计算机组成原理与接口技术——基于MIPS架构

Apr, 2022

第7讲 I/O接口

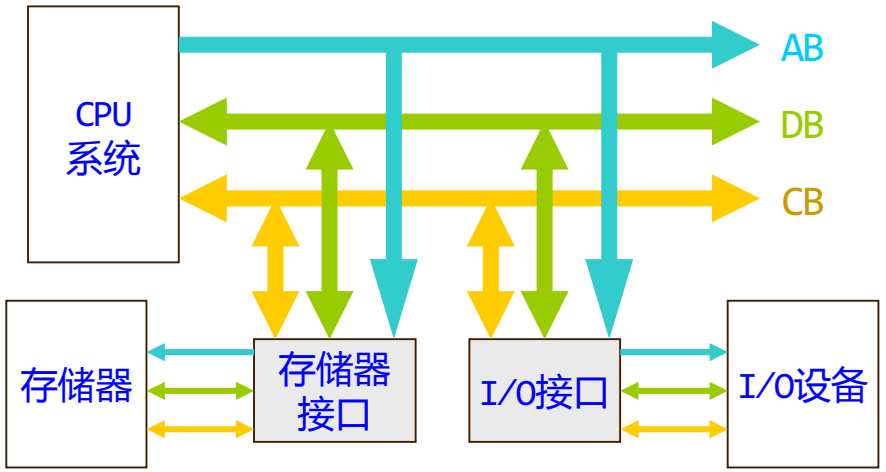
杨明
华中科技大学电信学院
myang@hust.edu.cn

Agenda

- 内容
  - 接口的编址方式
  - 并行IO接口设计
- 目的
  - 理解接口的不同编址方式的特点
  - 理解简单并行IO接口设计原理
  - 掌握独立开关、矩阵式键盘、LED、7段数码管以及并行AD转换器接口设计
  - 掌握基于GPIO控制器的接口设计
  - 了解外设控制器的接口设计

7.1 接口的基本概念

- 概念
  - 接口包含两方面的含义：为实现两个设备间数据交换的电子线路（硬件）及其通信协议（软件）
  - 接口并不局限在中央处理器与存储器或外设之间，也可在存储器与外设之间

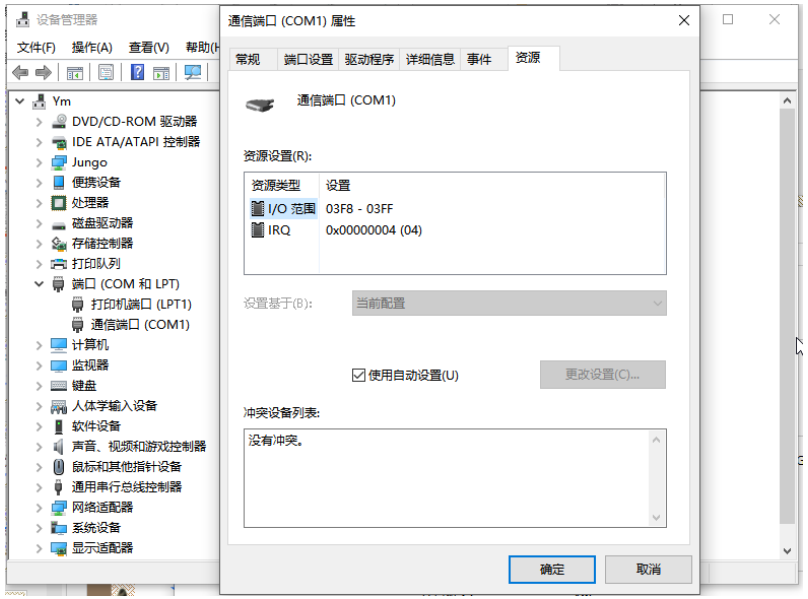


- 接口之间通信包括以下几个方面：
  - 命令译码
  - 数据交换
  - 状态反馈
  - 地址译码

7.2 I/O接口编址方式

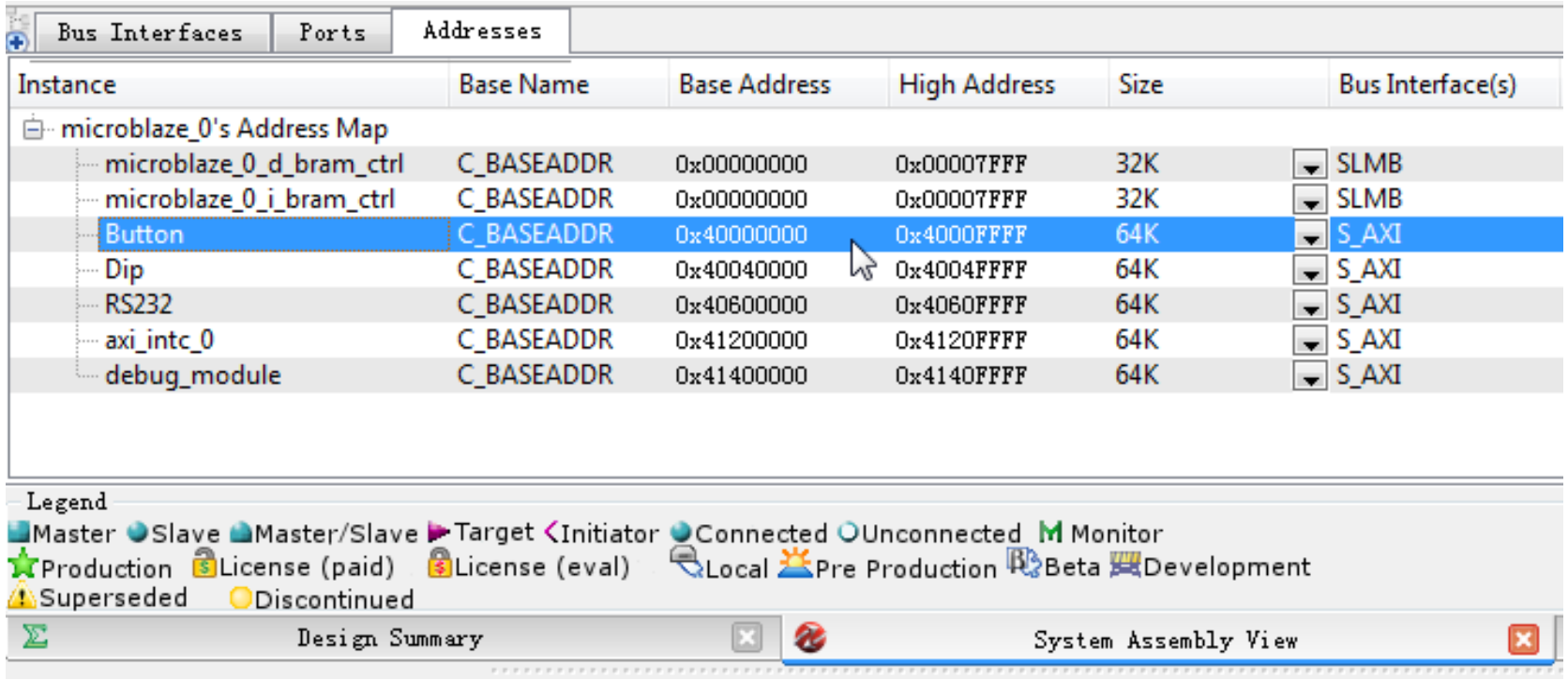
▶ I/O设备的端口地址

- 常见的I/O设备有键盘、鼠标、显示器、打印机等，不同的设备往往采用不同的接口和CPU交换信息：
  - 键盘——AT、PS/2、USB
  - 鼠标——PS/2、COM、USB
  - 打印机——并行口（LPT）
  - 显示器——显卡，显卡和CPU之间通过AGP/PCIe等连接
- CPU如何区分不同的I/O设备？
  - 如同不同的存储单元具有唯一的一个地址编号——存储单元地址一样，不同的I/O接口也具有不同地址编号——端口地址。



7.2 I/O接口编址方式

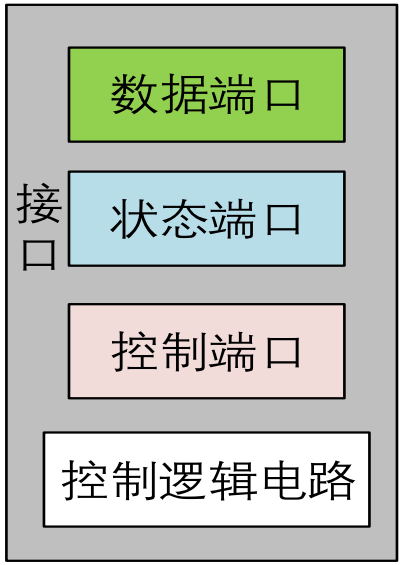
- I/O设备的端口地址
  - CPU如何区分不同的I/O设备？
    - 如同不同的存储单元具有唯一的一个地址编号——存储单元地址一样，不同的I/O接口也具有不同地址编号——端口地址。



7.2 I/O接口编址方式

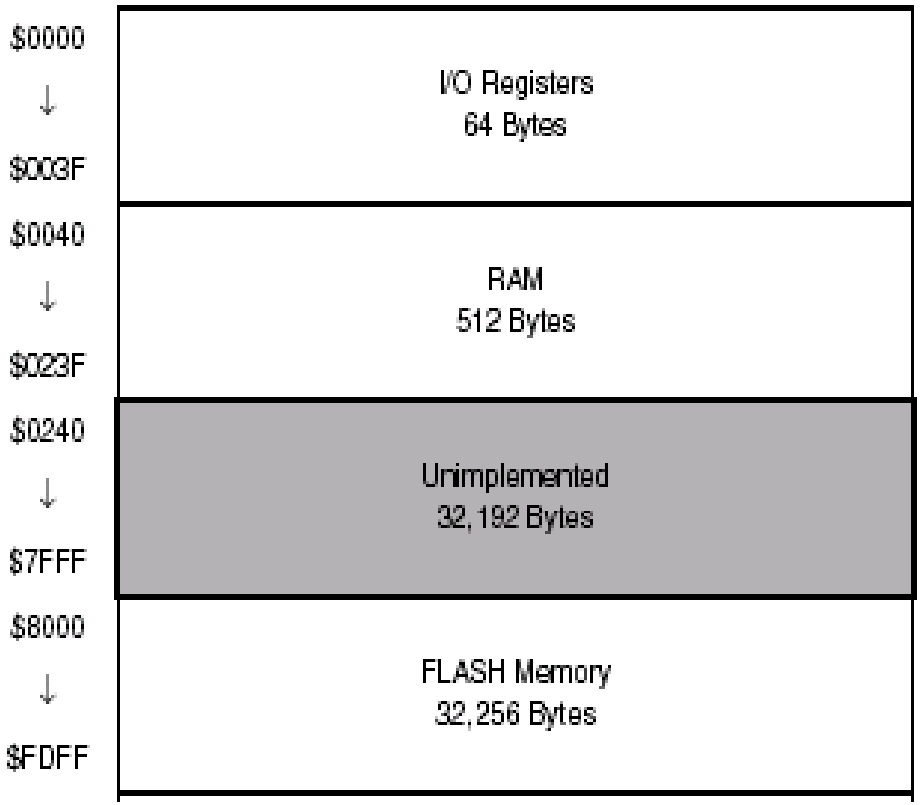
I/O设备的端口地址

- CPU如何区分不同的I/O设备？
  - 如同不同的存储单元具有唯一的一个地址编号——存储单元地址一样，不同的I/O接口也具有不同地址编号——端口地址。
- 同一I/O接口通常具有多个端口(命令口、数据口、状态口)：
  - COM1：3F8H ~ 3FFH
  - COM2：2F8H ~ 2FFH
- 接口基本结构中的寄存器叫做端口
  - 根据寄存器的不同功能，把这些寄存器分别叫做
    - 控制端口
    - 状态端口
    - 数据端口



7.2 I/O接口编址方式

- 编址方式
  - I/O端口与存储单元统一编址（存储器映像I/O寻址）
    - I /O端口与存贮器公用同一个地址空间。此时存贮器与 I /O的区别只是所用地址不同。
    - 嵌入式微处理器基本上都采用统一编址方式
      - 如：Freescale（飞思卡尔）的68HC08、9S08系列单片机、Intel-51单片机
    - 系统中没有单独的对I/O操作的控制信号IOR#、IOW#；
    - 系统对I /O端口的输入/输出操作也没有专用的输入/输出指令，对存贮器和I/O进行读/写操作都是用访问存贮器指令(如MOV)。



68HC08单片机统一编址

7.2 I/O接口编址方式

- 编址方式
  - I/O端口与存储单元独立编址（独立I/O寻址）
    - I /O端口和内存单元各自编址，I /O端口的地址空间与存贮器地址空间是独立的，分开的，即I /O口地址不占用存贮器地址空间；
      - I/O端口：0000H～0FFFFH
      - MEM：　0000H ～0FFFFH
    - CPU对I /O的输入/ 输出是用专门的输入/ 输出指令(如：IN/OUT)
    - 对I /O访问的控制信号是用专门的I /O控制信号(如：IOR#、IOW#)
    - PC机采用I/O端口独立编址方式

Agenda

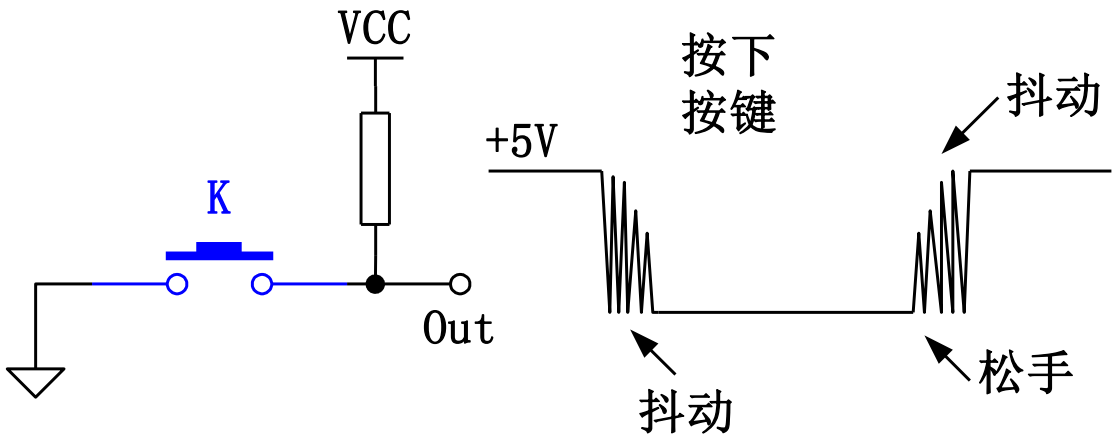
- 内容
  - 接口的编址方式
  - 并行IO接口设计
- 目的
  - 理解接口的不同编址方式的特点
  - 理解简单并行IO接口设计原理
  - 掌握独立开关、矩阵式键盘、LED、7段数码管以及并行AD转换器接口设计
  - 掌握基于GPIO控制器的接口设计
  - 了解外设控制器的接口设计

7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

▶ 独立开关输入接口

- 电路原理

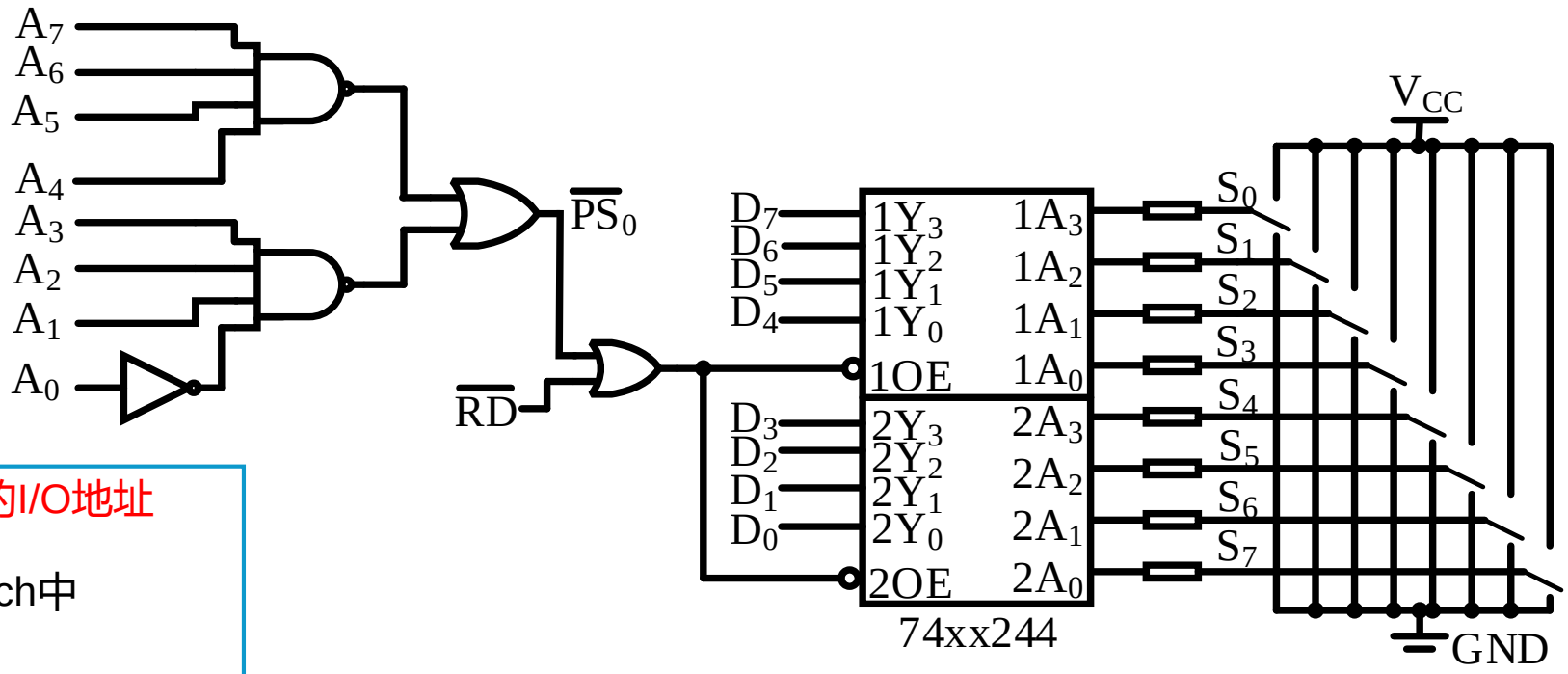


- 无论是按键或键盘都是利用机械触点的闭合、断开过程产生一个电压信号。但机械点的闭合、断开，均会产生抖动，抖动时间10ms-20ms。
- 了能够正确判断按键是否按下，需要采取消抖措施。本课假定按键理想，没有抖动。
- 消除按键抖动的方法
  - 硬件消抖
    - ▶RC吸收电路或RS触发器；
  - 软件消抖
    - ▶采用软件延时（>20ms）方法消除抖动

7.5 并行IO接口设计

▶ 独立开关输入接口

- 简单接口电路
  - Xilinx C代码



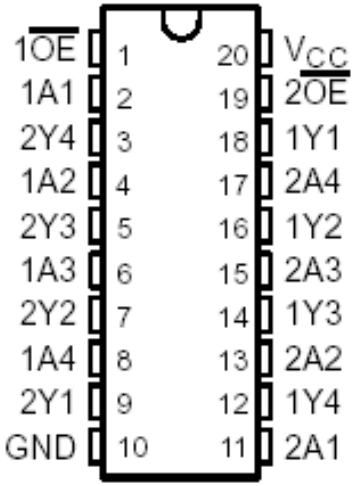

```
int port=0xfe;          // 开关的I/O地址

// 8位开关的状态保存在Switch中
char Switch=Xil_In8(port);
```

  - 【思考】若S0/S5按键分别按下，程序中的Switch值分别为多少？

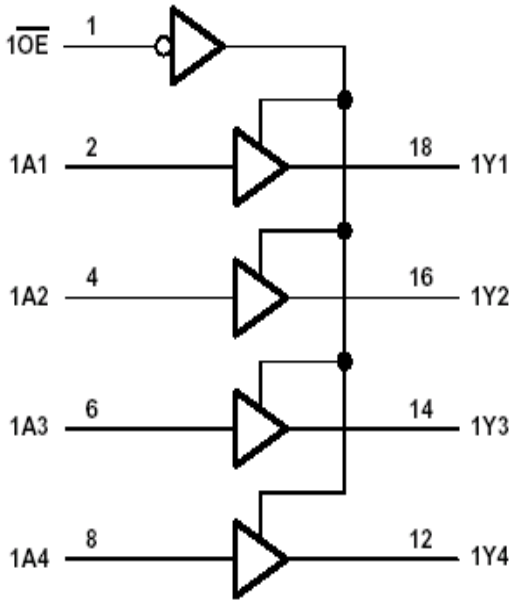
- 独立式按键电路配置灵活，软件简单，但是当按键较多时，需要更多的IO端口地址和IC芯片。
- 所以按键较多时一般采用行列式键盘——或称为矩阵式键盘。

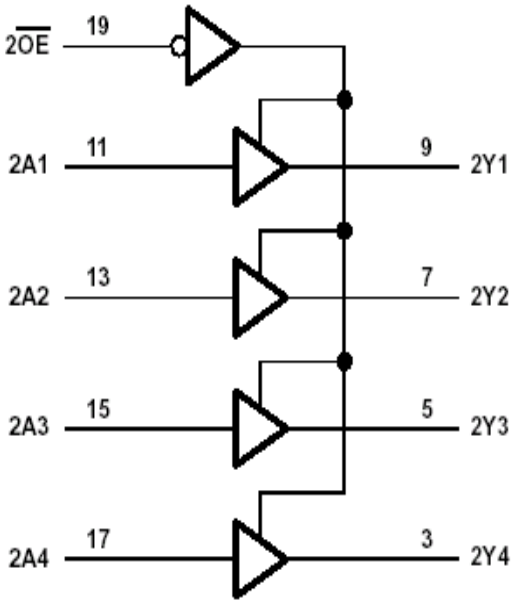
▶ 独立开关输入接口

- 244管脚和功能
  - 单向8位三态缓冲器：主要起隔离作



SN54AHCT244 . . . FK PACKAGE
(TOP VIEW)



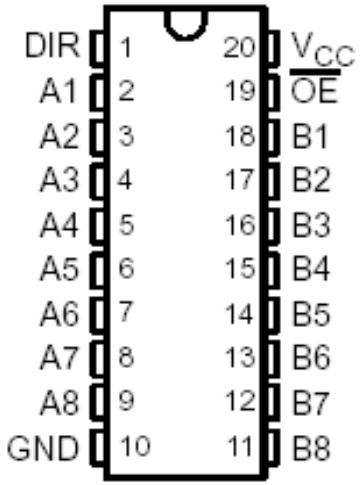


FUNCTION TABLE
(each buffer/driver)

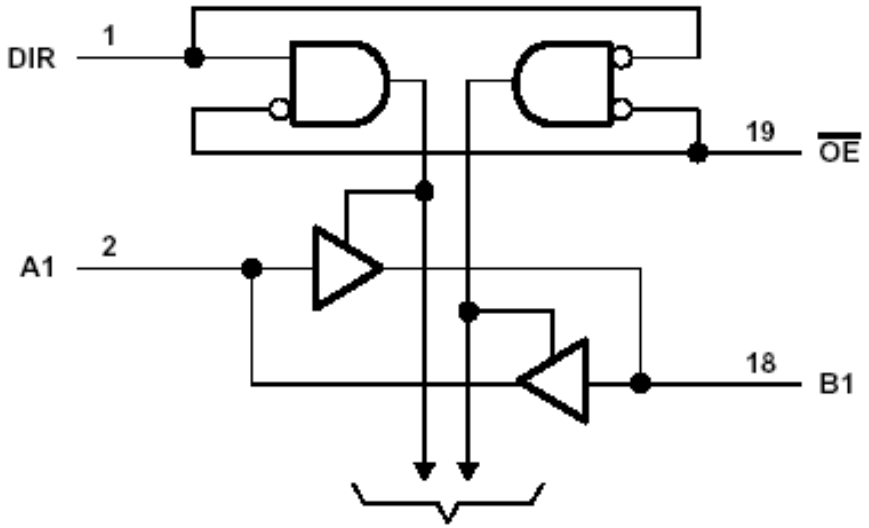
INPUTS		OUTPUT
$\overline{OE}$	A	Y
L	H	H
L	L	L
H	X	Z

▶ 独立开关输入接口

- 245管脚和功能
  - 双向8位三态缓冲驱动器



SN54AHCT245 . . . FK PACKAGE
(TOP VIEW)



To Seven Other Channels

FUNCTION TABLE

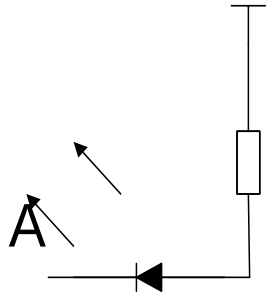
INPUTS		OPERATION
$\overline{OE}$	DIR	
L	L	B data to A bus
L	H	A data to B bus
H	X	Isolation

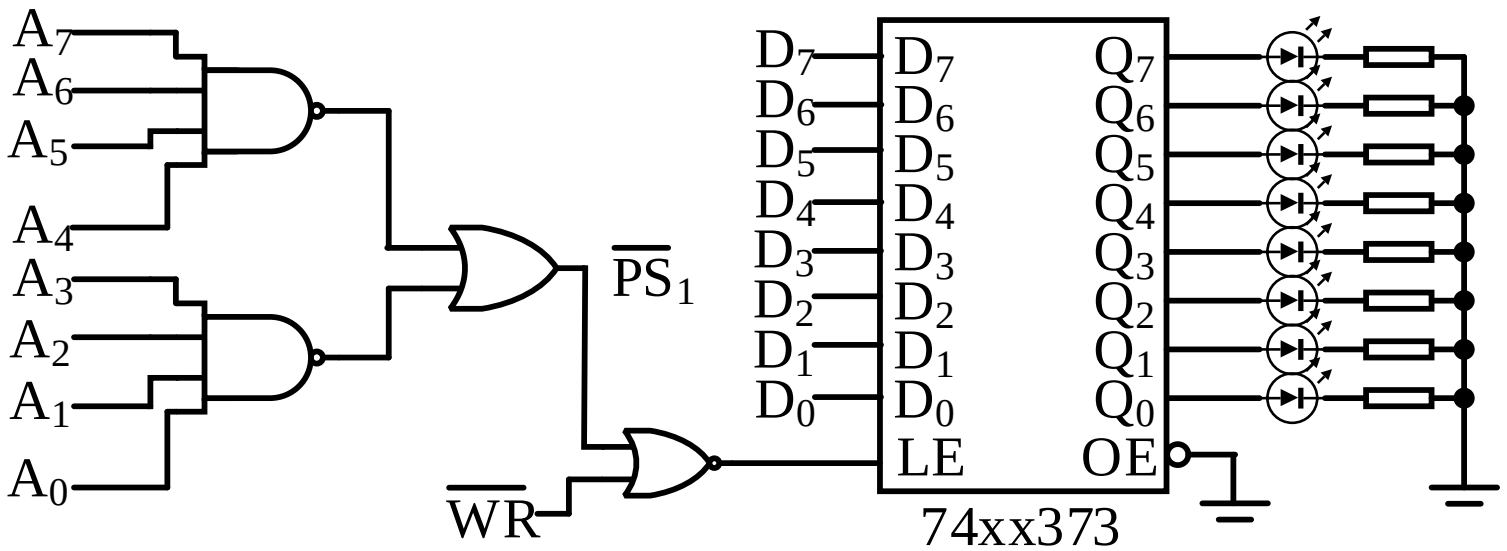
7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

▶ 发光二极管输出接口

- 电路原理
  - 必须具有数据锁存功能





- 【思考】要让右图中的8个LED呈现走马灯的效果，该如何编写代码？

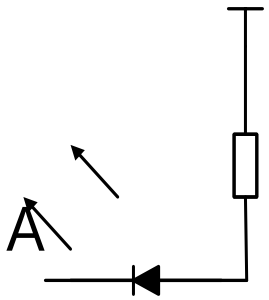
- Xilinx C代码

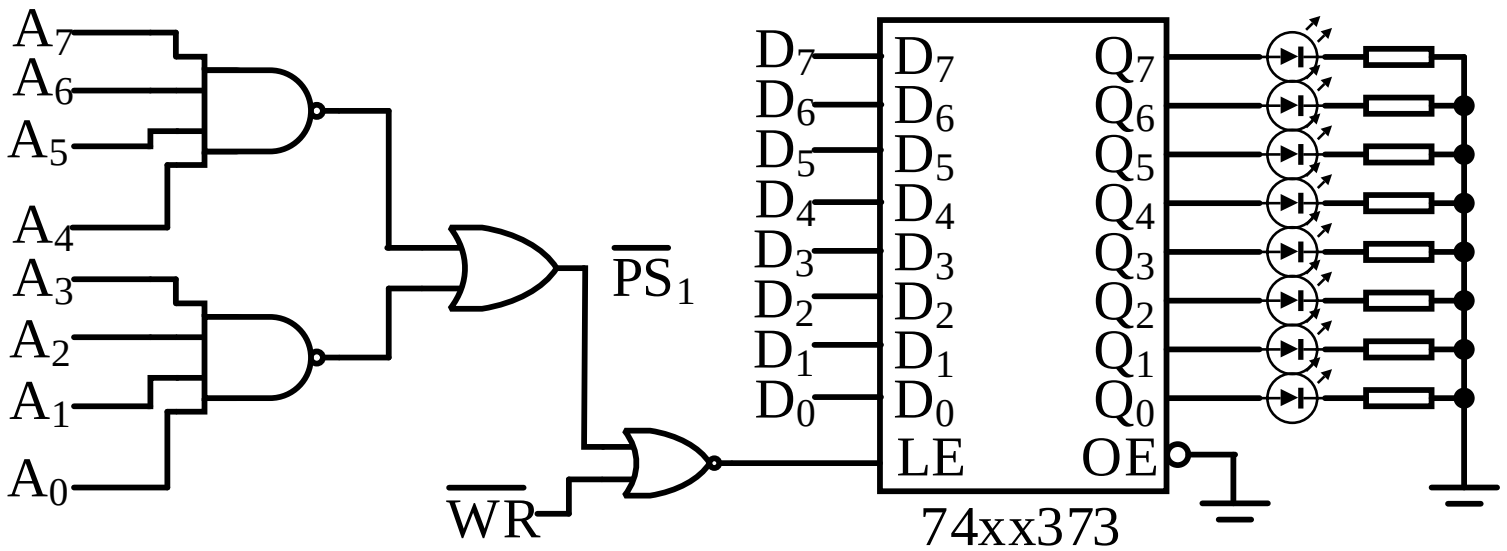
```
int    port = 0xff;
Xil_Out8 (port,0xFF);          // LED全亮
……
```

