第六章 接口技术

学习目标

- 了解接口的基本构成、数据传送方式以及控制方式
- 理解不同接口寻址方式的特点
- 理解接口译码原理、掌握接口译码电路的设计
- 掌握简单存储器接口设计
- 掌握基于存储控制器的存储器接口设计
- 理解简单并行IO接口设计原理
- 掌握独立开关、LED、7段数码管、矩阵式键盘以及并行AD转换器接口设计
- 掌握基于GPIO控制器、外设控制器的接口设计

接口的基本概念

- 接口是一组电路，是中央处理器与存储器、输入输出设备等外设之间协调动作的控制电路。
- 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

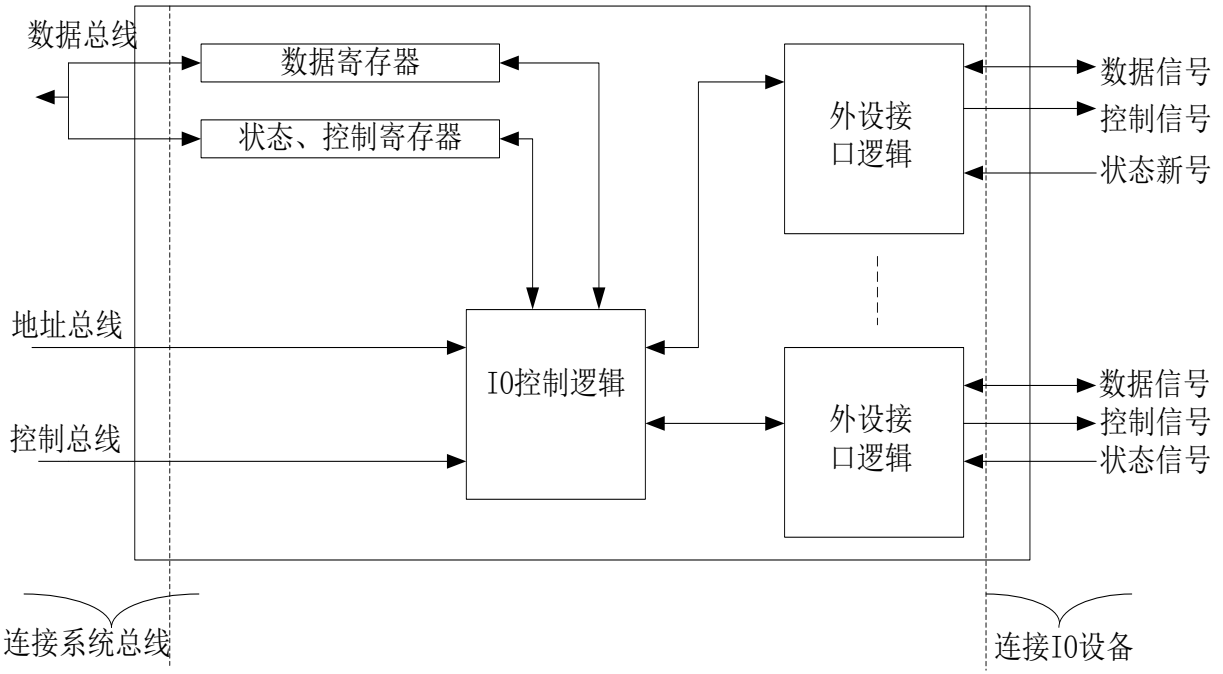
接口功能

- 任何接口通常都包括以下5种功能：
  - 控制和定时
  - 与微处理器通信
  - 与外设通信
  - 数据缓冲
  - 错误检测

- 微处理器与IO接口之间通信以及IO设备与IO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

接口基本结构

- 控制逻辑电路
- 状态设置和存储电路
- 数据存储和缓冲电



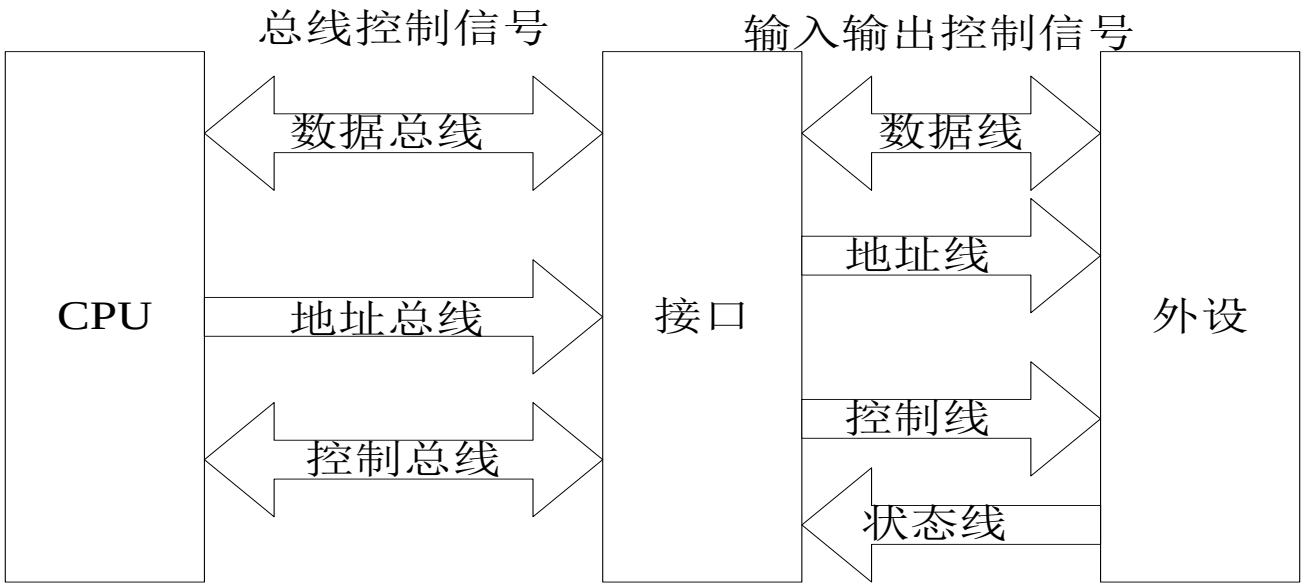
接口与外设间的数据传送方式

- 并行数据传送
  - 并行数据的每一位都对应独立的传输线路
- 串行数据传送
  - 将构成字符的每个二进制数据位，按一定的顺序逐位进行传送

接口控制方式

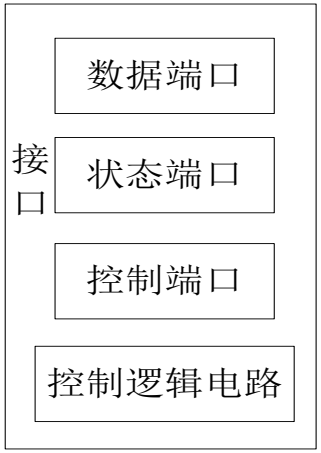
- 查询
  - 在数据传送之前通过接口的状态设置存储电路询问外设，待外设允许传送数据后才传送数据的操作方式
  - “无条件”传送方式
- 中断
  - 外设要与中央处理器传送数据时，外设向中央处理器发出请求，中央处理器响应后再传送数据
- DMA
  - 数据不经过中央处理器在存储器和外设之间直接传送

接口信号



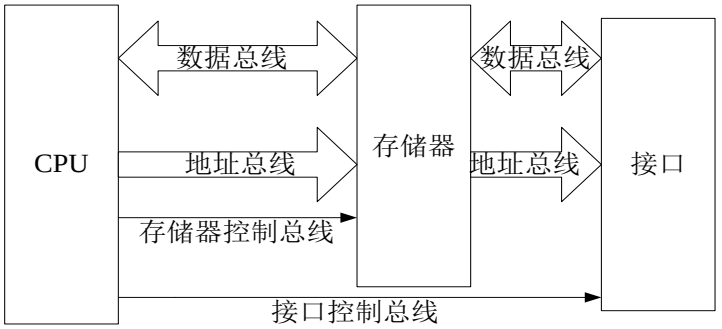
IO接口寻址方式

- 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、状态端口和数据端口

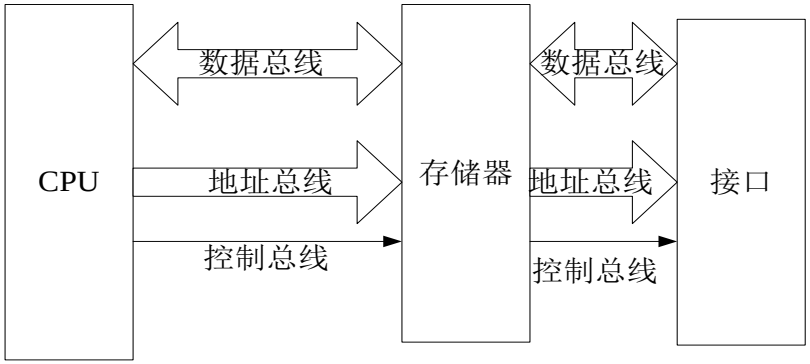


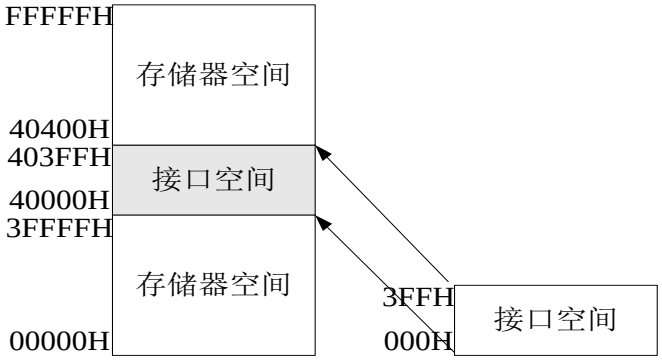
IO接口结构

- 标准（独立）I/O接口结构计算机系统



- 存储器映像I/O接口结构计算机系统





标准I/O 寻址方式特点

- I/O设备的地址空间和存储器地址空间是独立的、分开的。即I/O接口地址不占用存储器地址空间。
- 微处理器对I/O设备的管理是利用专用的IN(输入)和OUT(输出)指令来实现数据传送的。
- CPU对I/O设备的读/写控制是用I/O读/写控制信号($\overline{IOR}$、$\overline{IOW}$)。
- 微机系统的微处理器都采用标准I/O寻址方式

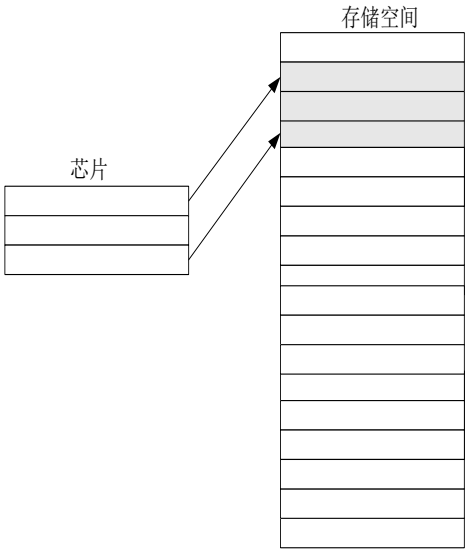
存储器映像I/O寻址方式

- I/O接口与存储器共用同一个地址空间。
  - I/O设备的每一个寄存器占用存储器空间的一个地址。
  - 存储器与I/O设备之间的唯一区别是其所占用的地址不同。
- CPU利用对存储器存储单元进行操作的指令来实现对I/O设备的管理。
- CPU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。
- 嵌入式微处理器基本上都采用存储器映像IO寻址方式

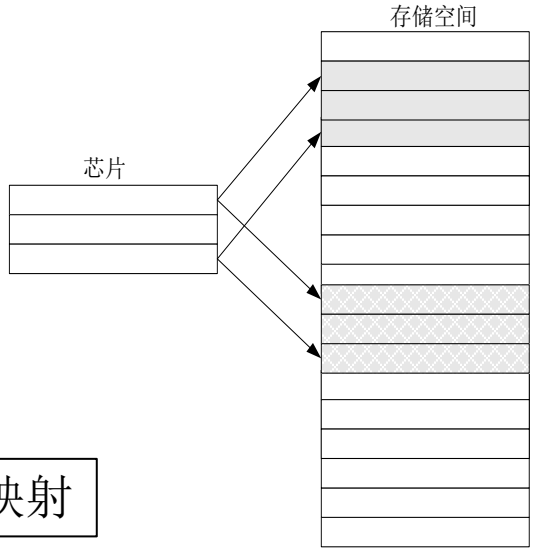
接口译码原理

- 直接译码
  - 直接采用地址总线来实现存储单元选址的译码方式
- 间接译码
  - 利用专门地址端口来实现地址译码

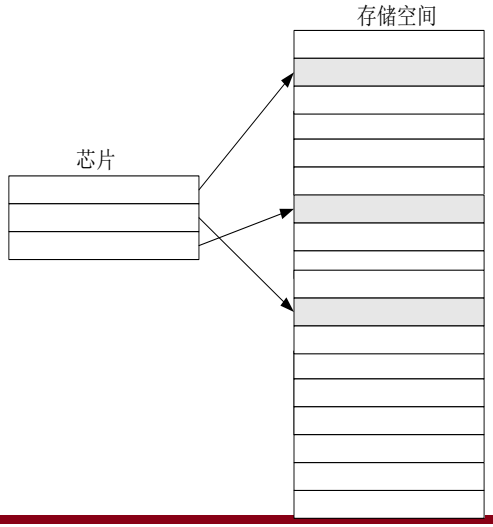
直接译码



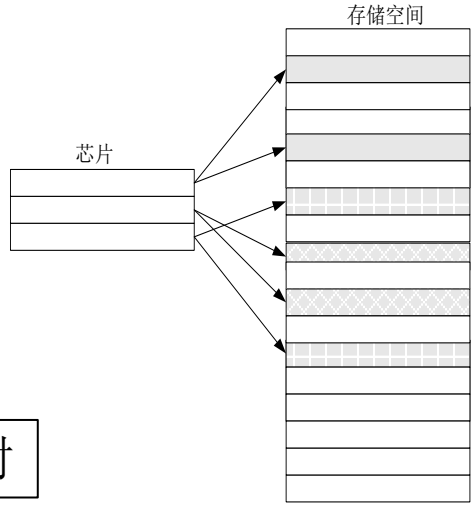
一对一整体映射



一对多整体映射



一对一离散映射



一对多离散映射

一对一整体映射举例

- 系统地址总线所有剩余高位地址都必须经过译码之后才能连接到芯片的片选控制端叫做全译码法。
- 例6. 1 将容量为256K的存储芯片一对一整体映射到存储空间为4G的计算机存储系统中，且要求映射地址范围为0xfff40000~0xfff7ffff，该如何实现地址译码？

- 256K的存储芯片具有18位地址总线：A0~A17。
- 4G存储空间的计算机系统具有32位地址总线：A0~A31。其中A0~A17与存储芯片的18位地址总线A0~A17直接相连。
- 剩余的高位地址线为A18~A31，由于地址范围为0xfff40000~0xfff7ffff。由此可知在整个地址范围内A18为0，A19~A31都为1。因此需要将A18~A31译码产生存储芯片片选信号，假设片选信号为CS，且低电平有效，那么
- $\overline{CS} = A18 + \overline{A19 \cdot A20 \cdot A21 \cdot A22 \cdot A23 \cdot A24 \cdot A25 \cdot A26 \cdot A27 \cdot A28 \cdot A29 \cdot A30 \cdot A31}$

一对多整体映射举例

- 部分高位地址线没有参与译码形成芯片片选控制信号。
  - 仅一位高位地址线参与译码，就叫做线选法；
  - 是多位但不是全部高位地址线参与译码，就叫做部分译码法。

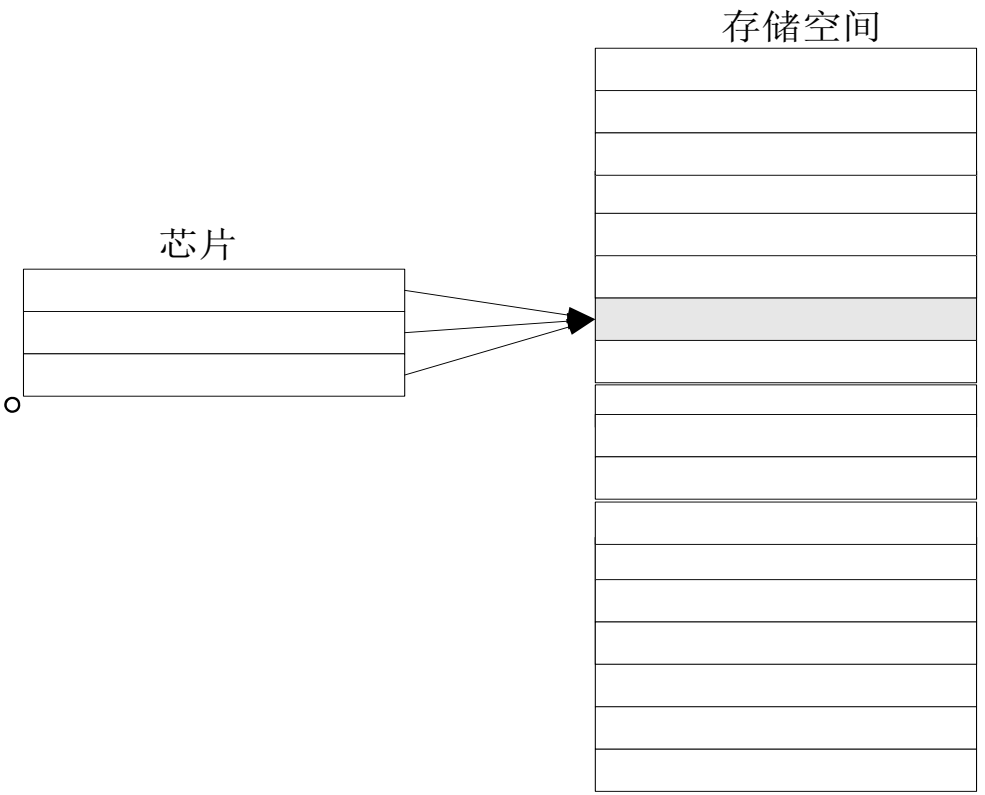
- 例6. 2将容量为256K的存储芯片一对多整体映射到存储空间为4G的计算机存储系统中，且要求地址范围为0xfff00000~0xfff3ffff或0xfff40000~0xfff7ffff，该如何实现地址译码？

