```
    aNand_Spare_Data[2] = (u8)((uMecc&0xFF0000)>>16);
    aNand_Spare_Data[3] = (u8)((uMecc&0xFF000000)>>24);
    aNand_Spare_Data[5] = (u8)0xFF;                    //标记有效块

    NF_SECC_UnLock(Controller);

    for(i=0;i<4;i++)
    {
        NF_WRDATA8(Controller, aNand_Spare_Data[i]);//写入剩余阵列(主要是 ECC)
        //NF8_Spare_Data[i]=aNand_Spare_Data[i];
        }

    NF_SECC_Lock(Controller);

    //剩余空间 ECC
    uSecc = Inp32(&NAND(Controller)->rNFSECC) & 0xFFFF;

    aNand_Spare_Data[8] = (u8)(uSecc&0xFF);
    aNand_Spare_Data[9] = (u8)((uSecc&0xFF00)>>8);

    for(i=4 ; i<uSpareSize; i++)
    {
        NF_WRDATA8(Controller, aNand_Spare_Data[i]);   //写入剩余阵列(剩余 ECC  标记)
        //NF8_Spare_Data[i]=aNand_Spare_Data[i];
    }

    NF_CLEAR_RnB(Controller);
    NF_CMD(Controller, 0x10);   //  写入第二个指令
}
```

```
else if(NAND_Inform[Controller].uNandType == NAND_Advanced8bit)
{
    // Main Area ECC
    uMecc = Inp32(&NAND(Controller)->rNFMECC0);

    aNand_Spare_Data[0] = (u8)0xFF;                    //标记有效块
    aNand_Spare_Data[1] = (u8)(uMecc&0xFF);
    aNand_Spare_Data[2] = (u8)((uMecc&0xFF00)>>8);
    aNand_Spare_Data[3] = (u8)((uMecc&0xFF0000)>>16);
    aNand_Spare_Data[4] = (u8)((uMecc&0xFF000000)>>24);

    NF_WRDATA8(Controller, aNand_Spare_Data[0]);   //写入标记有效块

    NF_SECC_UnLock(Controller);
    for(i=1;i<5;i++)
    {
        NF_WRDATA8(Controller, aNand_Spare_Data[i]);//写入剩余阵列  (主要 ECC)
        //NF8_Spare_Data[i]=aNand_Spare_Data[i];
        }
    NF_SECC_Lock(Controller);

    //  剩余 ECC
    uSecc = Inp32(&NAND(Controller)->rNFSECC) & 0xFFFF;

    aNand_Spare_Data[8] = (u8)(uSecc&0xFF);
    aNand_Spare_Data[9] = (u8)((uSecc&0xFF00)>>8);

    for(i=5 ; i<uSpareSize; i++)
    {
```

```
            NF_WRDATA8(Controller, aNand_Spare_Data[i]);   //写入剩余阵列(剩余 ECC  和标记)
            //NF8_Spare_Data[i]=aNand_Spare_Data[i];
        }

        NF_CLEAR_RnB(Controller);
        NF_CMD(Controller, 0x10);   //写入第二个指令
}

while(!g_Nand_RnBTransition)
{
        if(g_Nand_IllegalAccError == 1)
        {
            NAND_Reset(Controller);
            g_Nand_IllegalAccError = 0;
            return eNAND_ProgramError;
        }
}

//Read Program Status
NF_CMD(Controller, 0x70);

while(!((uStatus = NF_RDDATA8(Controller)) & (1<<6)));
if(uStatus & 0x01)
        eError = eNAND_ProgramError;          //  执行中错误
else
        eError = eNAND_NoError;

NF_nFCE_H(Controller);
```

```
    uTemp = NAND_ReadNFCONRegister(Controller);
    uTemp &= ~((1<<10) | (1<<9));          // 非法访问 RnB 转换中断无效
    Outp32(&(NAND(Controller)->rNFCONT), uTemp);

    if(NAND_Inform[Controller].uECCtest == 1)
    {
        for(i=0 ; i<uSpareSize ; i++)
            aNand_Spare_Data_Temp[i] = aNand_Spare_Data[i];
    }

    for(i=0 ; i<uSpareSize ; i++)
    {
*aSpareBuffer++ = aNand_Spare_Data[i];
    }
    INTC_Disable(NUM_NFC);
    return eError;
}
```

