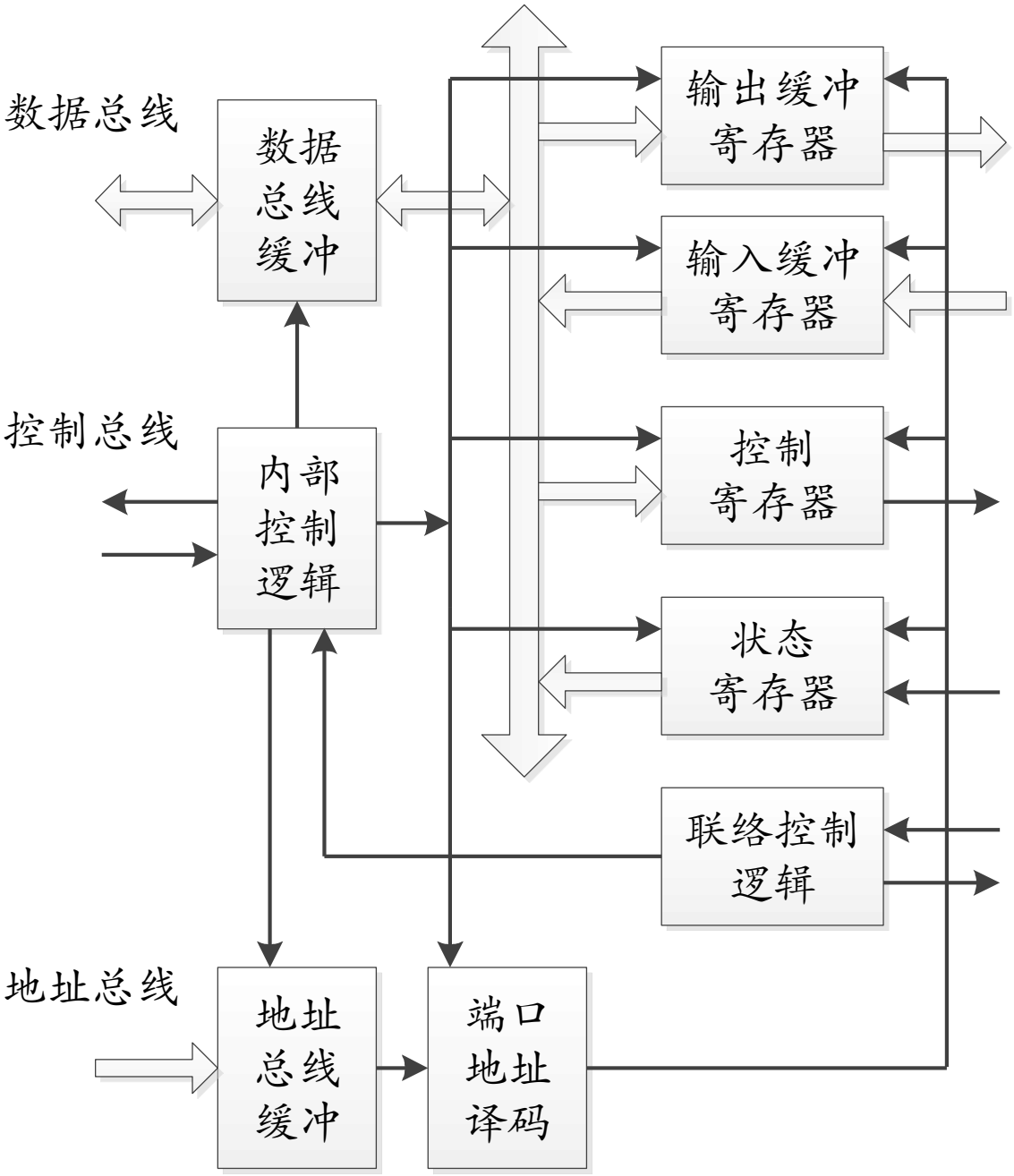
I/O端口地址译码

I/O端口地址译码

设备选择功能是接口电路应具备的基本功能，I/O端口地址译码电路进行设备端口选择，是每个接口电路中不可缺少的部分

1. I/O端口基本概念
2. I/O端口译码基本原理、方法
3. 译码电路的设计


数据总线
数据
总线
缓冲
控制总线
内部
控制
逻辑
地址总线
地址
总线
缓冲
端口
地址
译码
输出缓冲
寄存器
输入缓冲
寄存器
控制
寄存器
状态
寄存器
联络控制
逻辑

地址译码电路

2.1 I/O端口的寻址方式

一、I/O端口：处理器与I/O设备直接通信的寄存器地址。实际应用中，通常把I/O接口（Interface）电路中能被CPU直接访问的寄存器或某些特定器件称之为端口（Port）

【注意】

1. 通过端口向接口电路中的寄存器发送命令，读取状态和传送数据。一个接口可以有几个端口，如命令口、状态口和数据口，分别对应于命令寄存器、状态寄存器和数据寄存器
2. 对端口有不同的操作，有的端口只能写或只能读，有的既可以写也可以读

2.1 I/O端口的寻址方式

【注意】

3. 计算机给接口电路中的每个寄存器分配一个端口（唯一性）
   - CPU在访问这些寄存器时，只需指明它们的端口，不需指出是什么寄存器。访问端口就是访问接口电路中的寄存器
4. I/O的操作是指对I/O端口的操作，而不是对I/O设备的操作
   - CPU所访问的是与I/O设备相关的端口，而不是I/O设备本身

2.1 I/O端口的寻址方式

- I/O端口地址
  - 对接口中的不同寄存器或电路的编号称为I/O端口地址，CPU通过向命令端口发命令来对接口（实际上最终是对设备）进行控制。访问设备实际上是访问相关的端口
- 命令、接口与I/O端口关系
  - 1个接口中有多个I/O端口
  - 1个I/O端口可接受多种命令

2.1 I/O端口的寻址方式

二、端口地址编址方式

对端口有两种编址方式:

1. 存储器映射方式：端口地址和存储器地址统一编址
2. I/O映射方式：I/O端口地址和存储器地址分开独立编址

2.1 I/O端口的寻址方式

1. 统一编址
   - 从存储器空间划出一部分地址空间给I/O设备，把I/O接口中的端口当作存储器单元一样进行访问，不设置专门的I/O指令，一部分对存储器使用的指令也可用于端口
   - Motorola系列、Apple系列微型机和一些小型机采用这种方式

2.1 I/O端口的寻址方式

【优点】

- 使用访问存储器的指令对I/O设备进行访问，所以指令类型多，功能齐全（不仅使访问I/O端口可实现输入/输出操作，而且还可对端口内容进行算术逻辑运算、移位等）
- 能给端口有较大的编址空间，外设数目可以没有限制

【缺点】

- 端口地址占用了存储器的地址空间，使存储器容量减小
- 指令长度比专门I/O指令长，因而执行速度较慢，I/O端口地址译码时间较长

2.1 I/O端口的寻址方式

2. 独立编址
   - 对接口中的端口地址进行单独编址，而不和存储空间合在一起
   - IBM-PC系列微机、Z-80系列机采用这种方式大型计算机通常采用这种方式

2.1 I/O端口的寻址方式

【优点】

- I/O端口地址不占用存储器空间
- 使用专门的I/O指令对端口进行操作，I/O指令短，执行速度快
- 由于专门I/O指令与存储器访问指令有明显的区别，使程序中I/O 操作和存储器操作层次清晰，程序的可读性强

2.1 I/O端口的寻址方式

【缺点】

- 专用I/O指令增加指令系统复杂性，且I/O指令类型少
- 程序设计灵活性较差
- 要求处理器提供MEMR/MEMW和IOR/IOW两组控制信号，增加了控制逻辑的复杂性

2.1 I/O端口的寻址方式

三、在独立编址方式中，如何对端口访问呢?

