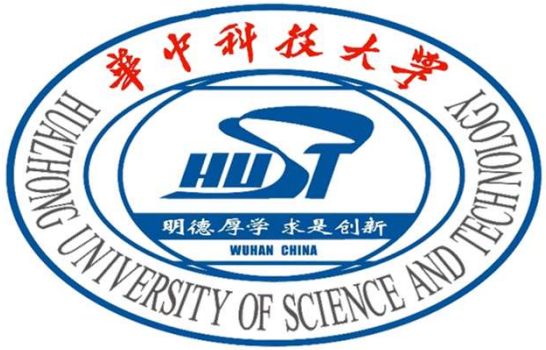
微机原理与接口技术

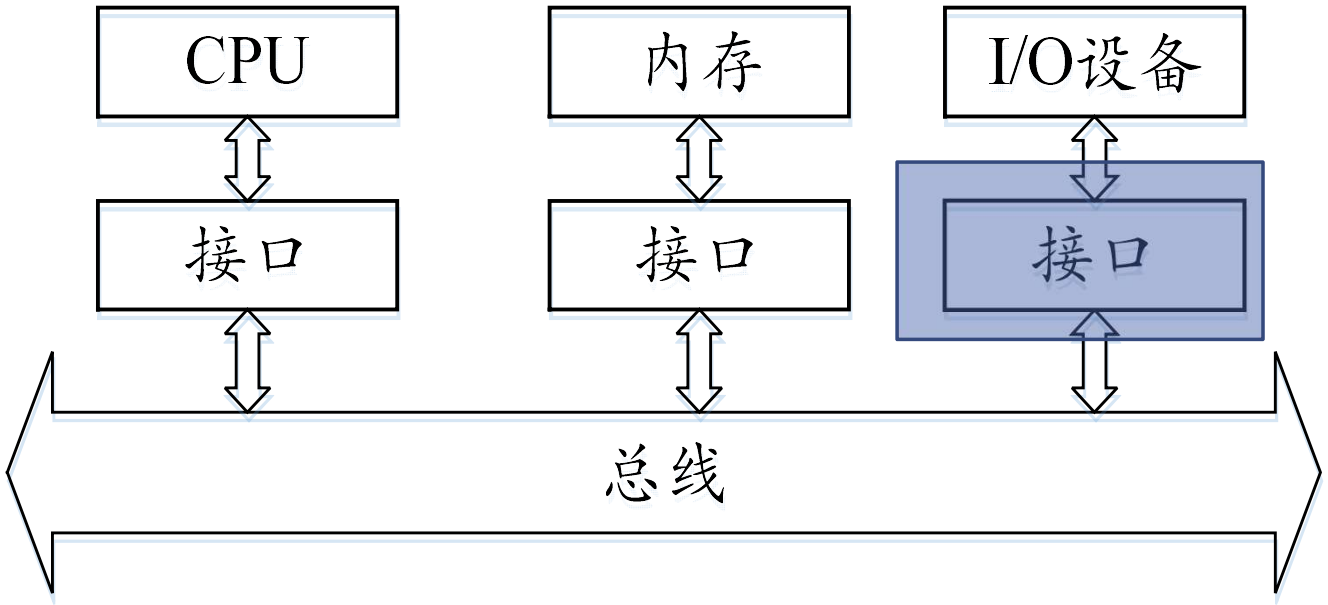
IO接口基础

华中科技大学　左冬红



接口定义

将设备特有的信号转换为适合在总线上传输的信号，并且实现两者之间速度匹配的电路



接口功能

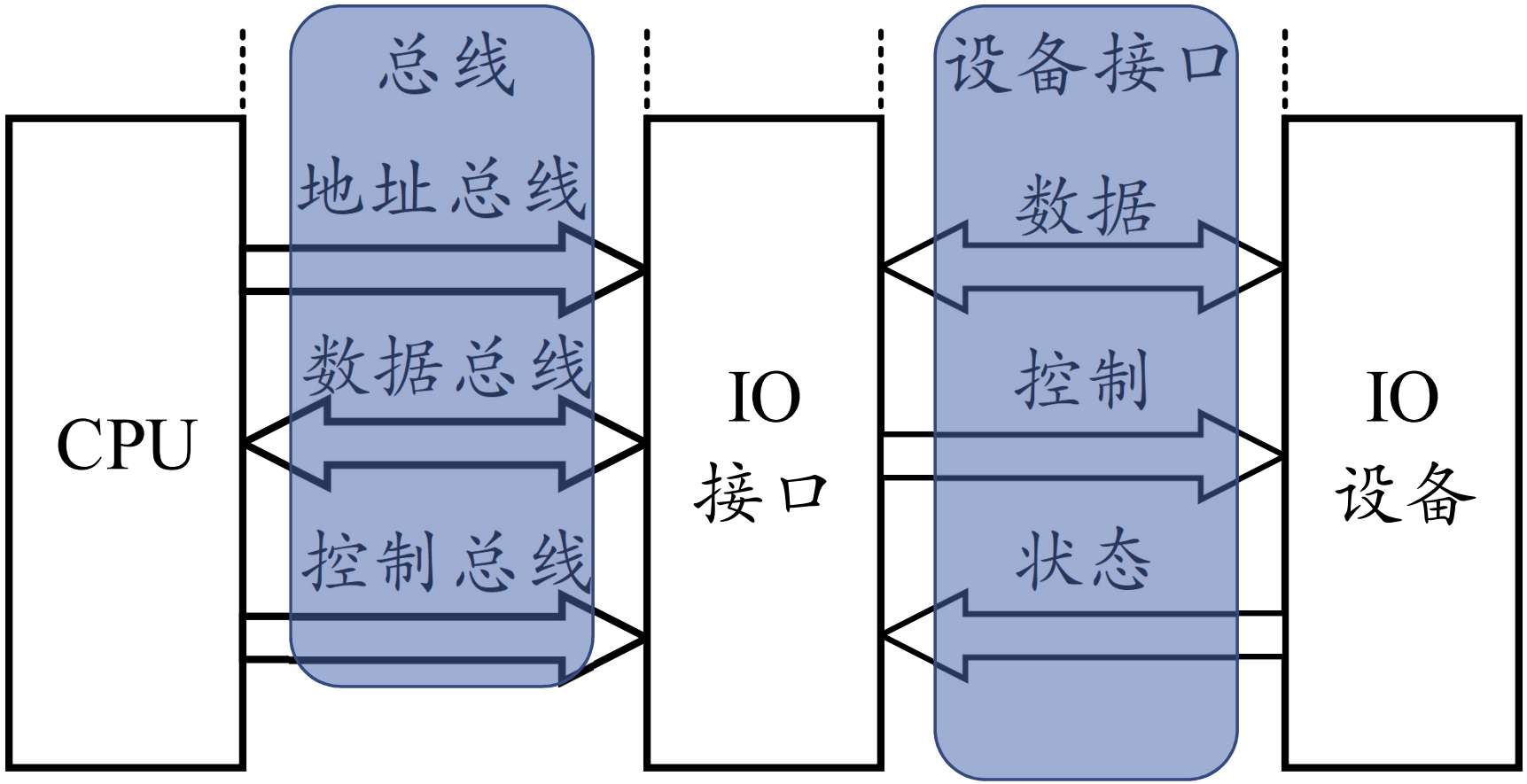
接口的基本功能是对数据传送实现控制

(1) 控制和定时
(2) 通过总线与微处理器通信
(3) 与从设备通信
(4) 数据缓冲
(5) 错误检测

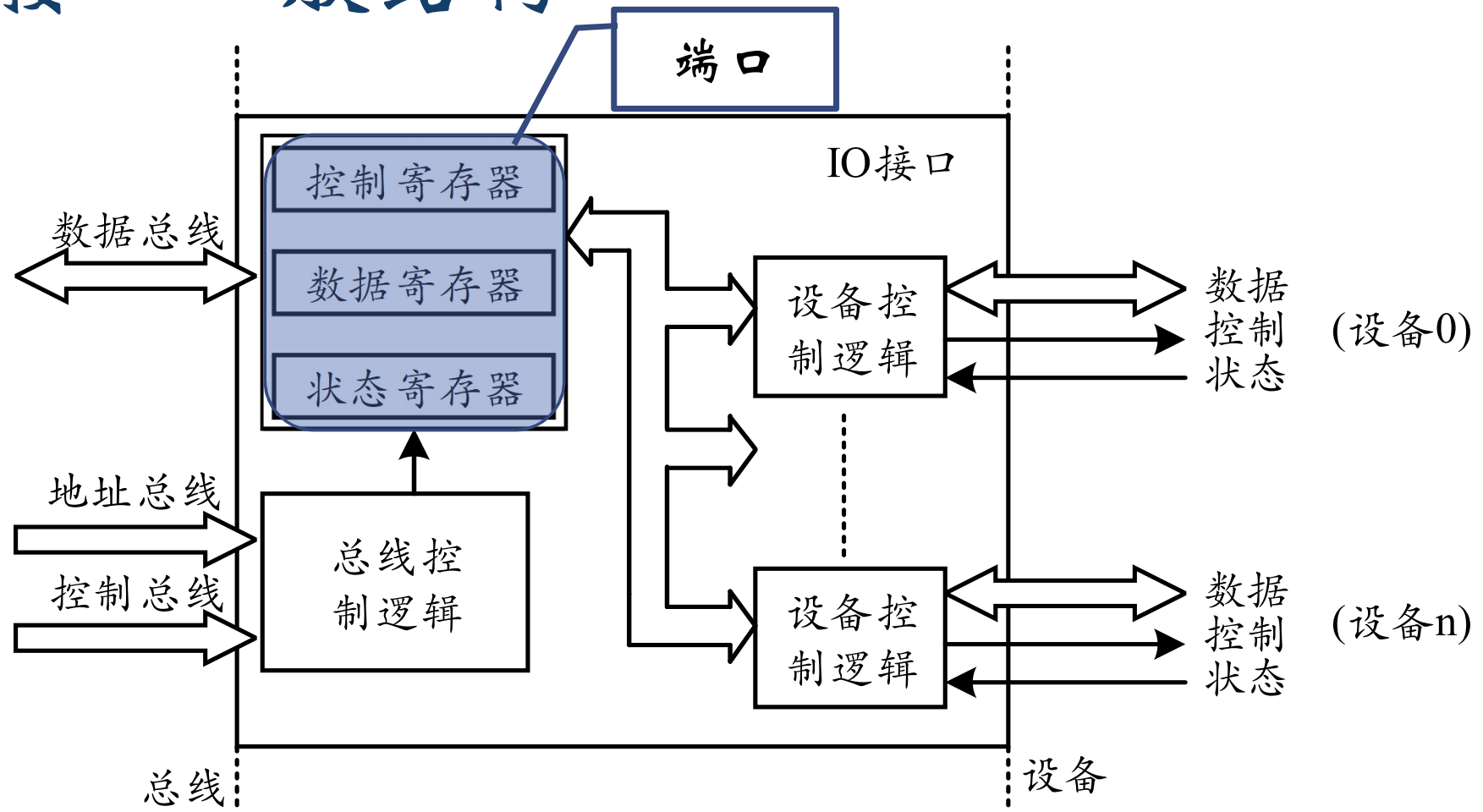
微处理器与IO接口之间通信内容

命令译码	接收微处理器的命令，译码产生控制外设的控制信号
数据缓冲	为实现传输速度匹配暂存数据
状态反馈	告诉微处理器IO设备的状态，包括忙、就绪等
地址译码	识别IO设备的地址

IO接口信号



IO接口一般结构


端口
IO接口
控制寄存器
数据寄存器
状态寄存器
数据总线
地址总线
控制总线
总线控制逻辑
设备控制逻辑
设备控制逻辑
数据
控制
状态
(设备0)
数据
控制
状态
(设备n)
总线
设备

IO接口数据传输方式

串行

并行

接口数据传输控制方式

数据传送完全在程序的控制下进行

程序控制

- 查询方式：询问外设状态，之后再通信
- “延时/无条件”传送方式，直接通信

中断：外设向微处理器发出请求，微处理器响应后再传送数据

DMA(Direct Memory Access)

数据不经过微处理器，而直接在存储器和外设之间进行传送

小结

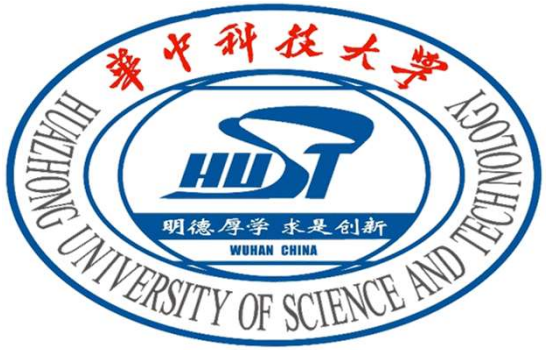
- 接口的基本概念
- 接口功能
- 接口结构
- 接口数据传输方式
- 接口数据传输控制方式

下一讲：IO接口寻址

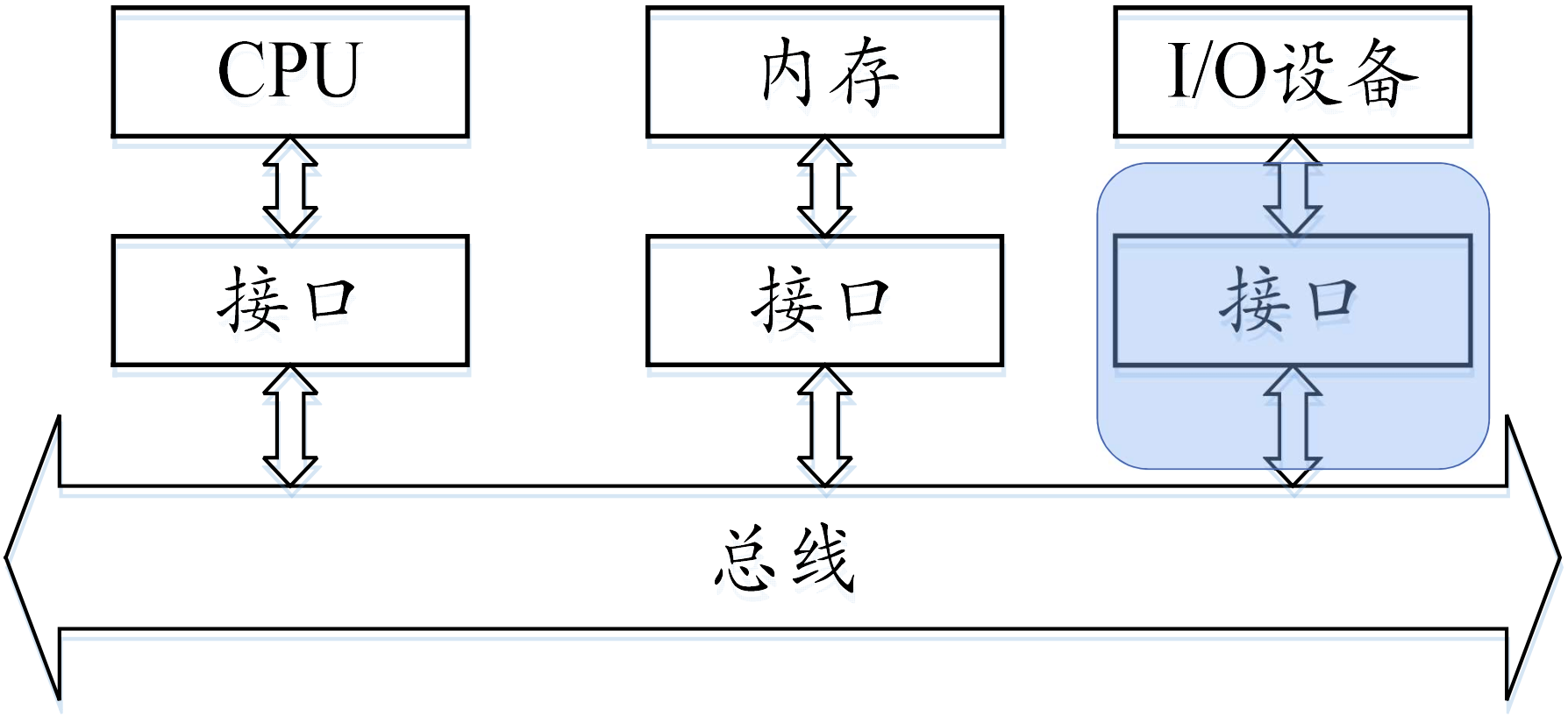
微机原理与接口技术

IO接口寻址

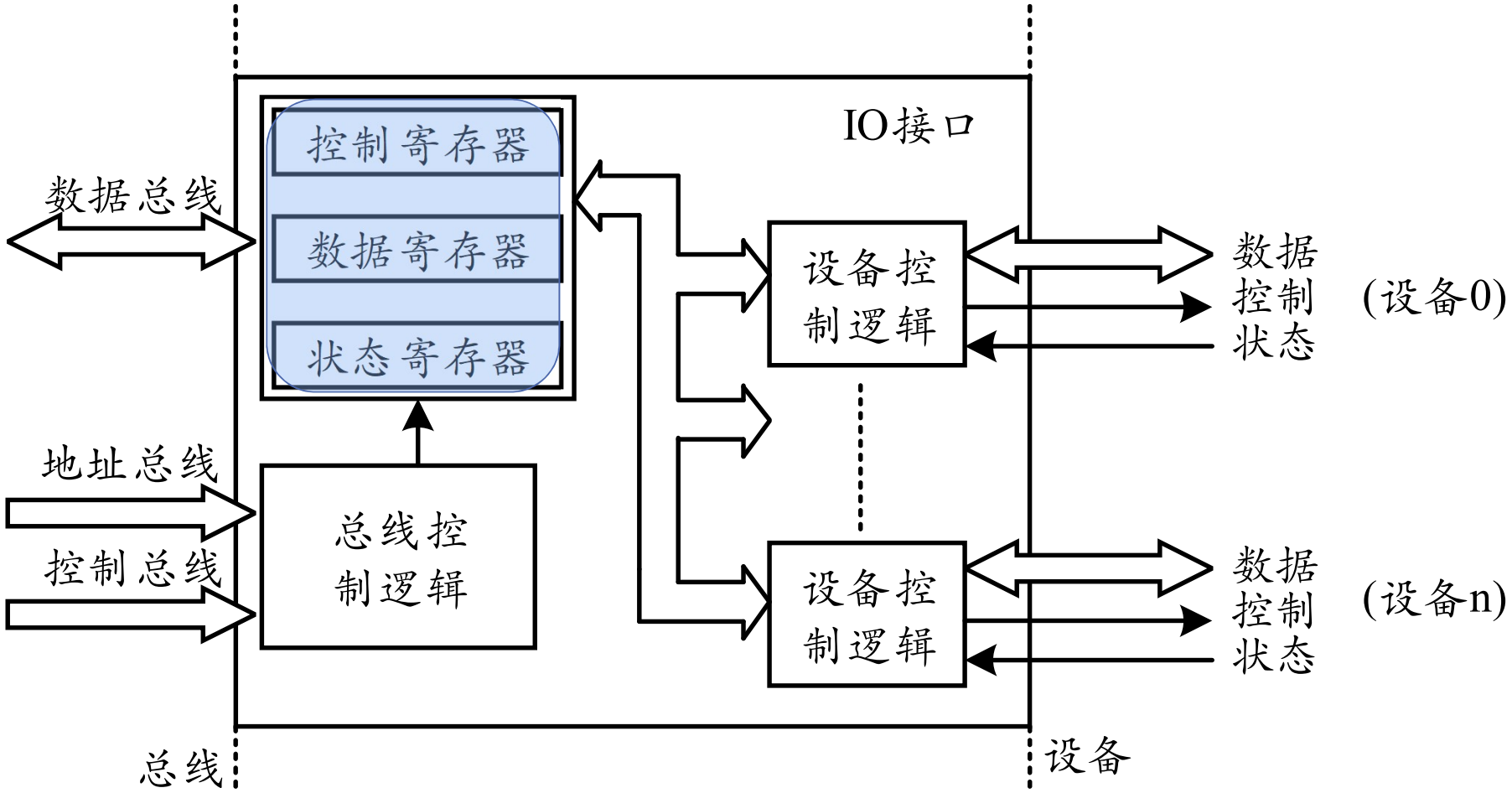
华中科技大学　左冬红



回顾



回顾——IO接口结构

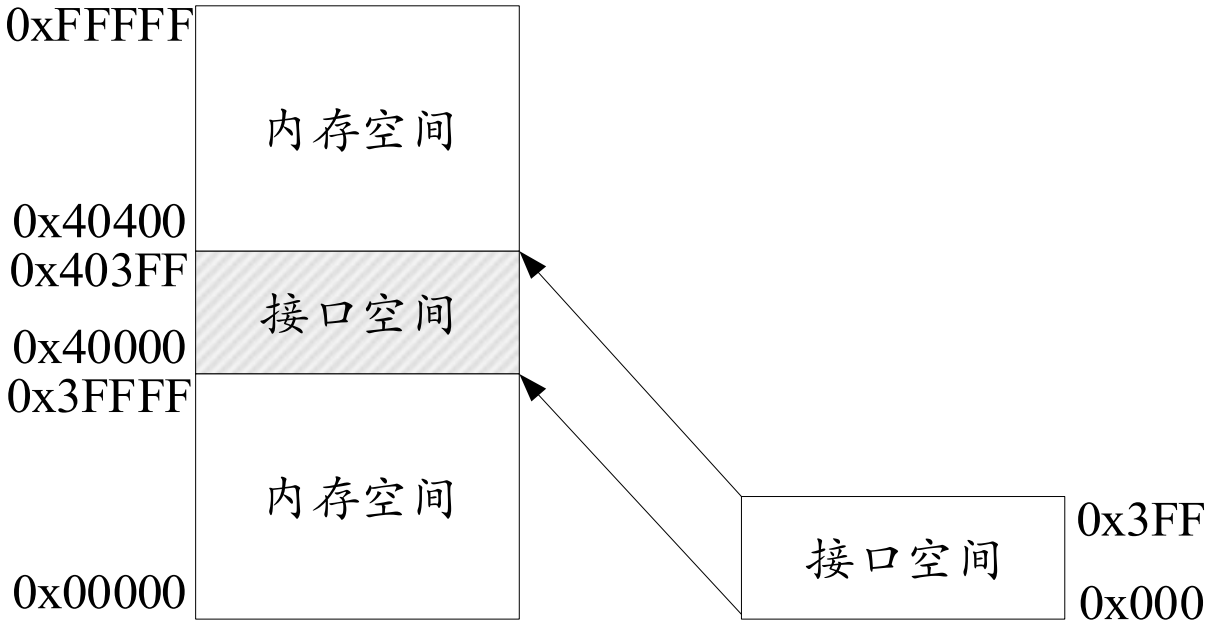

IO接口
控制寄存器
数据寄存器
状态寄存器
数据总线
地址总线
控制总线
总线控制逻辑
设备控制逻辑
设备控制逻辑
数据
控制
状态
(设备0)
数据
控制
状态
(设备n)
总线
设备