9 CF 控制器

下面主要介绍 CF 控制器在 S3C6410X RISC 微处理器中的功能及应用。

CF 控制器只有一个插槽。CF 控制器由两部分组成，分别是 PC 卡存储器和 ATA 控制器。在实际应用中只能选择一种模式，PC 卡存储器模式或者 Ture-IDE 模式。CF 控制器有一个顶级的 SFR，其中包括了卡电源启用位、输出端口使能位和模式选择(Ture-IDE 或 PC 卡)位。

9.1. PC 卡控制器的性能

PC 卡控制器有 2 个半字大小的写缓冲区和 4 个半字大小的读缓冲区。

PC 卡控制器有 5 个字尺寸大小的特殊功能寄存器：其中 3 个字为时序配置寄存器；一个字为状态控制配置寄存器；另一个字为中断源和屏蔽寄存器。

时序配置寄存器由三部分组成：安装程序、命令和运行。PC 卡接口包括 IDLE、SETUP、COMMAND、HOLD 四个区域，寄存器的不同部分指明了不同区域的运行时间。

9.2. ATA 控制器的性能

（1）.ATA 控制器与 ATA 标准相兼容。

（2）.ATA 控制器有一个 16x32 位的 FIFO。

（3）.ATA 控制器有内部 DMA 控制器（由 ATA 设备到内存或由内存到 ATA 设备）

（4）.AHB （DMA 控制器）支持 8 个突发字转换。

（5）.DMA 控制器支持直接模式和间接模式

直接模式：只支持 UDMA 模式。

间接模式：支持 I/O 模式、存储器模式、True-IDE 模式（除了 UDMA 模式）。

9.3. I/O 描述

表 9-1 I/O 信号描述

间接模式	直接模式 （UDMA 模式）	I/O	描述
XmOCSn[4]	XhiCSn XhiADR[8]	输出	卡使能触发 PC 卡模式：低字节使能触发 Ture-IDE 模式：芯片选择（nCS0）
XmOCSn[5]	XhiCSn-main XhiADR[9]	输出	卡使能触发 PC 卡模式：高字节使能触发 Ture-IDE 模式：芯片选择（nCS1）
XmOREGate	XmOREGate XhiADDR[6]	输出	CF 卡内触发寄存器 PC 卡模式：用来访问 CF 卡内寄存器 Ture-IDE 模式：DMA 应答
XmOOEata	XmOOEata	输出	输出使能触发 PC 卡模式：存储器的输出使能触发 Ture-IDE 模式：GND 地
XmORESETata	XmORESETata XhiADDR[4]	输出	CF 卡复位 PC 卡模式：高电平有效 Ture-IDE 模式：低电平有效
XmOWEata	XmOWeata	输出	写使能触发 PC 卡模式：存储器输出使能触发 Ture-IDE 模式：VCC 电源
XmOOEn	XhiCSn_sub XhiARD[10]	输出	I/O 模式读触发 UDMA 模式：主触发
XmOOEn	XhiWEn XhiARD[110]	输出	I/O 模式写触发
XmOADDR[0]	XmOADDR[0]	输出	CF 卡 地址

	XuRXD[2] XmmcDATA1[4]		PC 卡模式：所有地址有用 Ture-IDE 模式：只有 ADDR[2:0]有用
XmOADDR[1]	XmOADDR[1] XuRXD[2] XmmcDATA1[5]	输出	其他地址与地连接
XmOADDR[2]	XmOADDR[2] XmmcDATA1[6] XuRXD[3]	输出	
XmOADDR[3]		输出	
XmODATA[15:0]	XhiDATA[0]	B	CF 数据总线
	XhiDATA[1]	B	
	XhiDATA[2]	B	
	XhiDATA[3]	B	
	XhiDATA[4]	B	
	XhiDATA[5]	B	
	XhiDATA[6]	B	
	XhiDATA[7]	B	
	XhiDATA[8] （XhiDATA[16]）	B	
	XhiDATA[9] （XhiDATA[17]）	B	
	XhiDATA[10] （XhiCSn）	B	
	XhiDATA[11] （XhiCSn_main）	B	
	XhiDATA[12] （XhiCSn_sub）	B	
	XhiDATA[13] （XhiWE）	B	

