋值语句

过程赋值语句用于对寄存器类型的变量进行赋值。过程赋值有以下两种方式。

1）非阻塞赋值方式

赋值符号为<=，如 b<=a;

非阻塞赋值在块结束时才完成赋值操作，即 b 的值并不是立刻就改变的。

2）阻塞赋值方式

赋值符号为=，如 b=a;

阻塞赋值在该语句结束时就完成赋值操作，即 b 的值在该赋值语句结束后立刻改变。如果在一个块语句中，有多条阻塞赋值语句，那么在前面的赋值语句没有完成之前，后面的语句就不能被执行，就像被阻塞了一样，因此称为阻塞赋值方式。

3.3.2 条件语句

条件语句有 if-else 语句和 case 语句两种。它们都是顺序语句，应放在“always”块内。下面对这两种语句分别进行介绍。

（1）if-else 语句

其格式与 C 语言中 if-else 语句类似，使用方法有以下 3 种。

1）if（表达式）语句 1;

2）if（表达式）语句 1;

else 语句 2；

3）if （表达式 1） 语句 1；

else if（表达式 2）语句 2；

else if （表达式 3）语句 3；

……

else if （表达式 n）语句 n；

else 语句 n+1；

这三种方式中，“表达式”一般为逻辑表达式或关系表达式，也可能是一位的变量。系统对表达式的值进行判断，若为 0，x，z，按“假”处理；若为 1，按“真”处理，执行指定语句。语句可以是单句，也可是多句，多句时用“begin-end”语句括起来。对于 if 语句的嵌套，若不清楚 if 和 else 的匹配，最好也用“begin-end”语句括起来。

（2）case 语句

相对 if 语句只有两个分支而言，case 语句是一种多分支语句，故 case 语句多用于多条件译码电路，如描述译码器、数据选择器等。case 语句有 case、casez、casex 共 3 种表示方式，这里分别予以说明。

1）case 语句

case 语句的使用格式如下：

```
case（敏感表达式）
    值 1：语句 1；
    值 2：语句 2；
    ……
    值 n：语句 n；
    default：语句 n+1；
endcase
```

当敏感表达式的值为值 1 时，执行语句 1；为值 2 时，执行语句 2；依次类推，如果敏感表达式的值与上面列出的值都不相同，则执行 default 后面的语句。

2）casez 与 casex 语句

case 语句中，敏感表达式与值 1~值 n 间的比较是一种全等比较，必须保证两者的对应位全等。casez 与 casex 语句是 case 语句的两种变体，三者的表示形式中唯一的区别是三个关键词的不同。在 casez 语句中，如果分支表达式某些位的值为高阻 z，那么对这些位的比较就不予考虑，因此只需关注其他位的比较结果。而在 casex 语句中，则把这种处理方式进一步扩展到对 x 的处理，即如果比较双方有一方的某些位的值是 x 或 z，那么这些位的比较就都不予考虑。

3.3.3 循环语句

在 Verilog HDL 中存在 4 种类型的循环语句，用来控制语句的执行次数。这 4 种语句分别为：

1. forever　连续地执行语句，多用在“initial”块中，以生成周期性输入波形。

2. repeat　连续执行一条语句 n 次。

3. while　执行一条语句，直到某个条件不满足。

4. for 语句。

（1）for 语句

for 语句的使用格式如下（同 c 语言）

for （表达式 1；表达式 2；表达式 3） 语句

即：for（循环变量赋初值；循环结束条件；循环变量增值）执行语句

（2）repeat 语句

repeat 语句的使用格式为：

repeat（循环次数表达式） 语句；

或 repeat（循环次数表达式）begin

……

end

（3）while 和 forever 语句

while 语句的使用格式如下：

while（循环执行条件表达式） 语句；

或 while（循环执行条件表达式）begin

……

end

while 语句在执行时，首先判断循环条件表达式是否为真。若为真，执行后面的语句或语句块，然后再回头判断循环执行条件表达式是否为真；若为真，再执行一次后面的语句，如此不断，直到条件表达式不为真。

forever 语句的使用格式如下：

forever　语句；

或 forever begin

……

end

forever 循环语句连续不断地执行后面的语句或语句块，常用来产生周期性的波形，作为仿真激励信号。

3.3.4 结构说明语句

Verilog HDL 中的任何过程模块都从属于以下 4 种结构说明语句；initial、always、task、function。在一个模块中，使用 initial 和 always 语句的次数是不受限制的。initial 说明语句一般用于仿真中的初始化，仅执行一次；always 块内的语句则是不断重复执行的；task 和 function 语句可以在程序模块中的一处或多处调用。

（1）always 块语句

always 块语句模板如下：

always @ （<敏感信号表达式>）

begin

//过程赋值

//if 语句

//case 语句

//while，repeat，for 循环

//task，function 调用

end

敏感信号表达式又称事件表达式或敏感表，当该表达式的值改变时，就会执行一遍块内语句。

对于时序电路，事件是由时钟边沿触发的。为表达边沿这个概念，Verilog HDL 提供了 posedge 和 negedge 两个关键字来描述。

（2）initial 语句

initial 语句的使用格式如下：

initial

begin

语句 1；

语句 2；

……

end

（3）task 和 function 语句

task 和 function 语句分别用来定义任务和函数。利用任务和函数可以把一个大的程序模块分解成许多小的任务和函数，以方便测试，并且能使写出的程序清晰易懂。

task 定义与调用的格式如下：

定义：**task <任务名>**

端口及数据类型声明语句；

其他语句；

endtask

任务调用的格式为：

<任务名>（端口 1，端口 2，……）；

函数的目的是返回一个用于表达式的值。函数的定义格式为：

function <返回值位宽或类型说明> 函数名；

端口声明；

局部变量定义；

其他语句；

endfunction

函数的调用是通过将函数作为表达式中的操作书来实现的。调用格式如下：

<函数名>（<表达式><表达式>）；

函数的使用与任务相比有更多的限制和约束。例如，函数不能启动任务，在函数中不能包含任何的时间控制语句，同时定义函数时至少要有一个输入参量等。

3.3.5 编译预处理语句

Verilog HDL 和 c 语言一样也提供了编译预处理功能。Verilog HDL 允许在程序中使用特殊的编译预处理语句。在编译时，通常先对这些特殊语句进行预处理，然后再将预处理的结果和源程序一起进行编译。

预处理命令以符号“′ ”开头，以区别于其他语句。Verilog HDL 提供了十几条编译预处理语句，比较常用的有′ define，′ include 等语句，下面分别介绍这些常用的预处理语句。

（1）′ define 语句

′ define 语句用来将一个简单的名字或标志符（或称为宏名）来代表一个复杂的名字或字符串，其一般形式为：

′ define 标志符（宏名）字符串

采用宏定义，可以简化程序的书写，而且便于修改。

（2）′ include 语句

′ include 是文件包含语句，它可将一个文件全部包含到另一个文件中。其一般形式为：

′ include “文件名”

′ include 语句可以出现在源程序的任何地方，被包含的文件若与包含文件不在同一个目录下，必须指明其路径。

3.4 Verilog HDL 设计数字电路实例

具备了 Verilog HDL 程序设计的基本语法规则后，在本节中将通过具体的实例说明如何用 Verilog HDL 设计出计算机主机相关部件电路。本节中所有程序均采用 Quartus Ⅱ软件设计、编译及模拟仿真加以实现，在此给出时序模拟仿真波形图。

3.4.1 时序发生器的设计

在计算机系统主机设计过程中，针对由不同形式所组成的 CPU 中的控制部件，可以有不同的时序系统的设计。

对于微程序设计而言，由于每个微周期执行一条微指令，为了在主机系统设计过程中尽量发挥出机器数据通路的并行性，利用 Verilog HDL 语言设计出了相应的时序发生器。时序发生器提供一个微周期中的八个电平及脉冲型控制信号，可供主机系统设计时使用。时序发生器的详细设计源代码如下：

```
module counter (clk, halt,m,m1,m2,m3,m4,m5,m6,m7);

    input clk,halt;                          //定义输入管脚
    output m,m1,m2,m3,m4,m5,m6,m7;           //定义 8 个输出管脚

    reg  m,m1,m3,m5,m6,m7,m12,m14;
    reg  [2:0] cnt;                          //定义计数变量
```

```
assign m2=m12 & m1 & halt;                    //生成 m2 信号
assign m4=m14 & m3 & halt;                    //生成 m4 信号
always @ (negedge m)
        begin
                m12 <= m1 & ~m & halt;
                m14 <= m3 & ~m & halt;
        end
always @ (posedge clk)
        begin
                if (halt)
                        m <= ~m;              //产生 m 信号
                else
                        m <= 0;
        end
always @ (posedge m)                          //通过计数生成其他输出管脚信号
        begin

                if (halt)
                        case (cnt)
                                0:      {m5, m1, m3, m6, m7} <= 5'b10000;
                                1:      {m5, m1, m3, m6, m7} <= 5'b00010;
                                2:      {m5, m1, m3, m6, m7} <= 5'b00001;
                                5:      {m5, m1, m3, m6, m7} <= 5'b01000;
                                6:      {m5, m1, m3, m6, m7} <= 5'b00100;
                                default: {m5, m1, m3, m6, m7} <= 5'b00000;
                        endcase
                cnt <= cnt+1'b1;
        end
endmodule
```

在 Quartus II 软件中经过编译、模拟仿真、封装后得到其芯片封装图如图 3-1 所示。封装后的芯片名为 counter。时序发生器相应的输出波形如图 3-2 所示。

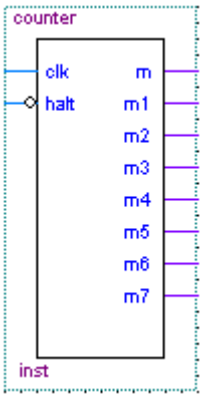


图 3-1 时序发生器芯片封装图

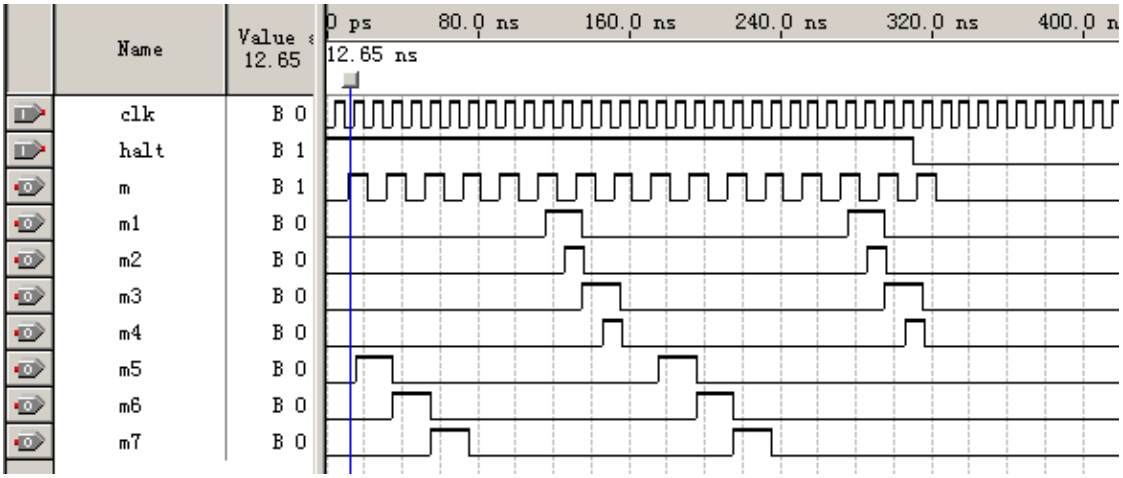


图 3-2　时序发生器输出波形图

图 3-2 中的 CLK 是控制机器运行的唯一外部时钟脉冲输入管脚，可以在测试或使机器运行时按需要自行设定。HALT 是输入管脚，由此控制着时序发生器的工作与停止。在 HALT 为高电位时，时序发生器在 CLK 时钟脉冲的控制下，循环往复地产生相应的时序信号，当 HALT 为低电位时，则使时序发生器停止各时序信号的产生，由此而达到停机的效果。M、M_1～M_7 是八个输出管脚，可作为机器设计时选用的各个时序信号。其中 M 管脚输出的信号相当于时钟脉冲 CLK 经过分频后得到的控制机器的工作脉冲信号，可用于控制机器各部件的运转。M_2、M_4 管脚输出的是脉冲型控制信号，M_1、M_3、M_5、M_6 、M_7 管脚输出的是电位型控制信号。各信号可以单独使用，也可以相互配合控制相应器件的操作。

3.4.2 运算器的设计

运算器（ALU）功能部件是为了完成主机系统设计中的算术/逻辑运算功能而设计的功能部件，是计算机进行算术/逻辑运算的核心部件。在本例中设计的运算器功能部件可以对 8 位数据进行算术/逻辑运算。此部件采用了两片 4 位片的 74181，通过串行进位而扩展成 8 位运算器。运算器的详细设计源代码如下：

```
module Alu(M, CPSA, CPSB, ALU_BUS, CN, IB_IN, S, IB_OUT);

    input      M, CPSA, CPSB, ALU_BUS, CN;          //定义输入管脚
    input [7:0] IB_IN;
    input      [3:0] S;
    output [7:0] IB_OUT;                            //定义输出管脚

    wire [7:0] z;
    wire cn4;

    reg [7:0] a, b;

    \74181    inst1 (.A0N(a[0]), .A1N(a[1]), .A2N(a[2]), .A3N(a[3]),
              .B0N(b[0]), .B1N(b[1]), .B2N(b[2]), .B3N(b[3]),
```

```
                .S0 (S[0]), .S1 (S[1]), .S2 (S[2]), .S3 (S[3]),
                .M  (M),    .CN (CN),
                .F0N(z[0]), .F1N(z[1]), .F2N(z[2]), .F3N(z[3]),
                .CN4(cn4));

        \74181    inst2 (.A0N(a[4]), .A1N(a[5]), .A2N(a[6]), .A3N(a[7]),
                .B0N(b[4]), .B1N(b[5]), .B2N(b[6]), .B3N(b[7]),
                 .S0 (S[0]), .S1 (S[1]), .S2 (S[2]), .S3 (S[3]),
                .M  (M),    .CN (cn4),
                .F0N(z[4]), .F1N(z[5]), .F2N(z[6]), .F3N(z[7]));

        assign IB_OUT = ~ALU_BUS ? z : 8'hzz;          //根据 ALU_BUS 控制信号给输出赋值

        always @ (posedge CPSA)                  //根据 CPSA 和 CPSB 产生时刻复用输入管脚
                begin
                        a <= IB_IN;
                end

        always @ (posedge CPSB)
                begin
                        b <= IB_IN;
                end
endmodule
```

在 Quartus II 软件中经过编译、模拟仿真、封装后得到其芯片封装图如图 3-3 所示。封装后的芯片名为 Alu。时序发生器相应的输出波形如图 3-4 所示。

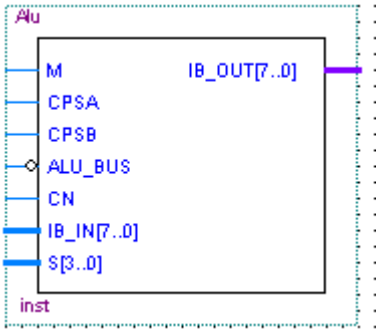


图 3-3　8 位运算器芯片封装图

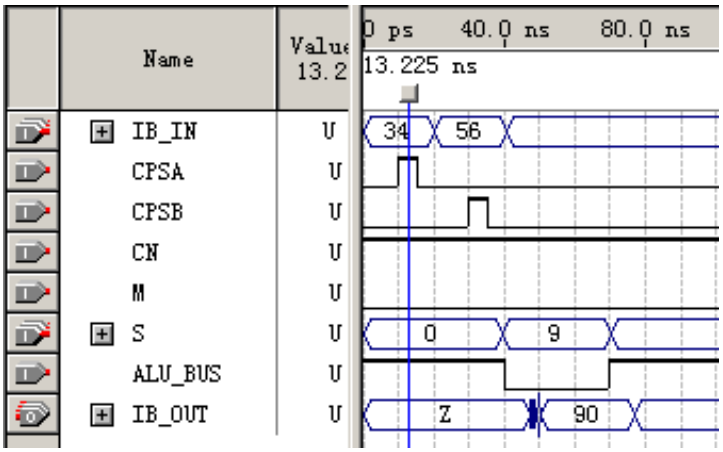


图 3-4　8 位运算器输出波形图

图 3-4 中 CPSA 和 CPSB 为脉冲型控制信号输入管脚，分别控制完成接收 SA 和 SB 暂存器输入管脚（即：A 端和 B 端）的数据，IB_IN_0～IB_IN_7 根据 CPSA 和 CPSB 的具体产生时刻分别作为两个 8 位操作数的输入管脚，IB_OUT_0～IB_OUT_7 为算术/逻辑运算 8 位数据输出管脚。ALU_BUS 控制信号，完成将算术/逻辑运算结果输出的功能。CN、M、S_0～S_3 为算术/逻辑运算控制信号。

3.4.3 Verilog HDL 设计主机系统

通过上述介绍的时序发生器和运算器设计实例，可以看出利用 Verilog HDL 语言可以方便、快速地通过编写程序的手段设计出相应的计算机主机系统。

在 EDA 平台中，利用 Verilog HDL 语言设计一台计算机主机系统的基本过程如下：首先利用 Verilog HDL 语言实现并封装计算机主机系统的各个部件，如：主存储器功能部件、运算器功能部件、程序计数器、总线暂存器、通用寄存器组等；然后合理设计出计算机主机系统数据通路，并拟定相应的指令系统；在此基础上，根据信息在数据通路中流动、读取及存储的需要，设计出相应的各种微操作命令，从而在相应时序系统的配合下，利用 Verilog HDL 语言编程，完成各部件的设计、编译及模拟仿真。最终，利用 Verilog HDL 语言实现一台计算机主机系统的设计。最后，使设计出的计算机主机在机器加电、产生频率稳定的主振信号后，能够根据控制部件发出的控制信号自动地、连续地执行存储在主存（RAM）中的程序。通过验证机器运行时建立的时序模拟仿真输出波形图，以验证机器逻辑设计的正确性。

第 4 章 运算器功能部件设计与调试范例

4.1 运算器功能部件的设计

4.1.1 运算器功能部件的总体设计

运算器（ALU）功能部件是为了完成计算机主机系统设计实践的算术/逻辑运算功能而设计的功能部件，是计算机进行算术/逻辑运算的核心部件。在本范例中设计的运算器功能部件可以对 8 位数据进行算术/逻辑运算。此部件采用了两片 4 位片的 74181，通过串行进位而扩展成 8 位运算器。暂存器(74273)对从总线上面传来的数据进行寄存，可以起到暂存数据的作用。三态门(74244)由控制信号 ALU-BUS 控制，保证 ALU 运算所得到的结果在需要时送上总线，完成算术逻辑运算。运算器功能部件的设计原理图如图 4-1 所示。该运算器功能部件的芯片封装图（芯片外特性图）如图 4-2 所示。封装后的芯片名为 1812。