7.5 并行IO接口设计

▶ 发光二极管输出接口

- 电路原理
  - 必须具有数据锁存功能






```
while(1){
  char data=0x80;
  for( int i=0;i<8;i++) {
    Xil_Out8 (port,data);          // 点亮一个LED
    delay_500ms();
    data = data>>1;
  }
};
```

- 软件延时

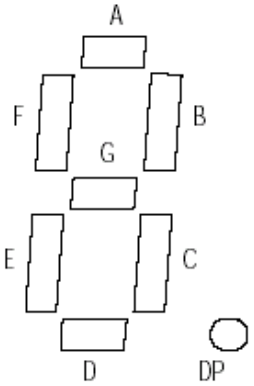
```
void delay_500ms()
{
    for(int i=0;i<500*0x10000;i++);
}
```

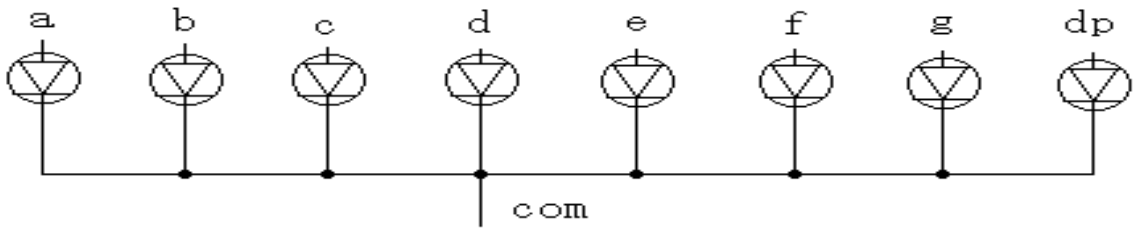
7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

▶ 七段数码管动态显示接口

- 七段数码管是计算机系统中常用的输出装置，它由7个条形发光二极管和一个圆点发光二极管组成。由各管的亮暗组合成数字或符号所有发光二极管有一端连接在一起，构成了共阴极或共阳极显示器，以共阴极为例：





- 字型码
  - 字形编码因实际连接的不同而异

D7	D6	D5	D4	D3	D2	D1	D0
DP	A	B	C	D	E	F	G
0	1	0	1	1	0	1	1
1	1	0	1	1	0	1	1

则显示“5”的字形码为：5BH

“5.”：DBH

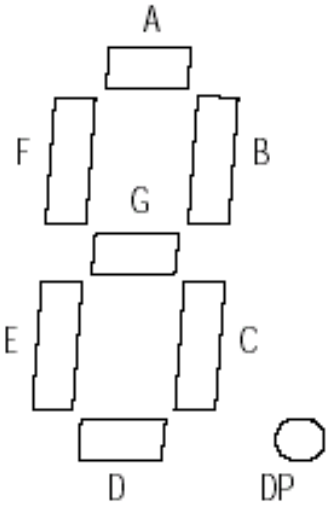
▶ 七段数码管动态显示接口

- 字型码
  - 字形编码因实际连接的不同而异

D7	D6	D5	D4	D3	D2	D1	D0
DP	A	B	C	D	E	F	G
0	1	0	1	1	0	1	1
1	1	0	1	1	0	1	1

则显示“5”的字形码为：5BH

“5.”：DBH



D7	D6	D5	D4	D3	D2	D1	D0
DP	G	F	E	D	C	B	A
0	1	1	0	1	1	0	1
1	1	1	0	1	1	0	1

则显示“5”的字形码为：6DH

“5.”：EDH

思考：“0”按上表连接的字形码为：7EH

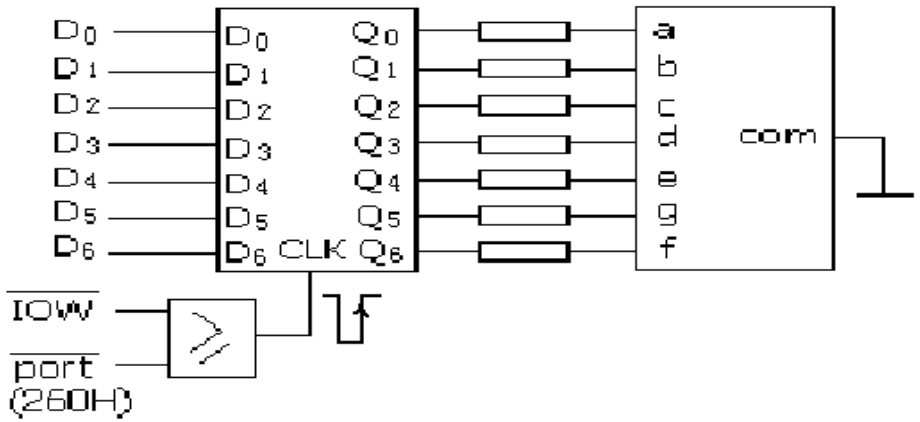
按下表连接的字形码为：3FH

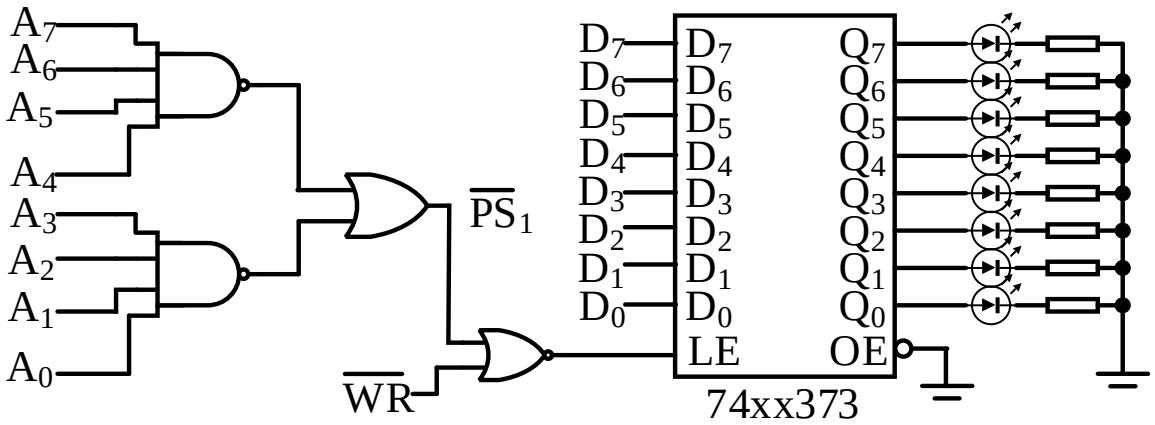
“1”按上表连接的字形码为：30H

按下表连接的字形码为：06H

▶ 七段数码管动态显示接口

- 静态显示





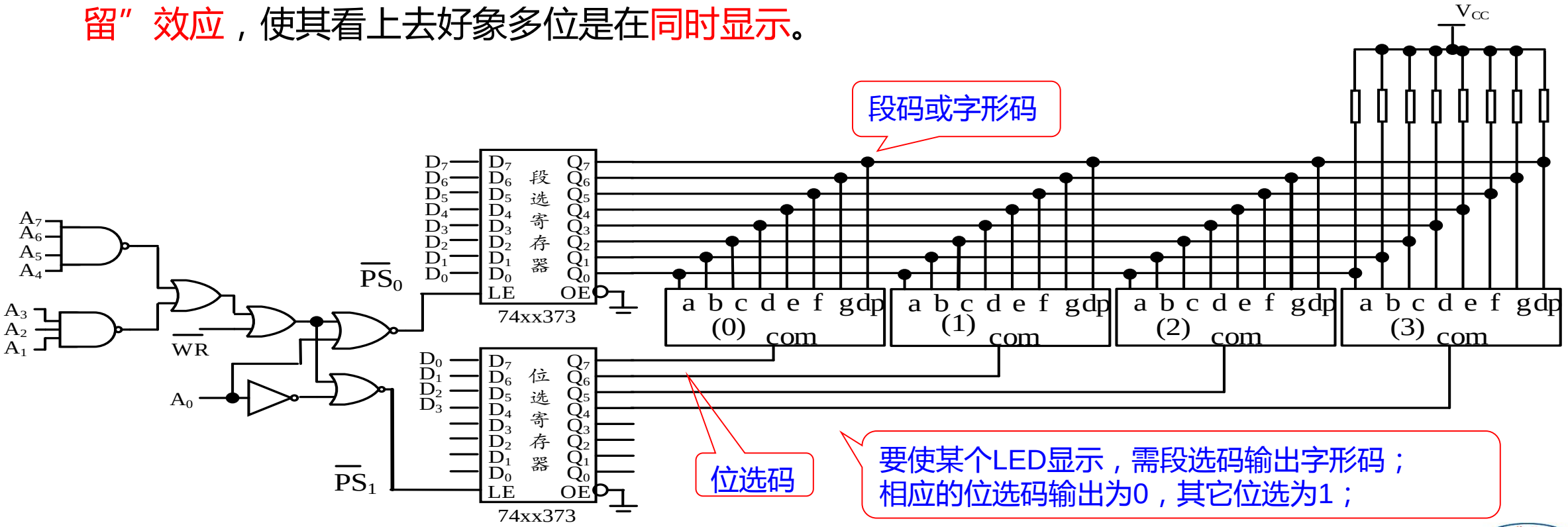

```
int    port = 0x0260;
Xil_Out8 (port, 字形码 );        // 显示一个字形
……
```

```
int    port = 0xff;
Xil_Out8 (port,0x80);            // LED7亮
……
```

- 静动态显示的程序设计简单，但是在进行多位显示时，需要的I/O端口和IC器件都很多，如要显示8个字符，则需要8个I/O端口地址和8个373。为节省硬件资源，可采用动态显示方式。

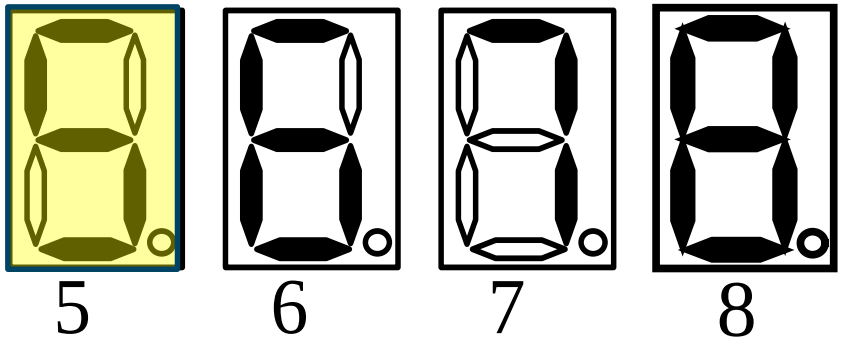
▶ 七段数码管动态显示接口

- 动态显示
  - 将所有数码管的同名段并联，由一个输出I/O控制，称为段选寄存器；而所有共阴极点由另一个I/O口控制，称为位选寄存器；——只需两个8位I/O口
  - 每一个瞬间，只有一个数码管显示。适当选择循环速度逐个循环亮点数码管；利用人眼“视觉暂留”效应，使其看上去好象多位是在同时显示。

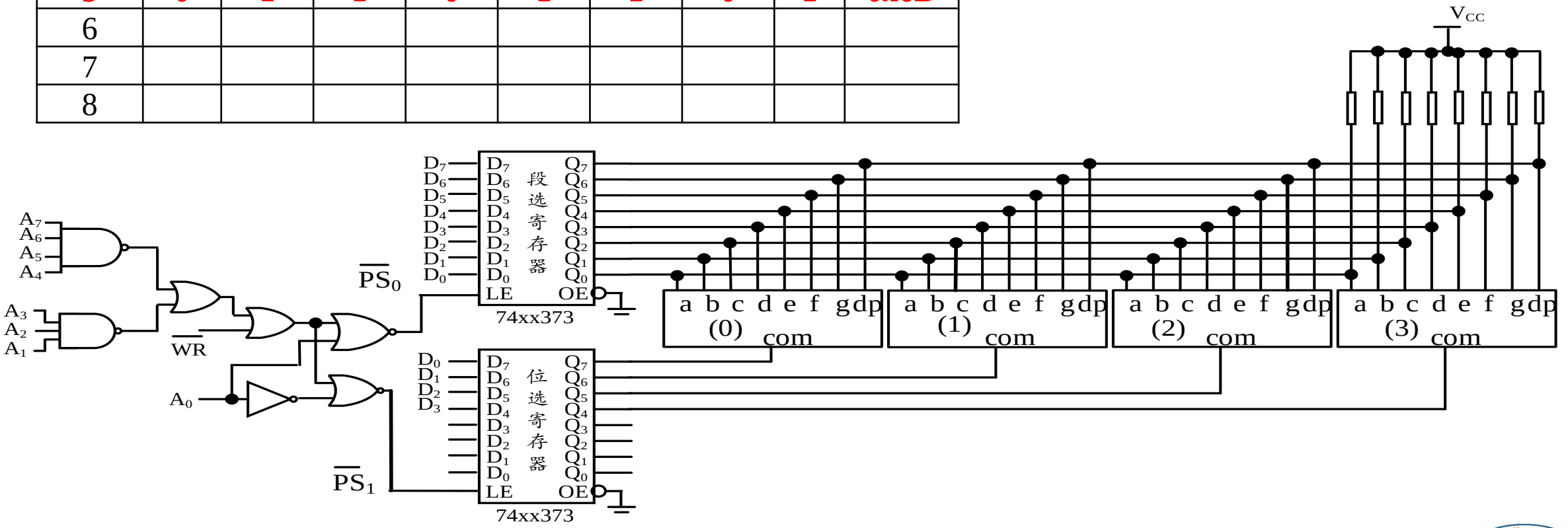


▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样

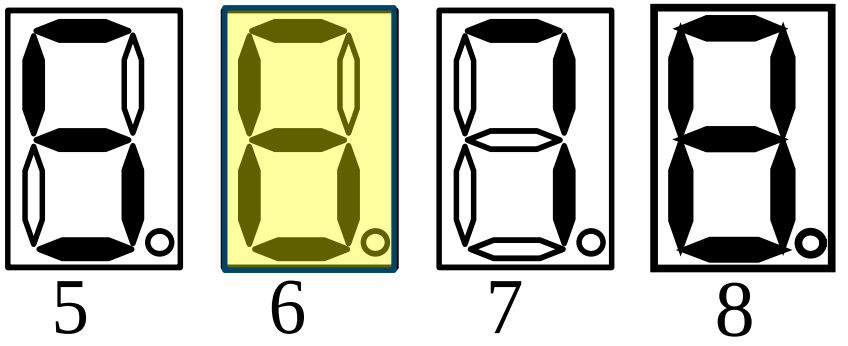


数据线	D7	D6	D5	D4	D3	D2	D1	D0	段码
数字	dp	g	f	e	d	c	b	a	
5	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0x6D**
6									
7									
8									

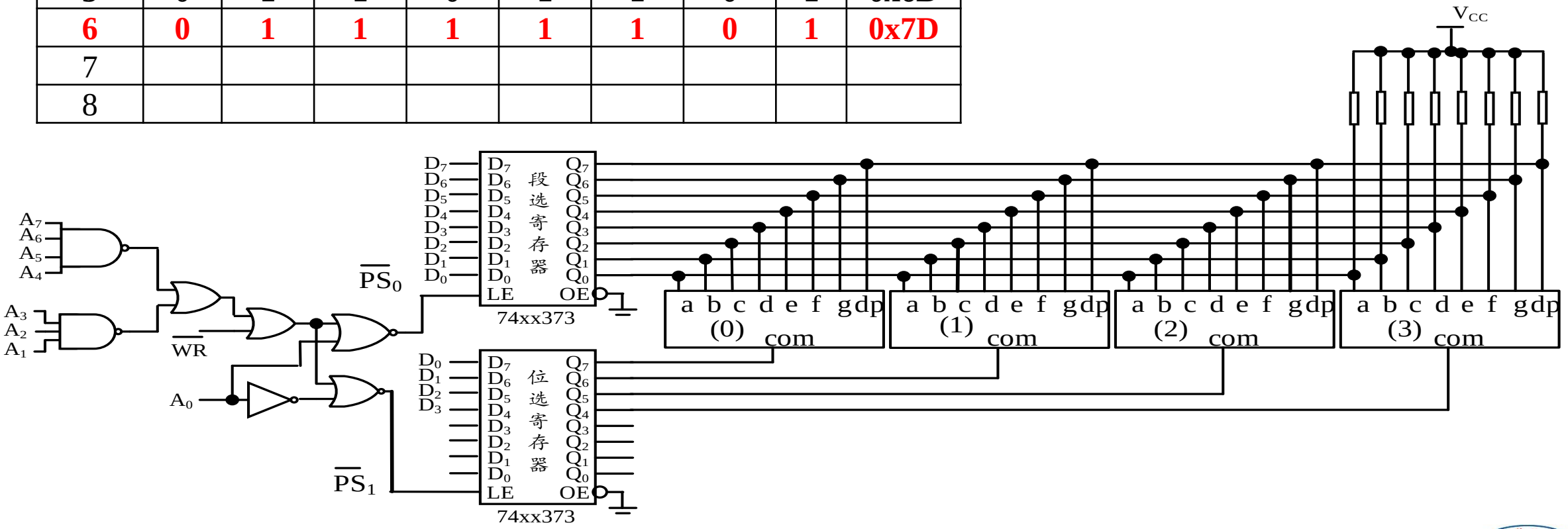


▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样

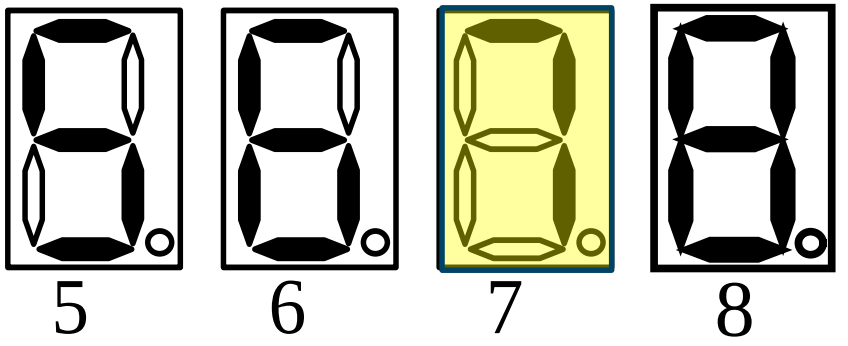


数据线	D7	D6	D5	D4	D3	D2	D1	D0	段码
数字	dp	g	f	e	d	c	b	a	
5	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0x6D**
6	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0x7D**
7									
8									

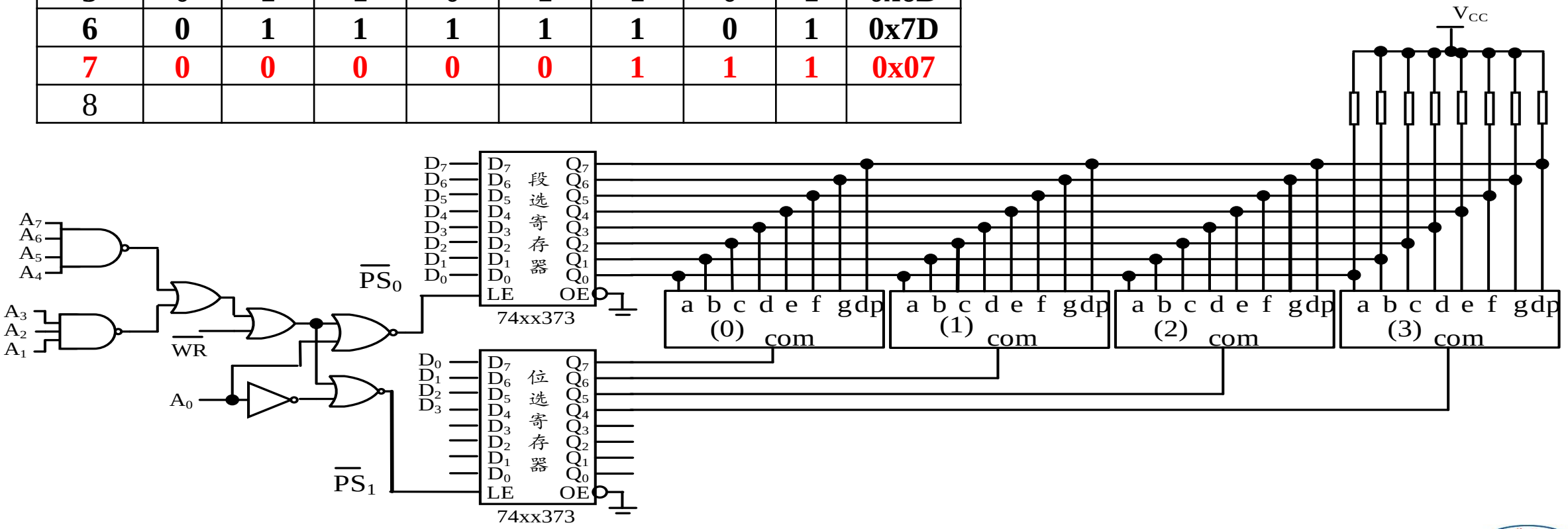


▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样

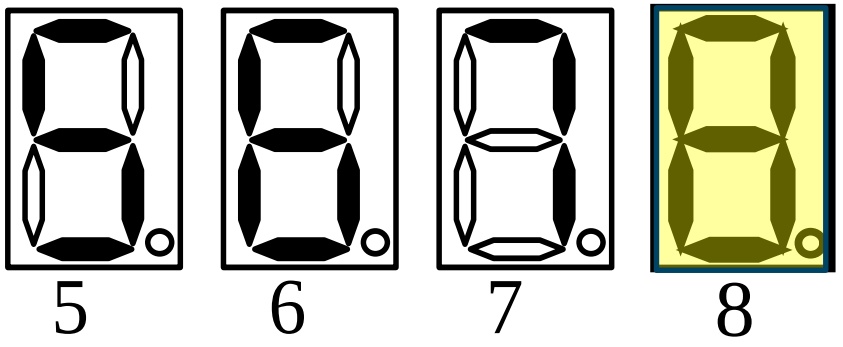


数据线	D7	D6	D5	D4	D3	D2	D1	D0	段码
数字	dp	g	f	e	d	c	b	a	
5	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0x6D**
6	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0x7D**
7	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0x07**
8									

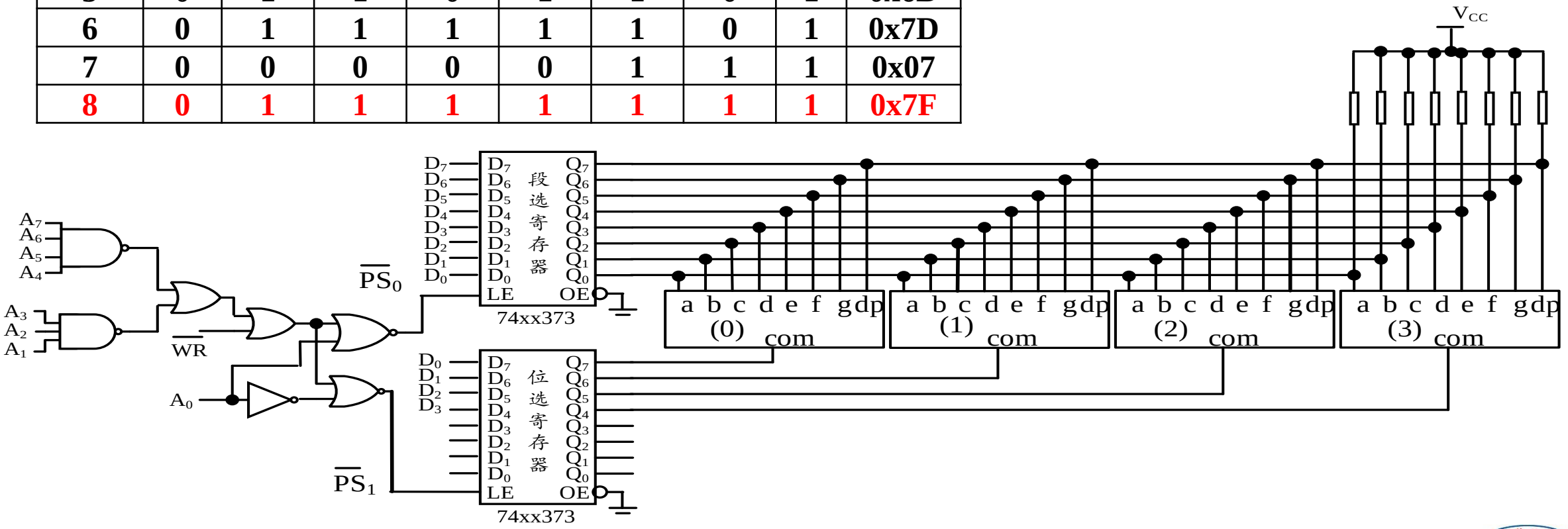


▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样



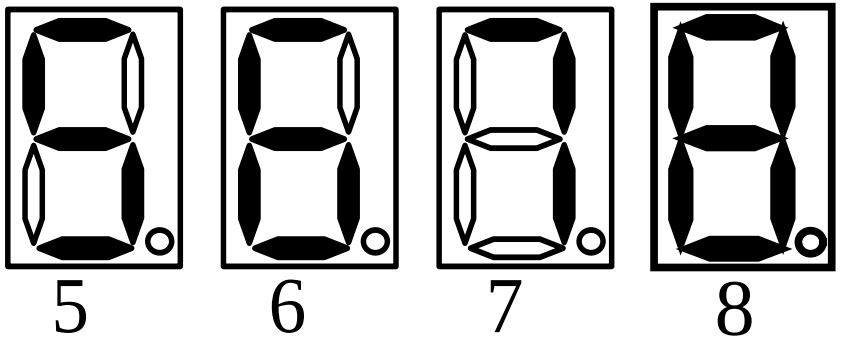
数据线	D7	D6	D5	D4	D3	D2	D1	D0	段码
数字	dp	g	f	e	d	c	b	a	
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F

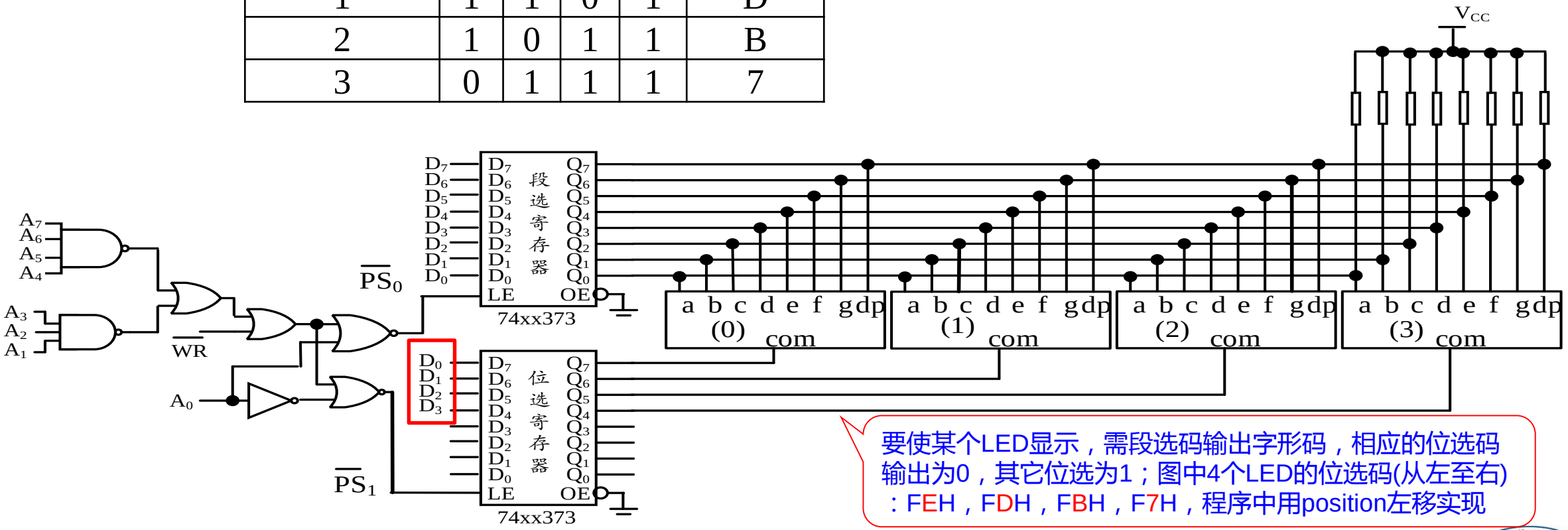


▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样
  - 位选码

数码管位置	D_3	D_2	D_1	D_0	位码
0	1	1	1	0	E
1	1	1	0	1	D
2	1	0	1	1	B
3	0	1	1	1	7





要使某个LED显示，需段选码输出字形码，相应的位选码输出为0，其它位选为1；图中4个LED的位选码(从左至右)：FEH，FDH，FBH，F7H，程序中用position左移实现