	XhiDATA[14] （XhiOEn）	B	
	XhiDATA[15] （XhiRQn）	B	
XmOCData	XmOCData	输入	卡检测信号
	XhiADDR[7]		
XmOINTata	XmOINTata	输入	CF 卡的中断请求 PC 卡模式：低电平有效（存储器模式：水平触发； I/O 模式：边沿触发） Ture-IDE 模式：高电平有效
	XhiADDR[3]		
XmOWAITn	XhiADR[12]	输入	CF 卡等待信号 UDMA 模式：设备触发
	XhiOEnl		
XmOINPACKata	XmOINPACKata	输入	I/O 模式输入响应： PC 卡模式：还未应用 True-IDE 模式：DMA 请求
	XhiADDR[5]		

9.4. 模块图

CF 控制器模块图如下图 9-1 所示。

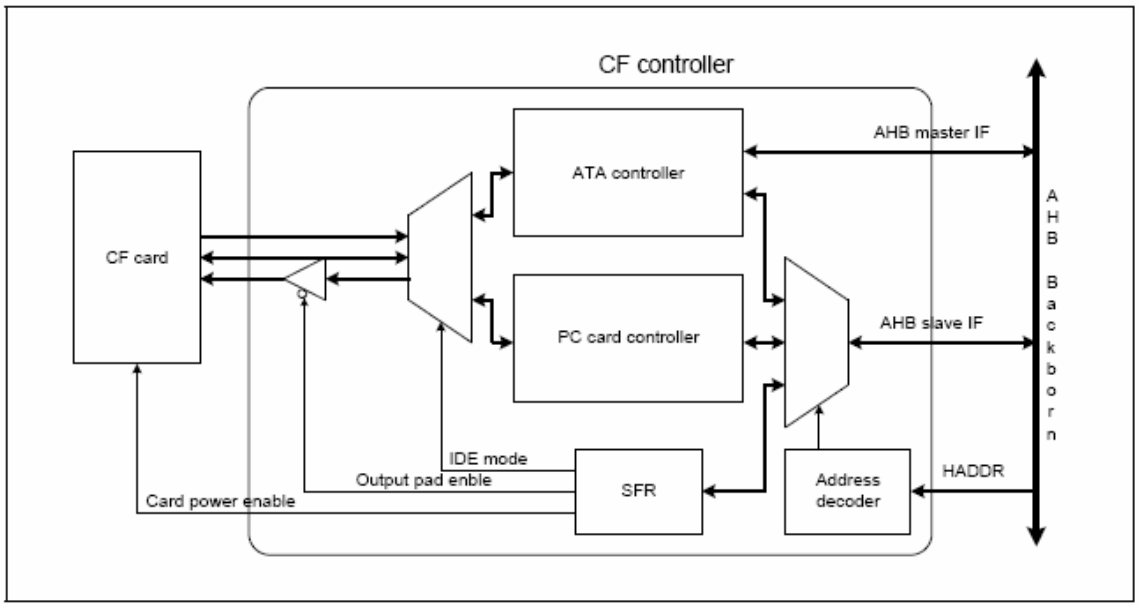

图 9-1 CF 控制器模块图

9.5.时序图

9.5.1.PC 卡模式