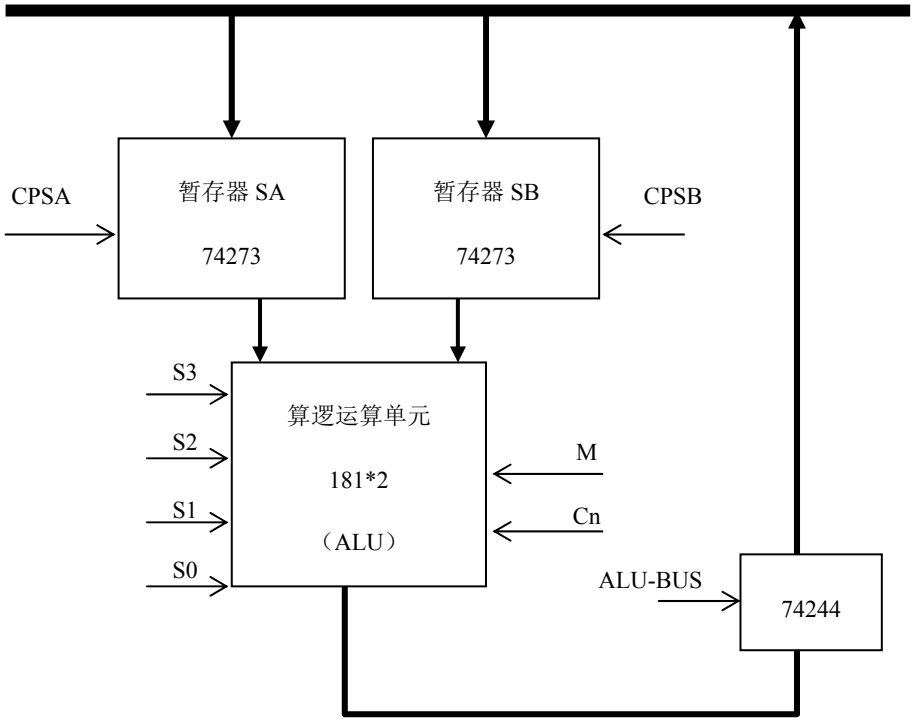


图 4-1 运算器功能部件原理图

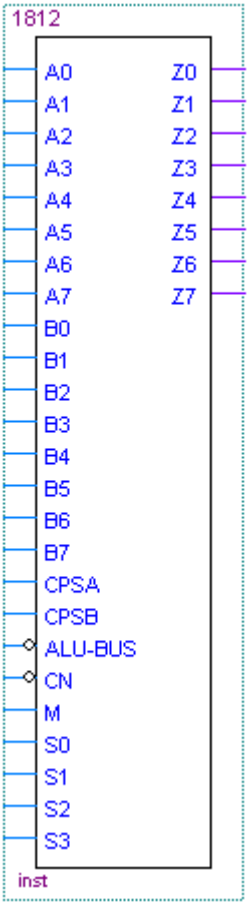


图 4-2 运算器功能部件芯片封装图

图 4-2 中 A_0～A_7 和 B_0～B_7 作为两个 8 位操作数的输入管脚，Z_0～Z_7 为算术/逻辑运算 8 位数据输出管脚。CPSA 和 CPSB 为脉冲型控制信号输入管脚，分别控制完成接收 SA 和 SB 暂存器输入管脚（即：A 端和 B 端）的数据。ALU-BUS 控制信号，完成将算术/逻辑运算结果通过三态门（74LS244）输出的功能。C_n、M、S_0～S_3 为算术/逻辑运算控制信号。

4.1.2 运算器功能部件的详细设计

运算器功能部件设计流程如下所示：

1．设置项目名并指定项目需保存的目录（指定项目的方法详见第 2 章）。

2．打开图形编辑器，建立新文件（打开图形编辑器及建立新文件的方法详见第 2 章）。

3．选择芯片，在合适的位置上输入相应的芯片符号。在此基础上进行芯片间的连线（选择芯片的方法详见第 2 章）。

4．确定输入/输出管脚（确定输入/输出管脚的方法详见第 2 章）。至此，完成了运算器功能部件的设计。在此基础上存盘，以保存设计好的部件。运算器功能部件详细设计图如图 4-3 所示。

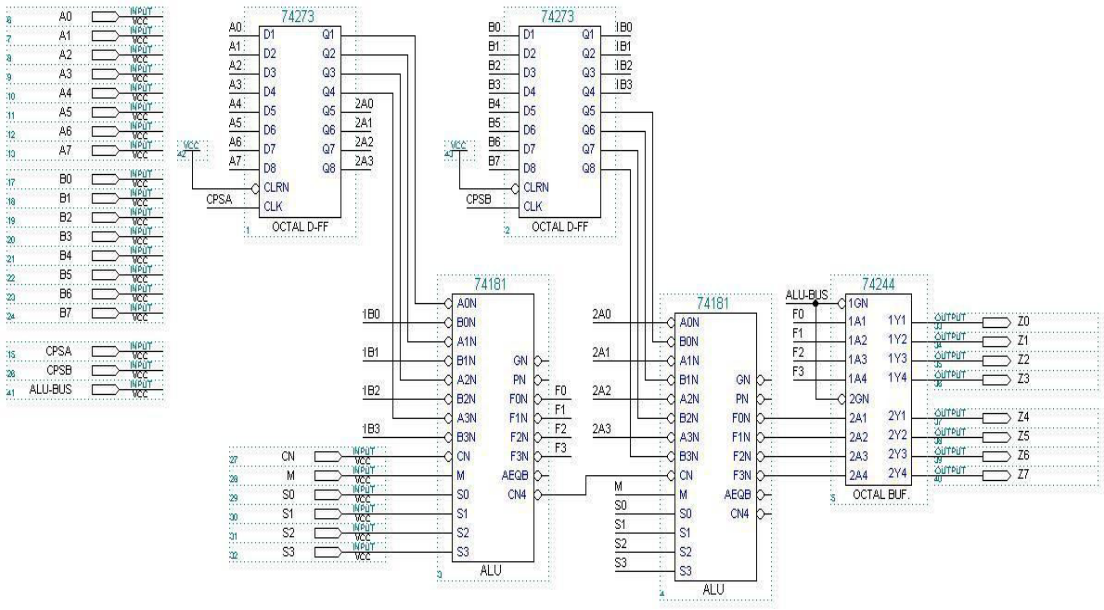


图 4-3 运算器功能部件详细设计图

4.2 运算器功能部件的调试与封装

4.2.1 运算器功能部件的调试

为了检测部件设计的正确性，需要对设计出的部件进行测试。

1．编译

编译的目的是为了检测设计的线路在物理上是否正确。若编辑的图形文件有错，则编译停止，需要对设计的线路进行修改，直至编译通过（编译方法详见第 2 章）。

2．模拟仿真

模拟仿真是为了验证设计好的部件在逻辑设计上的正确性。首先创建、编辑波形文件，按照模拟要求输入每个管脚的电位及脉冲信号，建立起测试部件所需要的测试输入波形。然后，进行模拟仿真。在模拟仿真通过后，可以通过模拟仿真的输出波形，以检测部件逻辑设计的正确性（模拟仿真方法详见第 2 章）。

例如：将 02H 作为 ALU 的 A 端数据写入暂存器 SA，将 03H 作为 ALU 的 B 端数据写入暂存器 SB 之后，对运算器功能部件 1812 进行 A 加 B、A 减 B、A 端加 1、A 与 B、A 或 B、A 端取反的测试波形如图 4-4 所示。

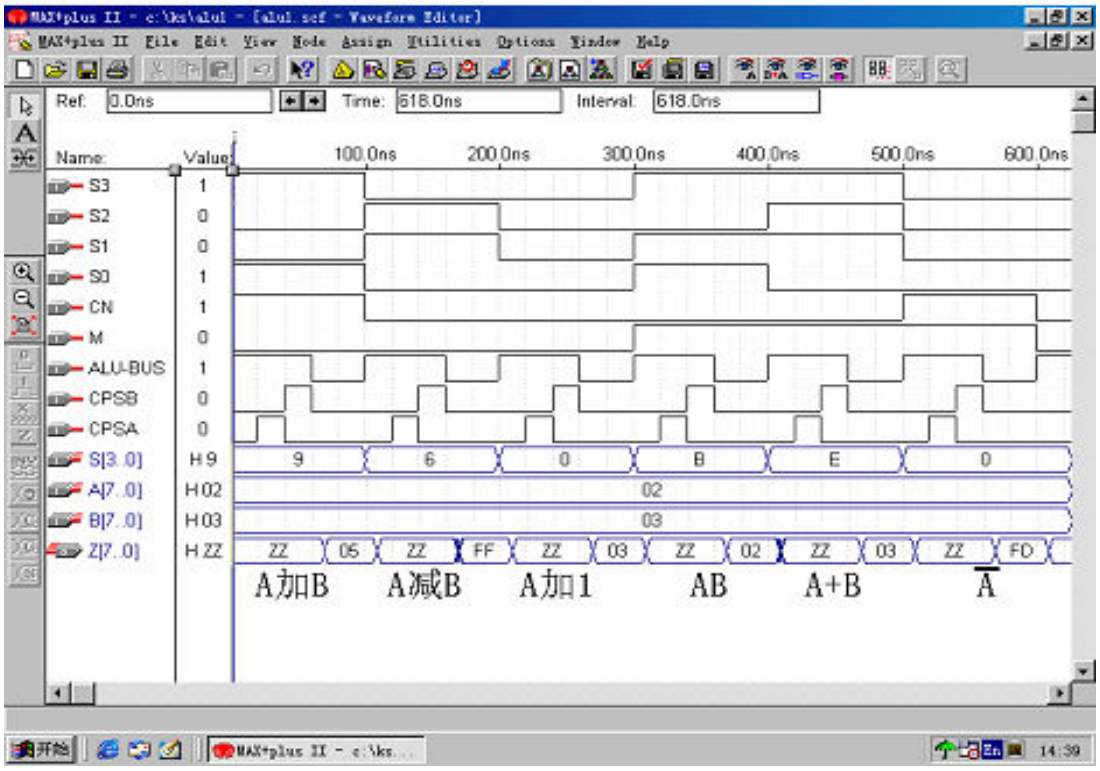


图 4-4 运算器功能部件测试波形图

4.2.2 运算器功能部件的封装

芯片封装的目的是为了把具有某一特定功能的设计图构成一个芯片，这样不仅可以避免某一图形设计文件过于复杂、难于修改，而且可以供其它设计调用。在调用封装好的芯片时，将不再考虑芯片的内部结构，而只需考虑封装后的输入/输出管脚的功能即可。在此包含了封装和继承的概念。芯片的封装是把当前设计的图形文件（*.gdf）封装成芯片文件（*.sym）。封装后的芯片的调用，与调用其它元器件的方法相似，不同之处只是不需选用芯片库，只要选定芯片所在的目录中的*.sym 文件即可(芯片封装的方法详见第 2 章)。运算器功能部件的芯片封装图可参见图 4-2。74181 的功能表详见附录 B。

第 5 章 计算机主机系统基本部件的设计及实验

在 EDA 平台中，要求可以进行计算机组成原理的功能部件等硬件的实验，也可以进行计算机主机系统设计。计算机主机系统设计是难度较大的实验，该课程实践主要进行一台完整计算机的数据通路、控制信号等的逻辑设计及详细设计，同时拟定该机的指令系统，最终使用该指令系统中的指令，在设计出的机器上编程，并经过机器的运行，以测试该机器逻辑设计的正确性。

为了能够更好地进行计算机主机系统设计，在此对实现主机系统设计所需要的基本部件进行了设计、测试及封装，供主机系统设计使用。

在各部件的设计过程中，为了使设计好的各部件可以直接与总线相连，考虑到总线在任一时刻最多只能由一个部件所占用，因此对于不具备三态输出的所有器件都采用三态门（74244）来隔离总线。只有当需要向总线传送数据时，才开启三态门，进行部件与总线的数据传输。

5.1 利用 MaxPlus II 及 Quartus II 设计基本部件

5.1.1 时序发生器的设计

在计算机系统主机设计过程中，针对由不同形式所组成的 CPU 中的控制部件，可以有不同的时序系统的设计。在此对于用微程序设计来实现控制部件的主机系统设计而设计出了相应的时序发生器。

对于微程序设计而言，由于每个微周期执行一条微指令，为了在主机系统设计过程中尽量发挥出机器数据通路的并行性，设计出了相应的时序发生器。时序发生器（timer）提供一个微周期中的八个电平及脉冲型控制信号，可供主机系统设计时使用。时序发生器的芯片封装图如图 5-1 所示。封装后的芯片名为 timer。时序发生器相应的输出波形如图 5-2 所示。时序发生器的详细设计图可参见附录 A 中的附图 A-3。

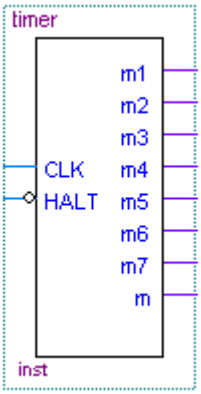


图 5-1 时序发生器芯片封装图

图 5-2 中的 CLK 是控制机器运行的唯一外部时钟脉冲输入管脚，可以在测试或使机器运行时按需要自行设定。HALT 是输入管脚，由此控制着时序发生器的工作与停止。在 HALT 为高电位时，时序发生器在 CLK 时钟脉冲的控制下，循环往复地产生相应的时序信号，当 HALT 为低电位时，则使时序发生器停止各时序信号的产生，由此而达到停机的效果。M、M_1～M_7 是八个输出管脚，其输出信号可作为机器设计时选用的各个时序信号。其中 M 管脚输出的信号相当于时钟脉冲 CLK 经过分频后得到的时序信号，可作为控制机器工作的工作脉冲信号，控制机器各部件的运转。M_2、M_4 管脚输出的是脉冲型控制信号，M_1、M_3、M_5、M_6 、M_7 管脚输出的是电平型控制信号。各信号可以单独使用，也可以相互配合控制相应器件的操作。

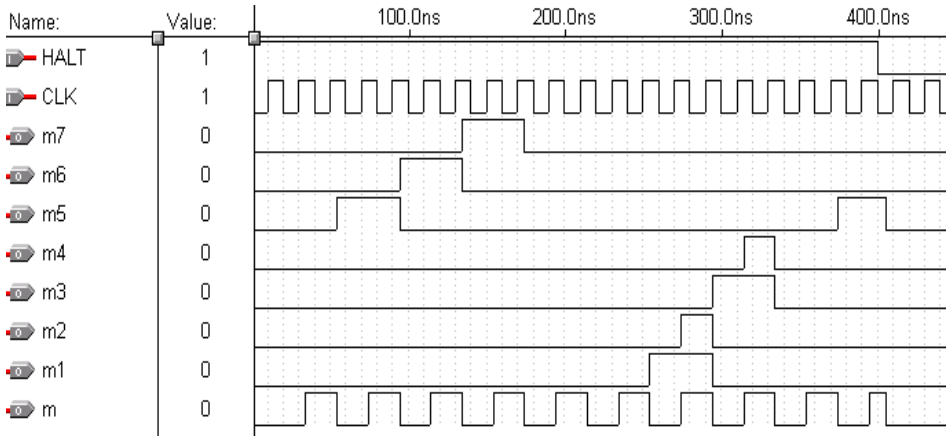


图 5-2　时序发生器输出波形图

5.1.2 存储器的设计

存储器功能部件（Memory）是为了提供存储数据和程序而设计的功能部件。可作为主机系统的主存储器使用。

在存储器功能部件中，设计了主存地址寄存器（MAR）和主存数据寄存器（MDR），作为主存与 CPU 进行数据交换的接口。MAR 接收、暂存总线上的主存地址，MDR 暂存输出到总线上的数据。存储器功能部件的设计原理图如图 5-3 所示。其相应的芯片封装图如图 5-4 所示。封装后的芯片名为 ram2。存储器功能部件的详细设计图可参见附录 A 中的附图 A-9。图 5-4 中 A_0～A_7 作为存储器的地址线，B_0～B_7 作为数据输入线，O_0～O_7 作为数据输出线。CPMAR 和 CPMDR 管脚分别作为主存地址寄存器（MAR）和主存数据寄存器（MDR）的数据接收控制信号。RD 作为存储器的读控制信号（上升沿触发），在 RD 为高电位（即：存储器数据输出有效）期间，应发出 CPMDR 脉冲控制信号，使存储器的读出数据锁存到存储器数据寄存器 MDR 中。WR 和 WRE 分别作为存储器的写控制信号和写使能控制信号，在 WRE 为高电位期间，发出 WR 脉冲控制信号（上升沿有效），则可以把输入数据写入到主存储器中。RAM-BUS 控制信号（低电位有效）完成将存储器的输出数据通过三态门输出的功能。

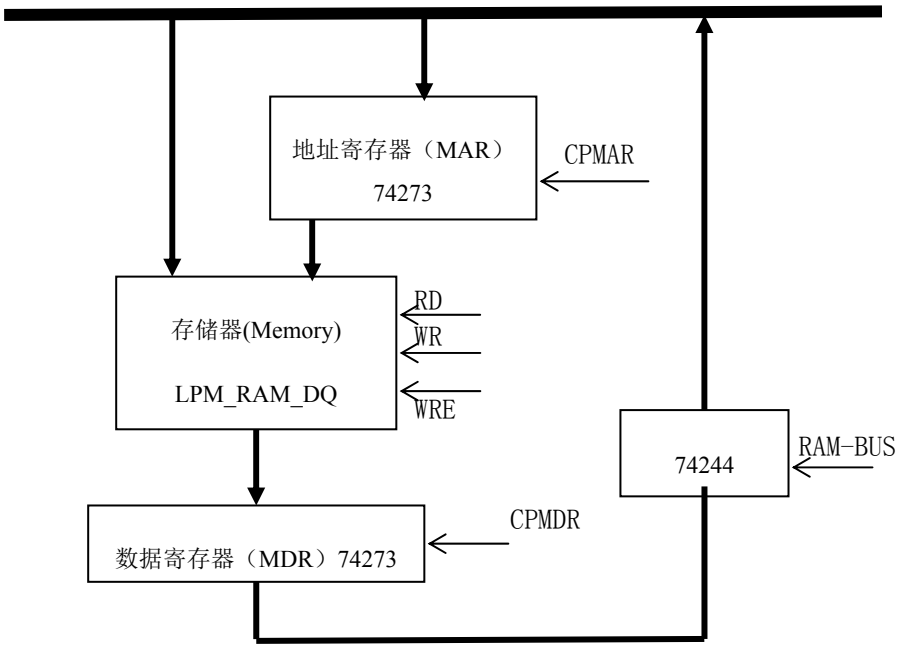


图 5-3 存储器功能部件设计原理图

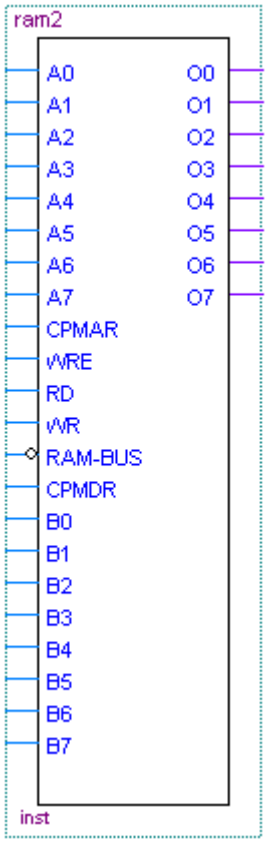


图 5-4 存储器功能部件芯片封装图

5.1.3 运算器的设计

运算器（ALU）功能部件是为了完成主机系统的算术/逻辑运算功能而设计的功能部件，是计算机进行算术/逻辑运算的核心部件。在本范例中设计的运算器功能部件可以对 8 位数据进行算术/逻辑运算。此部件采用了两片 4 位片的 74181，通过串行进位而扩展成 8 位运算器。暂存器(74273)对从总线上面传来的数据进行寄存,可以起到暂存数据的作用。三态门(74244)由控制信号 ALU-BUS（低电位有效）控制,保证 ALU 运算所得到的结果在需要时送上总线,完成算术逻辑运算。运算器功能部件的设计原理图如图 5-5 所示。该运算器功能部件的芯片封装图（芯片外特性图）如图 5-6 所示。封装后的芯片名为 1812。