【PC系列I/O端口访问】

- 由于使用专门的I/O指令访问端口，并且I/O端口地址和存储器地址是分开的，故I/O端口地址和存储器地址可以重叠，而不会相互混淆
- IBM-PC系列采用I/O（Input/Output）指令访问端口，实现数据的I/O传送。在I/O指令中可采用单字节地址或双字节地址寻址方式

【注意】与端口地址的位数有关

2.1 I/O端口的寻址方式

1. I/O端口地址空间
   - I/O端口地址空间
     - 独立编址的8位端口空间（256个）
     - 两个连续8位端口可作为16位端口（64K个）
   - I/O端口地址信号

     借用RAM地址线信号（地址总线）和IOW/IOR信号线组成

2.1 I/O端口的寻址方式

- 若用单字节地址作为端口地址，则最多可访问256个端口
- 系统主板上接口芯片的端口，采用单字节地址，并且是直接寻址，在指令中给出端口地址

【指令格式】

IN　　AL, PORT ;输入

OUT　PORT, AL ;输出

PORT是一个 8 位的字节地址

2.1 I/O端口的寻址方式

- 若用双字节地址作为端口地址，则最多可访问64K个端口
- I/O扩展槽的接口控制卡上的端口，采用双字节地址，间接寻址方式，用寄存器间接给出端口地址（地址总是存放在寄存器DX中）

【指令格式】

```
MOV DX, 300H  ;300H为扩展板8255A的PA端口
IN AL, DX
MOV DX, 301H  ;301H为扩展板8255A的PB端口
OUT DX, AL
```

2.1 I/O端口的寻址方式

2. I/O端口访问
   - 对端口的访问就是CPU对端口的读/写
   - 通常所说的微处理器（CPU）从端口读数据或向端口写数据，仅仅是指I/O端口与CPU的累加器AX之间的数据传送，并未涉及数据是否传送到存储器（RAM）的问题
   - 若要求输入时，将端口的数据传送到存储器，则除了把数据读入CPU的累加器之外，还要将累加器中的数据再传送到内存；或者相反，输出时，数据从存储器先送到CPU的累加器，再从累加器传送到I/O端口

2.1 I/O端口的寻址方式

- 例如：输入时

```
MOV DX, 300H   ;I/O端口
IN AL, DX      ;从端口读数据到AL
MOV [DI], AL   ;将数据从AL→存储器
```

输出时

```
MOV DX, 301H   ;I/O端口
MOV AL, [SI]   ;从内存取数到AL
OUT DX, AL     ;数据从AL→端口
```

2.1 I/O端口的寻址方式

3. 用C语言实现I/O端口访问

【C指令格式】

```
#include <dos.h>
#define PORT …
unsigned char c;
c = inportb(PORT);        //输入
outportb(PORT, c);        //输出
```

PORT是一个8/16位的（双）字节地址

inport, inportb, outport, outportb <DOS.H>

- inport reads a word from a hardware port
- inportb reads a byte from a hardware port
- outport outputs a word to a hardware port
- outportb outputs a byte to a hardware port

Declaration:

- int inport(int portid);
- unsigned char inportb(int portid);
- void outport(int portid, int value);
- void outportb(int portid, unsigned char value);

Remarks:

- inport works just like the 80x86 instruction IN. It reads the low byte of a word from portid, the high byte from portid + 1.
- inportb is a macro that reads a byte
- outport works just like the 80x86 instruction OUT. It writes the low byte of value to portid, the high byte to portid + 1.
- outportb is a macro that writes value

Argument

- portid — Inport port that inport and inportb read from; outport port that outport and outportb write to
- value — Word that outport writes to portid; byte that outportb writes to portid.

- If you call inportb or outportb when DOS.H has been included, they are
- treated as macros that expand to inline code.

- If you don't include DOS.H, or if you do include DOS.H and #undef the
- macro(s), you get the function(s) of the same name.

Return Value:

- inport and inportb return the value read
- outport and outportb do not return

Portability:

- DOS

2.1 I/O端口的寻址方式

【指令格式】

```
#include <conio.h>
#define PORT …
int c;
c = inp(PORT);     //输入
outp(PORT, c);     //输出
```

PORT是一个8/16位的（双）字节地址

inp, inpw, outp, outpw <CONIO.H>

- inp reads a byte from a hardware port
- inpw reads a word from a hardware port
- outp outputs a byte to a hardware port
- outpw outputs a word to a hardware port

Declaration:

- int inp(unsigned portid);
- unsigned inpw(unsigned portid);
- int outp(unsigned portid, int value);
- unsigned outpw(unsigned portid, unsigned value);

Remarks

- Both inp and inpw are macros that read from the input port specified by portid.
- Both outp and outpw are macros that write to the output port specified by portid.

- inp reads a byte
- inpw reads a 16-bit word (the low byte of the word from portid, the high byte from portid + 1)
- outp writes the low byte of value
- outpw writes the low byte of value to portid, and the high byte to portid + 1, using a single 16-bit OUT instruction

- If you call any of these macros when CONIO.H has been included, they are treated as macros that expand to inline code.
- If you don't include CONIO.H, or if you do include CONIO.H and #undef the macro, you get the function of the same name.

Portability

- DOS

Return Value

- inp and inpw return the value read
- outp and outpw return the parameter "value"

2.1 I/O端口的寻址方式

- 【Visual C++指令格式】

```
#include <conio.h>
int _inp( unsigned short port );
unsigned short _inpw( unsigned short port );
unsigned long _inpd( unsigned short port );

int _outp( unsigned short port, int databyte );
unsigned short _outpw( unsigned short port, unsigned short
dataword );
unsigned long _outpd( unsigned short port, unsigned long
dataword );
```

Parameters（输入函数）

- port
- I/O port number.

- Return Value
- The functions return the byte, word, or double word read from port. There is no error return.

Remarks

- The _inp, _inpw, and _inpd functions read a byte, a word, and a double word, respectively, from the specified input port. The input value can be any unsigned short integer in the range 0 – 65,535.
- Because these functions read directly from an I/O port, they might not be used in user code in Windows NT, Windows 2000, Windows XP, and Windows Server 2003. For information about using I/O ports in these operating systems, use the Win32 Communications API.

Parameters（输出函数）

- port
  - Port number.
- databyte, dataword
  - Output values.

Return Value

- The functions return the data output. There is no error return.

Remarks

- The _outp, _outpw, and _outpd functions write a byte, a word, and a double word, respectively, to the specified output port. The *port* argument can be any unsigned integer in the range 0 – 65,535; databyte can be any integer in the range 0 – 255; and dataword can be any value in the range of an integer, an unsigned short integer, and an unsigned long integer, respectively.
- Because these functions write directly to an I/O port, they cannot be used in user code in Windows NT, Windows 2000, Windows XP, and Windows Server 2003. For information about using I/O ports in these operating systems, search for "Serial Communications in Win32" at MSDN.

2.2 I/O端口地址分配

一、I/O接口硬件分类

按照I/O设备的配置情况和复杂程度，I/O接口的硬件分成两类：

1. 系统板上的I/O芯片
   - 多为大规模集成电路，如定时/计数器，中断控制器，DMA控制器，并行接口
2. I/O扩展槽上的接口控制卡
   - 由若干个集成电路按一定的逻辑组成的一个部件，例如，显卡，声卡，软驱卡，硬驱卡，打印卡，串行通讯卡
   - 处于不断的发展变化中

2.2 I/O端口地址分配

二、 I/O端口地址分配

- 不同的微机系统对I/O端口地址的分配不同
- PC微机是根据上述I/O接口的硬件分类，把I/O 空间分成两部分

2.2 I/O端口地址分配

- PC微机I/O地址线可有16根，对应的I/O端口编址可达64K字节
- IBM公司设计微机主板及规划接口卡时（PC/XT），端口地址译码是采用非完全译码方式
  - 只考虑低10位地址线A0~A9，而没有考虑高6位地址线A10~A15，故I/O端口地址范围是000H~003FFH，总共只有1024个端口
  - 前512个端口分配给了主板，后512个端口分配给了扩展槽上的常规外设

