高等院校电类专业新概念教材 · 卓越工程师教育丛书

项目驱动

——CAN-bus现场总线基础教程

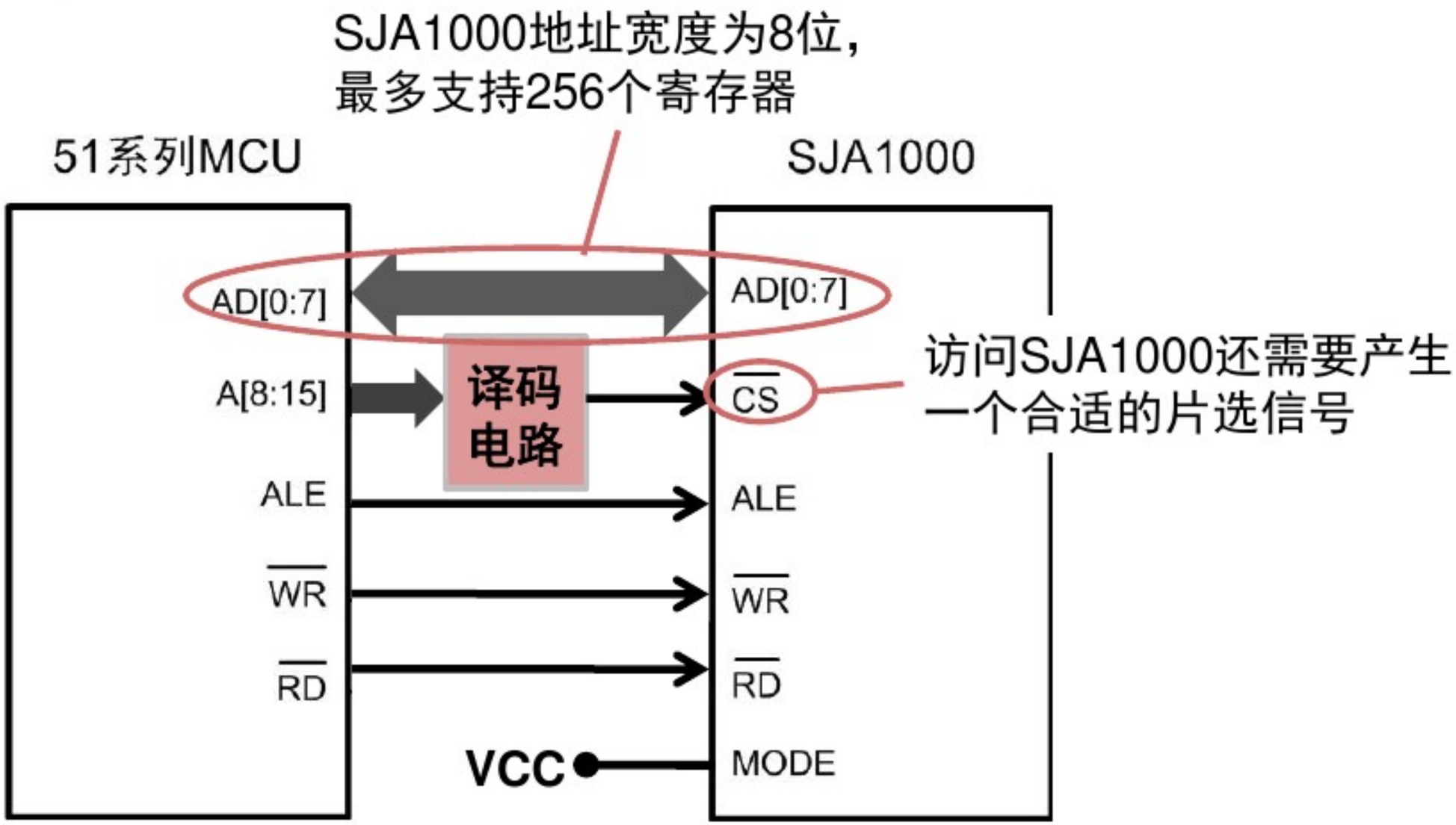
3.1 SJA1000编程基础

广州周立功单片机科技有限公司

目 录

接口电路