PC 卡模式时序图如图 9-2 所示。

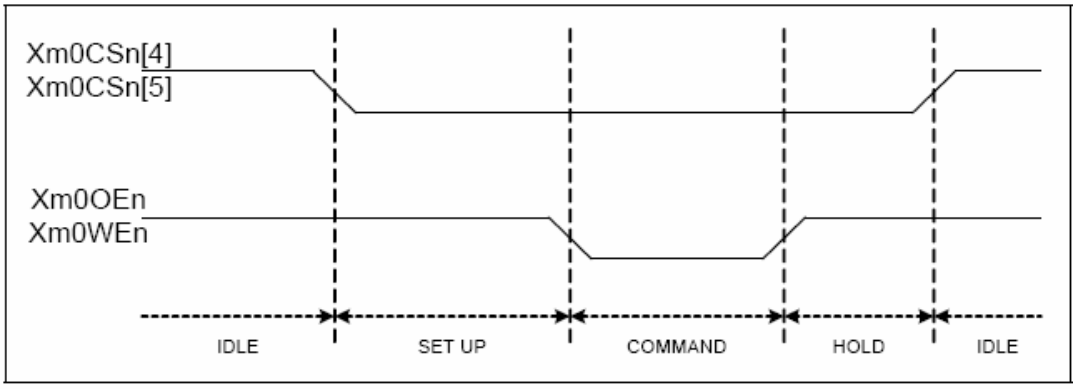

图 9-2 PC 卡模式状态定义

PC 卡模式的时序参数见表 9-2

表 9-2 PC 卡模式的时序参数

Area	Attribute memory	I/O interface	Common memory
	(min, Max) nS		
Set up	(30, --)	(70, --)	(30, --)
Command	(150, --)	(165, --)	(150, --)
Hold	(30, --)	(20, --)	(20, --)
S + C + H	(300, --)	(290, --)	(--, --)

9.5.2. Ture-IDE 模式

PIO 模式波形如图 9-3 所示

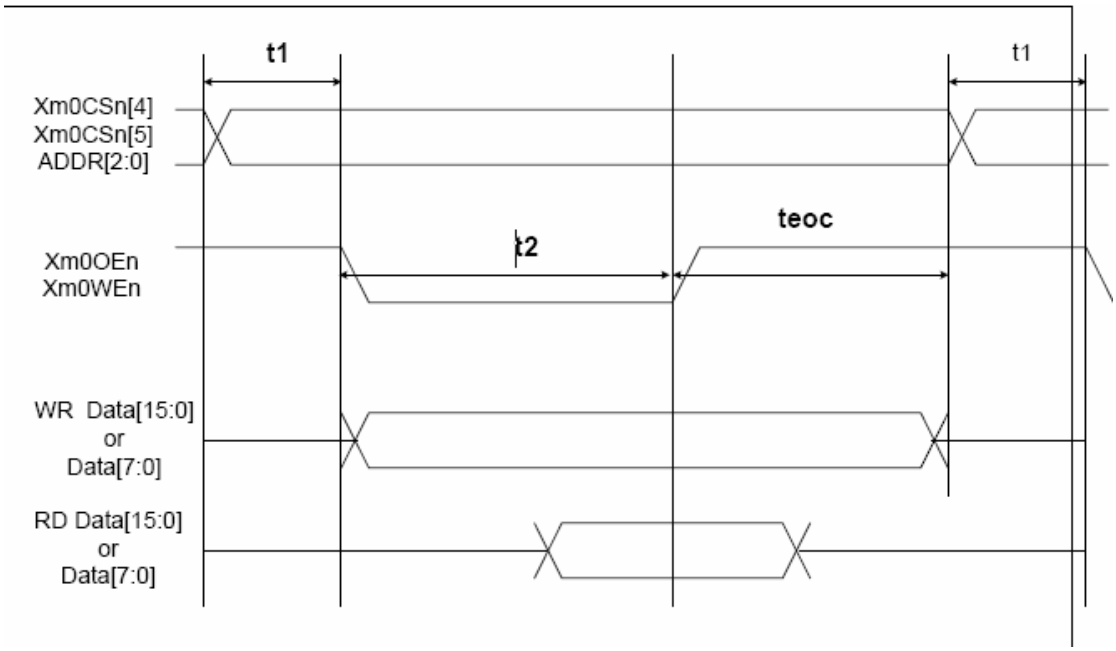

图 9-3 PIO 模式波形

PIO 模式时序参数见表 9-3。