2.2 I/O端口地址分配

- 在PC/AT（Advanced Technology）系统中作了一些调整
  - 前256个端口（000~0FFH）供系统板上的I/O接口芯片使用
  - 后768（100~3FFH）供扩展槽上的I/O接口控制卡使用

【背景】

- PC/XT（1983）CPU8088，寻址1M，扩展槽62脚，4.77MHz，8/16位数据线
- PC/AT（1986）CPU80286，扩展槽62+36脚ISA（Industry Standard Architecture），寻址16M，16位数据线

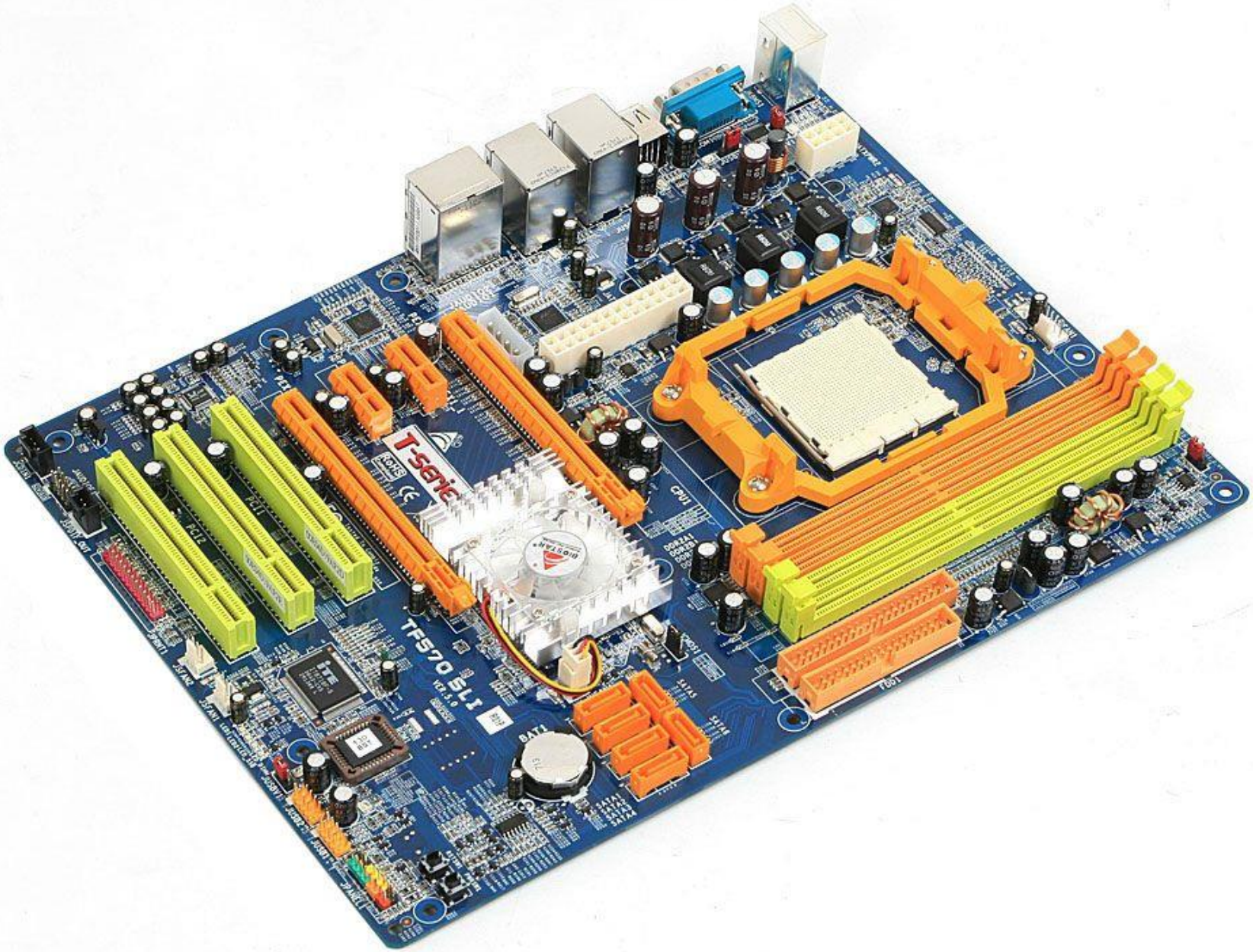
  CPU80386，EISA（Expanded-ISA），寻址16M/4G，32位数据线

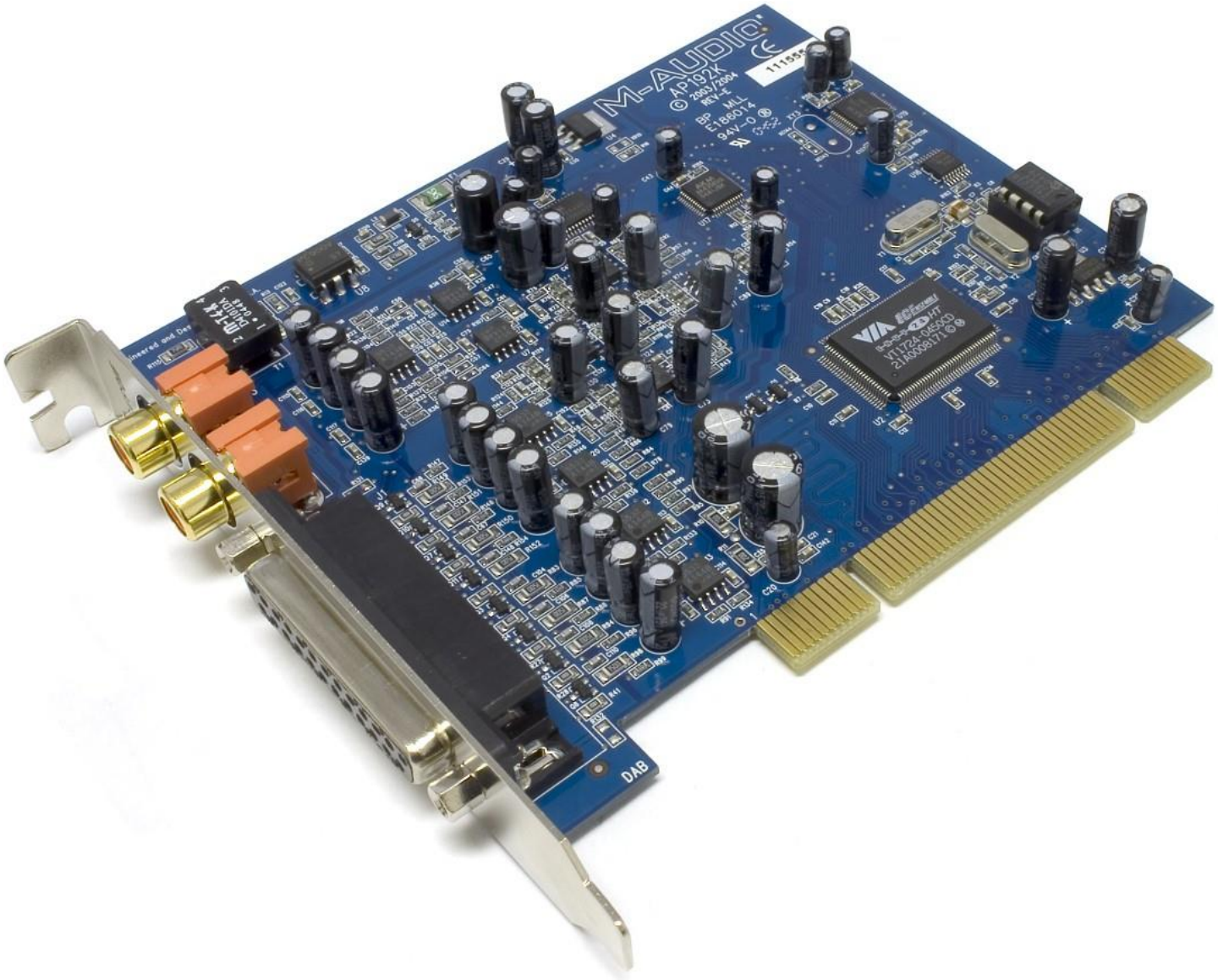
系统板上接口芯片的端口地址

I/O芯片名称	端口地址
DMA控制器1	000 – 01FH
DMA控制器2	0C0 – 0DFH
DMA页面寄存器	080 – 09FH
中断控制器1	020 – 03FH
中断控制器2	0A0 – 0BFH
定时器	040 – 05FH
并行接口芯片（键盘接口）	060 – 06FH
RT/CMOS RAM	070 – 07FH
协处理器	0F0 – 0FFH