7.5 并行IO接口设计

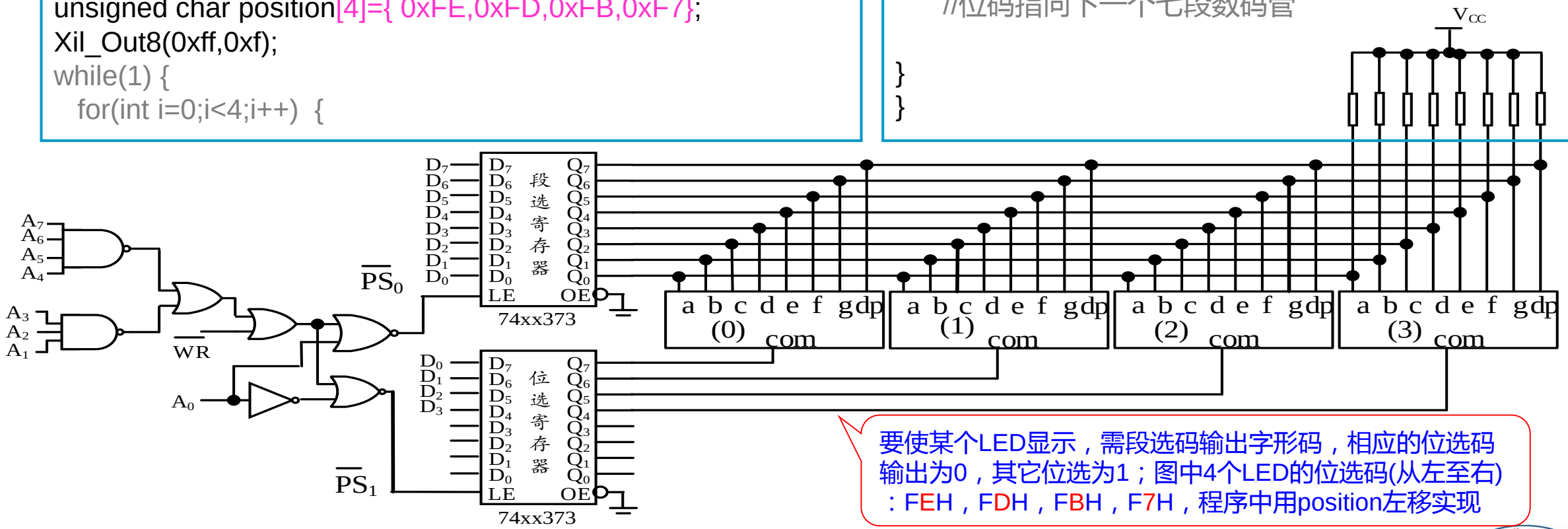
▶ 七段数码管动态显示接口

- 动态显示：在4个数码管上显示“5678”字样

PS0表示端口地址0xfe，而PS1表示端口地址0xff，Xilinx C语言程序段为如下所示：

```
unsigned char seg_code[4]={ 0x6d,0x7d,0x07,0x7f};
unsigned char position[4]={ 0xFE,0xFD,0xFB,0xF7};
Xil_Out8(0xff,0xf);
while(1) {
   for(int i=0;i<4;i++)  {
```

```
while(1) {
    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[i]);      //输出第i位的位码
    delay_4ms();                          //延时
     //位码指向下一个七段数码管

}
}
```



要使某个LED显示，需段选码输出字形码，相应的位选码输出为0，其它位选为1；图中4个LED的位选码(从左至右)：FEH，FDH，FBH，F7H，程序中用position左移实现

▶ 七段数码管动态显示接口

- 软件延时

```
for(int i=0;i<0x10000;i++);
```

  - 延时时间为：

$$d = (N_a + N_c + N_j) * T * M$$

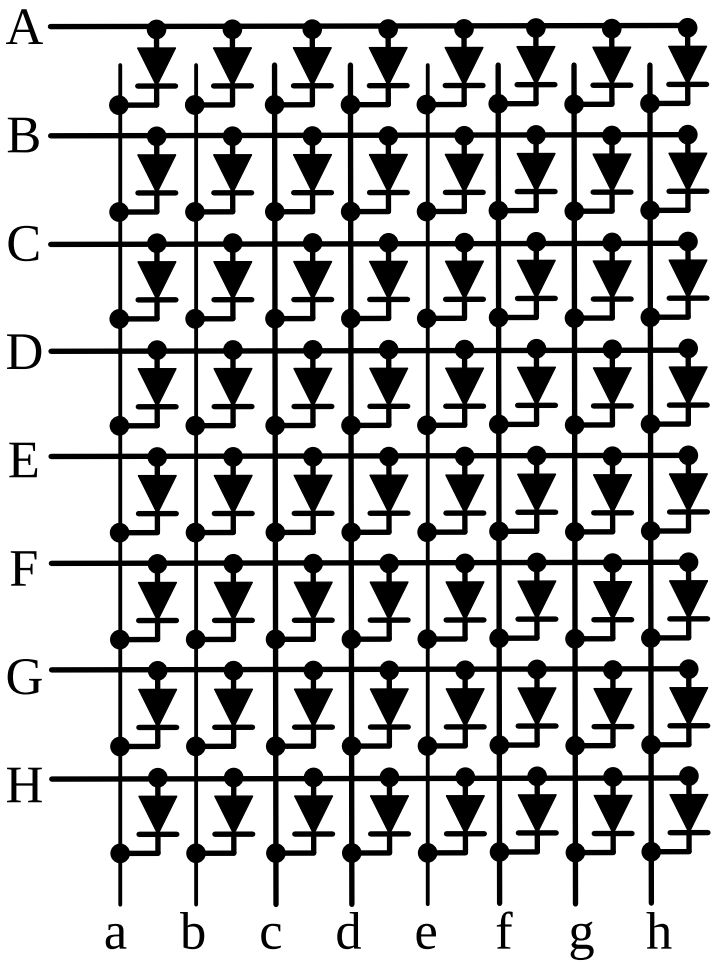
    - 执行加法指令所需的时钟周期个数为Na，
    - 执行比较指令所需的时钟周期个数为Nc，
    - 执行条件跳转指令所需的时钟周期个数为Nj，
    - 微处理器时钟周期为T，
    - 循环次数为M

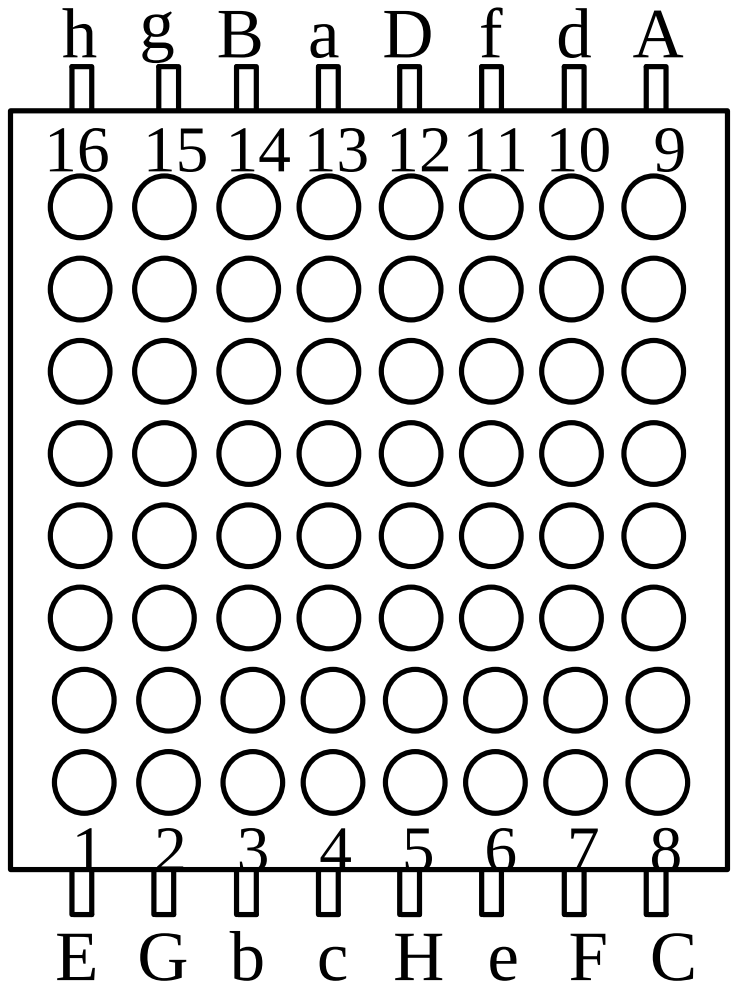
7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

7.5 并行IO接口设计

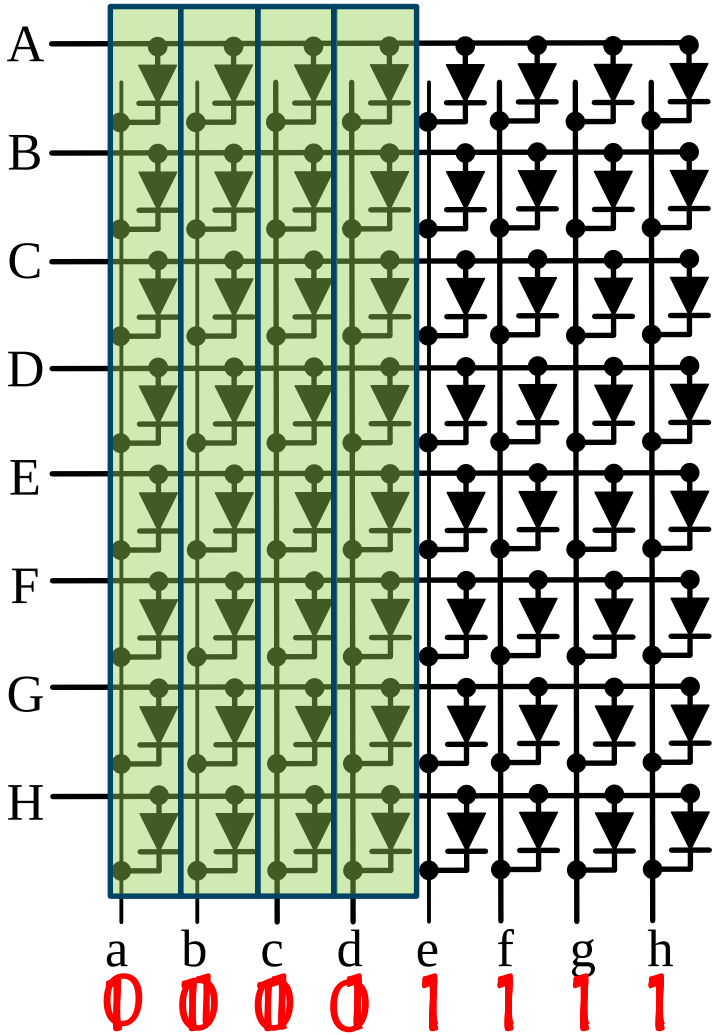
- LED点阵
  - 8X8点阵

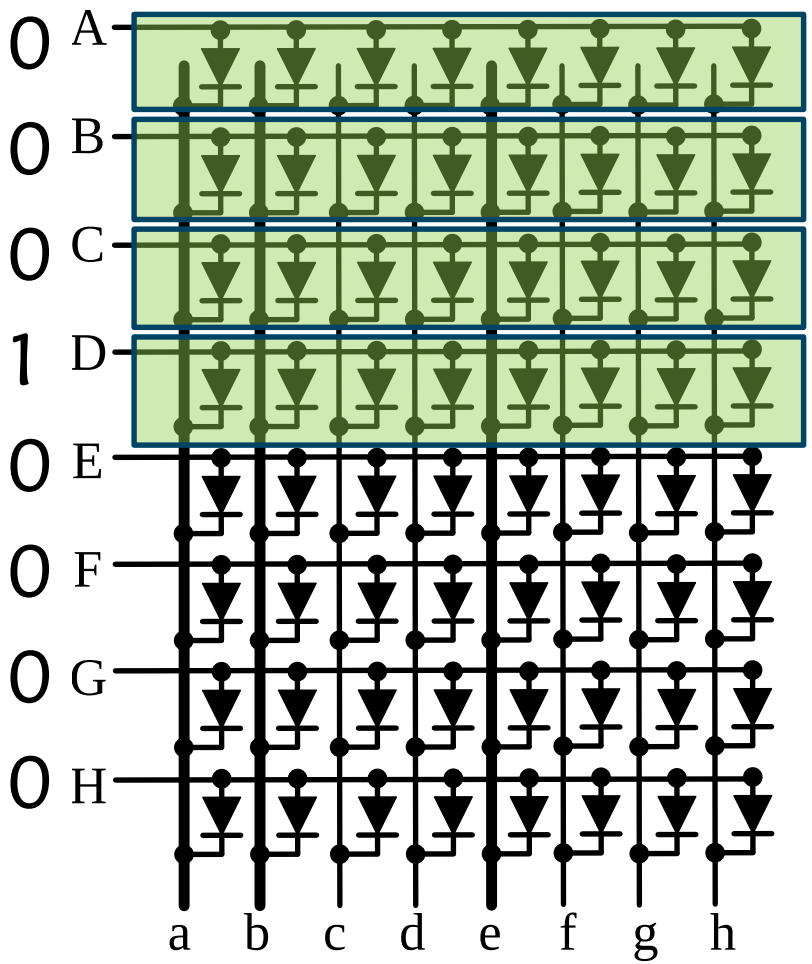




7.5 并行IO接口设计

- LED点阵
  - 人眼视觉暂留效应
    - 逐列扫描
      - 共阴
    - 逐行扫描
      - 共阳



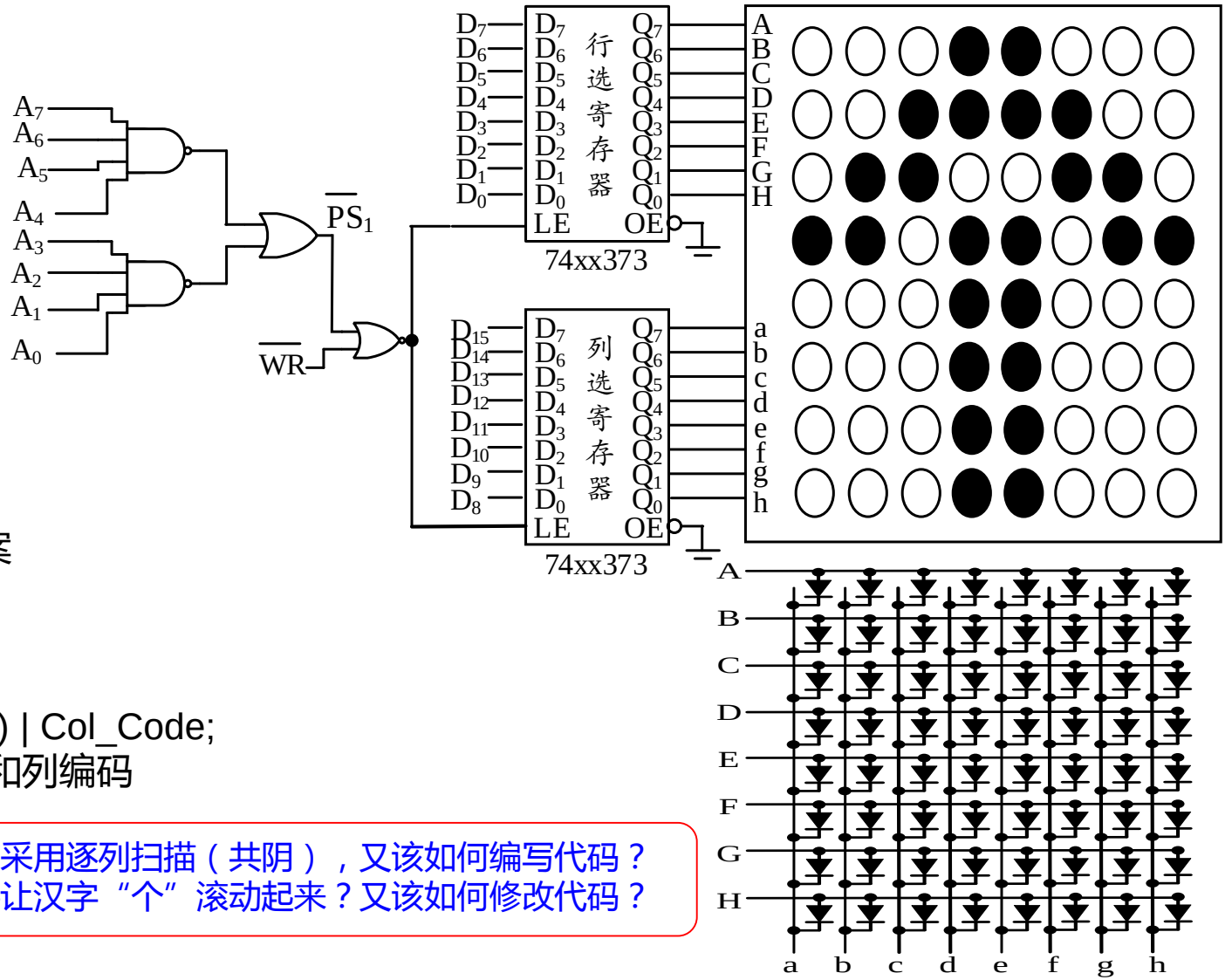


7.5 并行IO接口设计

- LED点阵
  - 8X8点阵
    - 逐行扫描（共阳）程序：

PS1表示端口地址0xff，Xilinx C语言：

```
unsigned char Row_Code[8]=
{0xe7,0xc3,0x99,0x24,0xe7,0xe7,0xe7,0xe7};
unsigned char Col_Code=0x80;
unsigned short code;

while(1) //无限循环控制8×8 LED点阵显示图案
{  Col_Code=0x80;
    for(int i=0;i<8;i++)
    {
      code=((unsigned short) Row_Code [i]<<8) | Col_Code;
      Xil_Out16(0xff,code);   //输出第i行的行选和列编码
      delay_2ms(); //延时
      Col_Code = Col_Code >> 1;
    }
}
```



【思考1】若采用逐列扫描（共阴），又该如何编写代码？
【思考2】要让汉字“个”滚动起来？又该如何修改代码？

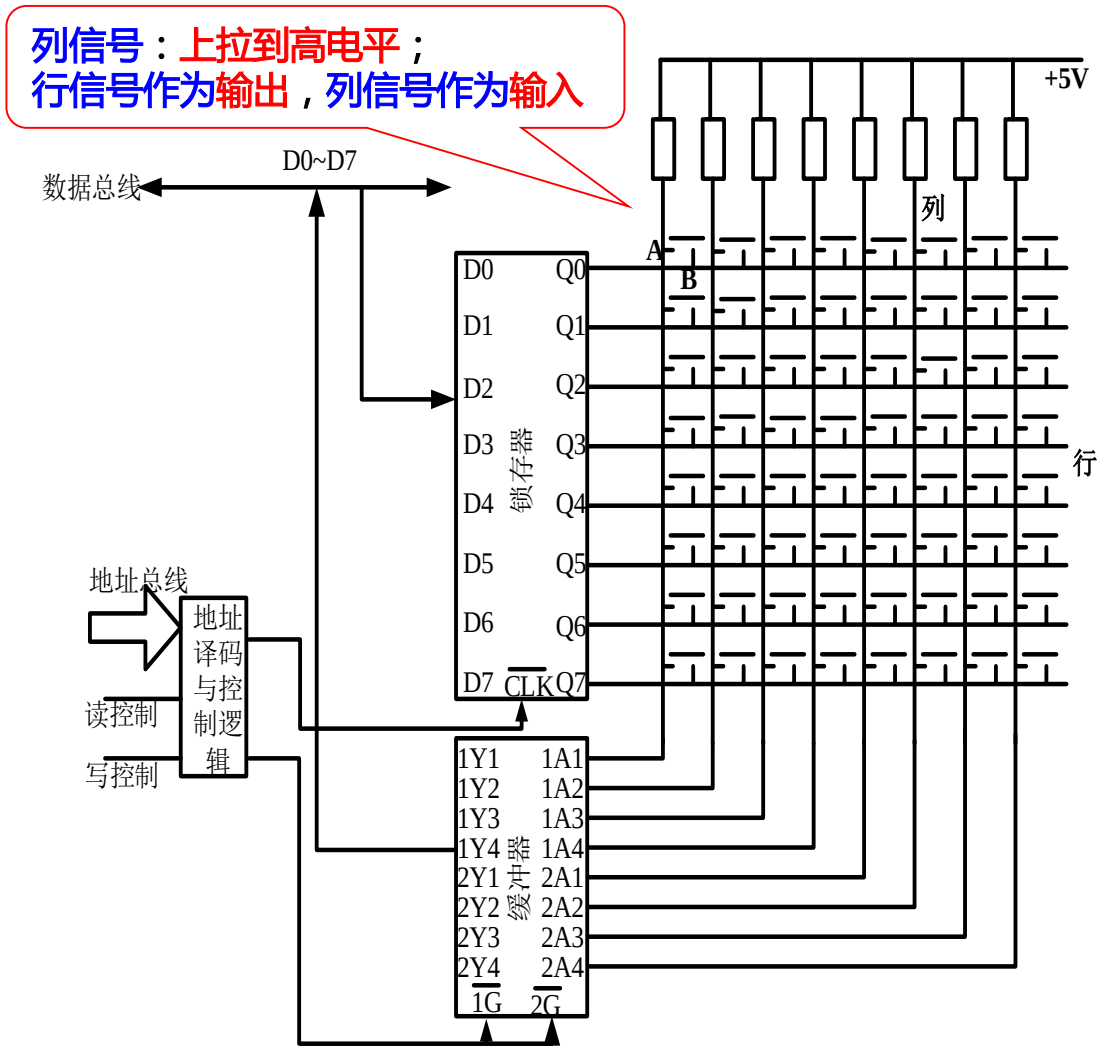
7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

7.5 并行IO接口设计

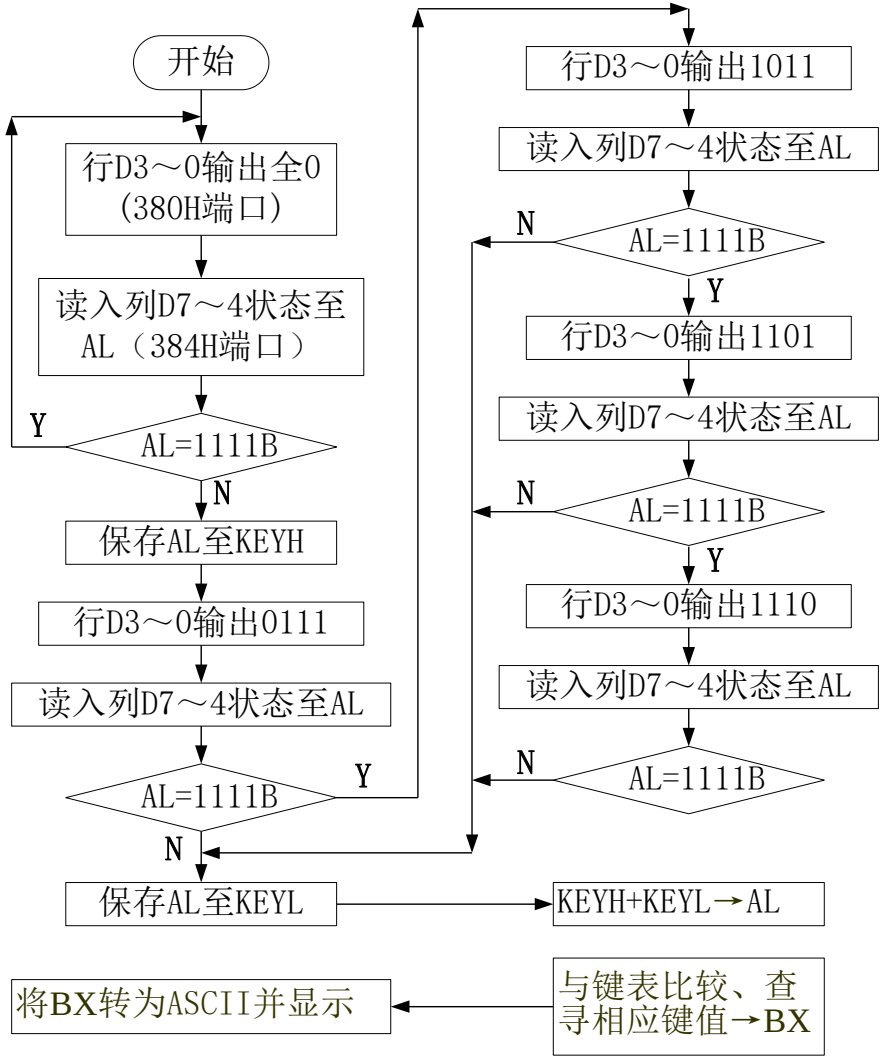
▶ 矩阵式键盘接口

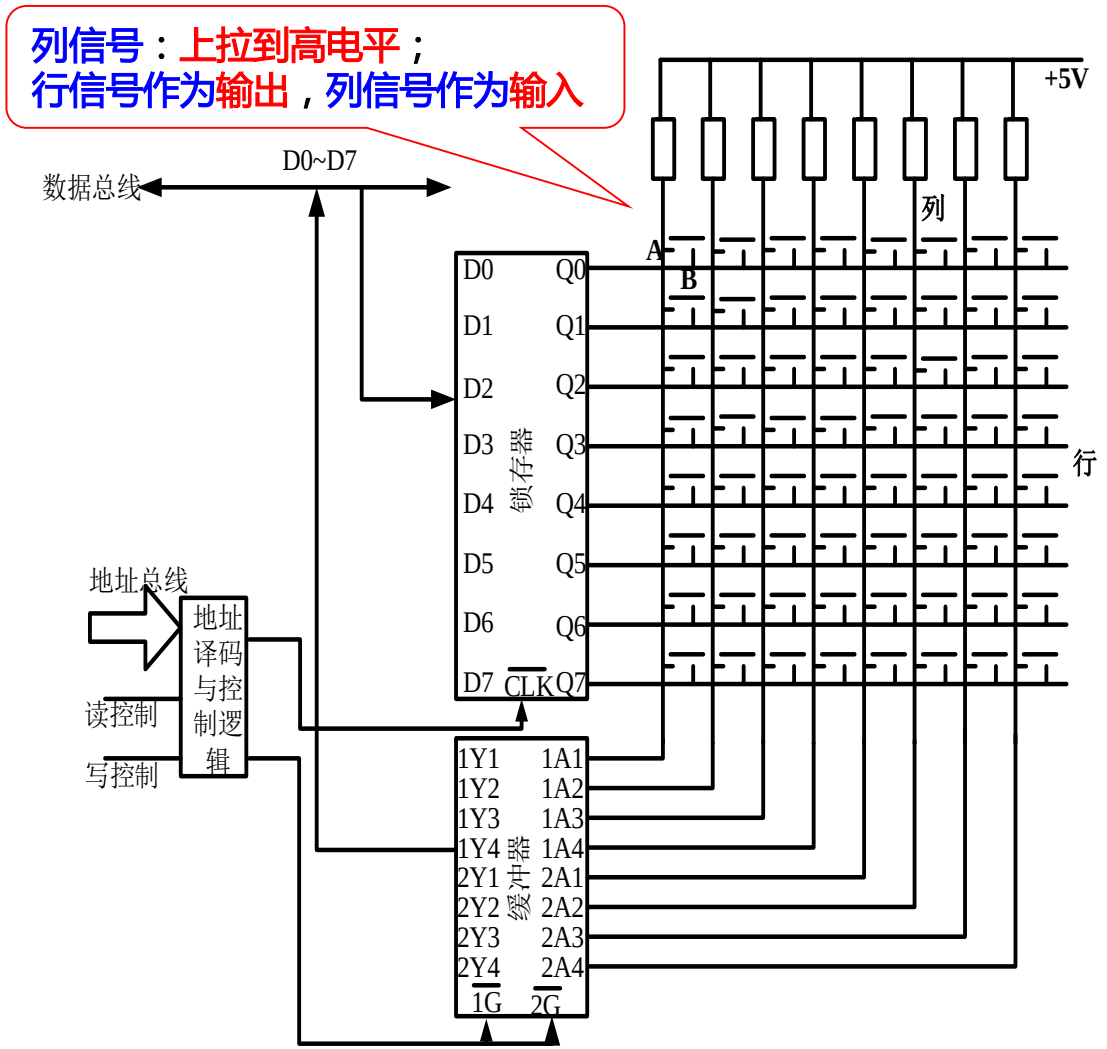
- 微处理器检测矩阵式键盘的状态流程
  - 1）首先使所有行都输出低电平；
  - 2）读取键盘列信号；
  - 3）检测是否有键按下，若没有键按下，获得0xFF，重复步骤2）3）；当有键按下时，此时获得的8位数据不等于0xff，从第一行开始逐行扫描，进入步骤4）；
  - 4）输出某一行为低电平，其余行为高电平
  - 5）读取键盘列信号；
  - 6）检测是否是该行的键按下，若不是该行的键按下则获得0xff，指向下一行，重复步骤4）~6）；当是该行键按下时，此时获得的8位数据不等于0xff，进入步骤7）
  - 7）利用输出的行信号以及读取的列信号，从而确定属于该行该列交叉处的按键被按下。