- 由于地址范围可以为0xfff00000~0xfff3ffff或0xfff40000~0xfff7ffff，
- 那么A18既可以是0也可以是1，而A19~A31固定为1，因此可以不考虑A18这根地址线，
- 仅用A19~A31译码产生存储芯片的片选信号，假设片选信号为CS，且低电平有效，那么

$$\overline{CS} = \overline{A19 \cdot A20 \cdot A21 \cdot A22 \cdot A23 \cdot A24 \cdot A25 \cdot A26 \cdot A27 \cdot A28 \cdot A29 \cdot A30 \cdot A31}$$

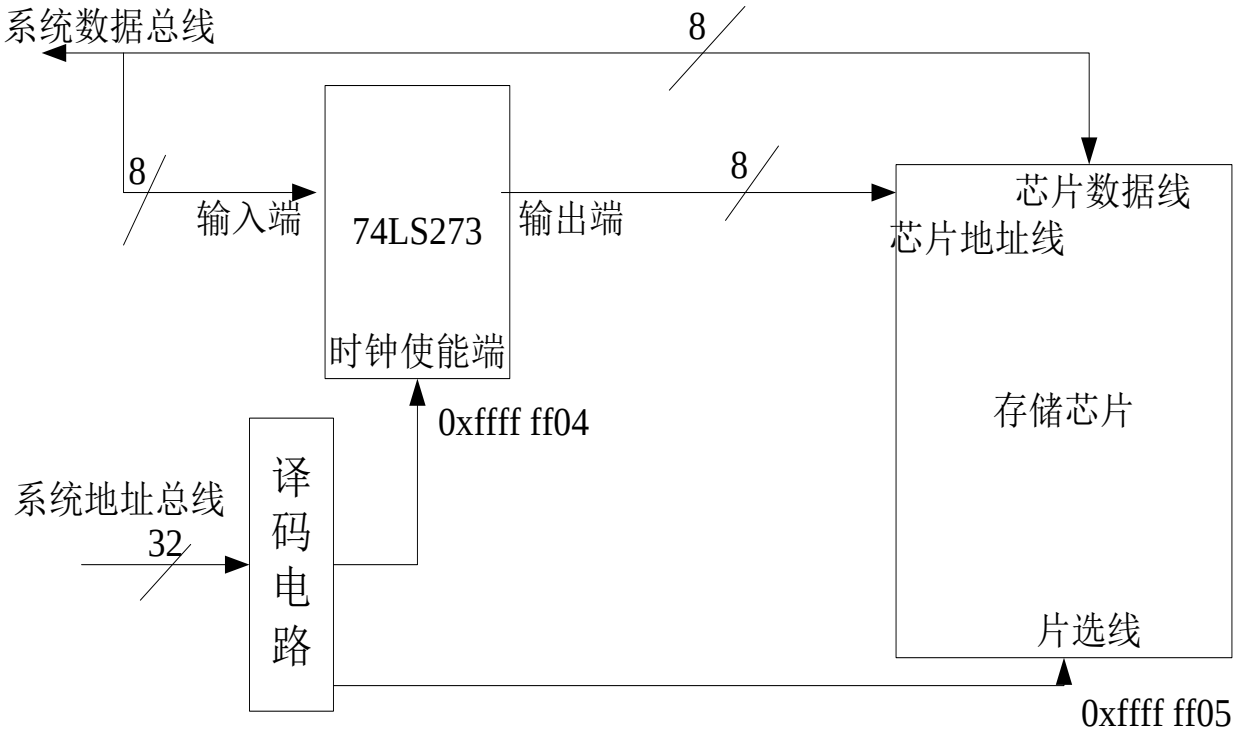
多对一整体映射

- 微处理器给出的一个地址对应了多个物理存储空间，必须采用IO接口来实现芯片内各个存储单元的访问。该类接口包含一个数据端口和一个地址端口



- 例6. 3 一个具有256个存储单元的芯片，要求将其所有的存储单元都映射到具有32位地址总线、32位数据总线的计算机系统存储空间中的一个存储单元中。假设该存储单元地址为0xffff ff05，并要求能对芯片内任意一个存储单元进行访问，试设计接口。

微处理器需要首先提供一个地址信息给存储芯片，以便存储芯片选择具体的存储单元。这个地址不由地址总线提供，而是由一个特定端口的数据信号来提供，这个端口就是接口的地址端口，假设其地址为0xffff ff04。因此在实现多对一的地址映射时需占用两个存储单元地址：数据端口和地址端口。



$$Y = \prod_{k=8}^{31} A_k$$

$$CLK = \overline{Y \cdot \overline{A7} \cdot \overline{A6} \cdot \overline{A5} \cdot \overline{A4} \cdot \overline{A3} \cdot A2 \cdot \overline{A1} \cdot \overline{A0}}$$

$$\overline{CS} = Y \cdot \overline{A7} \cdot \overline{A6} \cdot \overline{A5} \cdot \overline{A4} \cdot \overline{A3} \cdot A2 \cdot \overline{A1} \cdot \overline{A0}$$

采用MIPS汇编指令来实现对存储芯片0x80单元字节类型数据的访问

```
lui $s0,0xffff

ori $t0,$s0,0xff04 #将地址端口的地址0xffff ff04保存到$t0中

addi $s1,$zero,0x80 #将芯片存储单元地址0x80保存到寄存器$s1中

sbu $s1,0($t0) #将0x80输出到地址端口，锁存在74LS273的输出端

ori $t0,$s0,0xff05 #将数据端口的地址0xffff ff05保存到$t0中

lbu $s2,0($t0) #从数据端口中读取无符号字节类型数据
```

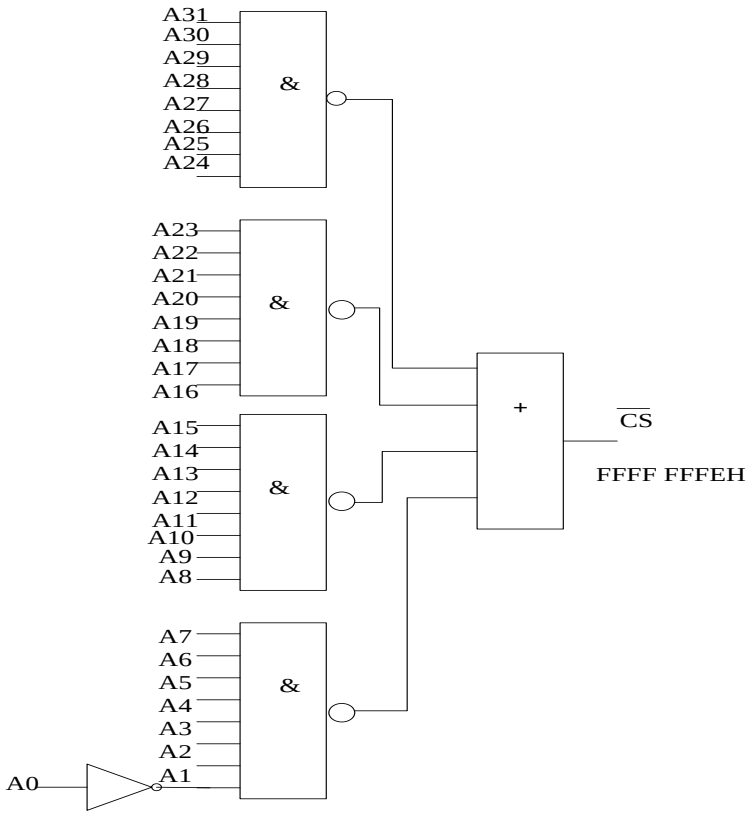
采用Xilinx C语言来实现

```
int AddressPort=0xffffff04;

int DataPort=0xffffff05;

Xil_Out8(AddressPort,0x80);

char data=Xil_In8(DataPort);
```

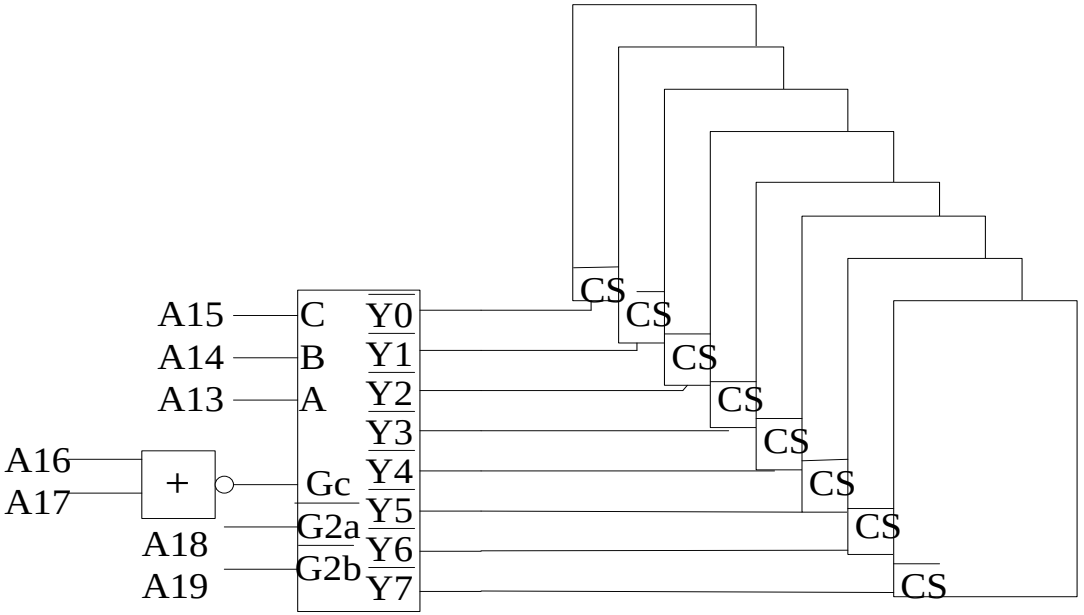
接口译码电路

- 逻辑门电路
  - AND ,NAND ,OR ,NOR….
- 专用译码芯片
  - 138，…
- 可编程逻辑器件
  - CPLD,GAL,FPGA

逻辑门


A31
A30
A29
A28
A27
A26
A25
A24
&
A23
A22
A21
A20
A19
A18
A17
A16
&
A15
A14
A13
A12
A11
A10
A9
A8
&
A7
A6
A5
A4
A3
A2
A1
&
A0
+
CS
FFFF FFFEH

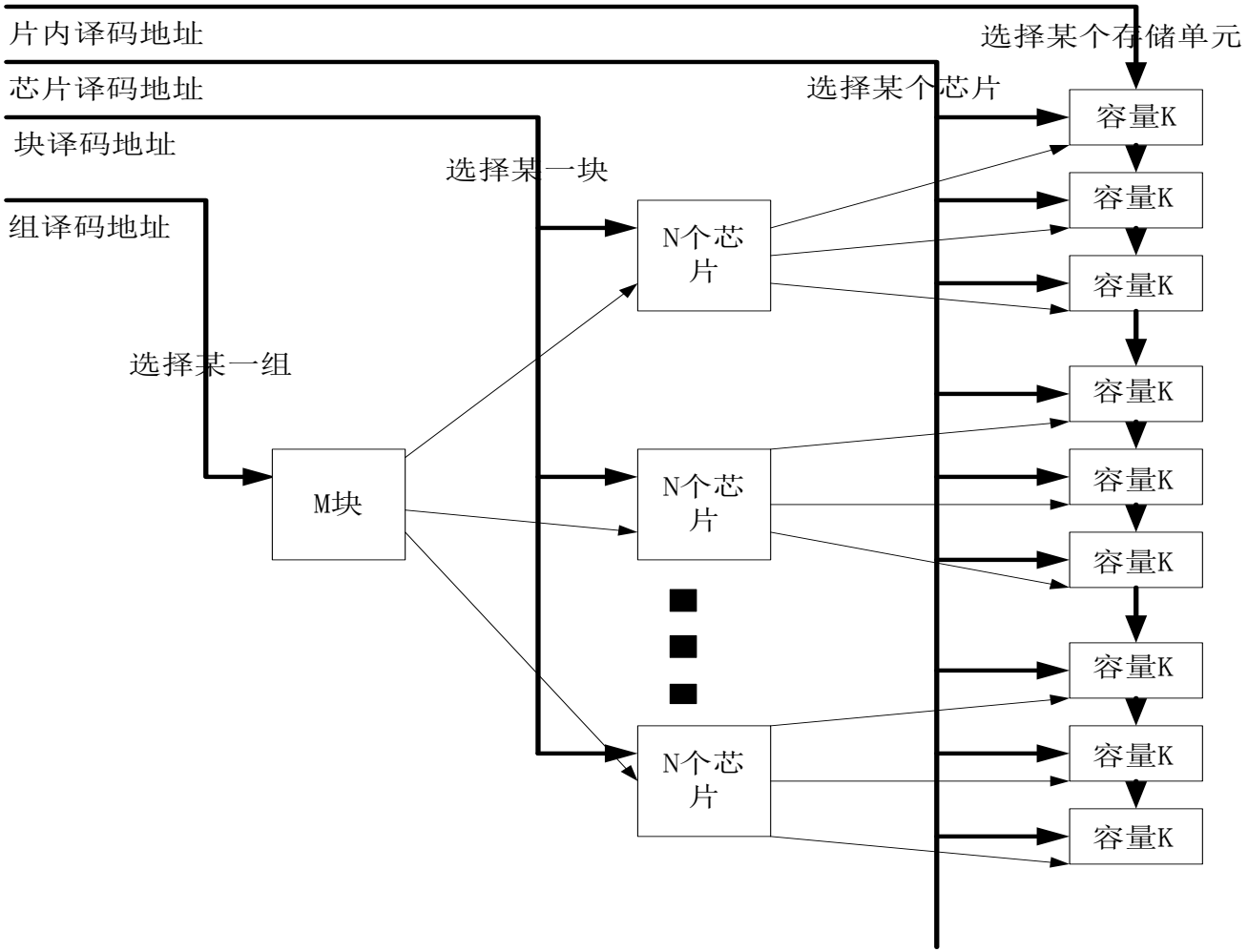
138译码器


A15
A14
A13
A16
A17
A18
A19
C
B
A
+
Gc
G2a
G2b
Y0
Y1
Y2
Y3
Y4
Y5
Y6
Y7
CS
CS
CS
CS
CS
CS
CS
CS

FPGA译码

```
module DECODER(
   input [19:13] A, //输入地址信号
                    output [7:0] CS //输出片选信号
   );
                   reg [7:0] CS1; //设置输出寄存器
                   assign CS[7:0]=CS1[7:0];//输出引脚与寄存器相连
   always @(A)
begin
if (A[19:16] != 4'b0000)
CS1[7:0] <= 8'b11111111;//A[19:16]!=0000，所有CS无效
else
case (A[15:13])
3'b000  : CS1[7:0] <= 8'b11111110;
3'b001  : CS1[7:0] <= 8'b11111101;
3'b010  : CS1[7:0] <= 8'b11111011;
3'b011  : CS1[7:0] <= 8'b11110111;
3'b100  : CS1[7:0] <= 8'b11101111;
3'b101  : CS1[7:0] <= 8'b11011111;
3'b110  : CS1[7:0] <= 8'b10111111;
3'b111  : CS1[7:0] <= 8'b01111111;
endcase
                   end
endmodule
```

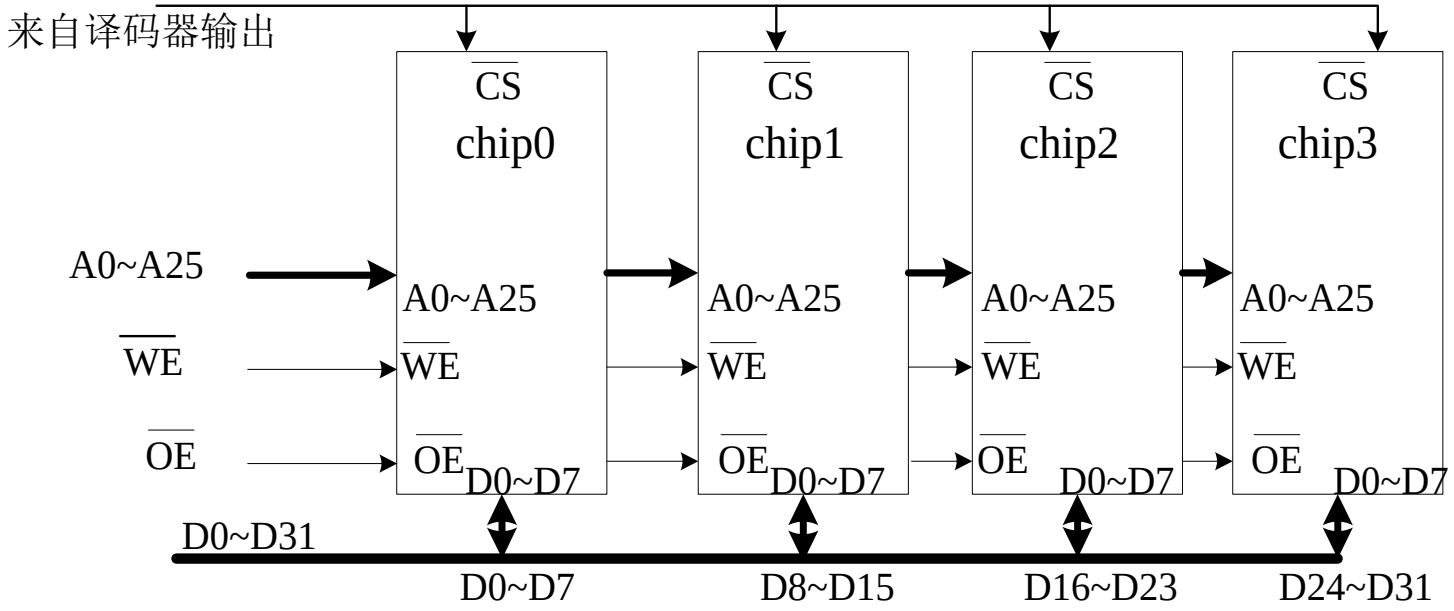
分级译码


片内译码地址
芯片译码地址
块译码地址
组译码地址
选择某一组
选择某一块
选择某个芯片
选择某个存储单元
M块
N个芯片
N个芯片
N个芯片
容量K
容量K
容量K
容量K
容量K
容量K
容量K
容量K
容量K