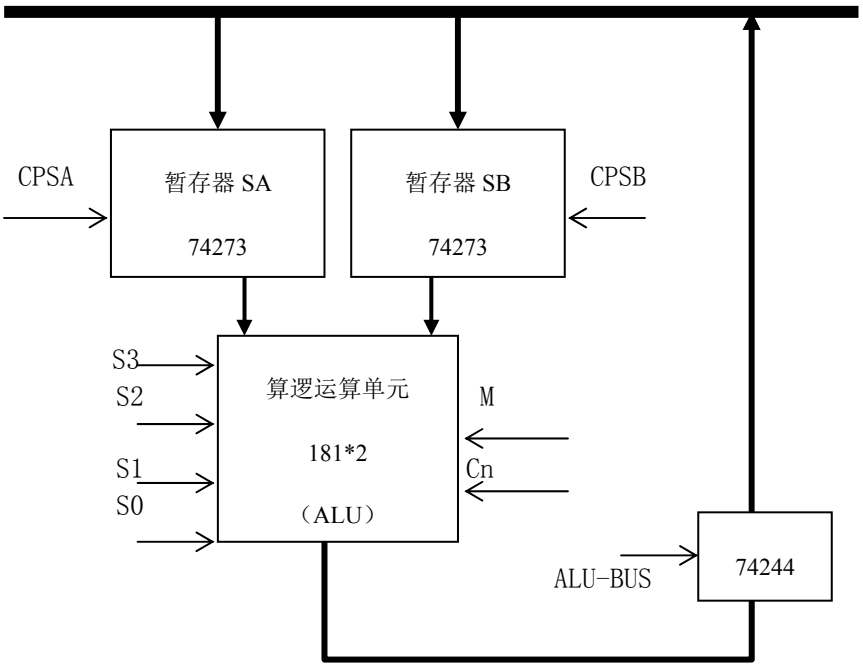


图 5-5 运算器功能部件原理图

图 5-6 中 A_0～A_7 和 B_0～B_7 作为两个 8 位操作数的输入管脚，Z_0～Z_7 为算术/逻辑运算 8 位数据输出管脚。CPSA 和 CPSB 为脉冲型控制信号输入管脚，分别控制完成接收 SA 和 SB 暂存器输入管脚（即：A 端和 B 端）的数据。ALU-BUS 控制信号，完成将算术/逻辑运算结果通过三态门（74LS244）输出到总线的功能。C_n、M、S_0～S_3 为算术/逻辑运算控制信号。

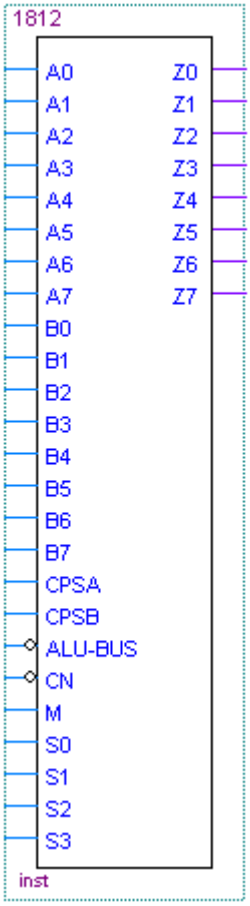


图 5-6 运算器功能部件芯片封装图

5.1.4 程序计数器的设计

程序计数器（PC）是为能够进行主机系统设计实验，控制程序的自动运行而设计的部件。该程序计数器可以提供 8 位的程序地址，具有自增 1 和接收转移地址的功能。程序计数器的设计原理图如图 5-7 所示，其相应的芯片封装图如图 5-8 所示。封装后的芯片名为 1612。程序计数器的详细设计图可参见附录 A 中的附图 A-5。

从程序计数器的设计原理图中可以看出，该程序计数器有两个输入控制信号 PC+1（对应于芯片封装图中的 PC1）和 CPPC（上升沿有效），一个输出控制信号 PC-BUS。在芯片封装图中，D_0～D_7 为程序计数器的八个数据输入管脚，可以接收八位数据。O_0～O_7 为八个数据输出管脚。PC1 与 CPPC 两个控制信号相配合，可以完成程序计数器的自动加“1”操作以及接收转移地址的操作。当 PC1 为高电位时，与时钟脉冲信号 CPPC 的上升沿相配合，完成程序计数器的自动加“1”操作。当 PC1 为低电位时，与时钟脉冲信号 CPPC 的上升沿相配合，将外部数据打入计数器（完成程序转移时，转移地址的打入操作）。相应的计数输入控制如表 5-1 所示。

从表 5-1 中可以看出，当 PC+1 为低电位，CPPC 管脚得到一个脉冲的上升沿时，程序计数器数据输入管脚上的数据则输入、存储到程序计数器中。当 PC+1 为高电位，CPPC 管

脚得到一个脉冲的上升沿时，程序计数器完成自动加 1 的计数操作，此时程序计数器的数据输入无效。若 PC+1 为高电位，CPPC 管脚无脉冲的上升沿时，程序计数器处于数据的保持状态，维持原数据不变。

程序计数器的数据在 PC-BUS 输出控制信号（低电位有效）的控制下，通过三态门（74LS244）输出。

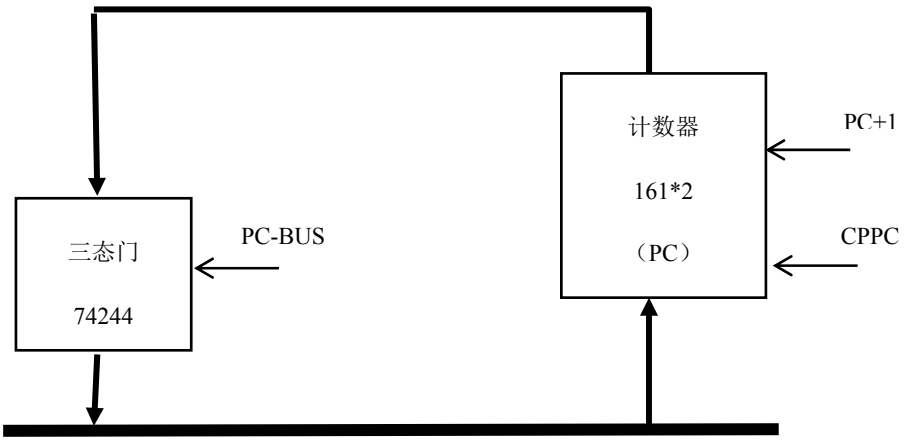


图 5-7 程序计数器设计原理图

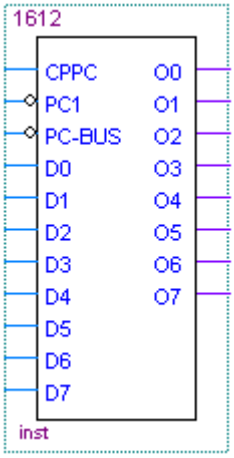


图 5-8 程序计数器芯片封装图

表 5-1 计数输入控制

PC+1	CPPC	D［0..7］	功能
L	↑	XXH	输入存储
H	↑	X	加 1（计数）
H	无↑	X	保持

5.1.5 暂存器的设计

总线暂存器（Bus Latch）是为了提供暂存总线上的数据而设计的寄存器。该暂存器可为寄存器间的数据传送提供临时数据存储的空间。总线暂存器的设计原理图如图 5-9 所示，其相应的芯片封装图如图 5-10 所示。总线暂存器的详细设计图可参见附录 A 中的附图 A-13。

图 5-10 中 CPC 输入控制管脚接收脉冲型控制信号，完成将暂存器数据输入管脚 D_0～D_7 上的数据输入及锁存的功能。C-BUS 控制信号（低电位有效），完成将暂存器内容通过三态门输出到数据输出管脚 O_0～O_7 端的功能。

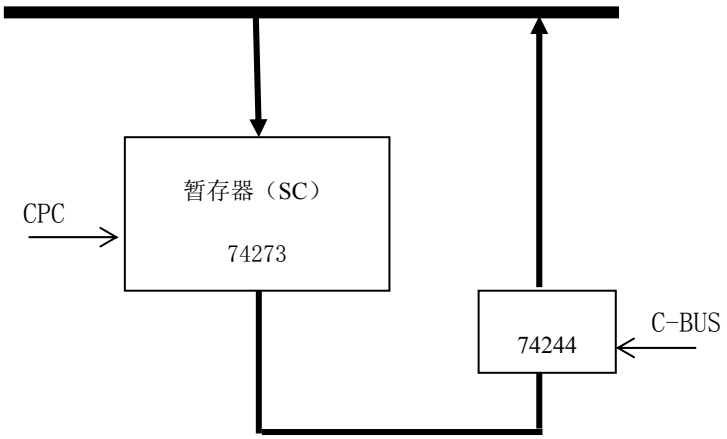


图 5-9 总线暂存器设计原理图

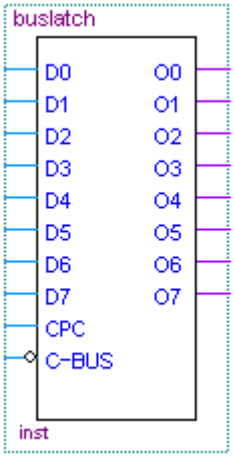


图 5-10 总线暂存器封装图

5.1.6 寄存器组的设计

通用寄存器组（General Registers）是为了提供进行主机系统设计时所需要的寄存器而设计的。通用寄存器组采用具有三态输出功能的两片 4 位片的 74LS670，通过并联组合，构成 4 字×8 位的寄存器组。该寄存器组可以暂存程序运行过程中所需要的各种数据。通用寄存器组的设计原理图如图 5-11 所示，其相应的芯片封装图如图 5-12 所示。封装后的芯片名为 6702。通用寄存器组的详细设计图可参见附录 A 中的附图 A-11。

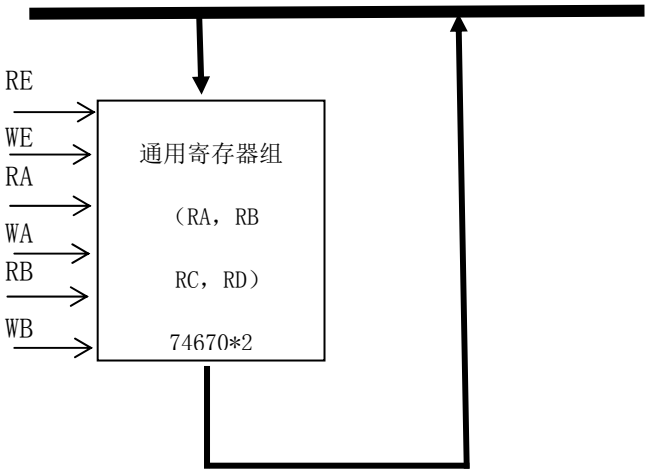


图 5-11 通用寄存器组设计原理图

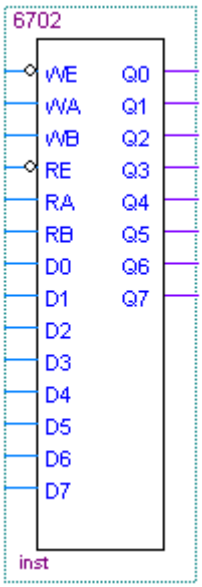


图 5-12 通用寄存器组芯片封装图

图 5-12 中 RA、RB 输出控制管脚接收需要读出信息的通用寄存器的地址，在 RE 读控制信号的作用下，输出相应寄存器中的数据到输出管脚 Q_0～Q_7 端。WA、WB 输入控制管脚接收需要写入信息的通用寄存器的地址，在 WE 写控制信号的作用下，把输入管脚 D_0～D_7 的输入数据写入到相应的寄存器中。其通用寄存器组的“写”与“读”功能控制方式可参见表 5-2 及表 5-3。

表 5-2 写功能控制方式

WE	WA	WB	选定寄存器
L	L	L	RA
L	L	H	RB
L	H	L	RC
L	H	H	RD
H	X	X	X

表 5-3 读功能控制方式

RE	RA	RB	选定寄存器
L	L	L	RA
L	L	H	RB
L	H	L	RC
L	H	H	RD
H	X	X	X

其中：L 为低电位，H 为高电位，X 为任意数据。

5.2 计算机组成原理课程实验

计算机组成原理课程是计算机专业本科生的一门非常重要的专业必修基础课，在多门专业课中占有举足轻重的地位。在这门课中，不仅讲授计算机的基本组成与工作原理的基础知识，还要训练一定的硬件实践动手能力。

从这门课的内容特点看，它属于工程性、技术性和实践性都非常强的一门课程。因此，在开展理论教学的同时，必须对实践教学环节给予足够的重视。

计算机组成原理实验是利用中大规模集成电路等器件，对组成计算机的各相关部件进行逻辑设计、连线及测试。在实验过程中，通过对各部件的实现原理进行逻辑设计，经过对器件的选择及连线、编译、模拟仿真等工作后，对于设计出的各个部件进行正确性测试。实验的参考总学时为 16～20 学时。

5.2.1 实验一：16 位并行进位运算器功能部件的设计与实现

一、实验目的

了解并行进位运算器的工作原理和过程，利用多个芯片采用扩展的方式设计出 16 位并行进位运算器功能部件，并封装调试。

二、实验内容

1. 分析并设计 16 位并行进位运算器的基本结构。
2. 选择芯片及若干元器件进行物理连接，完成 16 位并行进位运算器功能部件的设计，并实现部件的封装。
3. 对设计出的 16 位并行进位运算器功能部件进行测试，检查运算器功能部件是否能够正确完成数值运算的功能。

三、实验的实施

1. 设计出部件的逻辑原理图，画出部件的逻辑电路布线图。
2. 拟定测试数据及测试方法。
3. 检测模拟仿真测试结果的正确性。
4. 对设计出的部件进行封装，并写出封装后芯片的功能表。

5.2.2 实验二：带字位扩展存储器功能部件的设计与实现

一、实验目的

了解随机存储器的工作原理和过程,熟悉随机存储器的读写原理。根据存储器的工作原理，并且按照存储器字位扩展的基本原则完成存储器功能部件的设计，并实现器件封装，测试存储器的读写功能。

二、实验内容

1. 设计出存储器功能部件的基本结构。
2. 选择芯片及若干元器件进行物理连线，完成存储器功能部件的设计并实现部件的封装。
3. 对该部件进行模拟仿真测试，检查存储器功能部件的数据读写是否正确。

三、实验的实施

1. 设计出部件的逻辑原理图，画出部件的逻辑电路布线图。
2. 拟定测试数据及测试方法。
3. 检测模拟仿真测试结果的正确性。
4. 设计出的部件进行封装，并写出封装后芯片的功能表。

5.2.3 实验三：寄存器组及具有移位功能暂存器的设计与实现

一、实验目的

了解寄存器组及暂存器的工作原理和过程，设计出功能完善的寄存器组，并且使暂存器自身能完成逻辑左移、逻辑右移、算术左移、算术右移等数据移位功能，最终实现相应部件，并进行封装调试。

二、实验内容

1. 分析并设计寄存器组及具有移位功能暂存器的基本结构。
2. 选择芯片进行连接，完成寄存器组及具有移位功能暂存器的设计并实现相应部件的封装。
3. 对寄存器组进行测试，检查寄存器组是否能够正确完成数据的存储及读取。
4. 对暂存器进行测试，检查暂存器是否能够正确进行数据的暂存和读取，并检测数据移位功能是否正确。

三、实验的实施

1. 设计出部件的逻辑原理图，画出部件的逻辑电路布线图。
2. 拟定测试数据及测试方法。
3. 检测模拟仿真测试结果的正确性。
4. 对设计出的部件进行封装，并写出封装后芯片的功能表。

5.2.4 实验四：运算器、存储器功能部件与寄存器组之间数据传输方式的设计与实现

一、实验目的

了解运算器功能部件、存储器功能部件及寄存器组的工作原理和过程。实现寄存器组与运算器功能部件及存储器功能部件之间的连接测试，最终通过波形测试，验证设计的正确性。

二、实验内容

1. 设计运算器功能部件、存储器功能部件及寄存器组（可参照上述三个实验）。
2. 在上述基础上完成运算器功能部件与寄存器组及存储器功能部件间的连接，并且实现下面各种数据传输方式的设计：

(1) 某两个寄存器中的数据通过运算器功能部件完成相应运算，最终结果写入某一寄存器中；

(2) 寄存器与存储器中的数据通过运算器功能部件完成相应运算，最终结果写入某一寄存器中；

(3) 寄存器与存储器中的数据通过运算器功能部件完成相应运算，最终结果写入存储器某一存储单元中。

3. 进行波形测试，验证设计的正确性。

三、实验的实施

1. 设计出部件的逻辑原理图，画出部件的逻辑电路布线图。
2. 拟定测试数据及测试方法。
3. 检测模拟仿真测试结果的正确性。
4. 对重新设计的各个部件进行封装，并写出封装后芯片的功能表。

第 6 章 计算机主机系统设计

6.1 计算机主机系统设计的目标及要求

计算机组成原理课程是计算机专业的主干课程。计算机主机系统设计是在学完该课程后，利用所学的理论知识，真实地进行一台计算机主机系统的设计，并且在基于 EDA 平台的实验环境中，完成其模拟仿真测试。最终通过运行调试程序，以验证主机系统逻辑设计的正确性。通过进行主机系统底层电路的设计、实现，使理论知识与实践得以结合，从而进一步提高分析问题、解决问题以及硬件实践的能力。主机系统设计的参考总学时为 60～80 学时。

6.1.1 系统设计目标

计算机主机系统设计的设计目标是进行一台计算机主机系统的总体结构设计以及指令系统设计，在 EDA 平台上对设计出的主机系统进行模拟仿真测试。最终使设计出的主机系统在机器加电、产生频率稳定的主振信号后，能够自动地、连续地执行存储在主存储器中的程序。

6.1.2 系统设计要求

计算机主机系统设计要求进行主机系统的总体结构设计，设计出主机系统的数据通路、控制信号（微操作命令），并且完成该机的指令系统设计。

在此基础上，按照主机系统的数据通路及指令系统中各条指令的功能，拟定各条机器指令的指令流程及相应的微操作命令序列。在相应时序系统的配合下，根据该系统控制部件产生微操作命令的方式，设计出能够控制机器自动运行的控制部件。

在各部件设计、连接及模拟仿真测试完成之后，进行计算机主机系统的总体调试。通过运行存储在设计好的计算机主机系统的主存储器中的调试程序，查验程序运行时所保存的每条指令的运行结果波形图文件的内容，对主机系统逻辑设计的正确性进行验证。