IO接口寻址方式

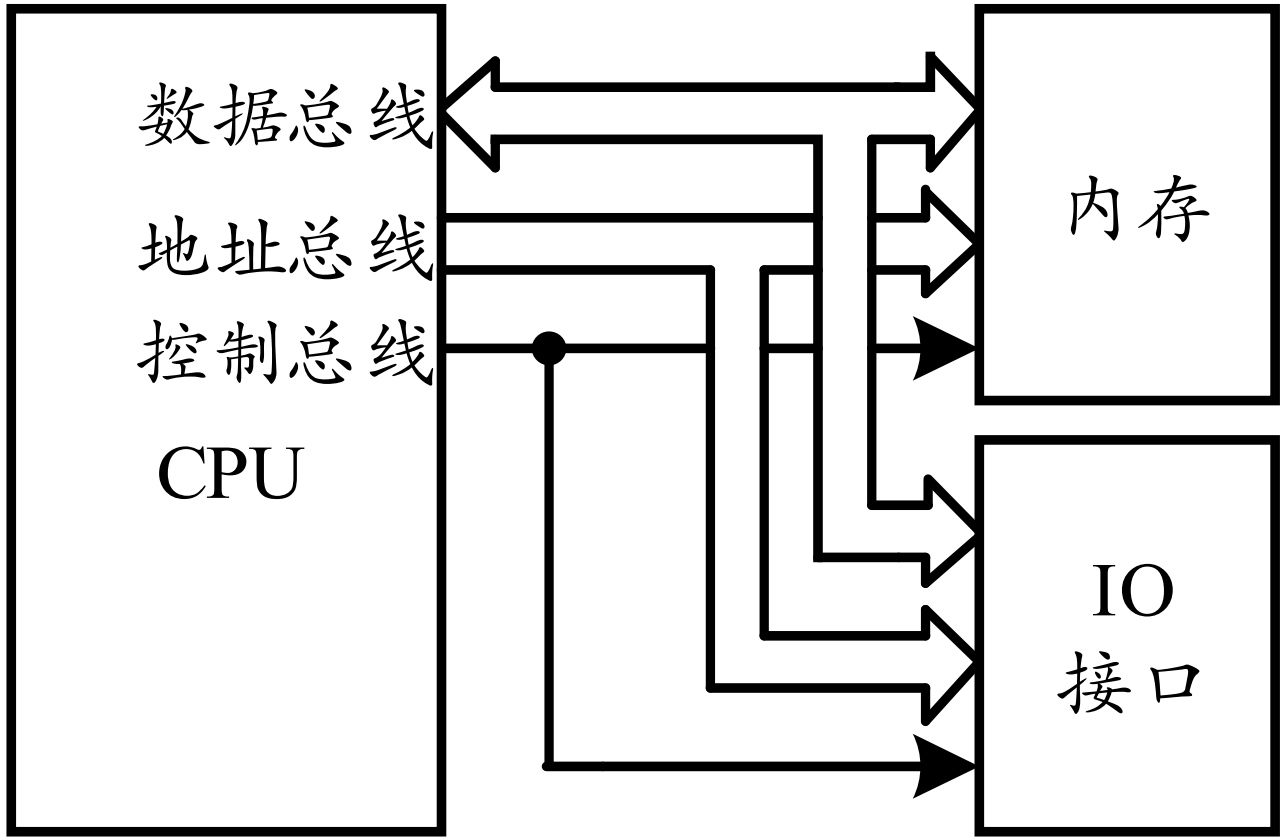
IO接口内的端口（寄存器）当做存储单元

存储器映像IO寻址

IO接口当做小容量存储器



存储器映像IO接口寻址



存储器映像IO寻址特点

I/O接口与内存共用同一个地址空间

CPU利用内存操作指令实现I/O设备管理

CPU利用存储器读/写控制信号($\overline{MEMR}$、$\overline{MEMW}$)对I/O设备进行读/写控制

CPU硬件设计简单、存储空间部分被IO接口占用，通过地址译码电路区分存储器\IO

嵌入式系统

独立IO寻址（标准IO寻址）

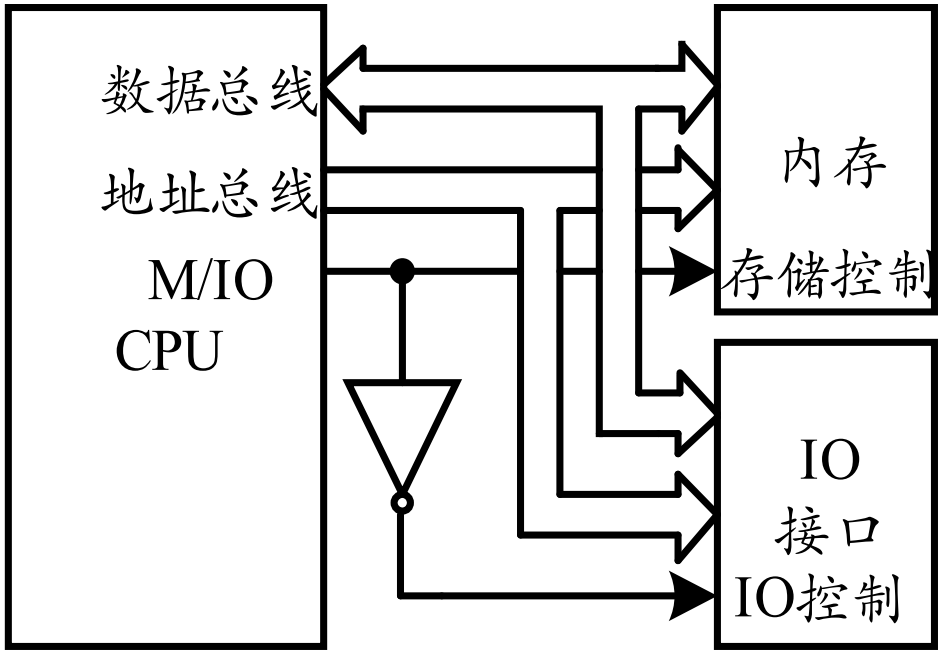
将IO空间与存储空间独立

CPU硬件需提供控制信号区分IO接口\存储器

访问IO接口\存储器采用不同指令

IO接口空间不占用存储空间

PC机



小结

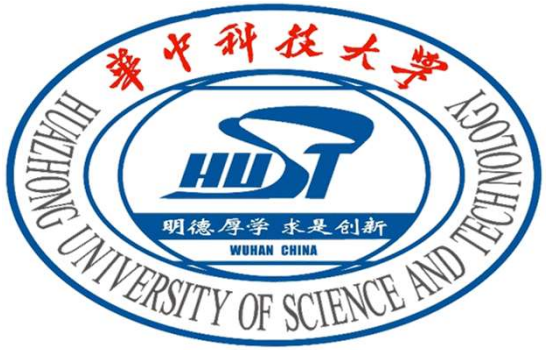
- 两种IO寻址方式
  - 特点
  - 适应场景

下一讲：IO接口操作函数

微机原理与接口技术

IO接口操作函数

华中科技大学　左冬红



MIPS汇编指令

存储器映像IO寻址

lw、lh(u)、lb(u)、sw、sh、sb

C语言函数

Standalone BSP 端口读写函数

Xil_In8(Addr)	static INLINE u8 Xil_In8(UINTPTR Addr) { return *(volatile u8 *) Addr;}
Xil_In16(Addr)	static INLINE u16 Xil_In16(UINTPTR Addr) { return *(volatile u16 *) Addr;}
Xil_In32(Addr)	static INLINE u32 Xil_In32(UINTPTR Addr) { return *(volatile u32 *) Addr;}
Xil_Out8(Addr, Value)	static INLINE void Xil_Out8(UINTPTR Addr, u8 Value) { volatile u8 *LocalAddr = (volatile u8 *)Addr; *LocalAddr = Value;}
Xil_Out16(Addr, Value)	static INLINE void Xil_Out16(UINTPTR Addr, u16 Value) { volatile u16 *LocalAddr = (volatile u16 *)Addr; *LocalAddr = Value;}
Xil_Out32(Addr, Value)	static INLINE void Xil_Out32(UINTPTR Addr, u32 Value) { volatile u32 *LocalAddr = (volatile u32 *)Addr; *LocalAddr = Value;}

示例

某计算机系统具有8位、16位、32位等不同端口，其中对应不同宽度IO端口地址如表12所示，若要求基于Xilinx Standalone BSP 端口读写函数分别将不同宽度输入端口中的数据读入之后输出到输出端口中，试编写控制程序段。

端口类型	输入端口地址	输出端口地址
8位	0x56	0x789
16位	0xfe	0x345
32位	0xcd	0x123

```
Xil_Out8(0x789, Xil_In8(0x56));// 读入8位输入端口数据输出到8位输出端口
Xil_Out16(0x345, Xil_In16(0xfe)); //读入16位输入端口数据输出到16位输出端口
Xil_Out32(0x123, Xil_In32(0xcd)); //32位输入端口数据输出到32位输出端口
```

小结

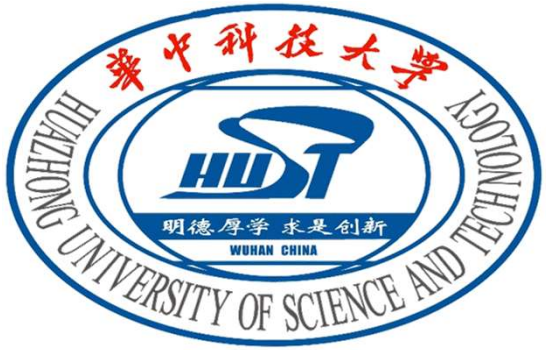
- 接口编程采用C语言
- 编程环境为StandAlone BSP操作系统
- MicroBlaze微处理器

下一讲：接口电路总线控制逻辑

微机原理与接口技术

IO接口总线控制逻辑

华中科技大学　左冬红



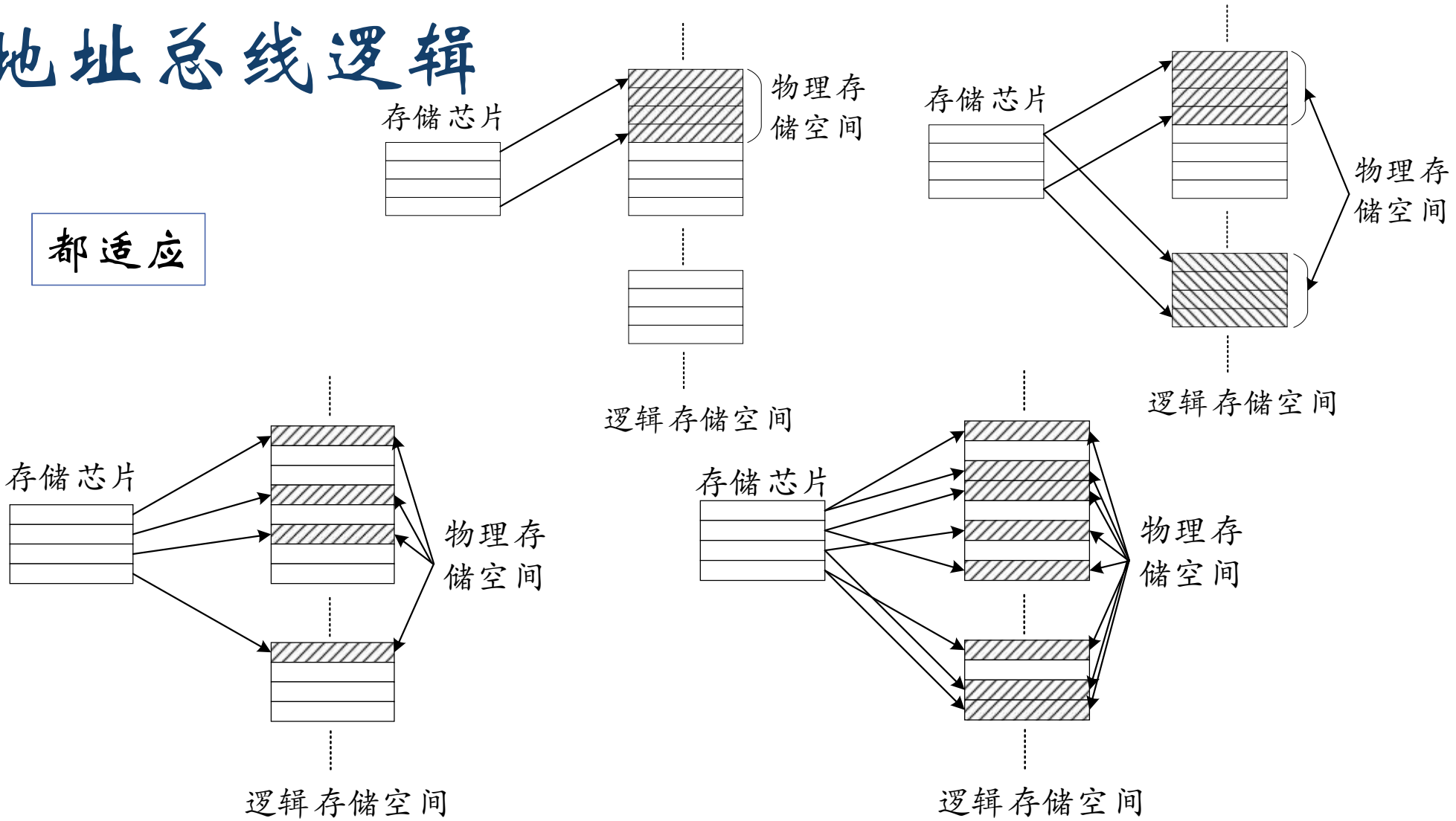
总线控制逻辑电路

由地址译码和控制执行逻辑组成，完成接口寻址和总线操作、定时控制等

地址总线	IO端口映射到IO接口空间（存储空间）、间接译码
数据总线	IO接口数据线宽度与总线宽度匹配、多类型数据访问
控制总线	存储器映像\独立IO寻址

地址总线逻辑

都适应


存储芯片
物理存储空间
逻辑存储空间
存储芯片
物理存储空间
逻辑存储空间
存储芯片
物理存储空间
逻辑存储空间
存储芯片
物理存储空间
逻辑存储空间

IO端口数大于IO空间——间接端口译码

已知某计算机系统总线具有7位地址总线$A_{6\text{-}0}$、8位数据总线$D_{7\text{-}0}$以及独立的读写控制信号WR、RD，现需为该计算机系统扩展一片62256 SRAM存储芯片。试设计该存储芯片与计算机系统总线的接口电路。

7位地址总线，无法访问32K个存储单元

数据总线需分时传送两类信息：

8位数据总线D7-0传输片内存储单元地址需两次总线操作

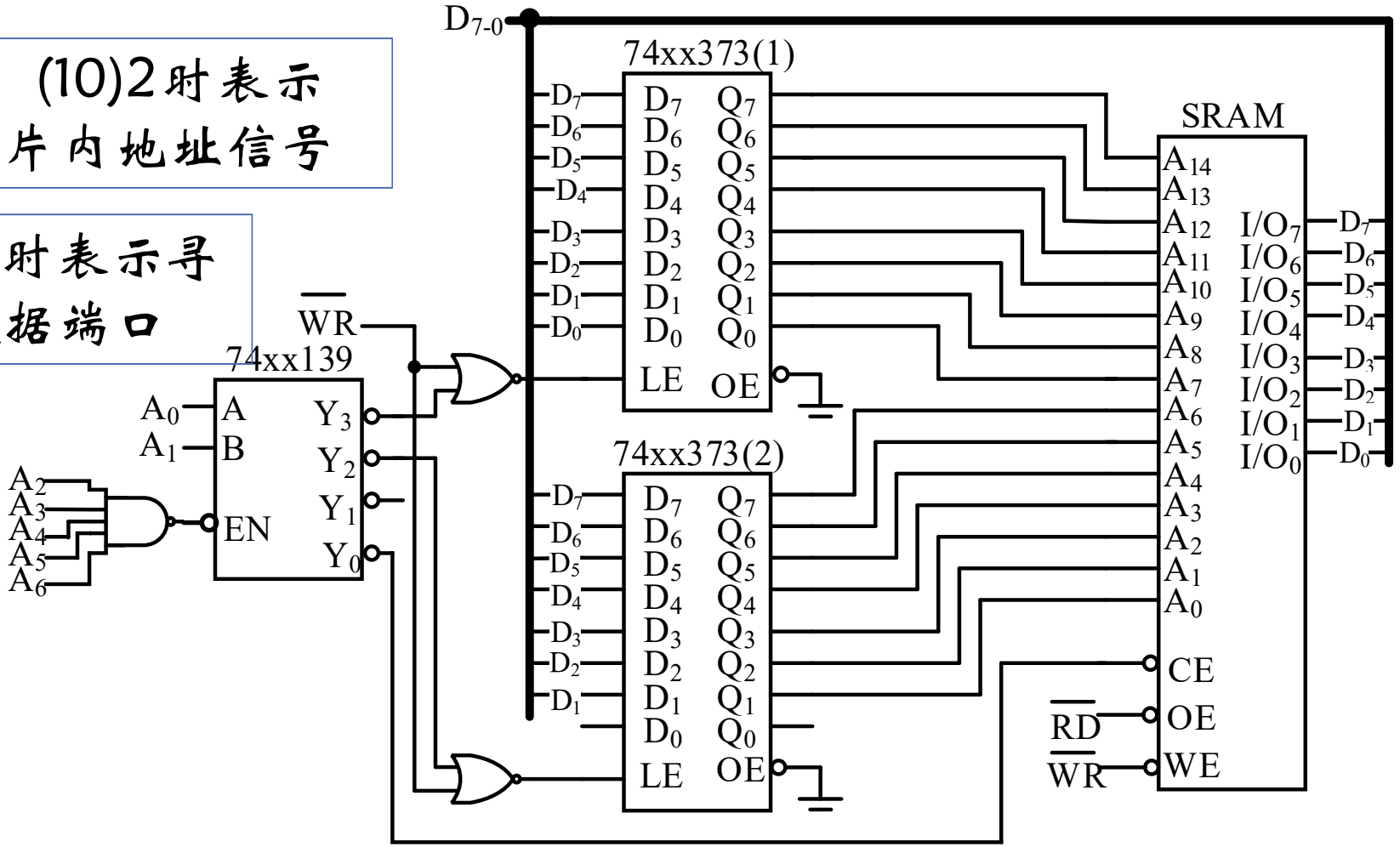
1）寻址片内存储单元的地址信息；

2）存储单元数据信息

IO端口数大于IO空间——间接端口译码

A1~0为(11)2、(10)2时表示传输存储芯片片内地址信号

A1~0为(00)2时表示寻址存储芯片数据端口



数据总线

当数据总线宽度大于或等于设备数据线宽度时，可选取总线中与接口数据线同样宽度的连续多位数据与接口数据线对应连接即可

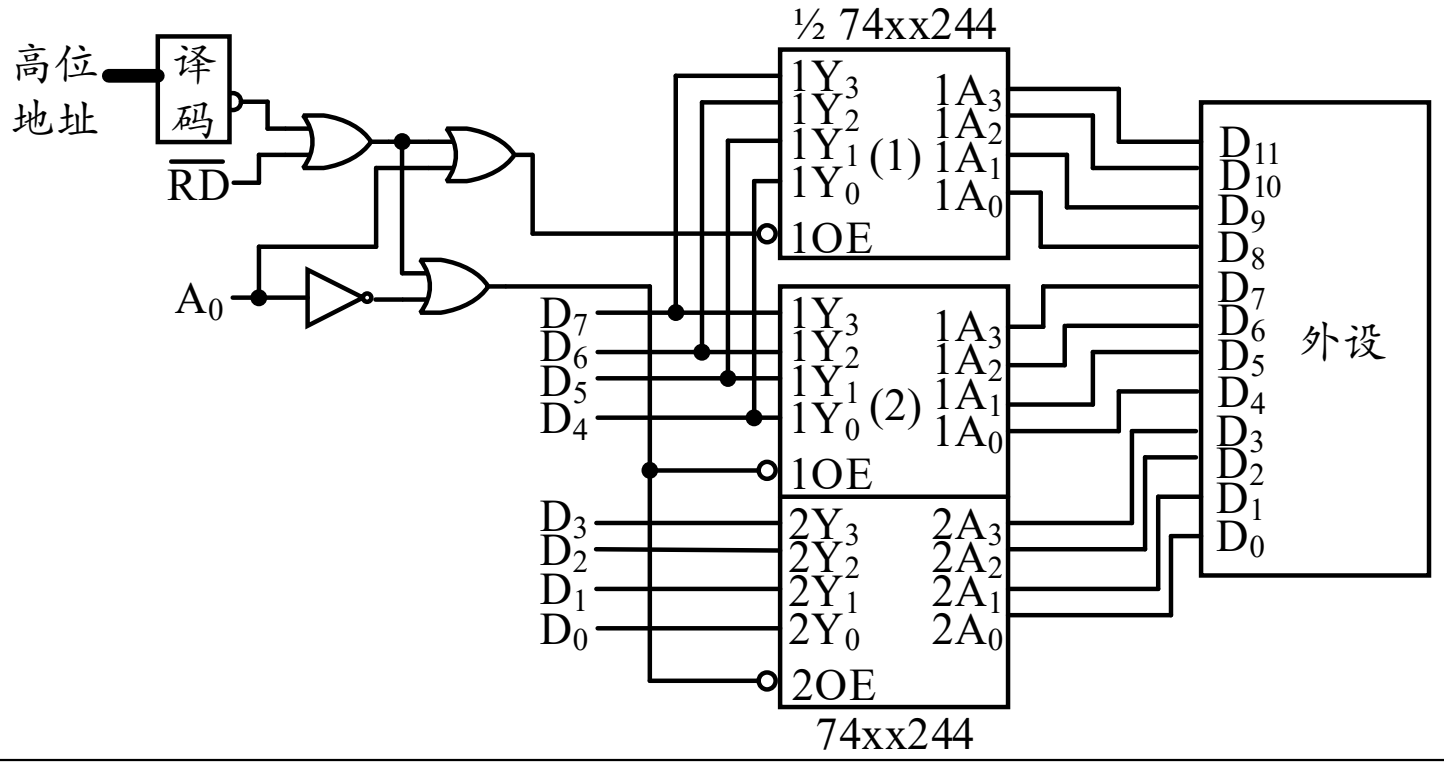
接口字节使能信号译码原理与存储器多类型数据访问译码原理相同。

若设备接口数据线宽度大于总线数据宽度，需采用不同端口传输设备的不同数据位

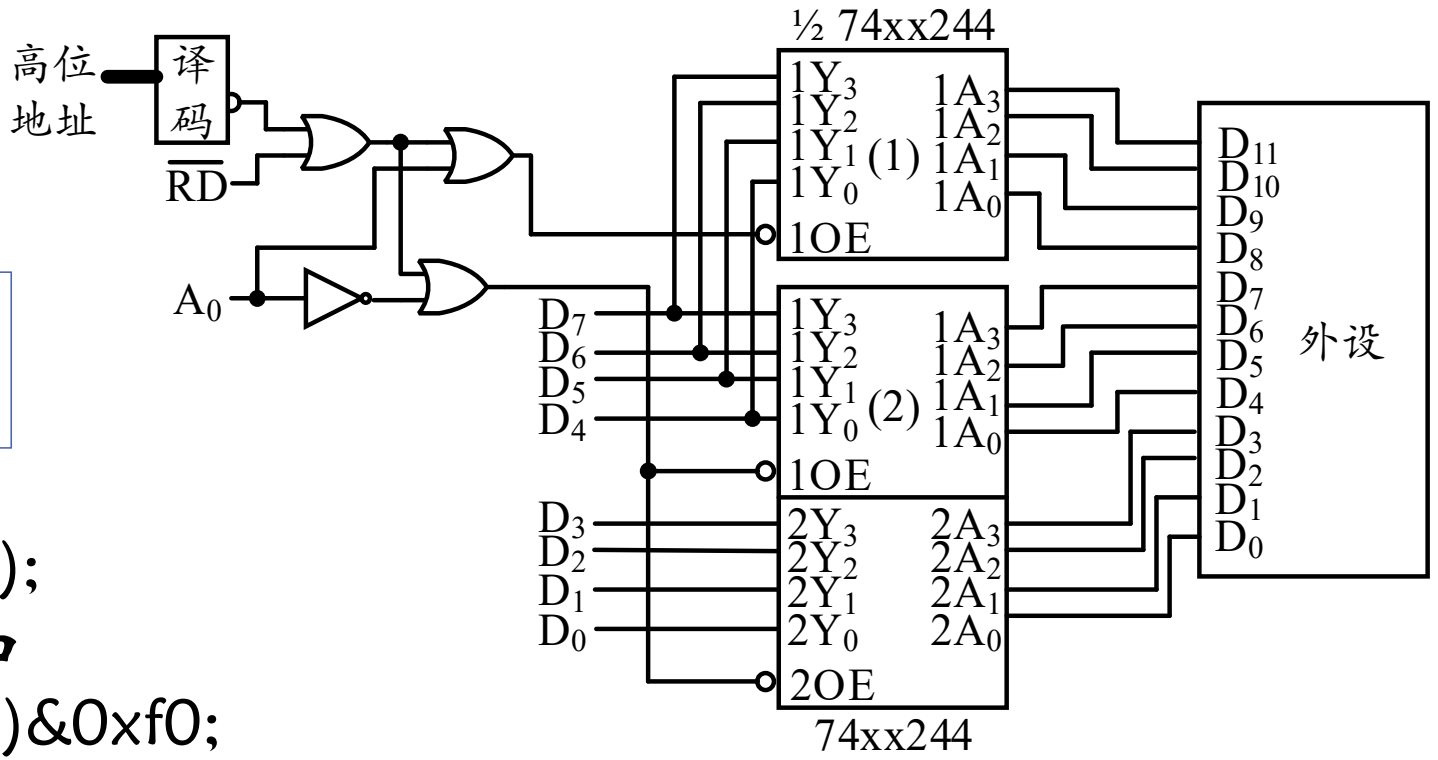
示例1

已知某计算机系统数据总线宽度仅为8位D7~0，外设具有12位数据输出D11~0。若需将外设数据通过该总线读入计算机系统，试设计接口电路，并基于Xilinx Standalone BSP 端口读写C语言函数编写程序段读入外设一个12位数据。

两个数据端口



示例1



假定地址为0x378或0x379

```
byte0=Xil_In8(0x379);
 // 读地址0x379的端口
byte1= Xil_In8(0x378)&0xf0;
// 读地址0x378的端口且仅保留高4位D7~4
Peri_data = ((unsigned short)byte1<<4)|byte0;
//高4位左移4位与低8位合并为12位
```

示例2

已知某计算机系统数据总线宽度仅为8位D7~0，外设具有16位数据输入引脚D15~0。若计算机系统需向该外设写入16位数据，且这16位数据需同步到达外设数据输入引脚D15~0，试设计接口电路，并基于Xilinx Standalone BSP 端口读写C语言函数编写向外设写一个16位数据0x3456的程序段。