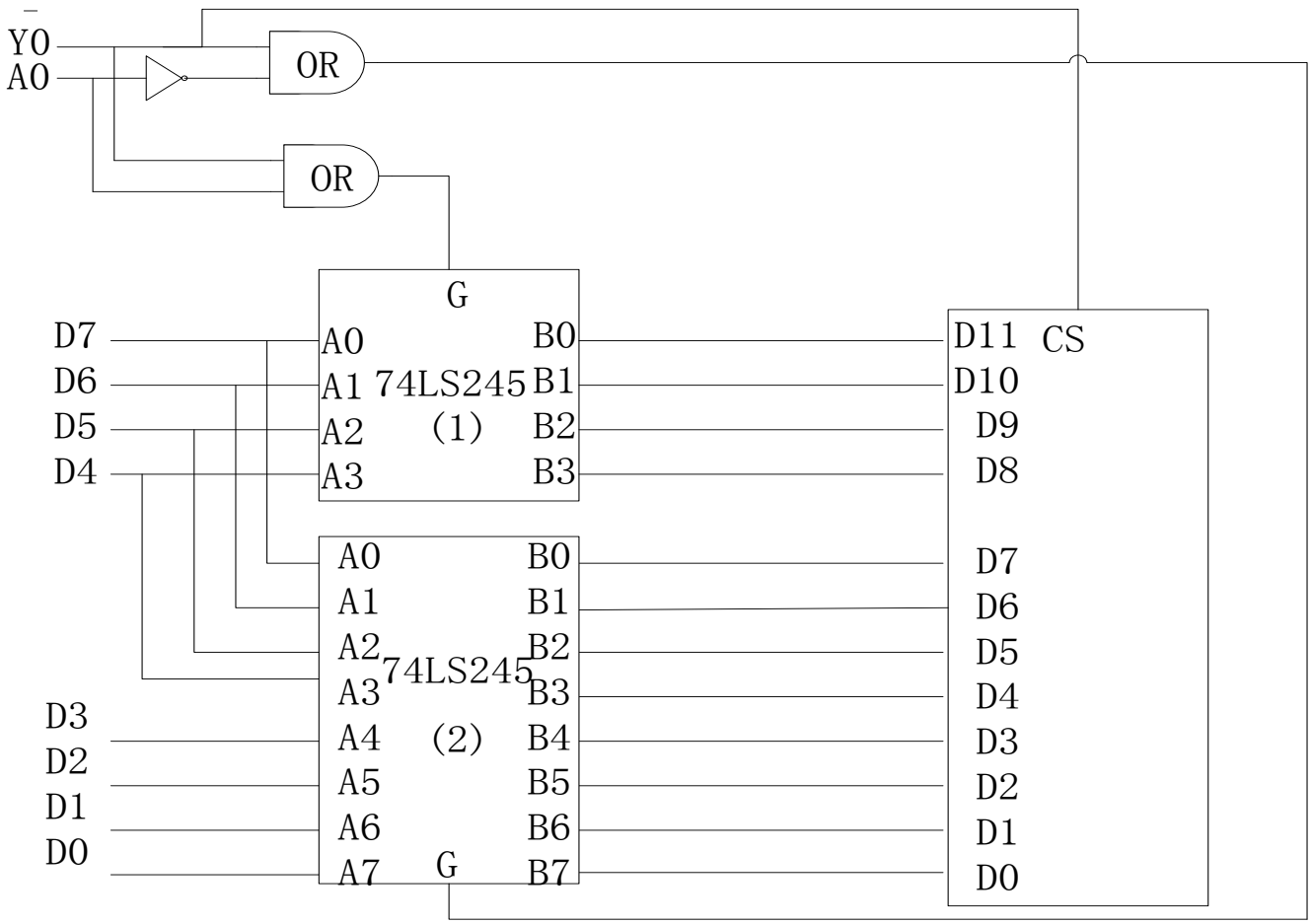
接口总线连接原则

- 地址总线连接原则
  - 地址总线中的低位地址总线与接口的地址总线相连，
  - 地址总线中剩余的高位地址总线通过译码电路译码之后连接到接口的控制信号上
- 数据总线连接原则
  - 接口位宽与数据总线位宽一致，那么仅需要将接口数据线与数据总线对应位直接相连即可
  - 接口位宽大于数据总线位宽，在数据总线宽度一定的情况下，需要复用部分数据总线来传输高位接口数据，
  - 8位数据总线接口芯片来构建更高位宽的接口，则需要多个同类型接口芯片采用并联方式，即字长扩展。由1位，2位，4位等不同位宽芯片来构建8位宽整数倍的接口同样也是采用这种方式。

字长扩展多芯片并联


来自译码器输出
CS
chip0
chip1
chip2
chip3
A0~A25
WE
OE
D0~D7
D0~D31
D0~D7
D8~D15
D16~D23
D24~D31

复用数据总线


Y0
A0
OR
OR
G
D7
D6
D5
D4
A0
A1
A2
A3
74LS245
(1)
B0
B1
B2
B3
D11
D10
D9
D8
CS
A0
A1
A2
A3
A4
A5
A6
A7
74LS245
(2)
G
B0
B1
B2
B3
B4
B5
B6
B7
D7
D6
D5
D4
D3
D2
D1
D0
D3
D2
D1
D0

通过8位数据总线输出12位数据

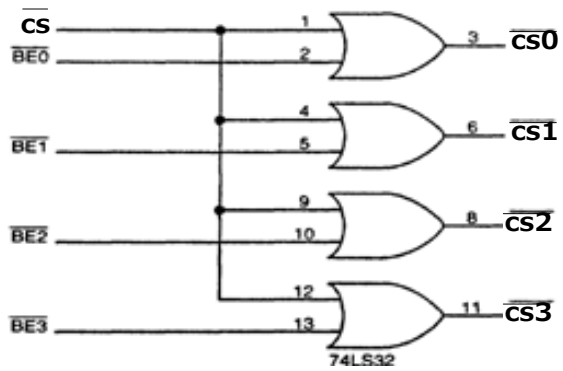
```
int data=0x789;
int port0=0x378,port1=0x379;
char byte0,byte1;
byte0=(char) data; //byte0=0x89
data=data>>4;//data=0x0078
byte1=(char)(data) ;//byte1=0x78
Xil_Out8(port0,byte1);//输出0x78，但是低4位无效
Xil_Out8(port1,byte0);//输出0x89
```

控制总线连接原则

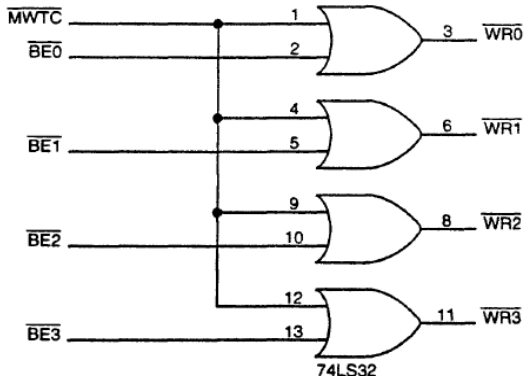
- 接口控制总线主要有读写控制信号、片选信号等信号
  - 独立IO结构计算机系统中，则需分别对IO接口和存储器接口的读写控制信号与总线提供的相对应读写控制信号相连：如IO接口的读写控制信号与微处理器提供的 $\overline{\text{IOR}}$，$\overline{\text{IOW}}$相连；存储器接口的读写控制信号与总线提供的$\overline{\text{MEMR}}$，$\overline{\text{MEMW}}$相连
  - 存储器映像IO结构，不需要区分是IO接口还是存储器接口，只需要直接将接口的读写控制信号与总线提供的读写控制信号相连
- 片选信号通常来自高位地址译码

不同位宽接口兼容

- 高位地址与字节使能信号译码产生不同的片选控制信号



- 写控制信号与字节使能信号译码产生不同的字节写控制信号



存储器接口设计

半导体存储芯片分类

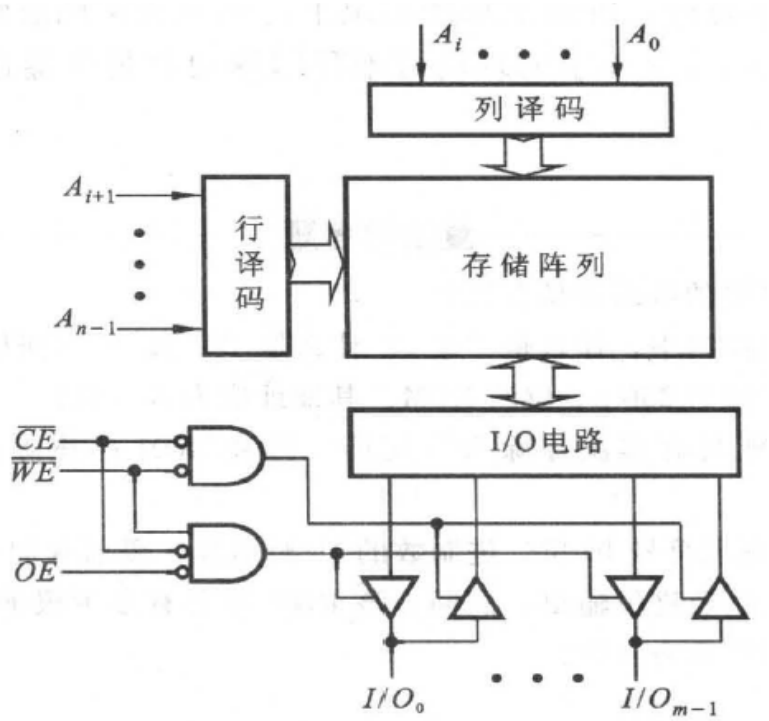
- 读写操作功能
  - 只读型存储器(Read Only Memory，ROM)
    - ROM一般用于存放程序代码与固定数据
    - ROM、OTP ROM、EPROM、EEPROM
  - 随机存取存储器(Random Access Memory，RAM)。
    - RAM通常用于构成PC主存储器
    - SRAM与DRAM
- 读写操作时序：异步存储器和同步存储器
- 数据传输的方式：并行存储器与串行存储器

并行存储器的引脚

- 地址线
  - 地址线的多少可以表征存储器芯片的容量
- 数据线
  - 数据线的多少表征存储器的数据宽度
- 片选线
  - 片选线用于选中某一指定的存储器芯片
- 控制线
  - 控制数据的传输方向，是读数据还是写数据
- 如果是同步存储器则还有时钟信号线

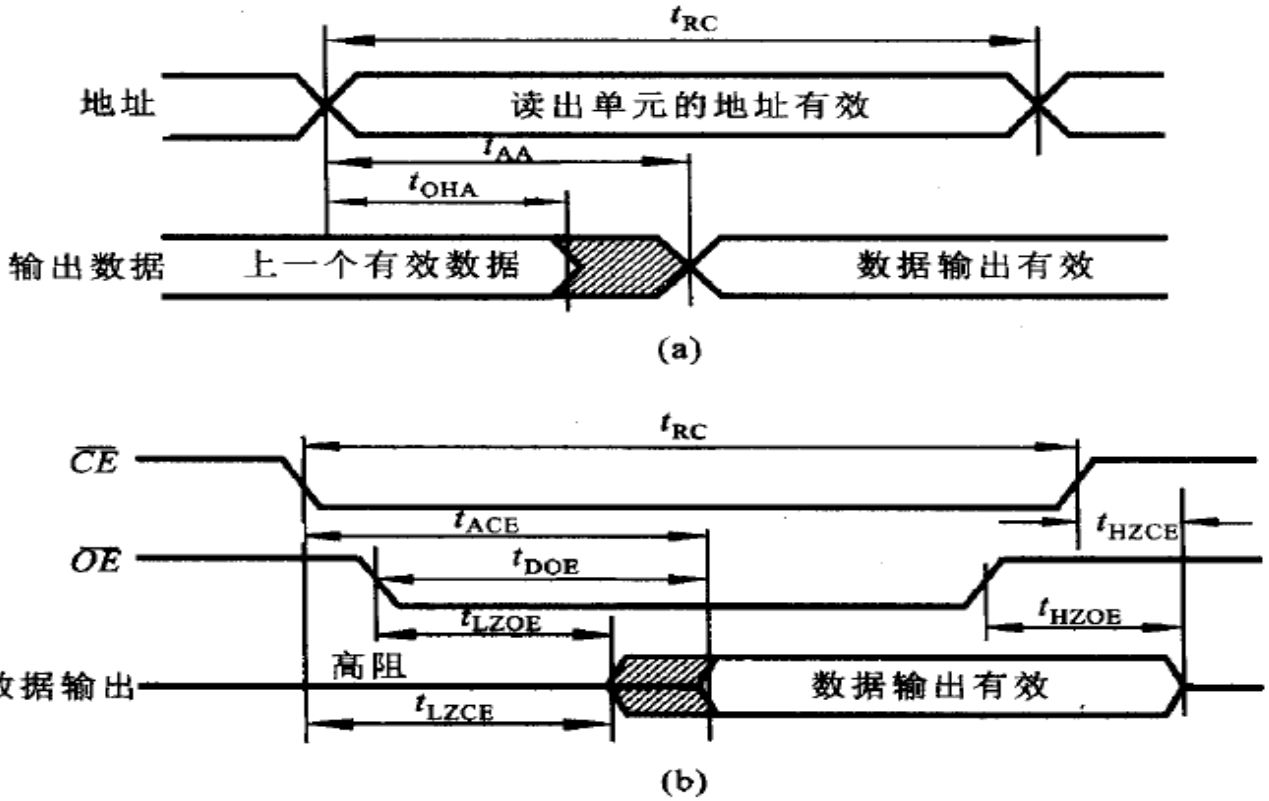
典型存储芯片接口

- 异步SRAM存储器接口



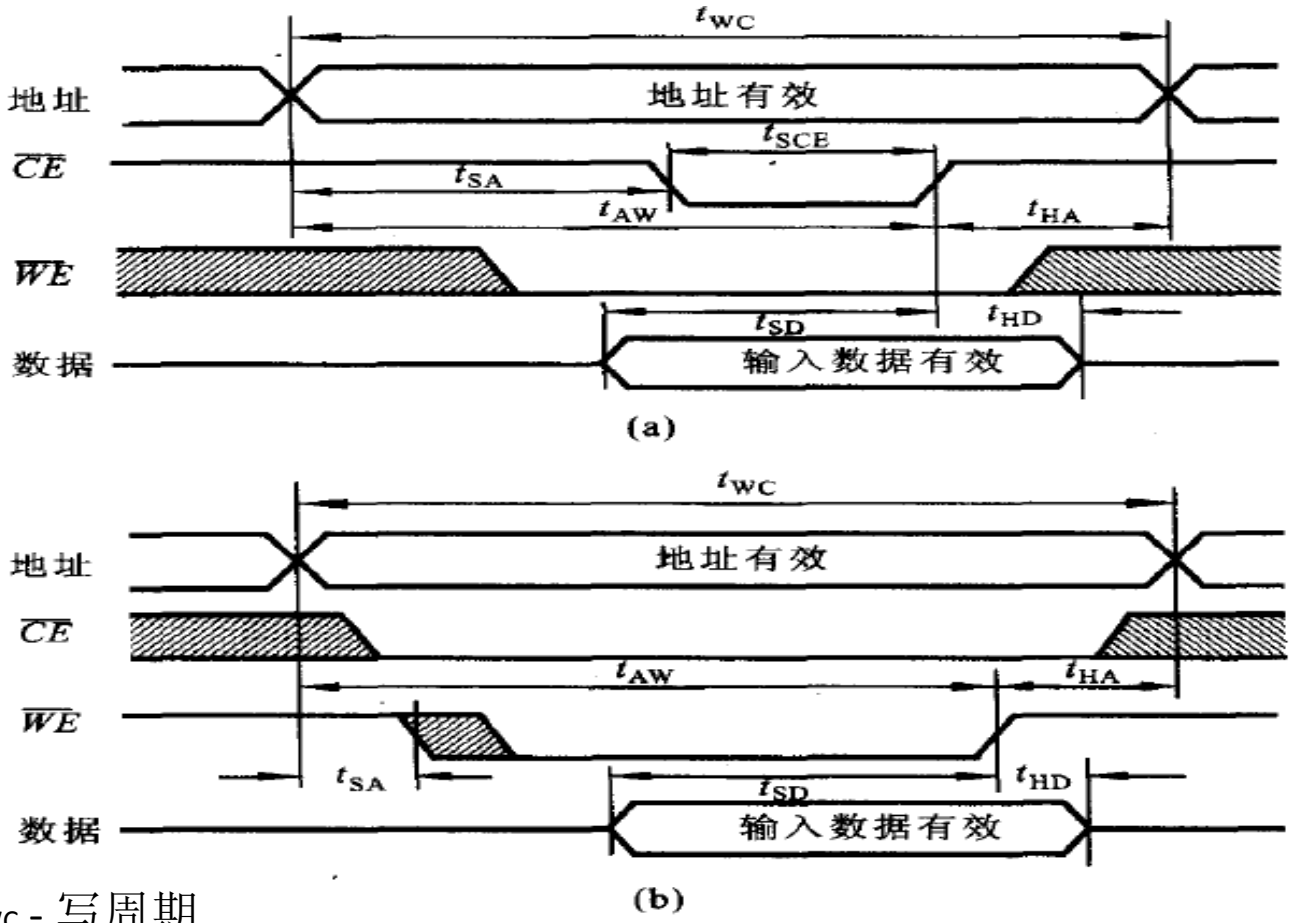
工作模式	$\overline{CE}$	$\overline{WE}$	$\overline{OE}$	I/O_0– I/O_m
保持（微功耗）	1	X	X	高阻
读	0	1	0	数据输出
写	0	0	1	数据输入
输出无效	0	1	1	高阻

