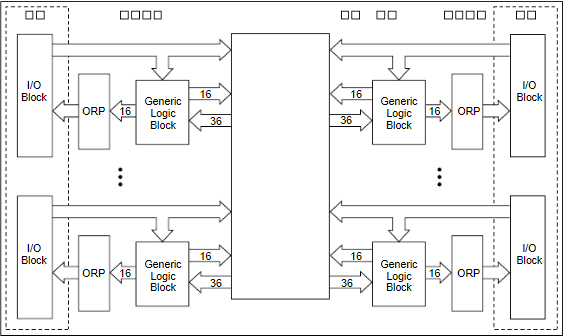
isp MACH芯片的功能及源程序分析

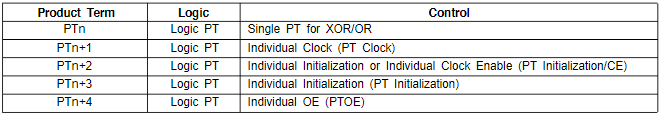
参与构成实验室TEC-XP-II实验箱上的第一个系统的isp MACH芯片是由Lattice公司生产的一款复杂可编程逻辑器件（CPLD），这款芯片写入不同的程序就能实现不同的硬件功能，从而可以让用户通过编程实现硬件电路设计。isp MACH系列芯片具有下述一些特征：

1. 高性能，最大工作频率为400MHz；
2. 便于设计，增强的宏单元和单独的时钟、复位、置位及时钟使能控制；
3. 低功率；
4. 不同型号器件支持多个温度范围；
5. 简单的系统集成

概述

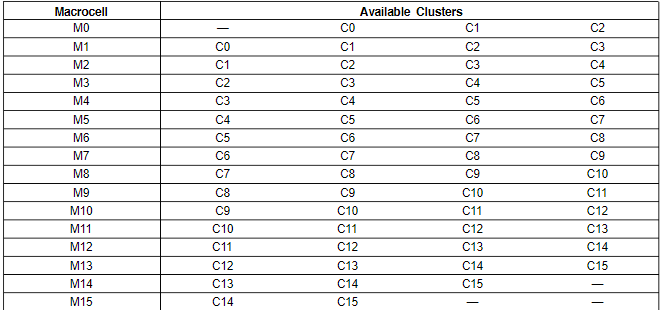
isp MACH 4000系列芯片由多个36输入、16宏单元通用逻辑块组成，通过一个全局路由池（GRP）互联，输出由路由池将GLBs连接到IO块，其中包含多个输入输出单元，体系结构如图1所示：  
图1 结构框图

isp MACH 4000中的IO块被分成两组，每组都有独立的IO电源，输入可以支持各种独立于芯片或者组的电源标准，输出支持与提供电源相匹配的标准。

将用作逻辑的产品术语引入与集群相关的5输入或门，用作控制的产品术语引入与集群相关的宏单元或IO单元。表1显示了集群中每个产品项可用的函数，或门输出连接到相关的IO单元，为窄组合函数提供快捷路径，并连接到逻辑分配器。 表1

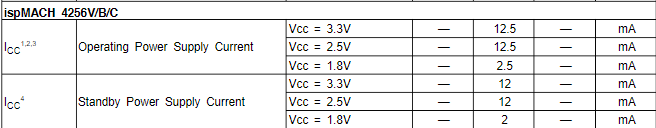
集群分配器

集群分配器允许将集群引入到附近的宏单元，从而允许创建具有更多产品术语的函数，表2显示哪些集群可以被引入到哪些宏单元，以这种方式使用，集群分配器可用于形成最多20个产品术语的函数。