6.2 计算机主机系统总体结构设计

6.2.1 微型计算机主机系统设计示例

在进行计算机主机系统设计过程中，首先根据设计需求，对主机系统的总体结构进行设计。在设计过程中，需要考虑组成一台主机系统所需设置的各大部件及其各大部件的组成；各部件之间采取什么样的连接方式，采用什么样的数据通路，即进行数据通路结构设计。在此基础上，对各部件的结构进行设计，并且根据信息在数据通路中的流动、读取及

存储的需要，设计出控制各部件之间进行信息传输所需要的各种控制信号（微操作命令）。

在本小节中，通过一台微型计算机主机系统总体结构的设计示例，进一步阐明主机系统总体结构的设计方法。

例如，设计一台总线宽度为 8 位的主机系统。总线采用双向单总线结构、为各部件之间进行信息传输提供分时复用的数据通路。各部件与总线（BUS）的连接方式如图 6-1 所示。

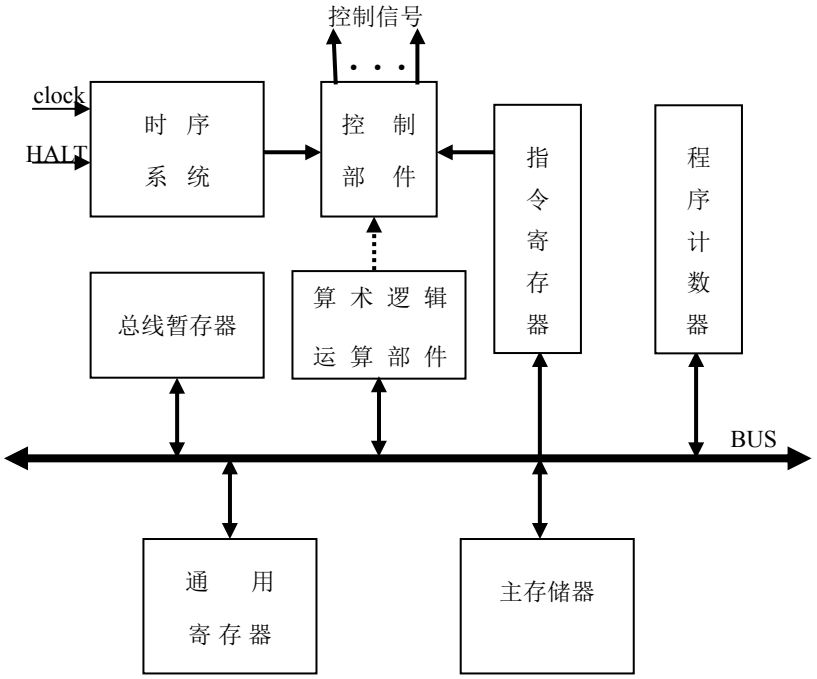


图 6-1 微型计算机主机系统结构图

在此主机系统中设有：算术逻辑运算部件、通用寄存器组、程序计数器、指令寄存器、总线暂存器、控制部件、时序系统以及主存储器等几大部件。在此例中，采用微程序控制方式构成控制部件。为测试运行结果的正确性，系统中设计了总线监视器作为监测数据在数据通路中的流动状况。主机系统中各部件的结构、数据通路结构以及控制各部件工作的控制信号（微操作命令）的设计如图 6-2 所示。

在进行控制信号的设计过程中，应在对各部件进行详细设计的基础上进行控制信号的设计。在进行各部件的详细设计时，首先需要了解各器件的功能及工作原理，然后，根据部件的功能要求选择合适的器件。例如，在对图 6-1 中的总线暂存器部件进行设计时，由于总线暂存器部件的功能为保存数据，并且应该能够根据需要，把所存数据发送到总线上或使该部件与总线之间保持高阻状态。根据部件的功能要求，在此可以选择器件 74273 或 74373 作为暂存器（相应器件的芯片外特性及功能表可参见附录 B）。若使用 74273 作为暂存器，由于该器件不具有三态输出功能，为了在需要数据输出时才向总线输出数据，所以需要在 74273 器件的输出端连接一个三态门（例如；74244）作为三态控制。在不需要信息输出时，应该使该部件呈现高阻态，以实现部件与总线的隔离。因此，对于控制总线暂存器工作的控制信号，至少需要设计两个。一个作为接收数据的控制信号，另一个作为三态

输出的控制信号，在此分别设计为 CPC（脉冲信号）和 C-BUS（电位信号）。当控制部件发出 CPC 脉冲控制信号时，暂存器则接收此时总线上的数据，并进行保存；当控制部件发出 C-BUS 电位控制信号时，则把暂存器中保存的数据送到总线上，供其它部件接收使用。对于控制信号的设计，可以按照此方法逐一进行设计。

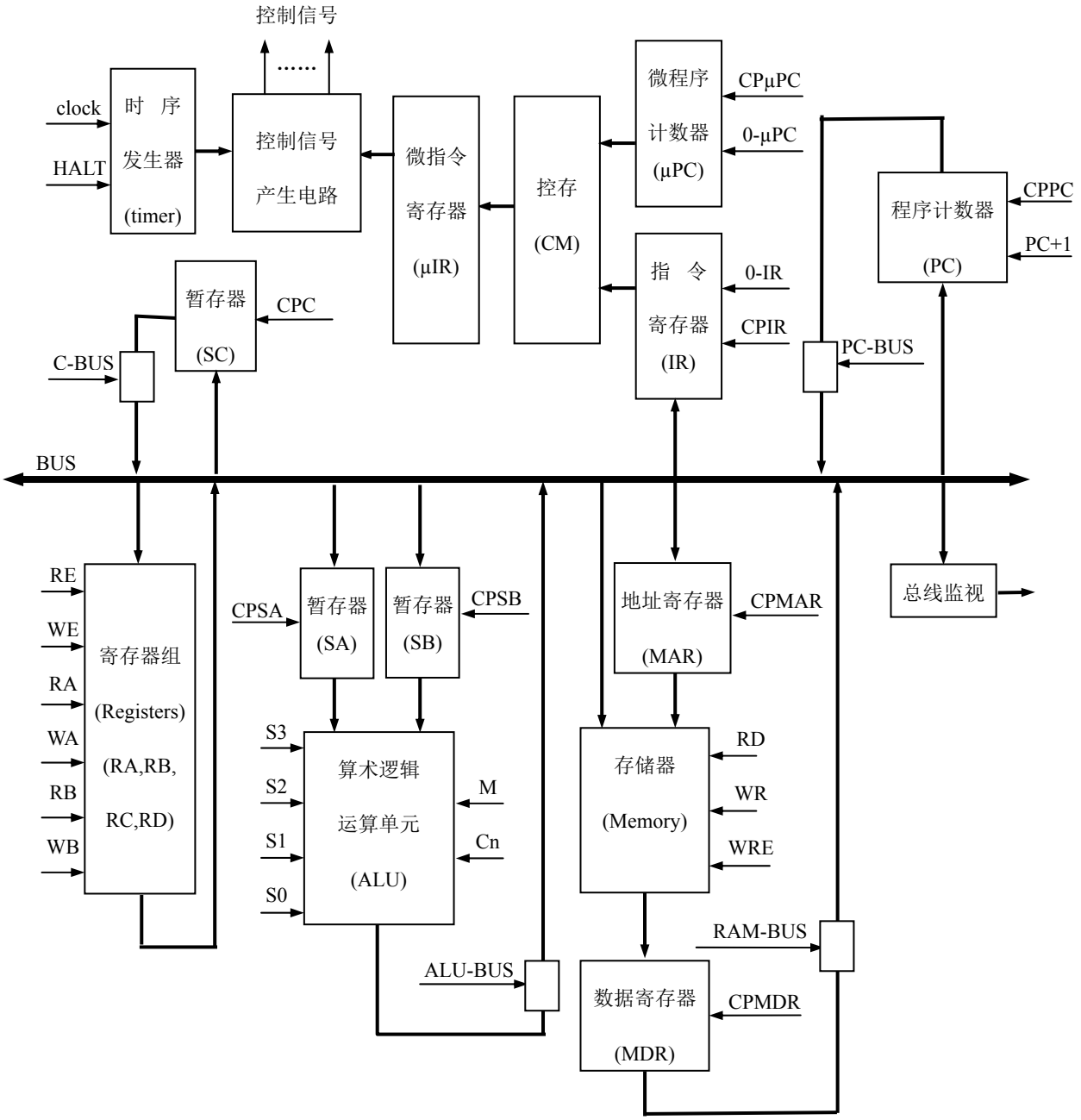


图 6-2 微型计算机主机系统数据通路结构图

6.2.2 主机系统总体结构设计要求

对于计算机主机系统总体结构设计，要求设计出一台具有 8 位或 16 位数据通路的主机系统。系统可以采用单总线结构，采用 8 位或 16 位通用寄存器。控制部件的设计可以采用微程序控制部件或组合逻辑控制部件的设计方式。主存储器可设计为单体或分为奇偶存储体的存储器，其编址方式可采用按字编址或按字节编址方式，存储器的字宽可按需要设计。

设计者可以根据自己掌握理论知识的深度及其各自的状况自行进行设计。在设计过程中也可以参考、使用第 5 章所介绍的各部件进行主机系统总体结构的设计。

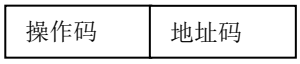
在设计过程中，要求画出主机系统总体结构设计图；根据总体结构中各部件的设置需求，选择相应的器件，并画出数据通路结构图；在此基础上，根据各部件的工作原理，设计出控制各部件工作所需要的控制信号。

6.3 指令系统设计

在进行主机系统总体结构设计的同时，需要对该机的指令系统进行设计。指令系统指的是一台计算机中所有机器指令（简称：指令）的集合。指令系统的功能和格式的设计与机器的总体结构的设计有着密切的关系。因此，应该把指令系统的设计与机器硬件结构的设计相互配合进行，以设计出效率高、性能强的机器。

6.3.1 指令格式设计

一条机器指令的格式，通常由操作码和地址码两部分组成。指令格式的基本形式为：



在进行指令格式设计时，一般要考虑操作码结构的设计、地址码结构的设计以及一条指令的长度。对于指令长度而言，为了存取方便，一条指令可以设置为单字节、双字节等一个字节至多个字节的不等长度。指令格式的设计可以参考、采用类似于 8086/8088 指令系统的可变长字节的指令格式。

6.3.2 操作码结构设计

指令中的操作码表示该条指令应该进行何种性质的操作，是编程者指示机器进行相应操作的命令。

操作码是由若干位二进制编码所组成。每个操作码都对应一条机器指令，命令机器执行相应的操作。操作码的位数决定着指令系统中完成不同操作的指令的条数。如果某机器指令格式中操作码的位数为 n 位二进制数，则该机器的指令系统最多可以设置 2^n 条指令。例如：需要设计 16 条指令，则操作码字段可以设计为 4 位，即 4 位操作码可以设计有 16 条不同操作的指令。

在设计操作码结构时，应该根据指令的长度、所需设置指令的条数等需求综合考虑并进行设计。操作码的结构可以设计成固定长度的操作码，也可以根据需要设计成可变长度

的操作码。总之，指令系统中所有指令都需要设有相应的操作码与之对应，同时还应考虑尽量降低操作码的冗余量。

在设计操作码结构时需要特别注意的是，如果对于不同寻址方式的同一种指令采用不同的操作码的设计方法时，则需要为具有不同寻址方式的同一种指令设置不同的操作码。例如：对于只有 4 位二进制数的操作码结构的“加法”指令而言，若两个相加的操作数都在通用寄存器中，设置此条“加法”指令的操作码为二进制数表示的 1001；若一个操作数在通用寄存器中，另一个操作数在主存储器中，则可设置此条“加法”指令的操作码为二进制数表示的 1010。由此可见，具有相同“加法”功能的指令，可能会占据多个操作码的码点。因此，在设计操作码的位数时，应该考虑指令系统中设置的各种指令以及每种指令所占据操作码的码点数，由各种指令占有的所有码点数来决定应设置的操作码的位数。
在设计机器的指令系统时，至少应包含传送类指令、算术/逻辑运算类指令、转移类指令、停机指令等相应指令。

6.3.3 地址码结构设计

指令中的地址码表示该条指令所处理的操作数在机器中的存放位置。地址码结构的设计主要考虑一条指令中操作数应该设置几个、每个操作数在机器中怎样存放以及用多少位二进制数表示每个操作数存放的地址。

指令中的操作数可以存放在通用寄存器中，也可以存放在主存储器中。当存放在主存储器中时又可分为存放在指令中（指令的地址码字段）或者存放在相应的主存单元两种情况。操作数是表示指令进行运算、操作的对象。但对于转移类指令而言，操作数则是程序转移的目的地，是机器即将执行的下一条指令所在的主存地址。在程序顺序执行时，由程序计数器中不断增量的内容控制着机器到主存储器中顺序取出指令去执行；在程序转移时，则把指令的地址码字段提供的操作数作为程序计数器的内容，按照此内容到主存储器去取指令供机器执行。

在进行指令的地址码结构设计的过程中，应根据要实现的指令功能，设计相应的地址码字段。例如，对于“加法”指令，可以设计成具有二地址的指令；对于“停机”指令，可以设计为零地址指令。对于指令的地址码字段的设计而言，可以根据指令的不同功能及所需操作数个数的不同设计出具有二地址、一地址及零地址的指令。在设计中还应考虑地址码结构中为表示每个操作数地址所需占据的二进制位数，与此同时，需要进行寻址方式的设计。寻址方式表示指令的地址码字段是怎样指定指令中所需要的操作数所存放的位置的，即：寻找操作数的方式。

在指令系统设计中，最基本、常用的寻址方式有立即寻址方式、寄存器寻址方式、寄存器间接寻址方式、存储器（直接）寻址方式、变址寻址方式、基址变址寻址方式及相对寻址方式等。设计者可以根据指令系统中描述各条指令中的操作数的存放位置而进行相应的寻址方式的设计。

例如，对于一条具有双操作数、指令长度为单字节的指令，其指令格式可以按照图 6-3 所示的格式进行设计。

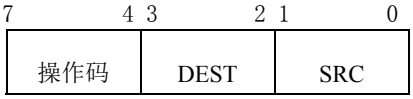


图 6-3 单字节指令格式图

在图 6-3 中，DEST 为目的操作数地址字段，SRC 为源操作数地址字段。

6.3.4 指令助记符与机器指令代码

指令中的操作码及地址码是由若干位二进制数编码而成的。在进行指令系统设计时，为了方便人们的记忆、阅读及编程，一般对机器指令中用二进制数表示的操作码和部分地址码用相应的若干个英文字母表示，通常称此为指令助记符。

例如：对于一条指令格式如图 6-3 所示的两个通用寄存器内容进行相加操作、其二进制代码为“10010001”的机器指令而言，可以用指令助记符“ADD AX，BX”来表示。该指令的功能是把通用寄存器 AX 的内容与通用寄存器 BX 的内容相加，其结果存入通用寄存器 AX 中。在这条加法指令中，用英文字母 ADD 作为指令中操作码的助记符，代表用二进制编码的操作码“1001”；用英文字母 AX 作为指令中目的操作数地址的助记符，代表目的操作数地址（例如：0 号通用寄存器）的二进制数编码“00”；用英文字母 BX 作为指令中源操作数地址的助记符，代表源操作数地址（例如：1 号通用寄存器）的二进制数编码“01”。

在进行指令系统设计过程中，要求设计出机器指令代码以及与之相对应的指令助记符。与此同时，需要对指令格式进行详细设计、说明。首先需要确定各类指令的长度、操作码及操作数分别占据指令中哪些二进制位。并且进一步确定操作码及地址码（相应寻址方式）分别采用什么样的助记符表示。

例如，具有四位操作码（位于指令最高 4 位）的指令操作码助记符的相应设计如表 6-1 所示。对于单字节指令格式中（指令格式可参见图 6-3）具有两个地址码字段（分别位于指令的第 2、3 位及第 0、1 位）的指令地址码助记符（相对于寄存器直接寻址方式）的相应设计如表 6-2 所示。

表 6-1 指令操作码助记符设计示例

指令操作码助记符	机器指令代码	指令功能
HALT	0000	停机
ADD	1001	加法（寄存器与寄存器内容相加）
ADD	1010	加法（寄存器与存储器内容相加）
MOV	0001	数据传送（寄存器到寄存器的传送）
MOV	0010	数据传送（立即数到寄存器的传送）

表 6-2 指令地址码助记符设计示例（相对于寄存器直接寻址方式）

指令地址码助记符	机器指令代码	说明
AX	00	0 号通用寄存器作为操作数地址
BX	01	1 号通用寄存器作为操作数地址
CX	10	2 号通用寄存器作为操作数地址
DX	11	3 号通用寄存器作为操作数地址

综上所述，指令助记符是机器设计者为人们提供的便于编程的一组符号。由于机器只能识别“0”、“1”这样的二进制代码，在机器运行时，必须为机器输入用“0”、“1”

所表示的机器指令代码。因此，在编写程序时，应该直接用机器指令代码编程，用指令助记符作为注解；或者编写一个用于该指令系统的汇编程序，编程时使用指令助记符编程，通过运行自编的汇编程序，把用指令助记符编写的程序转换成用二进制表示的机器指令程序，让机器运行此程序，以测试机器设计的正确性。

6.4 控制部件设计原理

在计算机主机系统设计过程中，在总体结构、指令系统及各部件结构详细设计完成之后，应开始进行 CPU 中最核心的部件——控制部件的设计。控制部件是控制计算机正确运行的关键部件。计算机在加电开机后，则由控制部件发出的控制命令控制着计算机有条不紊地、一条一条地执行存储在主存储器中的程序。

控制部件的设计方式可以分为微程序控制的设计方式及组合逻辑控制的设计方式。按照不同设计方式设计出的控制部件相应地称作微程序控制部件及组合逻辑控制部件（硬连逻辑控制部件）。微程序控制部件具有规整性及设计效率高、可修改性及可扩充性强等特点，已广泛地应用到各种规模的计算机系统结构中。组合逻辑控制部件在设计过程中由于面向的是机器的高速度及使用尽量少的元器件，因此具有优化的电路结构及实现速度快等特点，目前该种结构的控制部件常在 RISC 及一些超高速计算机系统结构中被采用。

6.4.1 微程序控制部件设计原理

微程序控制部件是利用微程序设计技术及存储逻辑方式来实现产生控制信号的控制部件。

在微程序控制部件中，其核心部件是控制存储器（简称：控存或 CM）。控制存储器的功能是存放指令系统中各条机器指令所对应的一段一段相应的微程序。控制存储器中存放的每一段微程序是由一条一条的微指令构成。每段微程序完成一条机器指令或取指令等某些特定的功能。每条微指令是由控制机器运行的每个微命令组合而成。

控制存储器一般是由只读存储器（ROM、EPROM 等）构成。由微程序计数器（μPC）作为控存的地址寄存器，按照其中存放的微指令地址（微地址）到控存中取出微指令，把其存放到微指令寄存器（μIR）中。通过把存放在微指令寄存器（μIR）中的每条微指令中的各二进制位的编码与时序信号相配合，形成控制信号（微操作命令），由此来控制机器的运行。