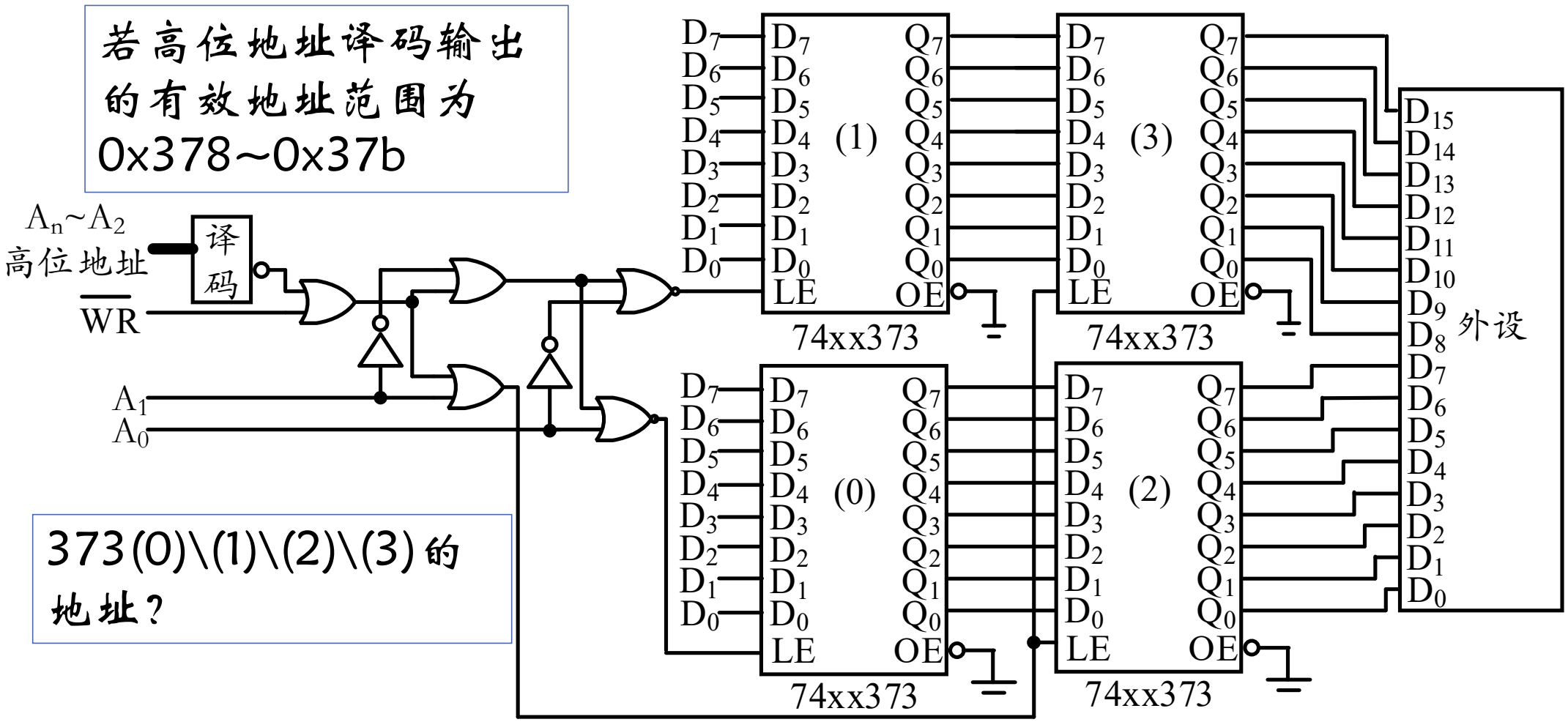
两次分别输出并锁存8位数据

第三次控制将锁存的16位数据同步输出

3个数据端口

示例2

若高位地址译码输出的有效地址范围为0x378~0x37b



373(0)\(1)\(2)\(3)的地址？

示例2

373锁存器编号	高位地址A_n~A_2	A_1	A_0	端口地址
(0)	$(0\cdots011011110)_2$	1	0	0x37a
(1)	$(0\cdots011011110)_2$	1	1	0x37b
(2)	$(0\cdots011011110)_2$	0	x	0x378或0x379
(3)	$(0\cdots011011110)_2$	0	x	0x378或0x379

```
unsigned char byte0,byte1;
unsigned short Peri_data=0x3456;
byte0=(unsigned char)Peri_data; // 获取低8位保存到byte0
byte1= (unsigned char)(Peri_data>>8); //获取高8位保存到byte1
Xil_Out8(0x37a,byte0); //使能74xx373(0)输出低8位数据并锁存
Xil_Out8(0x37b,byte1); //使能74xx373(1)输出高8位数据并锁存
Xil_Out8(0x378,0x0);//使能74xx373(2)、(3)
```

小结

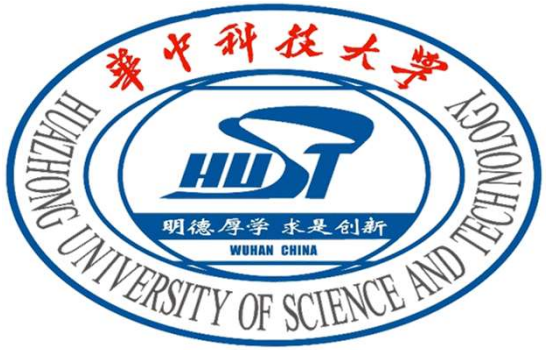
- IO接口总线控制逻辑
  - 基本原理与存储器接口一致
  - 不同之处
    - 存储芯片输入输出具有缓冲
    - 存储芯片具有信息保持功能
    - IO接口数据线连接到总线必须通过缓冲功能
    - 总线信号到IO接口需由IO接口提供锁存功能
    - IO接口可间接译码
    - IO接口可数据线复用

下一讲：常见数字并行IO设备接口电路

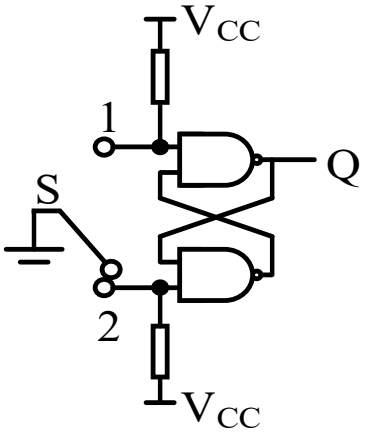
微机原理与接口技术

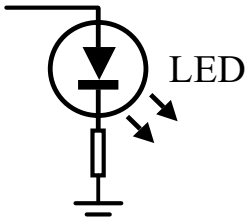
常用并行数字IO设备接口

华中科技大学　左冬红

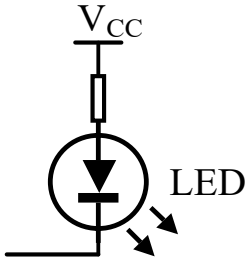


常用并行数字IO设备

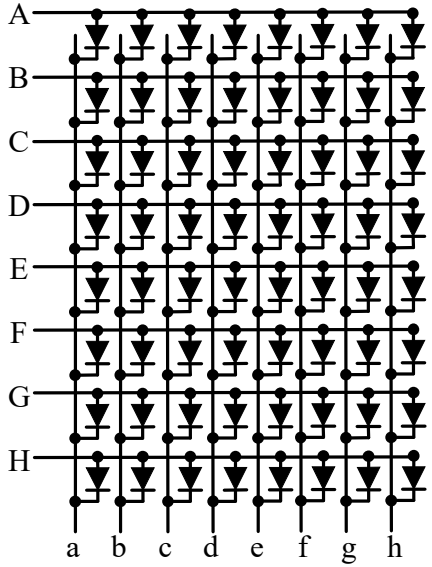


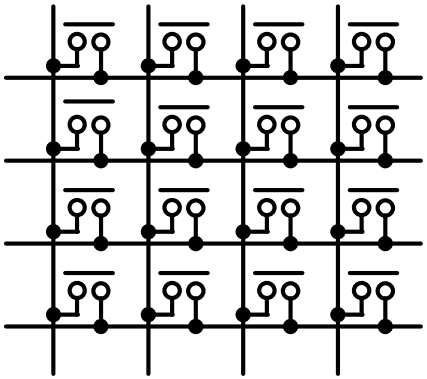


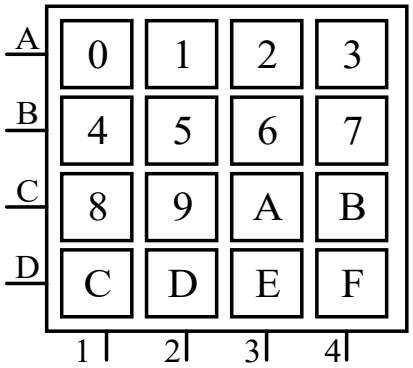
(a)

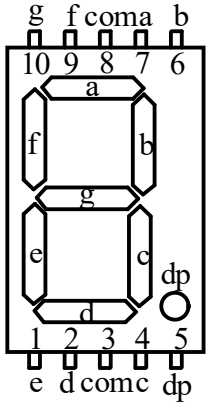


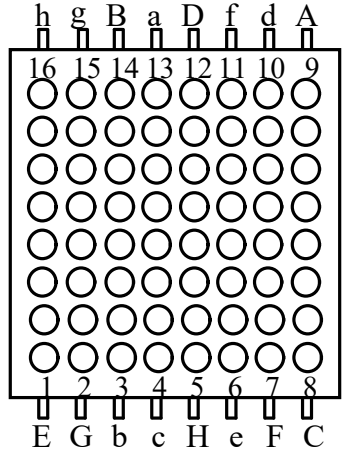
(b)











常用并行数字IO设备特点

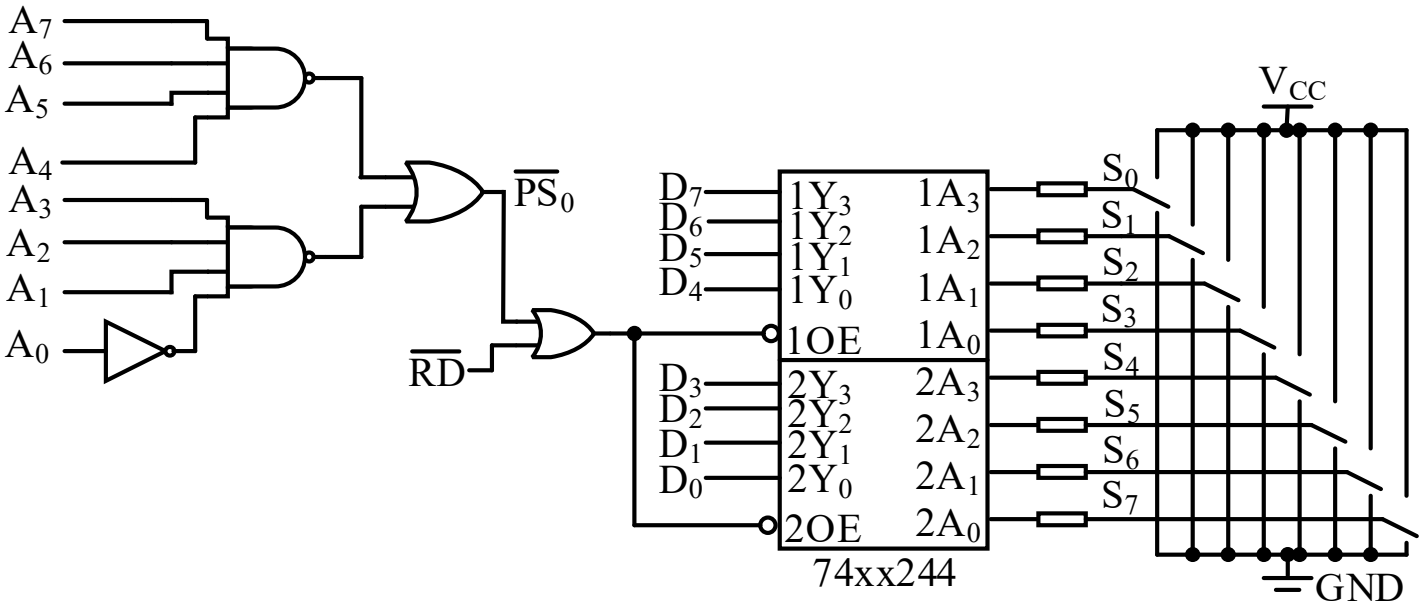
输出设备不具有数据保持功能

输入设备不具有数据缓冲功能

数据缓冲、锁存都必须由接口电路实现

独立开关接口电路示例

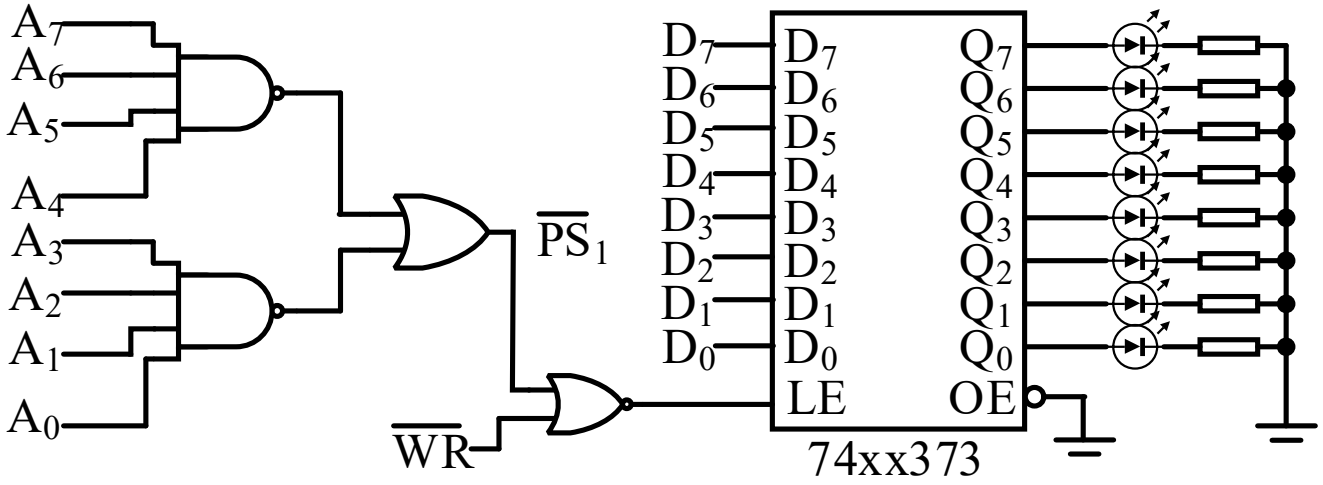
已知某计算机系统具有8位地址总线A7~0、8位数据总线D7~0，采用存储器映像IO寻址方式，要求为该计算机系统设计一个8位独立开关输入接口电路，且端口地址为0xfe，并编写控制程序段读入8位开关的状态。




```
Switch=Xil_In8(0xfe);
```

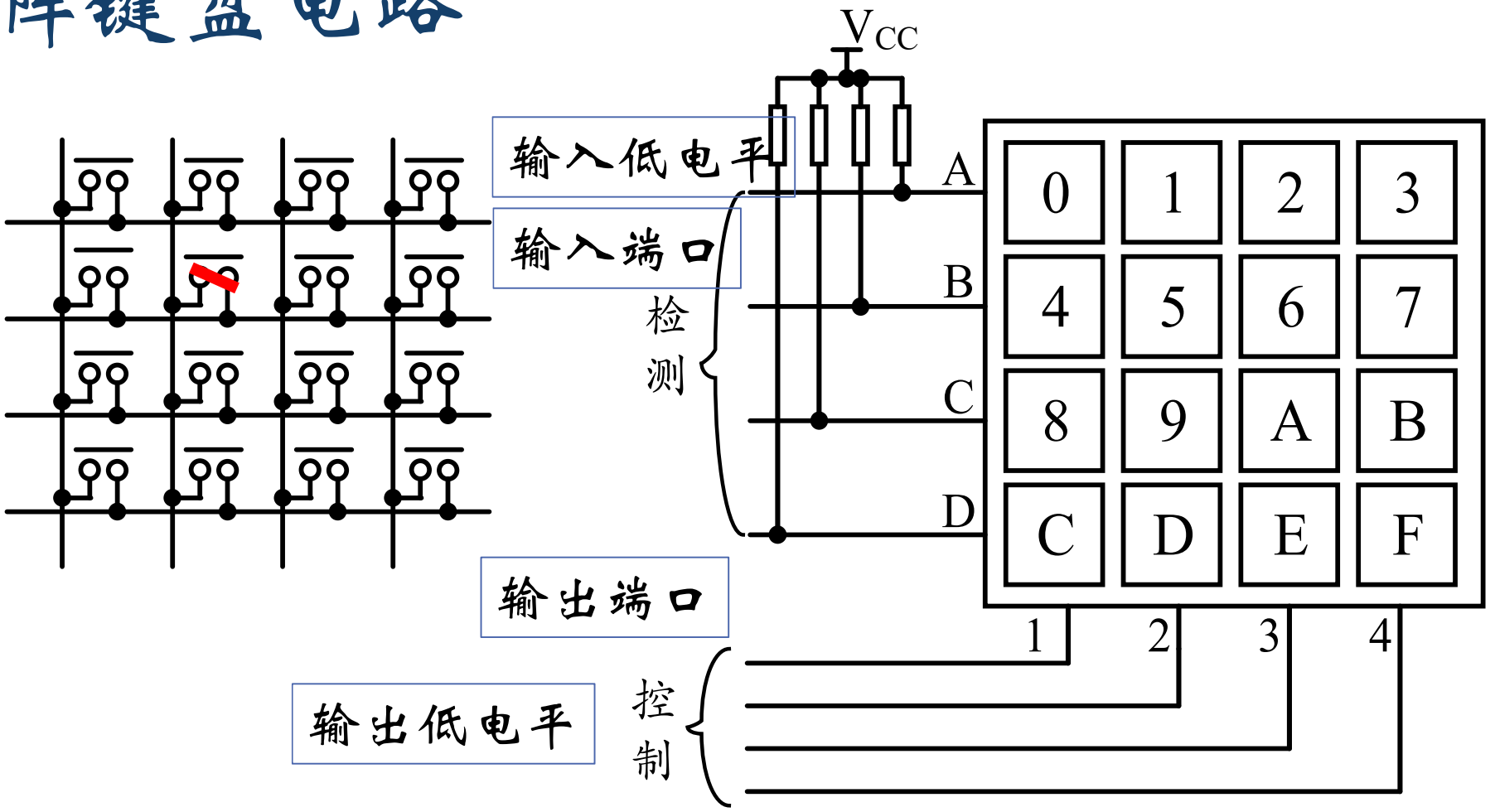
独立LED灯接口电路示例

已知某计算机系统具有8位地址总线A7~0、8位数据总线D7~0，采用存储器映像IO寻址方式，要求为该计算机系统设计一个8位发光二极管输出接口电路，且端口地址为0xff，并编写控制程序段将8个发光二极管点亮。




```
Xil_Out8( 0xff,0xff);
```

矩阵键盘电路



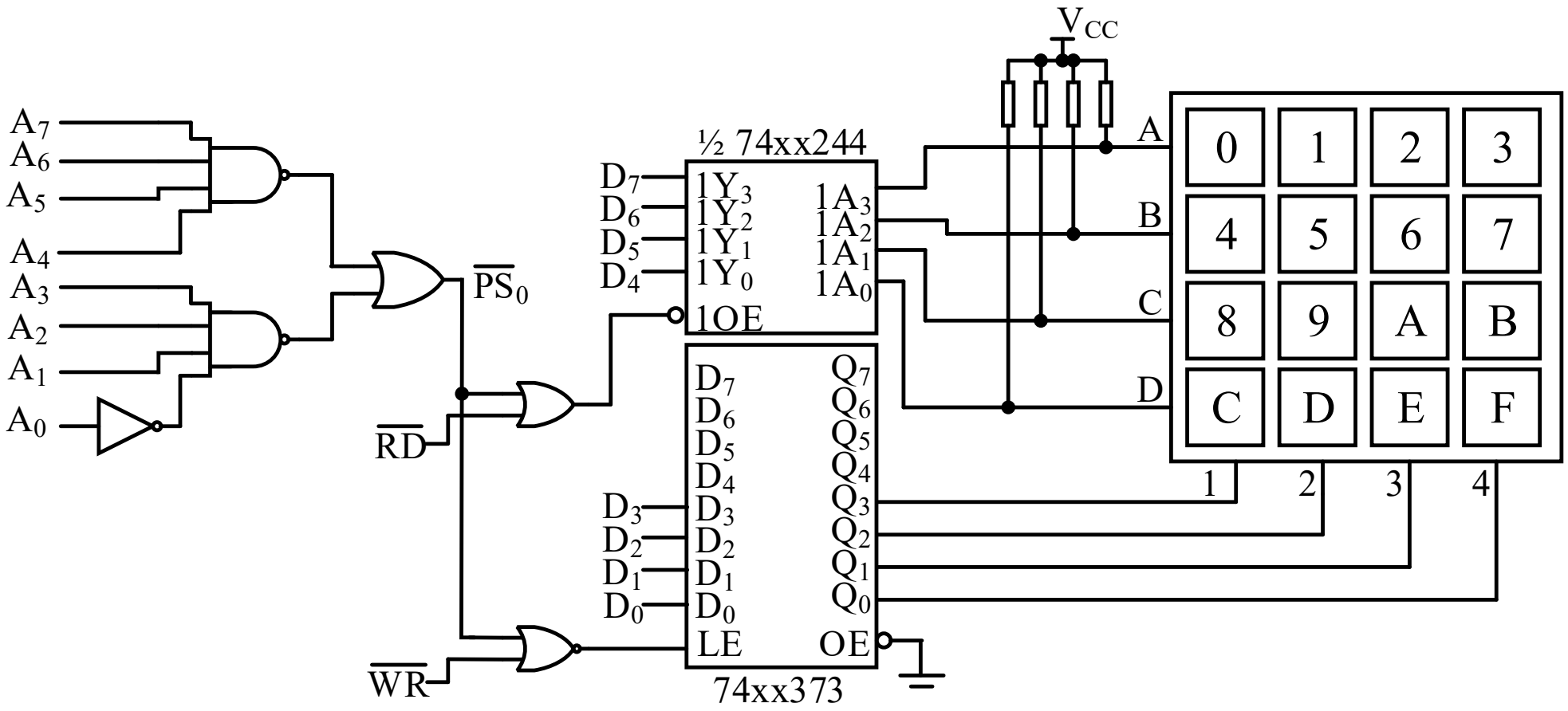
矩阵键盘接口电路示例

已知某计算机系统具有8位地址总线A7~0、8位数据总线D7~0，采用存储器映像IO寻址方式，要求为该计算机系统设计一个4×4矩阵键盘输入设备接口电路，且仅具有一个端口地址0xfe。编写控制程序识别按键并输出各个按键所表示的字符。

8位端口，4位输入，4位输出，输入输出由控制信号控制

端口地址0xfe分别与RD\WR信号译码控制输入、输出

矩阵键盘接口电路示例


V_{CC}
A_7
A_6
A_5
A_4
A_3
A_2
A_1
A_0
$\overline{PS_0}$
$\overline{RD}$
$\overline{WR}$
½ 74xx244
D_7
D_6
D_5
D_4
$1Y_3$
$1Y_2$
$1Y_1$
$1Y_0$
1OE
$1A_3$
$1A_2$
$1A_1$
$1A_0$
D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
LE
Q_7
Q_6
Q_5
Q_4
Q_3
Q_2
Q_1
Q_0
OE
74xx373
A
B
C
D
0 1 2 3
4 5 6 7
8 9 A B
C D E F
1 2 3 4