扩展槽上接口控制卡的端口地址

I/O芯片名称	端口地址
游戏控制卡	200 – 20FH
并行口控制卡1 并行口控制卡2	370 – 37FH 270 – 27FH
串行口控制卡1 串行口控制卡2	3F8 – 3FFH 2F0 – 2FFH
原型插件板（用户可用）	300 – 31FH
同步通信卡1 同步通信卡2	3A0 – 3AFH 380 – 38FH
单显MDA 彩显CGA 彩显EGA/VGA	3B0 – 3BFH 3D0 – 3DFH 3C0 – 3CFH
硬驱控制卡 软驱控制卡	1F0 – 1FFH 3F0 – 3F7H
PC网卡	360 – 36FH


T-serie
TF570 SLI
VER:1.0
BAT1
CPU1
PCI1
PCI2
PCIE1
PCIE16
SATA1
SATA2
SATA3
SATA4
SATA5
SATA6


M-AUDIO
AP192
DAB


www.segalaxy.com

2.2 I/O端口地址分配

三、I/O用端口地址选用的原则

只要设计I/O接口电路，就必然要使用I/O端口地址，为了避免端口地址发生冲突，在选用I/O端口地址时要注意：

1. 凡是被系统配置所占用了的地址一律不能使用
2. 原则上讲，未被占用的地址，用户可以使用
   - 对计算机厂家申明保留的地址，不要使用，否则，会发生I/O端口地址重叠和冲突，造成用户开发的产品与系统不兼容而无法使用

2.2 I/O端口地址分配

3. 用户可使用300~31FH地址，这是IBM-PC微机留作实验卡用的，用户可以使用
   - 由于每个用户都可以使用，所以在用户可用的这段I/O地址范围内，为了避免与其他用户开发的插板发生地址冲突，最好采用地址开关

2.3 I/O用端口地址译码

- 如何把来自地址总线的地址代码翻译成所要访问的端口（寄存器）？
  - CPU为了对I/O端口进行读写操作，就需确定与自己交换信息的端口（寄存器）。通过I/O地址译码电路把来自地址总线上的地址代码翻译成所需要访问的端口（寄存器）

2.3 I/O用端口地址译码

一、I/O地址译码电路的工作原理

1. 译码电路的输入信号

- I/O地址译码电路不仅仅与**地址信号**有关，而且与**控制信号**（例如/CS，AEN（DMA），/IOR，/IOW等）有关
- I/O端口地址译码电路的作用是把地址和控制信号进行**逻辑组合**，从而产生对接口芯片的选择信号

2.3 I/O用端口地址译码

2. 译码电路的输出信号

- 译码电路把输入的地址线和控制线经过逻辑组合后，所产生的输出信号线就是1根选中线，低电平有效
- 若译码电路的输出线为低，则表示译码有效；若输出线为高，则译码无效

2.3 I/O用端口地址译码

CPU利用译码电路来选择与之交换信息的接口电路

- 打开接口电路与系统总线的通路

  I/O地址译码有效 → 选中一个接口芯片 → 芯片内部的数据线打开，与系统总线相连

- 关闭了接口电路与系统总线的通路

  I/O地址译码无效 → 未选中该接口芯片 → 芯片内部呈高阻（抗）态，与系统总线隔离

2.3 I/O用端口地址译码

二、 I/O地址译码的方法

I/O端口地址译码的方法灵活多样，可按**地址信号**和**控制信号**不同的组合去进行译码。

一般原则是把地址线分为两部分：

1. **高位地址线**与**CPU的控制信号**进行组合，经译码电路产生I/O接口芯片的片选CS信号，实现系统中的**片间寻址**；
2. **低位地址线**不参加译码，直接连到I/O接口芯片，进行I/O接口芯片的**片内端口寻址**，即**寄存器寻址**

2.3 I/O用端口地址译码

三、 I/O端口地址译码电路设计

- 按译码电路的形式可分为
  - 固定式译码（接口卡采用，单端口/多端口）
    （门电路、译码器）
  - 可选式译码（系统版/接口卡，可扩充适应不同场合）
    （地址开关、译码器、比较器、异或门等）
- 按译码电路采用的元器件可分为
  - 门电路译码
  - 译码器译码
  - 可编程逻辑器件译码（FPGA/CPLD）

2.3 I/O用端口地址译码

1. 固定式端口地址译码

   （接口中用到的端口地址不能更改）

   - 接口卡中一般是采用固定式译码
   - 在固定式译码电路中，又分单个端口地址译码和多个端口地址译码两种情况

若仅需一个端口地址，可以采用门电路构成译码电路

2.3 I/O用端口地址译码

例1　使用74LS20/30/32和74LS04

74LS20　两个4输入与非门

74LS30　单个8输入与非门

74LS32　四个2输入或门

74LS04　六个反相器

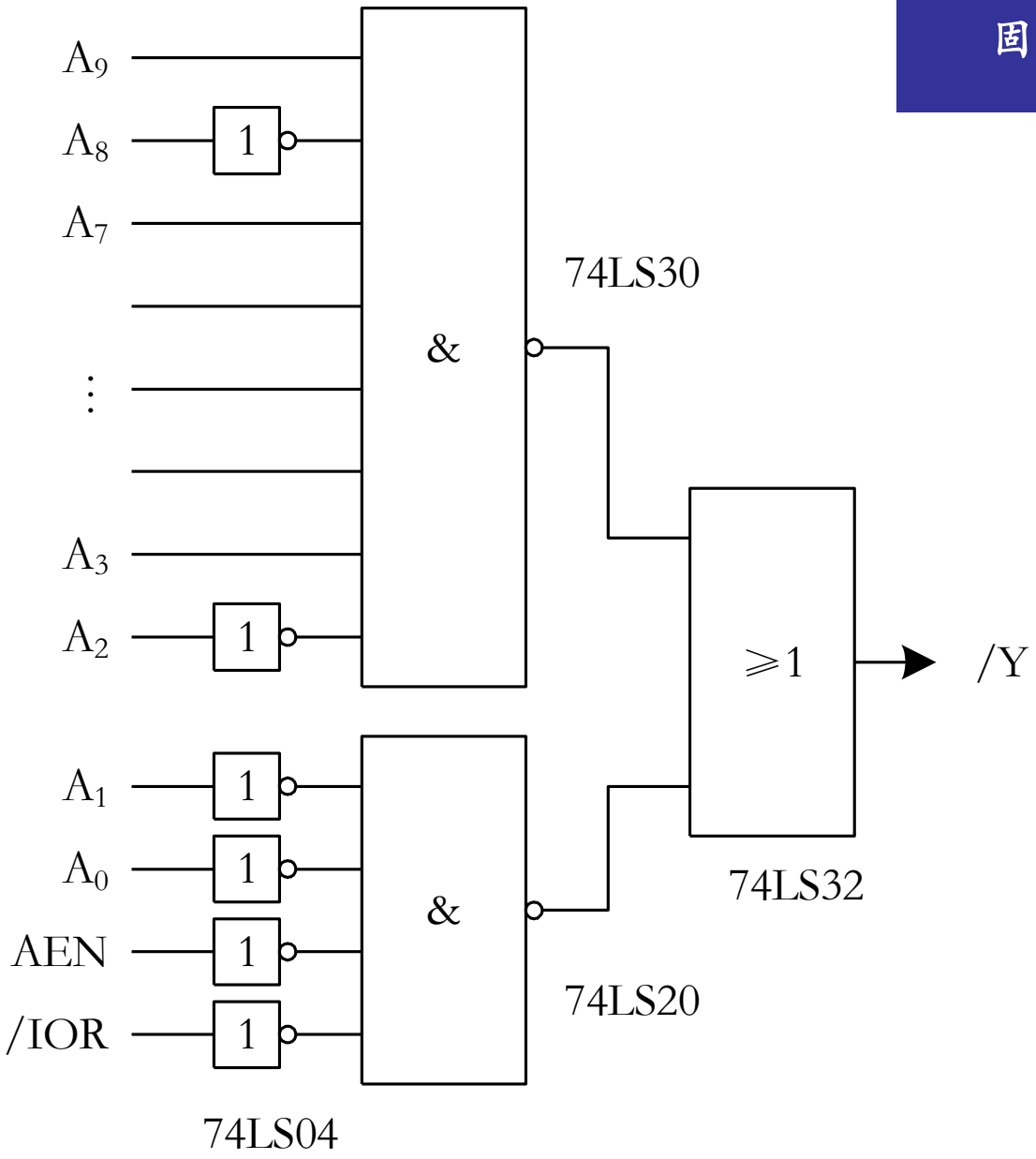
- 设计I/O端口地址为2F8H的只读译码电路
  $00A_9A_8 \sim A_0$ = 2F8H = 0010 1111 1000