异步SRAM读操作时序



T_{CEDV} - 读周期片选信号保持低电平到有效数据建立的时间
T_{AVDV} - 读周期地址保持有效到有效数据建立的时间
T_{PACC} - 页访问模式下的数据读周期
T_{HZCE}- 片选信号变为高电平到数据线变为高阻态的时间
T_{HZOE} – 读信号变为高电平到数据线变为高阻态的时间

异步SRAM写操作时序



CE控制

WE控制

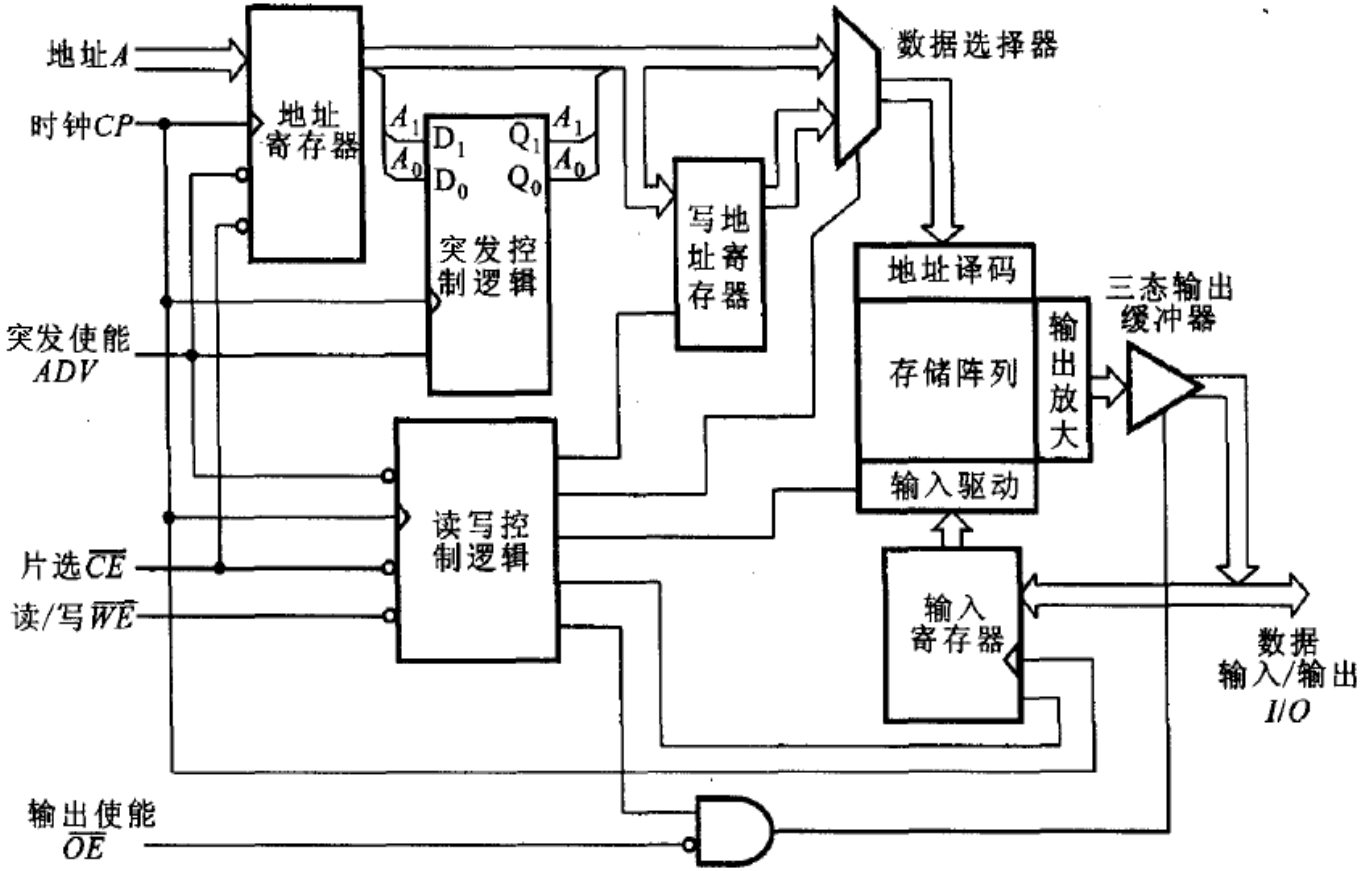
T_{WC} - 写周期

T_{WP} - 写使能信号低电平脉冲的最短时间

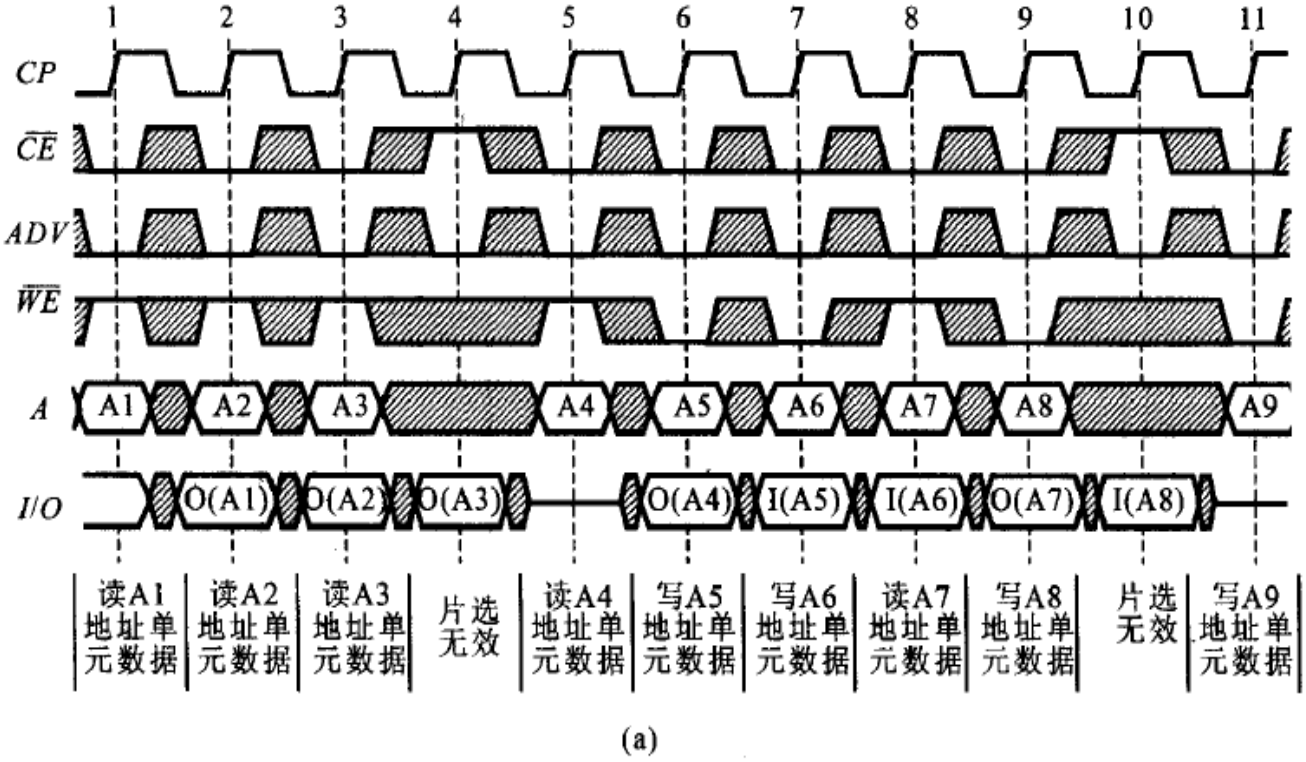
T_{WPH} - 写使能信号高电平的最短时间

T_{LZWE} - 写使能信号变为高电平到数据总线低阻态的时间

同步静态存储器（SSRAM）



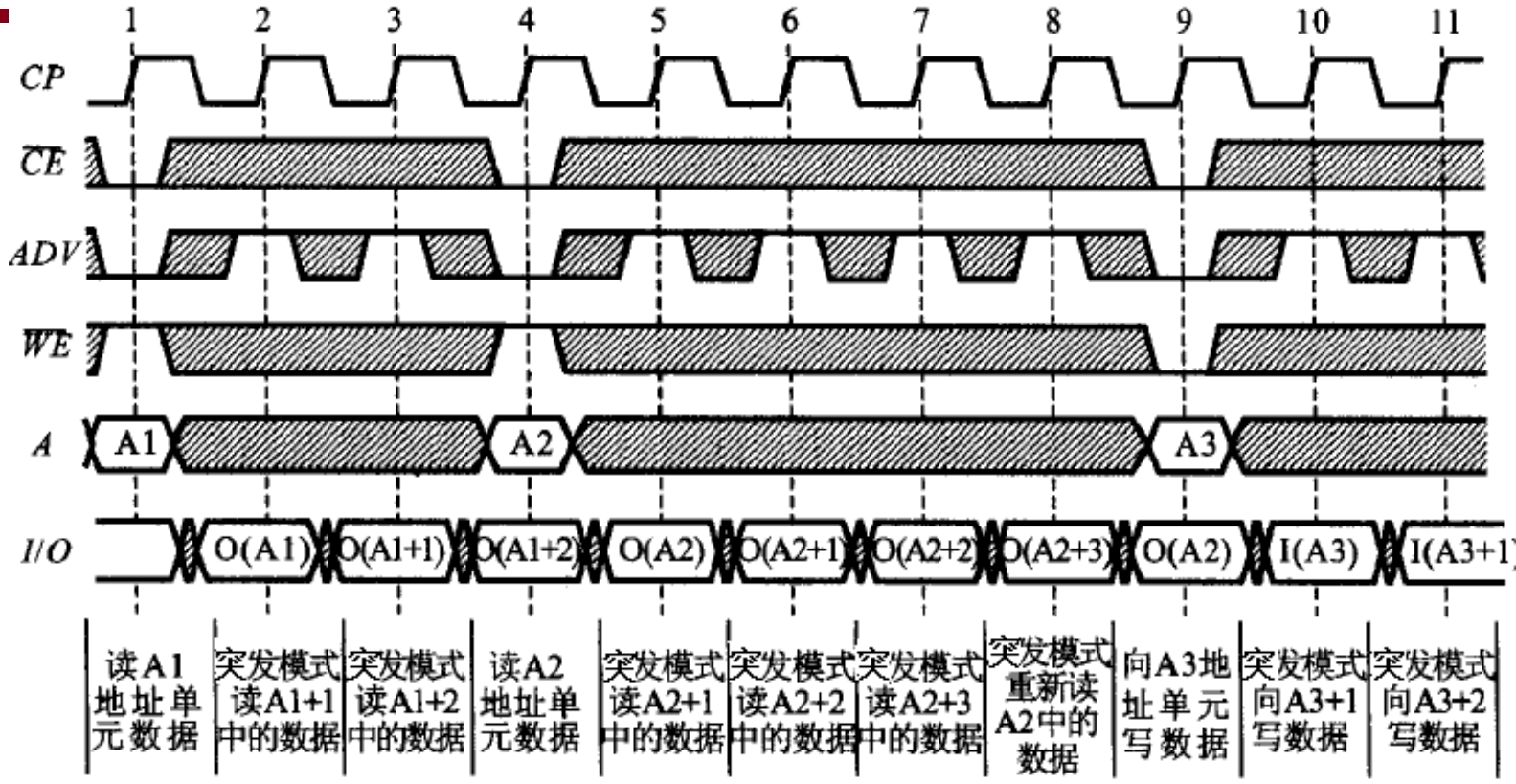
SSRAM读写操作时序_单个数据



(a)

当突发使能控制ADV为低电平时，A_1A_0可直接穿过突发控制逻辑电路，按外部给定的地址进行读/写，此时就是一般读写操作

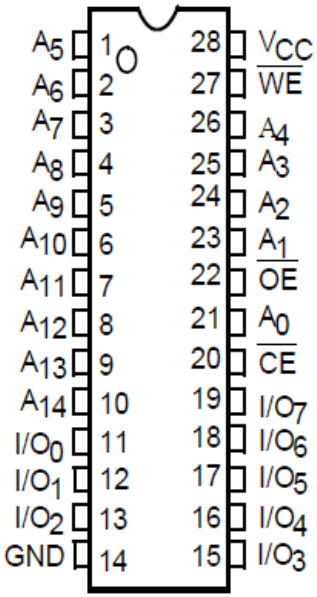
SSRAM读写操作时序_突发模式

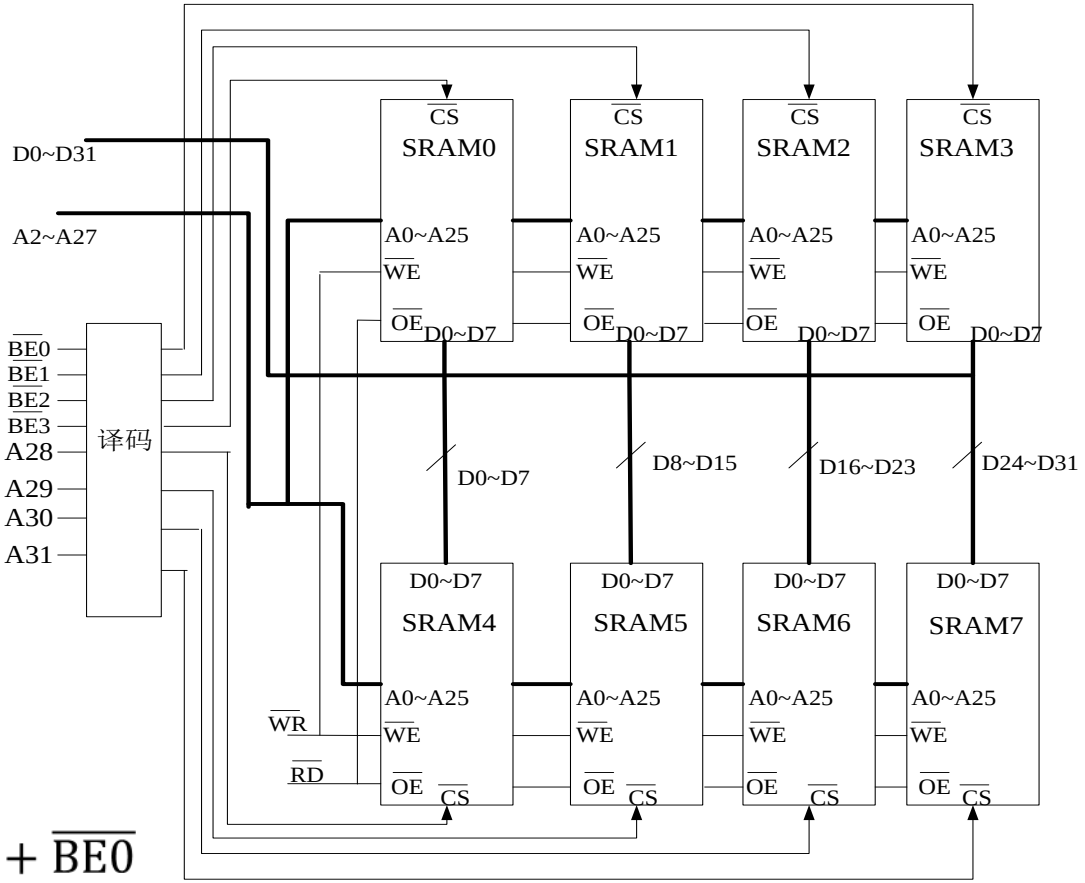


(b)

当突发使能控制ADV 为高电平时，地址寄存器不接收外部新地址而保持上一个时钟周期输入的地址，在CP下一个上升沿到来时，由突发计数器在上一个A_1A_0基础上，计数生成下一个地址的A_1A_0进行读/写。由于突发计数器是2 位计数器，所以在ADV 保持高电平时，可以连续生成4 个不同的地址

简单存储器接口





SRAM0: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28}} + \overline{BE0}$

SRAM1: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28}} + \overline{BE1}$

SRAM2: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28}} + \overline{BE2}$

SRAM3: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot \overline{A28}} + \overline{BE3}$

SRAM4: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot A28} + \overline{BE0}$

SRAM5: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot A28} + \overline{BE1}$