表2

Isp MACH 4256V推荐的工作条件，如表3

表3 操作条件



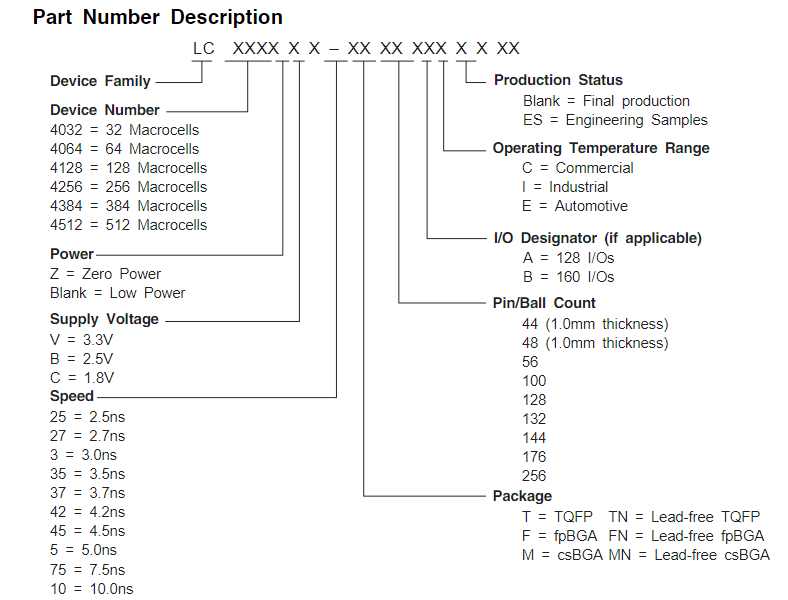
isp MACH系列芯片的命名规则：（见图2）  


图2

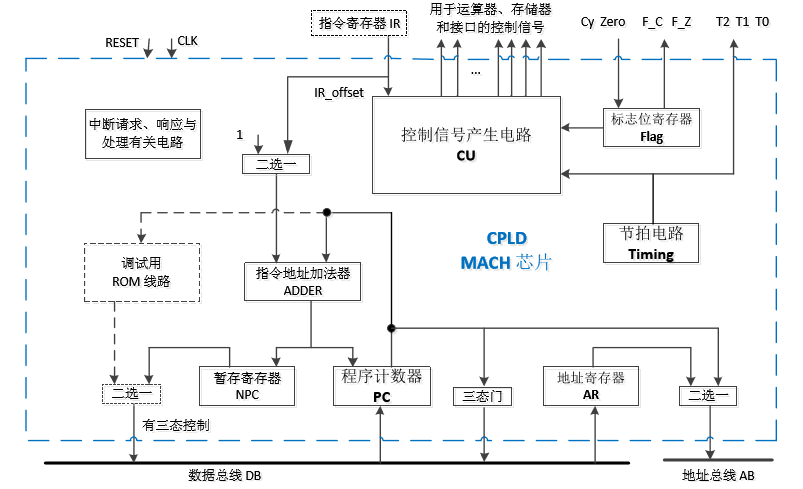
实验室所用的TEC-XP-II型实验箱上的MACH芯片的型号为LC4256V 75T176E，参考图2可知，这颗芯片内部有256个宏单元，支持的电压为3.3V，采用的封装方案为TQFP（thin quad flat package），即薄塑封四角扁平封装。其内部的管脚数为176，操作温度范围为自动。



图3 LC4256V系列芯片说明

isp MACH芯片内部电路构成和功能分析：

根据实验箱第一个硬件系统的组成来看，控制器MACH芯片内部的基本电路组成应包含有PC(程序计数器)、Timing(节拍发生器电路)、以及CU(控制信号产生电路)，控制器MACH芯片的基本构成如图4所示：

图4

其中，程序计数器PC的功能是存放当前正在执行的或者下一条将要执行的指令的地址。当执行一条指令时，首先根据PC中存放的指令地址到内存中取出指令送往指令寄存器中，与此同时，PC中的地址或者自动加1或者由转移指针直接给出下一条指令的地址。此后执行的操作就是分析指令，执行指令，完成第一条指令的执行后，根据PC中的地址取出第二条指令，如此循环执行后续指令。

地址寄存器AR用来保存当前CPU所要访问的内存单元或IO设备的地址。由于内存和CPU之间存在速度上的差别，所以必须使用地址寄存器来保存地址信息，直到读写操作完成为止。地址寄存器AR还能起到隔离和缓冲的作用。地址寄存器采用单纯的寄存器结构，在对主存或I/O端口进行访问时，地址寄存器存放当前访问的地址，CPU通过修改地址寄存器中的值，就可访问不同的存储器单元及不同的I/O端口。

控制信号产生电路的功能是产生计算机各部件使用的控制信号，各部件包括运算器部件、存储器部件、总线及输入输出接口电路等，也包括控制器部件自身，通常选用可现场编程，高集成度的，支持与或两级逻辑阵列的器件实现。

节拍发生器Timing，是用节拍的不同编码来区分和表示指令的执行步骤，以及计算机工作所需的节拍电位、节拍脉冲，通常由触发器构成，是典型的时序逻辑电路。

从图4中我们还可以看到MACH芯片内部还另外设置有指令地址加法器电路，用来产生数的和，以便于后续进行程序计数器PC的内容的变换，即指令地址的计算。调试用ROM线路是用来保存由指令代码组成的小的调试程序的，可以用于扩展指令过程中的调试操作。

TEC-XP-II实验箱第一个系统构成部分isp MACH芯片的源程序，选用ABEL语言对电路组成进行描述：

MODULE TEC\_new

TITLE 'controller component' //控制器组件源程序

"a\_16\_zh\_jb.abl 2016/10/31

DECLARATIONS //程序声明段，用来说明用到的信号和管脚分配情况

RESET,CLK pin 151,68; "系统复位和时钟信号

IR15..IR0 pin 64..57,54..47; "指令寄存器的输出信号

IR\_G,IR\_clk pin 174,175; "IR的的接收控制和时钟

Cy,Zero pin 169,171; "运算器芯片Am2901产生的标志位信息（进位标志位和零标志位）

AB15..AB0 pin 87..80,77..70; "地址总线

DB15..DB0 pin 24,23,26,25,28,27,30,29,32..39; "数据总线

MIO,REQ,WE pin 95,94,93 istype 'dc,com'; "控制内存和串口的信号

I8..I0 pin 14..21, 135 istype 'dc,com'; "控制运算器的信号

B3..B0,A3..A0 pin 9..12, 5..8 istype 'com';