- 设计I/O端口地址为2E2H的读/写译码电路
  $00A_9A_8 \sim A_0$ = 2E2H = 0010 1110 0010

译码电路输入地址线的值

地址线	$00A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$
二进制	0010	1111	1000
十六进制	2	F	8

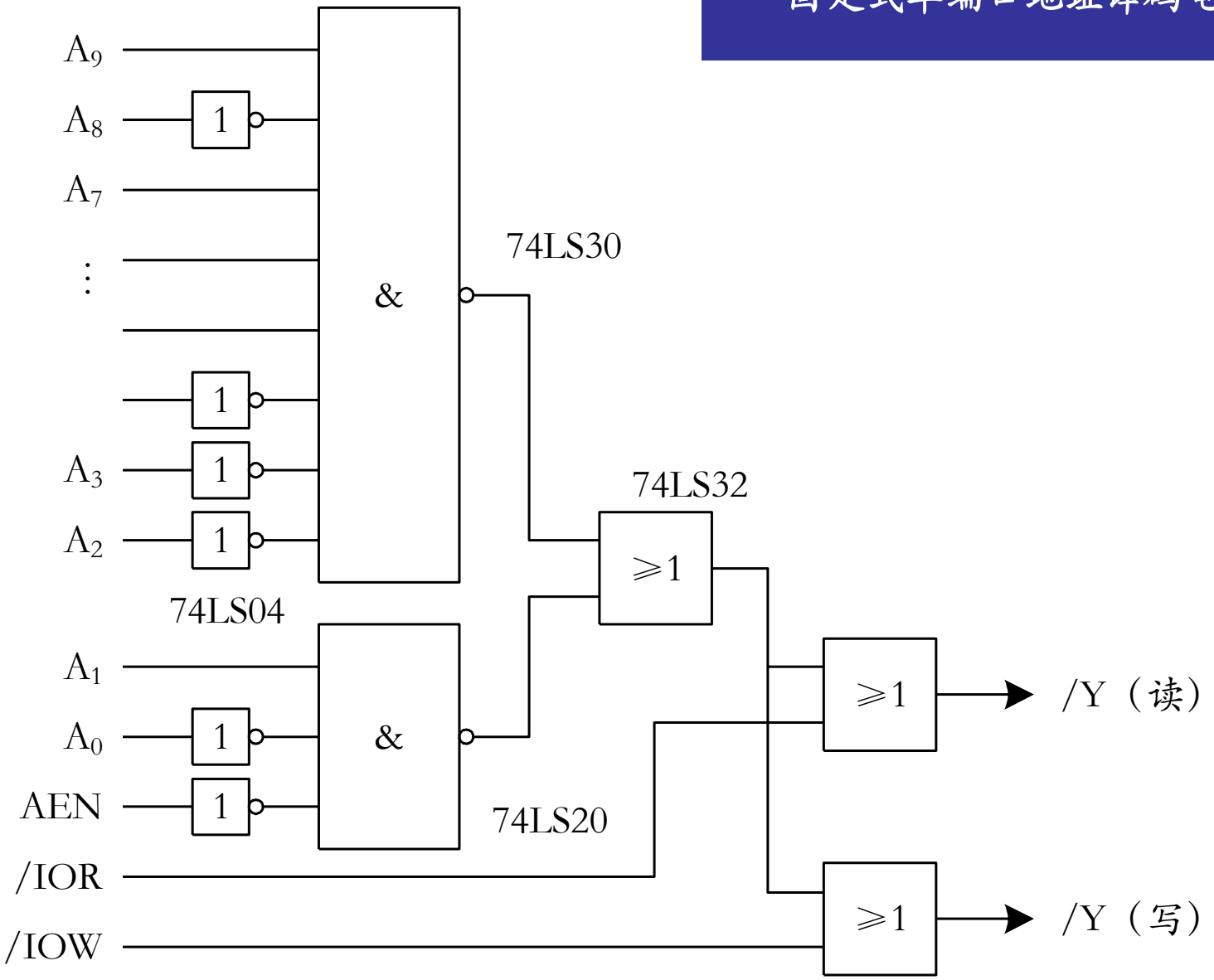
固定式单端口地址译码电路



译码电路输入地址线的值

地址线	$00A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$
二进制	0010	1110	0010
十六进制	2	E	2