SRAM6: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot A28} + \overline{BE2}$

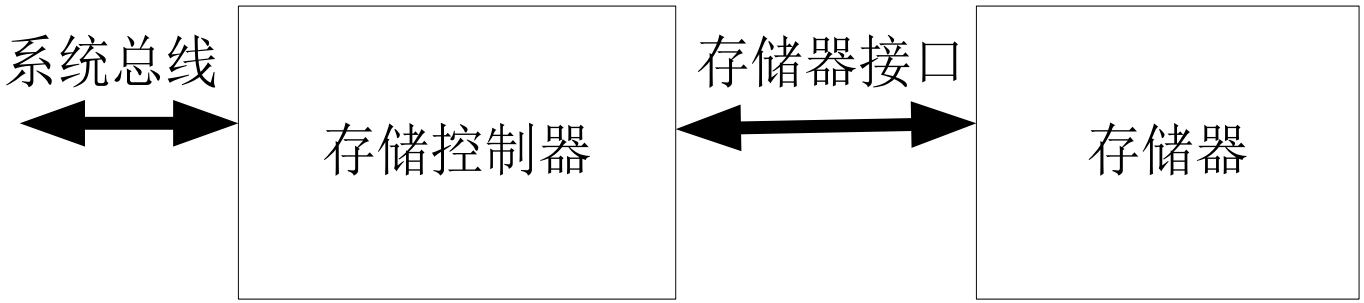
SRAM7: $\overline{CS} = \overline{\overline{A31} \cdot \overline{A30} \cdot \overline{A29} \cdot A28} + \overline{BE3}$

00000000H~0FFFFFFFH

10000000H~1FFFFFFFH

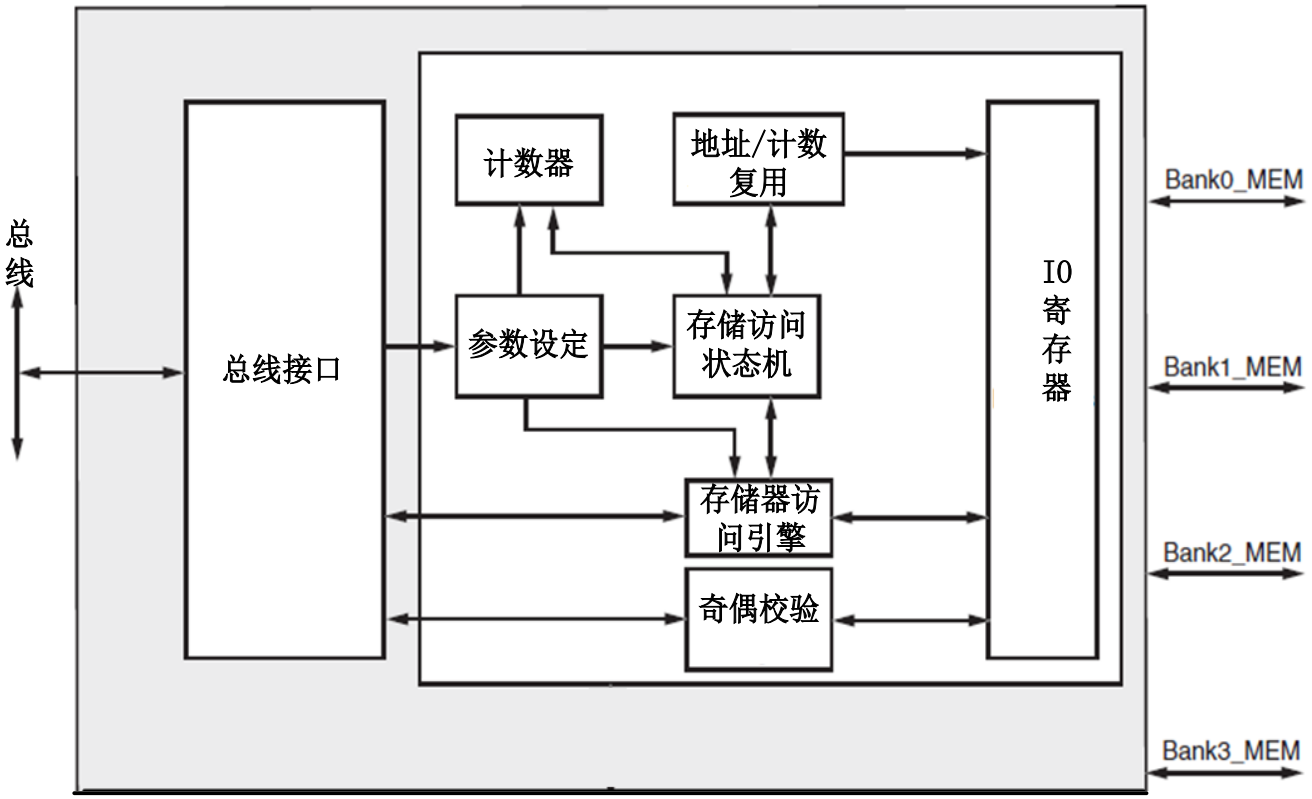
存储控制器

- 将系统总线转换为适合访问各类存储器总线信号的接口设备



存储控制器结构

- 存储芯片、存储模块、子存储系统之间的关系为：存储芯片（device）通过字长以及字数扩展构成存储模块（bank），不同的存储模块通过存储控制器组成子存储系统（sub-system）。



Xilinx异步SRAM存储器接口信号

信号类型描述	存储控制器引脚名称	存储芯片引脚名称
数据线	MEM_DQ(((DN+1)*DW)-1:DN*DW)	D(DW-1 : 0)
地址线	MEM_A(HAW-AS-1:HAW-MAW-AS)	A(MAW-1 : 0)
芯片使能线（低电平有效）	MEM_CEN(BN)	CEN
读使能线（低电平有效）	MEM_OEN	OEN
写使能线（低电平有效）	MEM_WEN	WEN (有字节使能的芯片)
写使能线（低电平有效）	MEM_QWEN(DN*DW/8)	WEN (无字节使能的芯片)
字节使能线（低电平有效）	MEM_BEN((((DN+1)*DW/8)-1):(DN*DW/8))	BEN(DW/8-1 : 0)

变量名称	具体含义	变量名称	具体含义
DN	存储块内芯片序号	HAW	总线地址宽度
BN	子存储系统存储块序号	MW	存储块数据位宽
DW	存储芯片数据总线位宽	AS	地址偏移宽度=$\log_2(\frac{AU*MW}{DW}/8)$
AU	存储芯片可寻址最小数据位宽	MAW	存储芯片地址总线宽度

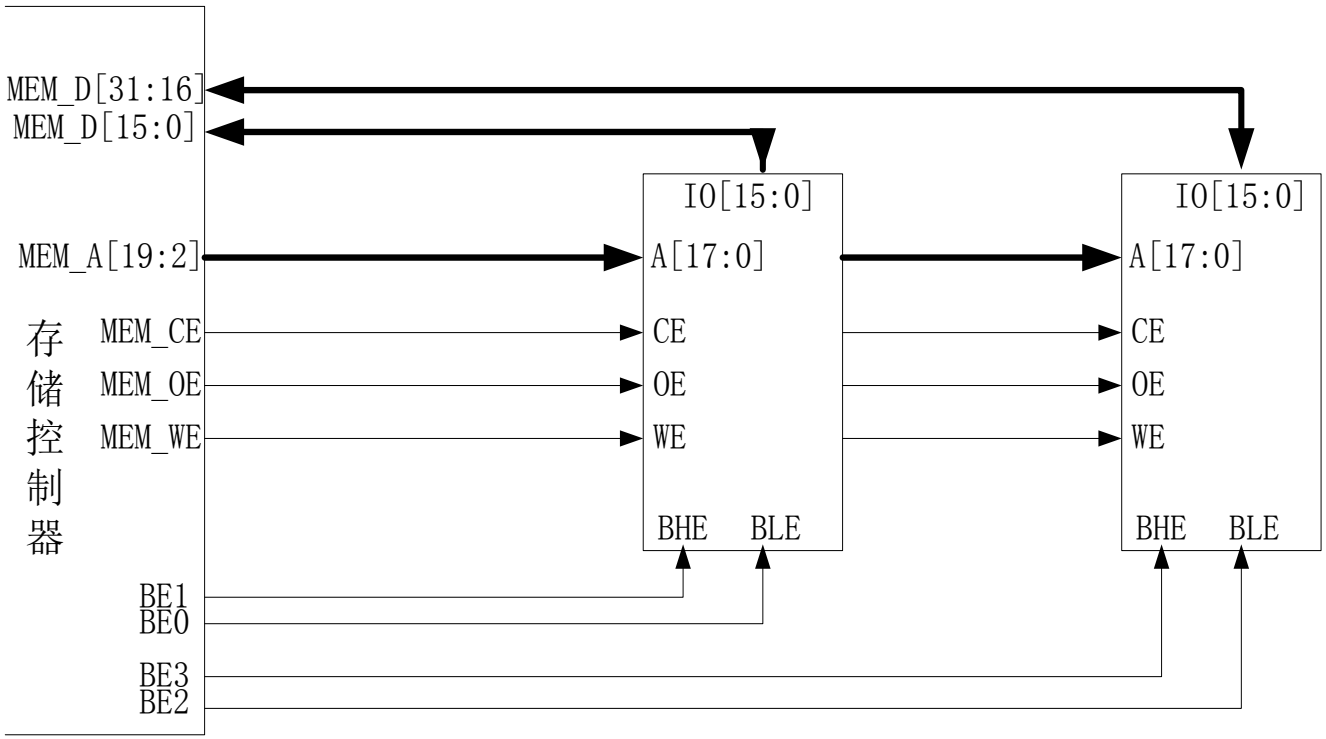
- 异步SRAM芯片IDT71V416S为256k*16b的存储芯片，支持字节访问。要求采用2片芯片通过存储控制器构建一个32位的存储模块，试设计存储控制器与存储芯片之间的接口。

变量名	值	含义
BN	0	一个存储块，序号为0
DN	0，1	两个存储芯片，序号为0，1；其中0号为高位数据，1号为低位数据
MW	32	子存储系统数据位宽32位
DW	16	存储芯片数据位宽16位
MAW	18	存储芯片地址总线18位
AU	16	存储芯片字长16位
AS	2	地址偏移2位=$\log_2(32*16/16)/8$
HAW	32	地址总线32位（AXI总线）

存储控制器与存储芯片之间的引线连接

设备号	引脚含义	存储控制器段引脚名称	存储芯片引脚名称
0	数据线	MEM_DQ(15 : 0)	I/O(15 : 0)
	地址线	MEM_A(19 : 2)	A(17 : 0)
	芯片使能	MEM_CEN(0)	CS
	读使能	MEM_OEN	OE
	写使能	MEM_WEN	WE
	字节使能	MEM_BEN(1 : 0)	BHE:BLE
1	数据线	MEM_DQ(31 : 16)	I/O(15 : 0)
	地址线	MEM_A(19 : 2)	A(17 : 0)
	芯片使能	MEM_CEN(0)	CS
	读使能	MEM_OEN	OE
	写使能	MEM_WEN	WE
	字节使能	MEM_BEN(3 : 2)	BHE:BLE

存储控制器与存储芯片IDT71V416S之间的接口

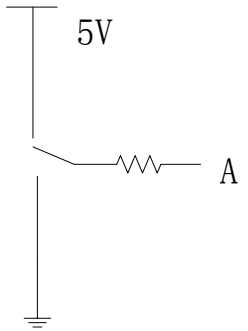


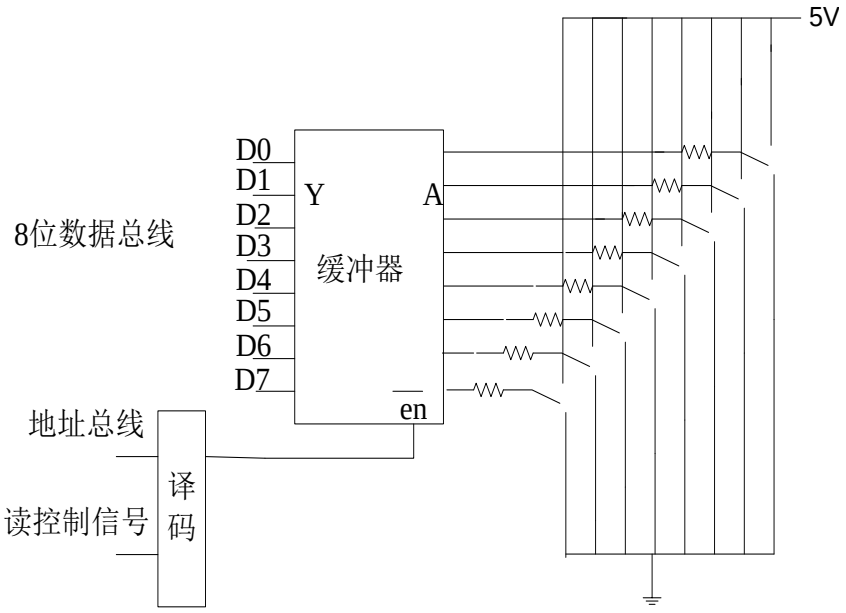
并行IO接口设计

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

独立开关输入接口

- RC吸收电路或RS触发器组成的闩锁电路来消除按键抖动；
- 采用软件延时方法消除抖动

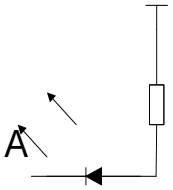


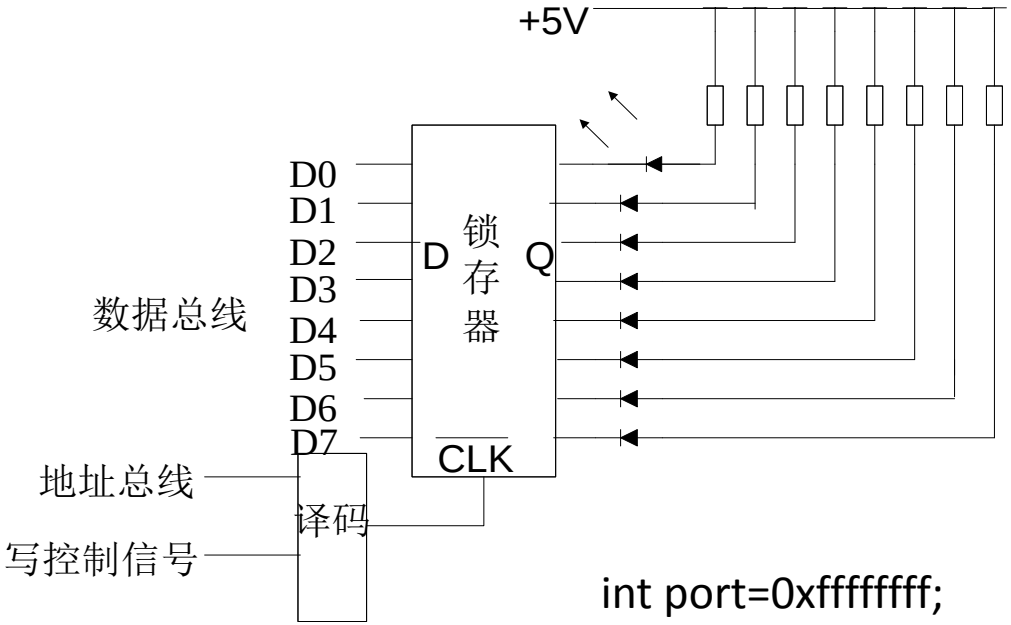


int port=0xffffffff;
char Switch=Xil_In8(port);8位开关的状态保存在Switch中

发光二极管输出接口

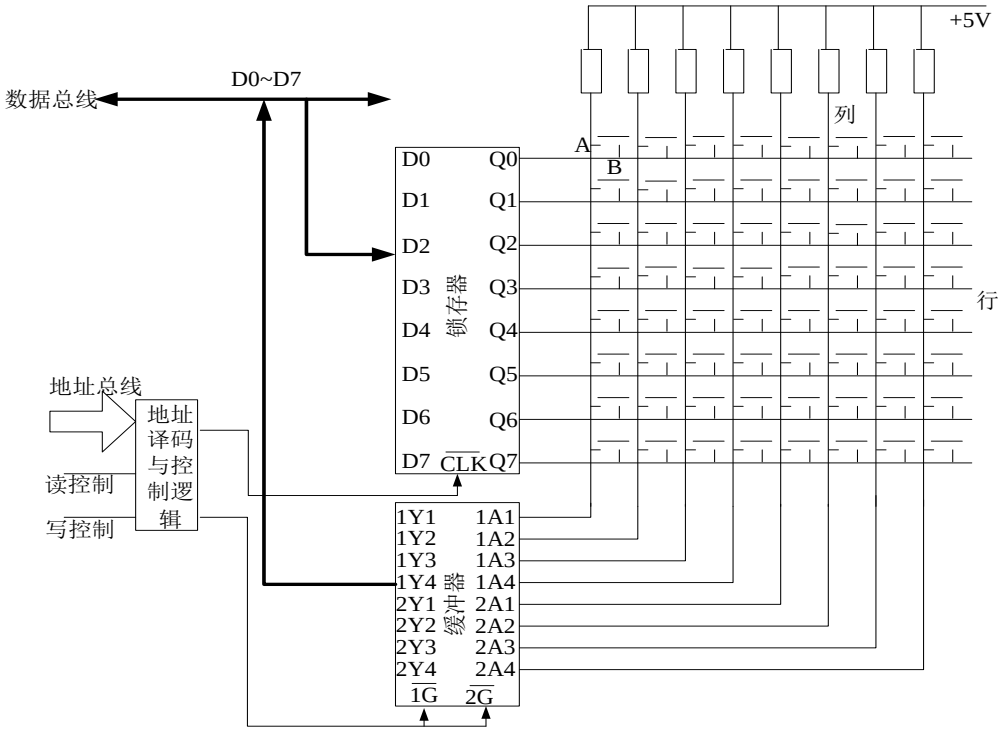
- 必须具有数据锁存功能

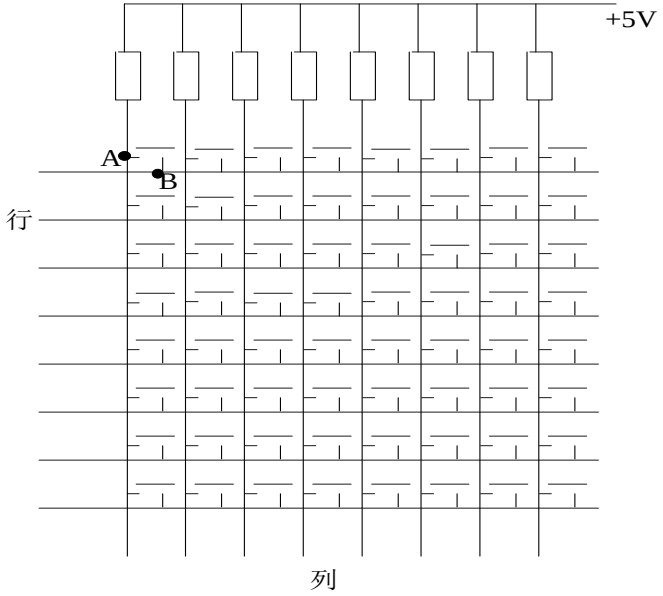





```
int port=0xffffffff;
Xil_Out8( port,0x0);
```