aluoe,ram15,ram0,c0 pin 136,137,139,141 istype 'com'; //运算器输出使能，移位输入及低位进位

T2..T0, F\_C,F\_Z pin 162,160,158,168,170; "显示节拍编码和标志位

pc15..pc0, ar15..ar0 node istype 'reg,keep'; "程序计数器、地址寄存器

npc15..npc0 node istype 'reg,keep'; "暂存主程序断点的寄存器

t\_2..t\_0, flag\_c, flag\_z node istype 'reg,keep'; "节拍发生器、标志寄存器

pc\_ce,ar\_ce node istype 'com'; "PC、AR的接收条件（使能）信号

flag\_c\_ce,flag\_z\_ce node istype 'com'; "标志位的接收条件信号

A\_,B\_ ,jr\_5, DB\_oe node istype 'com'; "中间信号、数据总线接收使能信号，允许MACH送信息到DB

sum15..sum0 node istype 'com'; "专用加法器电路

wk15..wk0, cy15..cy1 node istype 'com'; "加法器一路输入、每位进位输出

A\_zu,B\_zu, D\_zu, shift node istype 'com'; "指令分组

arith,logic,jr\_zu node istype 'com';

M\_IO,temp node istype 'com';

flag\_c\_in,AB\_sel node istype 'com'; //进位标志位信息，A，B寄存器选择

inst15..inst0 node istype 'com'; //16位的中断信号

dst\_2,dst\_1,dst\_0 pin 106,105,104 istype 'com';

timing=[t\_2..t\_0]; c,z,x=.C.,.Z.,.X.; "说明常量和集合

cif =(timing==[0,0,0]); //取指令

cmem1=(timing==[0,1,1]); //写内存

cexe=(timing==[0,1,0]); //指令执行

cmem2=(timing==[0,0,1]); //写内存

pc=[pc15..pc0]; IR=[IR15..IR0]; ar=[ar15..ar0]; npc=[npc15..npc0];

DB=[DB15..DB0]; AB=[AB15..AB0]; sum=[sum15..sum0];

ir\_op=[IR15..IR8]; "高8位设为指令操作码，给出每条指令的操作码和汇编语句名

ADD = (ir\_op==[0,0,0,0,0,0,0,0]); SUB = (ir\_op==[0,0,0,0,0,0,0,1]); "算术运算逻辑运算指令9条

DEC = (ir\_op==[0,0,0,0,1,0,0,0]); INC = (ir\_op==[0,0,0,0,1,0,0,1]);

CMP = (ir\_op==[0,0,0,0,0,0,1,1]); AND = (ir\_op==[0,0,0,0,0,0,1,0]);

XOR\_= (ir\_op==[0,0,0,0,0,1,0,0]); TEST= (ir\_op==[0,0,0,0,0,1,0,1]); //TEST指令将两个操作数进行与运算，并根据结果设置相关标志位，但不会保存运算结果

OR\_ = (ir\_op==[0,0,0,0,0,1,1,0]);

SHL = (ir\_op==[0,0,0,0,1,0,1,0]); SHR = (ir\_op==[0,0,0,0,1,0,1,1]); "移位指令2条（逻辑左移和逻辑右移）

MVRD= (ir\_op==[1,0,0,0,1,0,0,0]); //将数据传送到目的寄存器当中

MVRR= (ir\_op==[0,0,0,0,0,1,1,1]); //寄存器间进行传送

JRC = (ir\_op==[0,1,0,0,0,1,0,0]); //当进位标志位为1时，执行跳转

JRZ = (ir\_op==[0,1,0,0,0,1,1,0]); //当零标志位为1时，执行跳转

JRNC= (ir\_op==[0,1,0,0,0,1,0,1]); //与JRC相对，C=0时进行跳转

JRNZ= (ir\_op==[0,1,0,0,0,1,1,1]); //与JRZ相对，Z=0时进行跳转

JR = (ir\_op==[0,1,0,0,0,0,0,1]); //无条件跳转

JMPA= (ir\_op==[1,0,0,0,0,0,0,0]); //无条件跳转

IN\_ = (ir\_op==[1,0,0,0,0,0,1,0]); OUT = (ir\_op==[1,0,0,0,0,1,1,0]); "输入输出指令2条

LDRR= (ir\_op==[1,0,0,0,0,0,0,1]); STRR= (ir\_op==[1,0,0,0,0,0,1,1]); "读写内存指令6条

PUSH= (ir\_op==[1,0,0,0,0,1,0,1]); POP = (ir\_op==[1,0,0,0,0,1,1,1]);