7.5 并行IO接口设计

▶ 矩阵式键盘接口

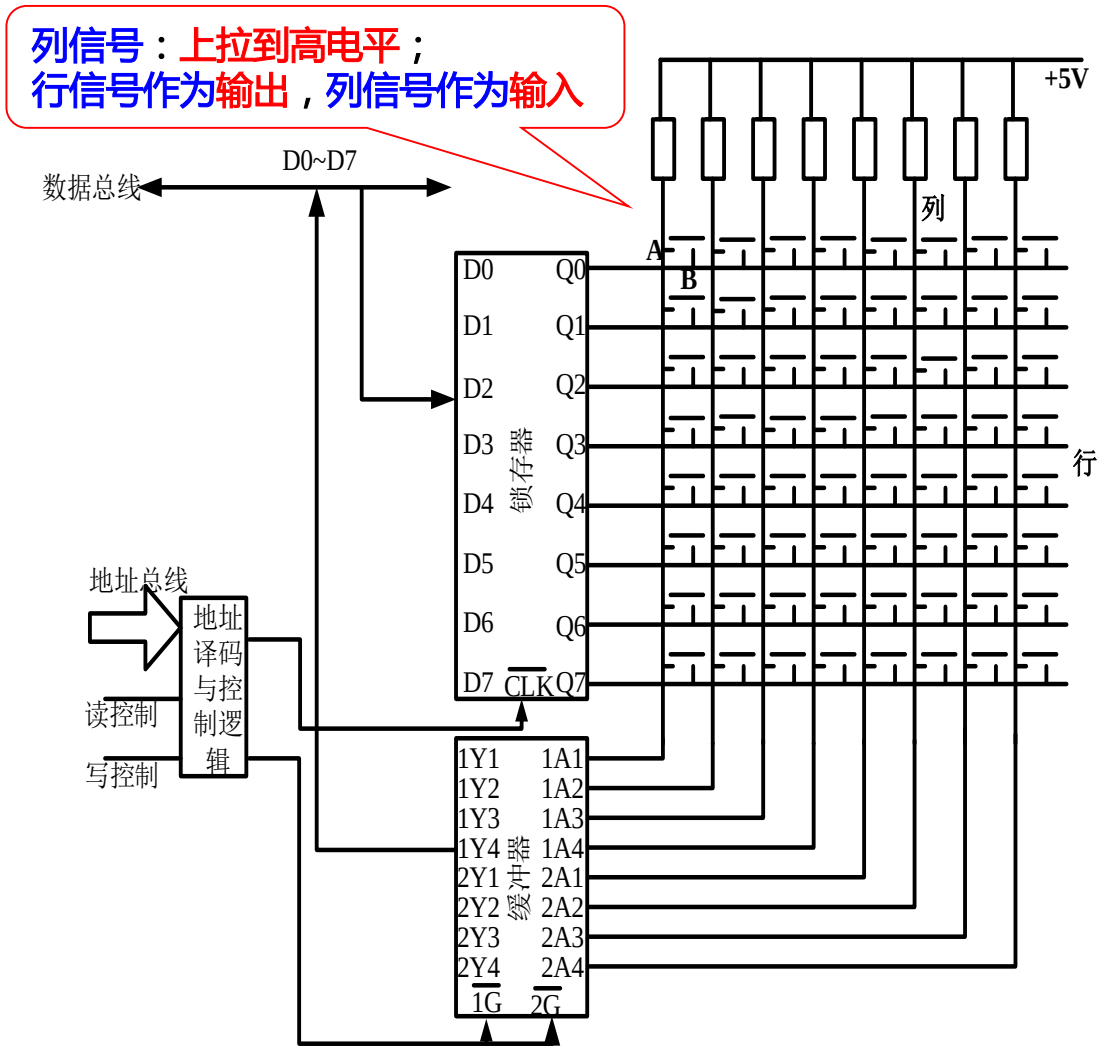




▶ 矩阵式键盘接口

1）首先使所有行都输出低电平；
2）读取键盘列信号；
3）检测是否有键按下，若没有键按下，获得0xFF，重复步骤2）3）；当有键按下时，此时获得的8位数据不等于0xff，从第一行开始，进入步骤4）；
4）输出某一行为低电平，其余行为高电平
5）读取键盘列信号；
6）检测是否是该行的键按下，若不是该行的键按下则获得0xff，指向下一行，重复步骤4）~6）；当是该行键按下时，此时获得的8位数据不等于0xff，进入步骤7）
7）利用输出的行信号以及读取的列信号，从而确定属于该行该列交叉处的按键被按下。

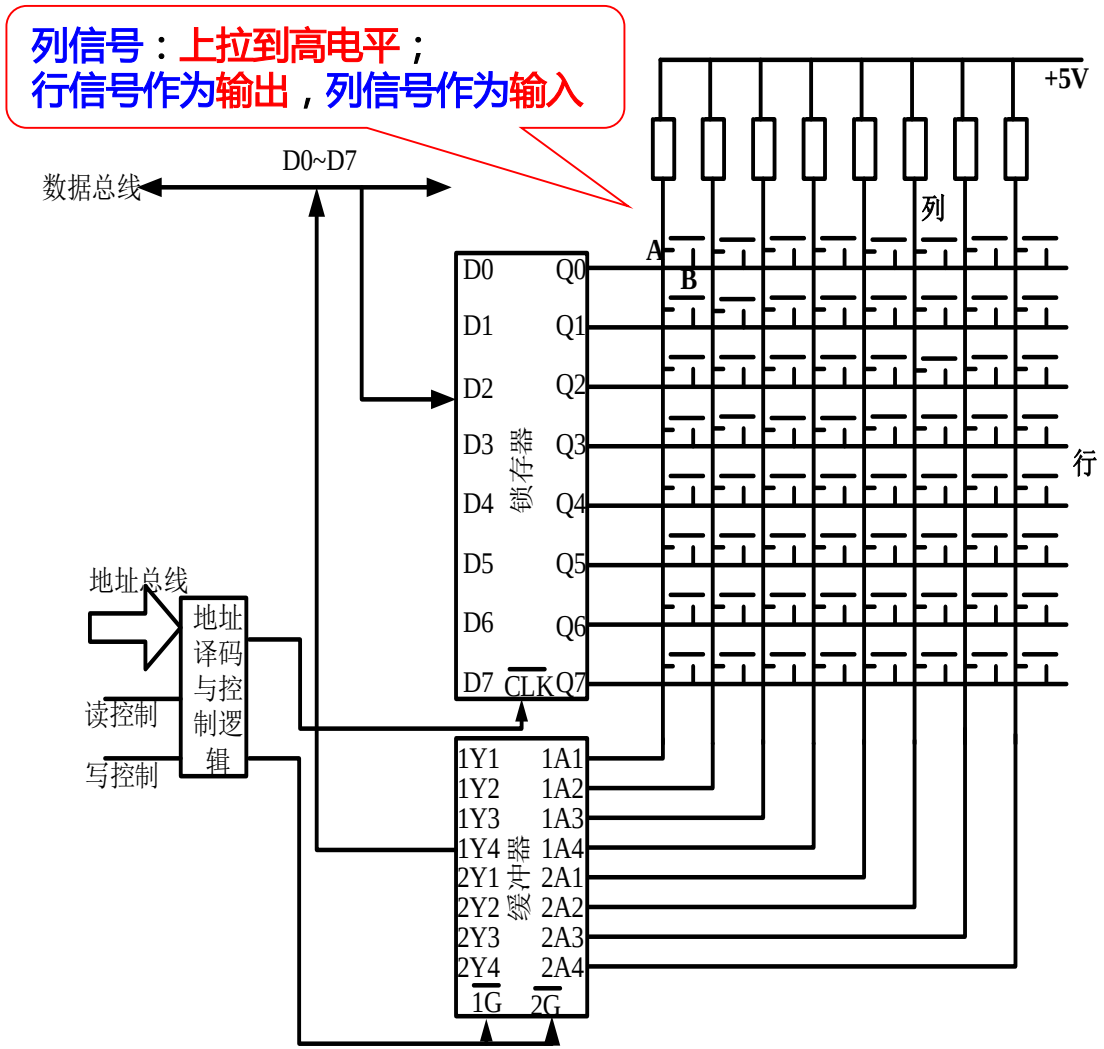
	列0	列1	列2	列3	列4	列5	列6	列7
行0	0xfefe	0xfefd	0xfefb	0xfef7	0xfeef	0xfedf	0xfebf	0xfe7f
行1	0x fdfe	0xfdfd	0xfdfb	0xfdf7	0xfdef	0xfddf	0xfdbf	0xfd7f
行2	0x fbfe	0xfbfd	0xfbfb	0xfbf7	0xfbef	0xfbdf	0xfbbf	0xfb7f
行3	0x f7fe	0xf7fd	0xf7fb	0xf7f7	0xf7ef	0xf7df	0xf7bf	0xf77f
行4	0x effe	0xeffd	0xeffb	0xeff7	0xefef	0xefdf	0xefbf	0xef7f
行5	0x dffe	0xdffd	0xdffb	0xdff7	0xdfef	0xdfdf	0xdfbf	0xdf7f
行6	0x bffe	0xbffd	0xbffb	0xbff7	0xbfef	0xbfdf	0xbfbf	0xbf7f
行7	0x 7ffe	0x7ffd	0x7ffb	0x7ff7	0x7fef	0x7fdf	0x7fbf	0x7f7f



▶ 矩阵式键盘接口

- 采用Xilinx C语言来读取按键的行列信息
  - 按键行和列信息分别保存在变量row和col中。

```
int AddressPort = 0xfffffffe, DataPort = 0xffffffff;
char col, row, rowtemp = 0x01;
Xil_Out8(AddressPort,0x00);                  // 行输出全0
While ((col=Xil_In8(DataPort))==0xff);       // 查询是否有键按下
do
{
    row = ~rowtemp;                          // 有则行扫描
    Xil_Out8 (AddressPort, row);             // 输出某一行为0
    rowtemp = rowtemp << 1;
}
while ((col=Xil_In8 (DataPort))==0xff)
```



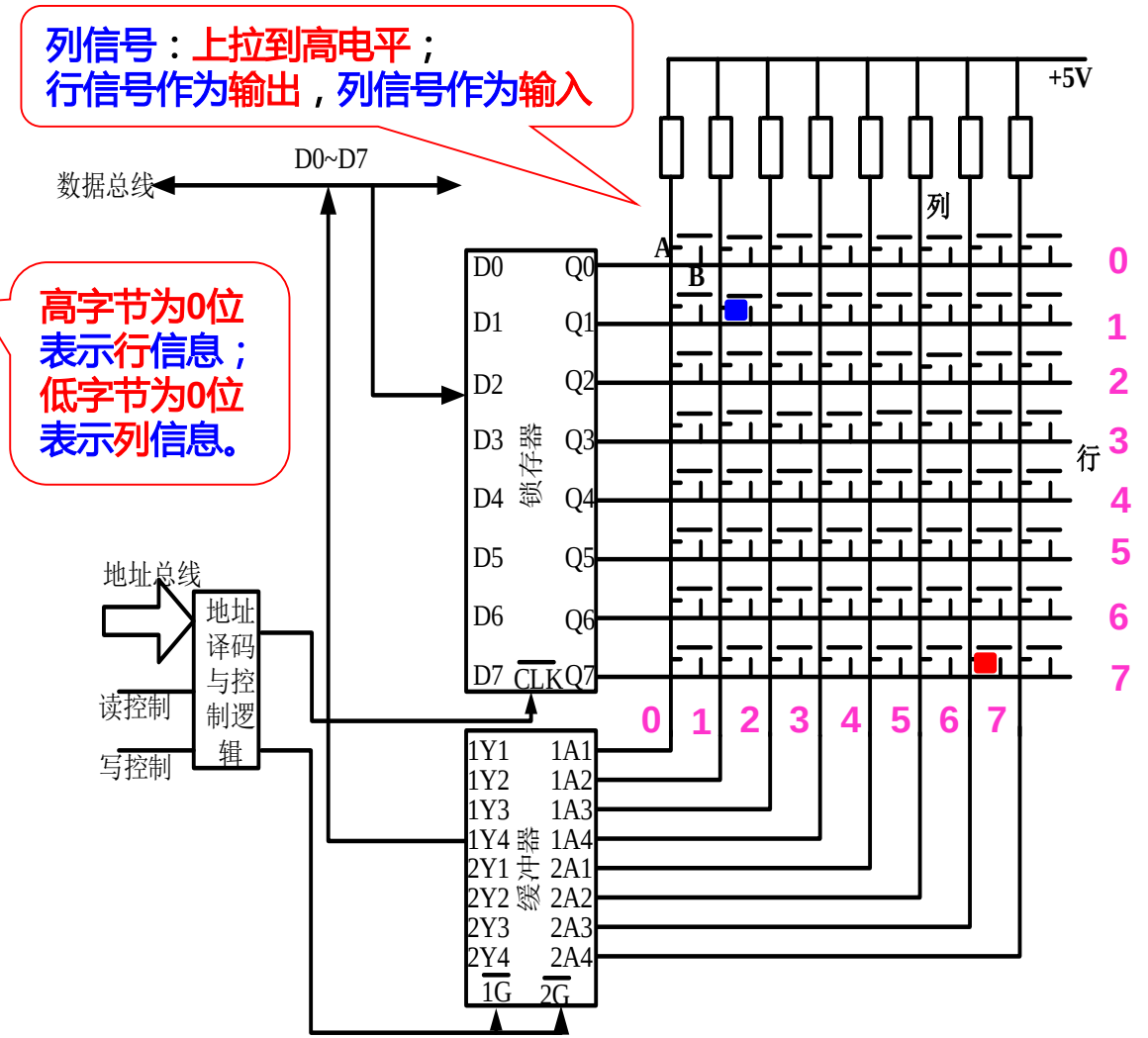
▶ 矩阵式键盘接口

- 键值与编码关系

```
short  int  scancode[64]=
{0xfefe, 0xfefd, 0xfefb, 0xfef7, 0xfeef, 0xfedf, 0xfebf, 0xfe7f,
 0xfdfe, 0xfdfd, 0xfdfb, 0xfdf7, 0xfdef, 0xfddf, 0xfdbf, 0xfd7f,
 0xfbfe, 0xfbfd, 0xfbfb, 0xfbf7, 0xfbef, 0xfbdf, 0xfbbf, 0xfb7f,
 0xf7fe, 0xf7fd, 0xf7fb, 0xf7f7, 0xf7ef, 0xf7df, 0xf7bf, 0xf77f,
 0xeffe, 0xeffd, 0xeffb, 0xeff7, 0xefef, 0xefdf, 0xefbf, 0xef7f,
 0xdffe, 0xdffd, 0xdffb, 0xdff7, 0xdfef, 0xdfdf, 0xdfbf, 0xdf7f,
 0xbffe, 0xbffd, 0xbffb, 0xbff7, 0xbfef, 0xbfdf, 0xbfbf, 0xbf7f,
 0x7ffe, 0x7ffd, 0x7ffb, 0x7ff7, 0x7fef, 0x7fdf, 0x7fbf,0x7f7f
};
```

```
KeyBtn=(short int)row;
KeyBtn=(KeyBtn<<8) | col;
for(i=0;i<64;i++)
{
    if(scancode[i]==KeyBtn)
    break;
}
xil_printf("Key %d is pressed\n\r",i);
```

【思考】若图中蓝色位置处的按键按下，KeyBtn的值为多少？i值又为多少？如果红色按键按下呢？

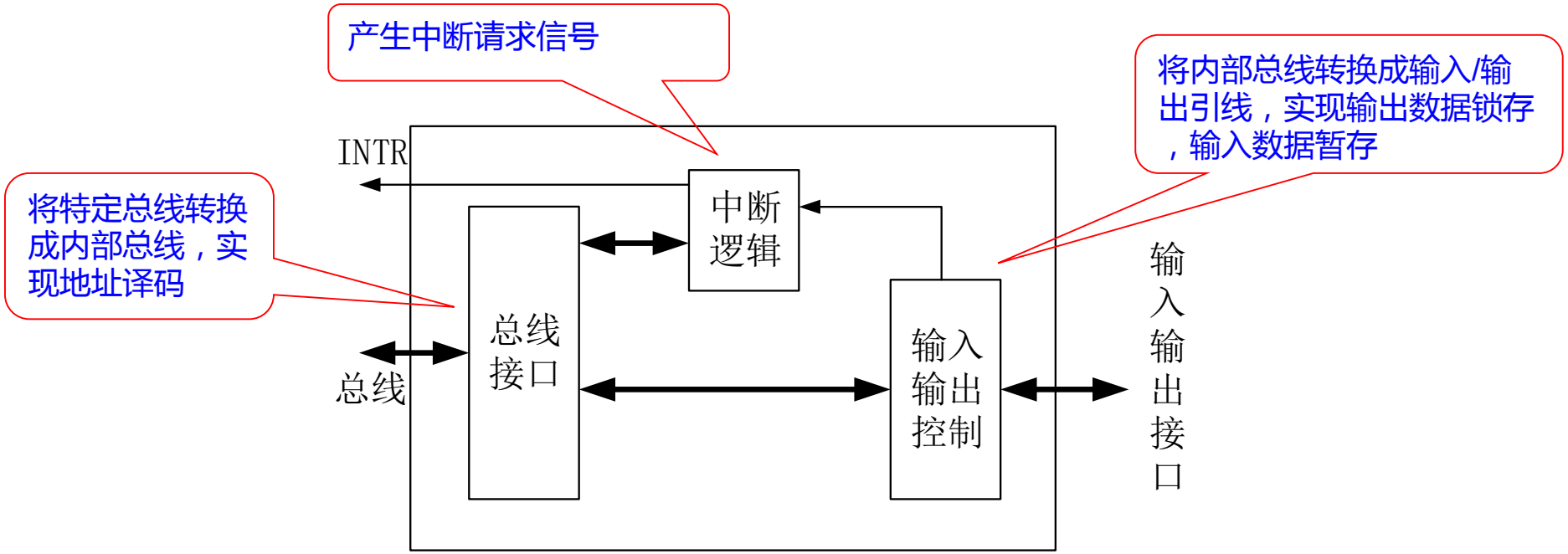


7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

▶ GPIO控制器

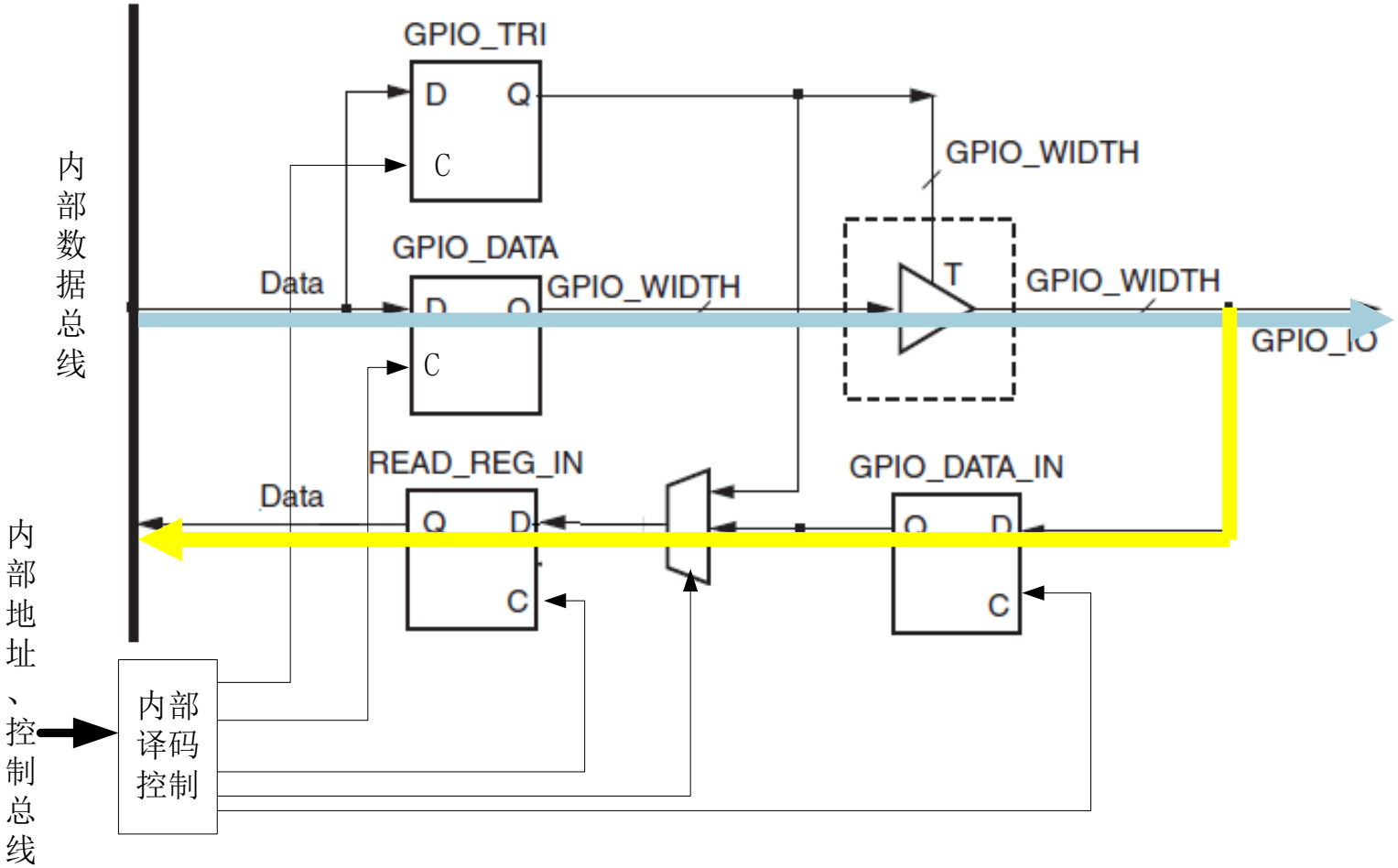
- GPIO（general purpose IO）是通用并行IO接口的简称。它将总线信号转换为IO设备要求的信号类型，实现地址译码、输出数据锁存、输入数据缓冲的功能



7.5 并行IO接口设计

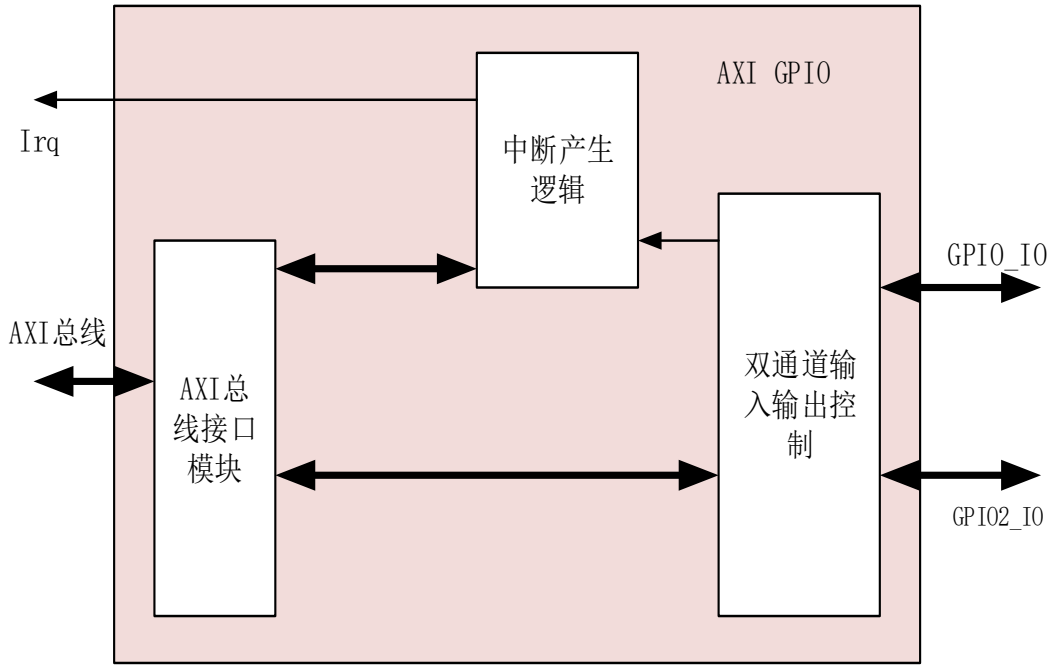
▶ GPIO控制器

- 1位输入输出控制模块内部框图



▶ GPIO控制器

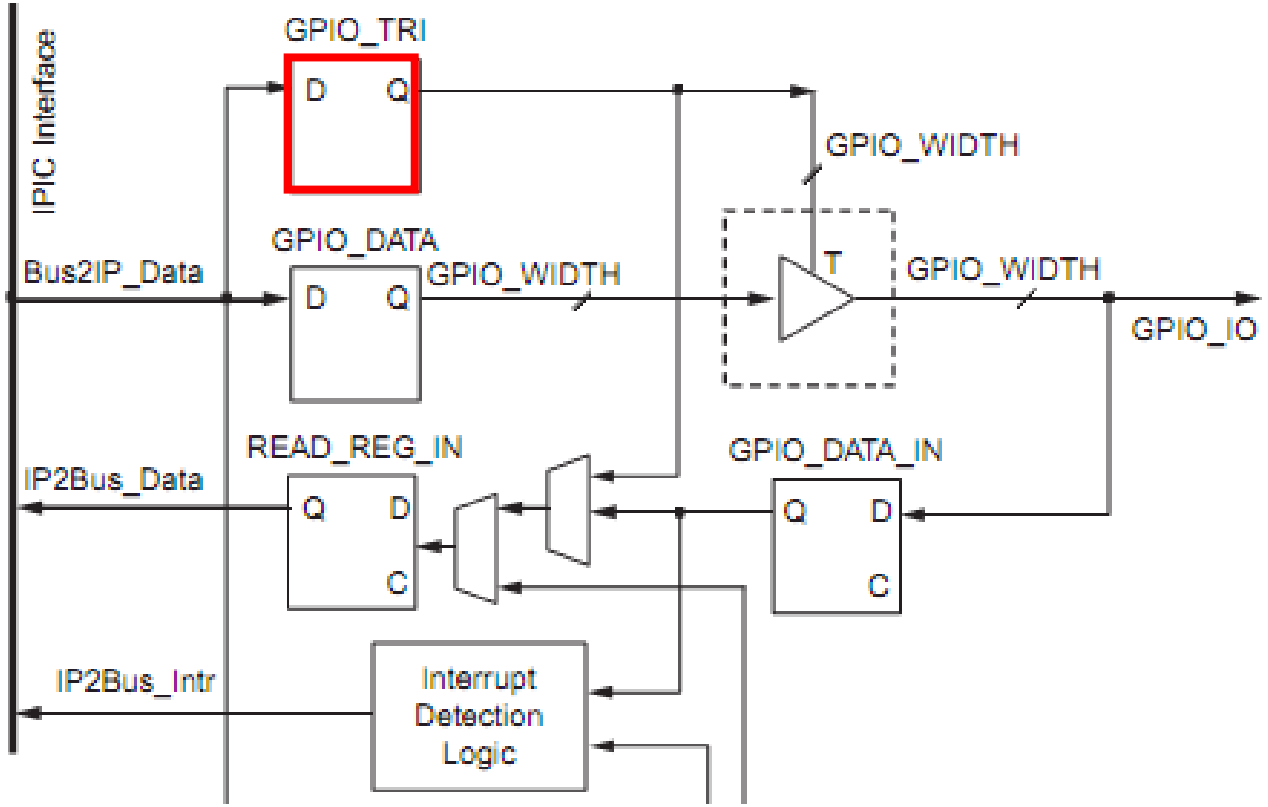
- Xilinx AXI总线GPIO IP核
  - 包括AXI总线接口模块、中断产生逻辑、双通道I/O模块
  - 每个通道都可以支持1~32位的数据输入输出，可以配置为单输入、单输出或双向输入输出 —— 怎么配置？？
    （通过寄存器）



7.5 并行IO接口设计

- GPIO控制器
  - Xilinx AXI总线GPIO IP核
    - I/O方向
      - 当GPIO_TRI某位为0时，GPIO相应的IO引脚配置为输出；
      - 当GPIO_TRI某位为1时，GPIO相应的IO引脚配置为输入；
    - I/O读写