在进行微程序控制部件的设计过程中，对应于指令系统中每条（每种）机器指令的微程序设计以及对微指令中每个微命令的时序分配是微程序控制部件的设计重点。

6.4.2 组合逻辑控制部件设计原理

组合逻辑控制部件是利用组合逻辑电路（或称：硬连逻辑线路）来实现产生控制信号的控制部件。

组合逻辑控制部件的设计原理是根据存放在指令寄存器（IR）中所要执行的指令的不同功能，经过硬连逻辑线路的译码解释，在时序系统所产生的周期、节拍、脉冲等时序信号的配合下，产生控制机器运行的控制信号（微操作命令），由此控制着机器自动地、有条不紊地执行从主存储器中读出的一条一条的机器指令。

在进行组合逻辑控制部件的设计过程中，时序系统的设计、指令流程的设计、微命令

序列的拟定、为各个微命令分配时序（列出微操作时间表）、列出各微操作命令（控制信号）的逻辑表达式及对其进行化简、优化，并用组合逻辑电路对各微操作命令的逻辑表达式加以实现。上述内容是组合逻辑控制部件的设计重点。

6.5 微程序控制部件的设计与调试

在设计微程序控制部件过程中，首先需要根据总体结构、各部件详细结构及数据通路中控制信号（微操作命令）的设计状况，进行微指令格式的设计，并且对微指令各位（相应的微命令）分配相应的时序，以此构成控制机器自动运行的控制信号。在此基础上，对指令系统中各条机器指令进行相应的微程序设计，并把设计好的微程序写入控制存储器中（把微程序进行固化）。机器在运行程序的过程中，每执行一条机器指令，则到控存中读出一段微程序执行，以此完成存储在主存储器中由机器指令组成的程序的执行。

6.5.1 微指令与控制信号的设计

1．微指令的设计

一条微指令其功能是给出完成机器中某种操作所需要的全部微命令，并且控制产生后继微指令地址（即：下一条即将执行的微指令地址）。在进行微指令结构设计时，可以把微指令的结构分为两部分：一部分设计为微操作控制字段，另一部分设计为顺序控制字段（如图 6-4 所示）。微操作控制字段主要由控制机器执行指令功能的各个微命令组合而成。顺序控制字段主要由控制产生下一条即将执行的微指令地址的微命令组成。微指令设计主要对微指令结构中的微操作控制字段及顺序控制字段进行设计。

图 6-4 微指令结构图

下面分别介绍微指令结构中微操作控制字段及顺序控制字段的设计方式。

（1）微操作控制字段的设计

微指令结构中微操作控制字段的设计，主要通过分析机器数据通路结构中各种控制信号在机器运行过程中可产生的并行性，研究对各个微命令进行分段组合的编码方法。

微命令是构成控制机器运行的控制命令的最小单位。微命令在时序信号的配合下，构成控制机器运行的控制信号（微操作命令）。控制部件通过控制总线在相应的时刻向机器的相关部件发出相应的控制信号，以控制机器进行相应的操作。机器通过执行一个个微操作而完成相应的微指令的功能。微操作是由微指令中的微命令配以相应的时序信号而形成的控制信号所控制实现的机器中的最基本的操作。机器的运行就是由一条条微指令所发出的微命令在时序信号的配合下产生一个个控制信号，由这些控制信号指挥着机器进行一个个微操作而实现的。

在进行微指令中微操作控制字段的设计时，对各个微命令常常采用的编码方法有直接控制编码法（不译码法或直接表示法）、分段直接编译法（分段直接译码编码法或显示编码法）及混合编码法（混合表示法）。

直接控制编码法的设计思想是使微指令的微操作控制字段中的每一位代表一种微命令。此编码方法具有直观、实现简单、操作并行性好及执行速度快等特点。对于数据通路结构较简单、微命令总的数量不是很多的情况，可以采用直接控制编码法（不译码法）作

为微指令中微操作控制字段的编码方法。在利用此方法设计微指令时，使一条微指令中微操作控制字段的每一位（即：每个微命令）与机器设置的各个微操作命令（控制信号）一一对应。在进行微程序设计时，根据每条微指令的功能，决定需要发出何种微命令；由此将此条微指令中表示该微命令的对应位设置成“1”即可，其余的位设置成“0”。在微程序执行过程中，把每条微指令中表示各个微命令被选用或不选用的（“1”或“0”）信号，与时序系统产生的电位、脉冲信号相配合，形成机器的数据通路中的各个控制信号（微操作命令），以控制整个机器的运行。

分段直接编译法是根据数据通路中可实现的各微操作的并行性状况，把在同一个微指令周期中不可能同时出现的一组具有相斥性的微命令（例如：控制各部件内容送总线的各个微命令）组合成一个小字段，通过字段译码器对各个小字段进行译码，以产生一个个相应的微命令。在采用这种方法对微命令进行设计时，通常将控制同一部件的操作或同类的操作中具有相斥性的微命令分配到同一字段中，对其独立进行编码；将在一个微指令周期中可以同时出现的（即可以并行发出的）具有相容性的微命令分配到不同的字段中。这样，在执行一条微指令时可以并行执行若干个微命令，不仅提高了机器操作的并行性，还可以相应地缩短一条微指令的长度。

混合编码法通常是指把几种编码方法相结合，在一条微指令的微操作控制字段，可以有多种编码方法。这种设计方法，即可缩短一条微指令的字长，又可兼顾到微程序的执行速度。

（2）顺序控制字段的设计

微指令中顺序控制字段的设计是为了解决微程序在执行过程中如何产生后继微指令地址的问题。

对于微程序的执行而言，机器设计者需要考虑两种情况。第一种情况是在机器加电开机后，如何执行存于控制存储器中的第一条应该执行的微指令。计算机在执行每条机器指令时，首先均需要进行从主存把相应的机器指令取出存放到 CPU 中的指令寄存器中的操作，因此可以将该“取指令”的操作编制成一段公共的“取机器指令”微程序。通常将该段微程序存放到控制存储器（简称：控存或 CM）的 0 号单元（或固定某单元）开始的一段控存空间中。机器加电开机后，在置程序计数器（PC）为第一条应执行的机器指令的主存地址的同时，还应自动把微程序计数器（μPC）的内容置为存放于控存中的“取机器指令”微程序的第一条微指令的地址（置 μPC 为全“0”或某特定值）。

第二个需要考虑的问题是机器在运行过程中如何控制产生后继微指令地址，即顺序控制字段的设计方式。

顺序控制字段的设计方式可以采用增量方式（计数器方式）、断定方式或二者相结合的方式。顺序控制字段的编码方法可以采用分段直接编译法或混合编码法。在增量方式设计中，通过设置转移方式字段控制微程序计数器（μPC）的增量（加“1”）操作或实现微程序的转移操作（即把转移地址置入微程序计数器中）。在断定方式设计中，通过设置测试字段控制产生微程序计数器的值，即根据机器相关状态的测试结果，而确定微程序计数器中的微地址，从而实现微程序的顺序或转移执行。

①微程序入口地址的形成方式

在顺序控制字段的设计过程中，首先需要考虑每条机器指令相对应的每段微程序（微主程序）入口地址的确定方法。在用微程序控制方式组成控制部件的机器中，指令系统中的每一条机器指令都由一段微程序（其中可包含一段微主程序及若干段微子程序）来实现其功能。当机器执行完公共的“取机器指令”微程序，即把一条机器指令从主存取出存放到指令寄存器（IR）之后，则可以根据指令寄存器中机器指令的操作码来形成该机器指令所对应的相应微程序的入口地址。例如，可以把指令的操作码作为相应微程序入口地址的高几位或低几位存入微程序计数器（μPC）中，微程序计数器中的其余位可以设置成固定

值。可以由顺序控制字段产生相应的控制信号来控制微程序入口地址的设置（即 μPC 内容的设置）。

②后继微地址的形成方式

计算机在“取机器指令”微程序执行完，转移到某条机器指令所对应的微程序的入口地址后，则开始执行本段微程序，以此来完成该条机器指令的功能。

在微程序执行过程中，主要具有顺序执行和转移执行两种状态。因此，顺序控制字段可以通过设置转移方式字段控制微程序的执行顺序。通常在微程序控制部件中，设置一个微程序计数器（μPC），使其具有自加“1”及接收转移微地址的功能。通过顺序控制字段所发出的微命令来控制微程序的顺序执行或转移执行。转移微地址的形成可以由微指令中某些位产生、设置，还可以根据机器的执行状态而控制相应转移微地址的形成。

2．时序系统的设计

对于采用微程序控制方式设计的机器，机器的运行是以一条条微指令的执行为基础的。因此，机器的时序系统的设计可以相对简单些。机器的时序可以设计为统一的微指令周期（微周期）。在每一个微指令周期中，机器执行一条微指令。一个微周期中可以设置若干个节拍、脉冲，以控制机器执行一条微指令所发出的各个微命令。

例如，可以设计一个时序发生器用于循环产生一个微周期所需要的一组等间隔或不等间隔的节拍、脉冲序列。在机器加电提供频率稳定的时钟脉冲信号后，时序发生器则可以循环往复地产生一组节拍、脉冲信号，作为每条微指令执行的时序信号（一种时序发生器的示例可参见附录 A-1）。

机器执行一条微指令的过程一般分为两个时间段：到控存中读取微指令时间段和执行该条微指令中各个微命令的时间段。读取微指令时间一般比较短，执行微指令时间相对比较长。

例如，可以把附录 A-1 中的时序发生器（timer）所产生的时序信号 m5～m4（m5、m6、m7、m1、m2、m3、m4）作为一个微周期，把每个微周期最后一段时间（如：m3 时间段）作为取微指令的时间段，其余时间段作为执行一条微指令的时间段。

3．控制信号的设计

在完成微指令设计及时序系统设计之后，在指令流程级，参照机器的数据通路结构，分析微指令中各个微命令与时序系统提供的时序信号之间的相关性，为每个微命令分配相应的时序信号，以此来产生最终控制机器自动运行的控制信号（微操作命令）。

例如，在机器执行取指令时，是按照程序计数器（PC）中的指令地址到主存中读取指令。因此，需要完成把 PC 内容送主存地址寄存器（MAR）的操作（控制信号可参见图 6-2）。假设:微指令中的第 3 位（在此用μIR_3代表微指令寄存器中的第 3 位）表示把位于总线上的访问主存的地址输入到 MAR 的微命令（相对于 CPMAR）；微指令中的第 5 位（μIR_5）表示把 PC 内容送总线的微命令（相对于 PC-BUS）。当某条微指令的第 3 位（μIR_3）和第 5 位（μIR_5）同时为“1”时，在时序信号的配合下，则可以完成把 PC 内容输入到 MAR 中的操作（即：PC→MAR）。在此例中，由于 CPMAR 微操作命令是脉冲型控制信号，上升沿有效，在向 MAR 打入数据时，MAR 的数据输入端（即：总线）上的数据应当是有效数据（即：指令所在的主存地址）。由于只有在 PC-BUS 电位信号发出期间，总线上的数据才有效，因此在设计产生这两个微操作命令时，应当考虑 PC-BUS 及 CPMAR 微操作命令是在同一时间段内有效，且 CPMAR 的上升沿应当包含在 PC-BUS 为高电位的信号中。由此可以考虑使μIR_5(相对于 PC-BUS)与时序信号 M_1(电位信号)相配合，产生 PC-BUS 的控制信号（微操作命令）；使μIR_3(相对于 CPMAR)与时序信号 M_2(脉冲信号)

相配合，产生 CPMAR 的控制信号（微操作命令）。这两个控制信号的时序关系可参见图 6-5，其时序信号可参见附录 A-1。

根据上述分析可以得出产生下面两个控制信号（微操作命令）的逻辑表达式为：

$$\text{PC-BUS} = \mu IR_5 \cdot M_1$$
$$\text{CPMAR} = \mu IR_3 \cdot M_2$$

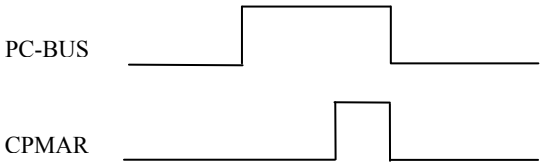


图 6-5 电位脉冲配合时序图

若所选择的三态门输出控制为低电位有效时（即：控制信号 PC-BUS 为低电位有效时），则控制信号 $\overline{\text{PC-BUS}}$ 的逻辑表达式为：

$$\overline{\text{PC-BUS}} = \mu IR_5 \cdot M_1$$

参照上述方法，可以设计、定序出完成各种控制功能的控制信号（微操作命令），以此来控制整个机器自动地、连续地执行存储在主存中的程序，直至遇到停机指令，则停止机器的运行。

6.5.2 微程序设计

在设计好机器指令系统及微指令结构之后，参照主机系统的数据通路结构，则可以开始进行微程序设计。微程序的设计过程可以分为拟定指令流程、编制微程序以及把编写好的微程序写入控制存储器中等几个步骤。

在进行微程序设计过程中，首先对于指令系统中的每条（每种）机器指令，均需要设计出相应的指令流程。指令流程主要用来指定机器完成相应指令功能的各种操作顺序。描述在一条机器指令的执行过程中，数据在寄存器（或暂存器）与寄存器（或暂存器）之间（可经过 ALU）、寄存器（或暂存器）与主存储器之间的传送过程及传送序列。指令流程的拟定是编写微程序的基础。

在指令流程拟定好之后，即可以开始按照指令流程进行微程序的编制。在编写微程序时，可以把指令流程中所描述的寄存器（暂存器）间或寄存器（暂存器）与主存储器之间的一次数据传送的过程作为一条微指令进行编写。每条微指令的编写可以用微命令助记符表示（在此微命令助记符可与微操作命令助记符相同），然后再转换成用“0”、“1”机器可识别的数字所表示的微指令（可用二进制、十六进制等数字表示微指令）；或者参照微指令结构直接用二进制或十六进制等数字编写微指令，用微命令助记符作为注解，以便于阅读。

在设计、编制微程序的过程中，对于执行每条机器指令都需要的“取机器指令”的功能，可以单独设计出一段“取机器指令”（简称：“取指令”）公共微程序，为各条机器指令的取指令功能的实现而共享。在“取指令”这段公共微程序中的最后一条微指令，应该设计成按照指令寄存器（IR）中的机器指令操作码进行相应的微程序转移的微指令。通过这条微指令的执行，使机器转移到由指令操作码所决定的相应的微程序段去执行，即把微程序计数器（μPC）中置入该条机器指令所对应的微程序所在控存的入口地址。

在进行各条机器指令所对应的各段微程序（可包括一段微主程序和若干段微子程序）

的设计过程中，当执行完每段微程序（完成一条机器指令功能）后，应该转移到“取指令”公共微程序去执行，进行取下一条机器指令的操作。因此，各段微程序的最后一条微指令，应该设计为无条件转移的微指令。由该条微指令完成把微程序计数器（μPC）的内容设置为“取指令”公共微程序的入口地址（例如：置μPC 为全“0”或某特定值）。

在微程序全部编制完成后，需要把各段微程序写入控制存储器中。在 MAX+plusⅡ或 QuartusⅡ的芯片库中，可以选择 LPM_ROM 作为控存（CM）来存放微程序。采用硬件描述语言 AHDL(ALTERA HARDWARE DESCRIPTION LANGUAGE)，把微程序以写入扩展名为.mif 文件的方式输入到控存中（控存 ROM 的.mif 文件的写入方法可参见第 2 章）。机器在运行程序的过程中，从控存中读出一条条微指令，通过对微程序的执行来完成机器指令的执行。

6.5.3 微程序控制部件的调试与封装

由于微程序控制部件是 CPU、主机系统以至整个机器的核心部件，因此在微程序控制部件设计完成后，应对其进行调试，以确认该部件设计的正确性。

微程序控制部件的设计过程，不仅包括微指令设计、控制信号设计、微程序设计，还需要对控制信号的产生电路进行设计连线、对微程序控制部件的硬件电路进行设计连线，最终实现一个完整的微程序控制部件。

在进行微程序控制部件调试时，可以按照下述方法调试：

首先，在时序信号的配合下，对微程序计数器（μPC）中给定的微地址，测试是否可以正确地从控存（CM）中读出相应的微指令，并将其存入微指令寄存器（μIR）中。可以通过查验微指令寄存器中的内容来检查读出内容正确与否。

在控存内容能够正确读出后，则可以开始进行下一步的调试。根据每个控制信号的逻辑表达式，在时序信号的配合下，对控制信号产生电路进行调试。在调试过程中，根据产生每个微操作命令的时间及条件，测试控制信号产生电路能否发出正确的控制信号。为调试方便，可以将控制信号产生电路封装成一个芯片（芯片封装方法详见第 2 章），并且把微程序计数器、控制存储器及微指令寄存器封装成一个芯片，在时序信号的控制下，进行微程序控制部件的整体调试。在调试过程中若发现错误，应对已封装好的芯片重新进行修改、更新、编译、仿真测试，并重新封装。

当微程序控制部件的设计正确无误时，可以将微程序控制部件封装成一个芯片，为以后主机系统的调试提供方便。

6.6 组合逻辑控制部件的设计与调试

在进行组合逻辑控制部件的设计过程中，首先应该对时序系统进行设计。然后，根据总体结构、各部件的结构、数据通路结构及指令系统进行指令流程的设计。在此基础上拟定微操作时间表，综归纳出全部微操作命令（控制信号）的逻辑表达式，并对其进行化简、优化。最后，用组合逻辑电路实现所有的控制信号，即设计实现微操作命令发生器（微操作信号发生器或控制信号发生器）。在整个控制部件设计完成后，对该部件进行调试，以验证微操作命令发生器设计的正确性。

6.6.1 时序系统的设计

对于采用组合逻辑控制方式设计的机器，通常采用三级时序作为同步控制的组合逻辑控制部件的定时信号。三级时序系统是把控制机器工作的时序信号（即：控制执行一条机器指令的时序信号）划分为机器周期（CPU 周期或工作周期）、节拍（时钟周期）及工作脉冲三种时序。机器在三级时序信号的控制下，循环往复地执行一条条机器指令。

通常把一条机器指令的执行过程分为若干个机器周期，每个机器周期可以包含若干个节拍，每个节拍可以包含一个或多个工作脉冲信号，以此控制着每条机器指令的执行。因此，在设计时序系统时，应分别设计出机器周期发生器、节拍发生器及工作脉冲发生器。

机器周期发生器也可以称作机器周期状态发生器。由此发生器决定着机器当前正运行在哪一个机器周期。例如，一条指令的执行过程共需要四个机器周期（取指周期、取源操作数周期、取目的操作数周期及执行周期），当机器周期发生器产生的机器周期状态是取指周期，则表明机器目前正处于到主存储器中取指令的工作状态。机器周期发生器可以由多个触发器组成，每个触发器代表一种机器周期状态。任何时刻有且仅有一个触发器的状态为“1”。机器在加电开机时以及每条指令执行结束后，机器周期发生器应该能够产生“取指周期”信号，即置“1”取指周期状态位（置“1”取指周期状态触发器），使机器运行到“取机器指令”的工作周期。在每个机器周期结束时，应该能够根据机器当前运行指令的操作码、寻址方式、工作周期、节拍及工作脉冲等状态，置“1”下一个应进入的机器周期状态触发器。根据指令功能的不同，每条指令的执行时间长短各异。例如，执行算术逻辑运算指令需要四个机器周期（取指周期、取源操作数周期、取目的操作数周期及执行周期），而执行转移指令可能只用两个机器周期（取指周期及执行周期）即可。因此在设计机器周期发生器时，应该把机器指令操作码字段、地址码字段（寻址方式）、机器周期状态、节拍、工作脉冲等信号作为机器周期状态发生器的输入。由输入的状态决定机器应进入哪一个机器周期，即产生相应的机器周期状态（置“1”相应的机器周期状态触发器）。