PSHF= (ir\_op==[1,0,0,0,0,1,0,0]); //状态标志入栈

POPF= (ir\_op==[1,0,0,0,1,1,0,0]); //弹出栈顶数据送状态标志寄存器

RET = (ir\_op==[1,0,0,0,1,1,1,1]); //子程序返回

CALA= (ir\_op==[1,1,0,0,1,1,1,0]); //调用首地址为ADR的子程序

LDPC=(ir\_op==[1,0,0,0,1,0,0,1]); "特权指令各1条

link1, link0 pin 164,163; "２个功能开关

YY7..YY4,YY1 pin 126,127,124,125,120; "提示可手拨指令的4个灯,244芯片的G信号

EQUATIONS //程序逻辑功能描述段，用来读电路功能进行描述

temp=cif#cexe&(MVRD#JMPA#CALA); "读取指令的节拍

MIO=M\_IO&((link1& link0#!link1&!link0) "正常的运行方式

#(link1&!link0#!link1&link0)&!temp) "片内或手拨方式非取指周期

#(link1&!link0#!link1&link0)&temp; "手拨或片内方式取指周期

[YY7..YY4]=link1&!link0&[temp,temp,temp,temp]; "４个灯亮提示可手拨指令

!YY1 =temp&link1&!link0; "244芯片的输出使能信号G

[T2,T1,T0]=[t\_2,t\_1,t\_0]; "显示节拍编码

F\_C=flag\_c; F\_Z=flag\_z; "显示标志位信息

[dst\_2,dst\_1,dst\_0]=[t\_2,t\_1,t\_0];

"寄存器电路的接收条件和时钟信号

pc\_ce=RESET#!RESET&( cif#cexe&(MVRD#JMPA#CALA#jr\_5)

# cmem1&(RET#LDPC) );

pc.clk=CLK; pc.ce=pc\_ce; timing.clk=CLK;

jr\_5=JR#JRNC&!flag\_c#JRC&flag\_c#JRNZ&!flag\_z#JRZ&flag\_z; "相对转移指令的转移条件

A\_zu=!IR15#MVRD#JMPA#IN\_#OUT;

B\_zu=LDRR#STRR#PUSH#POP#PSHF#POPF#RET#LDPC; "要读写数据存储器的指令

D\_zu=CALA;

arith=ADD#SUB#CMP#DEC#INC; "算术运算指令

logic=AND#TEST#OR\_#XOR\_; "逻辑运算指令

shift=SHL#SHR; "移位运算指令

"复位时,为PC、timing 赋初值为0

when RESET then { pc=0; timing:=[0,0,0]; } else "系统运行状态

{ when cif then timing:=[0,1,0]; "TIMING状态转换

when cexe&(B\_zu#D\_zu) then timing:=[0,1,1];

when cmem1&D\_zu then timing:=[0,0,1];

"PC接收指令地址

when cif#cexe&(MVRD#jr\_5) then pc=sum; "sum(PC+1或PC+offset)

when cexe &(JMPA#CALA) "来自内存或运算器

#cmem1&(RET#LDPC) then pc=DB;

} IR\_G=!cif; IR\_clk=CLK; "IR的接收条件和时钟信号

npc.clk=CLK; npc.ce=cexe&CALA; "NPC暂存主程序断点

when cexe&CALA then npc=sum;

ar\_ce=cexe&B\_zu #cmem1&CALA; ar.ce=ar\_ce; "AR 的接收条件

when ar\_ce then ar=DB; ar.clk=CLK; "AR 接收内存地址

AB\_sel=cmem1&B\_zu#cmem2&CALA; "AB 的信息来源选择

when AB\_sel then AB=ar; else AB=pc; "ar或PC

DB.oe=DB\_oe; DB\_oe=cmem2&CALA#cmem1&PSHF#temp&!link1&link0; "DB 接收MACH送来的信息

"要有三态控制

when cmem2&CALA then DB=npc; "存程序断点

when cmem1&PSHF then {[DB15..DB8]=^h00; "存程序状态字

[DB7..DB0]=[flag\_c,flag\_z,0,0,0,0,0,0];}

when temp&!link1&link0 then DB=[inst15..inst0]; "送片内指令到IR

"FLAG 的接收或保持

flag\_c\_ce =cexe&(arith#shift#logic)#cmem1&POPF;

flag\_c\_in =cexe&(SHR&ram0#SHL&ram15#arith&Cy#logic&0)#cmem1&POPF&DB7;

when flag\_c\_ce then flag\_c:=flag\_c\_in "flag\_c接收

else flag\_c:=flag\_c; flag\_c.clk=CLK;

flag\_z\_ce =cexe&(arith#logic) # cmem1&POPF;

when flag\_z\_ce then

flag\_z:=cexe&(arith#logic)&Zero #cmem1&POPF&DB6; "flag\_z接收

else flag\_z:=flag\_z; flag\_z.clk=CLK;

"运算器使用的寄存器编号转换

when([A\_,B\_]==[0,0]) then {[B3..B0]=[IR7..IR4]; [A3..A0]=[IR3..IR0];}

when ([A\_,B\_]==[0,1]) then {[B3..B0]=[0,1,0,0]; [A3..A0]=[0,1,0,0];}

when ([A\_,B\_]==[1,0]) then {[B3..B0]=[0,0,0,0]; [A3..A0]=[0,0,0,0];}

ram15=0; ram15.oe=!I7; "右移指令的最高位输入

ram0=0; ram0.oe = I7; "左移指令的最低位输入

@include 'adder\_pc.abl' "描述指令地址加法器的程序段

TRUTH\_TABLE "选用真值表描述用于控制运算器、内存、接口的信号

([t\_2..t\_0,IR15..IR8]->[aluoe,c0,I8..I6,I5..I3,I2..I0,A\_,B\_,M\_IO,REQ,WE]) "flag