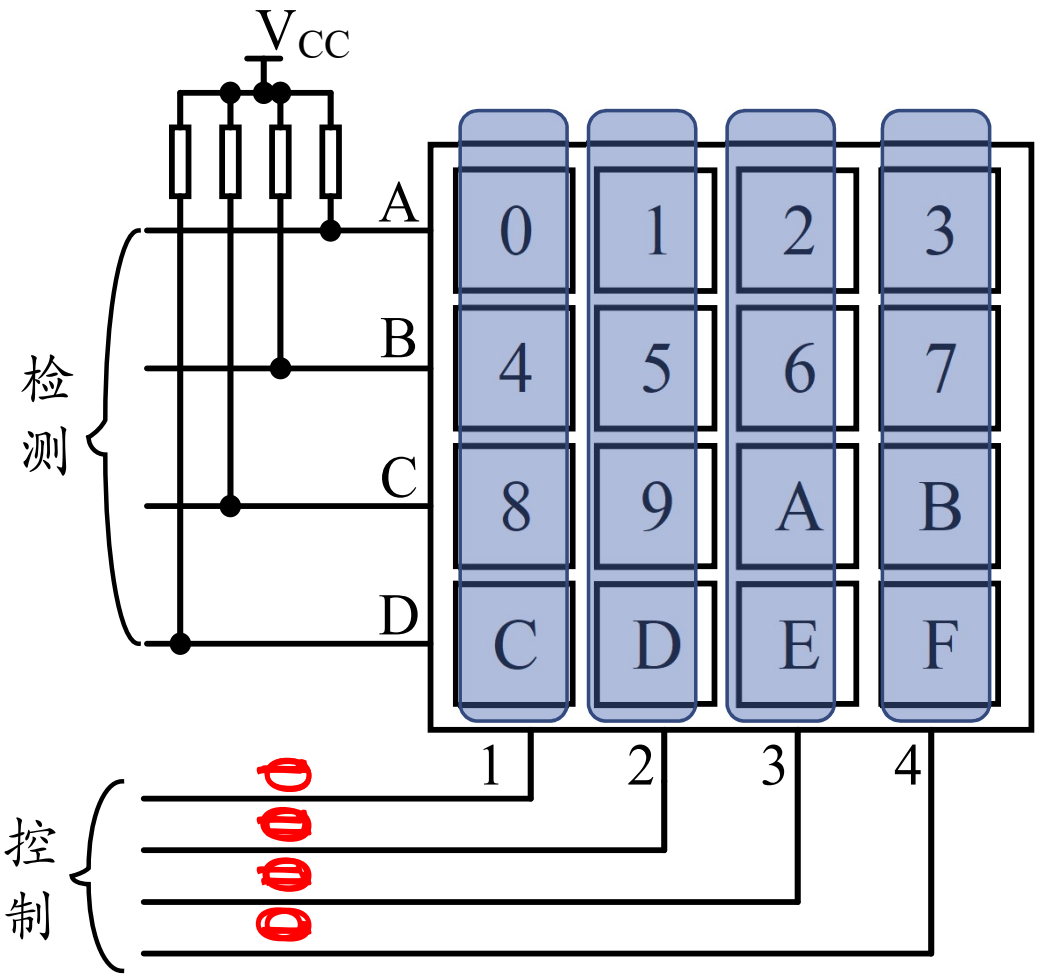
矩阵键盘按键识别

是否有键按下？

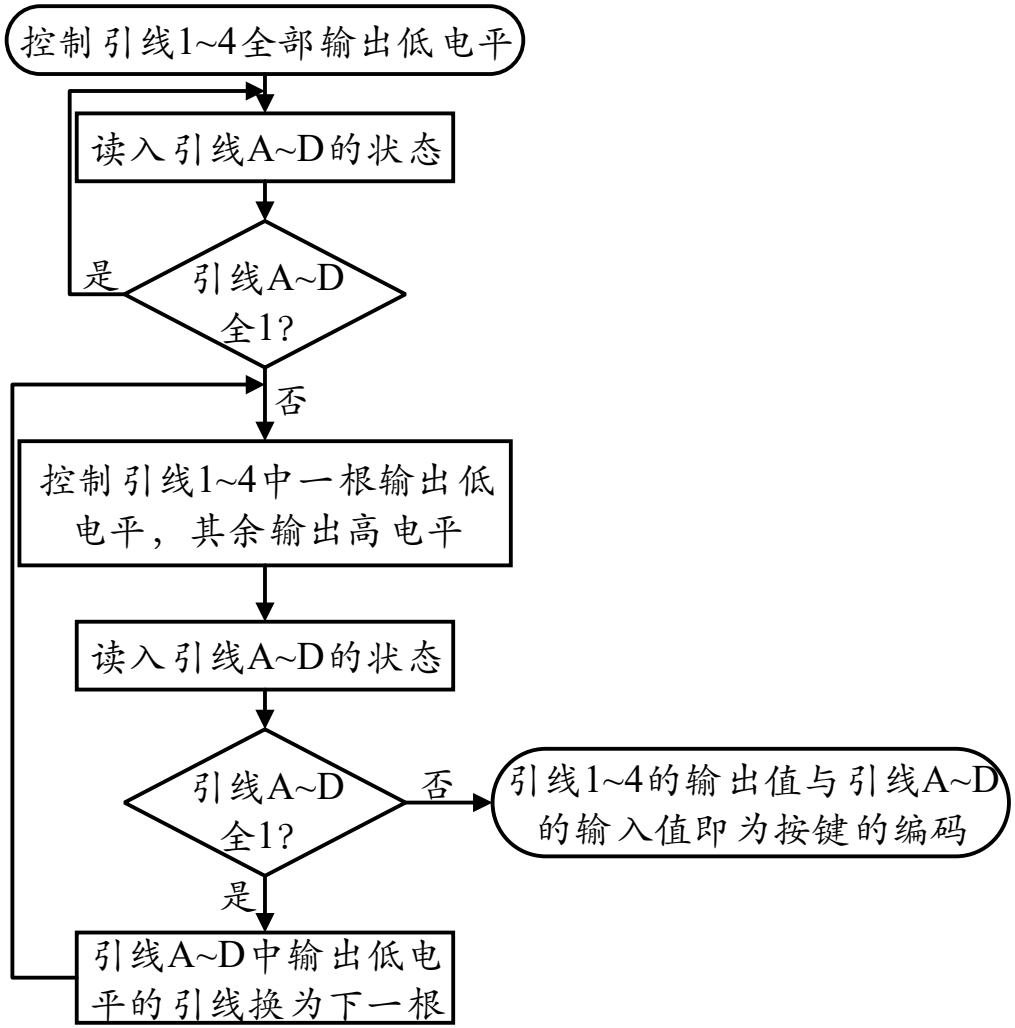
程序控制简单

哪个按键按下？

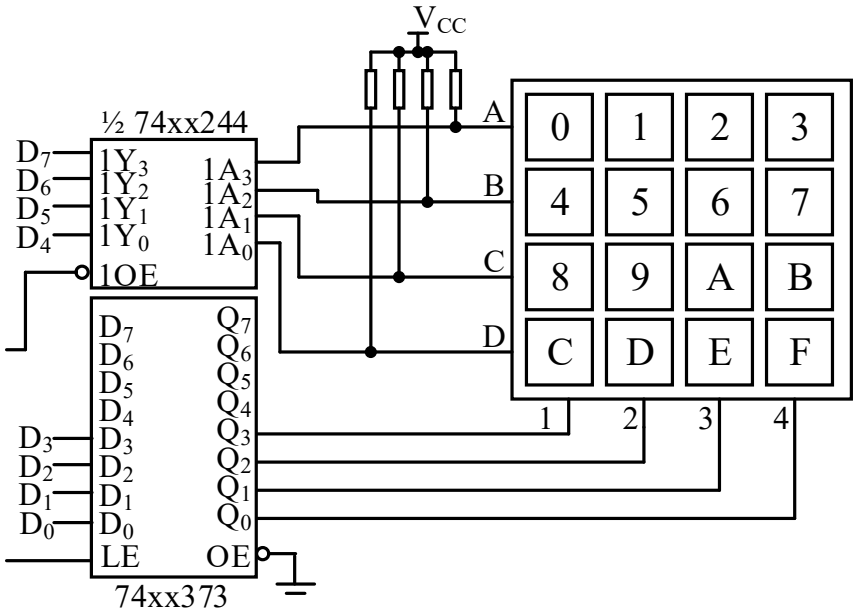
准确识别



矩阵键盘控制流程


控制引线1~4全部输出低电平
读入引线A~D的状态
是
引线A~D全1?
否
控制引线1~4中一根输出低电平，其余输出高电平
读入引线A~D的状态
引线A~D全1?
否
引线1~4的输出值与引线A~D的输入值即为按键的编码
是
引线A~D中输出低电平的引线换为下一根

控制程序




```
unsigned char KeyScancode;
char Row,Col=0xf7;
Xil_Out8(0xfe,0x00);//
While ((Row=Xil_In8(0xfe)&0xf0)==0xf0);//
Xil_Out8 (0xfe,Col);//
while ((Row=Xil_In8 (0xfe)&0xf0)==0xf0) //
  {
     Col=Col>>1; //
     Xil_Out8 (0xfe,Col);
   }
 KeyScancode=Row|(Col&0xf);
```

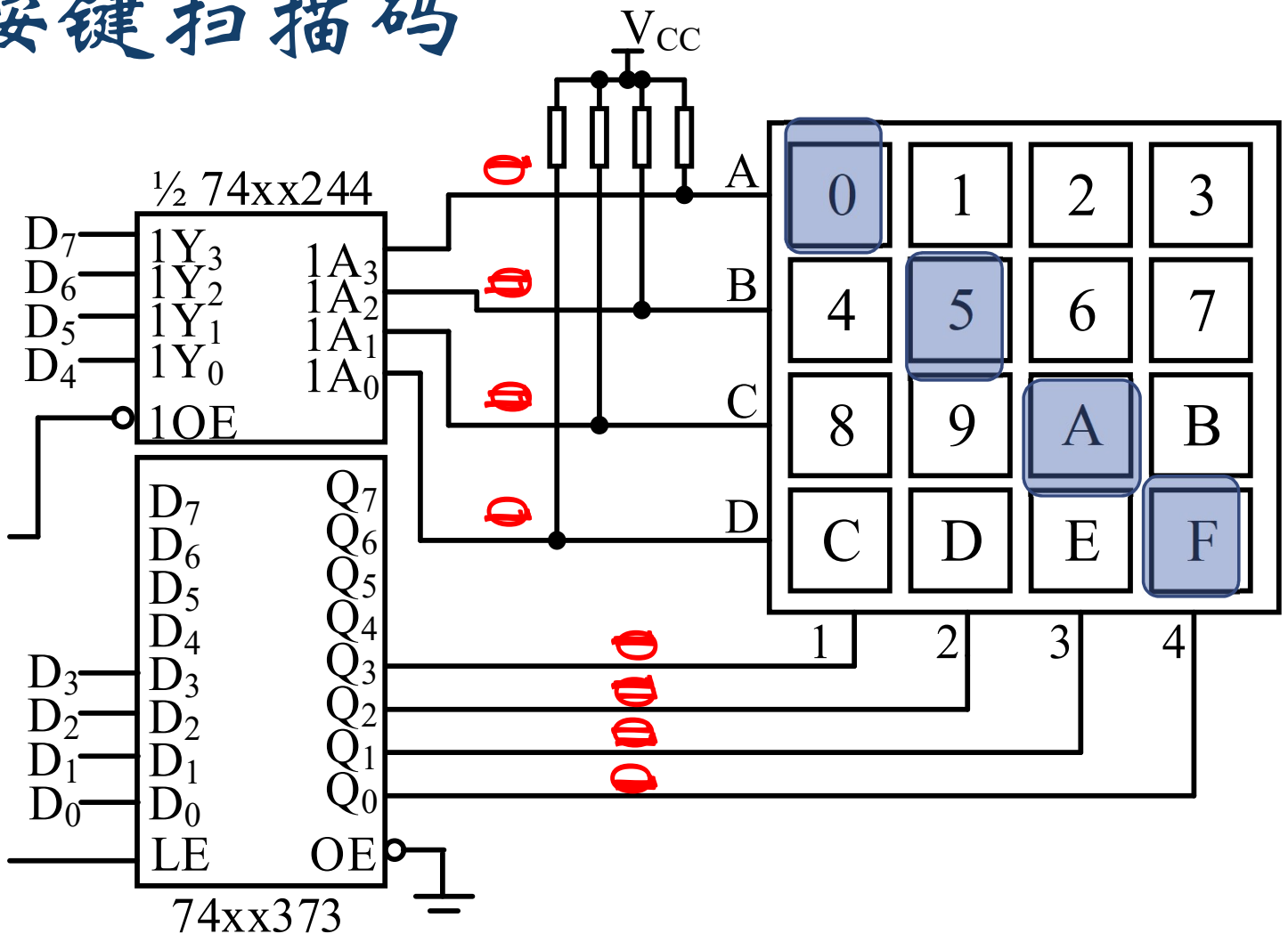
矩阵键盘按键扫描码

0的扫描码

5的扫描码

A的扫描码

F的扫描码



矩阵键盘按键扫描码

按键	引线A (D_7)	引线B (D_6)	引线C (D_5)	引线D (D_4)	引线1 (D_3)	引线2 (D_2)	引线3 (D_1)	引线4 (D_0)	编码
0	0	1	1	1	0	1	1	1	0x77
1	0	1	1	1	1	0	1	1	0x7b
2	0	1	1	1	1	1	0	1	0x7d
3	0	1	1	1	1	1	1	0	0x7e
4	1	0	1	1	0	1	1	1	0xb7
5	1	0	1	1	1	0	1	1	0xbb
6	1	0	1	1	1	1	0	1	0xbd
7	1	0	1	1	1	1	1	0	0xbe
8	1	1	0	1	0	1	1	1	0xd7
9	1	1	0	1	1	0	1	1	0xdb
A	1	1	0	1	1	1	0	1	0xdd
B	1	1	0	1	1	1	1	0	0xde
C	1	1	1	0	0	1	1	1	0xe7
D	1	1	1	0	1	0	1	1	0xeb
E	1	1	1	0	1	1	0	1	0xed
F	1	1	1	0	1	1	1	0	0xee

矩阵键盘按键键值

扫描码到数字

```
unsigned char
hex_table[16]={0x77,0x7b,0x7d,0x7e,0xb7,0xbb,0xbd,0xbe,
0xd7,0xdb,0xdd,0xde,0xe7,0xeb,0xed,0xee}
```

扫描码到ASCII字符

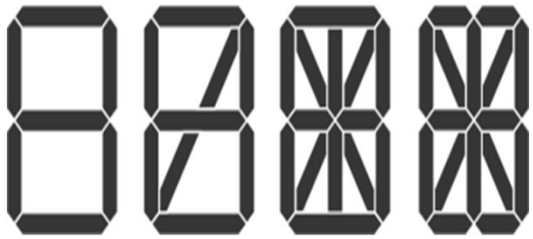
```
unsigned char
ascii_table[16][2]={0x77,0x30,0x7b,0x31,0x7d,0x32,0x7e,0x33,
0xb7,0x34,0xbb,0x35,0xbd,0x36,0xbe,0x37,
0xd7,0x38,0xdb,0x39,0xdd,0x41,0xde,0x42,
0xe7,0x43,0xeb,0x44,0xed,0x45,0xee,0x46}
```

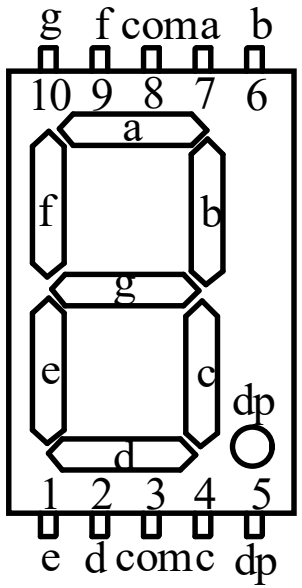
矩阵键盘按键键值获取程序段

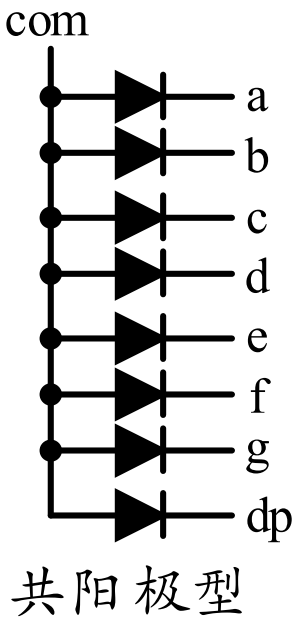
```
unsigned char Key_Hex, Key_Ascii;
for(int i=0;i<16;i++) // 获取按键ASCII字符
{
  if (ascii_table[i][0]==KeyScancode)
     {
Key_Ascii = ascii_table[i][1];
break;
}
 }
```

```
for(int i=0;i<16;i++) // 按键十六进制键值
{
  if (hex_table[i]==KeyScancode)
     {
        Key_Hex = i;
        break;
        }
 }
```

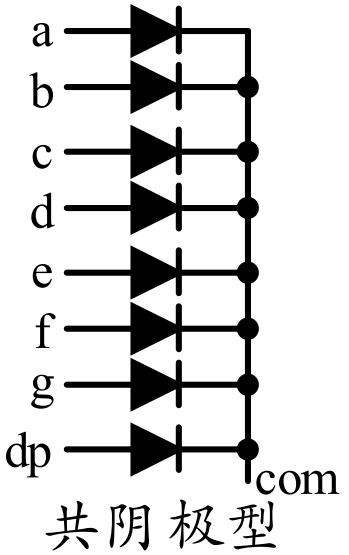
七段数码管



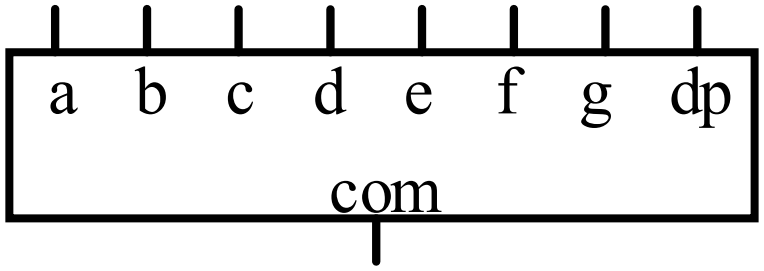




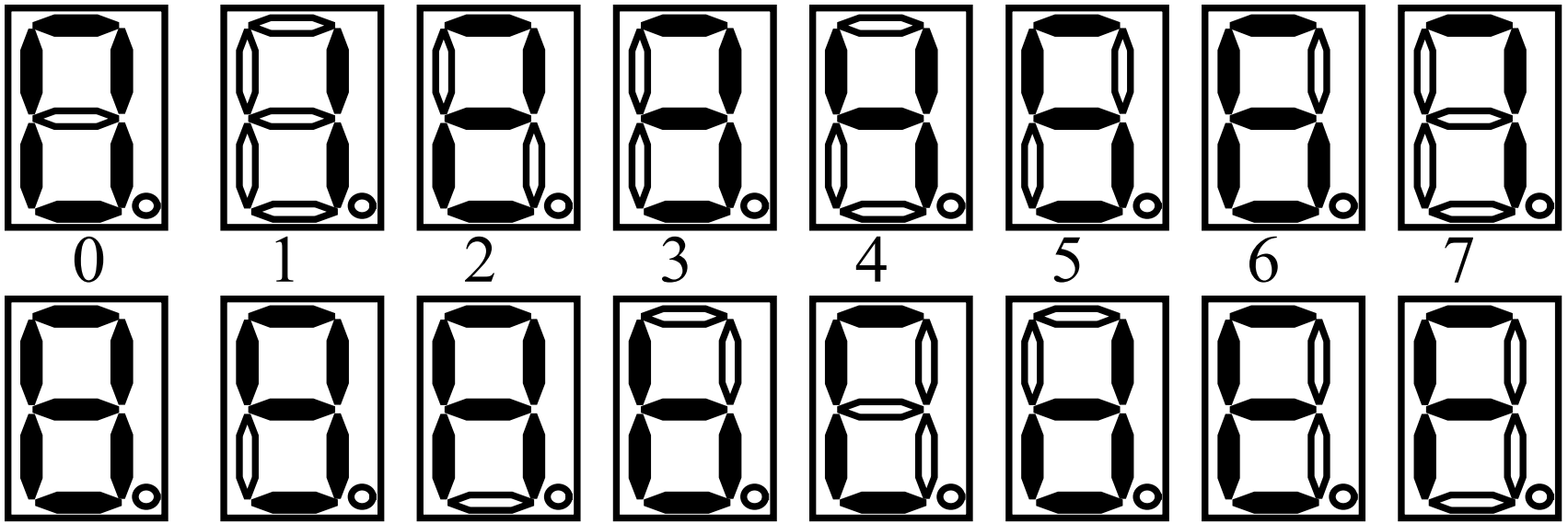
共阳极型



共阴极型



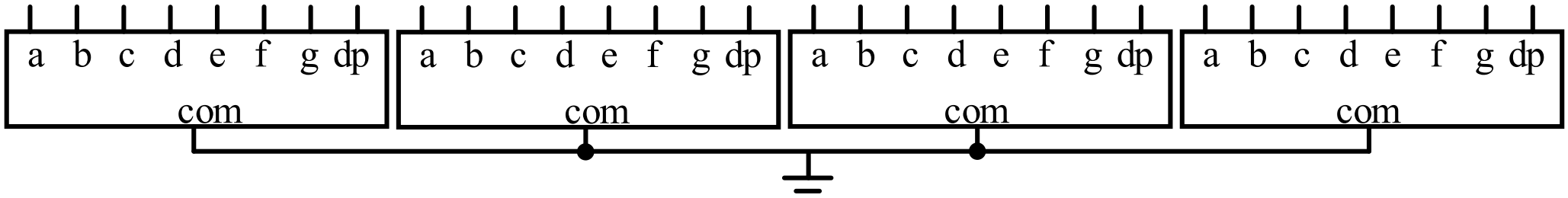
七段数码管十六进制数字字符字型



段码　　字型对应的数字编码,编码值与电路连接相关

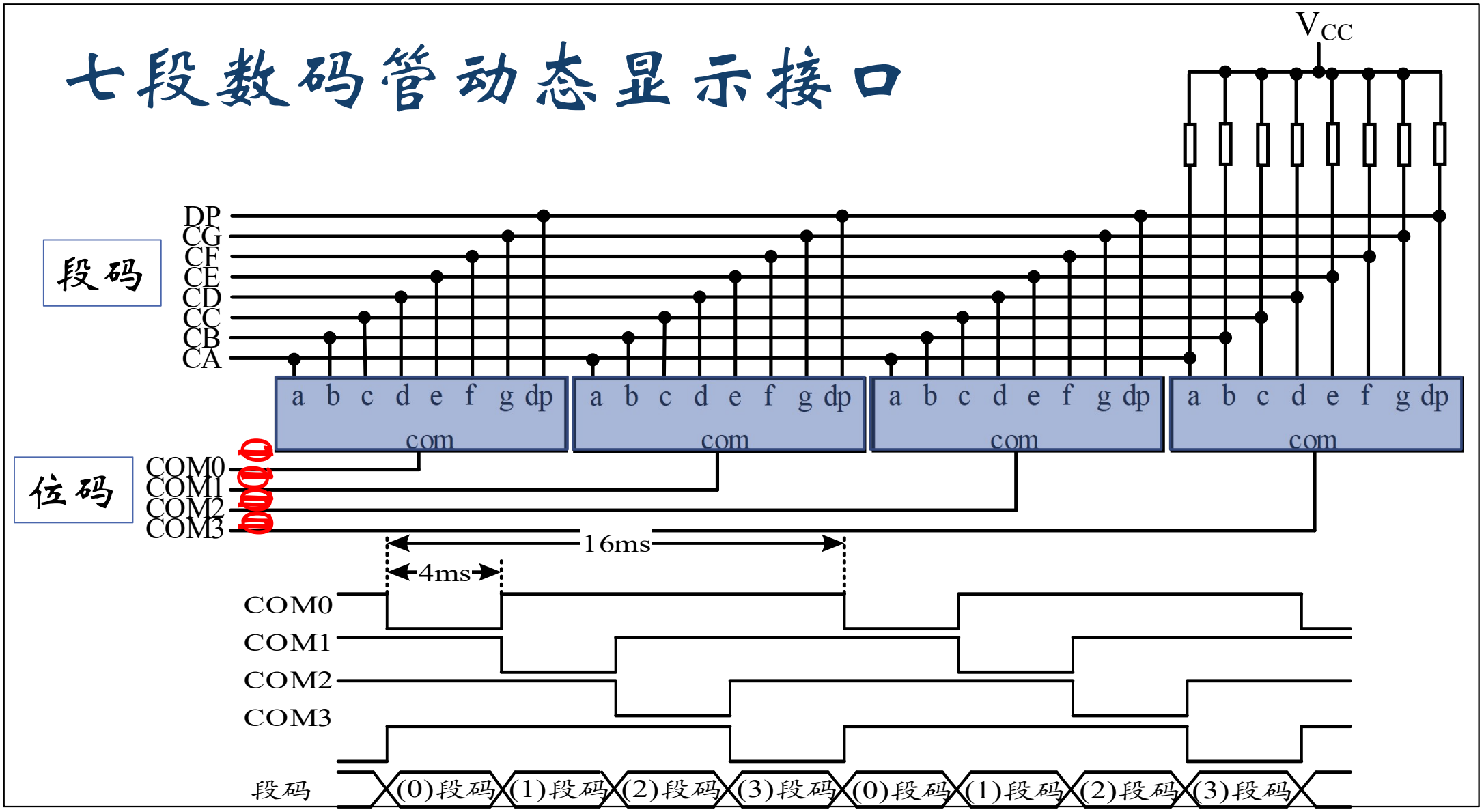
七段数码管静态显示接口

输出引脚多



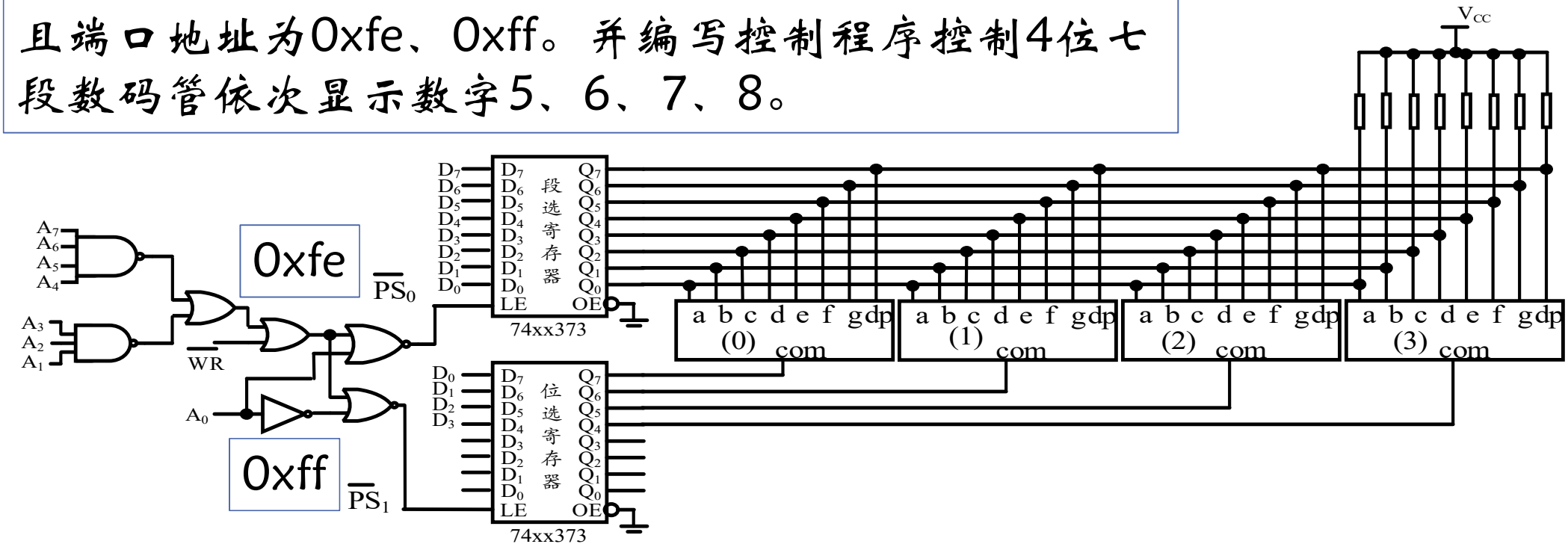
人眼具有“视觉暂留”效应，光的作用结束后，视觉形象并不立即消失

七段数码管动态显示接口



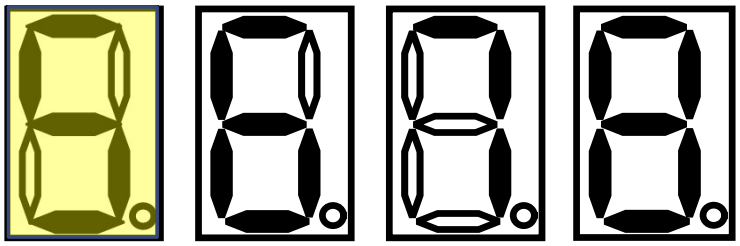
七段数码管应用示例

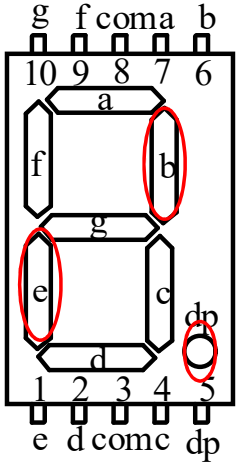
已知某计算机系统具有8位地址总线$A_{7\sim0}$、8位数据总线$D_{7\sim0}$，采用存储器映像IO寻址方式，要求为该计算机系统设计一个4位七段数码管动态显示接口电路，且端口地址为0xfe、0xff。并编写控制程序控制4位七段数码管依次显示数字5、6、7、8。