矩阵式键盘接口


+5V
数据总线
D0~D7
列
A
B
D0
D1
D2
D3
D4
D5
D6
D7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
锁存器
CLK
行
地址总线
地址译码与控制逻辑
读控制
写控制
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
缓冲器
1G
2G


+5V
A
B
行
列

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，进入步骤4）；
4. 输出该行为低电平，其余行为高电平
5. 读取键盘列信号；
6. 检测是否有键按下，若没有键按下，获得0xff，指向下一行，重复步骤4）~6）；当有键按下时，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。

矩阵式键盘按键编码表

	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

采用Xilinx C语言来读取按键的行列信息

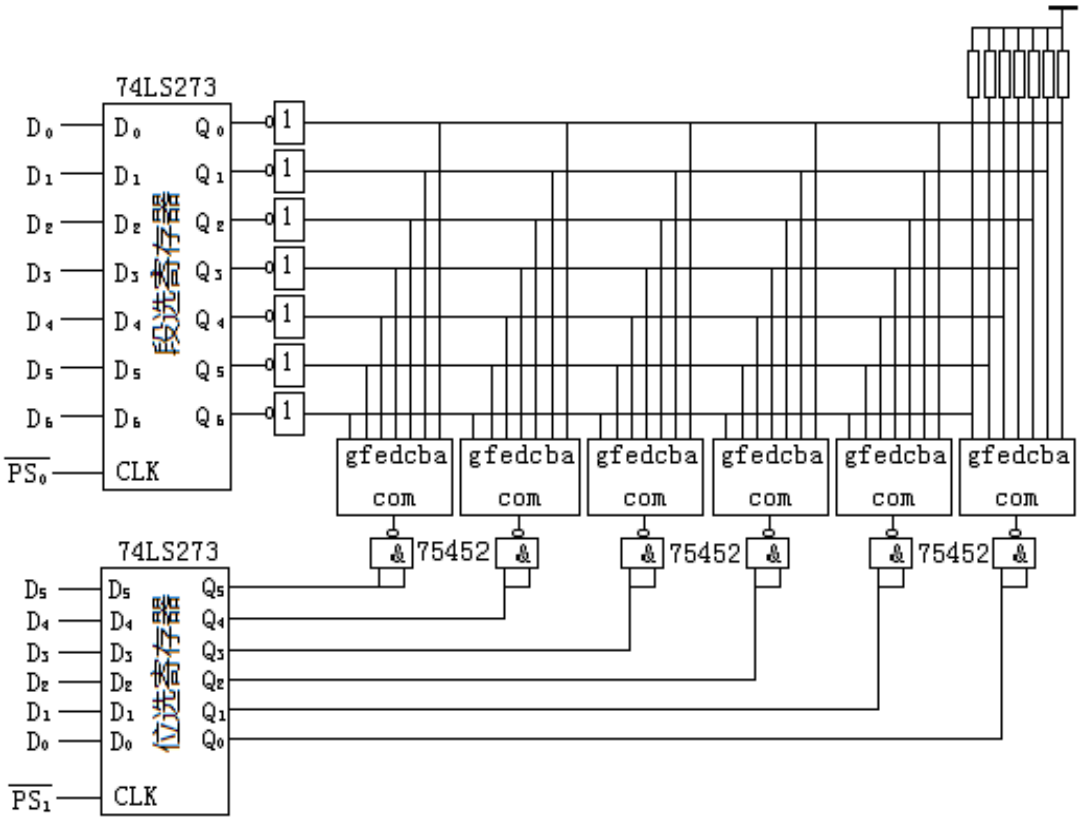
```
int AddressPort=0xfffffffe,DataPort=0xffffffff;
        char col, row, rowtemp=0x01;
        Xil_Out8(AddressPort,0x00);
        While ((col=Xil_In8(DataPort))==0xff);
        row=~rowtemp;
        Xil_Out8 (AddressPort,row);
while ((col=Xil_In8 (DataPort))==0xff)
                {       rowtemp=rowtemp<<1;
                        row=~rowtemp;
                        Xil_Out8 (AddressPort,row);
}
```

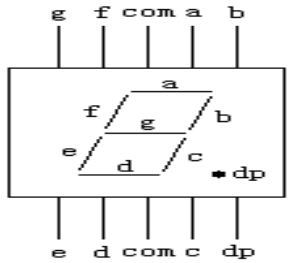
- 按键行和列信息分别保存在变量row和col中。

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);
```

七段数码管动态显示接口





10个十进制数的字形代码分别是0x40，0x79，0x24，0x30，0x19，0x12，0x02，0x78，0x00H，0x18

- PS0表示端口地址0x80，而PS1表示端口地址0x81，在6位显示器上分别显示1,2,3,4,5,6等字符，那么其Xilinx C语言程序段为如下所示：

```
char segment[5]={ 0x79,0x24,0x30,0x19,0x12,0x02};
int AddressPort=0x81,DataPort=0x80;
char position;
Xil_Out8(AddressPort,0x0);//使所有的七段数码管熄灭
While(1)
{               position=0x20;
for(inti=0;i<6;i++)
{
                        Xil_Out8(DataPort, segment[i]); //输出第一位的段码
                        Xil_Out8(AddressPort,position);//点亮第一位7段数码管
                        delay; //延时
                        position=position >> 1;//控制下一个7段数码管
}
}
```

软件延时

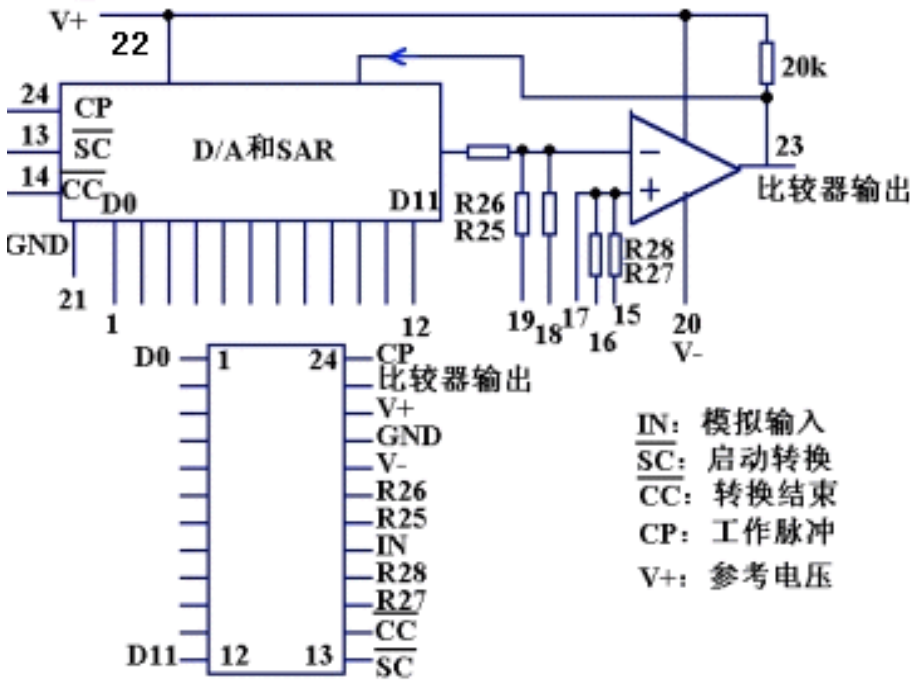
软件延时的例子：

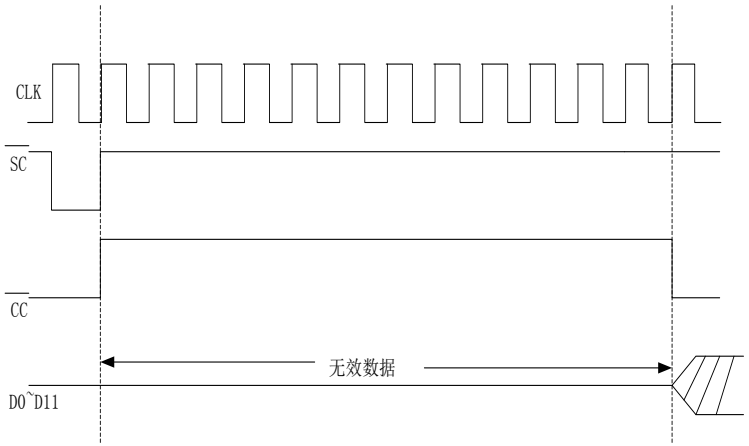
```
for(int i=0;i<0x10000;i++);
```

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

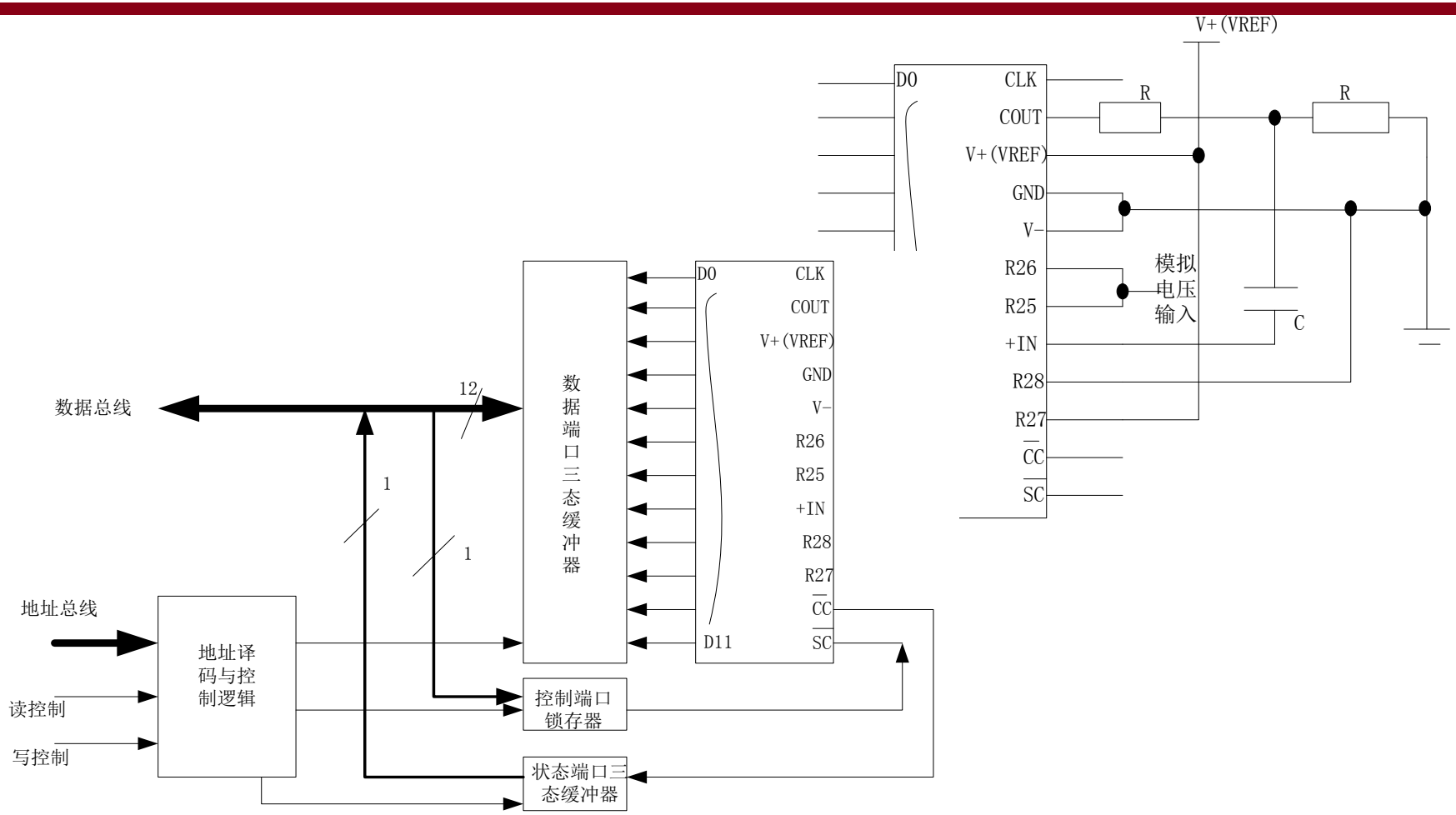
- 执行加法指令所需的时钟周期个数为N_a，
- 执行比较指令所需的时钟周期个数为N_c，
- 执行条件跳转指令所需的时钟周期个数为N_j，
- 微处理器时钟周期为T，
- 循环次数为M

AD转换器ADC1210接口

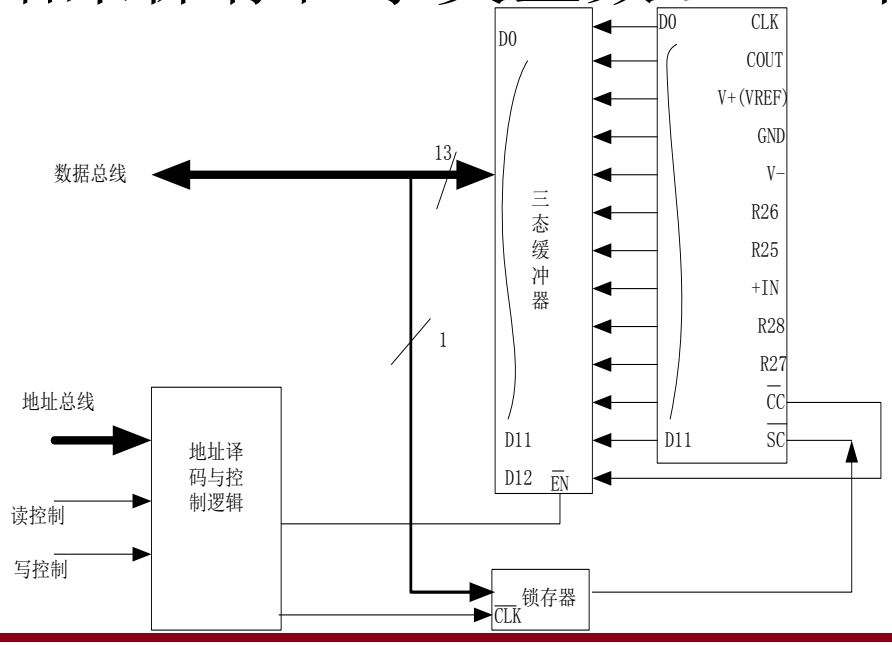




ADC1210单极性模拟电压输入电路连接



- 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

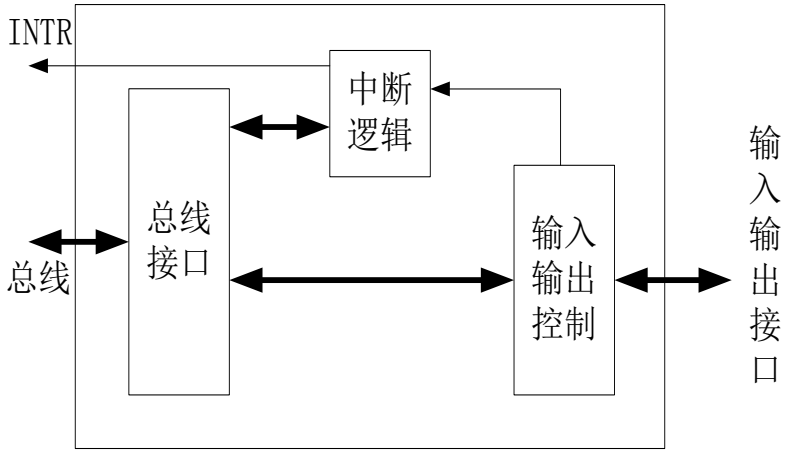


控制程序段

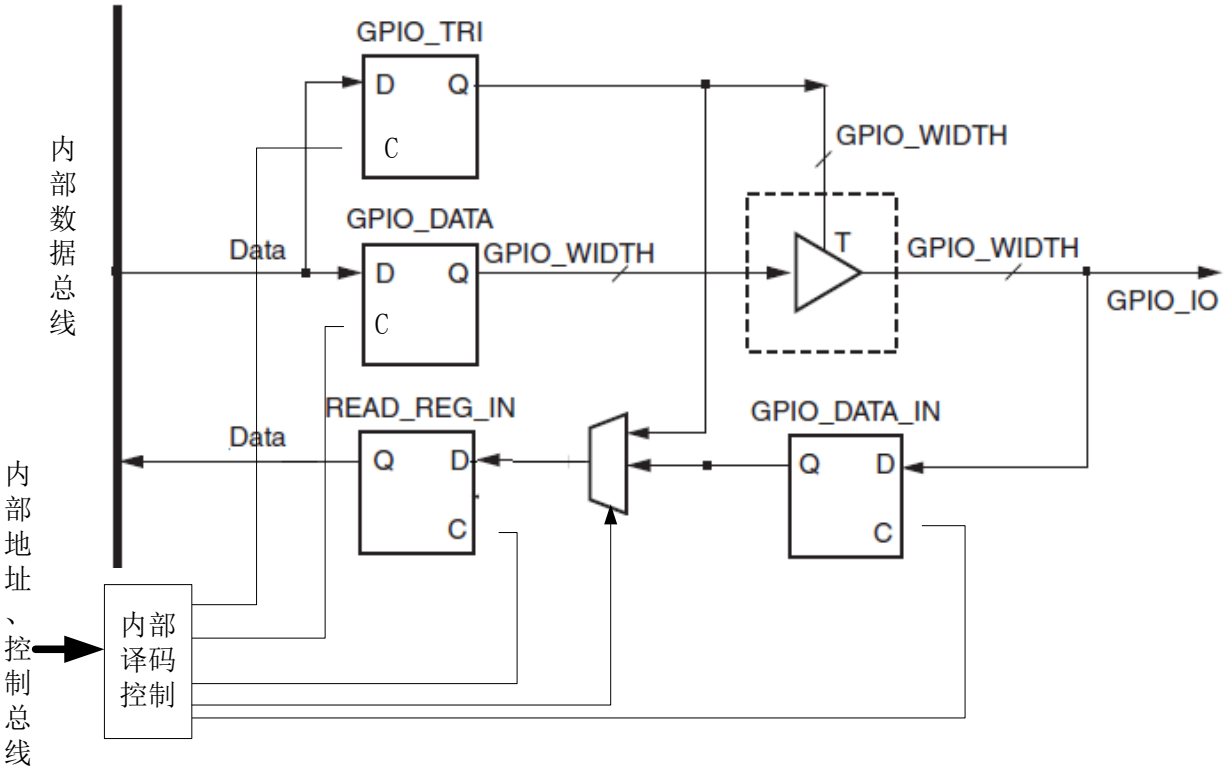
```
int volt[100]， i;
int port=0x80000000;
for(i=0;i<100,i++)
{
Xil_Out8(port,0x01);//输出数据使得$\overline{SC}$为高电平
Xil_Out8(port,0x00);//输出数据使得$\overline{SC}$为低电平
Delay;//延时一个ADC1210的时钟周期
Xil_Out8(port,0x01);//输出数据使得$\overline{SC}$为高电平，从而产生启动转换信号
While ((Xil_In16(port)&0x1000)!=0);//查询状态
Volt[i]=Xil_In16(port)&0x0fff;//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