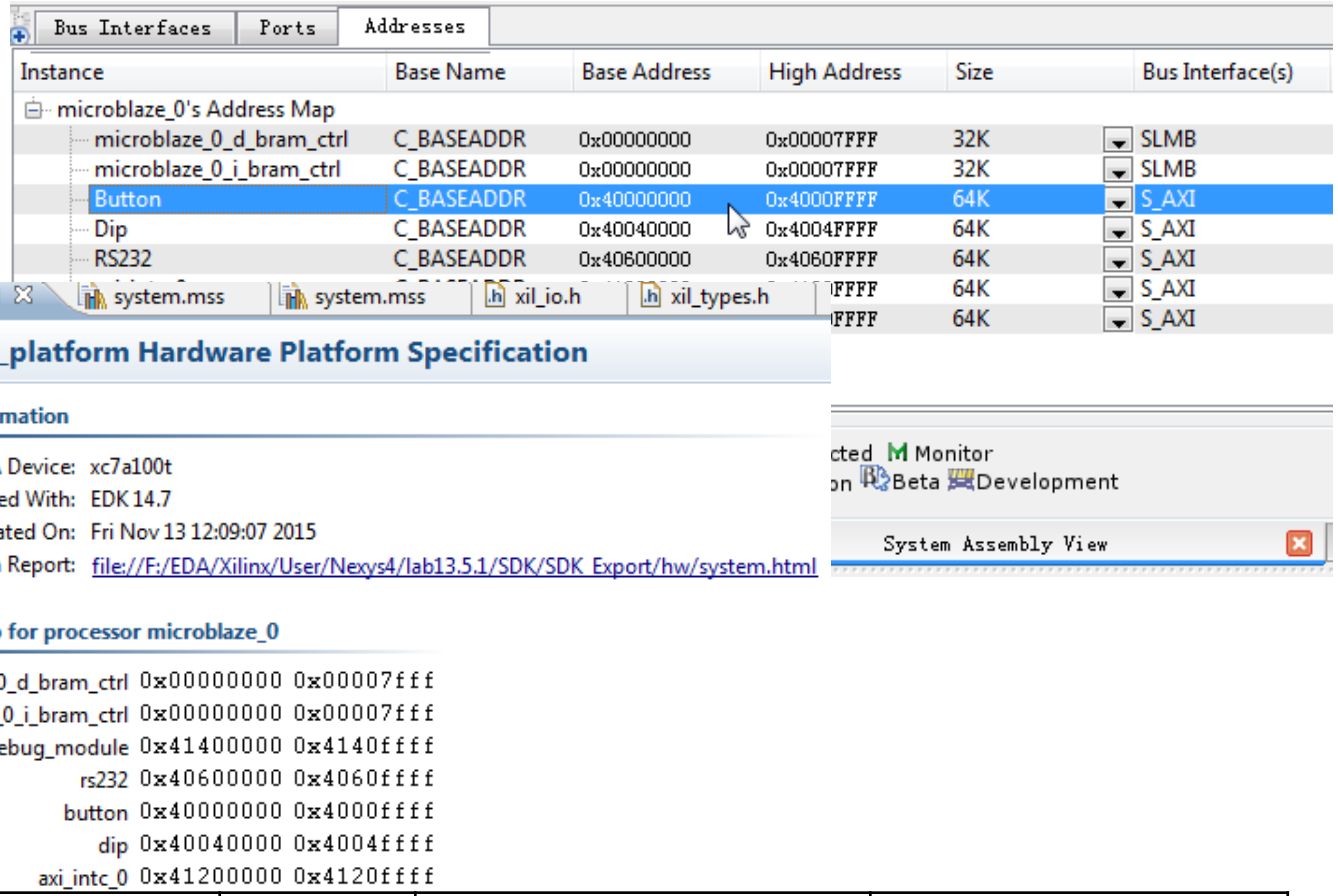
```
# include “xil_io.h”
…
Xil_In8(Addr);
Xil_In16(Addr);
Xil_In32(Addr);

Xil_Out8(Addr, Value);
Xil_Out16(Addr, Value);
Xil_Out32(Addr, Value);
```



寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

7.5 并行IO接口设计

- GPIO控制器
  - Xilinx AXI总线GPIO IP核
    - I/O模块的基地址
      - XPS和SDK中均可查看
      - xparameters.h文件也可查看

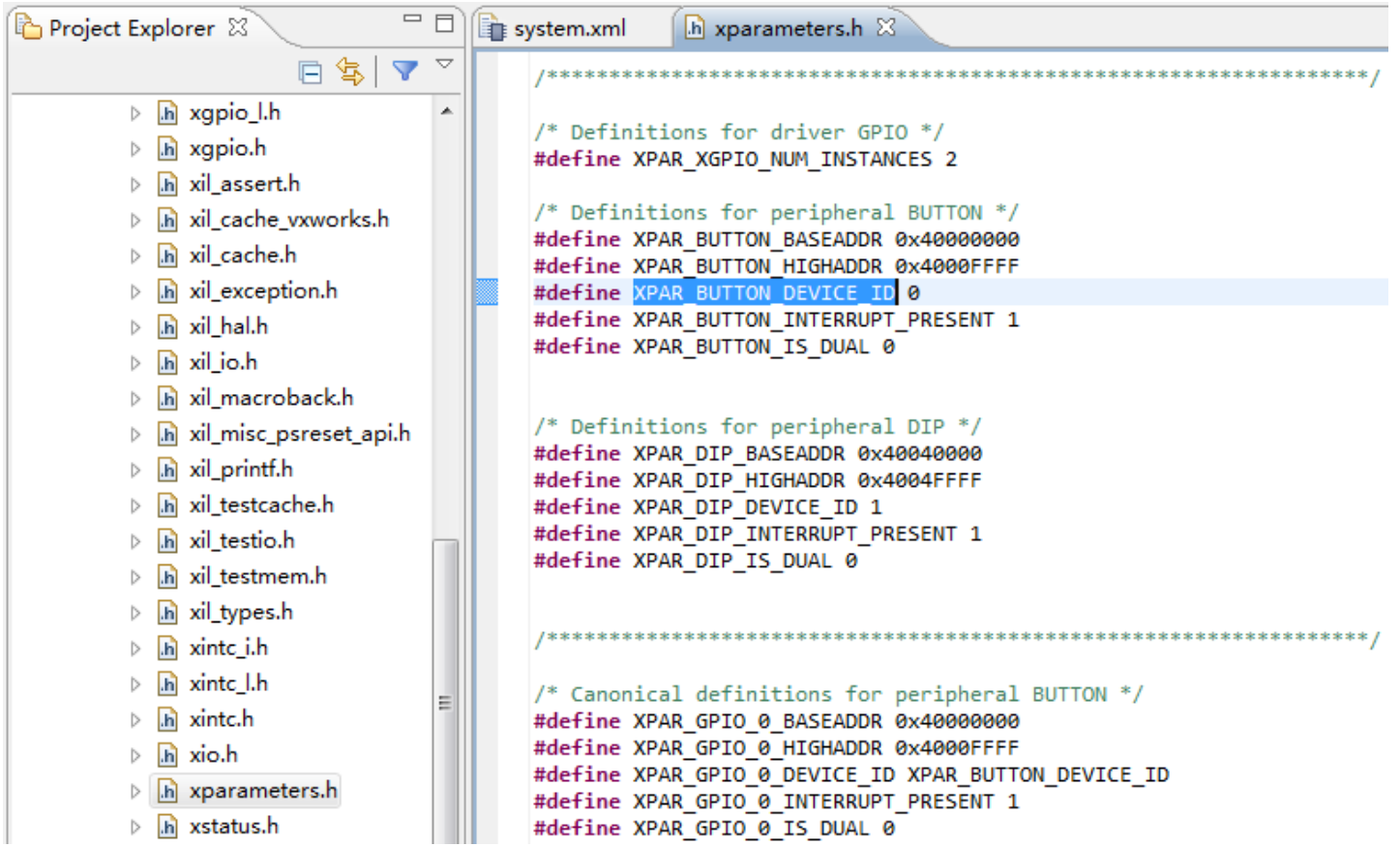



```
# include “xil_io.h”
…
Xil_In8(Addr);
Xil_In16(Addr);
Xil_In32(Addr);

Xil_Out8(Addr, Value);
Xil_Out16(Addr, Value);
Xil_Out32(Addr, Value);
```

寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

7.5 并行IO接口设计

- GPIO控制器
  - Xilinx AXI总线GPIO IP核
    - I/O模块的基地址
      - XPS和SDK中均可查看
      - xparameters.h文件也可查看

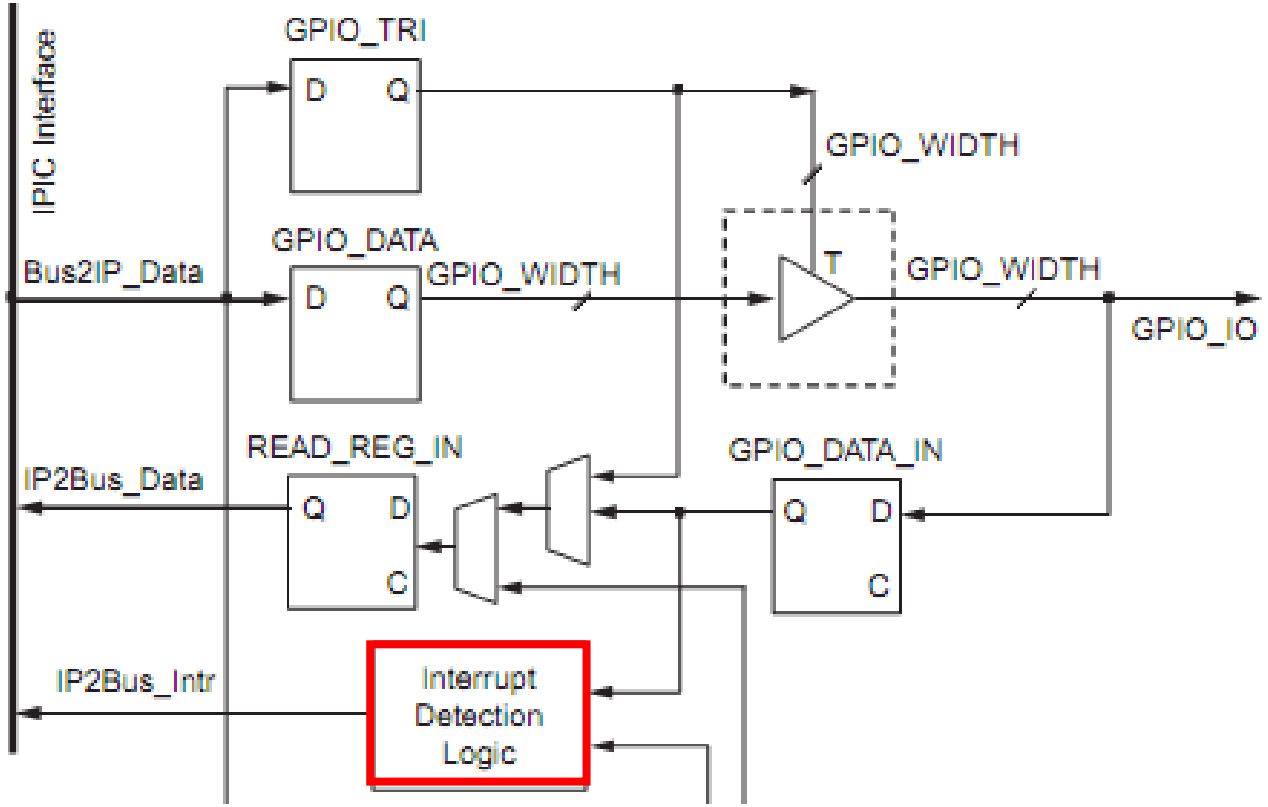



```
# include “xil_io.h”
…
Xil_In8(Addr);
Xil_In16(Addr);
Xil_In32(Addr);

Xil_Out8(Addr, Value);
Xil_Out16(Addr, Value);
Xil_Out32(Addr, Value);
```

寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

7.5 并行IO接口设计

- GPIO控制器
  - Xilinx AXI总线GPIO IP核
    - I/O中断

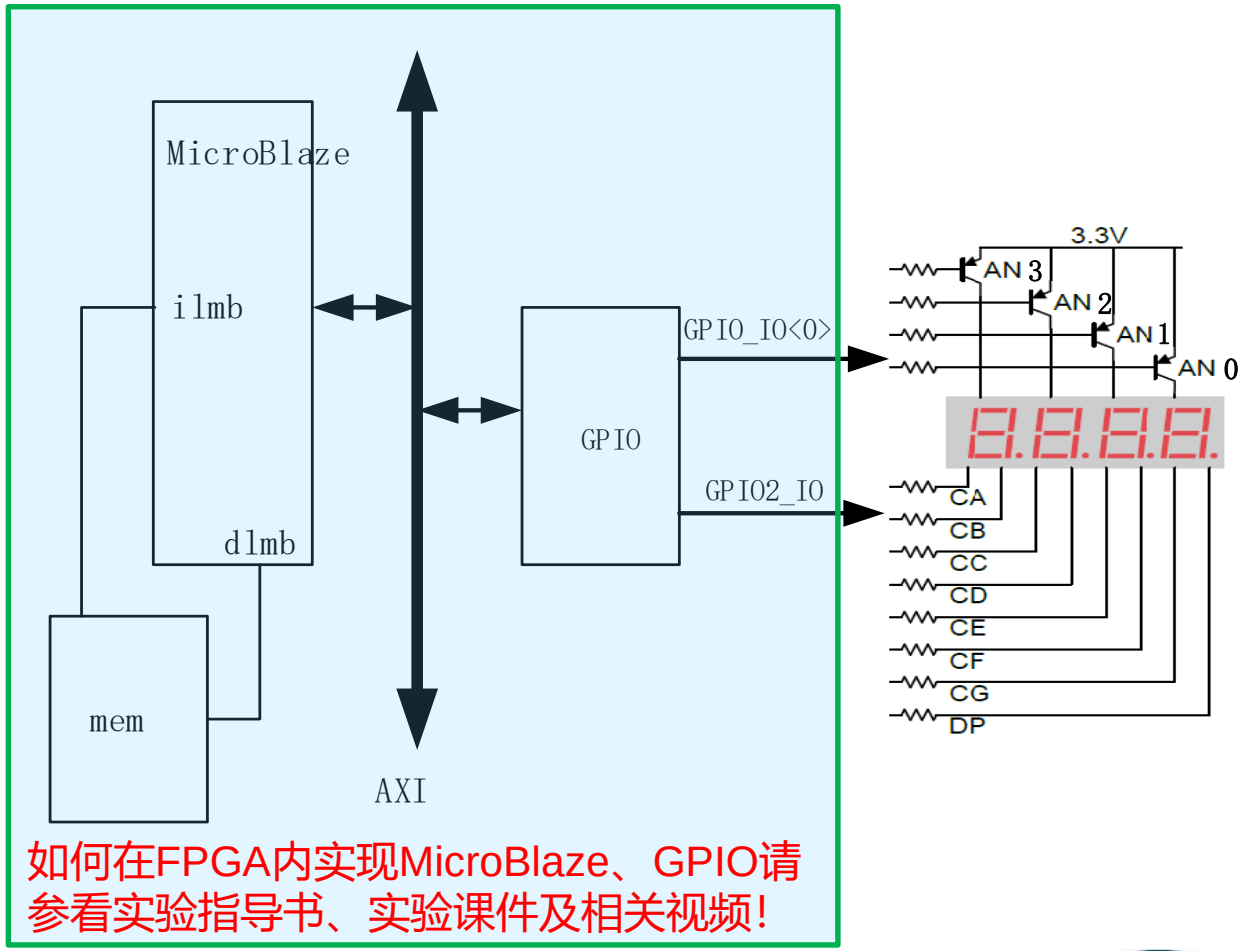


名称	偏移地址	含义	读写操作
GIER	0x11C	全局中断屏蔽寄存器	最高位bit31控制GPIO是否输出中断信号Irq
IP IER	0x128	中断屏蔽寄存器	控制各个通道是否允许产生中断 bit0-通道1；bit1-通道2
IP ISR	0x120	中断状态寄存器	各个通道的中断请求状态，写1将清除相应位的中断状态 bit0-通道1；bit1-通道2

7.5 并行IO接口设计

▶ GPIO控制器

- 【例】采用GPIO控制器，控制四位7段数码管显示字符PASS
  - 位选信号
    - GPIO_IO<0>连接AN0，
    - GPIO_IO<1>连接AN1，
    - GPIO_IO<2>连接AN2，
    - GPIO_IO<3>连接AN3；
  - 段选信号
    - GPIO2_IO<0>连接CA，
    - GPIO2_IO<1>连接CB，
    - GPIO2_IO<2>连接CC，
    - GPIO2_IO<3>连接CD，
    - GPIO2_IO<4>连接CE，
    - GPIO2_IO<5>连接CF，
    - GPIO2_IO<6>连接CG，
    - GPIO2_IO<7>连接DP。

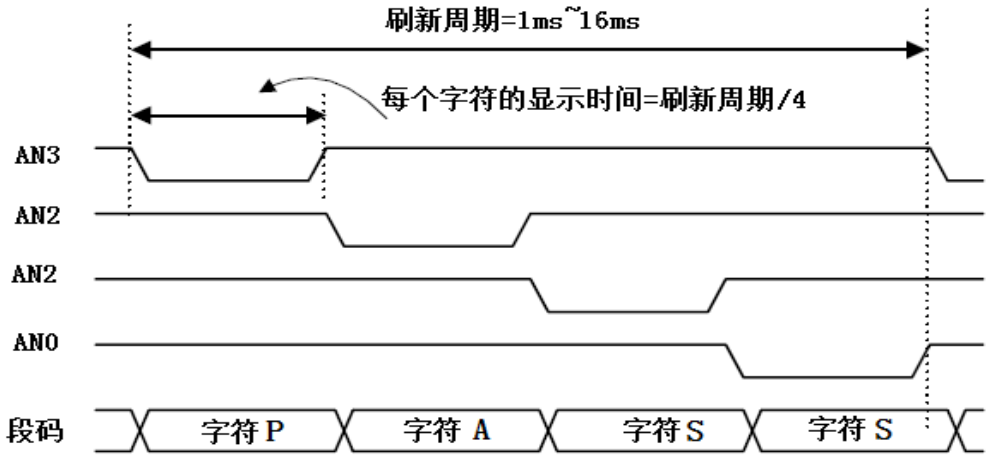


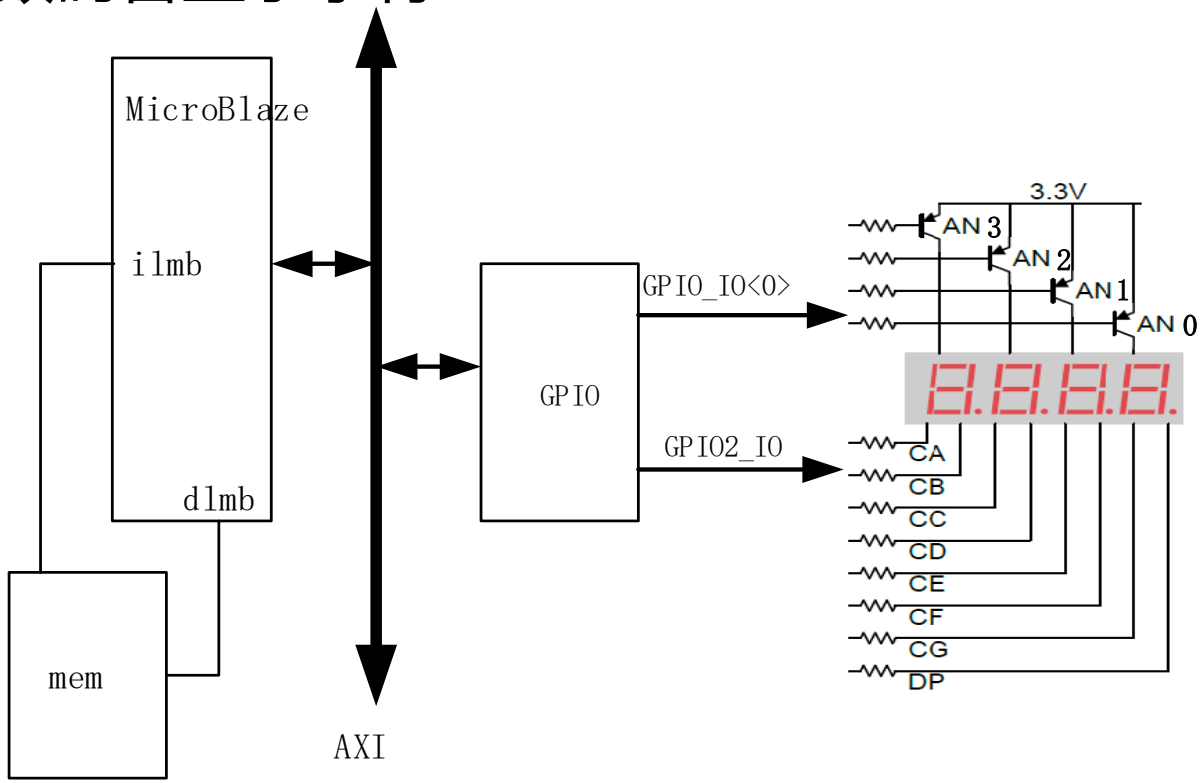
7.5 并行IO接口设计

▶ GPIO控制器

- 【例6】采用GPIO控制器，控制四位7段数码管显示字符PASS

GPIO控制器的基地址为XPAR_SEG_0_BASEADDR，
那么GPIO_DATA地址为XPAR_SEG_0_BASEADDR，
GPIO2_DATA地址为XPAR_SEG_0_BASEADDR+0x8，
GPIO_TRI地址为XPAR_SEG_0_BASEADDR+0x4，
GPIO2_TRI地址为XPAR_SEG_0_BASEADDR+0xC。
字符串 “PASS”的段码为：0x8c,0x88,0x92,0x92；
4位7段数码管从左到右的位码分别为0xf7,0xfb,0xfd,0xfe





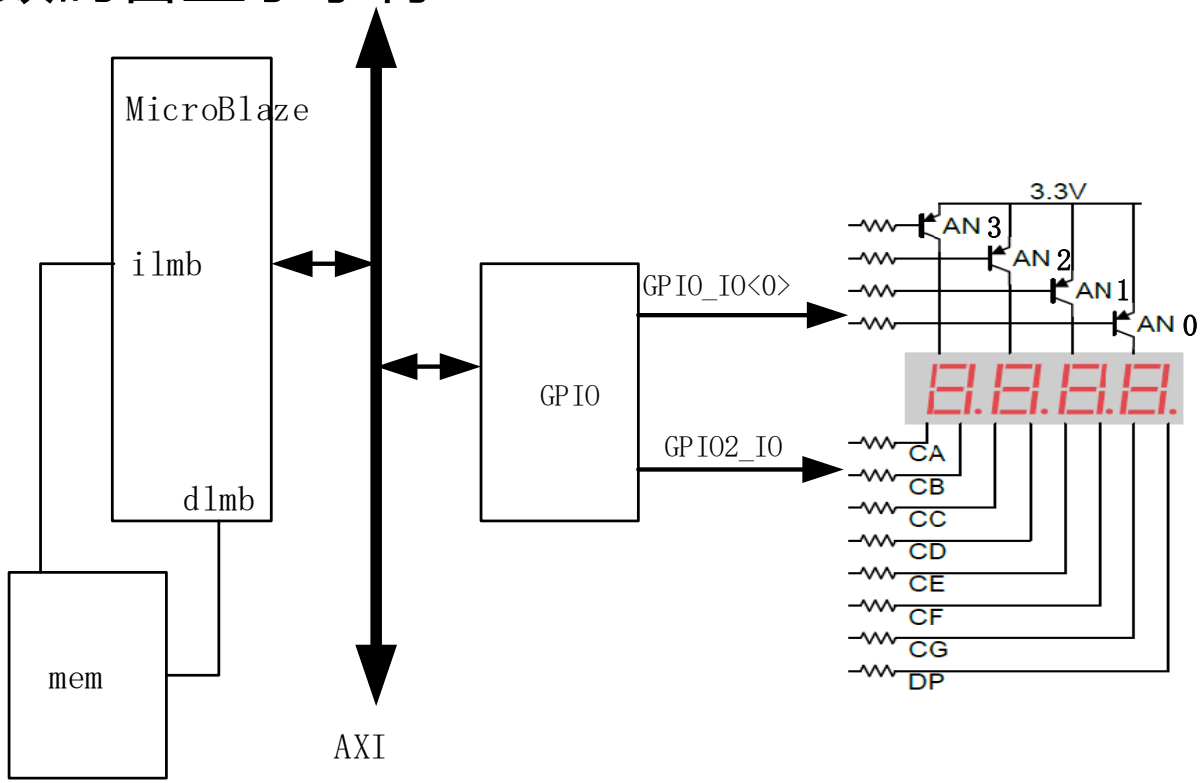
寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

7.5 并行IO接口设计

▶ GPIO控制器

- 【例6】采用GPIO控制器，控制四位7段数码管显示字符PASS

```
控制程序段
char segcode[4]={0x8c,0x88,0x92,0x92};
char pos=0xf7;   //初始化位码
int i,j;
Xil_Out8(XPAR_SEG_0_BASEADDR+0x4,0x0);  //使GPIO_IO通道输出
Xil_Out8(XPAR_SEG_0_BASEADDR+0xc,0x0);  //使GPIO2_IO通道输出
while(1)
{
    for(i=0;i<4;i++)
    {
    Xil_Out8(XPAR_SEG_0_BASEADDR,pos);   //输出位码
    Xil_Out8(XPAR_SEG_0_BASEADDR+0x8,segcode[i]); //输出
                                                  //段码
    for(j=0;j<10000;j++);  //延时，使人眼能正确的看到显示的字符
    pos=pos>>1;
    }
    pos=0xf7;
}
```

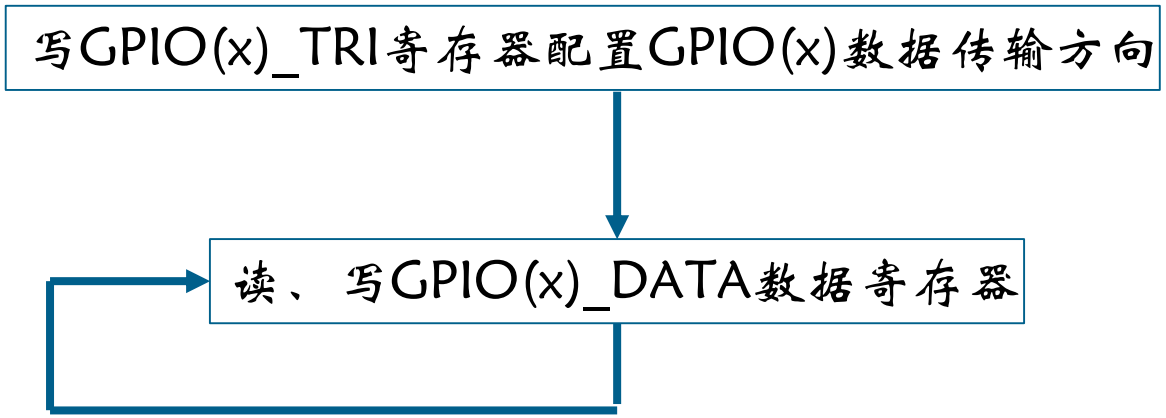


寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

▶ GPIO控制器

- 【例6】采用GPIO控制器，控制四位7段数码管显示字符PASS

```
控制程序段
char segcode[4]={0x8c,0x88,0x92,0x92};
char pos=0xf7;   //初始化位码
int i,j;
Xil_Out8(XPAR_SEG_0_BASEADDR+0x4,0x0);  //使GPIO_IO通道输出
Xil_Out8(XPAR_SEG_0_BASEADDR+0xc,0x0);  //使GPIO2_IO通道输出
while(1)
{
   for(i=0;i<4;i++)
   {
   Xil_Out8(XPAR_SEG_0_BASEADDR,pos);   //输出位码
   Xil_Out8(XPAR_SEG_0_BASEADDR+0x8,segcode[i]); //输出
                                                 //段码
   for(j=0;j<10000;j++);  //延时，使人眼能正确的看到显示的字符
   pos=pos>>1;
   }
   pos=0xf7;
}
```

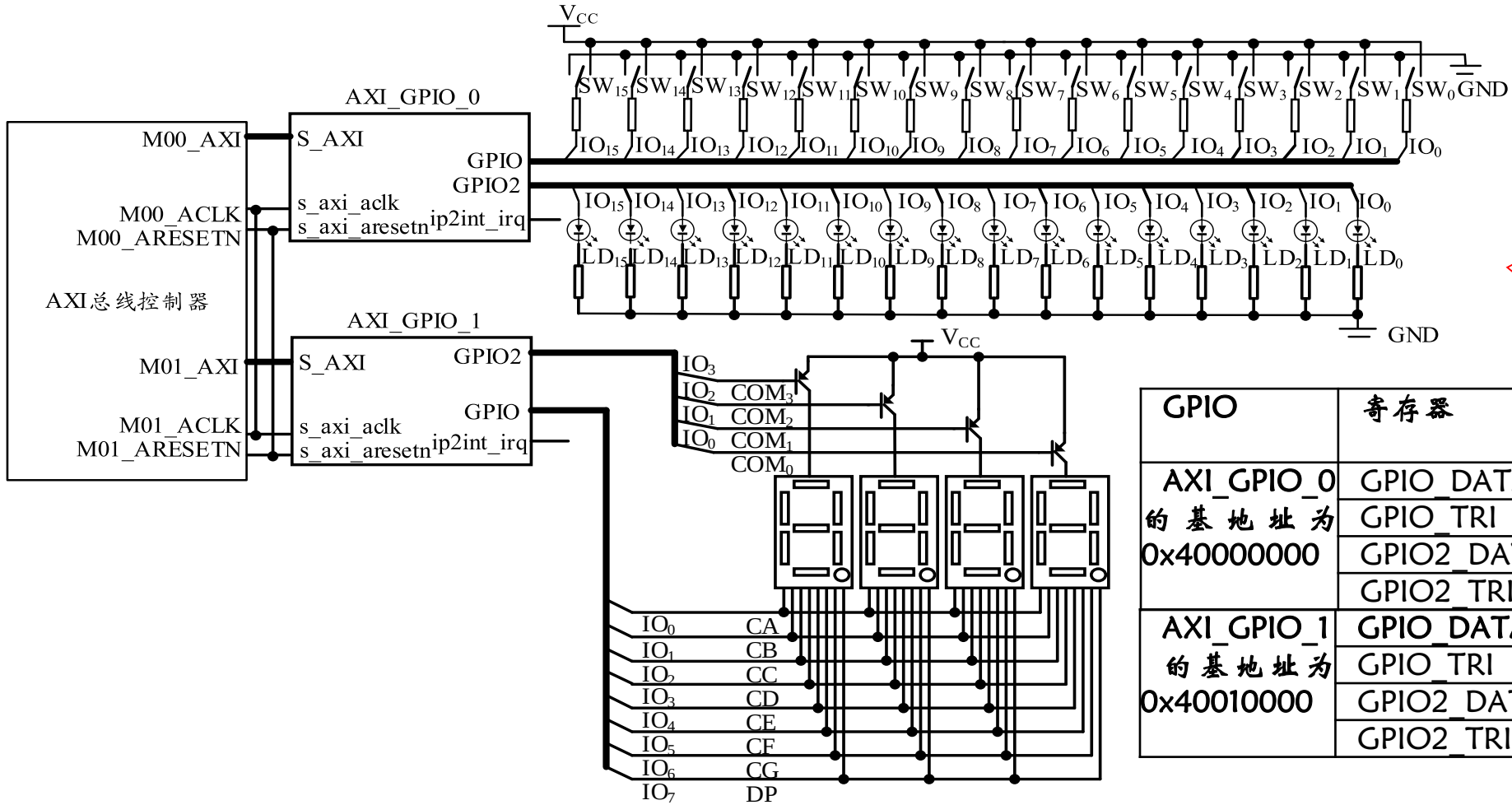


寄存器名称	偏移地址	初始值	含义	读写操作
GPIO_DATA	0x0	0	通道1数据寄存器	通道1数据
GPIO_TRI	0x4	0	通道1三态控制寄存器	写控制通道1传输方向
GPIO2_DATA	0x8	0	通道1数据寄存器	通道2数据
GPIO2_TRI	0xC	0	通道1三态控制寄存器	写控制通道1传输方向