节拍发生器（时钟周期发生器）的功能是产生机器工作所需要的节拍。根据指令执行的需要，节拍发生器不仅要求能够连续产生一个个节拍（例如：T0、T1、T2、T3 …），而且还应具备在任何时刻均可重新置回 T0 节拍，由此重新开始产生连续的一个个节拍的功能。由于每条指令的功能不同，每条指令在执行过程中，要求每个机器周期中所包含的节拍数可能不相同。在一个机器周期中的工作未完成时，则需要节拍发生器产生连续不断的节拍。而在一个机器周期内的工作完成时，则应使节拍发生器重新从 T0 节拍开始连续不断地产生一个新的机器周期所需要的节拍。由此可见，节拍发生器应具有根据需要随时进行初始化，即能够置回到起始节拍的输入控制端。当时间及条件满足时，则通过该控制端控制节拍发生器产生节拍的状态。在设计节拍发生器的过程中，应当参照各类指令在各个机器周期中所需要的最大节拍数进行节拍发生器逻辑电路的设计。

工作脉冲发生器通过把机器加电后由晶体振荡器产生的主振信号（时钟脉冲信号 CLK）进行分频，从而产生连续不断的工作脉冲信号。工作脉冲信号是指挥机器工作的节拍发生器、机器周期发生器的基准定时信号。工作脉冲发生器为机器稳定、协调地运行提供相应的工作脉冲信号。当遇到停机（HALT）信号时，工作脉冲发生器则停止产生工作脉冲信号，从而导致节拍发生器和机器周期发生器同时停止工作，最终使机器处于停止运行的停机状态。

6.6.2 指令流程与微操作时间表的设计

在设计控制部件的过程中，对指令系统中各类指令的指令流程的设计是一项非常重要

的工作。对于组合逻辑控制部件的设计，在拟定指令流程时，可以按照指令系统中各类指令的功能进行指令流程的设计。

指令流程的拟定是根据指令所需实现的功能，按照数据在寄存器、暂存器、主存储器（其间可经过 ALU）之间的传送过程及传送顺序，把每一次传送过程按照指令实现各个过程的步骤，分配到时序系统提供的相应的机器周期及节拍中。由此设计出指令系统中各类指令功能的实现过程（即：指令流程）。

在指令流程设计完成后，需要为指令流程的具体实现确定微操作时间表。微操作时间表的建立是把控制指令流程实现的各个微命令详细列出，为其分配、确定相应的时序，即把各个微命令与机器周期及节拍建立关系，并根据数据通路结构中控制信号的状况，确定各微命令是电位型微命令或脉冲型微命令。

6.6.3 微命令的逻辑综合与实现

在指令流程及微操作时间表拟定之后，即可以对各个微命令进行逻辑综合。对于分配时序后的微命令，在此称为微操作命令（控制信号）。

在进行微命令逻辑综合过程中，根据各微命令所处的各个微操作时间表的位置，列出各个微操作命令的逻辑表达式。微操作命令的逻辑表达式是由指令操作码的译码信号、指令地址码（寻址方式）的译码信号、ALU 标志寄存器的相应标志位、机器周期状态、节拍、时钟脉冲等信号作为其“与”、“或”项。通过对微操作命令的逻辑表达式进行综合、化简，构成最优的逻辑表达式。在对设计的正确性确保正确无误后，采用组合逻辑电路（硬连逻辑线路）对各个微操作命令逻辑表达式进行器件的选择及连线，从而实现组合逻辑控制部件中的核心部件——微操作命令发生器（微操作信号发生器或控制信号发生器）。

6.6.4 组合逻辑控制部件的调试与封装

在进行组合逻辑控制部件调试过程中，由于时序系统是控制该部件正确工作的指挥系统，因此首先需要对时序系统进行调试。然后再对微操作命令发生器进行调试。

在对时序系统进行调试的过程中，需要测试工作脉冲发生器在机器加电、输入稳定的时钟脉冲信号（CLK）后，是否能够产生连续不断的工作脉冲信号。并且，在停机命令的控制下，能够停止工作脉冲的产生。同时还需要对节拍发生器进行测试。测试该发生器能否连续产生一个机器周期中所需要的最大节拍数，并测试该发生器在不断产生连续节拍的过程中，能否根据需要把节拍置回初始状态，从初始节拍开始重新产生连续的节拍。

在此基础上，对机器周期发生器进行测试。按照不同指令的需求，测试能否根据需要在相应的时刻产生正确的机器周期状态。同时需要对时序系统产生的机器周期、节拍及工作脉冲的切换时刻在需要时能否达到同步进行测试，以确保时序信号的准确性。

在调试微操作命令发生器的过程中，可以利用调试正确的各级时序信号，对需要测试的各个微操作命令按需要设置相应的输入项进行测试。由于微操作命令发生器是组合逻辑控制部件以至整个机器的核心部件，因此应该对每个微操作命令、每个局部模块都进行细致调试，直至全部结果测试正确。

在调试完成后，可以把时序系统及微操作命令发生器（微操作信号发生器）分别封装成两个芯片，以供主机系统总体结构的逻辑电路设计、连线时使用。

6.7 计算机主机系统的调试

在进行计算机主机系统调试的过程中，建议采用各部件的局部调试与主机系统的整体调试相结合的调试方法，一步一步、自小而大、自简而繁、自下而上地对主机系统进行调试，最后，用设计出的各种类型的机器指令编出一段（或若干段）调试程序存入主存（采用 AHDL 硬件描述语言编程并写入主存），在机器加电开机后，使设计出的主机系统能够自动运行主存中的调试程序。在程序运行结束后，检测机器运行调试程序的时序模拟仿真输出波形文件，以确认各条机器指令运行的正确性。

6.7.1 取指令功能的调试

取指令功能是机器运行每条指令都需要的功能。取指令功能的调试主要测试机器能否按照程序计数器（PC）所给定的主存地址，到主存中把相应的机器指令取到指令寄存器（IR）中。同时验证程序计数器能否连续计数。在此可以按照局部测试法，重点关注程序计数器（PC）、指令寄存器（IR）和主存储器功能部件之间的连接以及与其相关的控制信号。在时序信号的控制下，验证机器能否按照程序计数器所给定的地址从主存中把指令正确地读取到指令寄存器中。

取指令功能的正确实现，是进行主机系统各部件调试及系统整体调试的基础。

6.7.2 通用寄存器组输入输出功能的调试

通用寄存器组输入输出功能调试主要测试能否把数据正确写入指定的寄存器中，或从指定的寄存器中读取信息。同时还可以对暂存器模块进行调试。

在调试过程中，可以采用传送指令进行测试。通过立即数送通用寄存器指令的测试，可以验正通用寄存器的写入是否正确。同时，也部分测试了主存储器与通用寄存器之间（主存到通用寄存器方向）数据通路的畅通性。通过通用寄存器内容送通用寄存器指令的测试，可以测试通用寄存器的读出及写入的正确性。由于在通用寄存器之间进行数据传送时，从通用寄存器中读出的数据需要用暂存器进行暂存，因此在测试该条指令的过程中，同时可以对暂存器的写入及读出状况进行正确性测试。通过这条指令的测试，可以测试各通用寄存器之间及通用寄存器与暂存器之间数据通路的畅通性。

通过通用寄存器组输入输出功能的成功测试，实际上也完成了指令寄存器中的指令操作码和地址码设计的部分正确性调试，同时也验证了控制部件发出的相关控制信号的正确性。

6.7.3 存储器功能部件的调试

存储器功能部件的调试主要测试能否把数据按指令中给定的主存地址正确写入主存储器中，以及能否从主存中按指令中给定的地址把数据正确读出。

在测试过程中，可以采用传送指令，把某通用寄存器中的数据写入某指定的主存单元，或把某主存单元中的数据读出送入某通用寄存器中。通过此部分的测试，不仅可以测试存储器功能部件写入、读出功能的正确性，而且也测试了通用寄存器组与主存储器功能部件之间数据通路及相应控制信号设计的正确性。

6.7.4 运算器功能部件的调试

运算器功能部件的调试主要测试能否按照算术逻辑运算指令给定的功能，使运算器功能部件给出正确的运算结果。

在调试过程中，可以采用算术运算或逻辑运算指令，完成对给定两个数据的运算功能的调试。对于参与运算的两个数据，可以都来自于通用寄存器，也可以一个来自于通用寄存器，一个来自于主存储器。运算的结果可以存入某个通用寄存器中，也可以存入指定的某个主存单元中。通过此部分的测试，可以检测通用寄存器组、存储器功能部件及运算器功能部件之间数据传送及数据运算的正确性。

6.7.5 指令系统功能及寻址方式正确性测试

在对各大部件及数据通路的调试完成后，则可以对指令系统的功能进行测试。

对指令系统的功能进行测试，不仅需要对每种指令的功能进行测试，为了确保机器设计的正确性，而且还需要对指令中所包含的各种寻址方式进行测试。对于每种指令而言，需要进行多条指令的测试。例如，对于数据传送指令（指令操作码助记符为“MOV”）而言，按照操作数的寻址方式的设置，需要测试多条包含不同寻址方式组合的“MOV”指令。在寻址方式的测试过程中，应当根据寻址方式设计方案的状况，对全部寻址方式或对各种指令中的不同寻址方式进行测试，以保证用于各种指令中的各种寻址方式实现的正确性。

在进行指令系统功能调试的过程中，可以按下述几大类别分类对机器指令逐一进行调试：

1. 调试停机指令。
2. 调试传送类指令。
3. 调试算术逻辑运算类指令。
4. 调试转移类指令。此类指令可以包含无条件转移及条件转移类指令，可对程序计数器（PC）能否正确接收总线上给定的数据（转移地址）进行测试。
5. 调试其它各类指令。

6.7.6 调试程序的编制及主机系统的调试

对各类指令调试完成后，编制调试程序对主机系统的整体运行进行调试。

调试程序要求包含指令系统中设计出的各种指令，而且应该包含各种寻址方式。程序中至少应有一条停机指令。调试程序的设计，在指令的安排上应当有一定的逻辑相关性。尽量减少重复指令的出现。把编写好的调试程序，以二进制或十六进制数等形式存入主存储器的初始化文件（即写入主存 RAM 的.mif 文件）中。对每条用数字表示的机器指令代码应采用英文进行注释。

在进行主机系统调试过程中，只允许输入唯一的机器主振信号（即：时钟脉冲信号 CLK）。时钟脉冲信号 CLK 的产生，相当于机器加电开机。在给定机器唯一的输入信号——时钟脉冲信号 CLK 后，使机器自动地、连续地运行存储在主存中的调试程序。在遇到停机指令后，则停止机器运行。机器运行结束后，检测机器运行调试程序的时序模拟仿真输出波形图，以确认各条机器指令运行的正确性。在进行主机系统调试时，要求按照此方法进行调试。

第 7 章 计算机组成原理课程实践报告要求

7.1 计算机组成原理实验报告要求

实验报告用 A4 纸打印，封面上要求有实验名称、班级、学号、姓名、同组人姓名及日期。实验报告在开始处要求有目录，最后可以对实验过程的收获、体会等进行总结。

实验报告要求包含以下内容：

1．给出设计出的各部件的总体结构框图。

2．给出设计出的各部件的逻辑电路图。

3．简述各部件的设计原理。

4．给出测试数据及测试结果的输出波形图。

5．给出各部件的芯片封装图及相应的功能表。

6．简述实验中遇到的问题及解决方法。

7.2 计算机主机系统设计报告要求

主机系统设计报告用 A4 纸打印，封面上要求有设计名称、班级、学号、姓名、同组人姓名及日期。设计报告在开始处要求有目录，最后可以对设计过程的收获、体会等进行总结。

主机系统设计报告要求包括以下内容：

1．给出设计出的主机系统总体结构框图。

2．给出数据通路结构设计图；其中包含各部件的结构设计及各微操作命令（控制信号）。

3．给出各部件的详细设计图（包括构成各部件的器件选用）。

4．给出设计出的指令系统中的各种机器指令（列出指令操作码助记符、操作码的机器指令代码及指令功能的对应关系表）。

5．给出设计出的指令格式、寻址方式。

根据控制部件的设计方法，对下述第 6 条内容及第 7 条内容进行二者选一。

6．写出微程序控制部件的设计方法及调试过程。

（1） 阐述时序系统的设计方法。

（2） 给出微指令的结构图（说明微指令中各位的含义）、微命令功能说明、控制信号的逻辑表达式并给出相应的逻辑电路图（微指令各位与时序信号相配合的逻辑电路图）。在此，可以用列表的形式给出各部分内容。