[0,0,0, x,x,x,x,x,x,x,x]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "MEM→IR,PC+1→PC

[0,1,0, 0,0,0,0,0,0,0,0]->[0,0, 0,1,1, 0,0,0, 0,0,1, 0,0, 1,0,0]; "ADD DR+SR→DR c,z,s

[0,1,0, 0,0,0,0,0,0,0,1]->[0,1, 0,1,1, 0,0,1, 0,0,1, 0,0, 1,0,0]; "SUB DR-SR→DR c,z,s

[0,1,0, 0,0,0,0,0,0,1,1]->[0,1, 0,0,1, 0,0,1, 0,0,1, 0,0, 1,0,0]; "CMP DR-SR c,z,s

[0,1,0, 0,0,0,0,0,1,1,0]->[0,0, 0,1,1, 0,1,1, 0,0,1, 0,0, 1,0,0]; "OR DR#SR→DR 0,z,s

[0,1,0, 0,0,0,0,0,0,1,0]->[0,0, 0,1,1, 1,0,0, 0,0,1, 0,0, 1,0,0]; "AND DR&SR→DR 0,z,s

[0,1,0, 0,0,0,0,0,1,0,1]->[0,0, 0,0,1, 1,0,0, 0,0,1, 0,0, 1,0,0]; "TEST DR&SR 0,z,s

[0,1,0, 0,0,0,0,0,1,0,0]->[0,0, 0,1,1, 1,1,0, 0,0,1, 0,0, 1,0,0]; "XOR\_ DR$SR→DR 0,z,s

[0,1,0, 0,0,0,0,1,0,0,0]->[0,0, 0,1,1, 0,0,1, 0,1,1, 0,0, 1,0,0]; "DEC DR-1→DR c,z,s

[0,1,0, 0,0,0,0,1,0,0,1]->[0,1, 0,1,1, 0,0,0, 0,1,1, 0,0, 1,0,0]; "INC DR+1→DR c,z,s

[0,1,0, 0,0,0,0,1,0,1,0]->[0,0, 1,1,1, 0,0,0, 0,1,1, 0,0, 1,0,0]; "SHL DR\*2→DR c / /

[0,1,0, 0,0,0,0,1,0,1,1]->[0,0, 1,0,1, 0,0,0, 0,1,1, 0,0, 1,0,0]; "SHR DR/2→DR c / /

[0,1,0, 0,0,0,0,0,1,1,1]->[0,0, 0,1,1, 0,0,0, 1,0,0, 0,0, 1,0,0]; "MVRR SR→DR

[0,1,0, 0,1,0,0,0,0,0,1]->[0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0]; "JR PC+offset →PC

[0,1,0, 0,1,0,0,0,1,0,0]->[0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0]; "JRC PC+offset?→PC

[0,1,0, 0,1,0,0,0,1,0,1]->[0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0]; "JRNC PC+offset?→PC

[0,1,0, 0,1,0,0,0,1,1,0]->[0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0]; "JRZ PC+offset?→PC

[0,1,0, 0,1,0,0,0,1,1,1]->[0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0]; "JRNZ PC+offset?→PC

[0,1,0, 1,0,0,0,0,0,1,0]->[1,0, 0,1,1, 0,0,0, 1,1,1, 1,0, 0,1,1]; "IN\_ (PORT)→R0

[0,1,0, 1,0,0,0,0,1,1,0]->[0,0, 0,0,1, 0,0,0, 1,0,0, 1,0, 0,1,0]; "OUT R0→(PORT)

[0,1,0, 1,0,0,0,0,0,0,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "JMPA MEM→PC

[0,1,0, 1,0,0,0,1,0,0,0]->[1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1]; "MVRD MEM→DR,PC+1→PC

[0,1,0, 1,1,0,0,1,1,1,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "CALA MEM→PC,PC+1→NPC

[0,1,0, 1,0,0,0,1,0,0,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 1,0,0]; "LDPC SR→AR

[0,1,0, 1,0,0,0,0,0,0,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 1,0,0]; "LDRR SR→AR

[0,1,0, 1,0,0,0,0,0,1,1]->[0,0, 0,0,1, 0,0,0, 0,1,1, 0,0, 1,0,0]; "STRR DR→AR

[0,1,0, 1,0,0,0,0,1,1,1]->[0,1, 0,1,0, 0,0,0, 0,1,1, 0,1, 1,0,0]; "POP sp→AR,sp+1→sp

[0,1,0, 1,0,0,0,1,1,0,0]->[0,1, 0,1,0, 0,0,0, 0,1,1, 0,1, 1,0,0]; "POPF sp→AR,sp+1→sp

[0,1,0, 1,0,0,0,1,1,1,1]->[0,1, 0,1,0, 0,0,0, 0,1,1, 0,1, 1,0,0]; "RET sp→AR,sp+1→sp