7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，控制16个开关、16个按键，四位数码管



GPIO_0_GPIO ：输入；
GPIO_0_GPIO2：输出；

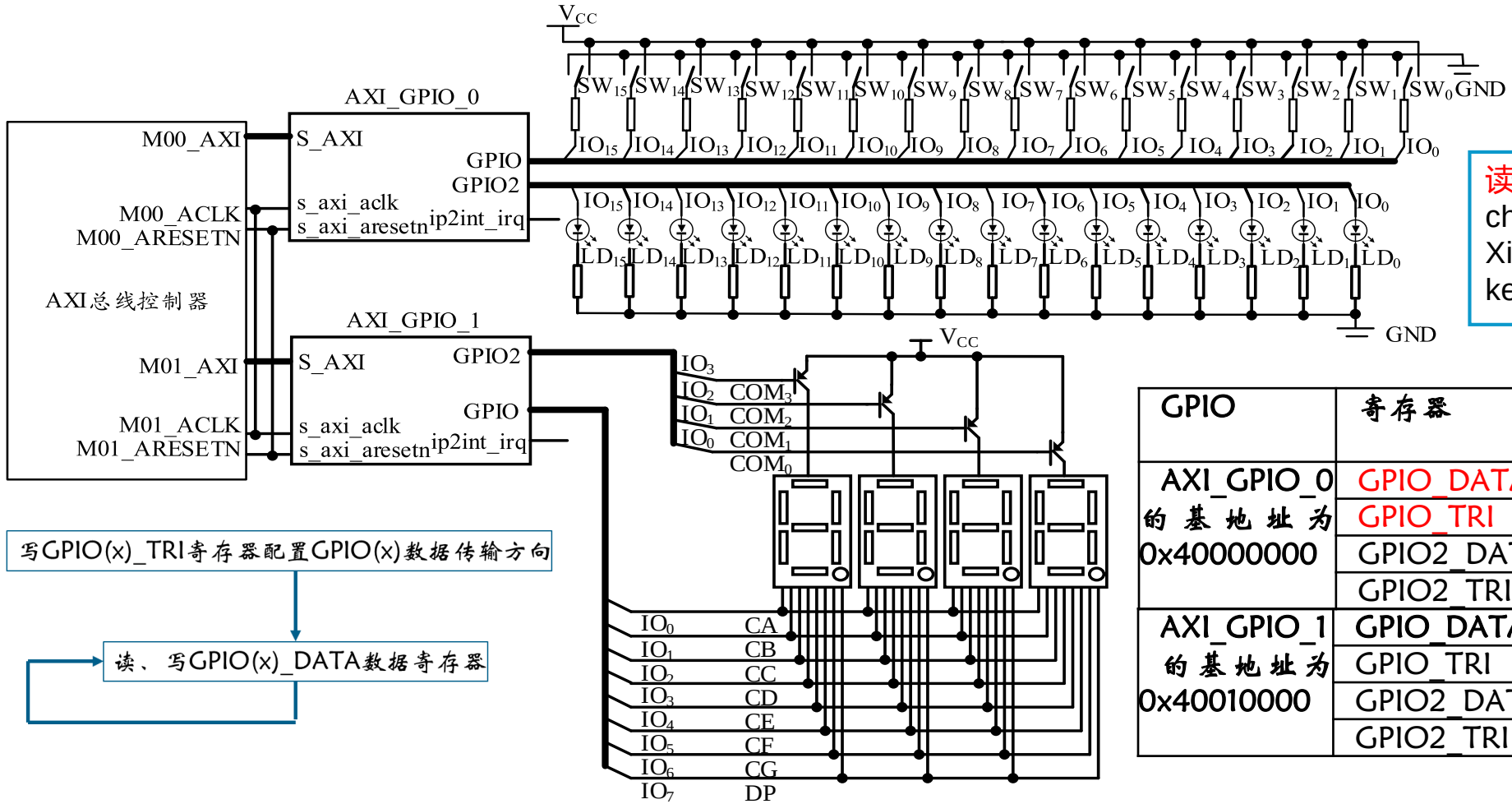
GPIO_1_GPIO ：输出；
GPIO_1_GPIO2：输出；

GPIO	寄存器	地址	有效数据位	读写控制
AXI_GPIO_0的基地址为0x40000000	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的基地址为0x40010000	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

7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，控制16个开关、16个按键，四位数码管



读SW[15:0]程序段：

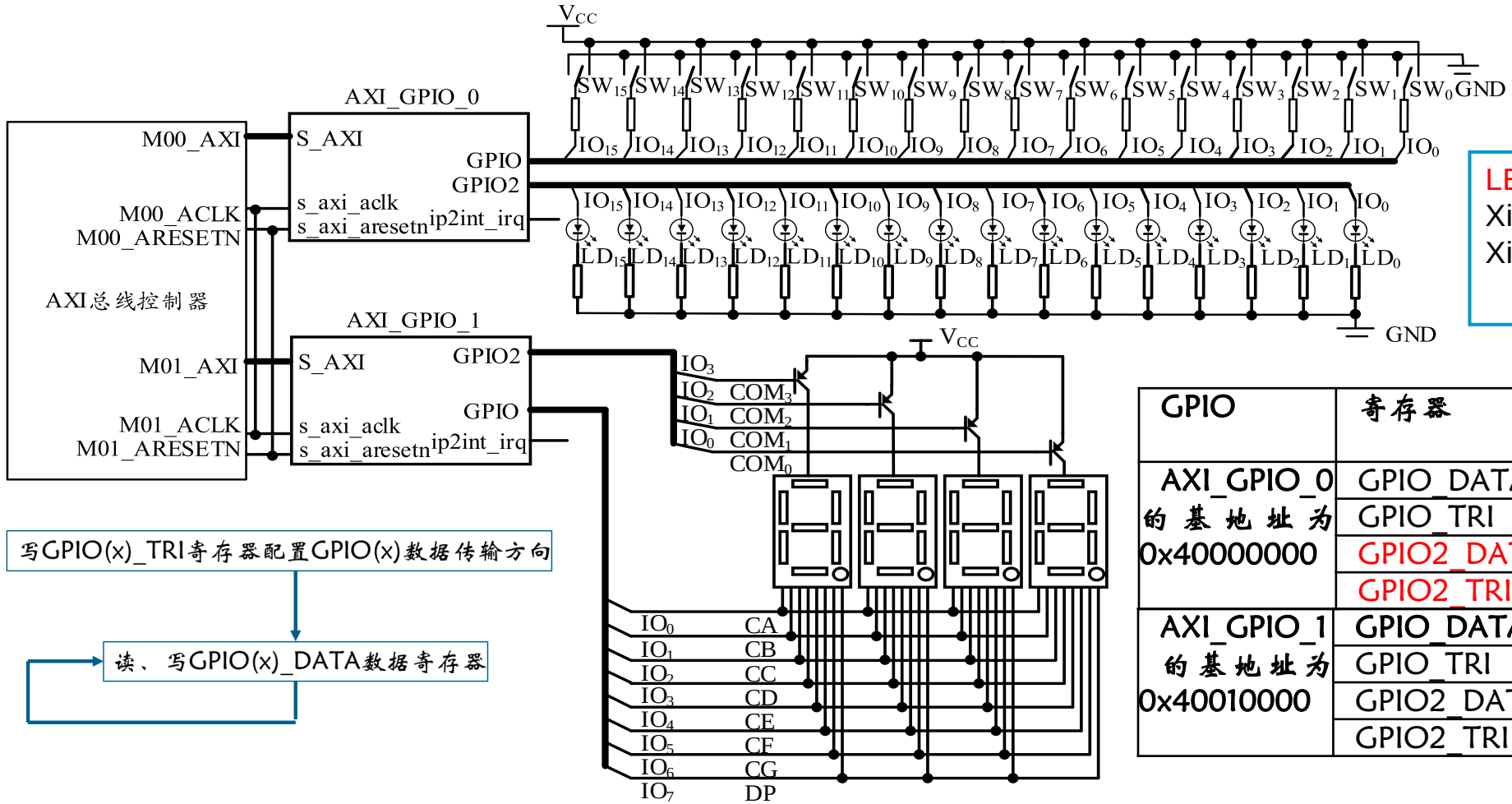
```
char unsigned short key；
Xil_Out16(0x40000004,0xffff);
key = Xil_In16(0x40000000);
```

GPIO	寄存器	地址	有效数据位	读写控制
AXI_GPIO_0的基地址为0x40000000	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的基地址为0x40010000	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

7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，控制16个开关、16个按键，四位数码管



LED[15:0]间隔点亮程序段：

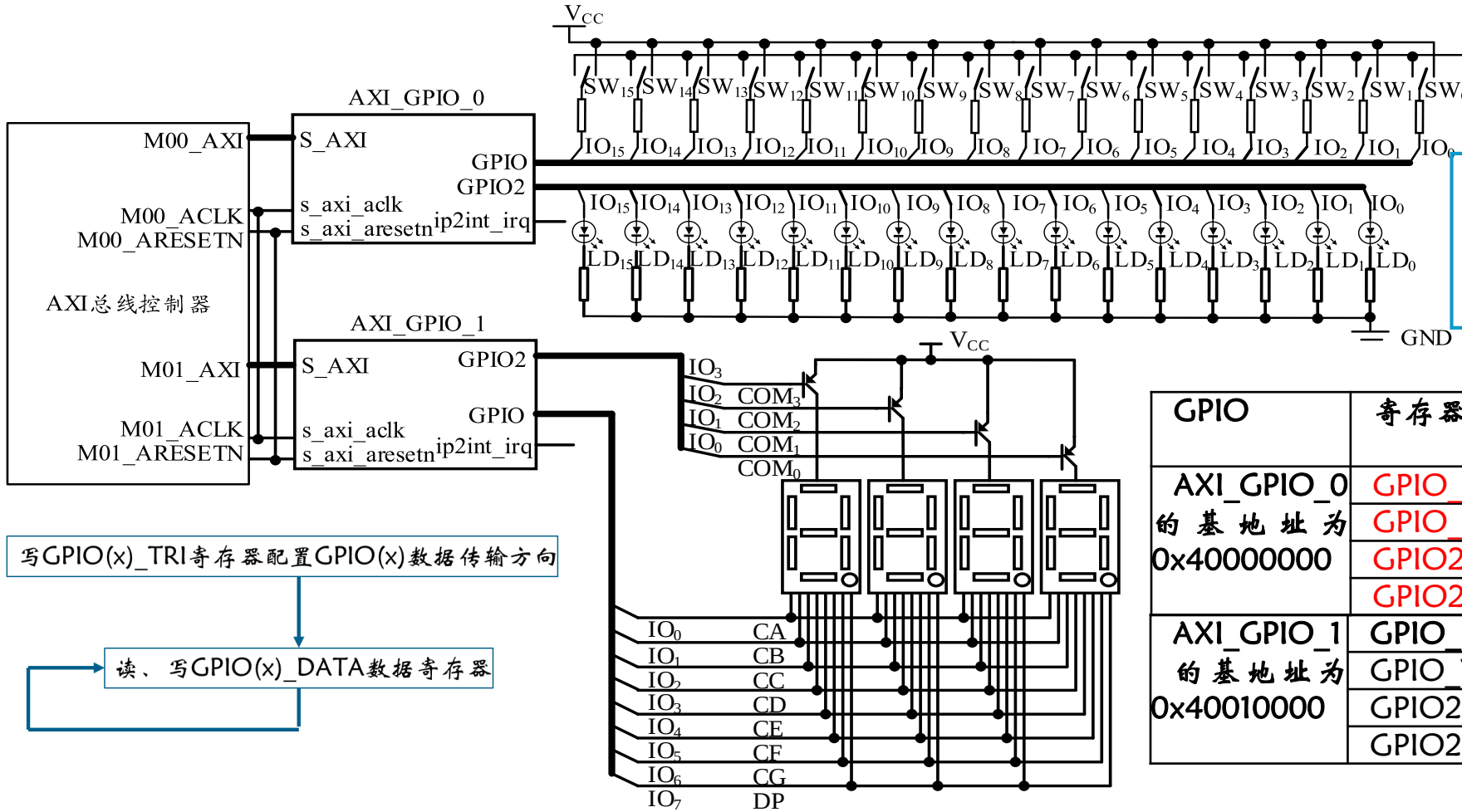
```
Xil_Out16(0x4000000c,0x0);
Xil_Out16(0x40000008,0x5555);
```

GPIO	寄存器	地址	有效数据位	读写控制
AXI_GPIO_0的基地址为0x40000000	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的基地址为0x40010000	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

7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，控制16个开关、16个按键，四位数码管



开关状态实时反应到对应LED：

```
Xil_Out16(0x40000004,0xffff);
Xil_Out16(0x4000000c,0x0);
while (1)
Xil_Out16(0x40000008,Xil_In16(0x40000000));
```

GPIO	寄存器	地址	有效数据位	读写控制
AXI_GPIO_0的基地址为0x40000000	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的基地址为0x40010000	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

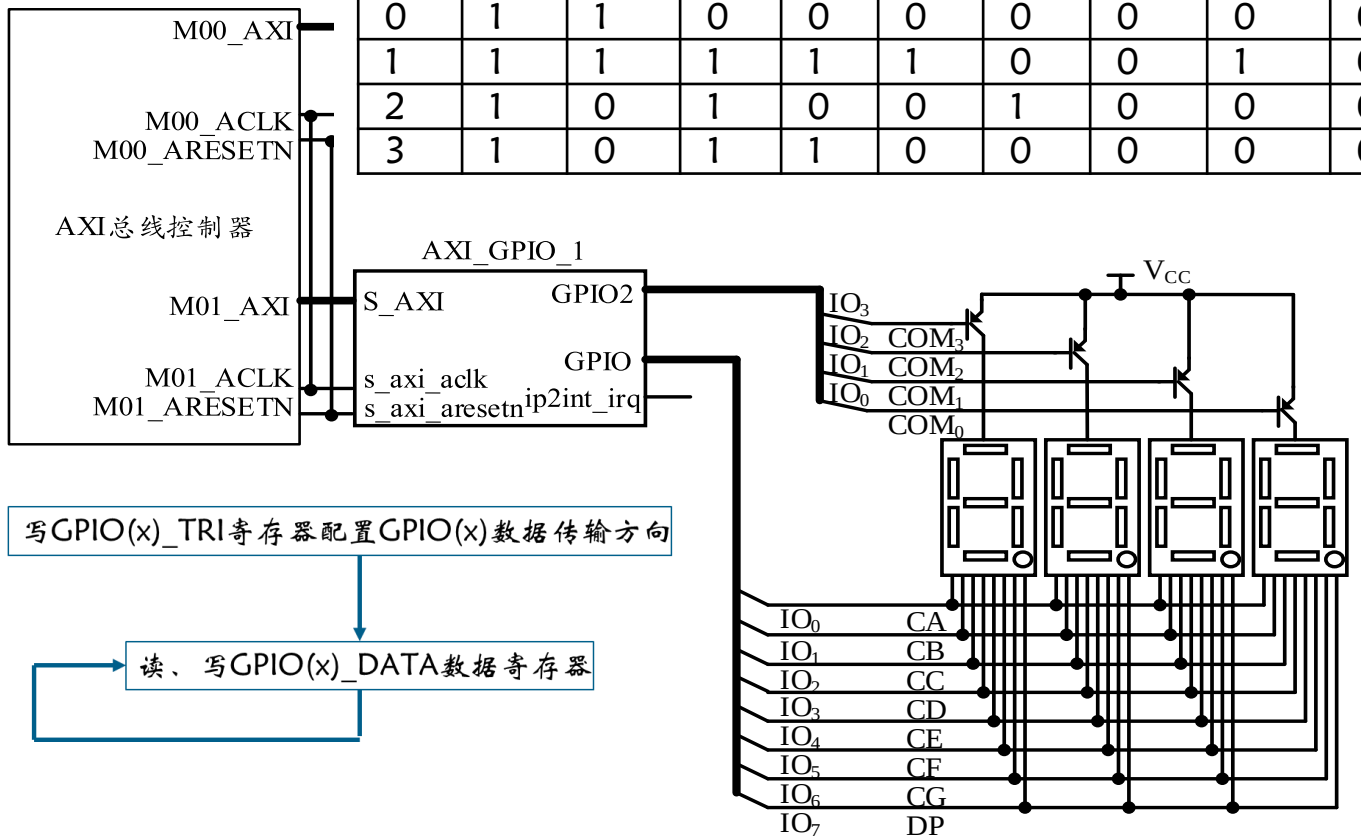
7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，控制四位数码管显示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

位选信号	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



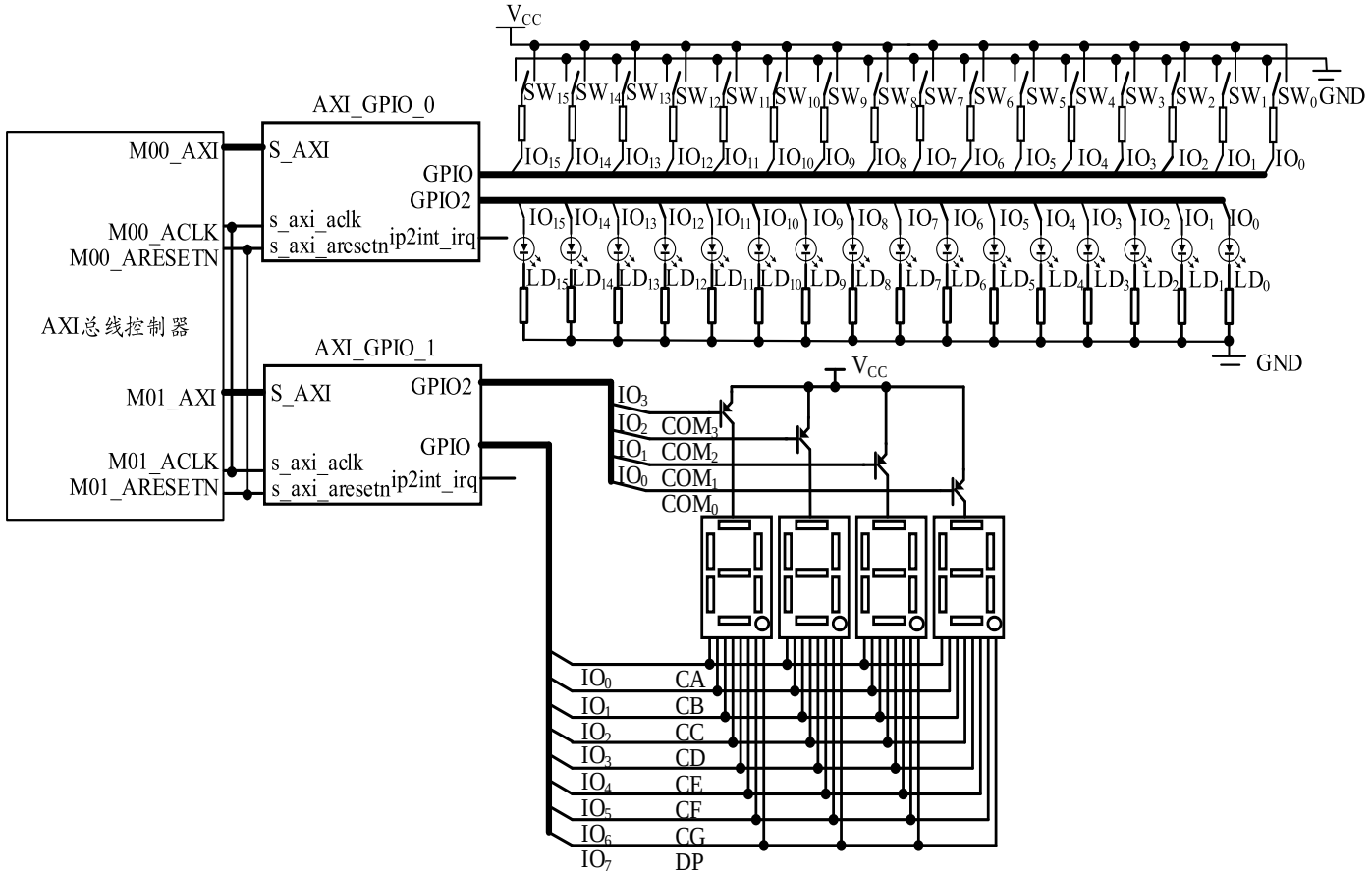
四位数码管显示0-3：

```
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;
}
```

7.5 并行IO接口设计

▶ GPIO控制器

- 【例7.12】采用GPIO控制器，同一控制程序将SW15~0状态实时反应到对应LED LD15~0上，同时将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




```
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
   }
}
```

▶ GPIO控制器：总结

- GPIO控制器可以解决微处理器通过AXI总线与简单输入输出设备之间的通信问题，如前面讲述的
  - 独立开关、
  - 独立发光二极管、
  - 矩阵式键盘、
  - 7段数码管

  以及ADC1210与微处理器之间的通信。
- 这些设备内部没有寄存器，不需要读写控制信号，数据、控制以及状态信息相对独立，完全可以通过GPIO通道的IO引脚进行通信。

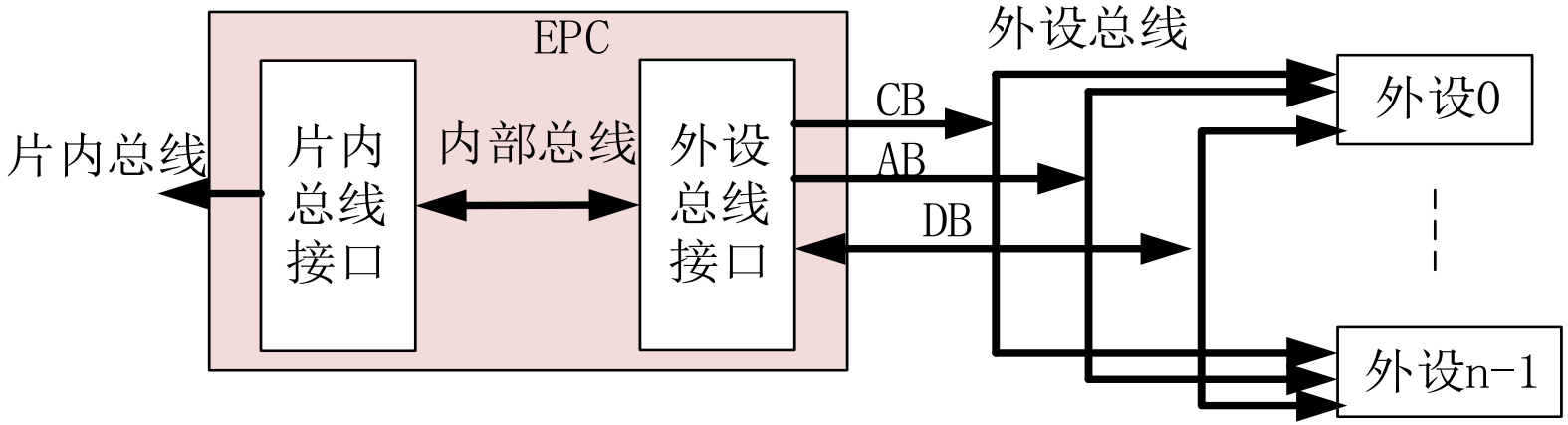
- 另一类有数据线、地址线和控制线的外设，如 网卡、USB协议转换器等设备，不能使用GPIO控制器与微处理器连接，需要使用外设控制器。

7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

▶ 外设控制器（EPC）

- 外设控制器（EPC，External Peripheral Controller）是将片内总线转换为外设外部并行总线信号的一种接口控制器。
- 作用：
  - 实现总线信号转换，
  - 实现高位地址译码，产生外设片选信号。



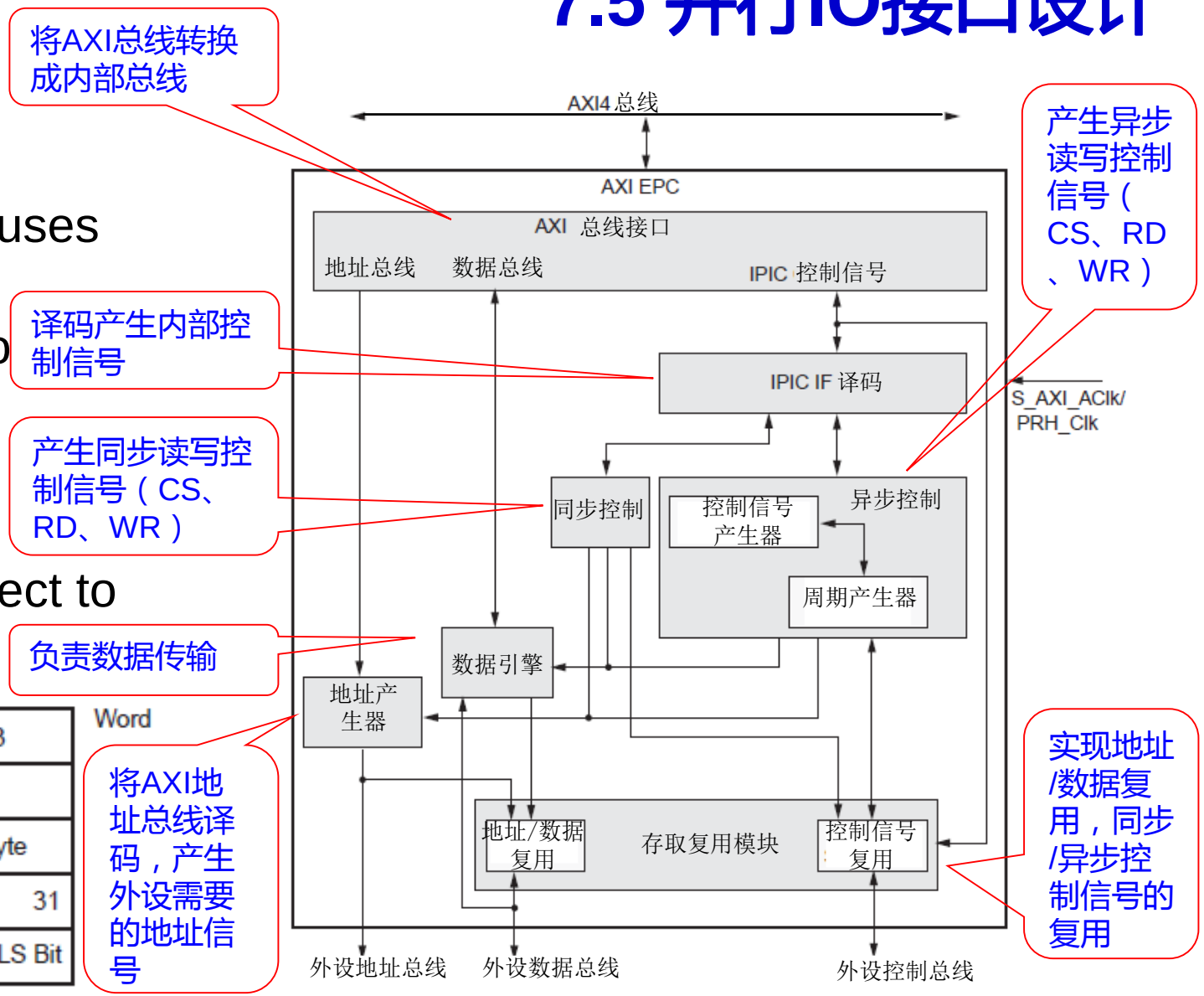
7.5 并行IO接口设计

▶ 外设控制器（EPC）

- Xilinx AXI总线EPC
  - peripheral data bus (PRH_Data) uses big endian bit labelling
  - the 8-bit device should connect to PRH_Data[0 : 7],
  - the 16-bit wide peripheral should connect to PRH_Data[0 : 15]
  - the 32-bit peripheral should connect to PRH_Data [0 : 31].