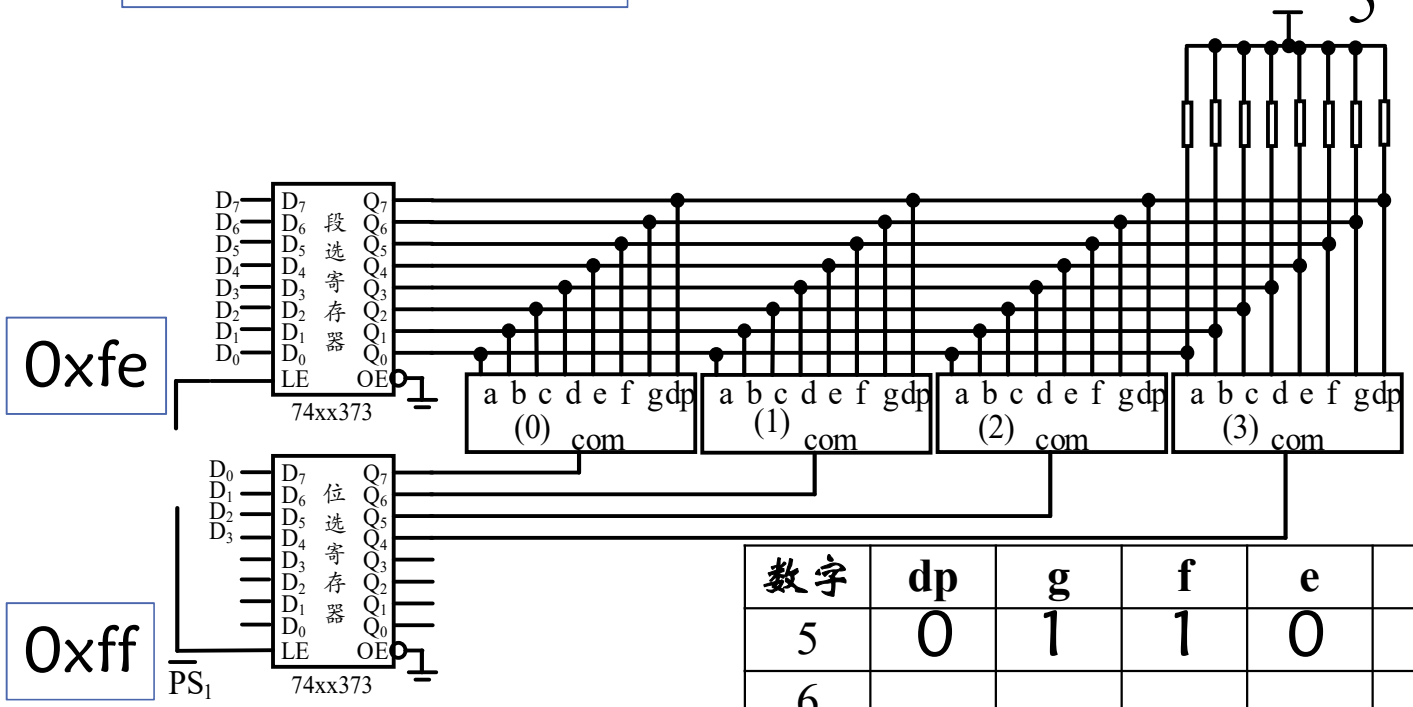


七段数码管应用示例

显示数字5,6,7,8



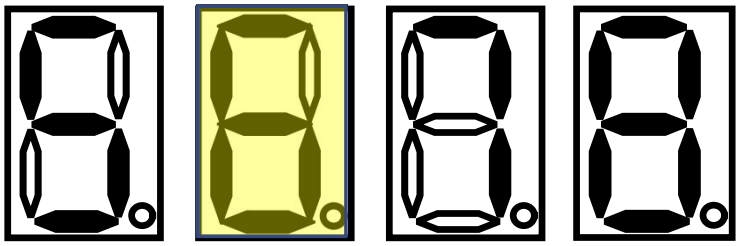


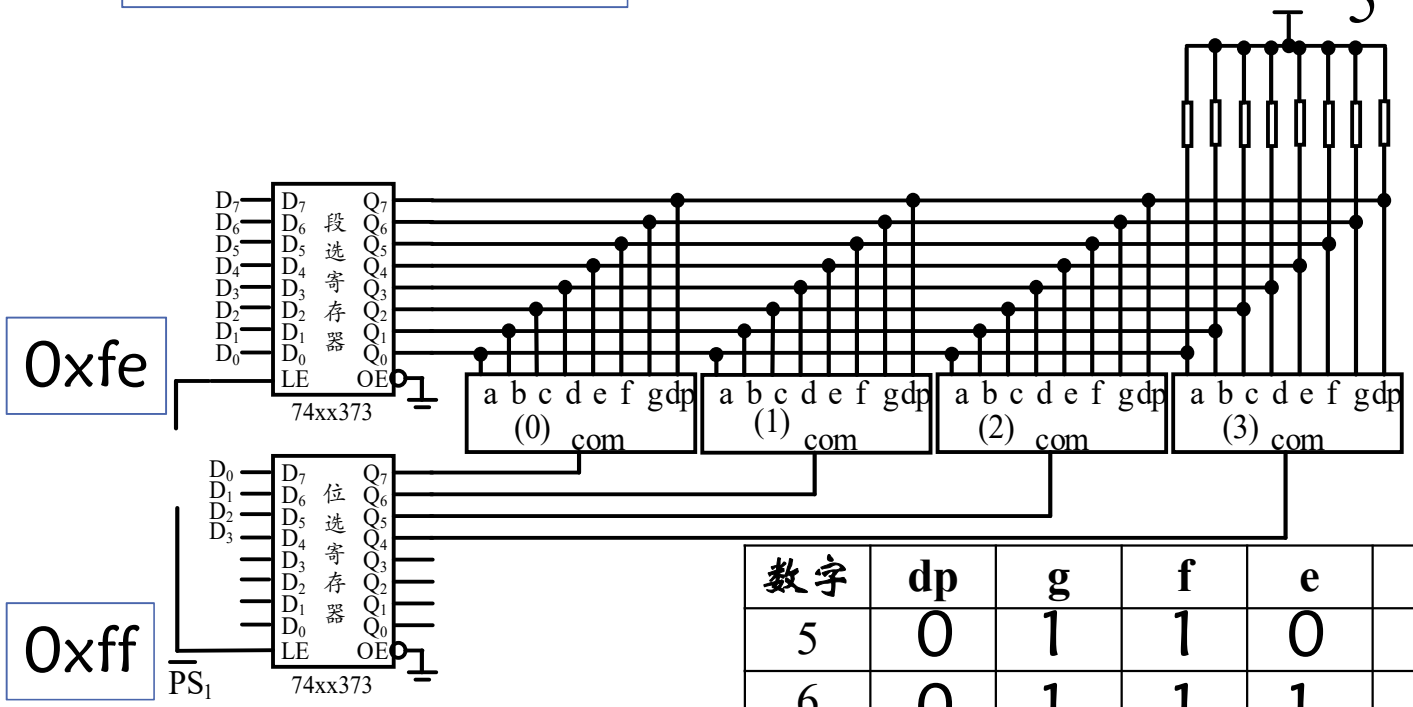


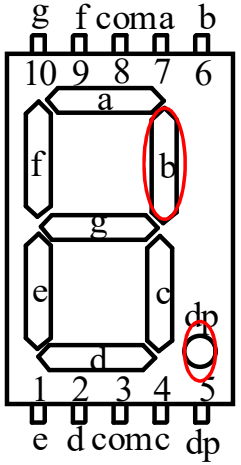
数字	dp	g	f	e	d	c	b	a	段码
5	0	1	1	0	1	1	0	1	0x6d
6									
7									
8									

七段数码管应用示例

显示数字5,6,7,8



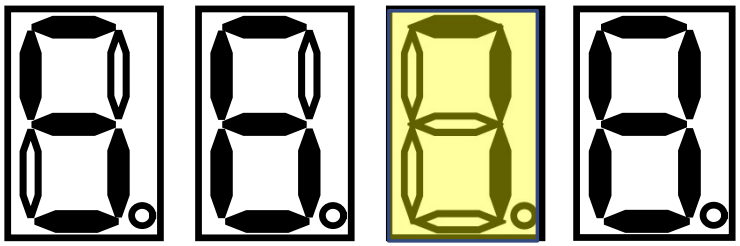


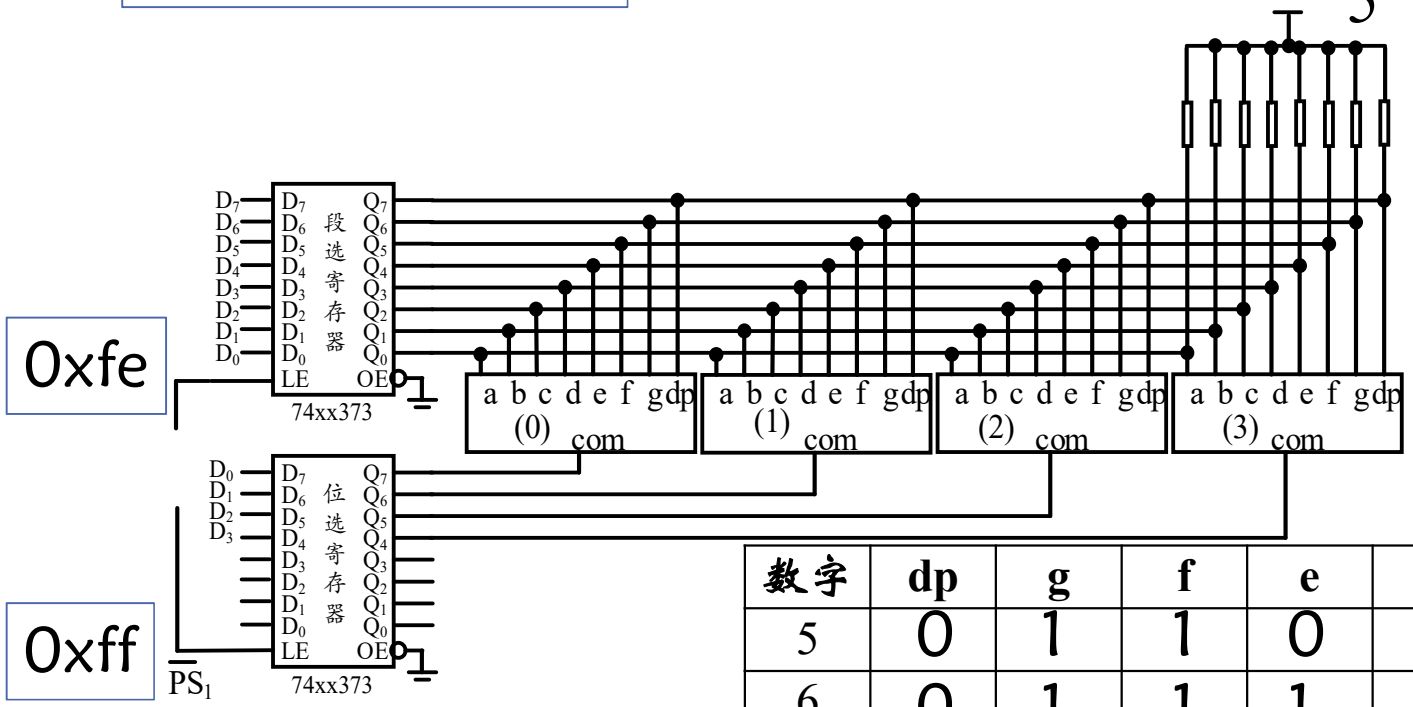


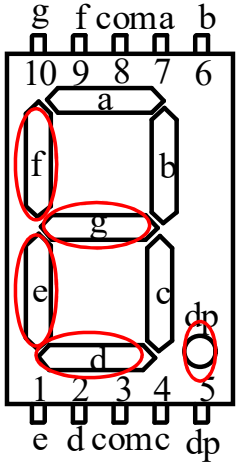
数字	dp	g	f	e	d	c	b	a	段码
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7									
8									

七段数码管应用示例

显示数字5,6,7,8



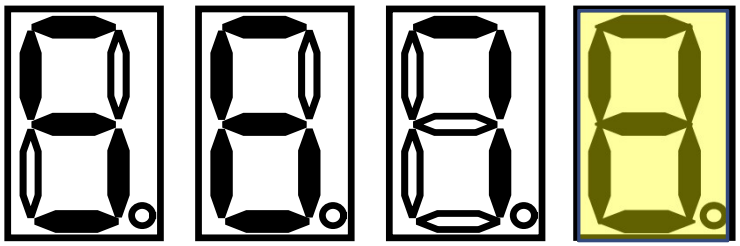


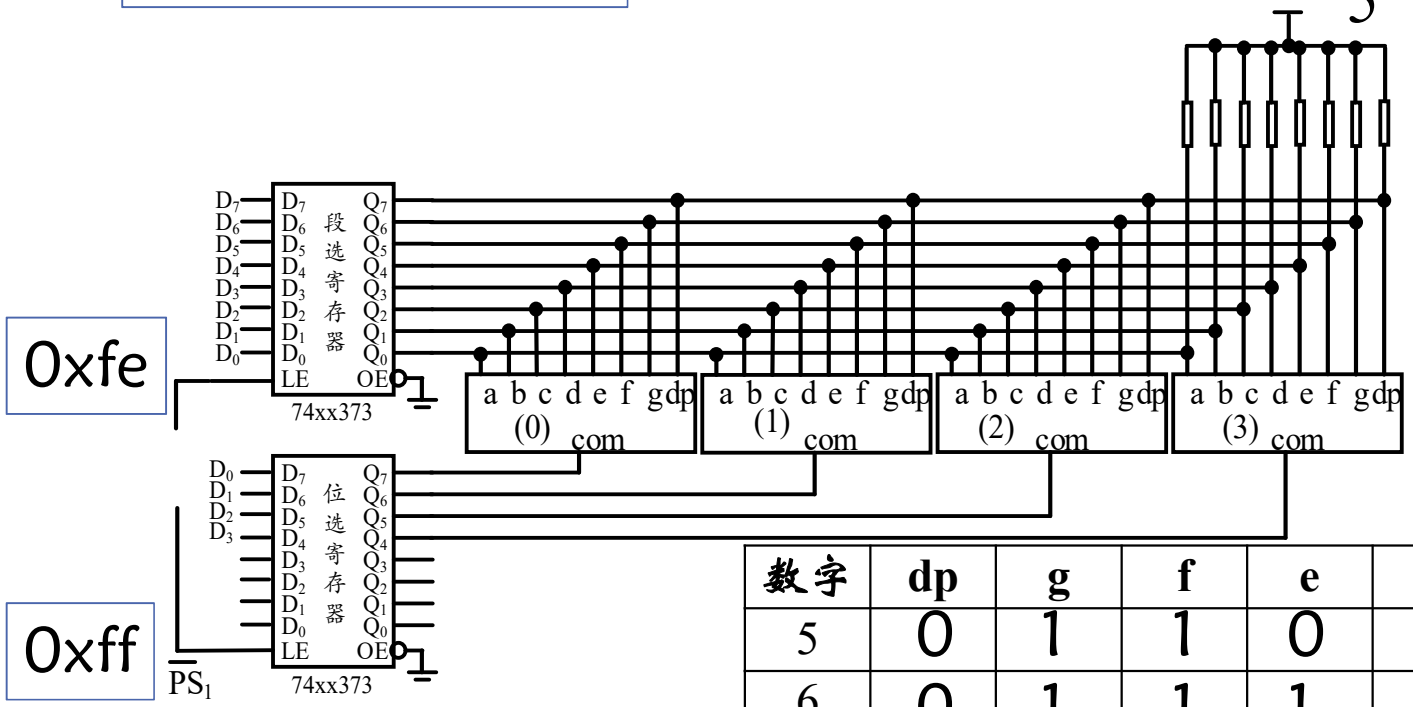


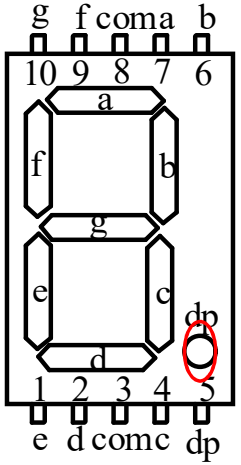
数字	dp	g	f	e	d	c	b	a	段码
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7	0	0	0	0	0	1	1	1	0x07
8									

七段数码管应用示例

显示数字5,6,7,8

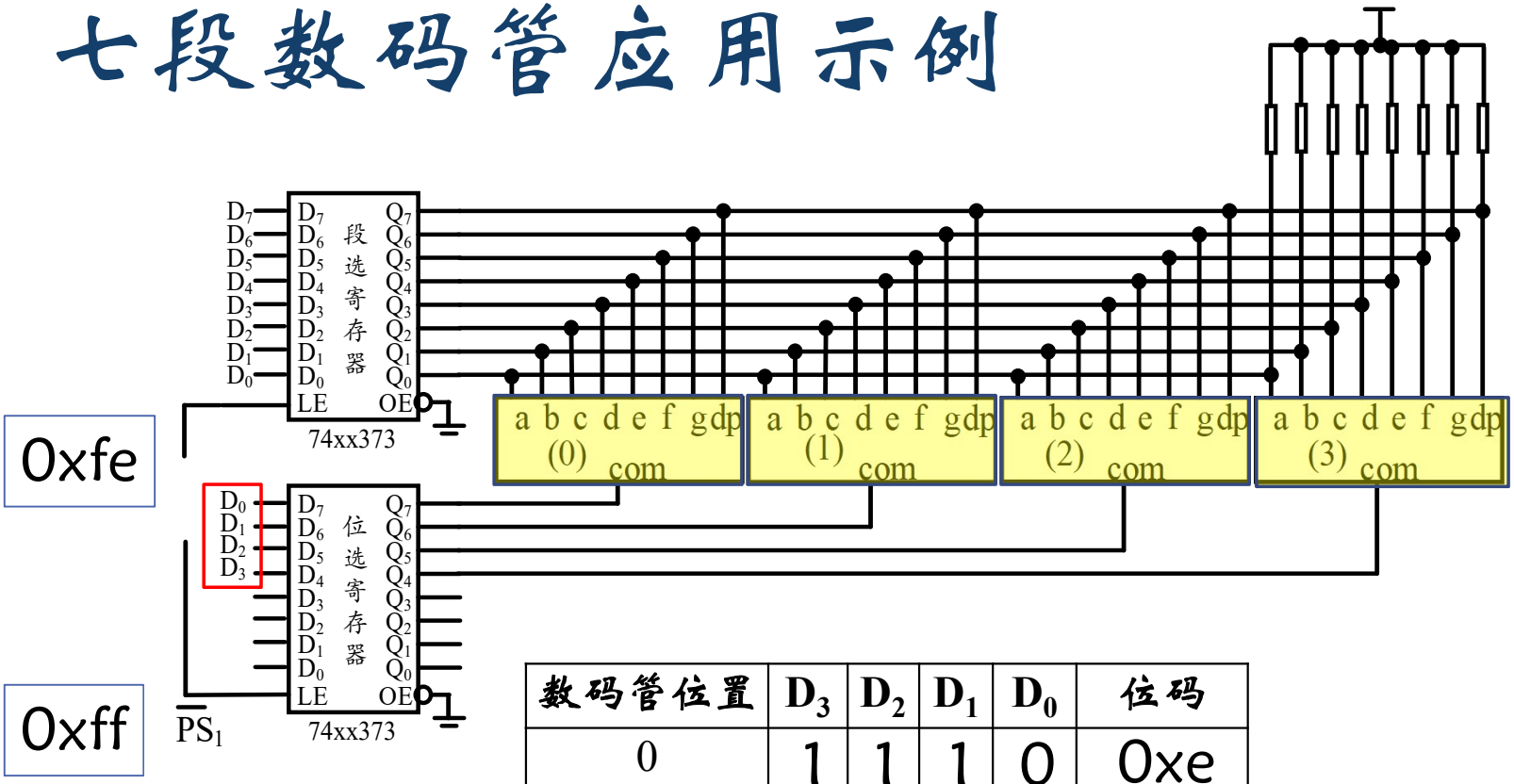






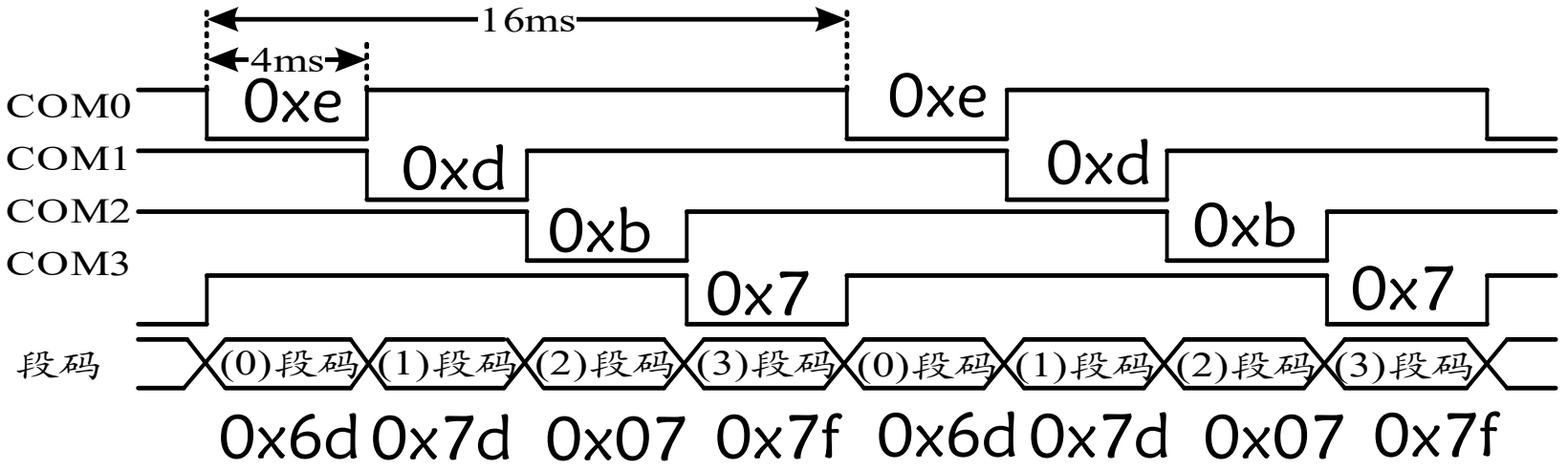
数字	dp	g	f	e	d	c	b	a	段码
5	0	1	1	0	1	1	0	1	0x6d
6	0	1	1	1	1	1	0	1	0x7d
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7f

七段数码管应用示例



数码管位置	D_3	D_2	D_1	D_0	位码
0	1	1	1	0	0xe
1	1	1	0	1	0xd
2	1	0	1	1	0xb
3	0	1	1	1	0x7

七段数码管应用示例


16ms
4ms
COM0
COM1
COM2
COM3
段码
0xe
0xd
0xb
0x7
0xe
0xd
0xb
0x7
(0)段码
(1)段码
(2)段码
(3)段码
(0)段码
(1)段码
(2)段码
(3)段码
0x6d 0x7d 0x07 0x7f 0x6d 0x7d 0x07 0x7f

七段数码管应用示例

显示缓冲区

```
unsigned char seg_code[4]={ 0x6d,0x7d,0x07,0x7f};
unsigned char position=0x01;
Xil_Out8(0xff,0xf);//
while(1) //
{    position=0x01;
       for(int i=0;i<4;i++)
       {       Xil_Out8(0xfe, seg_code[i]); //输出第i位的段码
               Xil_Out8(0xff,~position);  //输出第i位的位码
               delay_ms( ); //延时
               position=position << 1;//位码指向下一个七段数码管
       }
}
```