[0,1,0, 1,0,0,0,0,1,0,1]->[0,0, 0,1,1, 0,0,1, 0,1,1, 0,1, 1,0,0]; "PUSH sp-1→AR,sp

[0,1,0, 1,0,0,0,0,1,0,0]->[0,0, 0,1,1, 0,0,1, 0,1,1, 0,1, 1,0,0]; "PSHF sp-1→AR,sp

[0,1,1, 1,1,0,0,1,1,1,0]->[0,0, 0,1,1, 0,0,1, 0,1,1, 0,1, 1,0,0]; "CALA sp-1→AR,SP

[0,1,1, 1,0,0,0,0,0,0,1]->[1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1]; "LDRR MEM→DR

[0,1,1, 1,0,0,0,0,1,1,1]->[1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1]; "POP MEM→DR

[0,1,1, 1,0,0,0,1,0,0,1]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "LDPC MEM→PC

[0,1,1, 1,0,0,0,1,1,1,1]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "RET MEM→PC

[0,1,1, 1,0,0,0,1,1,0,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "POPF MEM→flag c,z,s

[0,1,1, 1,0,0,0,0,0,1,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 0,0,0]; "STRR SR→MEM

[0,1,1, 1,0,0,0,0,1,0,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 0,0,0]; "PUSH SR→MEM

[0,1,1, 1,0,0,0,0,1,0,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,0]; "PSHF flag→MEM

[0,0,1, 1,1,0,0,1,1,1,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,0]; "CALA NPC →MEM

"([t\_2..t\_0,IR15..IR8]-> [aluoe,c0,I8..I6,I5..I3,I2..I0,A\_,B\_,MIO,REQ,WE])

TRUTH\_TABLE "16个字的ROM电路,用于编辑、保存调试程序

([pc4..pc0]->[inst15..inst0])

[0,0,0,0,0]->[1,0,0,0,1,0,0,0, 0,0,0,0,0,0,0,0]; "mvrd r0,4e

[0,0,0,0,1]->[0,0,0,0,0,0,0,0, 0,1,0,0,1,1,1,0]; "

[0,0,0,1,0]->[1,0,0,0,0,1,1,0, 1,0,0,0,0,0,0,1]; "out 81

[0,0,0,1,1]->[1,0,0,0,1,0,0,0, 0,0,0,0,0,0,0,0]; "mvrd r0,37

[0,0,1,0,0]->[0,0,0,0,0,0,0,0, 0,0,1,1,0,1,1,1]; "

[0,0,1,0,1]->[1,0,0,0,0,1,1,0, 1,0,0,0,0,0,0,1]; "out 81

[0,0,1,1,0]->[1,0,0,0,1,0,0,0, 0,1,0,0,0,0,0,0]; "mvrd r4,2780

[0,0,1,1,1]->[0,0,1,0,0,1,1,1, 1,0,0,0,0,0,0,0]; "

[0,1,0,0,0]->[1,0,0,0,0,1,1,0, 1,0,0,0,0,0,0,0]; "out 80

[0,1,0,0,1]->[0,0,0,0,0,1,1,1, 0,0,0,1,0,0,0,0]; "mvrr r1,r0

[0,1,0,1,0]->[1,0,0,0,0,0,1,0, 1,0,0,0,0,0,0,1]; "in 81

[0,1,0,1,1]->[0,0,0,0,1,0,1,1, 0,0,0,0,0,0,0,0]; "shr r0

[0,1,1,0,0]->[0,1,0,0,0,1,0,1, 1,1,1,1,1,1,0,1]; "jrnc 000a

[0,1,1,0,1]->[0,0,0,0,0,1,1,1, 0,0,0,0,0,0,0,1]; "mvrr r0,r1

[0,1,1,1,0]->[1,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0]; "jmpa 0008

[0,1,1,1,1]->[0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,0]; "

END

在对MACH芯片的程序进行分析后可以得出，对芯片内部的两种不同类型的电路的描述方法是不同的：  
（1）在描述程序计数器，节拍产生电路以及地址寄存器这些电路的时候，采用了逻辑方程语句进行描述，所谓逻辑方程，就是根据这些电路需要在哪一条指令的哪一个步骤接收从哪里来的信息或是送出信息到何处，来写出它们相应的逻辑方程式。其中，哪一条指令的哪一个执行步骤分别由指令的操作码和节拍编码来决定。

（2）对MACH芯片内部的控制信号产生电路的描述方法则是采用了真值表来进行描述，主要是根据某一条指令的某一个执行步骤来产生用于控制运算器，存储器和串行输入输出接口的控制信号，通过填写真值表中每一行的控制信号的取值来避免去编写控制信号的逻辑方程，这样一来大大简化了工作量，也有利于设计者改正设计中出现的一些错误。

上面这段程序是基于ABEL语言编写的后缀名为.abl的文件，要想把程序下载到芯片中，必须生成可以烧录的后缀名为.jed类型的文件，这里可以用ispLEVER软件来完成。