SJA1000使用并行接口总线与MCU连接，SJA1000可以认为是一个外扩的RAM。



访问SJA1000的地址决定于低8位与高8位地址，也就是片选信号。

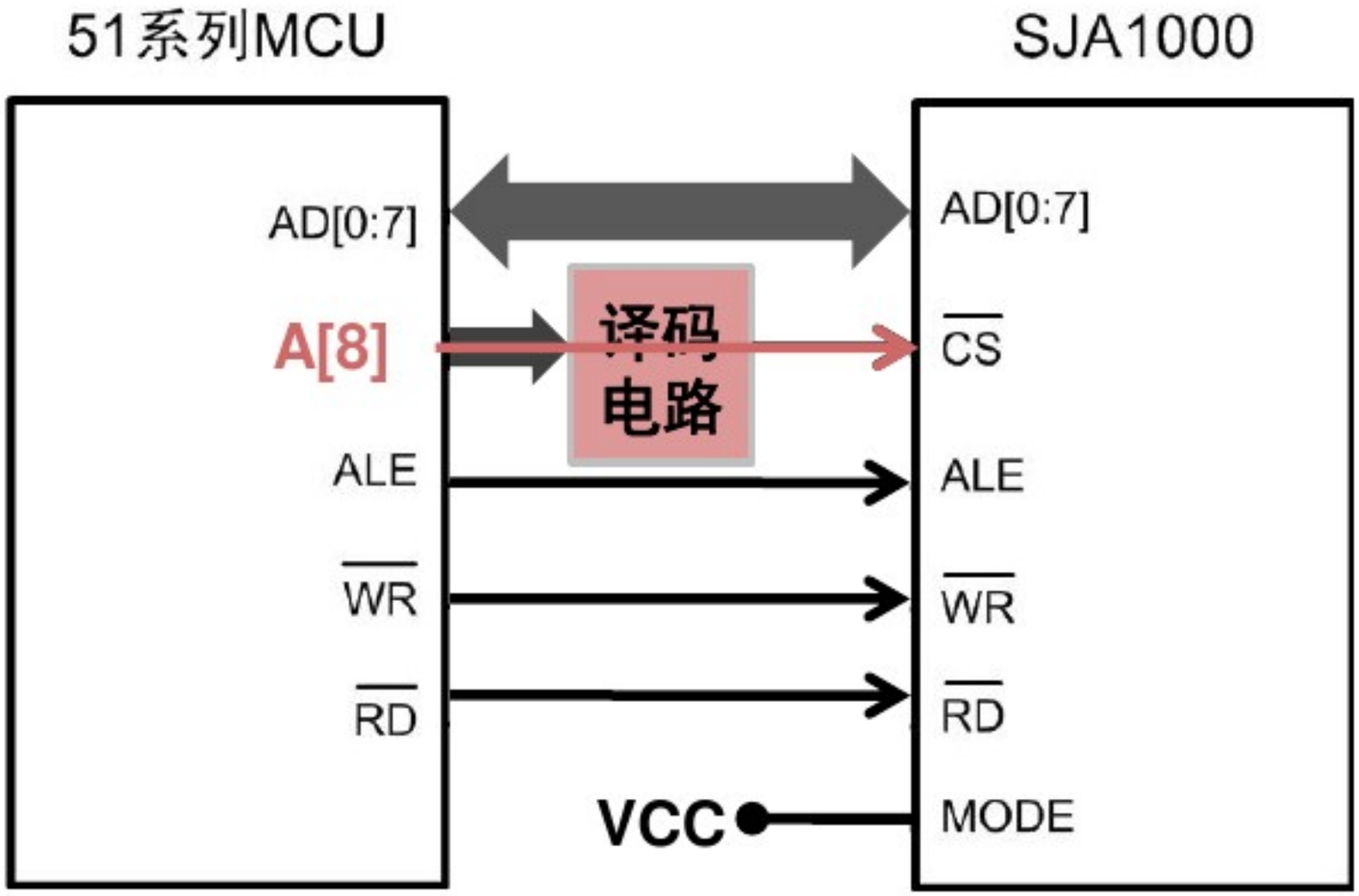
片选信号产生方式

MCU产生片选信号有一下3种：

1. 线选法
2. 全地址译码法