固定式单端口地址译码电路



2.3 I/O用端口地址译码

例2 使用74LS138设计一个系统板上接口芯片的I/O端口地址译码电路，并让接口芯片内部的端口数位32个。

- 74LS138　3-8线译码器
- 系统板上I/O端口地址分配在000~0FFH，只使用低8位地址线

  $00A_9A_8 \sim A_0 = 0000A_7A_6A_5A_4A_3A_2A_1A_0$

$A_9A_8 = 00$，用于控制

$A_7A_6A_5$片选（0~7H）

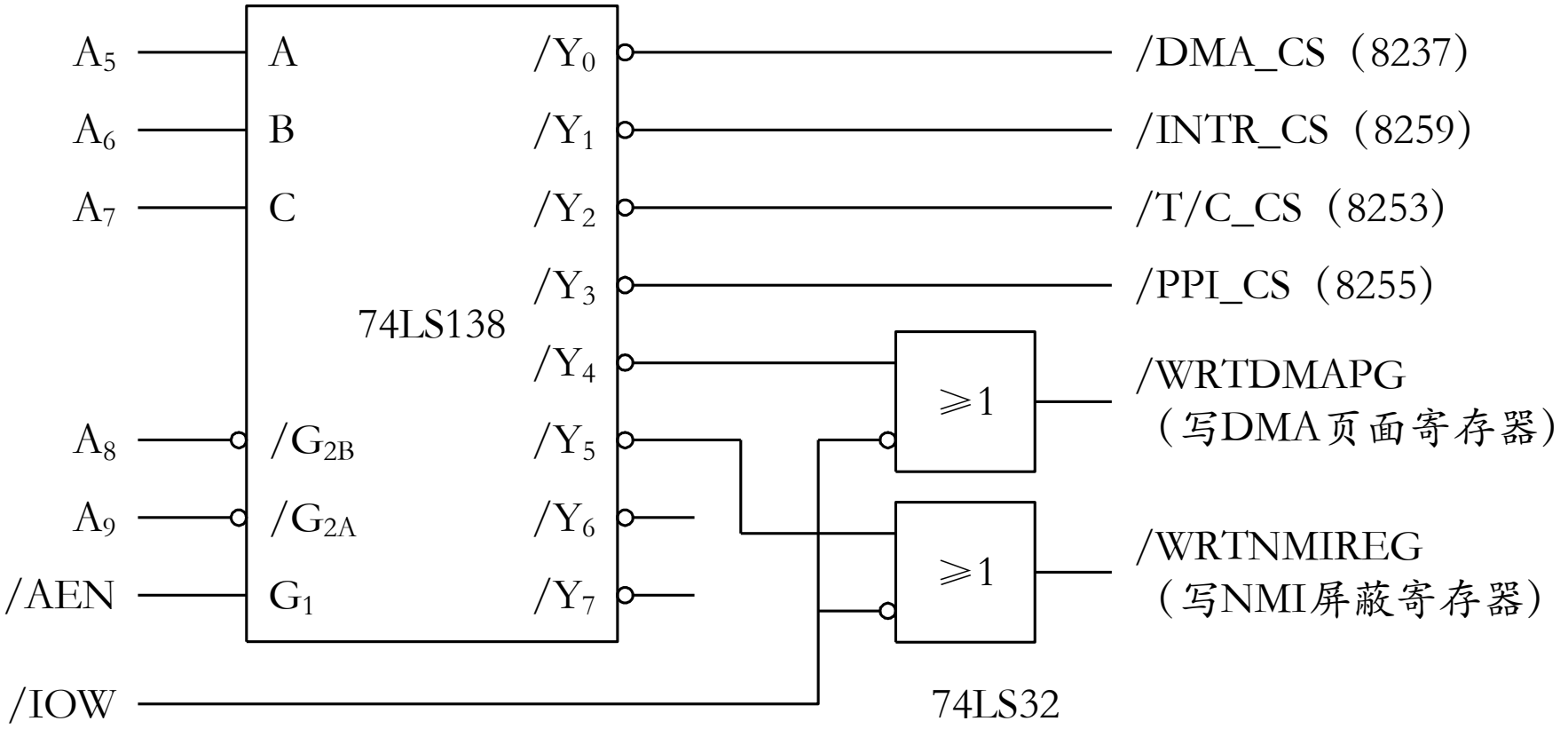
$A_4A_3A_2A_1A_0$片内端口寻址（0~1FH）

译码电路输入地址线的值

地址线	$00A_9A_8$	$A_7A_6A_5$	$A_4A_3A_2A_1A_0$
用途	控制	片选	片内端口寻址
十六进制	0H	0-7H	0-1FH

74LS138译码器的真值表

输 入		输 出
G_1 $\overline{G_{2A}}$ $\overline{G_{2B}}$	C B A	$\overline{Y_7}$ $\overline{Y_6}$ $\overline{Y_5}$ $\overline{Y_4}$ $\overline{Y_3}$ $\overline{Y_2}$ $\overline{Y_1}$ $\overline{Y_0}$
1 0 0	0 0 0	1 1 1 1 1 1 1 0
1 0 0	0 0 1	1 1 1 1 1 1 0 1
1 0 0	0 1 0	1 1 1 1 1 0 1 1
1 0 0	0 1 1	1 1 1 1 0 1 1 1
1 0 0	1 0 0	1 1 1 0 1 1 1 1
1 0 0	1 0 1	1 1 0 1 1 1 1 1
1 0 0	1 1 0	1 0 1 1 1 1 1 1
1 0 0	1 1 1	0 1 1 1 1 1 1 1
0 × × × 1 × × × 1	× × × × × × × × ×	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1



固定式多端口地址译码电路

2.3 I/O用端口地址译码

2. 可选式端口地址译码

采用开关式端口地址译码（可以通过开关使接口卡的I/O端口地址根据要求加以改变而无需改动线路）

- 用户要求接口卡的端口地址能适应不同的地址分配场合
- 为系统以后扩充留有余地

例3 由地址开关、译码器、比较器或异或门几种元器件组成可选式端口地址译码。

2.3 I/O用端口地址译码

例3 设计扩展板上的I/O端口地址译码电路，要求让扩展板上每个接口芯片的内部端口数目为4个，且端口地址可选。例如，选择地址范围为300H~31FH（用户原型板）。

- 地址开关、比较器和译码器3个元器件的工作原理
- 300H~31FH：11 0000 0000 ~ 11 0001 1111

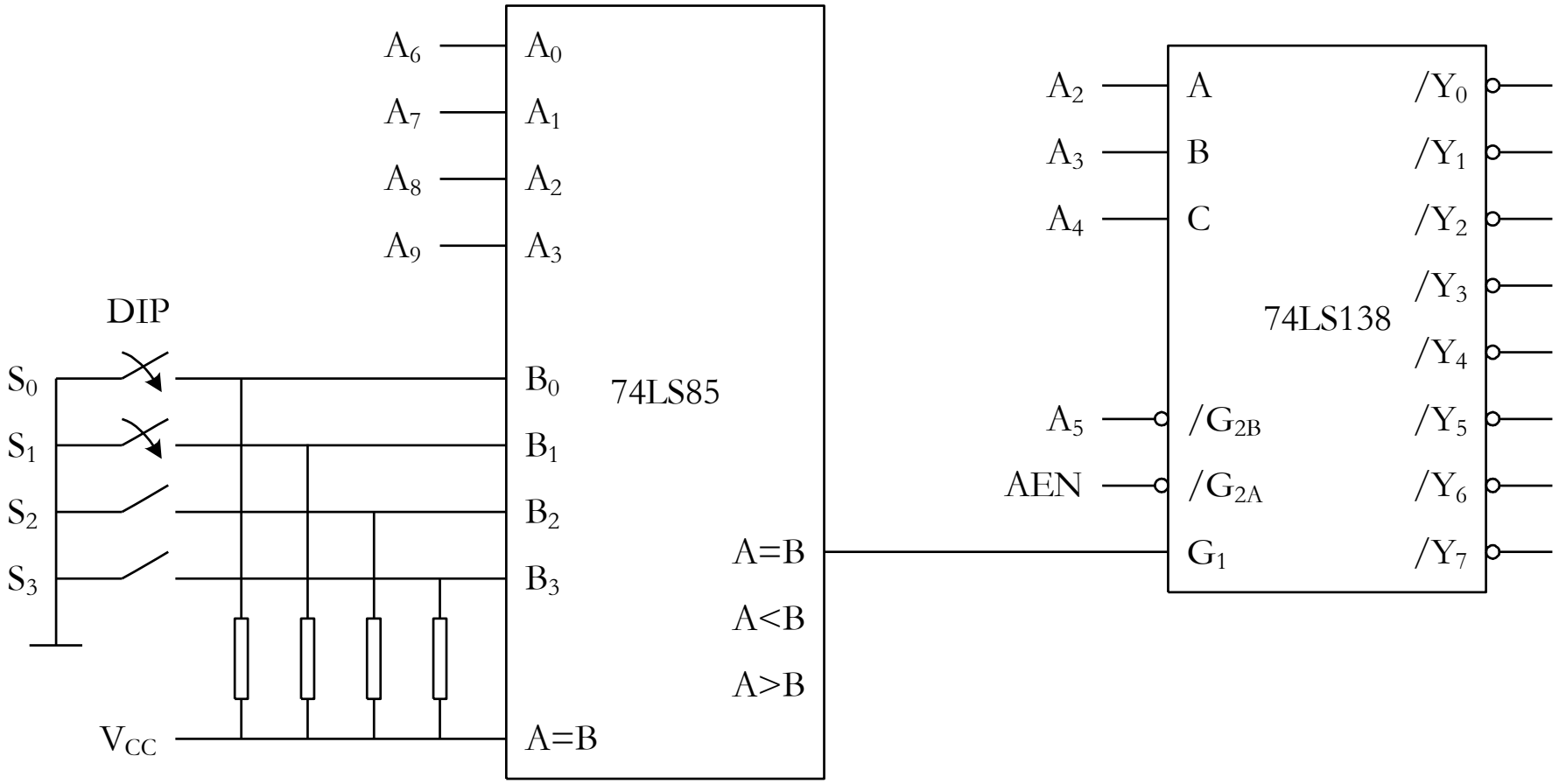
$A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$ = 11000$A_4A_3A_2A_1A_0$（$A_{9\sim6}$不变，但可以用开关设置）