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

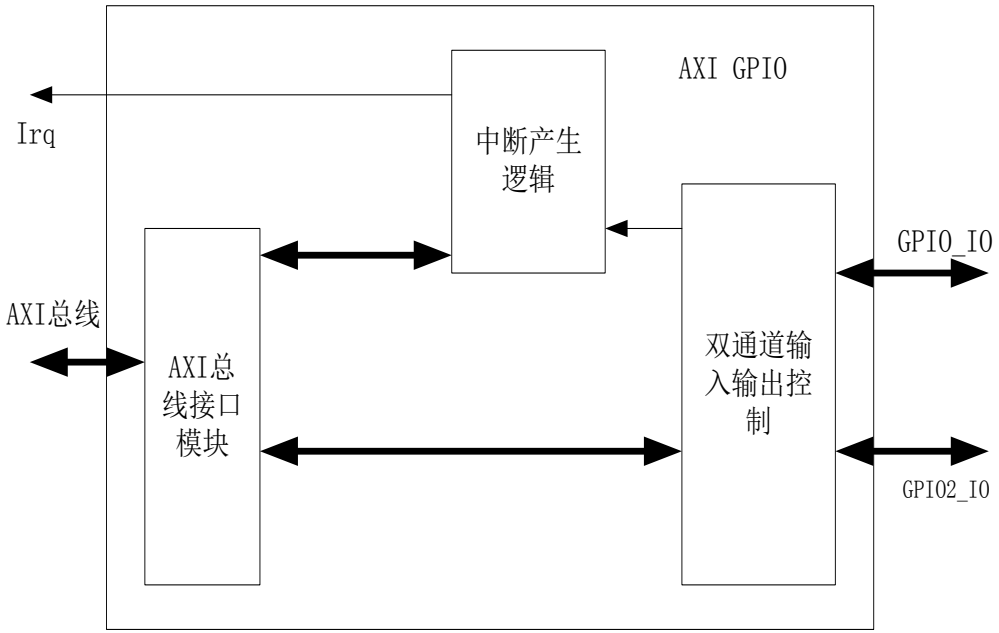


输入输出控制模块内部框图



Xilinx AXI总线GPIO IP核

- 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

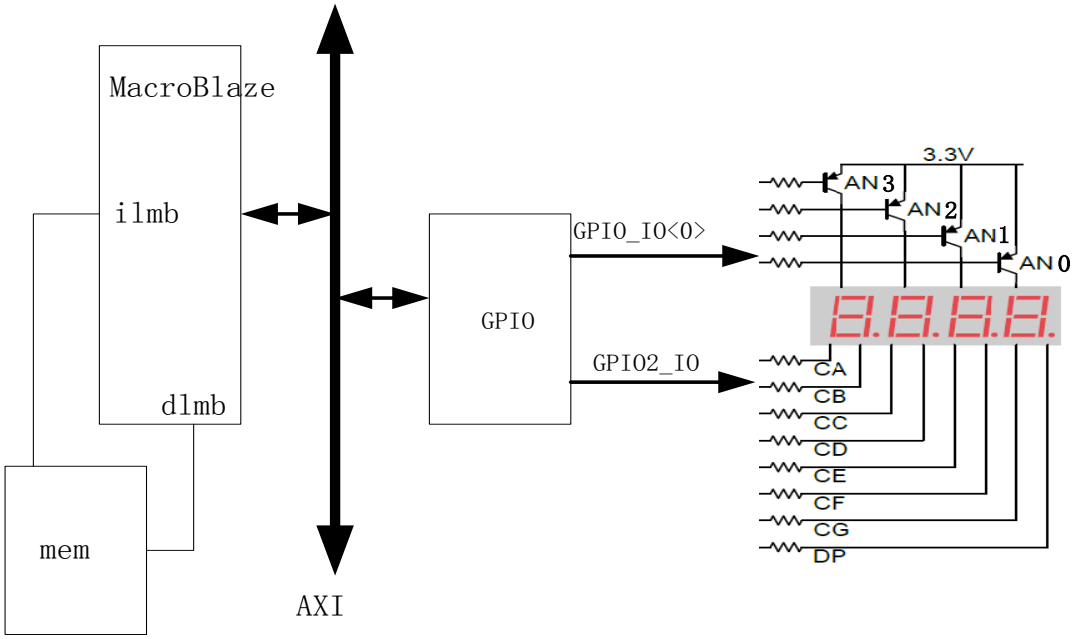


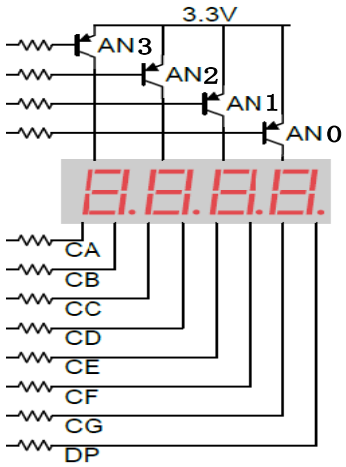
GPIO内部寄存器

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2数据寄存器	通道2数据
GPIO2_TRI	0XC	0	通道2三态控制寄存器	写控制通道2传输方向

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

采用GPIO控制器控制该四位7段数码管显示字符PASS

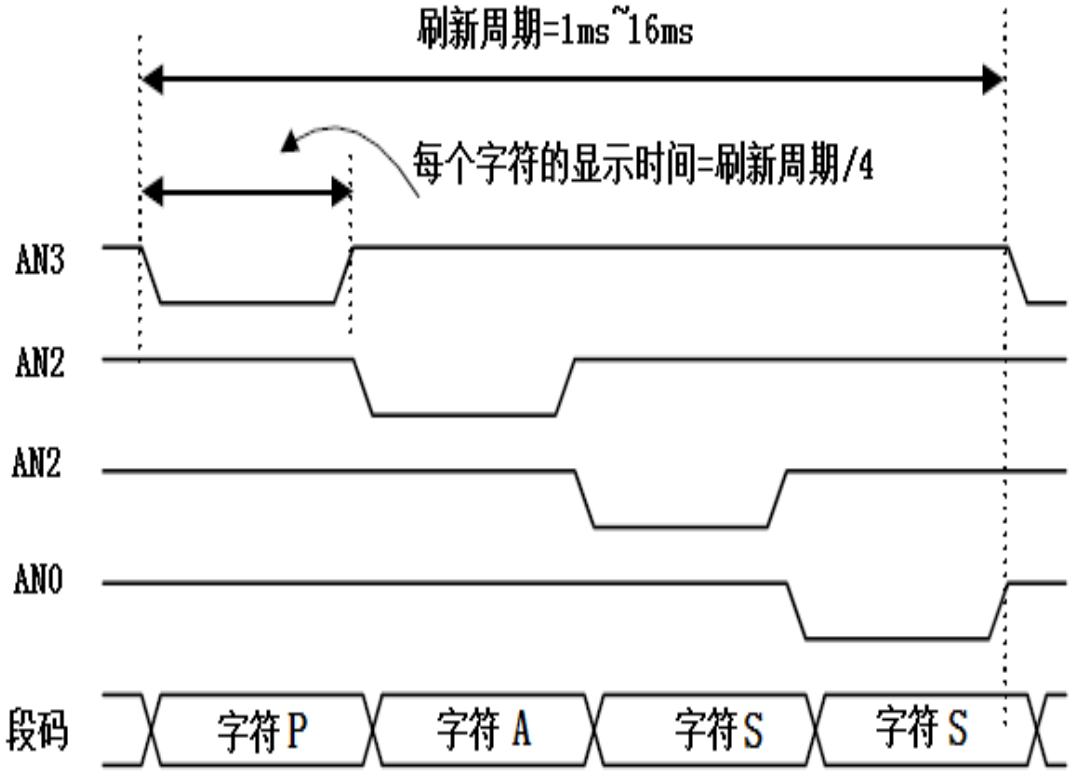




GPIO_IO<0>连接AN0，GPIO_IO<1>连接AN1，
GPIO_IO<2>连接AN2，GPIO_IO<3>连接AN3；
GPIO2_IO<0>连接CA，GPIO2_IO<1>连接CB，
GPIO2_IO<2>连接CC，GPIO2_IO<3>连接CD，
GPIO2_IO<4>连接CE，GPIO2_IO<5>连接CF，
GPIO2_IO<6>连接CG，GPIO2_IO<7>连接DP。

- 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。

4位7段数码管显示PASS的控制时序



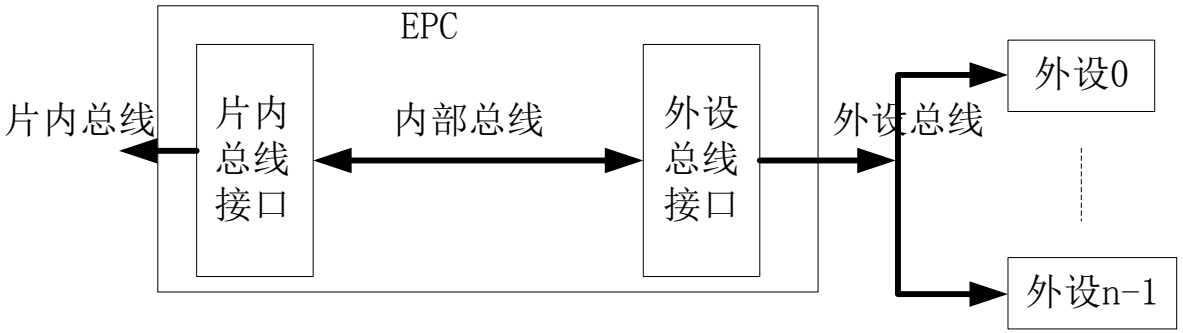
控制程序段

```
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;
}
```

- 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。

外设控制器（EPC）

- 外设控制器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。它一方面实现总线信号转换，另一方面还实现高位地址译码，产生外设片选信号。

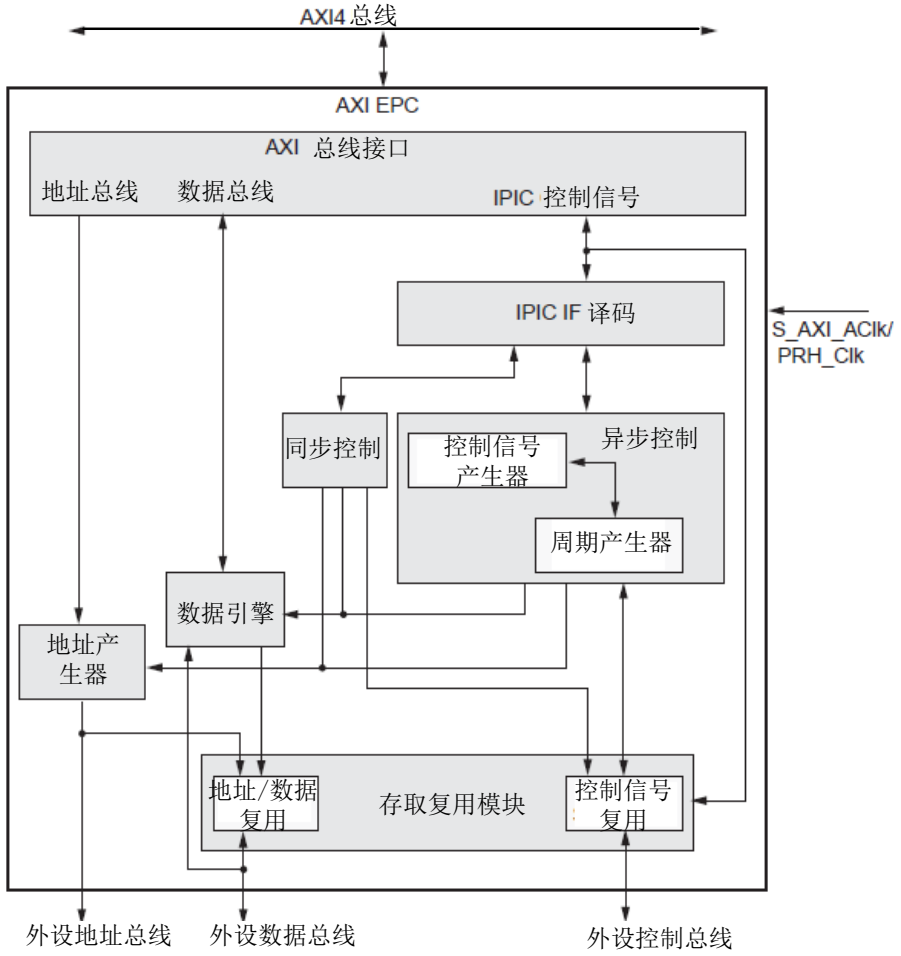


Xilinx AXI总线EPC

- 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] and the 32-bit peripheral should connect to PRH_Data [0 : 31].