3. 部分地址译码法

线选法

使用空闲的高位地址线作为作为扩展芯片的片选信号。



二进制格式

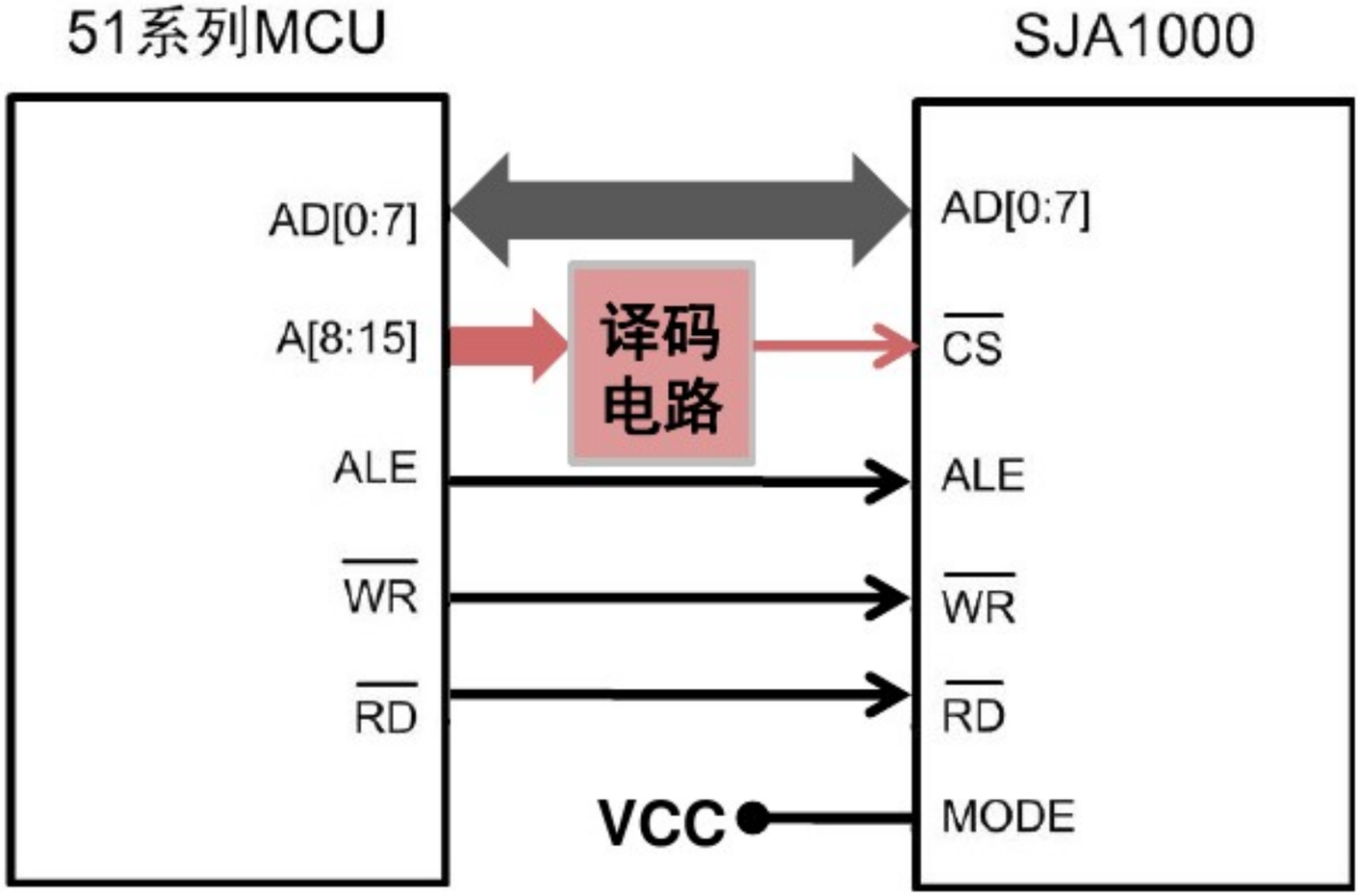
SJA1000的访问基址：A[0:7] = 0x00~0xFF，A[8:15] = 2_0xxxxxxx