$A_9A_8\,A_7A_6$ = 1100（地址开关控制）

A_5 = 0（不变，不能设置）

$A_4A_3A_2$片选（0~7H）

A_1A_0片内端口寻址（0~3H）



用比较器组成的可选式译码电路

74LS138译码器的真值表

输 入			输 出							
G_1	$\overline{G_{2B}}$	$\overline{G_{2B}}$	C	B	A	$\overline{Y_7}$	$\overline{Y_6}$	$\overline{Y_5}$	$\overline{Y_4}$	$\overline{Y_3}$	$\overline{Y_2}$	$\overline{Y_1}$	$\overline{Y_0}$
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
0	×	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	×	1	1	1	1	1	1	1	1
×	×	1	×	×	×	1	1	1	1	1	1	1	1

2.3 I/O用端口地址译码

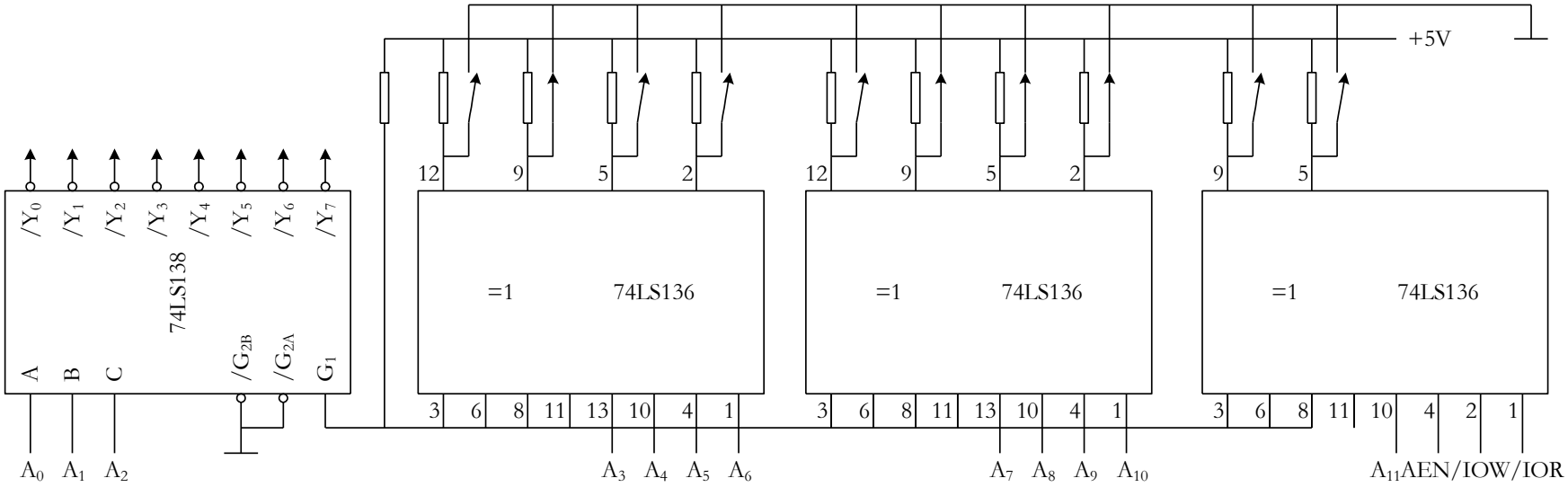
例4 采用异或门设计I/O端口地址译码电路，分析如图译码电路的功能。

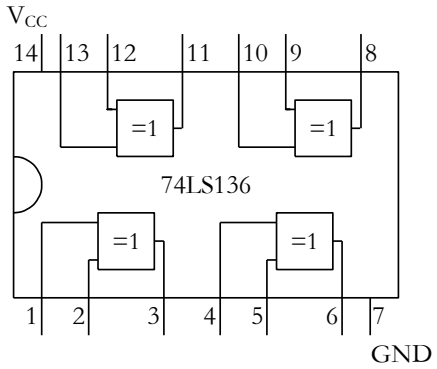
- 由3片异或门74LS136（芯片内部有4个异或门），9位DIP开关和译码器74LS138组成。
- 异或运算相当于二进制加法

  $Ai \oplus 0 = Ai$，$Ai \oplus 1 = /Ai$
- $A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3$ = 0111 0001 0
  （地址开关控制）
- $A_2A_1A_0$地址译码（0~7H）

译码电路输入地址线的值

地址线	$A_{11}A_{10}A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$
二进制值	0111 …… 0111	0001 …… 0001	0000 …… 0111