下面介绍ispLever软件、ispVM System软件的使用方法和操作过程。

ISPLEVER软件是Lattice Semiconductor公司的产品。ISPLEVER软件将器件选择，源文件的建立或导入，源文件的编辑、编译，功能模拟，生成编程文件等诸多功能都集成在工程项目引导器（ISPLEVER Project Navigator）中。引导器帮助用户完成整个设计的全过程。

给出的源文件可以使用文本编辑器进行编辑，假设文件被存放在D/shiyan1文件夹中。

（1）在工程项目引导器中创建一个新工程项目

可以按照以下步骤创建一个新的工程项目，首先运行ISPLEVER软件，并启动工程项目引导器（ISPLEVER Project Nevigator）。

a.文件菜单中，单击新工程项目（New Project）命令；

在ISPLEVER中一个工程项目就是一个设计，每一个工程项目对应一个独立的的目录，它包含所有的源文件、中间的数据文件和结果文件。

对于新启动的工程项目引导器，位于左边的源文件(Sources in Project)窗口中一般仍保存有前一次工程项目文件，在执行a步骤前，使用文件菜单中关闭工程项目（Close Project）命令将其清除。

b.在新创建的工程项目对话框中，选择或新建新的工程项目的保存目录，输入工程项目文件名（\*.syn）或使用默认的工程项目名untitled.syn。从四种工程项目类型（Project type）中ABEL、VHDL、Verilog HDL 、EDIF选择一种，如图-1所示。最后单击“保存”按钮，返回到图-2工程项目引导器窗口。

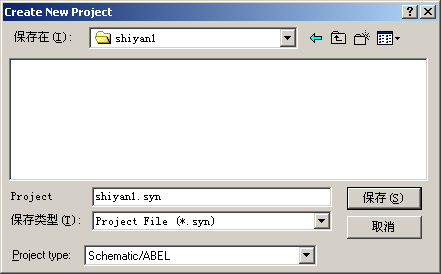
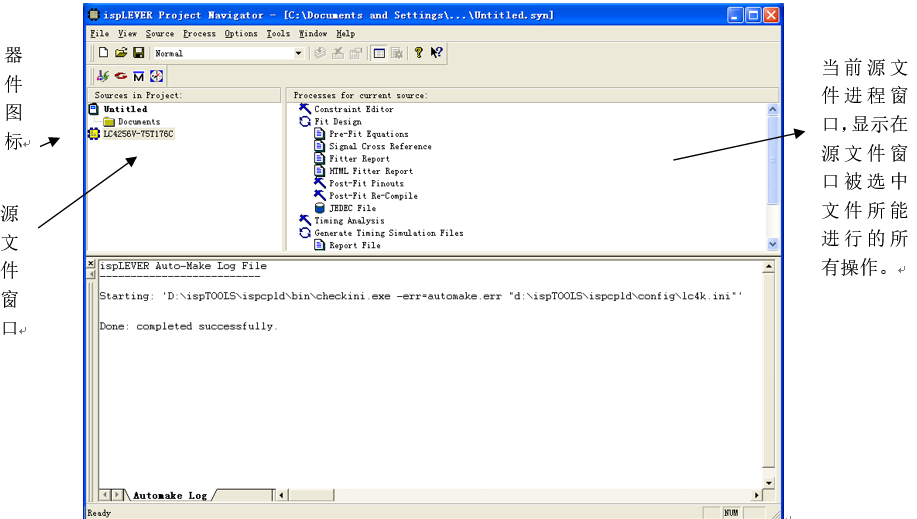
c.双击图-2中的器件图标，出现器件选择(Device Selector)对话框，通过下拉菜单在Family 和 Device两个选项中选择你所使用的器件，其它选项在选好Family 和 Device两个选项后会自动配置，不需选择。我们用的是Lattice LC4256V 器件，因此，Family选项选ispMACH 4000，Device选项选LC4256V。

图-1 创建新工程项目对话框

 图-2 工程项目引导器

导入一个已有的源文件或新建一个源文件

a.在Source菜单中，单击Import命令。出现Import File对话框，双击对话框中的shiyan1.abl文件，则该源文件出现在工程项目引导器源文件窗口中，如图-3所示。