优点：无需译码电路，线路简单

缺点：地址空间太多重复，地址空间没有被充分利用

全地址译码法

使用译码器对空闲的高位地址线进行译码，输出的译码信号作为扩展芯片的片选信号。



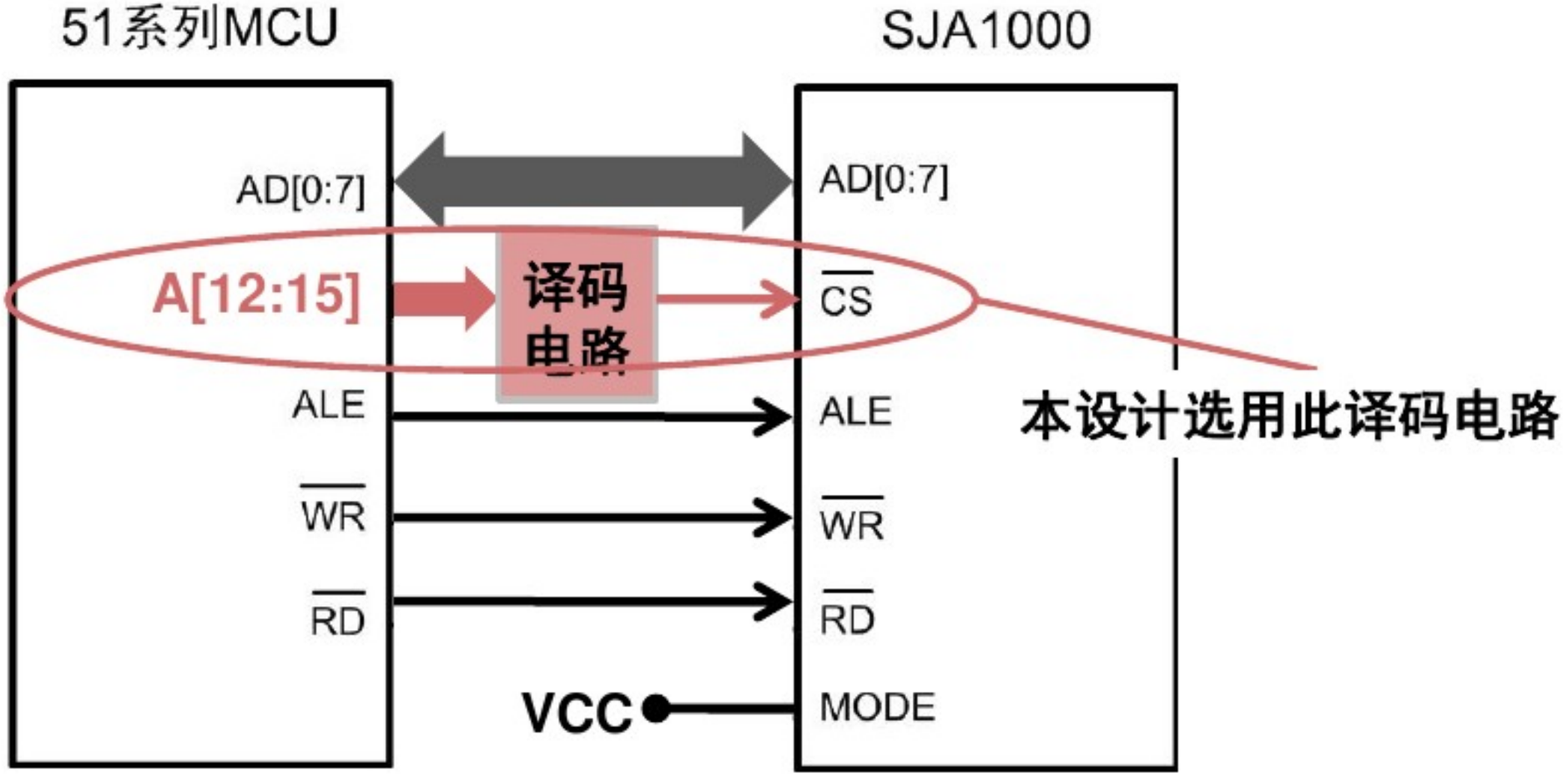
SJA1000的访问基址：具体地址由译码电路决定，地址唯一

优点：地址不重叠，充分利用地址空间

缺点：译码电路较多，线路复杂

部分地址译码法

使用译码器对空闲的部分高位地址线进行译码，输出的译码信号作为扩展芯片的片选信号。

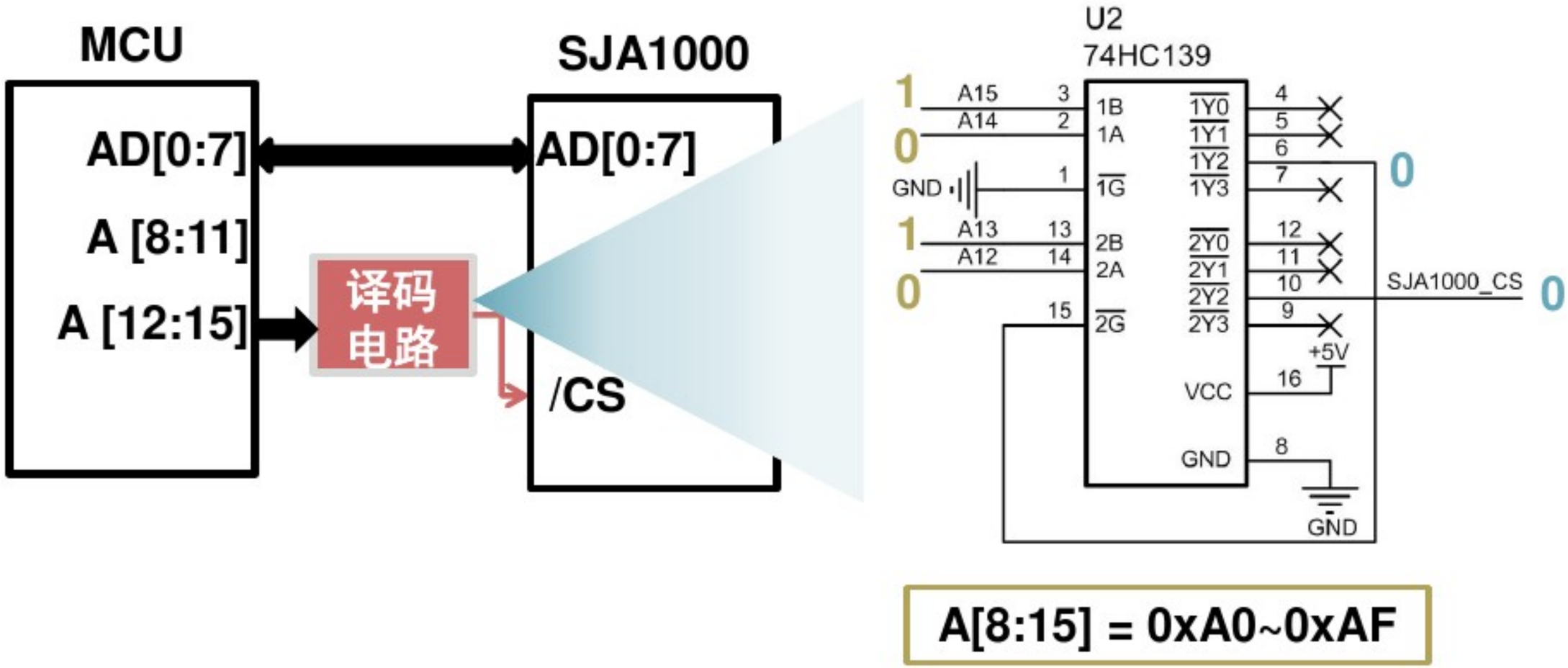


SJA1000的访问基址：具体地址有译码电路决定，地址唯一

优点：地址重叠少，译码电路少

缺点：地址仍有重叠

MCU访问SJA1000的地址



A[8:15] = 0xA0~0xAF

SJA1000的访问基址：0xA000~0xAF00

寄存器地址定义

绝对编址定义寄存器

如果译码电路改变，访问基值改为0xB000，寄存器的定义都需要改变。

```
#define  REG_CAN_MOD     0xB000      //  内部控制寄存器
#define  REG_CAN_CMR     0xB001      //  命令寄存器
#define  REG_CAN_SR      0xB002      //  状态寄存器
#define  REG_CAN_IR      0xB003      //  中断寄存器
```