软件延时

指令周期

指令执行需经历一定的时间

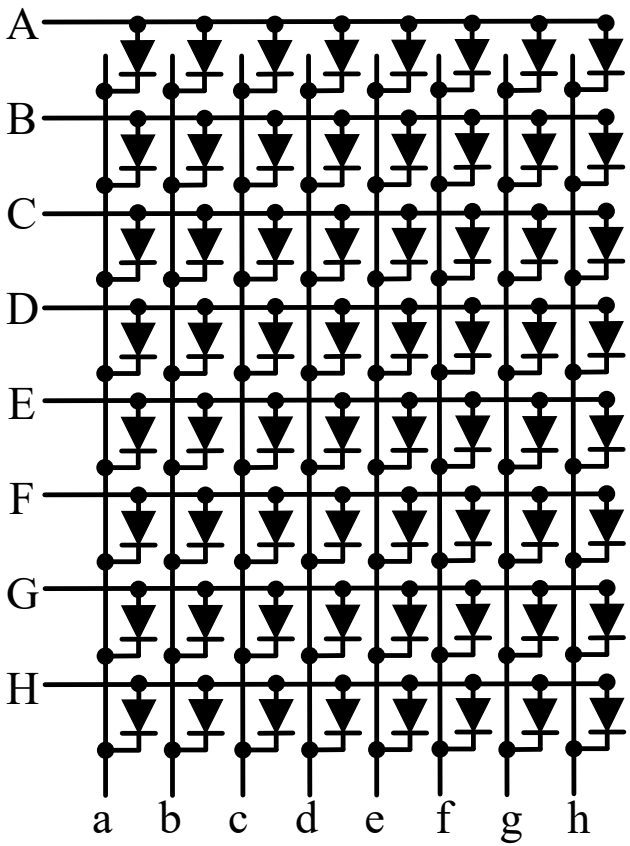
从取指令到存储结果的完整过程需经历的时间，

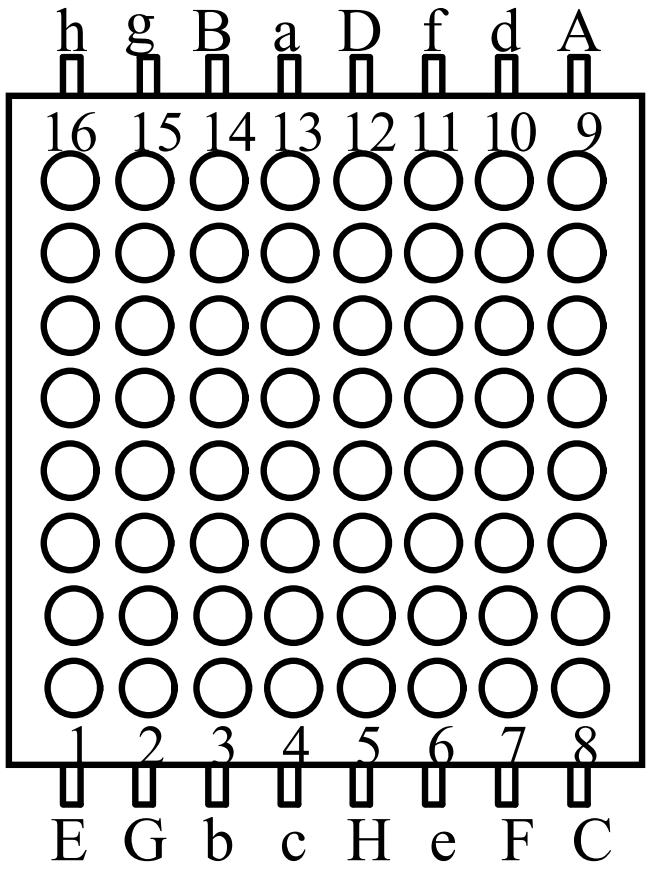
以微处理器时钟周期为单位

微处理器执行不对寄存器、外设以及存储器产生影响的指令

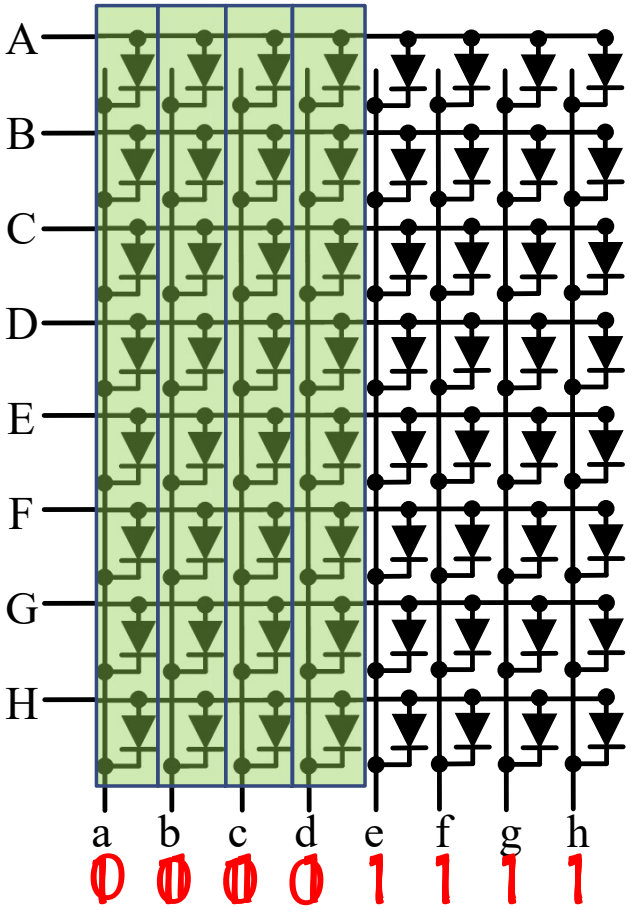
```
void delay_ms( int mil_sec)
{
for(int i=0;i<mil_sec*0x10000;i++);
}
```

LED点阵接口


A
B
C
D
E
F
G
H
a b c d e f g h

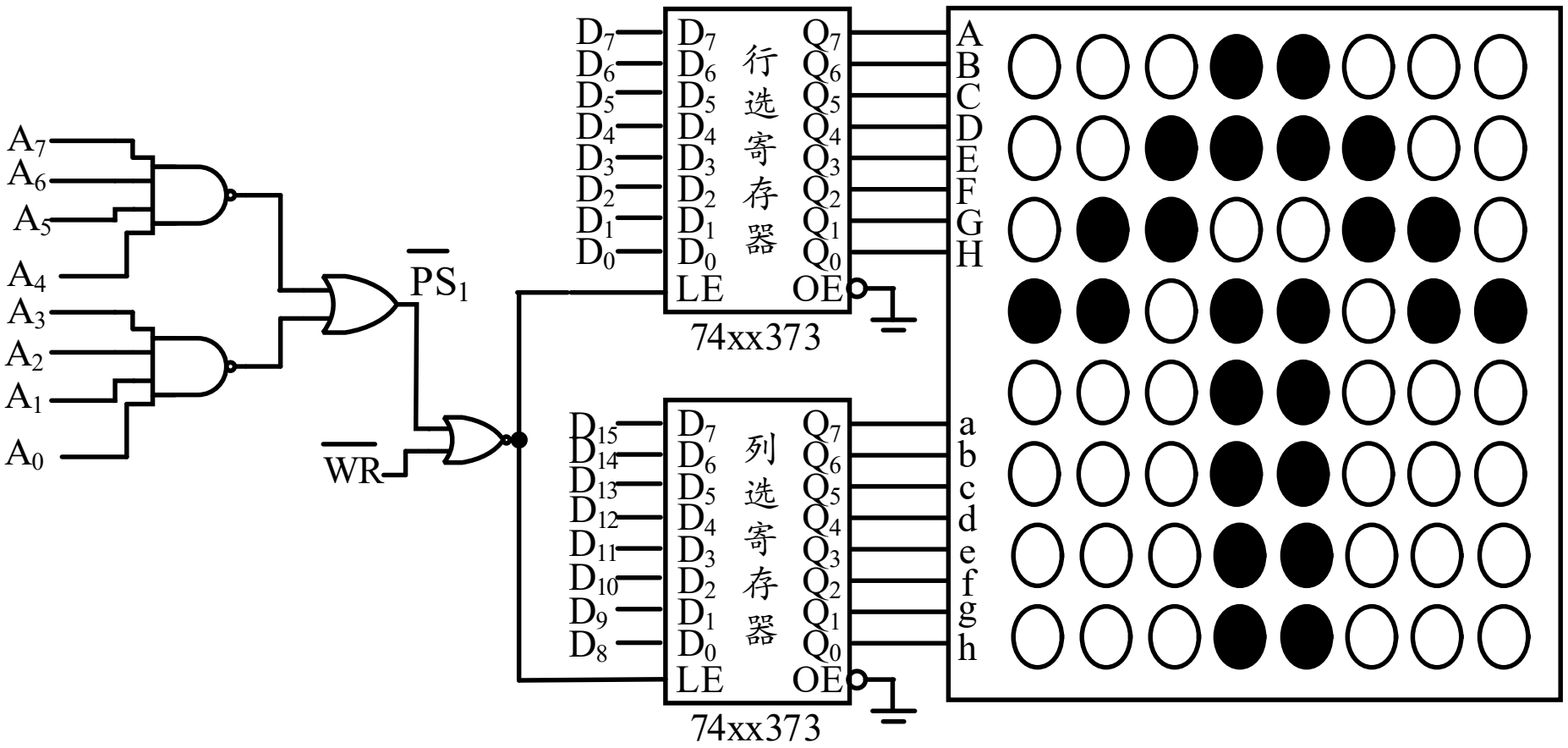

h g B a D f d A
16 15 14 13 12 11 10 9
1 2 3 4 5 6 7 8
E G b c H e F C

LED点阵工作原理



人眼视觉暂留效应

LED点阵接口


A7
A6
A5
A4
A3
A2
A1
A0
PS1
WR
D7
D6
D5
D4
D3
D2
D1
D0
LE
行选寄存器
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
OE
74xx373
A
B
C
D
E
F
G
H
D15
D14
D13
D12
D11
D10
D9
D8
列选寄存器
a
b
c
d
e
f
g
h

LED点阵接口

```
unsigned char Col_Code[8]={0xe7,0xc3,0x99,0x24,0xe7,0xe7,0xe7,0xe7};
unsigned char Rol_Code=0x80;
unsigned short code;
while(1) //无限循环控制8×8 LED点阵显示图案
{    Rol_Code=0x80;
for(int i=0;i<8;i++)
{
   code=((unsigned short) Col_Code [i]<<8)| Rol_Code;
         Xil_Out16(0xff,code);  //输出第i行的行选和列编码
         delay_ms( ); //延时
      Rol_Code = Rol_Code >> 1;
    }
}
```

小结

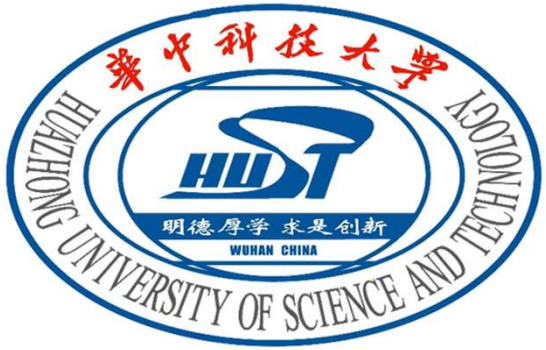
- 并行数字IO设备接口
  - 输入缓冲
  - 输出锁存
  - 动态扫描
  - 人眼视觉暂留效应-循环扫描
  - 软件延时

下一讲：模拟设备并行IO接口

微机原理与接口技术

模拟设备并行IO接口

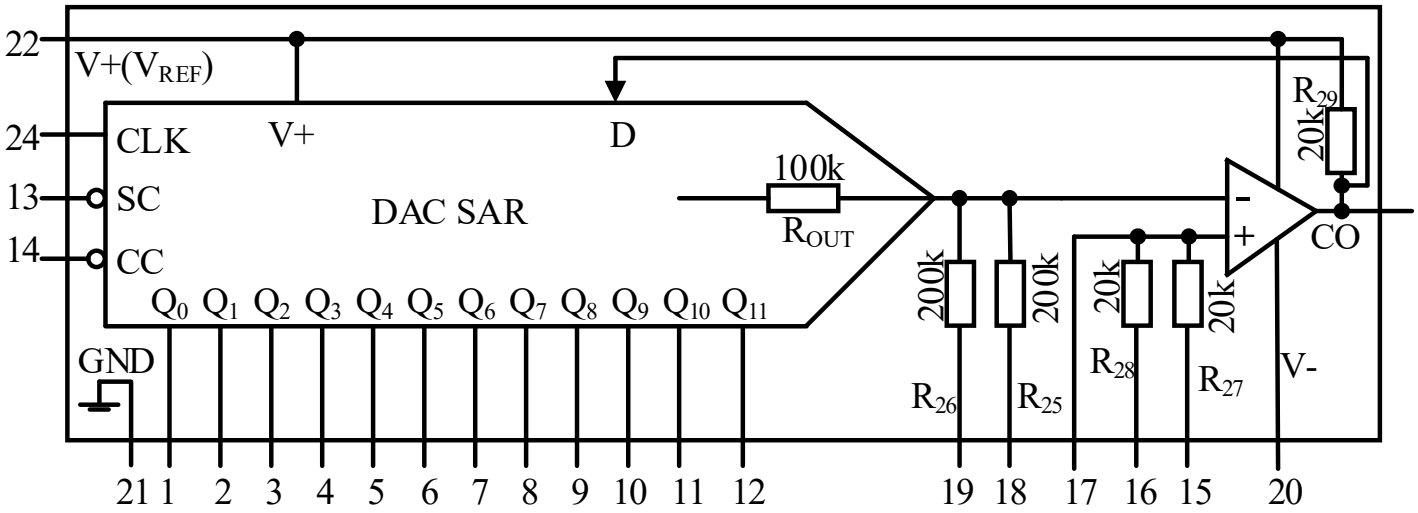
华中科技大学　左冬红

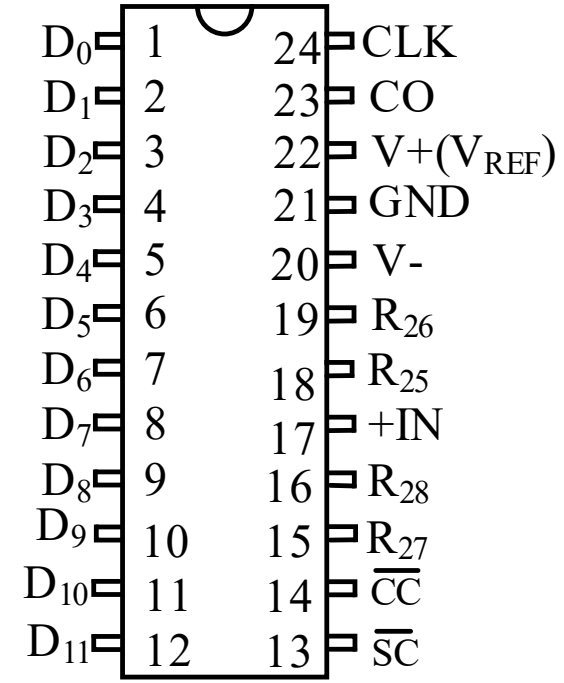


模拟设备

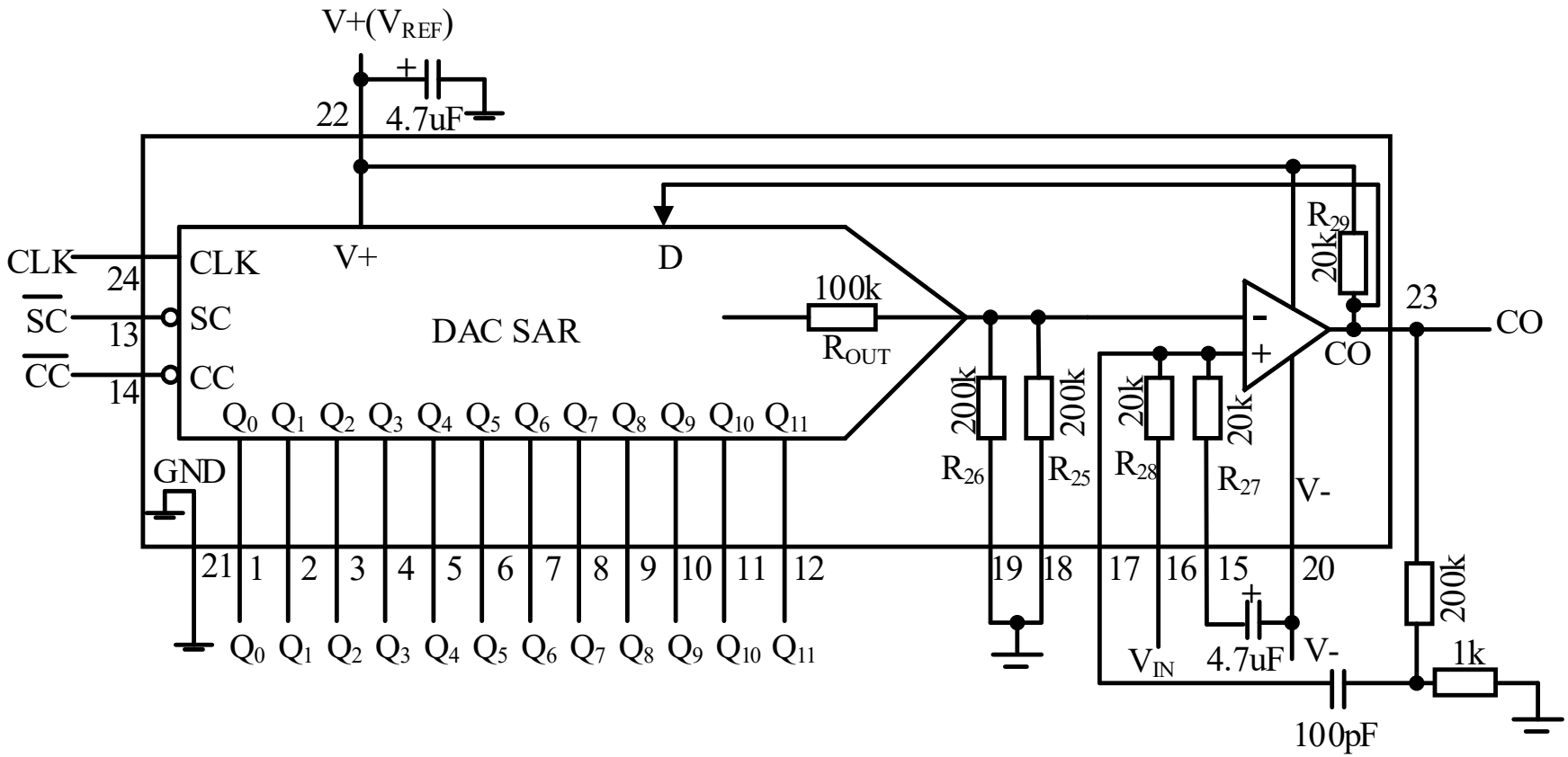
模拟信号接入计算机系统处理必须经过AD转换器

ADC1210


22
V+(VREF)
24
CLK
13
SC
14
CC
V+
D
DAC SAR
100k
ROUT
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11
GND
21 1 2 3 4 5 6 7 8 9 10 11 12
200k
200k
20k
20k
R26
R25
R28
R27
R29
20k
CO
V-
19 18 17 16 15 20

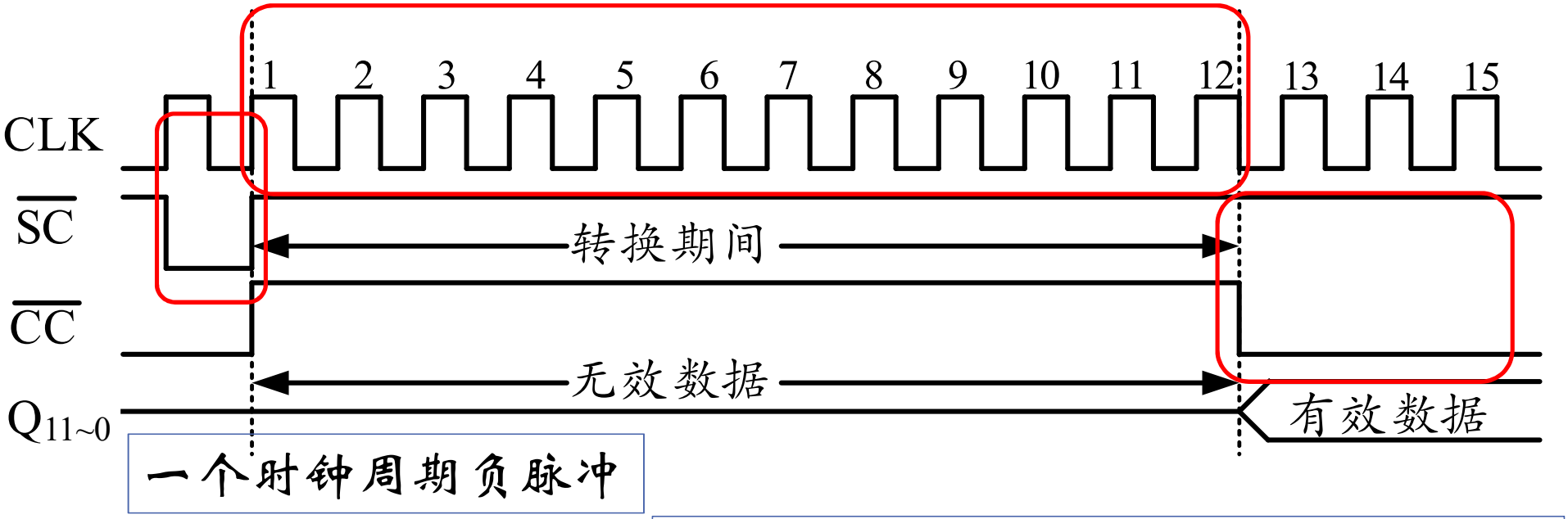

D0 1
D1 2
D2 3
D3 4
D4 5
D5 6
D6 7
D7 8
D8 9
D9 10
D10 11
D11 12
24 CLK
23 CO
22 V+(VREF)
21 GND
20 V-
19 R26
18 R25
17 +IN
16 R28
15 R27
14 CC
13 SC