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



外设总线信号类型以及含义

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	-

所有外设总线参数

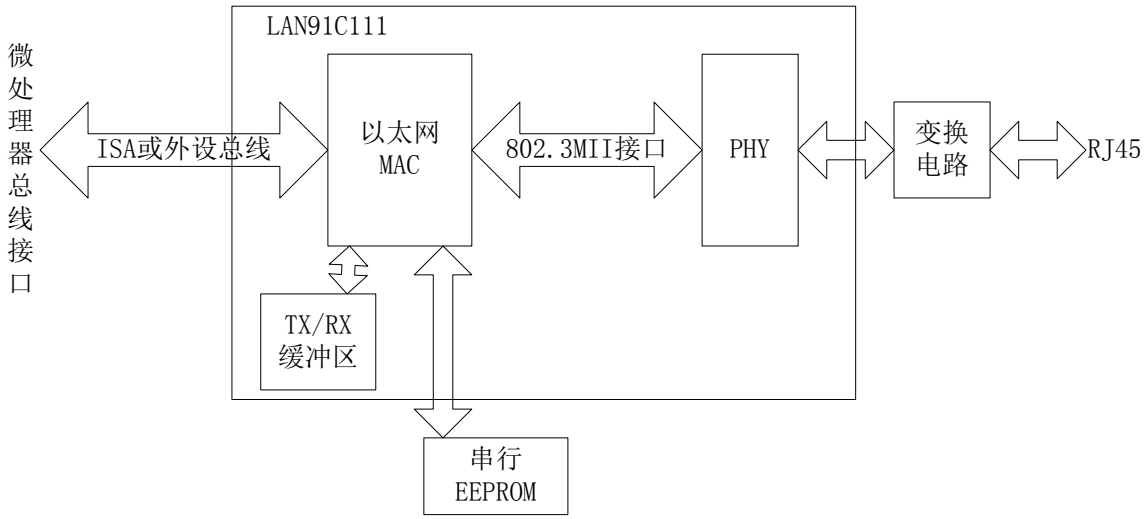
参数名称	含义	可能值	初始值
C_NUM_PERIPHERALS	外设数目	1-4	1
C_PRH_MAX_AWIDTH	最大地址总线宽度	3-32	32
C_PRH_MAX_DWIDTH	最大数据总线宽度	8，16，32	32
C_PRH_MAX_ADWIDTH	最大地址数据复用总线宽度	8-32	32
C_PRH_CLK_SUPPORT	外部时钟支持	1：外部时钟 0：AXI总线时钟	
C_PRH_BURST_SUPPORT	是否支持burst	0	0

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

以太网接口芯片LAN91C111接口设计

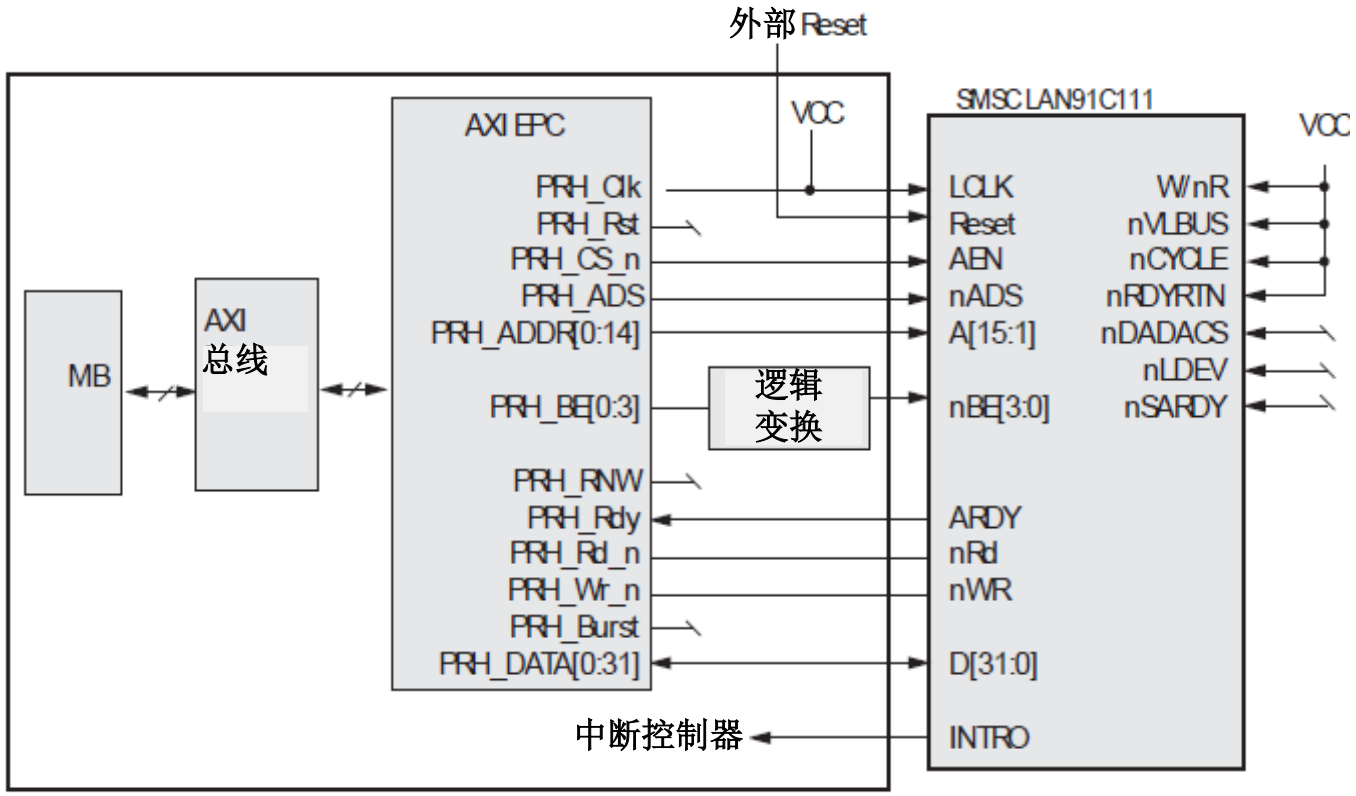
- 已知一采用小字节序的非PCI 接口支持MAC和PHY单一以太网接口芯片LAN91C111，支持8位、16位、32位的主机接口，内部具有8K FIFO存储体供发送和接收数据缓冲。



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

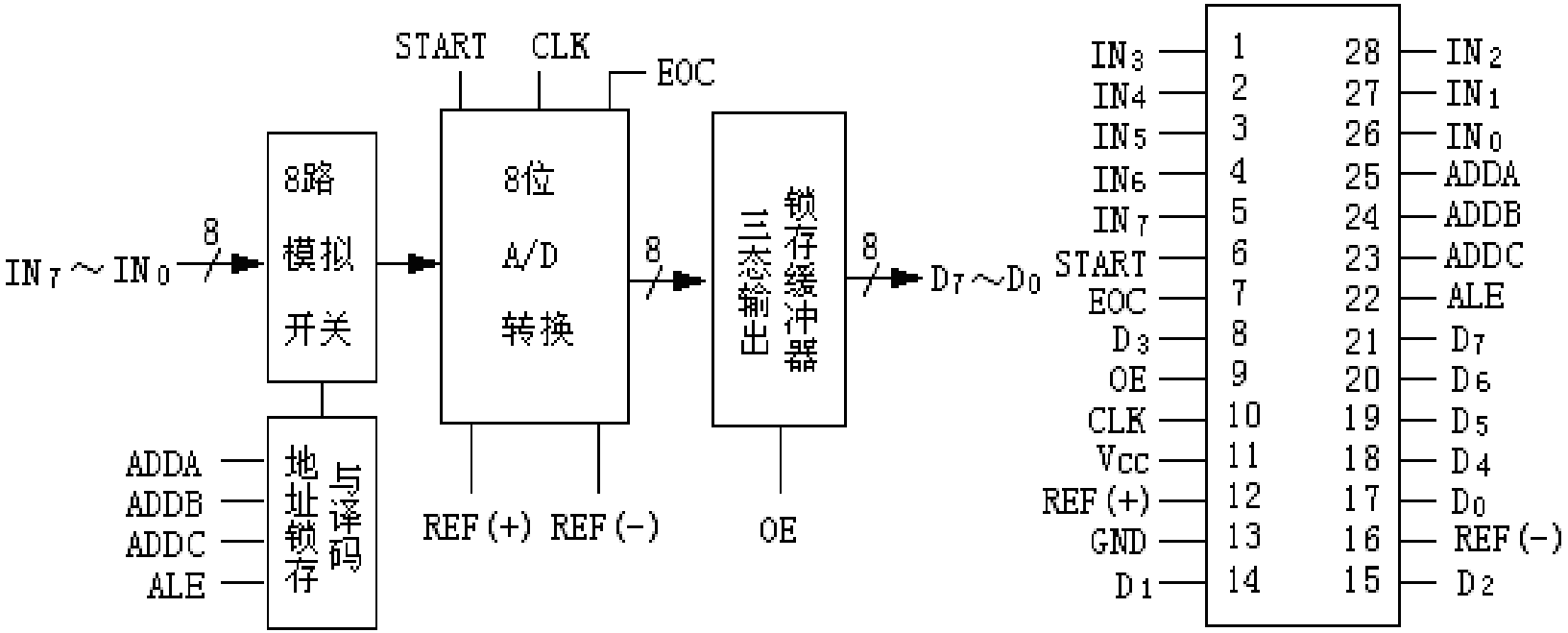
外设控制器与芯片LAN91C111的异步接口电路



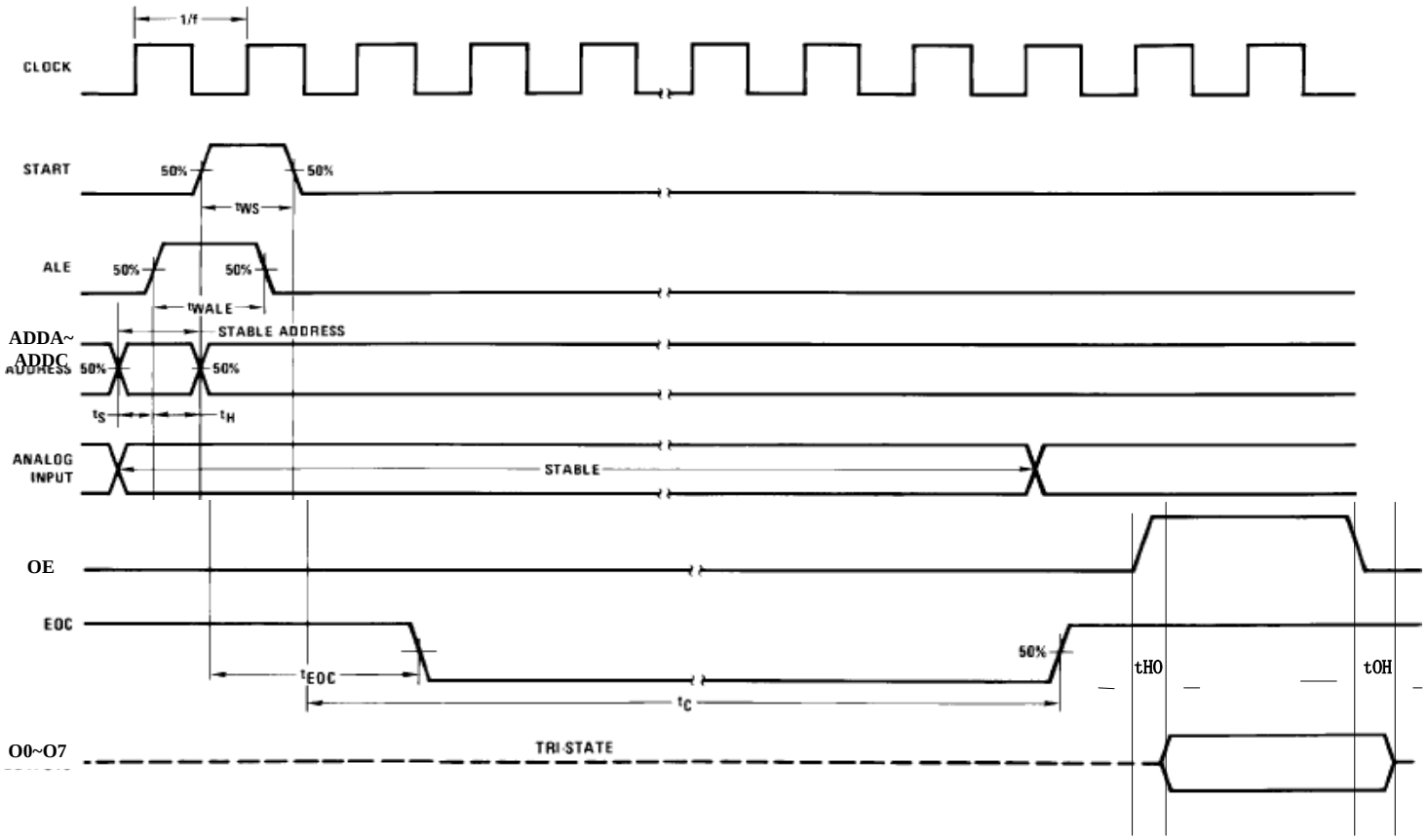
外设控制器总线参数设置

- C_PRH_CLK_SUPPORT = 0
- C_PRH0_AWIDTH = 16
- C_PRH0_DWIDTH = 32
- C_PRH0_SYNC = 0
- C_PRH0_BUS_MULTIPLEX = 0

A/D芯片ADC0808

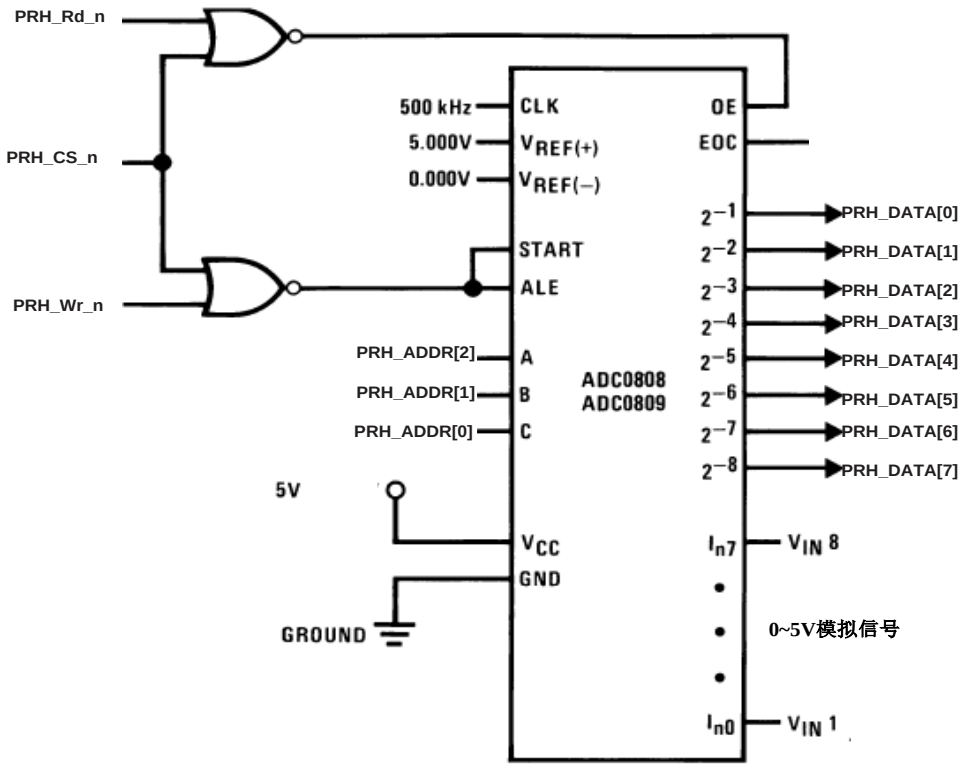


ADC0808时序


1/f
CLOCK
START
50%
50%
tWS
ALE
50%
50%
tWALE
ADDA~ ADDC
STABLE ADDRESS
50%
50%
tS
tH
ANALOG INPUT
STABLE
OE
EOC
50%
tHO
tOH
tEOC
tC
O0~O7
TRI-STATE

延时方式读取AD转换结果的电路原理

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



延时方式下读取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

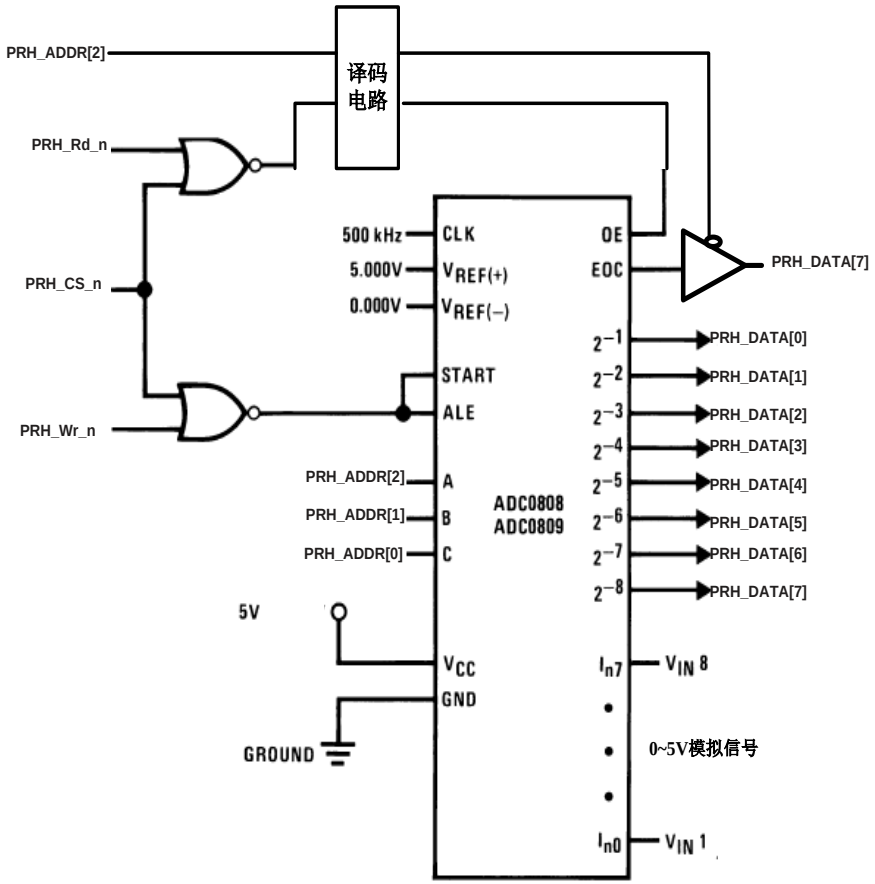
2）延时等待AD转换结束

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;
Xil_Out8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01,0x0);//启动通道1转换
for(j=0;j<10000;j++);//延时等待
data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x00);//读取转换结果
```

查询方式读取AD转换结果的电路原理



查询方式下读取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。

```
unsigned char data;
Xil_Out8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01,0x0);
Do
data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x01);
while((data&0x1)!=0x1);
data=Xil_In8(XPAR_AXI_EPC_0_PRH0_BASEADDR+0x00);
```

作业

- 5，7（作业本提交）
- 12.GPIO控制器，要求采用Nexys4板实验验证（控制程序作业本提交，16周实验课验收）
  - 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七段显示器显示出来。
  - 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.