（3）写出“取指令”公共操作及各条机器指令的指令流程。

（4）写出“取指令”微程序及对应各条机器指令的微程序。要求注明每段微程序的入口地址。在写出每条微指令时，用微命令助记符和 16 进制数两种方式标记。

（5）给出控制部件的逻辑连线图及芯片封装图。

（6）写出微程序控制部件的调试过程。

7．写出组合逻辑控制部件的设计方法及调试过程。

（1）说明时序系统的详细设计方法（包括机器周期发生器、节拍发生器及工作脉冲发生器的设计），并给出相应的逻辑电路图及其芯片封装图。

（2）写出“取指令”及各类机器指令的指令流程及微操作时间表。

（3）列出各微操作命令的逻辑表达式。

（4）给出微操作命令发生器（控制信号发生器）的逻辑电路图及其芯片封装图。

（5）写出组合逻辑控制部件的调试过程。

8．给出编写的调试程序，对每条机器指令要有注释。

9．给出与调试程序相对应的时序模拟仿真的测试结果输出波形图。

附录 A　基本部件详细设计图

1、时序发生器：

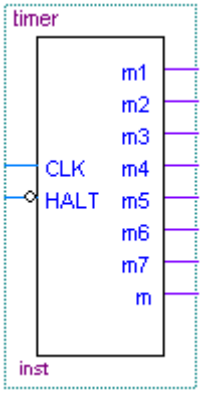


图 A-1　时序发生器封装图

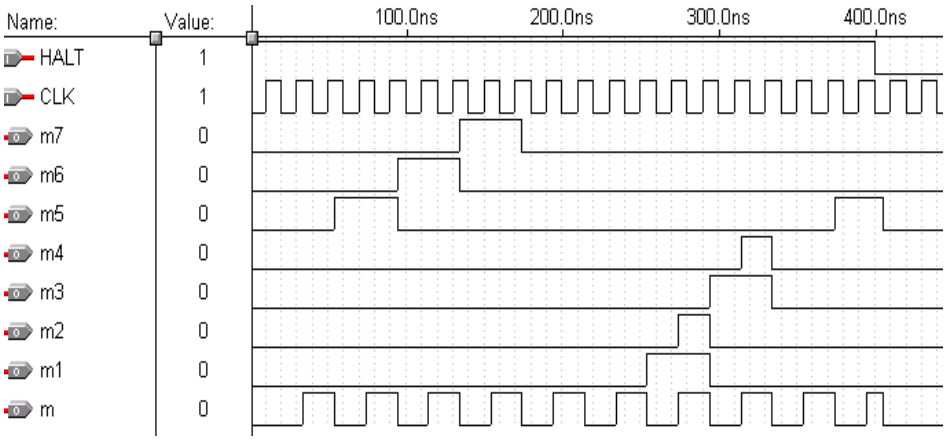


图 A-2　时序发生器仿真波形图

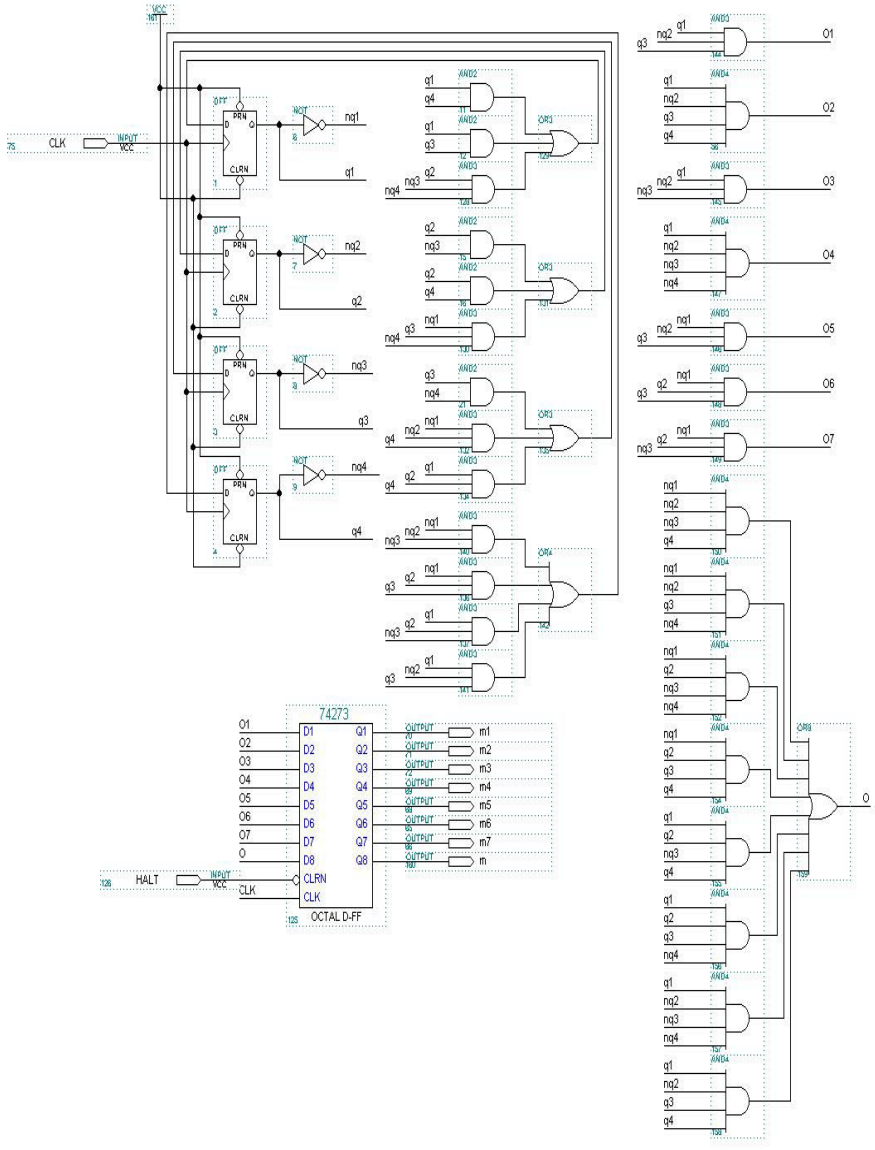


图 A-3　时序发生器详细设计图

2、程序计数器：

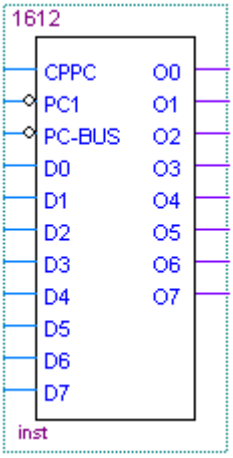


图 A-4　程序计数器封装图

表 A-1 程序计数器功能表

PC1	CPPC	D［0..7］	功能
L	↑	XXH	输入存储
H	↑	X	加 1（计数）
H	无↑	X	保持

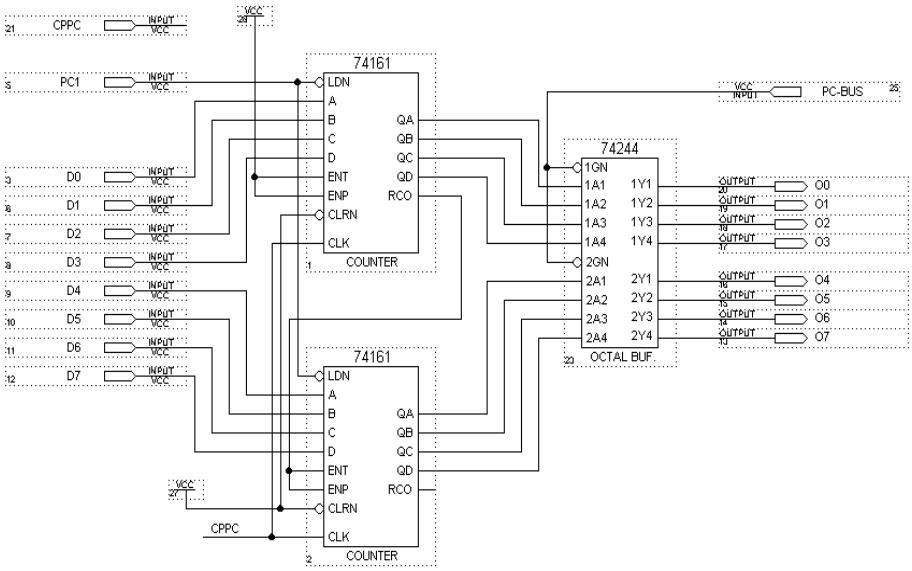


图 A-5　程序计数器详细设计图

3、运算器：

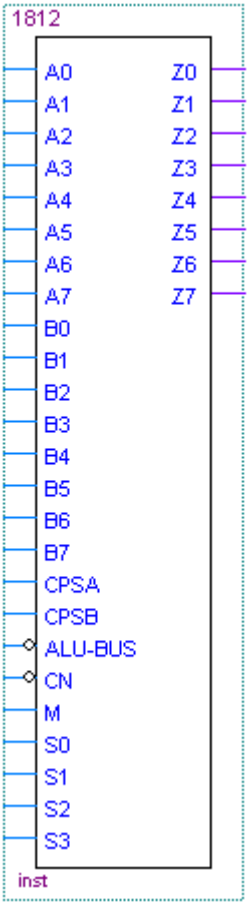


图 A-6　运算器封装图

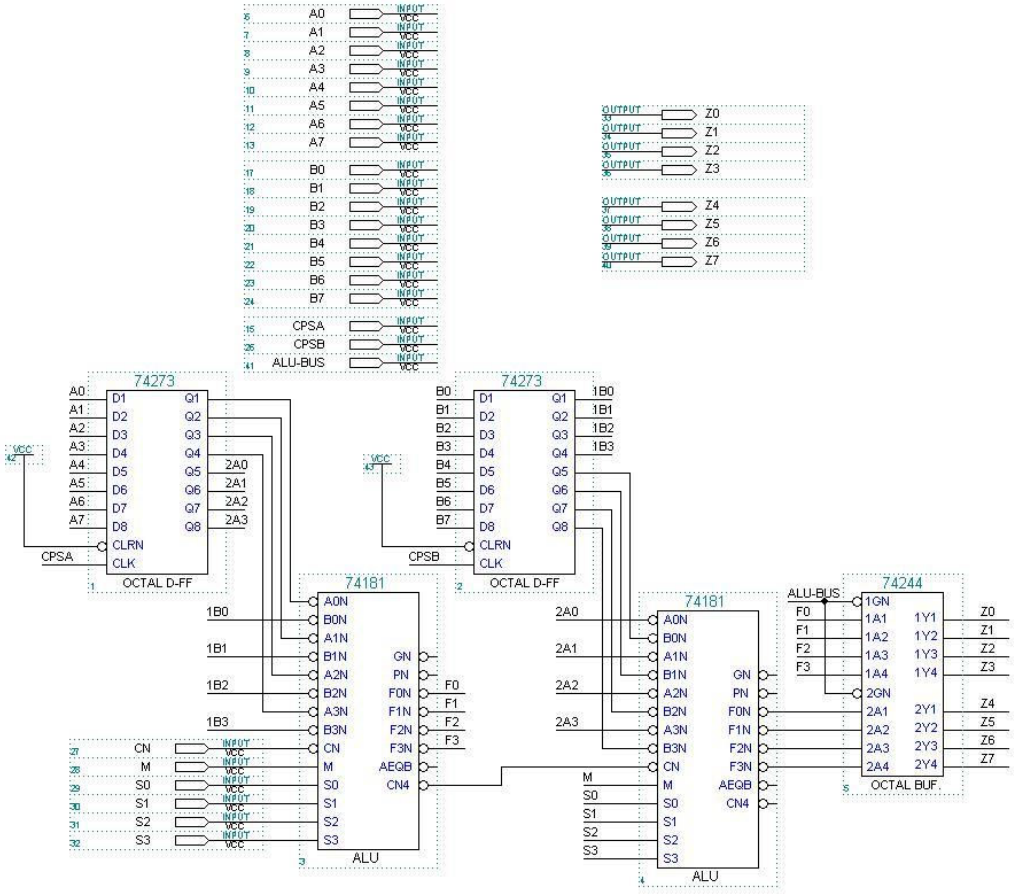


图 A-7　运算器详细设计图

4、存储器：

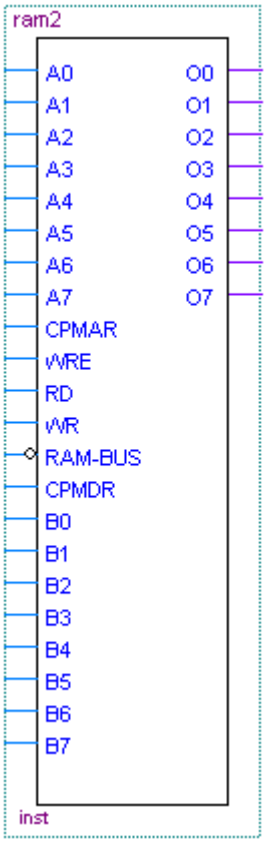


图 A-8　存储器封装图

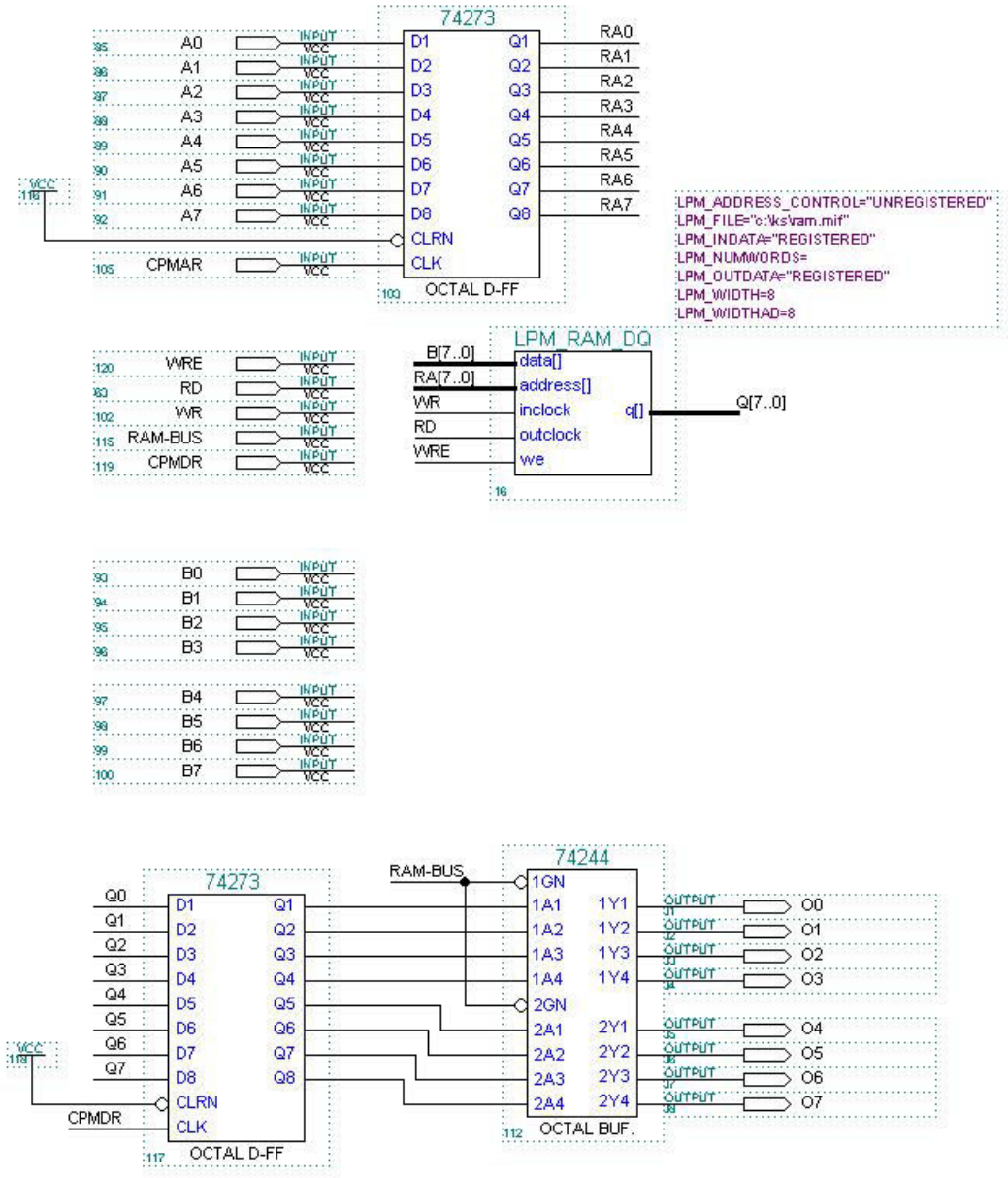


图 A-9　存储器详细设计图

5、通用寄存器组：

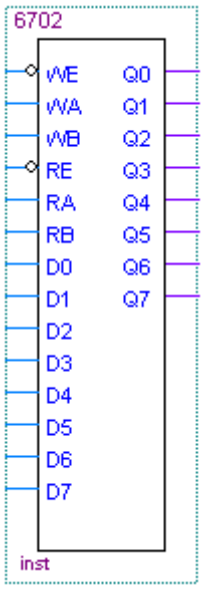


图 A-10　存储器封装图

表 A-2 通用寄存器组写功能控制方式

WE	WA	WB	选定寄存器
L	L	L	RA
L	L	H	RB
L	H	L	RC
L	H	H	RD
H	X	X	X

表 A-3　通用寄存器组读功能控制方式

RE	RA	RB	选定寄存器
L	L	L	RA
L	L	H	RB
L	H	L	RC
L	H	H	RD
H	X	X	X

其中：L 为低电位，H 为高电位，X 为任意数据。

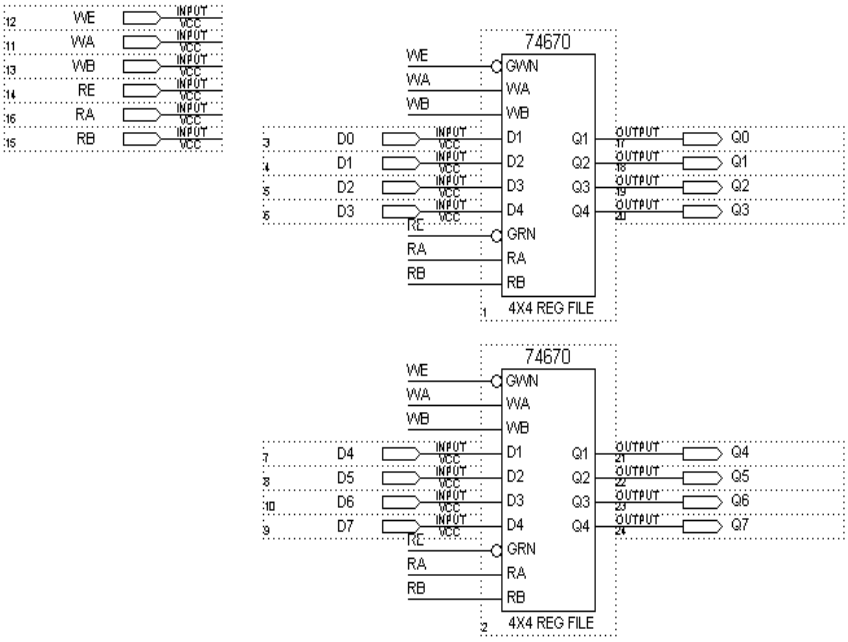


图 A-11　通用寄存器组详细设计图

6、总线暂存器：

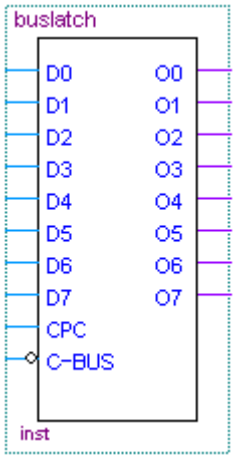


图 A-12　总线暂存器封装图

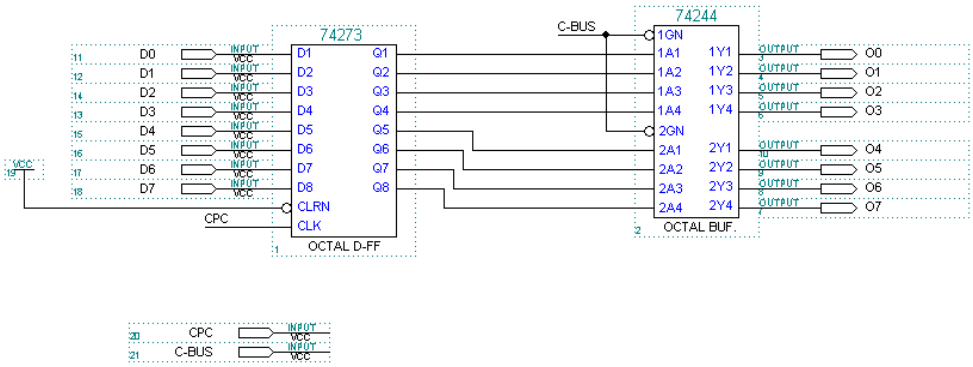
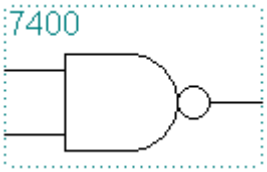


图 A-13　总线暂存器详细设计图

附录 B　集成电路芯片简介

1、7400：与非门

（1）芯片图：



（2）逻辑表达式：

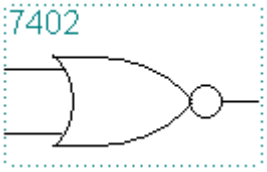
$Y = \overline{A\,B}$

（3）功能表：

输入		输出
A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

2、7402：或非门

（1）芯片图：



（2）逻辑表达式：

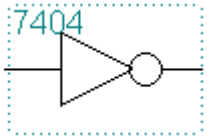
$Y = \overline{A + B}$

（3）功能表：

输入		输出
A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0

3、7404：非门

（1）芯片图：



（2）逻辑表达式：

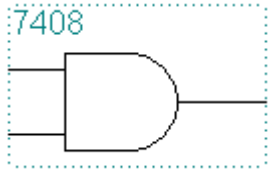
$Y = \overline{A}$

（3）功能表：

输入	输出
0	1
1	0

4、7408：与门

（1）芯片图：



（2）逻辑表达式：

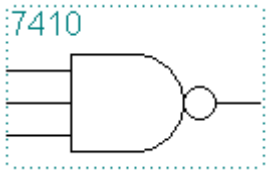
$Y = A\,B$

（3）功能表：

输入		输出
A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

5、7410：三与非门

（1）芯片图：



（2）逻辑表达式：

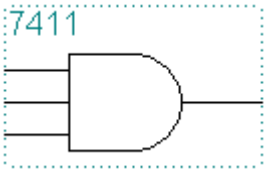
$Y = \overline{A\,B\,C}$

（3）功能表：

输入			输出
A	B	C	OUT
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

6、7411：三或门

（1）芯片图：



（2）逻辑表达式：

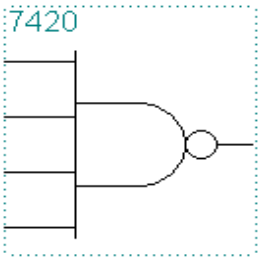
$Y = A\,B\,C$

（3）功能表：

输入			输出
A	B	C	OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

7、7420：四与非门

（1）芯片图：



（2）逻辑表达式：

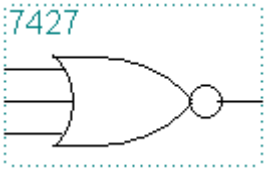
$Y = \overline{A\,B\,C\,D}$

（3）功能表：

输入				输出
A	B	C	D	OUT
0	X	X	X	1
X	0	X	X	1
X	X	0	X	1
X	X	X	0	1
1	1	1	1	0

8、7427：三或非门

（1）芯片图：



（2）逻辑表达式：

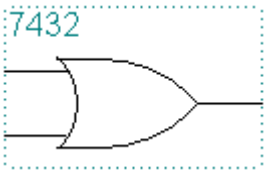
$Y = \overline{A + B + C}$

（3）功能表：

输入			输出
A	B	C	OUT
0	0	0	1
1	X	X	0
X	1	X	0
X	X	1	0

9、7432：或门

（1）芯片图：



（2）逻辑表达式：

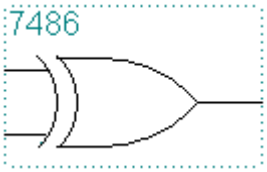
$Y = A + B$

（3）功能表：

输入		输出
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

10、7486：异或门

（1）芯片图：



（2）逻辑表达式：

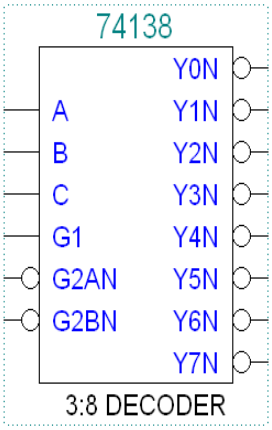
$Y = A \oplus B$

（3）功能表：

输入		输出
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

11、74138：3—8 译码器

（1）芯片图：



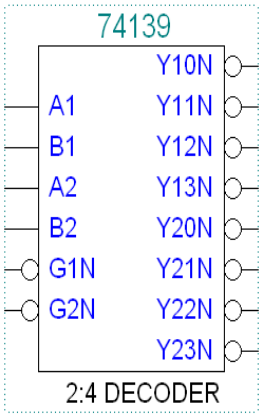
G1，G2AN，G2BN：使能端；
CBA：选择端；
Y7N~Y0N：输出端。

（2）功能表：

门控		输入			输出							
G1	G2AN+G2BN	C	B	A	Y7N	Y6N	Y5N	Y4N	Y3N	Y2N	Y1N	Y0N
X	1	X	X	X	1	1	1	1	1	1	1	1
0	X	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	1	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	0	1	1	1	1
1	0	1	0	1	1	1	0	1	1	1	1	1
1	0	1	1	0	1	0	1	1	1	1	1	1
1	0	1	1	1	0	1	1	1	1	1	1	1