基址加偏移量定义寄存器

若译码电路改变，访问基值改为0xB000，只需要改变基址的定义。

```
#define  REG_BASE_ADD    0xB000                    //  寄存器基址

#define  REG_CAN_MOD     0x00                      //  内部控制寄存器
#define  REG_CAN_CMR     0x01                      //  命令寄存器
#define  REG_CAN_SR      0x02                      //  状态寄存器
#define  REG_CAN_IR      0x03                      //  中断寄存器
```

实际访问时使用 基址 + 偏址 的方式读写寄存器

访问寄存器的常用操作

1. 对整个寄存器读写操作
2. 对寄存器位进行修改
3. 连续读写多个寄存器

目 录

定义寄存器读写指针

```
#define  REG_BASE_ADDR     0xA000                    //  寄存器基址

xdata unsigned char    *SJA_CS_Point  =
                    (xdata unsigned char *) REG_BASE_ADDR ;
```

定义SJA_CS_Point为指向外部存储器的指针，
通过指针访问**SJA1000**的寄存器

读写寄存器

```
//  写SJA1000寄存器
void WriteSJAReg(unsigned char RegAddr, unsigned char  Value)
{
    *(SJA_CS_Point  + RegAddr) = Value;
    return;
}
```

通过指针向指定地址（SJA1000的寄存器）写入数据

```
//  读SJA1000寄存器
unsigned char ReadSJAReg(unsigned char RegAddr)
{
    return (*(SJA  CS  Point  + RegAddr));
}
```

通过指针向指定地址（SJA1000的寄存器）读取数据

目 录

寄存器位操作

① 设置寄存器位 ：使用或操作

操作示例：设置寄存器第7位 0x0F | 0x80

寄存器0x00	1	0	0	0	1	1	1	1

② 清零寄存器位 ：使用与操作

操作示例：清0寄存器第0位 0x0F & (~0x01)

寄存器0x00	0	0	0	0	1	1	1	0

寄存器位操作

操作流程

读取寄存器当前值

↓

修改指定位

↓

回写寄存器

操作示例代码

```
temp = ReadSJAReg(RegAdr);

temp = temp | BitValue0;
temp = temp & (~BitValue1);

WriteSJAReg(RegAdr, temp);
```

目 录

连续读写寄存器

将ValueBuf[ ]缓存里的数据连续写入多个寄存器

```
……
for (i=0;i<len;i++) {
    WriteSJAReg(RegAdr+i,ValueBuf[i]);
}
……
```

将连续多个寄存器数据读取到ValueBuf[ ]缓存里

```
……
for (i=0;i<len;i++) {
    ReadSJAReg(RegAdr+i,ValueBuf[i]);
}
……
```

目 录

延时函数

延时是程序设计中常用的功能，常规的延时程序结构如下：

循环计数的方式实现软件延时

```
void Delay (unsigend int n)
{
    do {
        for (i=0;i<100;i++);
    } while(--n!=0);
}
```

这种延时程序需要根据实际测试效果调整循环次数才能得到期望的延时时间，且延时时间与MCU平台相关，移植性不好。

精确延时

使用定时器延时

```
//初始化定时器
void timerDelay(void)
{
    TMOD &= ~T0_MASK;      //  设置定时器模式
    TMOD |= 0x01;
}
void timerDelay(unsigned int n)
{
    do {
        TL0 = LOW_BYTE(65535UL-CUPCLK/100);
        TH0 = HIGH_BYTE(65535UL-CPUCLK/100);
        TR0 = 1;
        while(!TF0) {
                TR0 = 0;
                TF0 = 0;
          }
    } while(--n!=0);
}
```

操作流程