正逻辑、单极性正向模拟信号输入



ADC1210工作时序

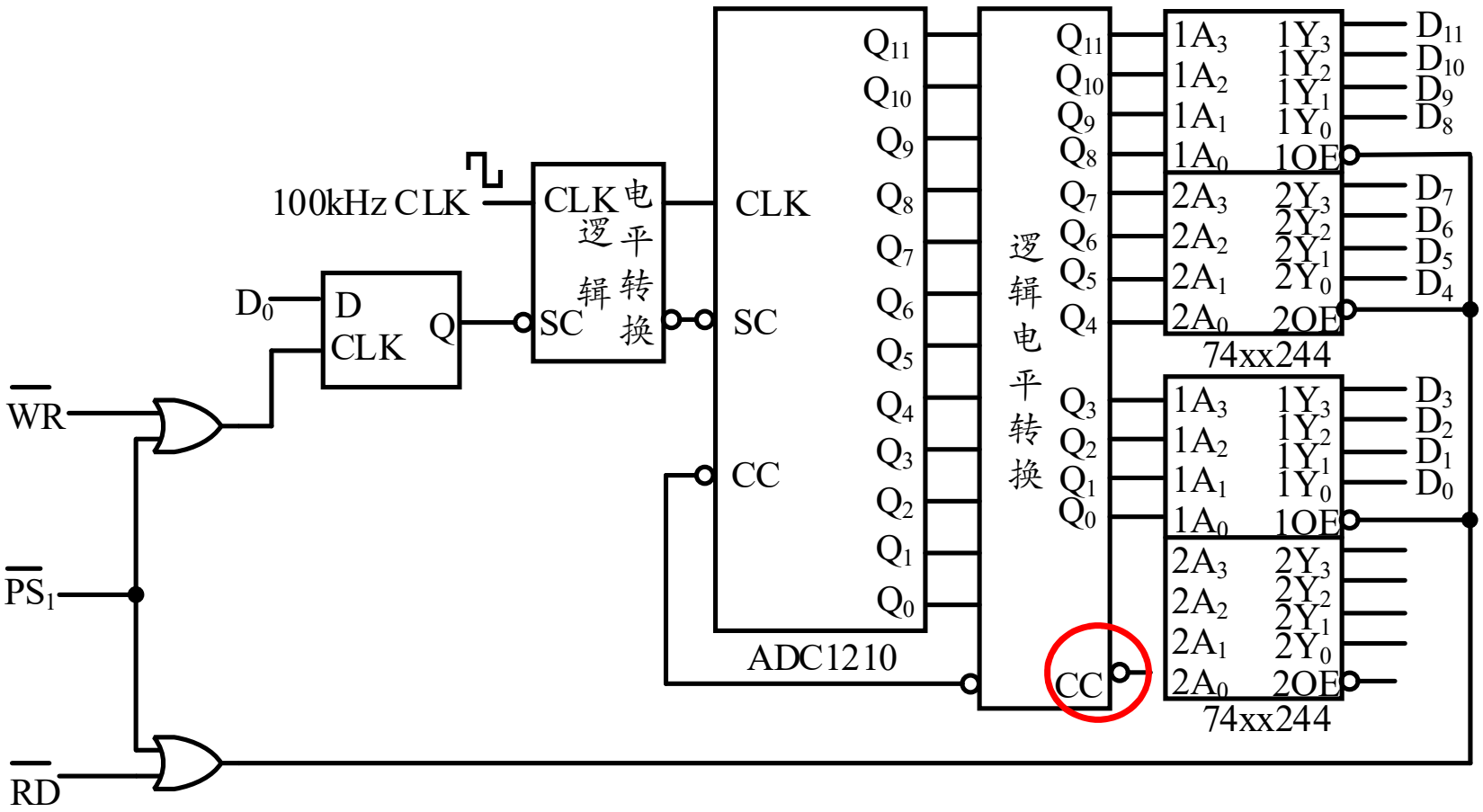
延时等待之后读取转换结果



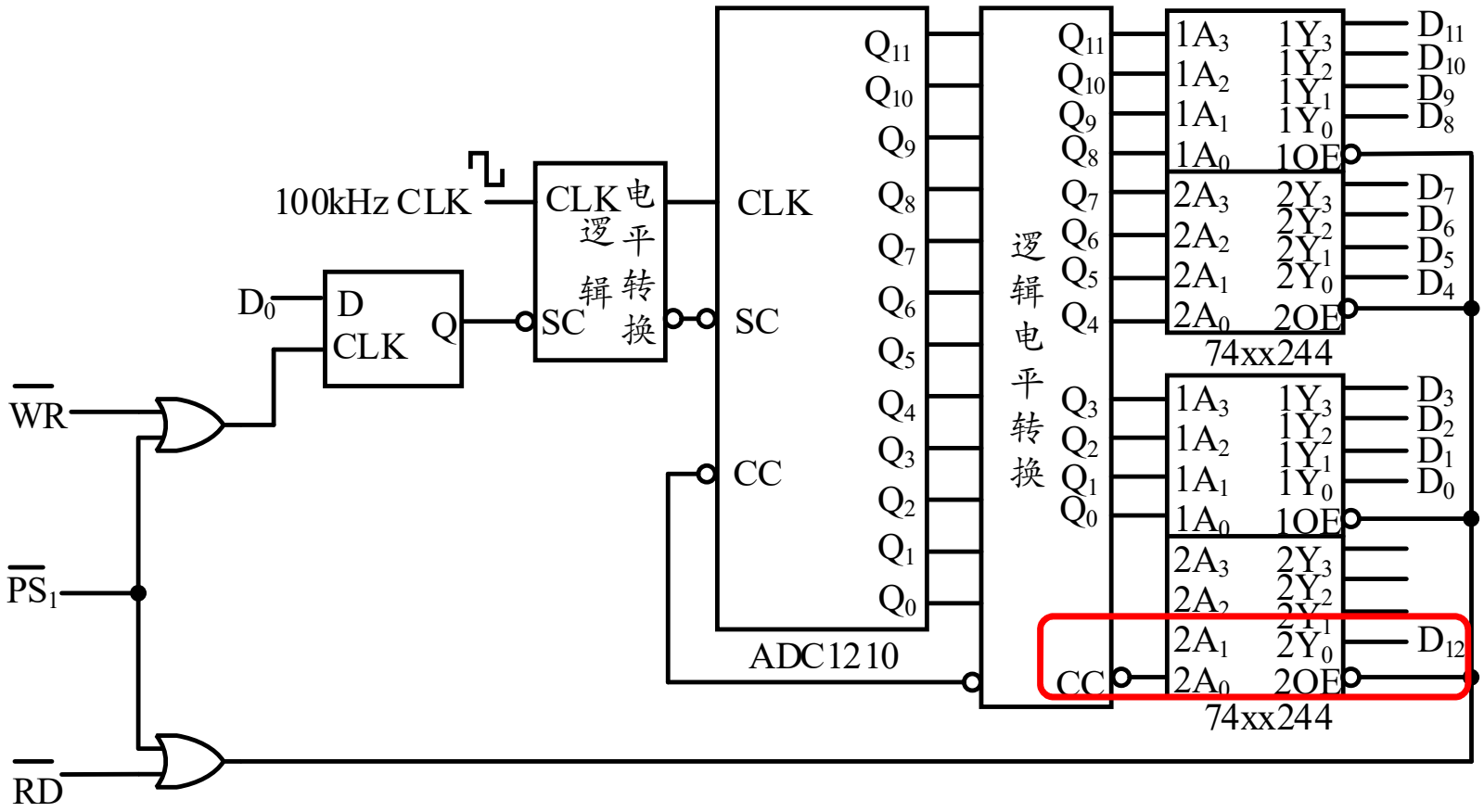
一个时钟周期负脉冲

查询CC，发现变低之后读取转换结果

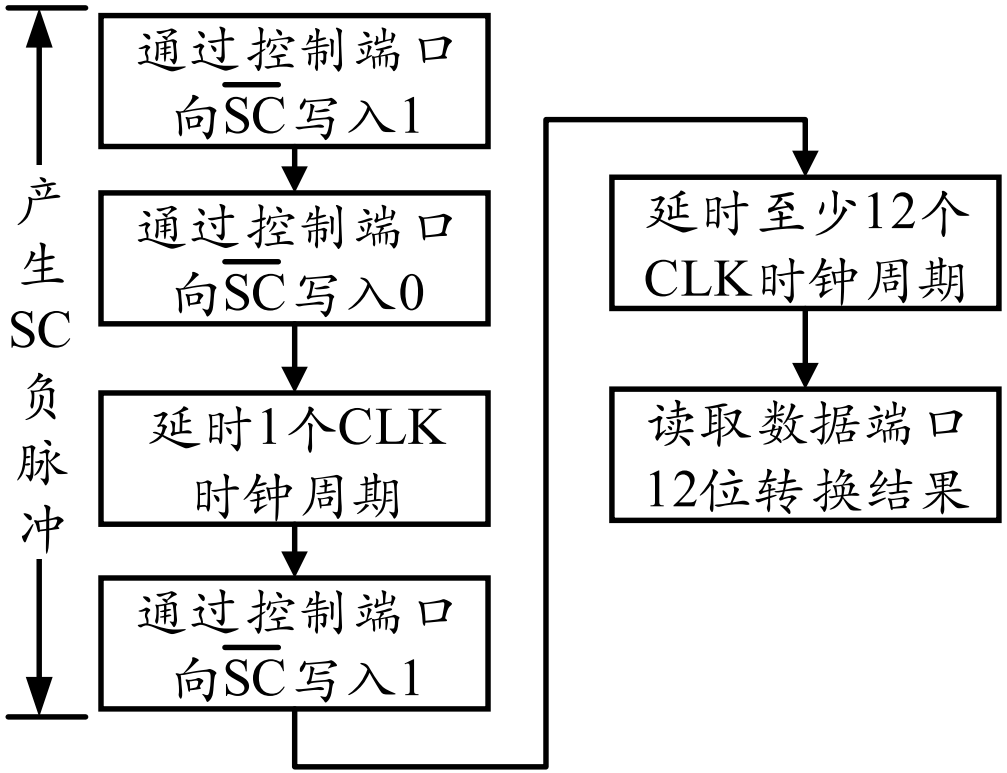
ADC1210延时接口电路



ADC1210查询接口电路

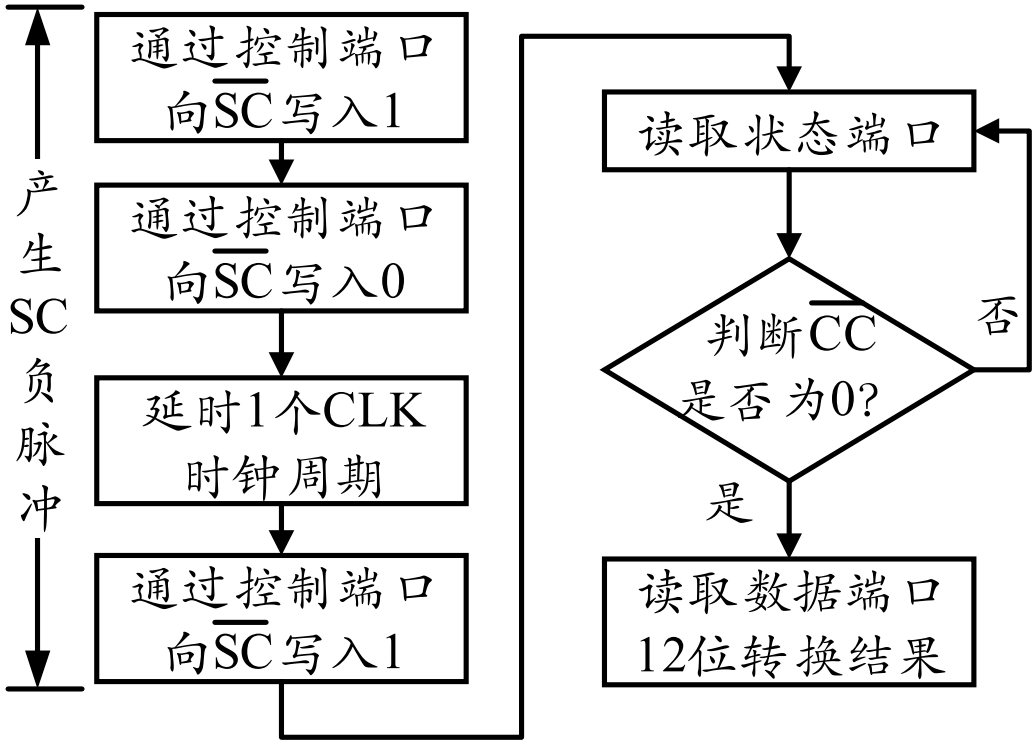


延时程序流程-采集12个数据




```
unsigned short volt[12];
for(int i=0;i<12;i++)//
{
Xil_Out8(0xff,0x1);//
Xil_Out8(0xff,0x0);
delay_10us();//
Xil_Out8(0xff,0x1);
delay_12us();//
volt[i]=Xil_In16(0xff)&0xfff;//}
```

查询程序流程-采集12个数据




```
unsigned short volt[12];
for(int i=0;i<12;i++)
 {
Xil_Out8(0xff,0x1);
Xil_Out8(0xff,0x0);
delay_10us();
Xil_Out8(0xff,0x1);
while((Xil_In16(0xff)&0x1000)!=0);
volt[i]=Xil_In16(0xff)&0xfff;
}
```

小结

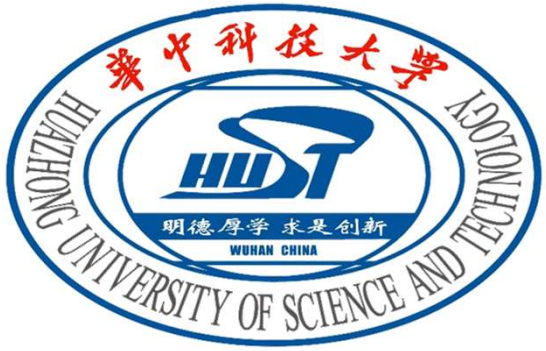
- 模拟设备并行IO接口
  - 控制端口-锁存
  - 状态端口-缓冲
  - 数据端口-缓冲

下一讲：通用并行IO接口

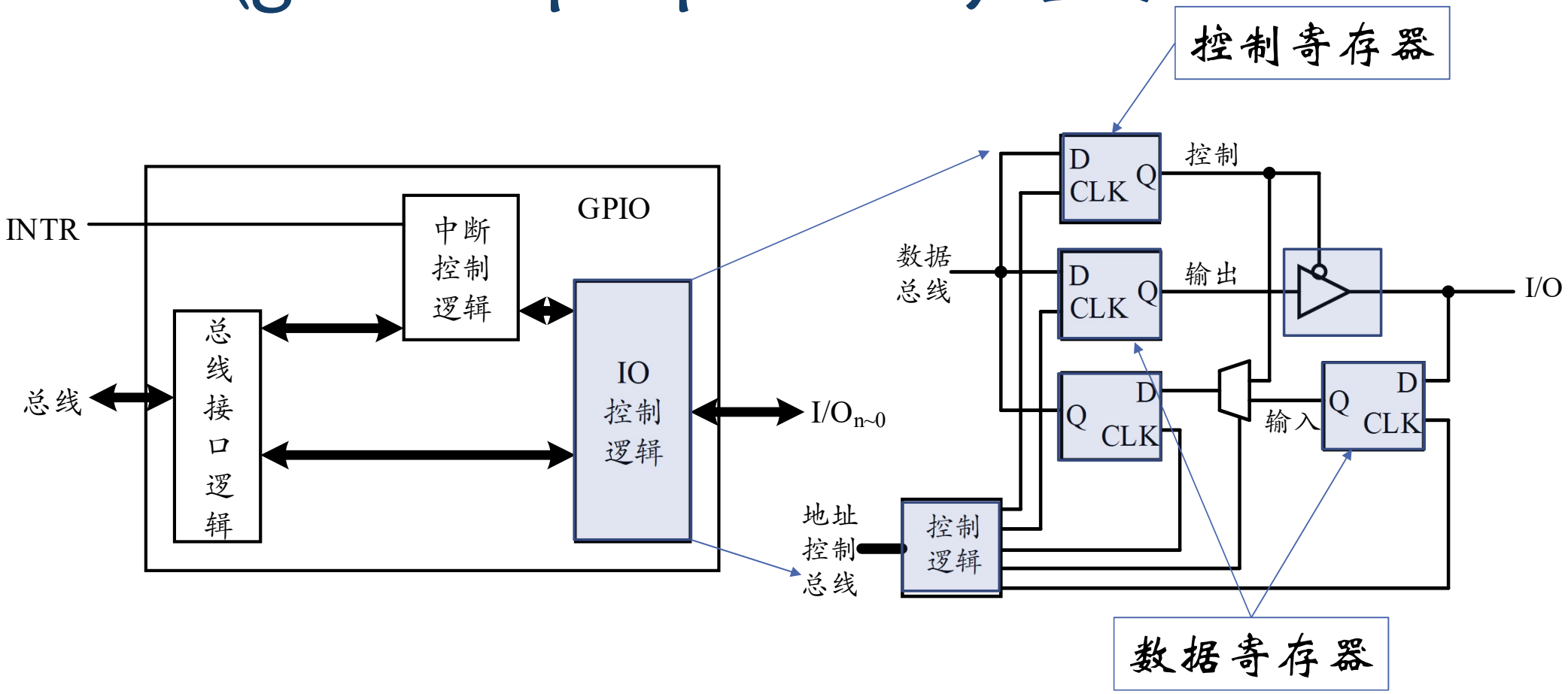
微机原理与接口技术

通用并行IO接口GPIO

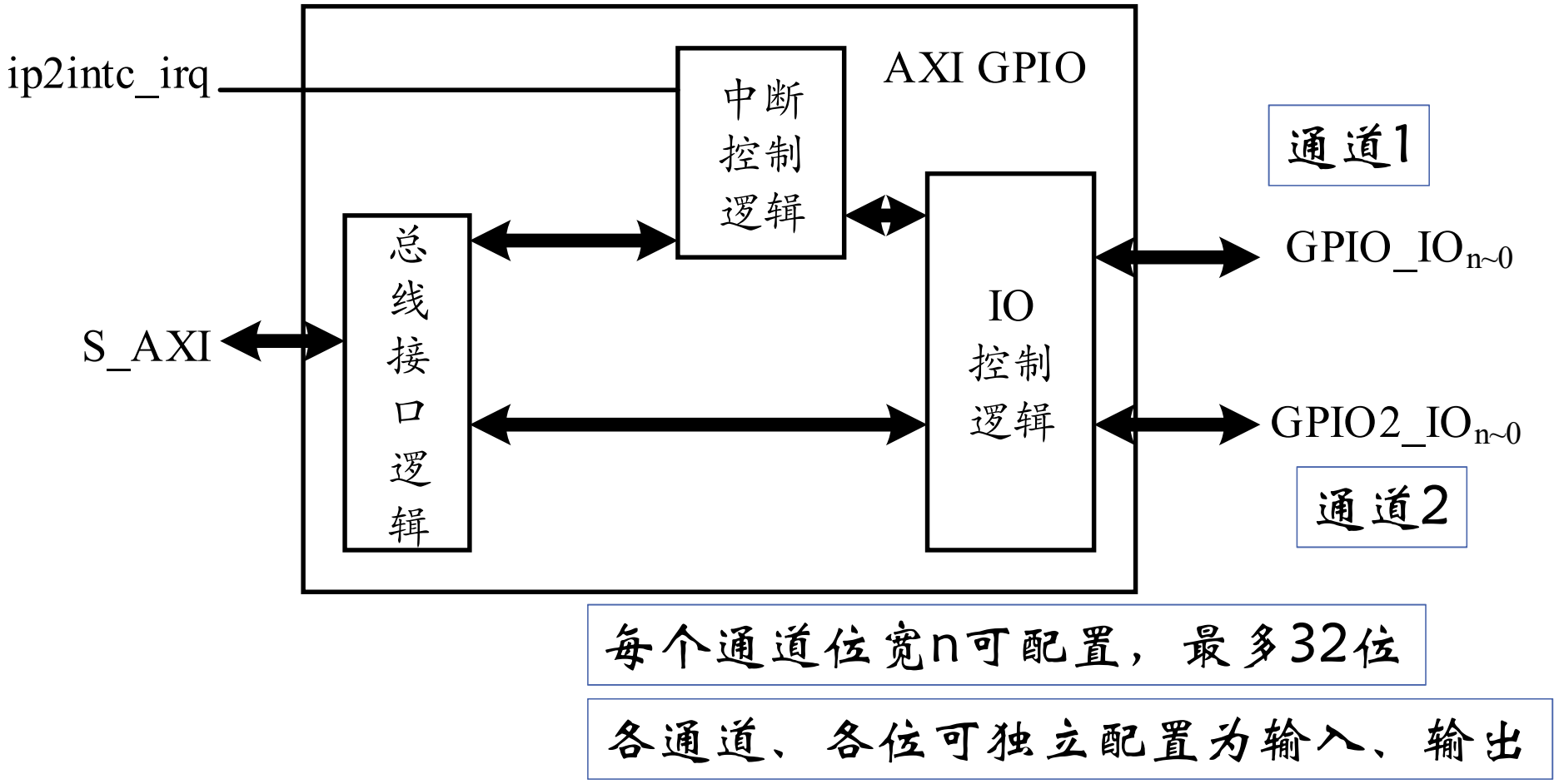
华中科技大学　左冬红



GPIO(general purpose IO)结构



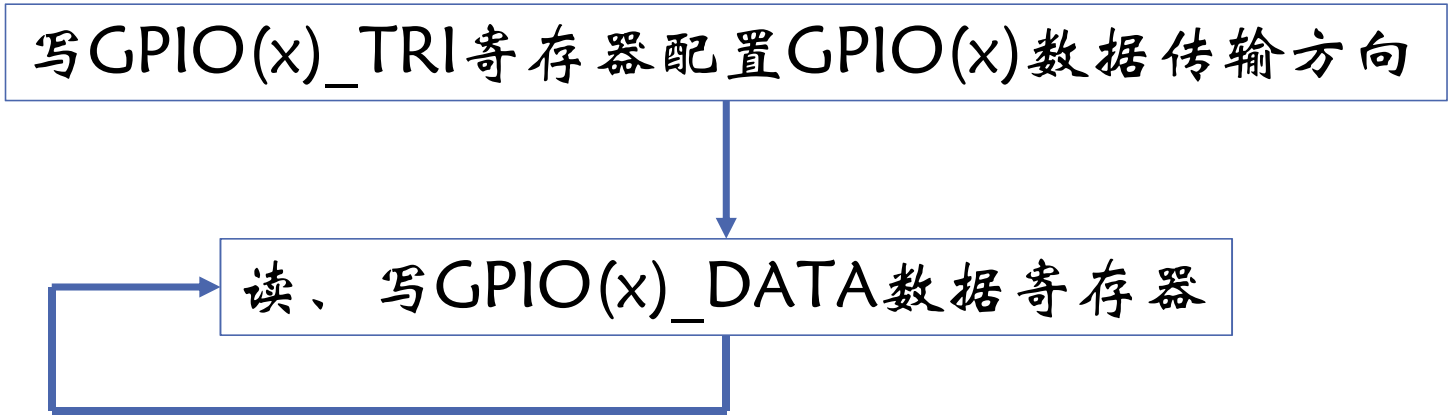
AXI GPIO


ip2intc_irq
中断
控制
逻辑
AXI GPIO
通道1
总
线
接
口
逻
辑
IO
控制
逻辑
GPIO_IO n~0
S_AXI
GPIO2_IO n~0
通道2
每个通道位宽n可配置，最多32位
各通道、各位可独立配置为输入、输出

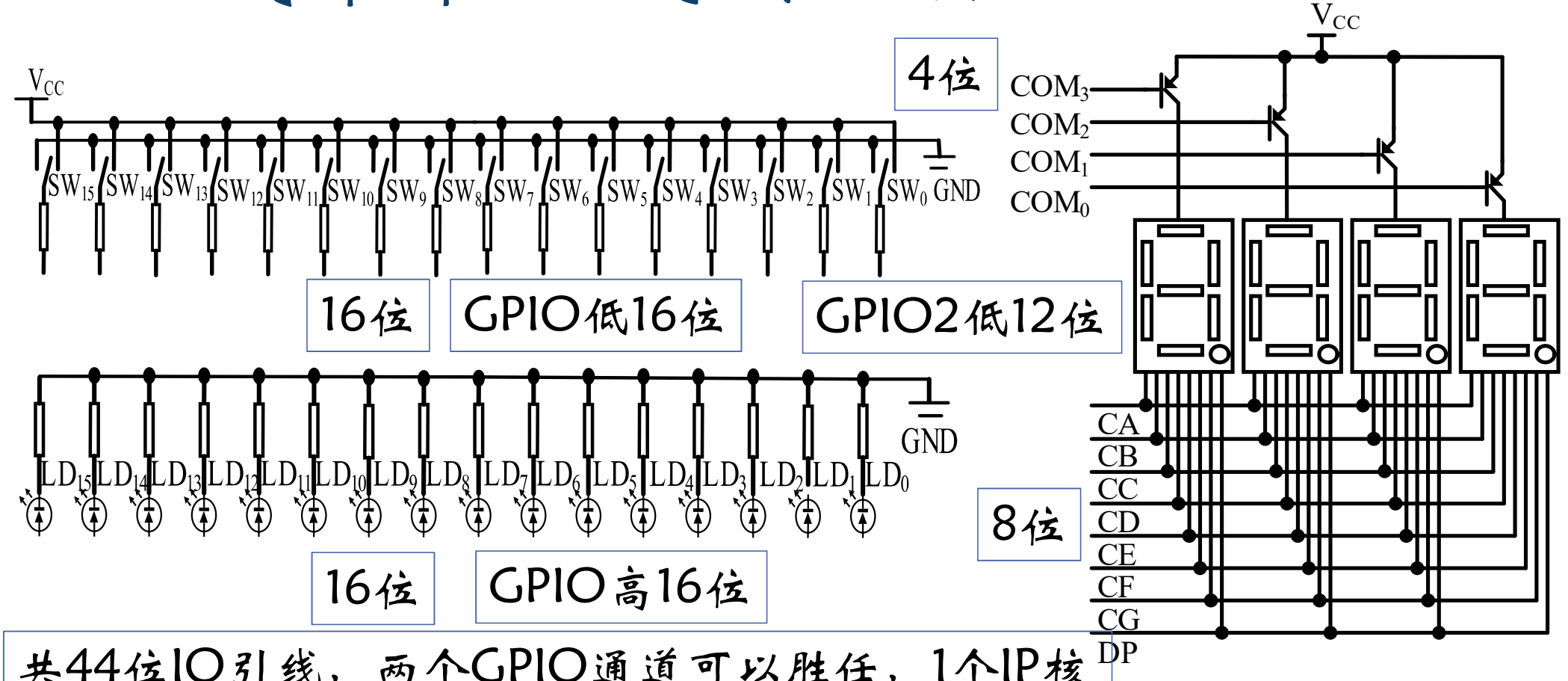
GPIO寄存器存储空间映射

寄存器名称	偏移地址	初始值	含义
GPIO_DATA	0x0	0	32位宽，通道GPIO_IO数据寄存器，D_i对应$GPIO_IO_i$
GPIO_TRI	0x4	0xffffffff	32位宽，通道GPIO_IO控制寄存器，D_i对应$GPIO_IO_i$的传输方向控制。1-输入；0-输出。
GPIO2_DATA	0x8	0	32位宽，通道GPIO2_IO数据寄存器，D_i对应$GPIO2_IO_i$
GPIO2_TRI	0xc	0xffffffff	32位宽，通道GPIO2_IO控制寄存器，D_i对应$GPIO2_IO_i$的传输方向控制。1-输入；0-输出。