Byte address	n	n+1	n+2	n+3
Byte label	0	1	2	3
Byte significance	MS Byte			LS Byte
Bit label	0			31
Bit significance	MS Bit			LS Bit

Word



- 外设控制器（EPC）
  - Xilinx AXI总线EPC
    - 外设总线信号类型以及含义

并不是所有外设总线都需要所有类型的信号，用户根据不同的外设总线信号类型，选择合适的信号连接。

信号类型	信号含义	I/O	初始值
PRH_Clk	外部时钟输入	I	-
PRH_Rst	外部复位	I	-
PRH_CS_n[0:外设数目- 1]	外设片选，低电平有效	O	1
PRH_Addr[0:外设地址总线最大宽度 - 1]	外设低位地址	O	-
PRH_ADS	地址锁存使能，高电平有效（地址/数据复用）	O	0
PRH_BE[0:外设数据总线最大宽度/8 - 1]	字节使能，低电平有效	O	1
PRH_RNW	读写复用控制（同步控制）	O	-
PRH_Rd_n	读控制，低电平有效	O	1
PRH_Wr_n	写控制，低电平有效	O	1
PRH_Burst	突发控制，高电平有效	O	0
PRH_Rdy[0:外设数目- 1]	准备就绪，高电平有效	I	-
PRH_Data_I[0:外设数据总线最大宽度- 1]	数据输入	I	-
PRH_Data_O[0:外设数据总线最大宽度- 1]	数据输出	O	-
PRH_Data_T[0:外设数据总线最大宽度- 1]	数据输出三态控制	O	-

- 外设控制器（EPC）
  - Xilinx AXI总线EPC
    - 所有外设总线参数

参数名称	含义	可能值	初始值
C_NUM_PERIPHERALS	外设数目	1-4	1
C_PRH_MAX_AWIDTH	最大地址总线宽度	3-32	32
C_PRH_MAX_DWIDTH	最大数据总线宽度	8，16，32	32
C_PRH_MAX_ADWIDTH	最大地址数据复用总线宽度	8-32	32
C_PRH_CLK_SUPPORT	外部时钟支持	1：外部时钟 0：AXI总线时钟	
C_PRH_BURST_SUPPORT	是否支持burst	0	0

▶ 外设控制器（EPC）

- Xilinx AXI总线EPC
  - 各个不同外设总线参数

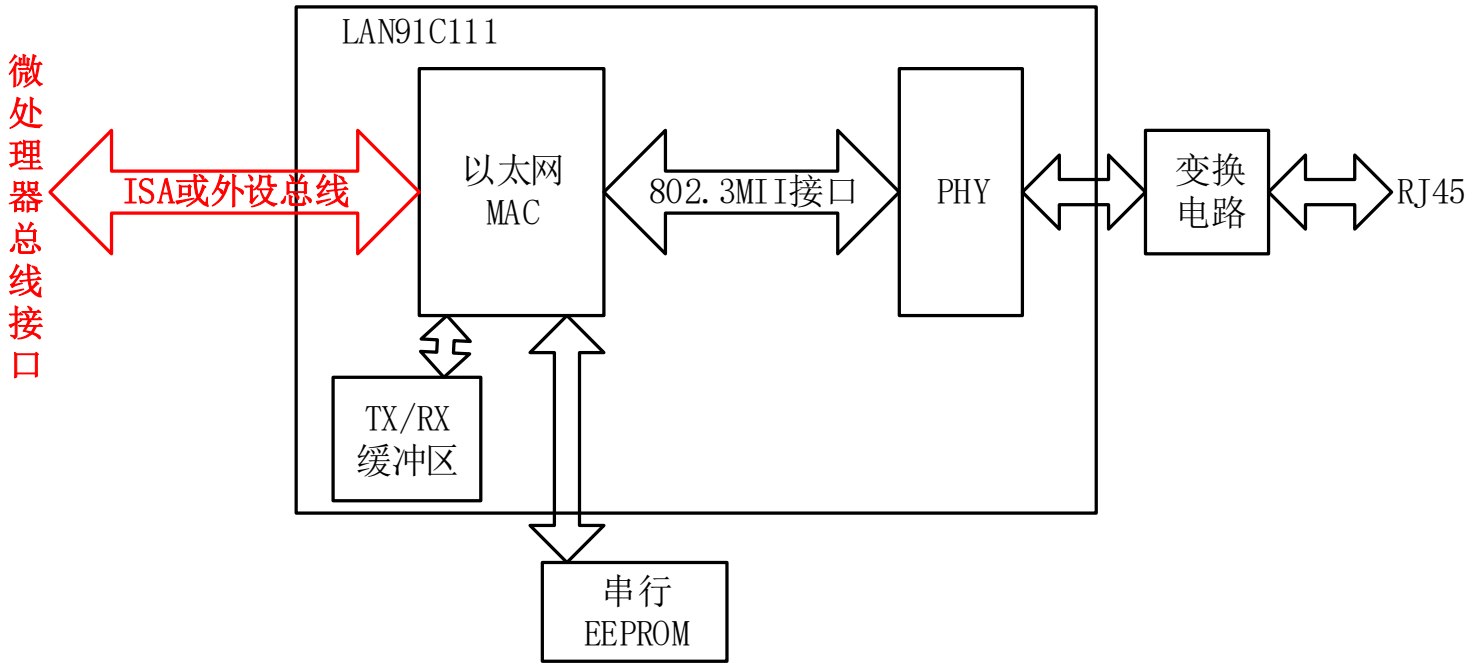
参数名称	含义	可能值	初始值
C_PRHx_AWIDTH	地址总线宽度	3-32	32
C_PRHx_DWIDTH	数据总线宽度	8，16，32	32
C_PRHx_ADWIDTH	地址数据复用总线宽度	8-32	32
C_PRHx_CLK_SUPPORT	外部时钟支持	1：外部时钟0：AXI总线时钟	
C_PRHx_BURST_SUPPORT	是否支持burst	0	0
C_PRHx_BASEADDR	外设最低地址	-	-
C_PRHx_HIGHADDR	外设最高地址	-	-
C_PRHx_FIFO_ACCESS	外设是否支持FIFO	0：没有FIFO1：有FIFO	0
C_PRHx_FIFO_OFFSET	FIFO相对于最低地址的偏移	-	0
C_PRHx_DWIDTH_MATCH	是否支持数据宽度匹配AXI总线数据宽度	0：不支持 1：支持	0
C_PRHx_SYNC	是否是同步总线	0：异步1：同步	0
C_PRHx_BUS_MULTIPLEX	是否地址数据复用	0：不复用1：复用	0

▶ 外设控制器（EPC）

- Xilinx AXI总线EPC
  - 时序参数

参数名称	含义	值
C_PRHx_ADDR_TSU	相对于ADS上升沿或RD/WR下降沿地址信号的建立时间	ps的整数倍，由用户根据外设总线时序设定
C_PRHx_ADDR_TH	相对于ADS下降沿或RD/WR上升沿地址信号的保持时间	
C_PRHx_ADS_WIDTH	ADS信号的最小脉宽	
C_PRHx_CSN_TSU	相对于RD/WR下降沿片选信号的建立时间	
C_PRHx_CSN_TH	相对于RD/WR上升沿片选信号的保持时间	
C_PRHx_WRN_WIDTH	WR信号的最小脉宽	
C_PRHx_WR_CYCLE	WR信号的最小周期	
C_PRHx_DATA_TSU	相对于WR下降沿数据信号的建立时间	
C_PRHx_DATA_TH	相对于WR上升沿数据信号的建立时间	
C_PRHx_RDN_WIDTH	RD信号的最小脉宽	
C_PRHx_RD_CYCLE	RD信号的最小周期	
C_PRHx_DATA_TOUT	相对于RD下降沿设备输出有效数据信号的时间	
C_PRHx_DATA_TINV	相对于RD上升沿数据总线进入高阻态的时间	
C_PRHx_RDY_TOUT	相对于RD/WR下降沿设备输出ready信号的时间	
C_PRHx_RDY_WIDTH	等待ready信号有效的最长时间	

- 外设控制器（EPC）
  - 以太网接口芯片LAN91C111接口设计
    - 已知一采用小字节序的非PCI 接口支持MAC和PHY单一以太网接口芯片LAN91C111，支持8位、16位、32位的主机接口，内部具有8K FIFO存储体供发送和接收数据缓冲。

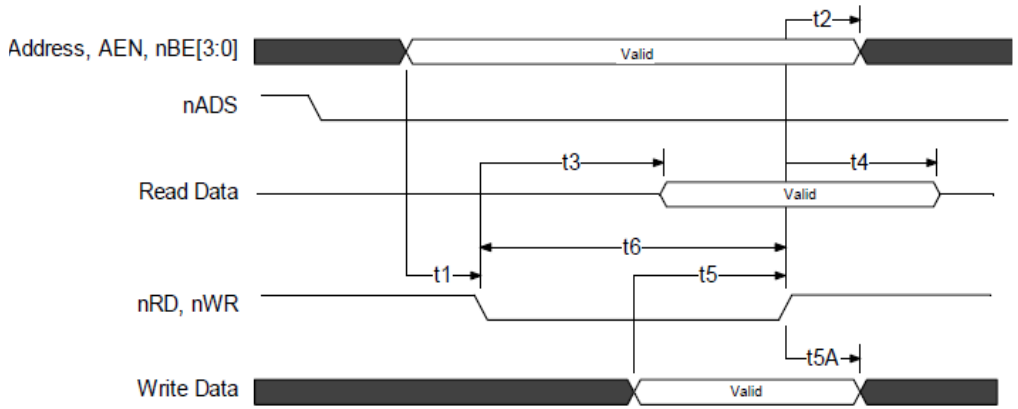


- 外设控制器（EPC）
  - 以太网接口芯片LAN91C111接口设计
    - LAN91C111总线引脚名称及含义

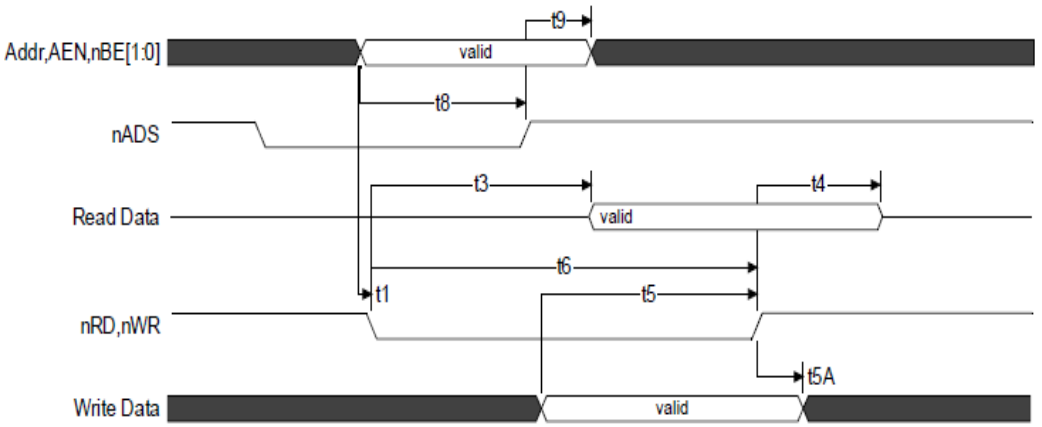
引脚类型	名称	功能	IO
地址总线	A1-A15	地址总线，用于译码选择内部寄存器	I
	AEN	地址译码使能，低电平有效	I
	nBE0-nBE3	字节使能，低电平有效	I
数据总线	D0-D31	数据总线	IO
控制总线	RESET	外部复位信号，高电平有效	I
	nADS	地址数据解复用，上升沿锁存地址信号	I
	LCLK	同步时钟输入	I
	ARDY	异步就绪，漏极开路输出，高电平有效	OD
	nRDYRTN	同步读结束	I
	nSRDY	同步就绪	O
	INTR0	中断请求	O
	nLDEV	本设备选中	O
	nRD	异步读信号	I
	nWR	异步写信号	I
	nDATACS	数据通路选择信号	I
	nCYCLE	同步突发模式周期	I
	W/nR	同步写非读，高电平表示写，低电平表示读	I
	nVLBUS	VL 总线配置	I

- 外设控制器（EPC）
  - 以太网接口芯片LAN91C111接口设计
    - 异步读写时序

名称	含义	最小值	最大值	单位
t1	地址建立到RD/WR信号下降沿的时间	2		ns
t2	RD/WR信号上升沿到地址保持有效的时间	5		
t3	RD/WR信号下降沿到读数据信号有效的时间		15	
t4	RD/WR信号上升沿到读数据保持有效的时间	2	15	
t5	写数据建立到RD/WR信号上升沿的时间	10		
t5A	RD/WR信号上升沿到写数据保持有效的时间	5		
t6	RD/WR信号脉宽	15		
t8	地址信号建立到ADS信号上升沿的时间	8		
t9	ADS信号上升沿到地址信号保持有效的时间	5		
t26	ARDY信号低电平脉宽	100	150	
t26A	控制信号有效到ARDY低电平建立的时间		10	
t13	数据有效到ARDY上升沿的时间	10		



（a）地址数据总线独立的异步读写时序



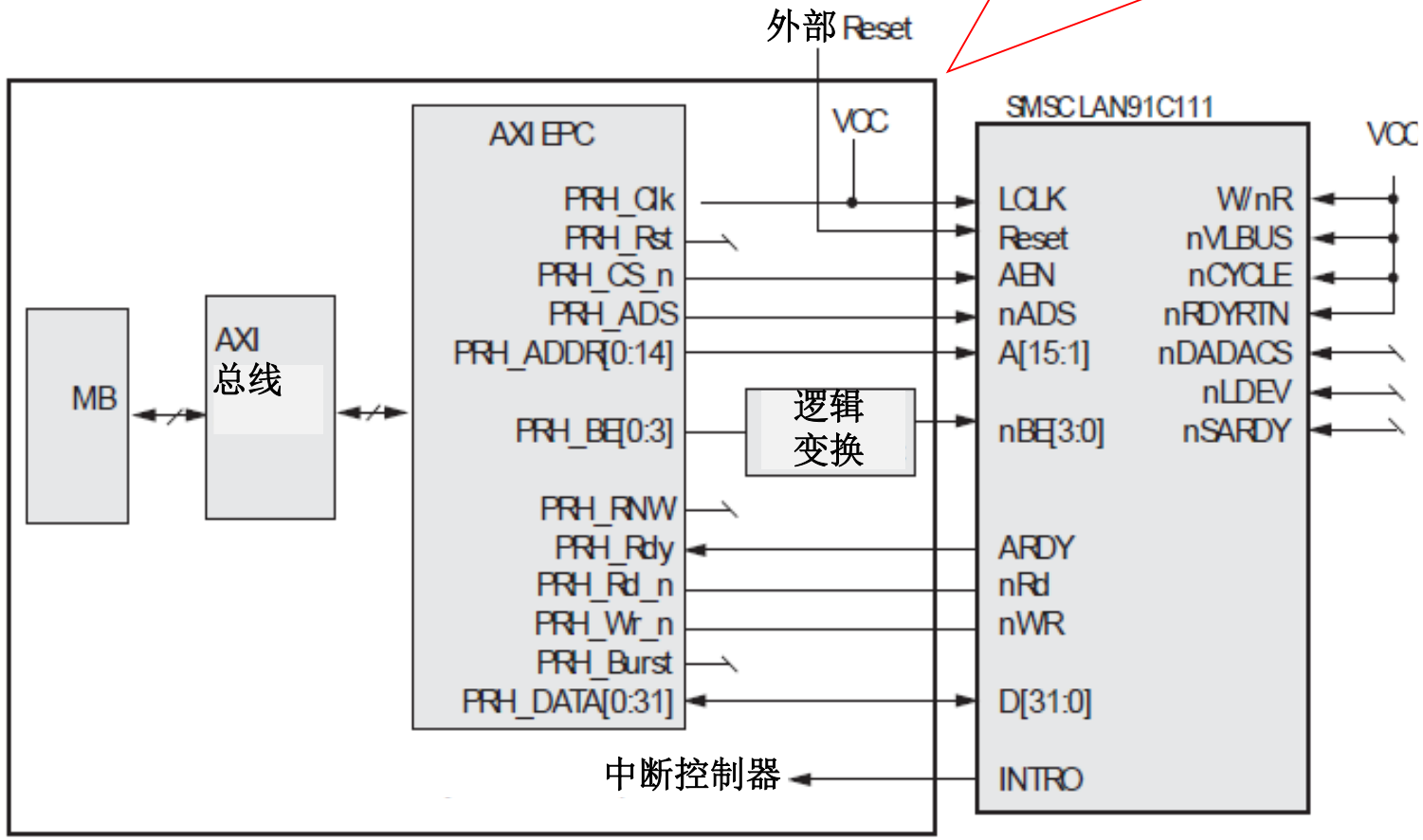
（b）地址数据总线复用的异步读写时序

▶ 外设控制器（EPC）

- 外设控制器与芯片LAN91C111的异步接口电路
  - 外设控制器总线参数设置
    - C_PRH_CLK_SUPPORT = 0
    - C_PRH0_AWIDTH = 16
    - C_PRH0_DWIDTH = 32
    - C_PRH0_SYNC = 0
    - C_PRH0_BUS_MULTIPLEX = 0
  - 异步时序参数对应关系

外设控制器	LAN91C111	最小值(ns)
C_PRHx_ADDR_TSU	t1	2
C_PRHx_ADDR_TH	t2	5
C_PRHx_ADS_WIDTH	>t8	8
C_PRHx_CSN_TSU	T1	2
C_PRHx_CSN_TH	T2	5
C_PRHx_WRN_WIDTH	T6	15
C_PRHx_WR_CYCLE	>t6+t4+t1	19
C_PRHx_DATA_TSU	T5	10
C_PRHx_DATA_TH	T5A	5
C_PRHx_RDN_WIDTH	t6	15
C_PRHx_RD_CYCLE	>t6+t4+t1	19
C_PRHx_DATA_TOUT	T3	15（最大）
C_PRHx_DATA_TINV	T4	2
C_PRHx_RDY_TOUT	T26A	10
C_PRHx_RDY_WIDTH	T26	100

EPC的总线为大字节序，而LAN91C111采用小字节序，因此多位的总线需要倒序连接。



7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

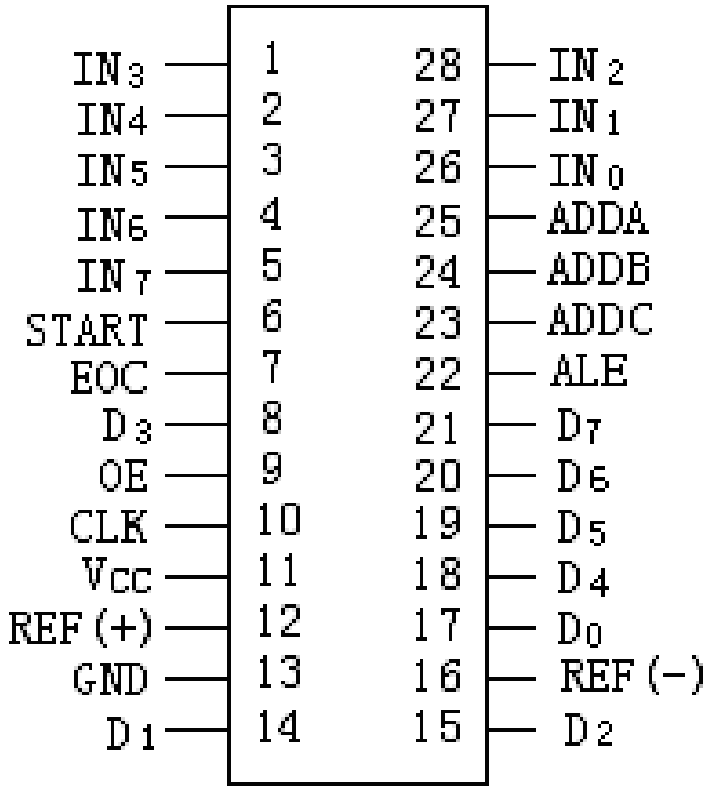
7.5 并行IO接口设计

ADC接口设计

- 所有AD接口设计问题可以归纳为下面两个基本问题：
  - 如何启动一次AD转换；
  - 如何判断一次AD转换已经结束？(如何读取一次AD转换的结果)
    - 无条件（延时）
    - 查询
    - 中断
- ADC接口应该根据具体IC芯片的管脚功能、工作时序灵活设计。
  - ADC0808
  - ADC1210

► A/D芯片ADC0808

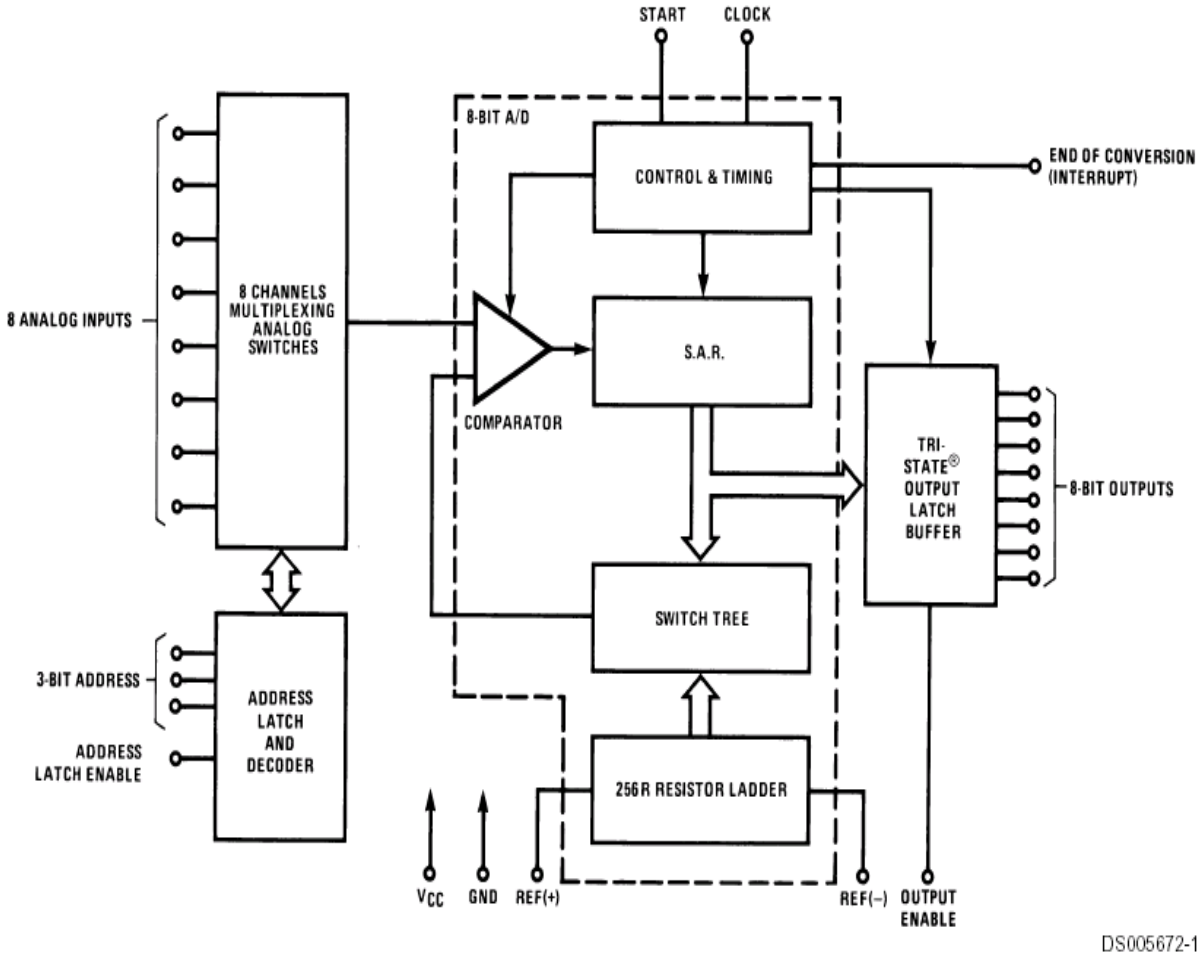
- 管脚和内部结构



- IN7～IN0：8个模拟电压(0～+5V)通道输入;
- ADDC～ADDA：3根模拟通道地址选择控制；
- ALE：地址锁存允许信号，上升沿有效，锁存3根地址线以选通相应模拟输入通道；
- D7～D0：8位数字输出；
- OE(9)：数字量输出使能，高电平有效；
- REF（+），REF（-）：参考电压输入；参考电压的精度直接影响转换结果，要求不高时直接电源；
- CLK→时钟脉冲输入，10/640/1280 KHz
- START→启动信号，正脉冲启动A/D转换（START的上升沿清0内部所有寄存器，下降沿开始A/D转换过程。）
- EOC→转换结束信号，转换开始后，EOC信号变低；转换结束时。EOC返回高电平。

7.5 并行IO接口设计

- A/D芯片ADC0808
  - 管脚和内部结构



- ALE：地址锁存允许信号，上升沿有效，锁存3根地址线以选通相应模拟输入通道；
- START→启动信号，（START的上升沿清0内部所有寄存器，下降沿开始A/D转换过程。）
- EOC→转换结束信号，转换开始后变低；转换结束时返回高电平。
- OE(9):数字量输出使能，高电平有效；

- A/D芯片ADC0808
  - 通道选择
    - ADDC ~ ADDA：3根模拟通道地址选择控制；
    - ALE：地址锁存允许信号，上升沿有效，锁存3根地址线以选通相应模拟输入通道；

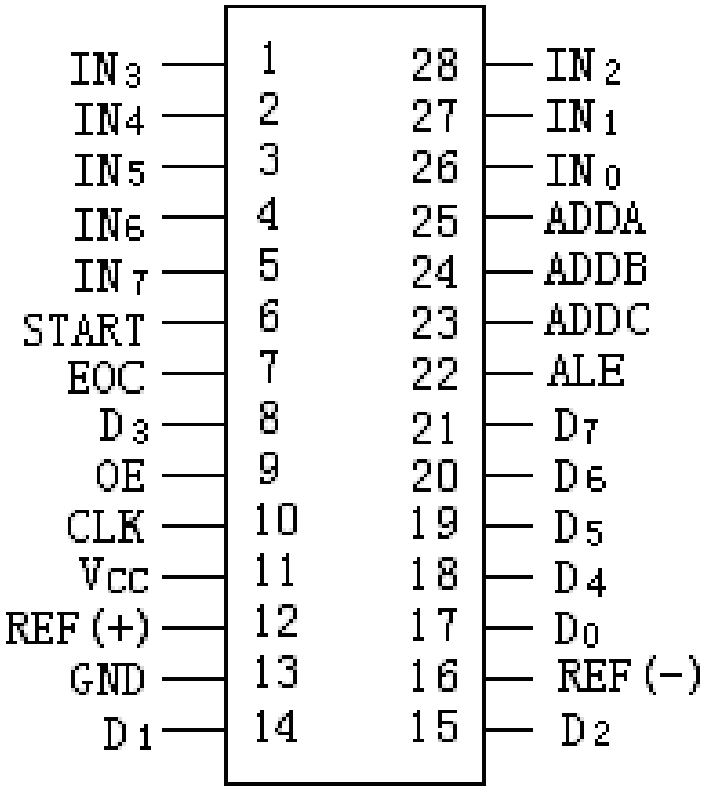
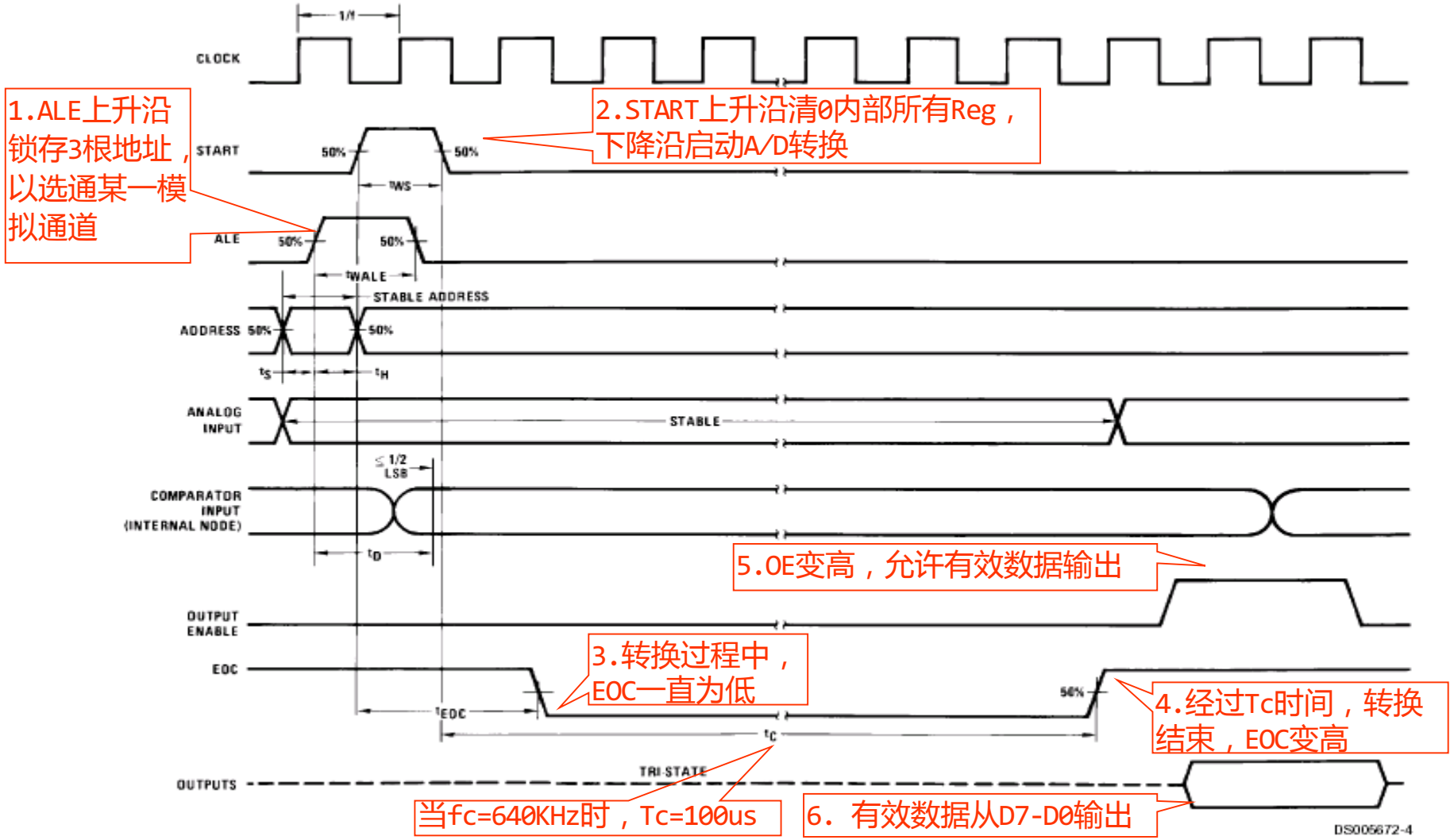


TABLE 1.