12、74139：两个 2—4 译码器

（1）芯片图：



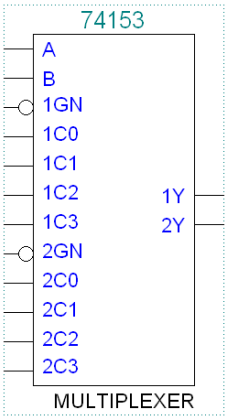
A1、B1、A2、B2：选择端
G1N，G2N：使能端；
Y10N~Y13N：输出端；
Y20N~Y23N：输出端；

（2）功能表：

使能	输入	输出
GN	B A	Y3N Y2N Y1N Y0N
1	X X	1 1 1 1
0	0 0	1 1 1 0
0	0 1	1 1 0 1
0	1 0	1 0 1 1
0	1 1	0 1 1 1

13、74153：双四选一线数据选择器

（1）芯片图：



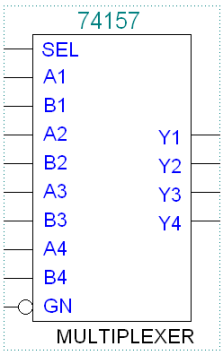
A、B：选择输入端；
1GN、2GN：选通端；
1C3~1C0、2C3~2C0：数据输入端；
1Y、2Y：输出端。

（2）功能表：

选通 GN	选择 B A	输出 Y
1	X X	0
0	0 0	Y = C0
0	0 1	Y = C1
0	1 0	Y = C2
0	1 1	Y = C3

14、74157：四二选一线数据选择器

（1）芯片图：



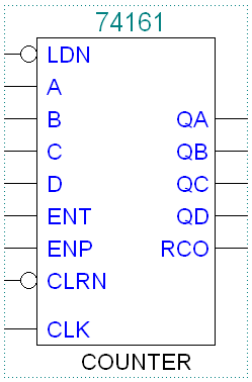
GN：选通输入端；
SEL：数据选择端；
A4~A1、B4~B1：数据输入端；
Y4~Y1：输出端。

（2）功能表：

选通 GN	选择 SEL	输出 Y
1	X	0
0	0	Y = A
0	1	Y = B

15、74161：4 位二进制同步计数器（直接清除）

（1）芯片图：



A~D：数据输入端
LDN：预置信号端；
CLRN：清零信号端；
CLK：时钟脉冲；
ENT、ENP：工作允许信号；
QD~QA：输出端；
RCO：串行进位信号端。

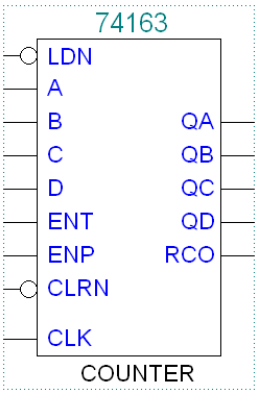
（2）功能表：

控制				功能
CLRN	CLK	LDN	ENP ENT	
0	X	X	X X	清零
1	↑	0	X X	接数
1	↑	1	1 1	计数加 1
1	X	1	0 X	保持
1	X	1	X 0	保持

RCO = QD & QC & QB & QA & ENT

16、74163：4 位二进制同步计数器（同步清除）

（1）芯片图：



A~D：数据输入端
LDN：预置信号端；
CLRN：清零信号端；
CLK：时钟脉冲；
ENT、ENP：工作允许信号；
QD~QA：输出端；
RCO：串行进位信号端。

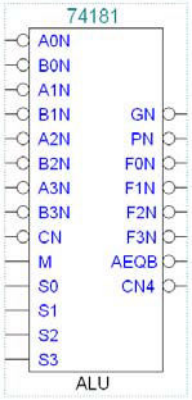
（2）功能表：

控制				功能
CLRN	CLK	LDN	ENP ENT	
0	↑	X	X X	清零
1	↑	0	X X	接数
1	↑	1	1 1	计数加 1
1	X	1	0 X	保持
1	X	1	X 0	保持

RCO = QD & QC & QB & QA & ENT

17、74181：4 位 ALU

（1）芯片图：



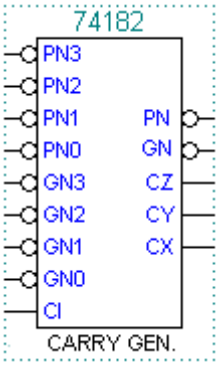
A0N~A3N：A 数据输入端；
B0N~B3N：B 数据输入端；
S0~S3：功能选择端；
CN：串行进位输入端；
M：模式控制输入端；
F0N~F3N：结果输出端；
GN：快速进位端；
PN：快速进位端；
AEQB：A 等于 B；
CN4：串行进位输出端。

（2）功能表：

选择	M = H	M = L 算术操作	
$S_3S_2S_1S_0$	逻辑操作	$\overline{CN}$ = H（无进位）	$\overline{CN}$ = L（有进位）
0000	F = $\overline{A}$	F = A	F = A 加 1
0001	F = $\overline{A+B}$	F = A + B	F =（A+B）加 1
0010	F = $\overline{A}$ B	F = A + $\overline{B}$	F =（A+$\overline{B}$）加 1
0011	F = 0	F = 减 1（2 的补）	F = 0
0100	F = $\overline{A\,B}$	F = A 加 A $\overline{B}$	F = A 加 A $\overline{B}$ 加 1
0101	F = $\overline{B}$	F =（A+B）加 $\overline{AB}$	F =（A+B）加 $\overline{A\,B}$ 加 1
0110	F = A ⊕ B	F = A 减 B 减 1	F = A 减 B
0111	F = A $\overline{B}$	F = A $\overline{B}$ 减 1	F = A $\overline{B}$
1000	F = $\overline{A}$ + B	F = A 加 AB	F = A 加 AB 加 1
1001	F = $\overline{A \oplus B}$	F = A 加 B	F = A 加 B 加 1
1010	F = B	F =（A+$\overline{B}$）加 A B	F =（A+$\overline{B}$）加 AB 加 1
1011	F = A B	F = A B 减 1	F = A B
1100	F = 1	F = A 加 A	F = A 加 A 加 1
1101	F = A + $\overline{B}$	F =（A+B）加 A	F =（A+B）加 A 加 1
1110	F = A + B	F =（A+$\overline{B}$）加 A	F =（A+$\overline{B}$）加 A 加 1
1111	F = A	F = A 减 1	F = A

18、74182：组间并行进位组件

（1）芯片图：



PN0~PN3：小组进位传递函数输入端；
GN0~GN3：小组进位产生函数输入端；
CI：组间进位输入端；
PN：大组进位传递函数输出端；
GN：大组进位产生函数输出端；
CX、CY、CZ：组间进位信号输出端；

（2）功能表：

输入	输出
GN3 GN2 GN1 GN0 PN3 PN2 PN1	GN
L X X X X X X	L
X L X X L X X	L
X X L X L L X	L
X X X L L L L	L
其他	H

输入	输出
PN3 PN2 PN1 PN0	PN
L L L L	L
其他	H

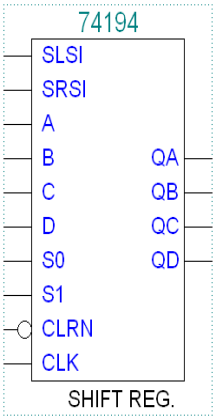
输入	输出
GN0 PN0 CI	CX
L X X	H
X L H	H
其他	L

输入	输出
GN1 GN0 PN1 PN0 CI	CY
L X X X X	H
X L L X X	H
X X L L H	H
其他	L

输入							输出
GN2	GN1	GN0	PN2	PN1	PN0	CI	GN
L	X	X	X	X	X	X	H
X	L	X	L	X	X	X	H
X	X	L	L	L	X	X	H
X	X	X	L	L	L	H	H
其他							L

19、74194：带并行输入的四位双向移位寄存器

（1）芯片图：



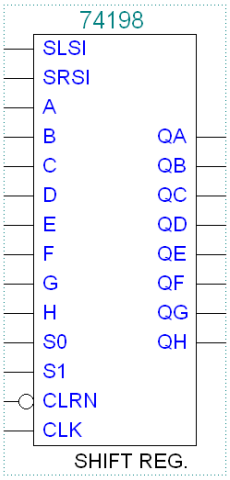
CLRN：清零；
S1、S0：模式选择端；
A~D：并行输入端；
SLSI、SRSI：串行输入端；
CLK：时钟；
QA~QD：输出端。

（2）功能表：

输入										输出			
	模式		时钟	串行		并行							
CLRN	S1	S0	CLK	SLSI	SRSI	A	B	C	D	QA	QB	QC	QD
0	X	X	X	X	X	X	X	X	X	0	0	0	0
1	X	X	0	X	X	X	X	X	X	QAo	QBo	QCo	QDo
1	1	1	↑	X	X	a	b	c	d	a	b	c	d
1	0	1	↑	X	1	X	X	X	X	1	QAn	QBn	QCn
1	0	1	↑	X	0	X	X	X	X	0	QAn	QBn	QCn
1	1	0	↑	1	X	X	X	X	X	QBn	QCn	QDn	1
1	1	0	↑	0	X	X	X	X	X	QBn	QCn	QDn	0
1	0	0	↑	X	X	X	X	X	X	QAo	QBo	QCo	QDo

20、74198：八位双向移位寄存器

（1）芯片图：



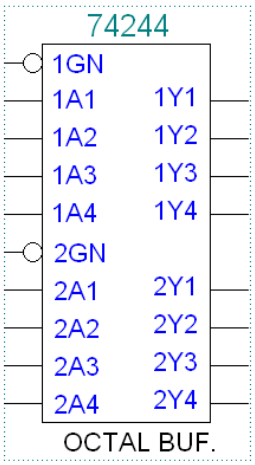
CLRN：清零；
S1、S0：模式选择端；
A~H：并行输入端；
SLSI、SRSI：串行输入端；
CLK：时钟；
QA~QH：输出端。

（2）功能表：

输入							输出		
	模式		时钟	串行		并行			
CLRN	S1	S0	CLK	SLSI	SRSI	A ..H	QA	QB..QG	QH
0	X	X	X	X	X	X	0	0..0	0
1	X	X	0	X	X	X	QAo	QBo..QGo	QHo
1	1	1	↑	X	X	a..h	a	b..g	h
1	0	1	↑	X	1	X	1	QAn..QFn	QGn
1	0	1	↑	X	0	X	0	QAn..QFn	QGn
1	1	0	↑	1	X	X	QBn	QCn..QHn	1
1	1	0	↑	0	X	X	QBn	QCn..QHn	0
1	0	0	↑	X	X	X	QAo	QBo..QGo	QHo

21、74244：原码三态输出传送门

（1）芯片图：



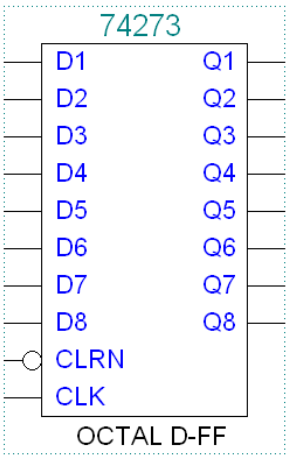
1GN、2GN：选通输入端；
1A4~1A1：A 数据输入端；
2A4~2A1：B 数据输入端；
1Y4~1Y1：A 输出端；
2Y4~2Y1：B 输出端。

（2）功能表：

输入		输出
GN	A	Y
0	0	0
0	1	1
1	X	浮空

22、74273：带清除端八 D 型触发器

（1）芯片图：



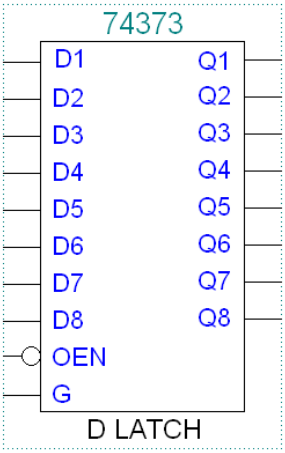
CLRN：清零信号端；
CLK：时钟脉冲；
D8~D1：数据输入端；
Q8~Q1：输出端。

（2）功能表：

输入			输出
CLRN	CLK	D	Q
0	X	X	0
1	↑	1	1
1	↑	0	0
1	0	X	Qo

23、74373：三态输出 8D 锁存器

（1）芯片图：



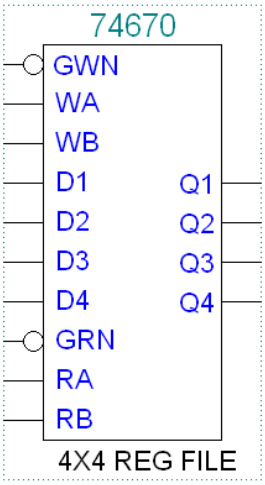
OEN：输出控制端；
G：接数控制端；
D8~D1：数据输入端；
Q8~Q1：输出端。

（2）功能表：

输出控制 OEN	接数控制 G	输出 Q
1	X	高阻态
0	0	Q = Qo
0	1	Q = D

24、74670：三态输出的 4x4 寄存器堆

（1）芯片图：



GWN：写控制端；
WA、WB：寄存器选择端；
D4~D1：数据写入端；
GRN：读控制端；
RA、RB：寄存器选择端；
Q4~Q1：输出端。

（2）功能表：

写功能表：

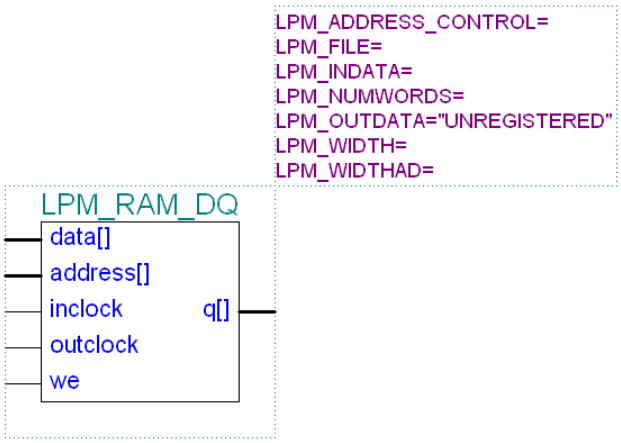
写控制			寄存器			
GWN	WB	WA	W0	W1	W2	W3
1	X	X	Qo	Qo	Qo	Qo
0	0	0	D	Qo	Qo	Qo
0	0	1	Qo	D	Qo	Qo
0	1	0	Qo	Qo	D	Qo
0	1	1	Qo	Qo	Qo	D

读功能表：

读控制			输出			
GRN	RB	RA	Q1	Q2	Q3	Q4
1	X	X	Z	Z	Z	Z
0	0	0	W0B1	W0B2	W0B3	W0B4
0	0	1	W1B1	W1B2	W1B3	W1B4
0	1	0	W2B1	W2B2	W2B3	W2B4
0	1	1	W3B1	W3B2	W3B3	W3B4

25、LPM_RAM_DQ：RAM

（1）芯片图：



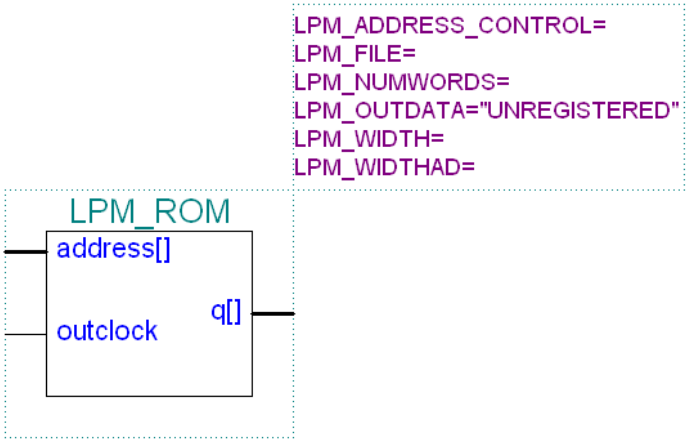
data[ ]：数据输入端；
address[ ]：地址输入端；
inclock：输入控制端；
outclock：输出控制端；
we：输入控制端；
q[ ]：输出端。

（2）功能表：

	输入控制			功能
	We (WRE) 写使能	Inclock (WR) 写控制	Outclock (RD) 读控制	
读	X	X	↑触发 1 有效	q[ ]输出 RAM 中相应 address[ ]地址中的内容
写	1	↑	X	在 RAM 中相应 address[ ]地址中写入数据 data[ ]

26、LPM_ROM：ROM

（1）芯片图：



address[ ]：地址输入端；
outclock：输出控制端：
q[ ]：输出端。

（2）功能：outclock 上升沿触发、高电位有效，q[ ]输出 ROM 中相应 address[ ]地址中的内容。

附录 C　设计调试过程中的注意事项

1. 在 MAX+plusII 软件中，计算机内部存储文件的路径名中不能出现中文。
2. 在 RAM、ROM 所带的初始化文件中，中文不能加入.mif 文件的注释中。
3. 更改某芯片的内容后,在调用该芯片的最外层电路连线图上,点击 Symbol/Update Symbol 则可完成芯片的更新。若是带.mif 的芯片，还可同时完成对.mif 文件的修改。
4. 改变 RAM、ROM 在磁盘中的位置时，需要将其附带的初始化文件的对应路径进行修改。
5. 总线上显示××内容，则表示在相应时刻有多于一个部件同时向总线发送信息，以导致总线数据不确定。
6. 在编译时若出现“TRI or OPNDRN buffer <net ID number> can only drive logic [(<net ID>)] if connected to a BIDIR pin” 类似的与三态输出相关的错误，应考虑将三态输出改为 2 值输出。
7. 在 QuartusII 设计软件中，功能仿真比时序仿真多一步：先要 Processing—Generate Functional Simulation Netlist，生成功能仿真网表文件，再仿真。否则会报错。
8. 利用 Verilog HDL 硬件描述语言编程时，变量名称不能用“-”符号，否则报错。

参考资料

1. 陈文智等. 嵌入式系统开发原理与实践. 北京：清华大学出版社，2005.8

2. 徐欣等. 基于 FPGA 的嵌入式系统设计. 北京：机械工业出版社. 2005.1

3. 杨恒等. FPGA/CPLD 最新实用技术指南. 北京：清华大学出版社. 2005.1

4. 于枫等. ALTERA 可编程逻辑器件应用技术. 北京：科学出版社. 2004.9

5. 俸远祯等. 计算机组成原理与汇编语言程序设计. 北京：电子工业出版社 2004.1

6. 夏宇闻. Verilog 数字系统设计教程. 北京：北京航空航天大学出版社，2003.7

7. 王金明等. 数字系统设计与 Verilog HDL. 北京：电子工业出版社. 2002.1

8. 易小琳等. 基于 EDA 平台的计算机系统硬件课程虚拟化实践的研究. 《中国大学教学》. 北京：高等教育出版社. 2005.7