$$A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3 = 0111\ 0001\ 0$$

用异或门组成的可选式译码电路

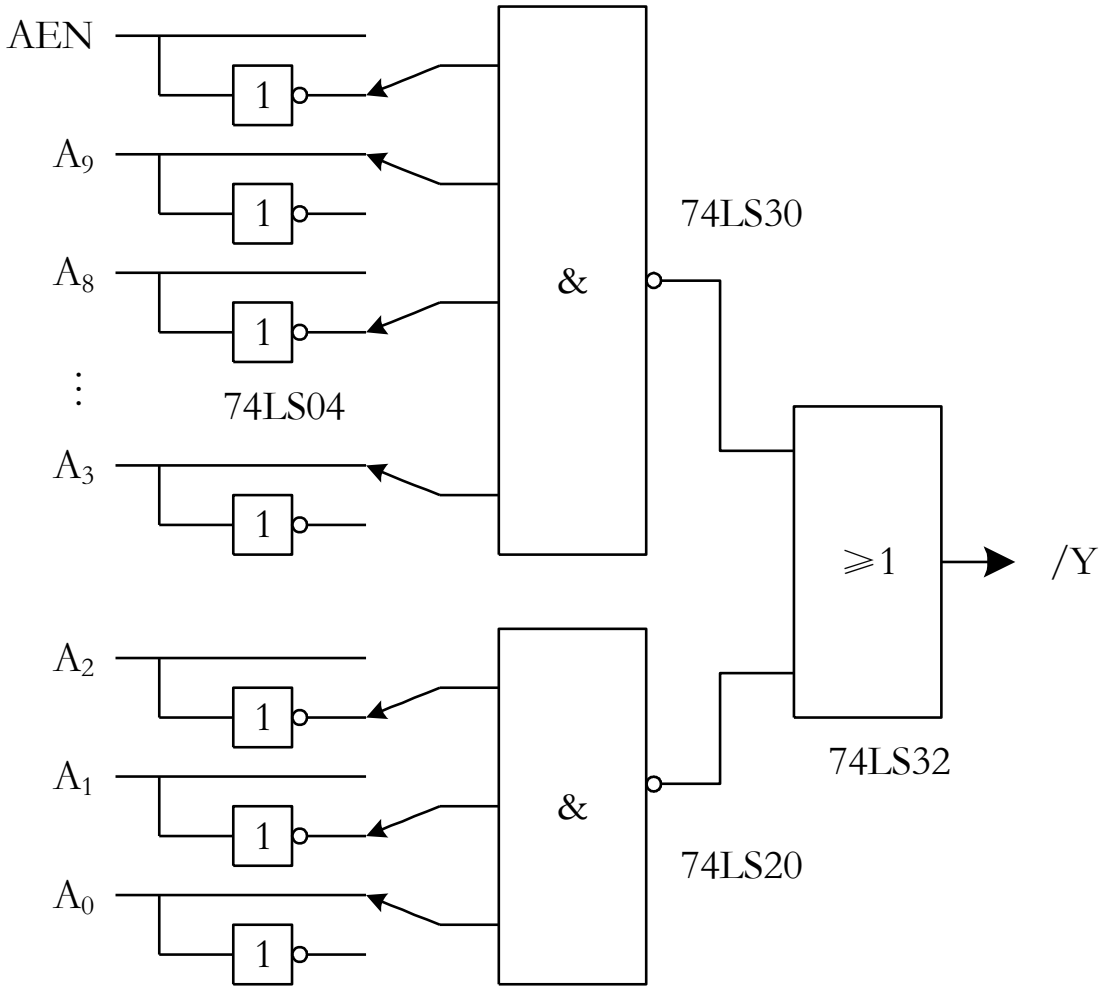
2.3 I/O用端口地址译码

例5 采用跳接开关设计I/O端口地址译码电路。

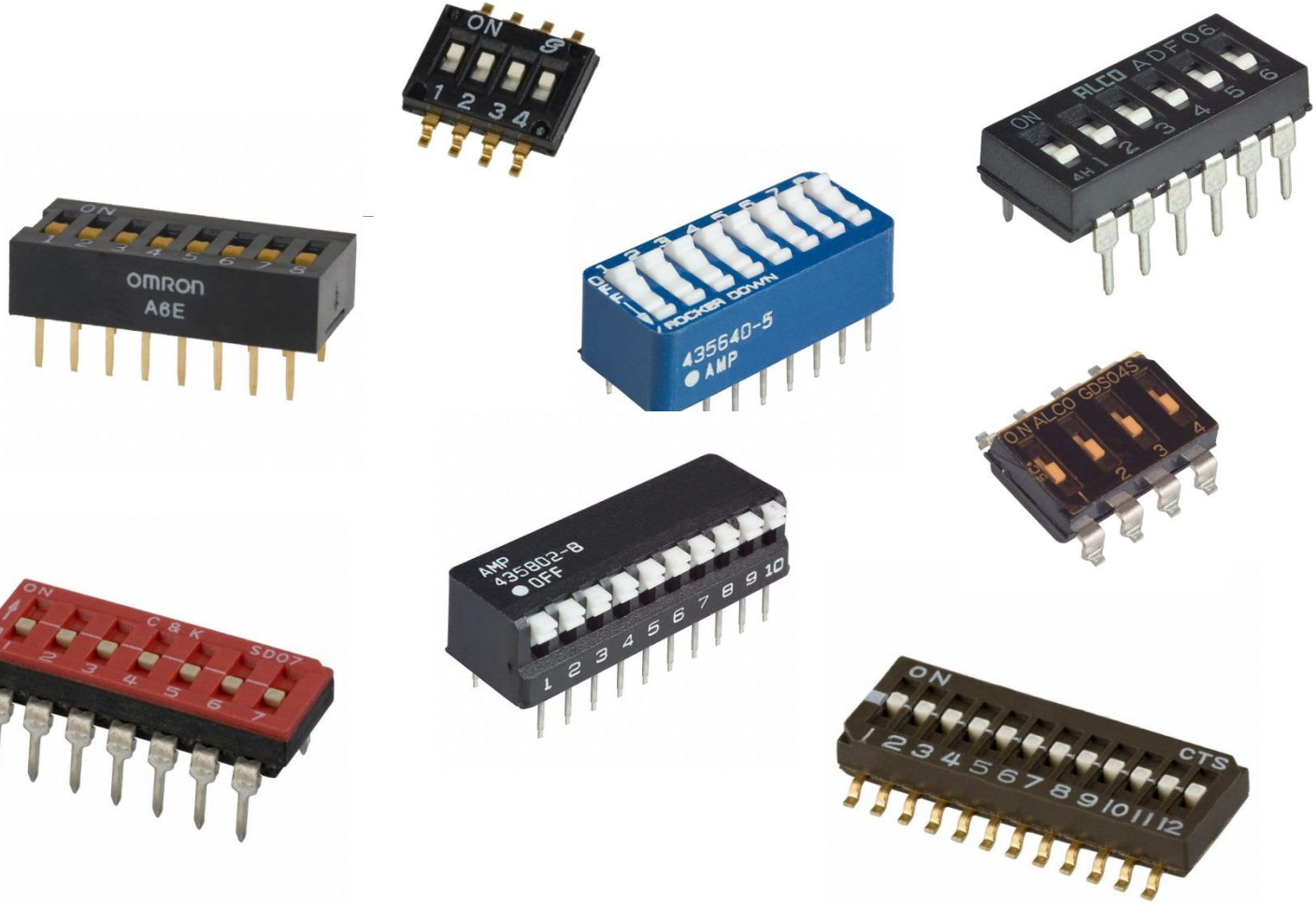
用跳接开关代替DIP开关

- 74LS30　　单个8输入与非门

  74LS20　　两个4输入与非门
- $A_9A_8\cdots A_0$共1024种选择
- 单端口地址译码

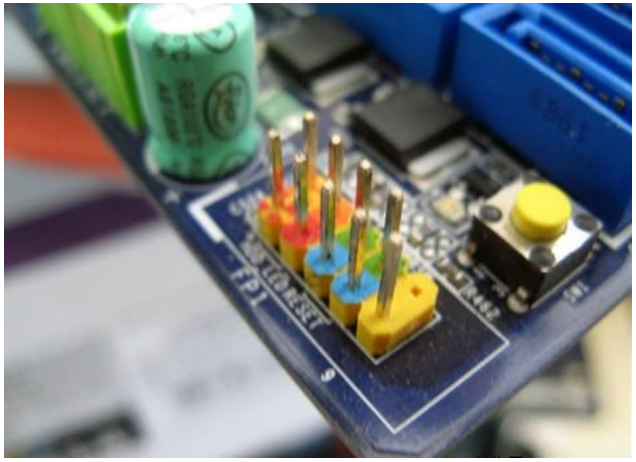


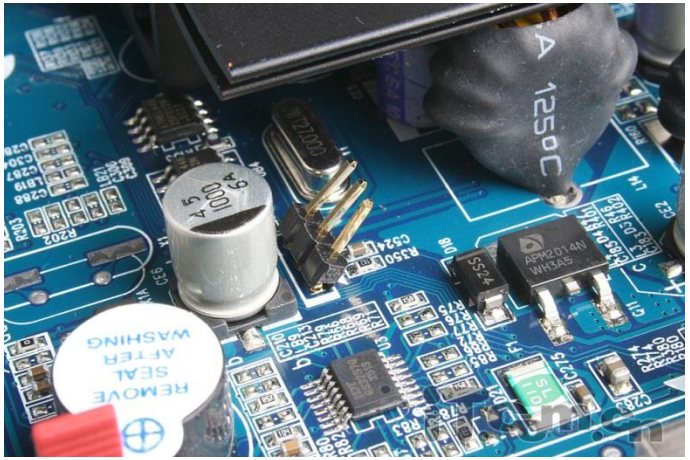
跳接开关可选式译码电路

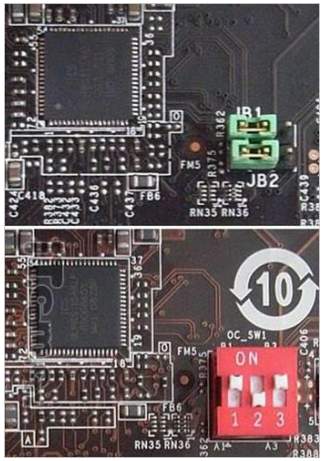


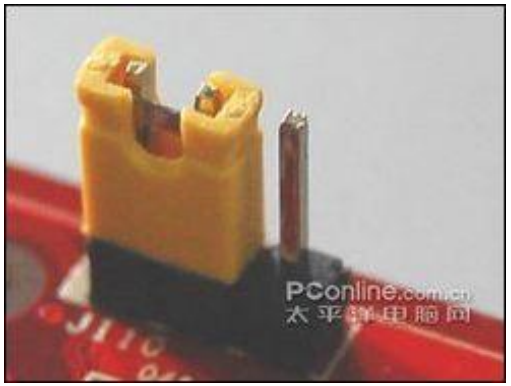
地址开关采用DIP Switch

地址开关采用跳线（Jumper）









2.4 FPGA在I/O地址译码中的应用

- 可编程逻辑器件在I/O地址译码中的应用
  - GAL（Generic Array Logic）
  - FPGA （Field Programmable Gate Array）
  - CPLD（Complex Programmable Logic Device）
- 硬件描述语言
  （Hardware Description Language）
  - Verilog HDL
  - VHDL