GPIO程序控制流程



GPIO简单并行数字外设接口电路示例



4位

16位 GPIO低16位 GPIO2低12位

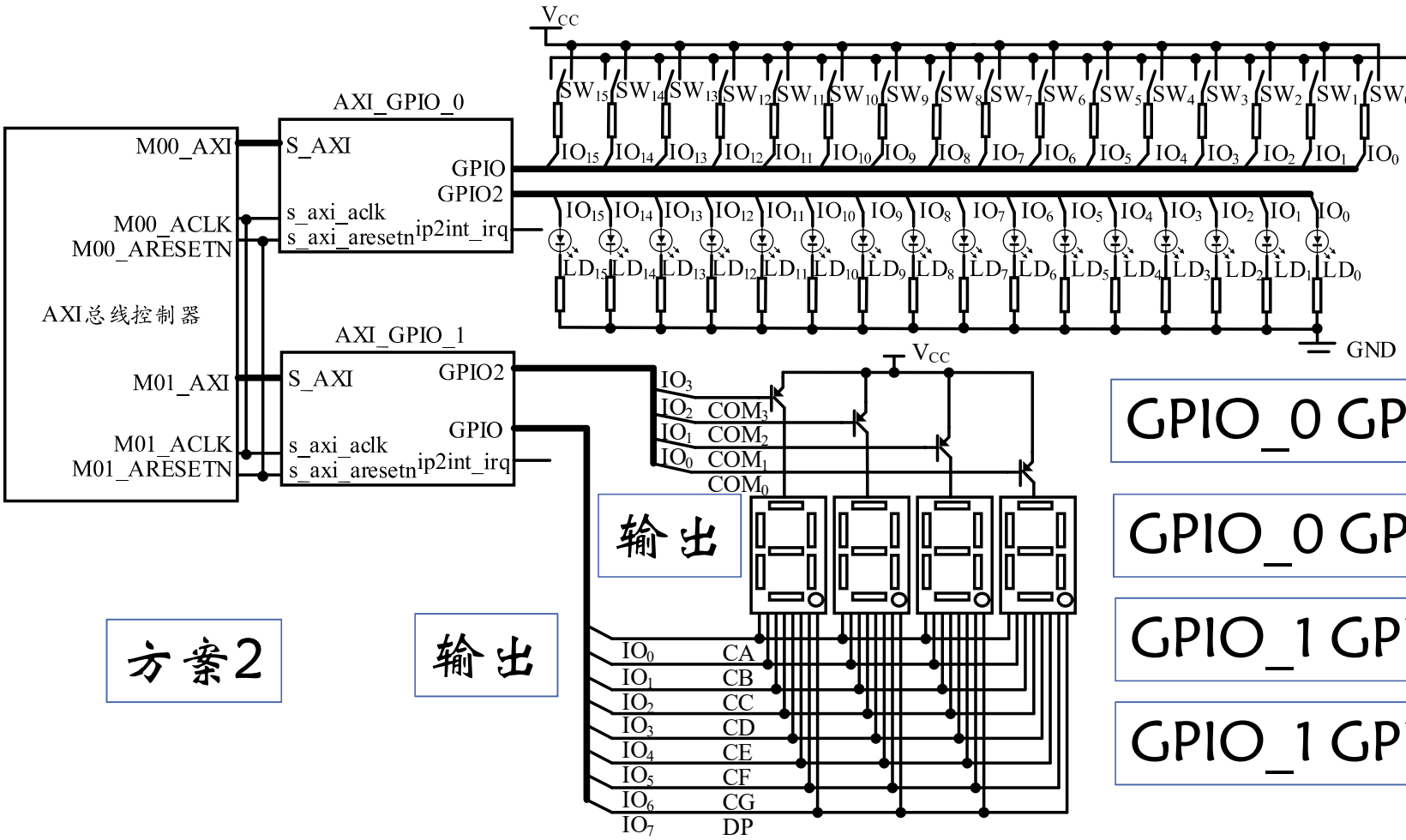
16位 GPIO高16位

8位

共44位IO引线，两个GPIO通道可以胜任，1个IP核

不同外设采用不同GPIO通道，2个IP核

GPIO简单并行数字外设接口电路



GPIO_0 GPIO_TRI 0xffff

GPIO_0 GPIO2_TRI 0x0

GPIO_1 GPIO_TRI 0x0

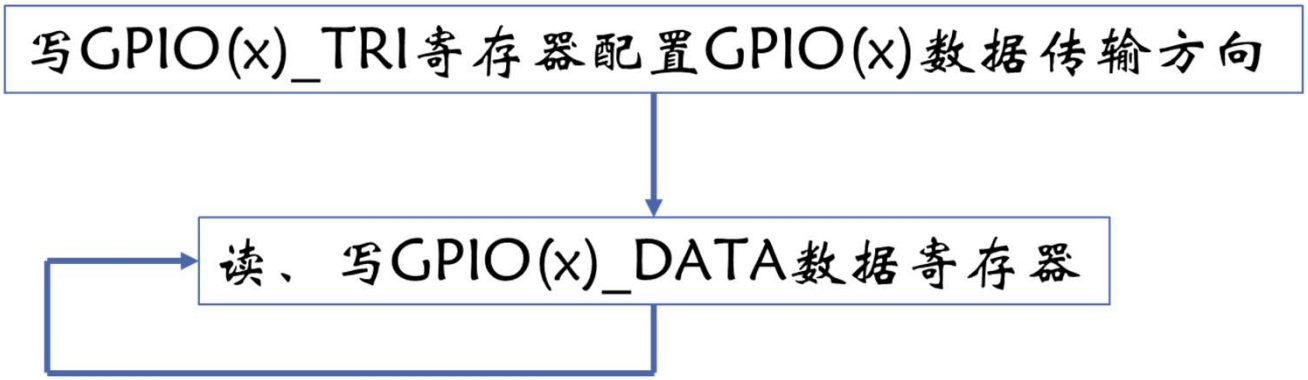
GPIO_1 GPIO2_TRI 0x0

GPIO存储空间映射

AXI_GPIO_0接口基地址为0x4000 0000，
AXI_GPIO_1接口基地址为0x4001 0000

GPIO	寄存器	地址	有效数据位	读写控制
AXI_GPIO_0	GPIO_DATA	0x4000 0000	$D_{15\sim0}$	读
	GPIO_TRI	0x4000 0004	$D_{15\sim0}$	写
	GPIO2_DATA	0x4000 0008	$D_{15\sim0}$	写
	GPIO2_TRI	0x4000 000c	$D_{15\sim0}$	写
AXI_GPIO_1	GPIO_DATA	0x4001 0000	$D_{7\sim0}$	写
	GPIO_TRI	0x4001 0004	$D_{7\sim0}$	写
	GPIO2_DATA	0x4001 0008	$D_{3\sim0}$	写
	GPIO2_TRI	0x4001 000c	$D_{3\sim0}$	写

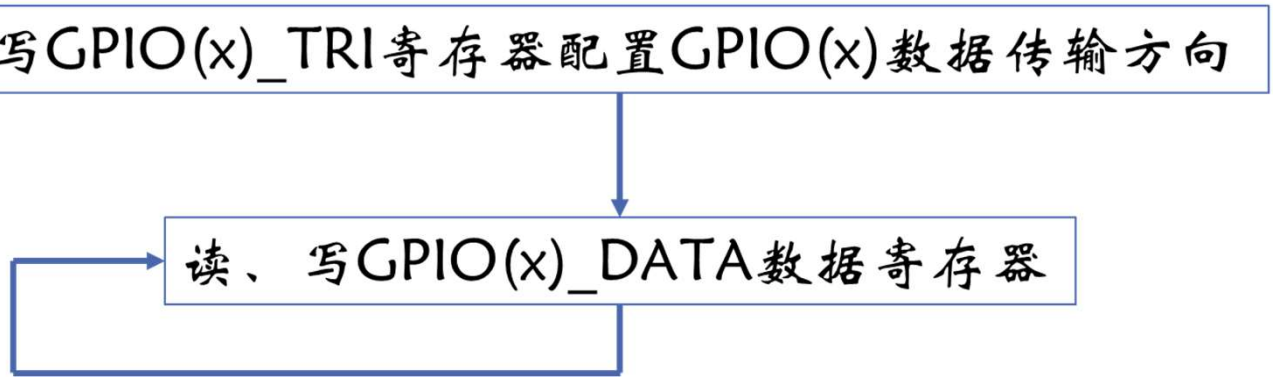
GPIO简单并行数字外设功能需求




```
unsigned short key;
Xil_Out16(0x40000004,0xffff);
key = Xil_In16(0x40000000)&0xffff;
```

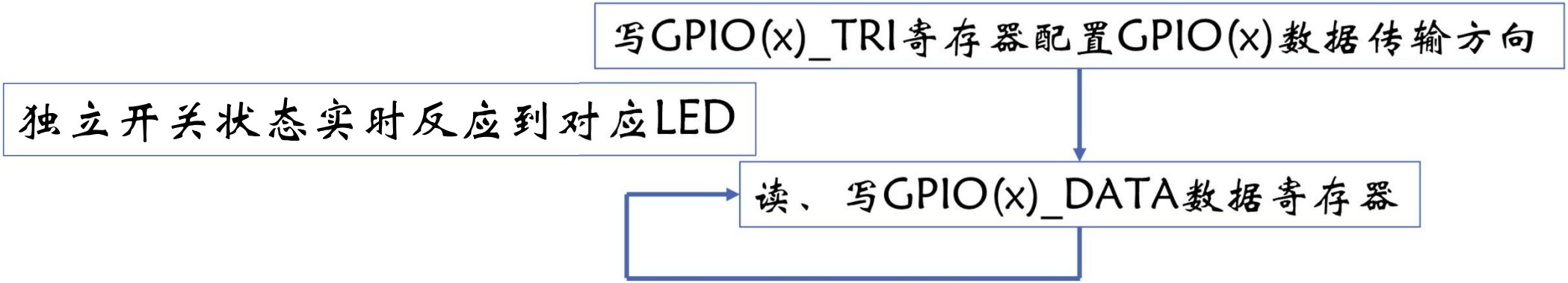
GPIO简单并行数字外设功能需求

16个LED间隔点亮




```
Xil_Out16(0x4000000c,0x0);
Xil_Out16(0x40000008,0x5555);
```

GPIO简单并行数字外设功能需求

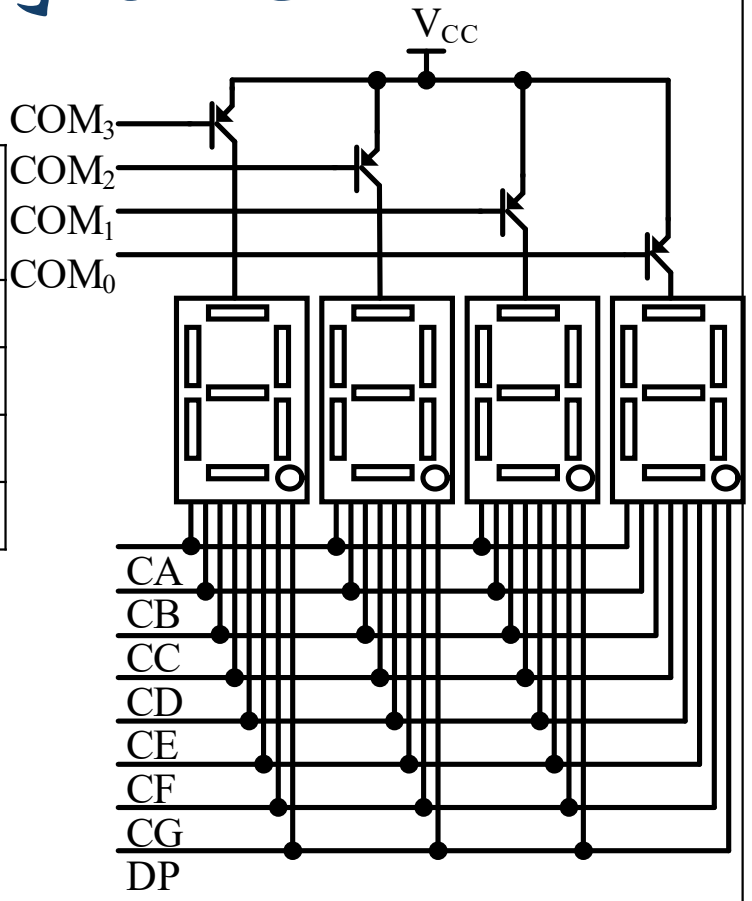



```
Xil_Out16(0x40000004,0xffff);//设置AXI_GPIO_0通道GPIO输入
Xil_Out16(0x4000000c,0x0);//设置AXI_GPIO_0通道GPIO2输出
while (1)
Xil_Out16(0x40000008, Xil_In16(0x40000000)&0xffff);
```

控制4位七段数码管显示数字0~3

数字	IO_7 (DP)	IO_6 (CG)	IO_5 (CF)	IO_4 (CE)	IO_3 (CD)	IO_2 (CC)	IO_1 (CB)	IO_0 (CA)	段码
0	1	1	0	0	0	0	0	0	0xc0
1	1	1	1	1	1	0	0	1	0xf9
2	1	0	1	0	0	1	0	0	0xa4
3	1	0	1	1	0	0	0	0	0xb0

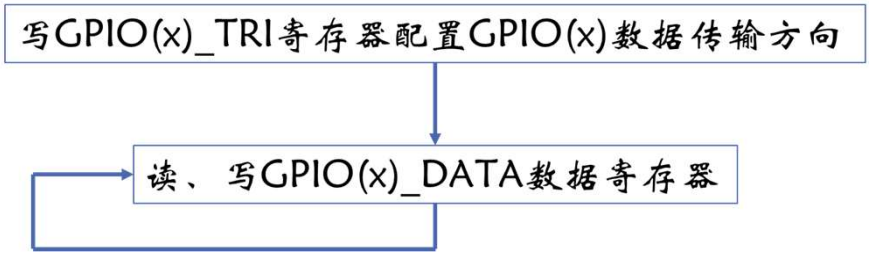
数码管控制信号	IO_3 (COM_3)	IO_2 (COM_2)	IO_1 (COM_1)	IO_0 (COM_0)	位码
COM_3	0	1	1	1	0x7
COM_2	1	0	1	1	0xb
COM_1	1	1	0	1	0xd
COM_0	1	1	1	0	0xe



GPIO简单并行数字外设功能需求

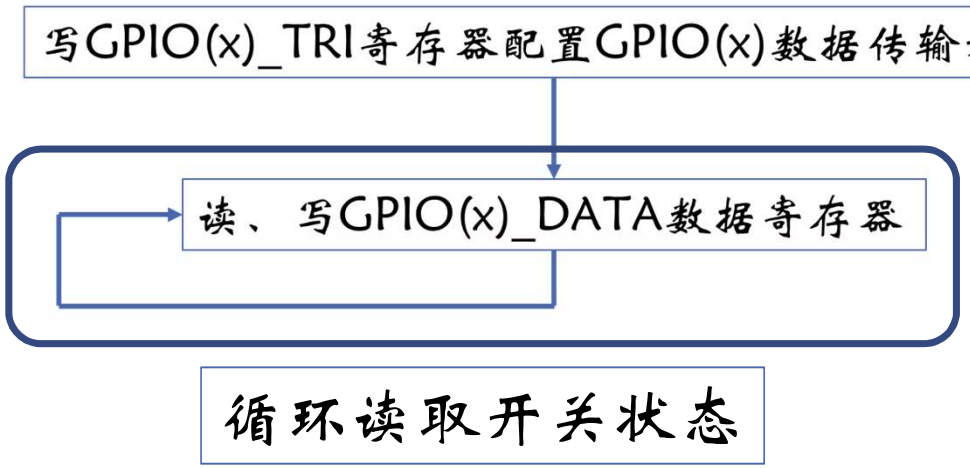
```
unsigned char segcode[8]={0xc0,0xf9,0xa4,0xb0};
unsigned char pos=0xf7;
Xil_Out8(0x40010004,0x0);
Xil_Out8(0x4001000c,0x0);
while(1){              //循环扫描
        for(i=0;i<4;i++) //4位扫描一遍
        {
            Xil_Out8(0x40010000,segcode[i]);//
            Xil_Out8(0x40010008,pos); //
            for(j=0;j<10000;j++); //
            pos=pos>>1;//
        }
        pos=0xf7;  //
    }
```

4位七段数码管显示数字0~3

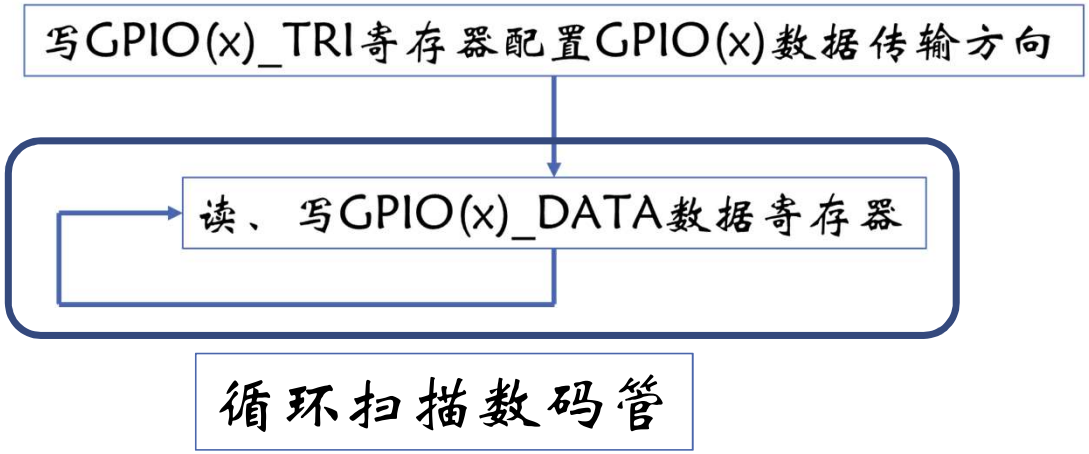


GPIO简单并行数字外设功能需求

同一控制程序将各个独立开关SW15~0状态实时反应到对应LED LD15~0上，同时将16位独立开关SW15~0表示的二进制数以十六进制形式显示在4位七段数码管上

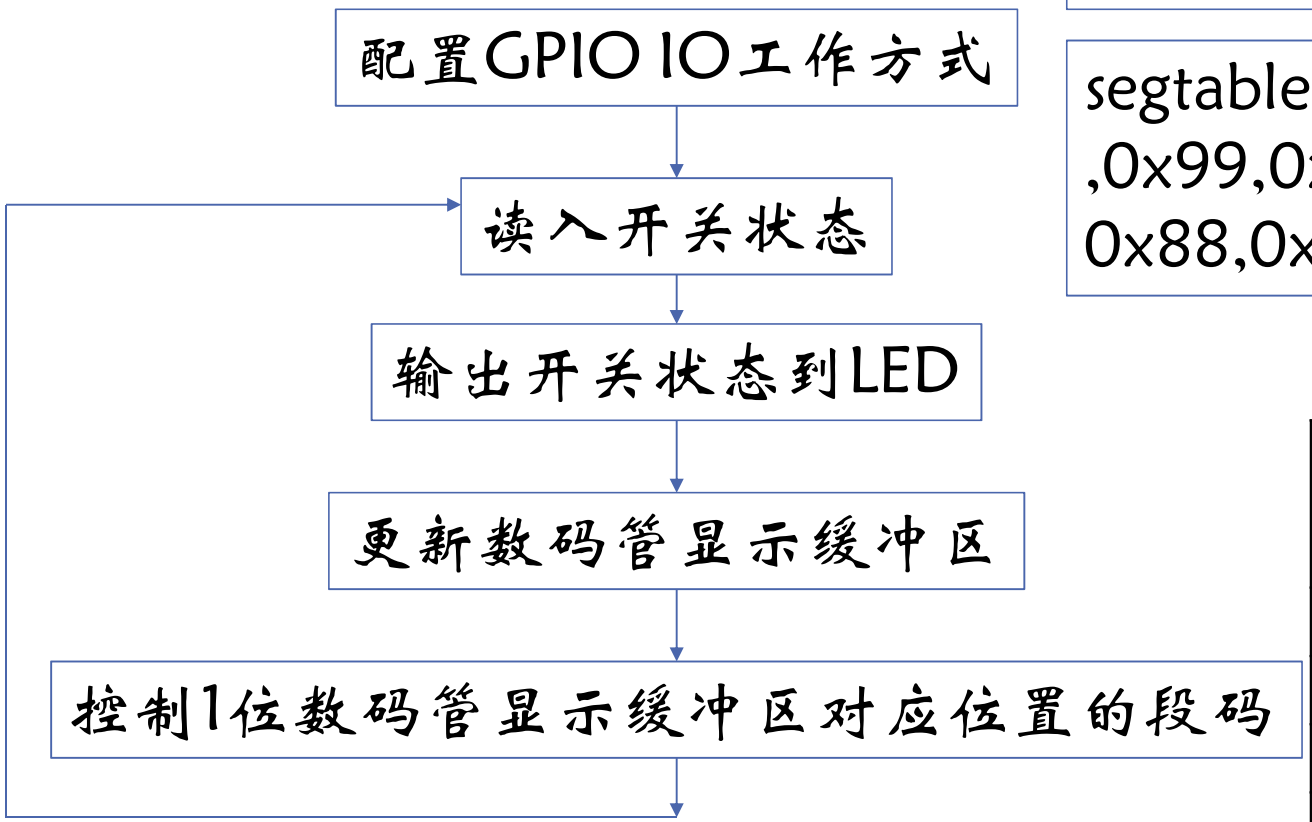


循环读取开关状态



循环扫描数码管

程序流程图(主)



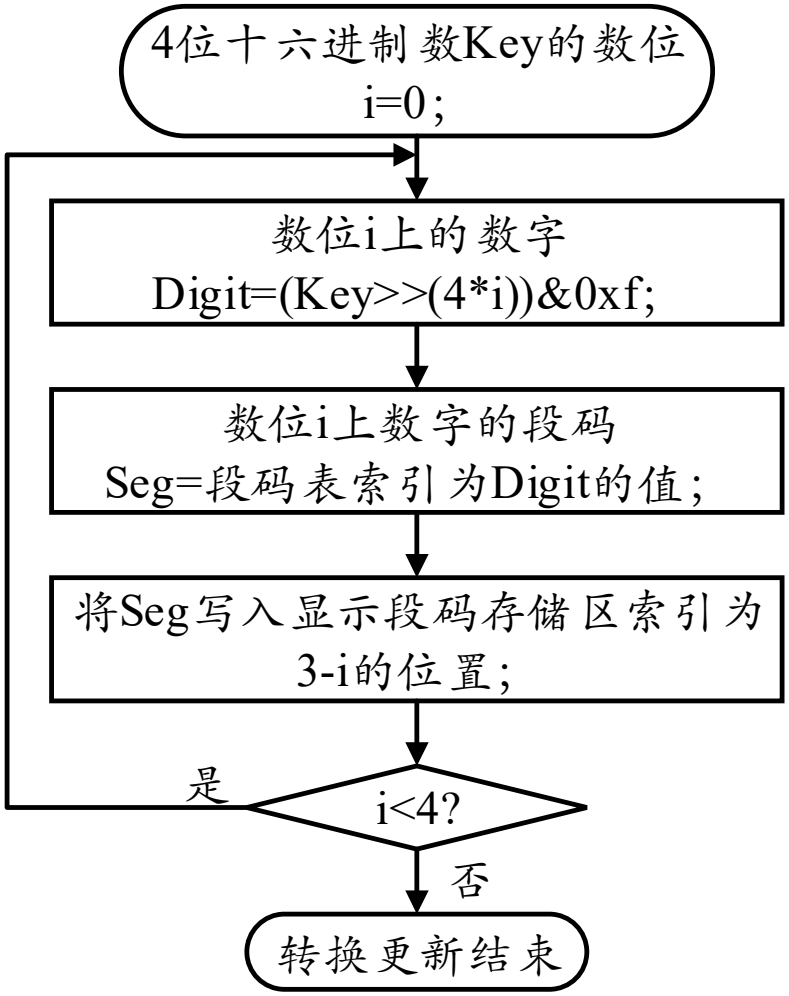
segcode[8]={0xc0,0xf9,0xa4,0xb0}

Sw=0x5678

segtable[16]={0xc0,0xf9,0xa4,0xb0
,0x99,0x92,0x82,0xf8,0x80,0x90,
0x88,0x83,0xc6,0xa1,0x86,0x8e,}

	(i)		(3-i)	4*i
数字	数位索引	段码	显示缓冲区索引	右移二进制数位
5	3	0x92	0	3*4
6	2	0x82	1	2*4
7	1	0xf8	2	1*4
8	0	0x80	3	0*4

程序流程图(子)-更新显示缓冲区



segcode[8]={0xc0,0xf9,0xa4,0xb0}

Sw=0x5678

segtable[16]={0xc0,0xf9,0xa4,0xb0
,0x99,0x92,0x82,0xf8,0x80,0x90,
0x88,0x83,0xc6,0xa1,0x86,0x8e,}

(i) (3-i) 4*i

数字	右移二进制数位	数位索引	显示缓冲区索引
5	3*4	3	0
6	2*4	2	1
7	1*4	1	2
8	0*4	0	3

数码管实时显示开关值的程序段

```
char segtable[16]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,\
        0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e,};//段码表
char segcode[4]={0xc0,0xc0,0xc0,0xc0};//显示缓冲区
short poscode[4]={0xf7,0xfb,0xfd,0xfe};//位码表
Xil_Out8(0x40010004,0x0);
Xil_Out8(0x4001000c,0x0);
Xil_Out16(0x40000004,0xffff);
Xil_Out16(0x4000000c,0x0);
```

```
while(1)
  {
     for(int i=0;i<4;i++)
      {
     short Key=Xil_In16(0x40000000);//
     Xil_Out16(0x40000008, Key);//
     for(int digit_index=0;digit_index<4;digit_index++)//
     segcode[3- digit_index]=segtable[(Key >>(4*digit_index))&0xf];
     Xil_Out8(0x40010000,segcode[i]);//
     Xil_Out8(0x40010008,poscode[i]); //
     for(int j=0;j<10000;j++);//延时控制
     }
     }
```

小结

- GPIO特征
  - 输入缓冲
  - 输出锁存
  - IO方向可编程配置，且各位独立控制
- GPIO编程
  - 写控制寄存器控制输入、输出方向
  - 读写数据寄存器实现输入、输出

下一讲：外设控制器EPC，自学，大字节序实例