SELECTED ANALOG CHANNEL	ADDRESS LINE		
	C	B	A
IN0	L	L	L
IN1	L	L	H
IN2	L	H	L
IN3	L	H	H
IN4	H	L	L
IN5	H	L	H
IN6	H	H	L
IN7	H	H	H

7.5 并行IO接口设计

- A/D芯片ADC0808
  - 工作时序



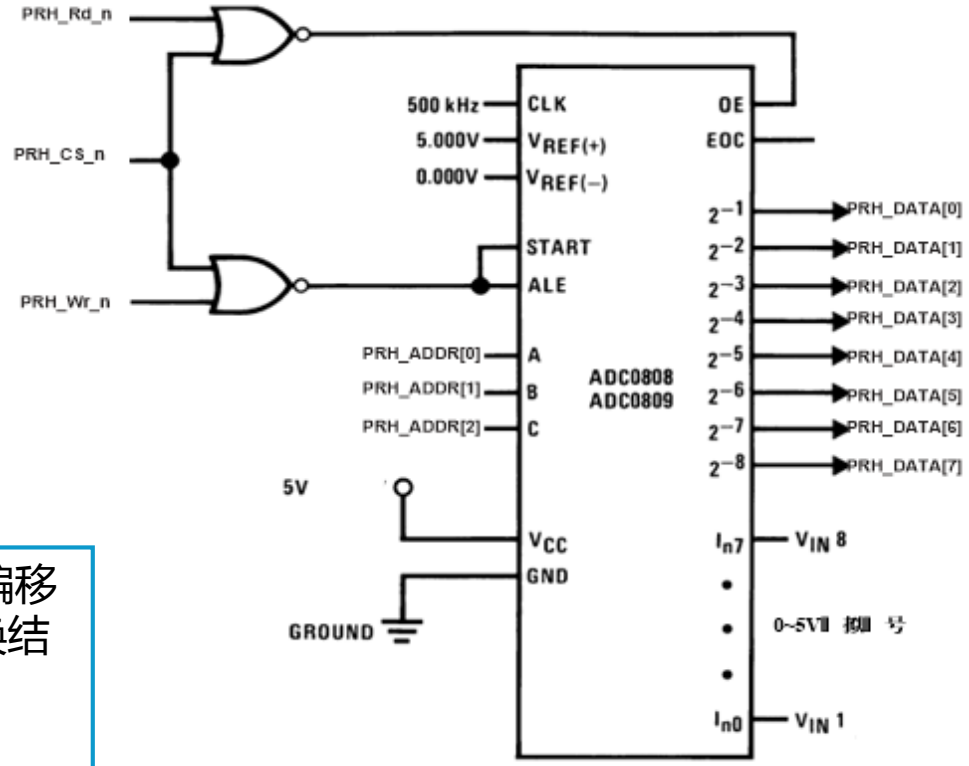
▶ A/D芯片ADC0808

- 时序
  - ADC0808时序参数

参数名称	含义	最小值	典型值	最大值
tWS	START信号脉宽		100ns	200 ns
tWALE	ALE信号脉宽		100 ns	200 ns
tS	地址信号有效到ALE信号上升沿的时间		25 ns	50 ns
tH	ALE上升沿之后地址信号维持有效的时间		25 ns	50 ns
tc	START信号下降沿到EOC信号有效的时间	90μs	100μs	116 μs
tEOC	START信号上升沿到EOC信号失效的时间	0	8/f+2us	
f	时钟频率	10 kHz	640 kHz	1280 kHz
tOH	OE下降沿到数据维持有效的时间		125 ns	250 ns
tHO	OE上升沿到数据输出有效的时间		125 ns	250ns

7.5 并行IO接口设计

- A/D芯片ADC0808
  - 延时方式读取AD转换结果
    - 读取AD转换数据的步骤
      - 1）通道选择，并启动AD转换，通过写操作完成，由地址译码产生通道选择信号
      - 2）延时等待AD转换结束
      - 3）读取AD转换结果，通过读操作完成，所有通道都是同一个端口地址。



假设外设控制器通道0的基地址为0x43E00000，那么通道1的偏移地址则为0x43E00001，因此通过延时方式读取通道1的AD转换结果的程序段

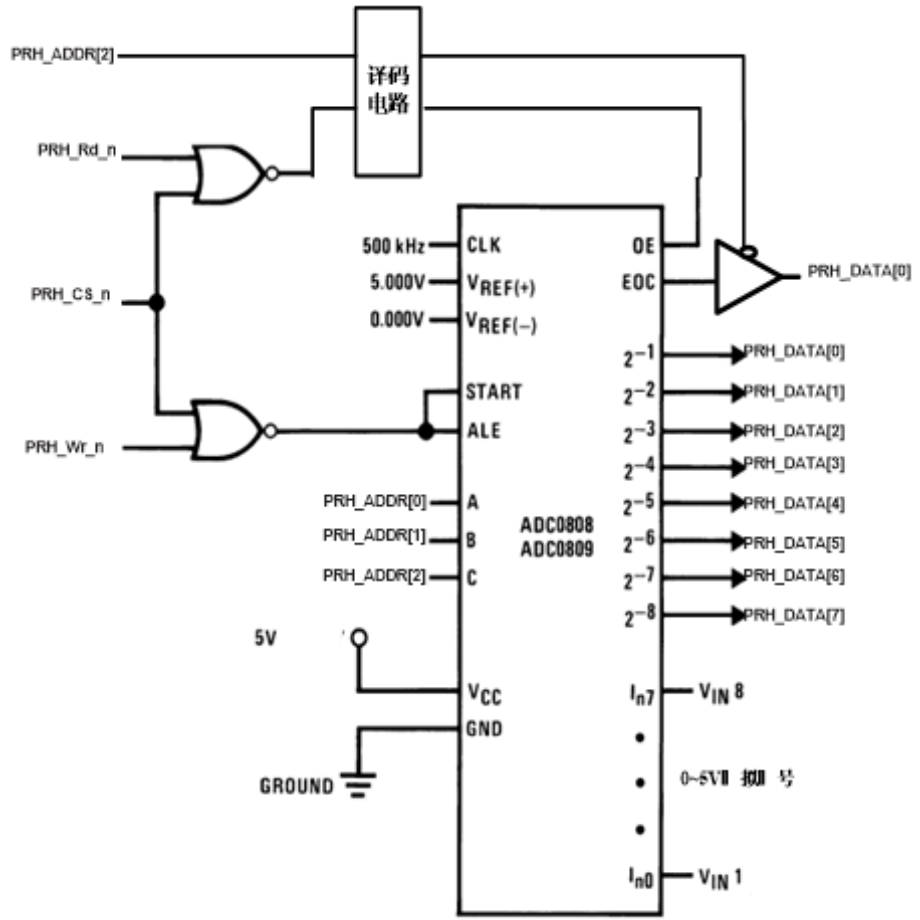
```
int j;
unsigned char data;
//启动通道1转换
Xil_Out8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01,0x0);
for(j=0;j<10000;j++);      //延时等待，需要保证延时 > 100us
//读取转换结果
data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x00);
```

如果需要对通道3进行AD转换，该如何修改代码？

- C_PRH_CLK_SUPPORT = 0
- C_PRH0_AWIDTH = 3
- C_PRH0_DWIDTH = 8
- C_PRH0_SYNC = 0
- C_PRH0_BUS_MULTIPLEX = 0

- A/D芯片ADC0808
  - 查询方式读取AD转换结果
    - 读取AD转换数据的步骤
      - 1）通道选择，并启动AD转换，通过写操作完成，由地址译码产生通道选择信号
      - 2）读取状态端口数据，并判断EOC是否为1，EOC=1进入3），否则继续步骤2）；
      - 3）读取AD转换结果，通过读数据端口操作完成，所有通道都是同一个数据端口地址。

假设译码电路：当PRH_ADDR[2]=0时，数据口OE高电平使能，当PRH_ADDR[2]=1时，读EOC的状态口低电平使能

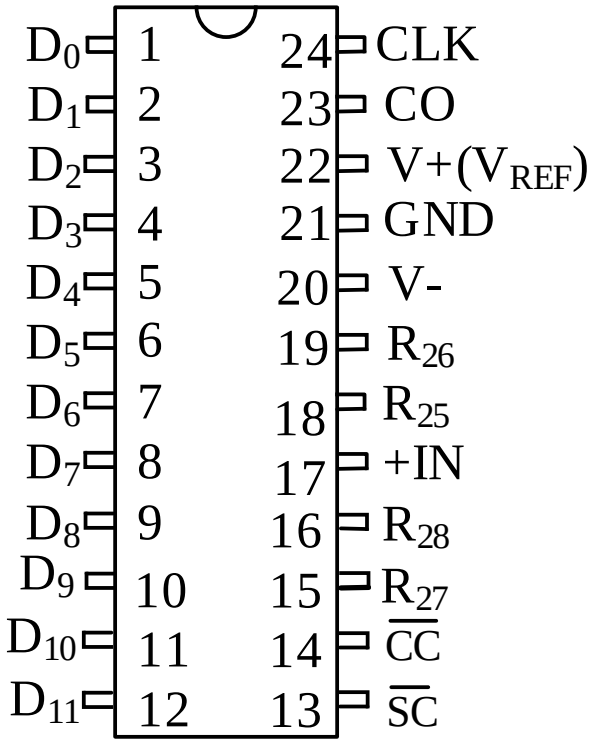


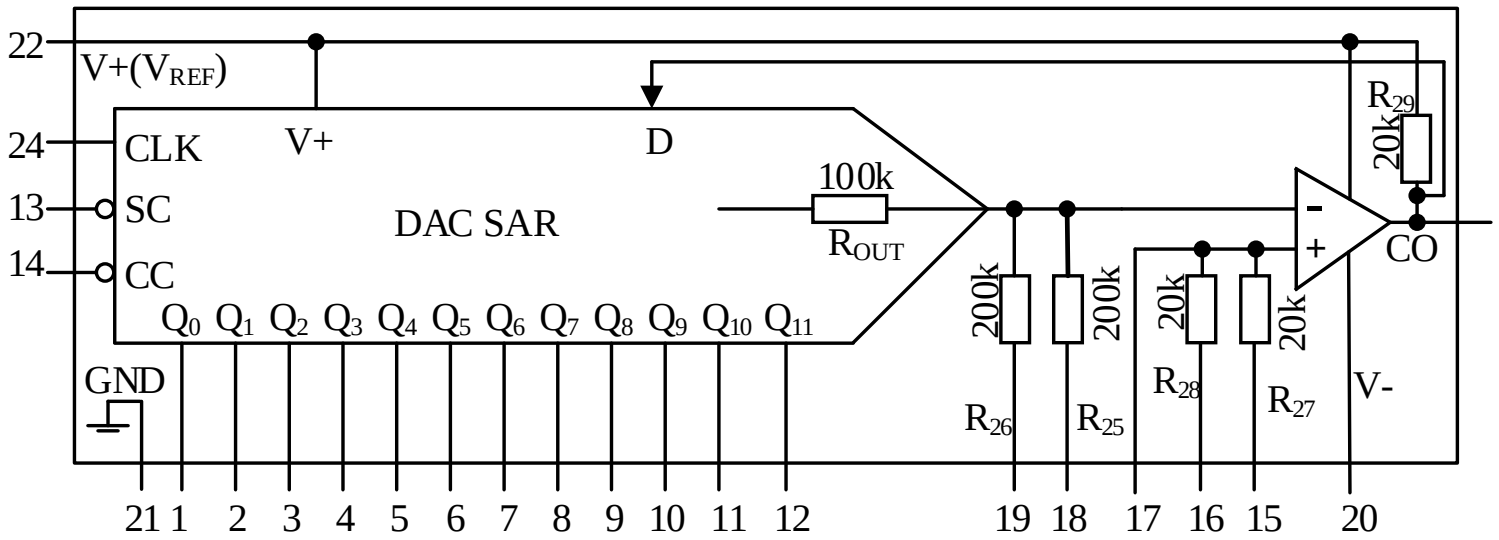

```
unsigned char data;
//启动通道1转换
Xil_Out8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01,0x0);

// 查询状态口
Do
   data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01);
while((data&0x1)!=0x1);

// 从数据端口读数据
data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x00);
```

7.5 并行IO接口设计

- 独立开关输入接口
- 发光二极管输出接口
- 七段数码管动态显示接口
- LED点阵
- 矩阵式键盘接口
- GPIO控制器
- 外设控制器（EPC）
- AD转换器
  - ADC0808接口
  - ADC1210接口

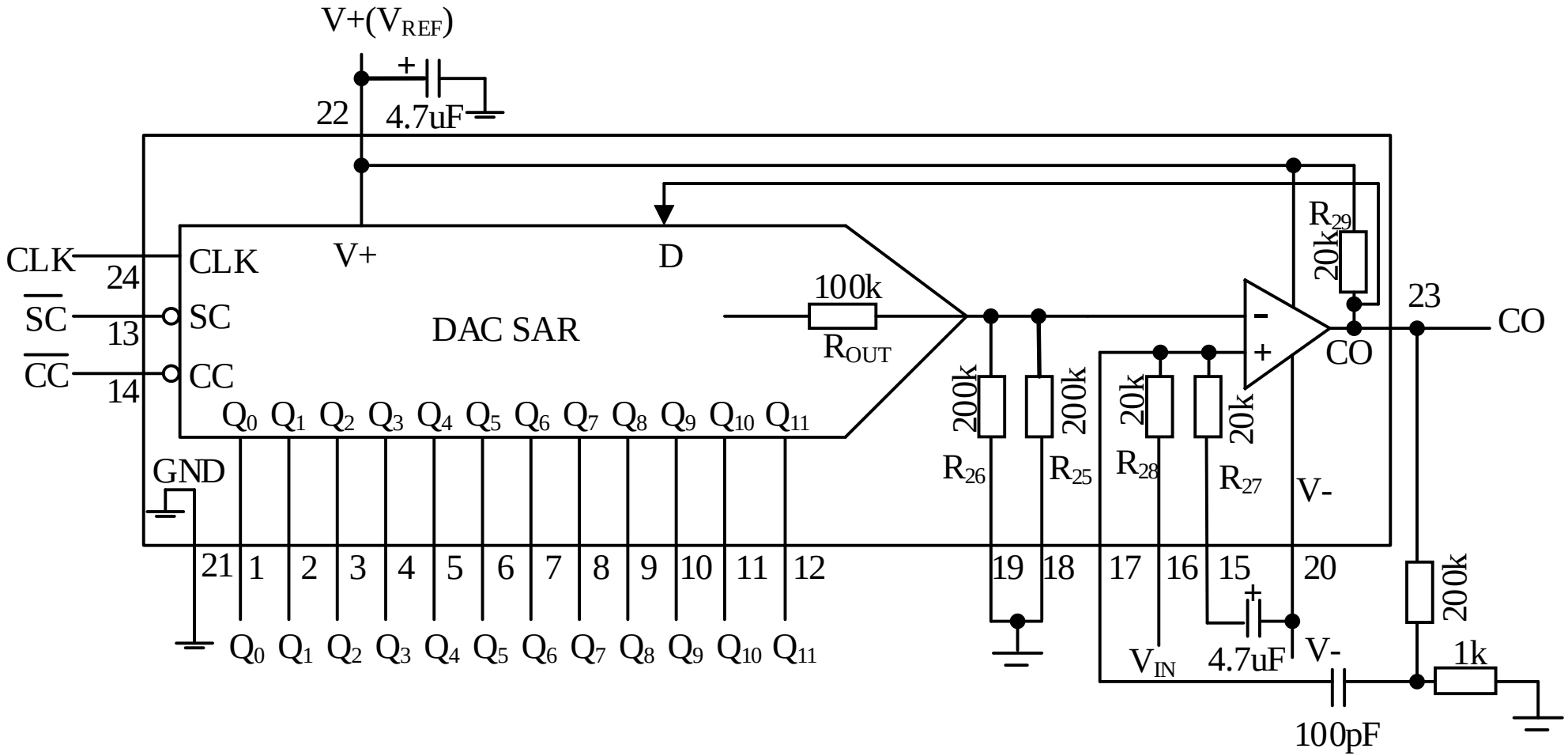
- AD转换器ADC1210接口
  - AD：模拟数字转换
  - ADC：模数转换器，模拟信号接入计算机系统处理必须先用ADC，转为数字信号
  - ADC1210管脚和功能





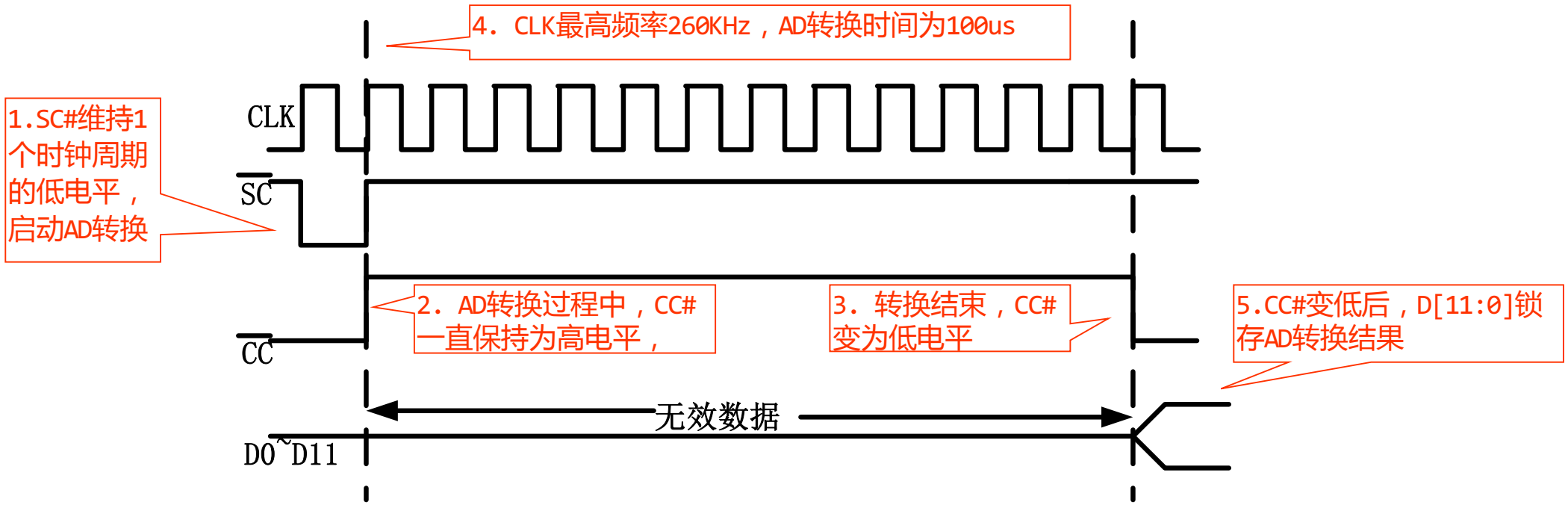
▶ AD转换器ADC1210接口

- 单极性模拟电压输入电路连接



▶ AD转换器ADC1210接口

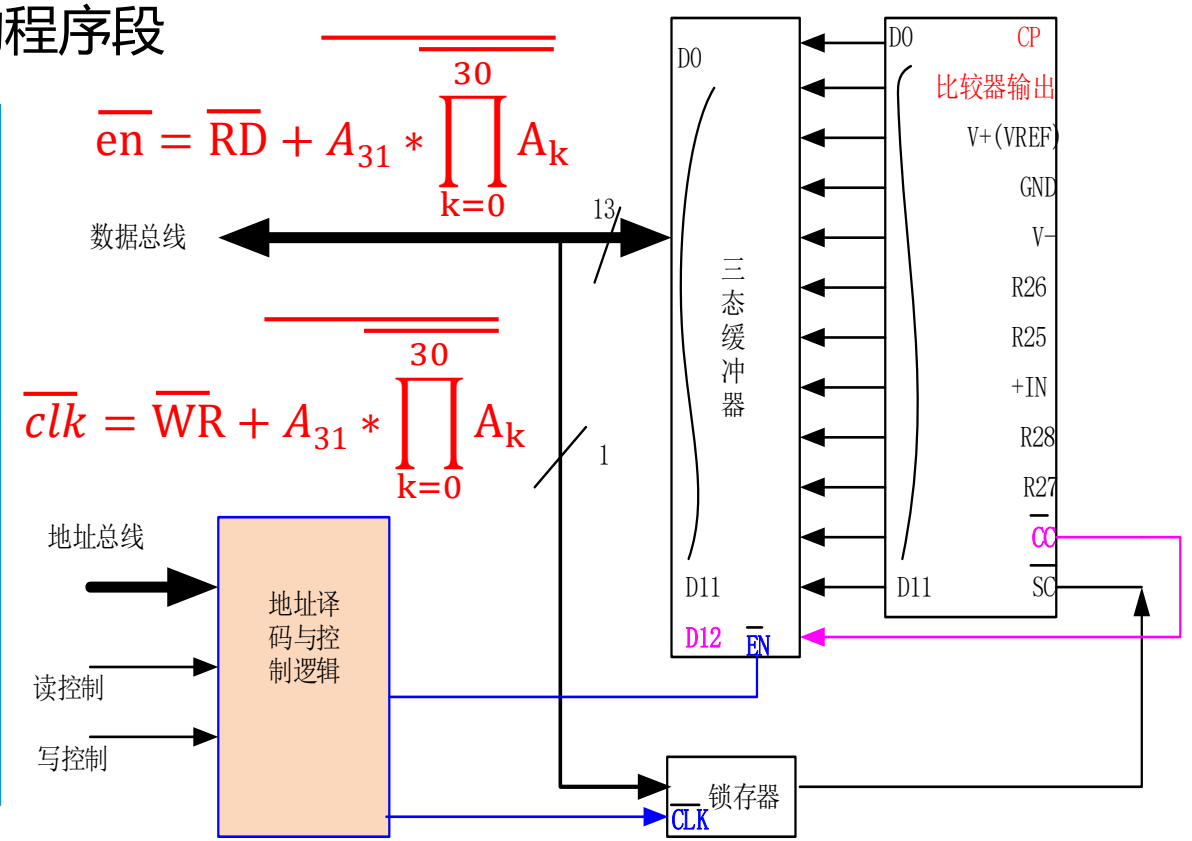
- 时序



▶ AD转换器ADC1210接口

- ADC1210单极性模拟电压输入电路连接
  - 延时方式设计ADC1210与32位MicroBlaze微处理器的接口电路原理图，要求该接口电路仅占据一个IO端口地址，且其地址为0x80000000。并基于Xilinx C语言编写控制该接口电路转换100个数据，且将转换结果保存在字类型数组volt中的程序段

```
int volt[100]，i;
int port = 0x80000000;
for(i=0; i<100, i++)
{
    Xil_Out8(port,0x01);    //输出数据使得(SC)为高电平
    Xil_Out8(port,0x00);    //输出数据使得(SC)为低电平
    Delay_10us;             //延时一个ADC1210时钟周期
    Xil_Out8(port,0x01);    //输出数据使得(SC)为高电平，
                            //从而产生启动转换信号
    Delay_120us;            //延时>100us
    Volt[i]=Xil_In16(port)&0x0fff;  //读取转换结果，且仅保
                                    //留低12位有效数据
}
```

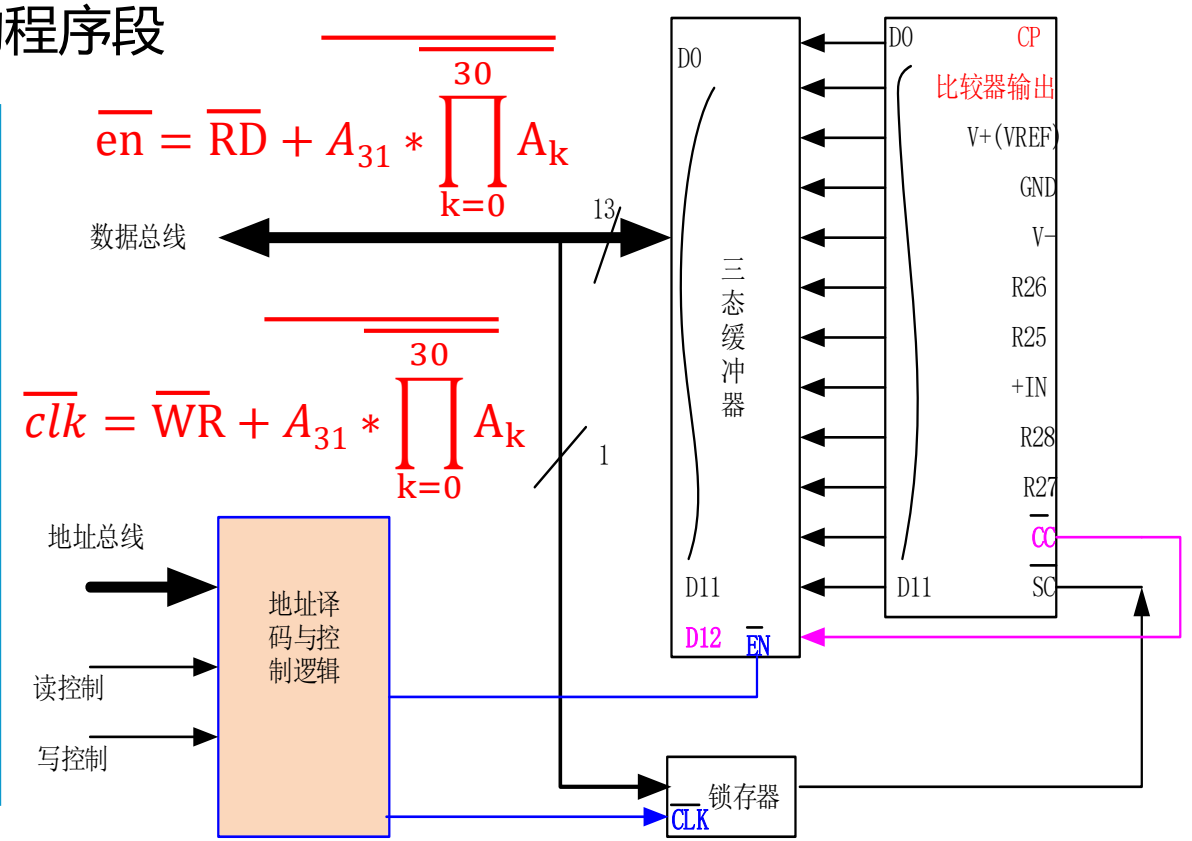


7.5 并行IO接口设计

▶ AD转换器ADC1210接口

- ADC1210单极性模拟电压输入电路连接
  - 查询方式设计ADC1210与32位MicroBlaze微处理器的接口电路原理图，要求该接口电路仅占据一个IO端口地址，且其地址为0x80000000。并基于Xilinx C语言编写控制该接口电路转换100个数据，且将转换结果保存在字类型数组volt中的程序段

```
int volt[100]，i;
int port = 0x80000000;
for(i=0; i<100, i++)
{
    Xil_Out8(port,0x01);    //输出数据使得(SC)为高电平
    Xil_Out8(port,0x00);    //输出数据使得(SC)为低电平
    Delay_10us;             //延时一个ADC1210时钟周期
    Xil_Out8(port,0x01);    //输出数据使得(SC)为高电平，
                            //从而产生启动转换信号
    While ((Xil_In16(port)&0x1000)!=0);    //查询状态
    Volt[i]=Xil_In16(port)&0x0fff;  //读取转换结果，且仅保
                                    //留低12位有效数据
}
```



第7章 作业（一）

作业题

- 12.GPIO控制器，要求采用Nexys4板实验验证
  - 8.采用某8位独立开关输入十六进制字符（0~9，a~f）的ASCII码，并将该ASCII码表示的十六进制字符通过一位7段数码管显示出来。
  - 9.设计一监视2台设备状态(switch代替）的接口电路和监控程序：若发现某一设备状态异常(由低电平变为高电平)，则发出报警信号(指示灯亮)，一旦状态恢复正常，则将其报警信号撤除。
  - 10.用8个理想开关输入二进制数，8只发光二极管显示二进制数。设输入的二进制数为原码，输出的二进制数为补码。
  - 11.用4个7段数码管显示数字1，2，3，4，且仅占用两个端口地址。

要求

- 做好设计，写好程序
- 微助教提交，下周5前
- 实验课上验证、验收

Thanks