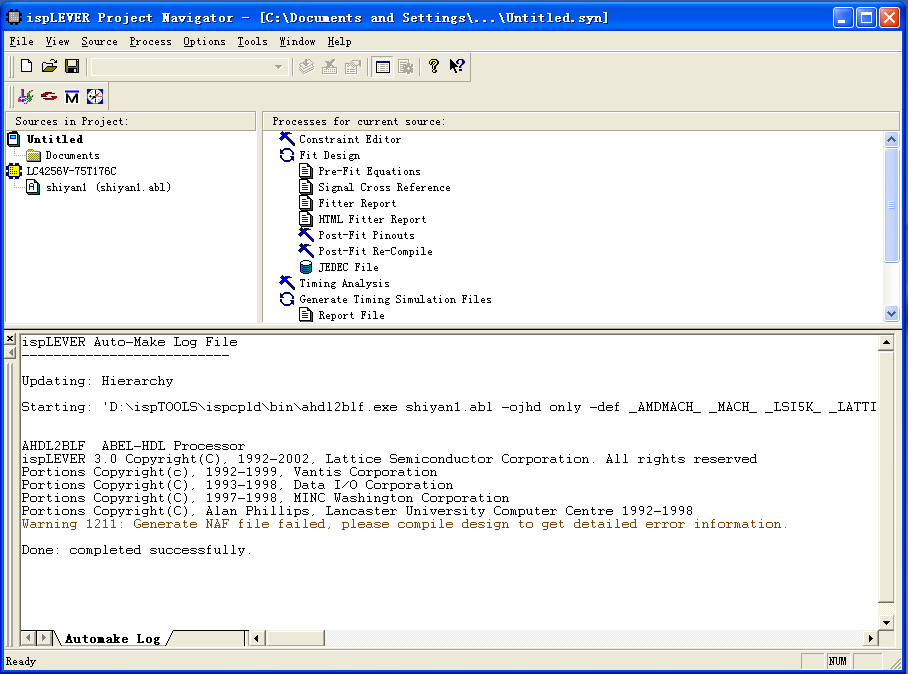
也可使用Source菜单中的New命令，创建一个新的源文件。

图-3 选择已有的或建立新的源文件

b. 双击图3所示的MACH文本文件图标，将运行文本编辑器（Text Editor），被编辑文件的内容将显示在工程项目引导器右侧的文本显示窗口中，接着可以开始编辑操作，编辑结束后需要通过保存选项保存编辑结果，这里的操作过程和屏幕显示内容从略。

（1）编译

单击图3所示的MACH文本文件（shiyan1.abl）图标，则在图3右侧当前源文件进程窗口中将显示Compile Logic任务项，双击该选项则启动对源文件shiyan1.abl编译操作，此项操作只是检查并指出源文件中的语法错误。如有错，就可进入文本编辑器来执行修改操作。如编译通过，系统会在Compile Logic任务项前以绿色的“√”标记。

（2）器件引脚分配

在源文件shiyan1.abl中直接对引脚进行定义。

（3）生成JEDEC编程文件

a. 在图3左侧源文件窗口中点击器件图标，则在图3右侧当前源文件进程窗口中将显示几种可执行的任务项，双击其中的JEDEC File任务项，就启动对原文件的编译和优化操作，这里的编译是针对所选择的器件进行的，要检查并指出器件引脚指定是否有错，芯片资源使用是否全部支持，不合理等会进行检查，如有错，就可进入文本编辑器来执行修改操作。

b. 若无错，结束编译后会在任务项前以绿色的“√”标记。若有警告信息，则以黄色的“!”标记，警告信息通常并不影响JEDEC文件的使用。

以MACH芯片为例讲述.jed类型文件的下载操作方法

启动Lattice Semiconductor公司的ispVM System软件，弹出主界面。

（1）接好教学计算机上在线MACH编程电缆（USB ispDOWNLOAD Cables），打开教学计算机电源。

（2）通过主界面的SCAN按钮找到在线编程器件，如下图-4所示。

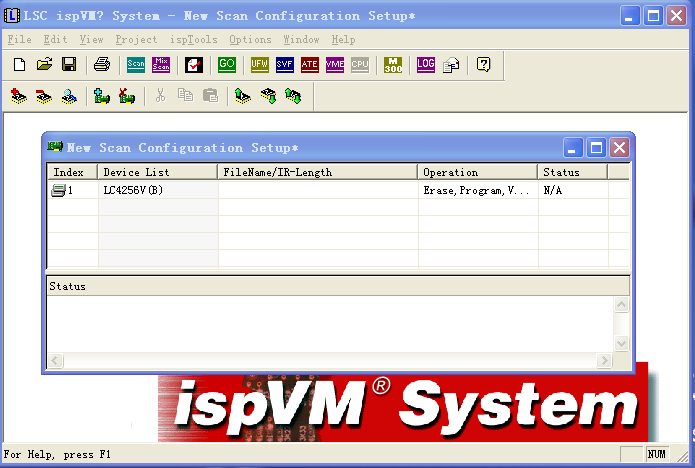


图-4扫描所用的器件型号

（3）双击图-4中所找到的编程器件LC4256V，窗口显示如图-5所示，通过Browse按钮选择已编译好的JED文件（shiyan1.jed），单击OK。

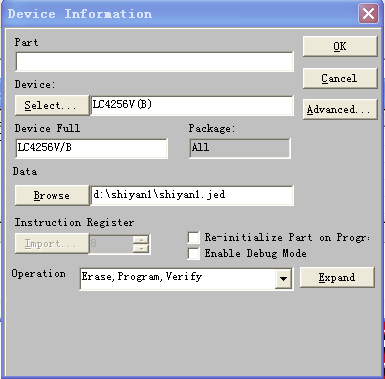


图-5在线编程操作

（4）重新回到主界面，点击go命令启动下载操作。编程完成后，可关闭编程窗口。至此，已经把编好的程序下载到了MACH芯片中。这样便实现了对MACH芯片的编程操作。

至此，就完成了针对MACH芯片的编程操作