```
极限:1clock)
    uPara_RFC = (((g_HCLK /1000 *eDDR_RFC) -1)/1000000 + 1);
    if (uPara_RFC <4)
        {
            uScheduled_Para = 3;
        }
    else
        {
            uScheduled_Para = uPara_RFC;
        }
    uPara_RFC = ((uScheduled_Para-3)<<3) | (uPara_RFC);          // 自动刷新到 cmd 时间(>=t_RC), (最小:80ns, 11@133MHz)
    Outp32(&DMC(uController)->rT_RFC, uPara_RFC);
    uPara_RP = (((g_HCLK /1000 *eDDR_RP) -1)/1000000 + 1);
    if (uPara_RP <4)
        {
            uScheduled_Para = 3;
        }
    else
        {
            uScheduled_Para = uPara_RP;
        }
    uPara_RP = ((uScheduled_Para-3)<<3) | (uPara_RP);
    Outp32(&DMC(uController)->rT_RP, uPara_RP);          // 预加电 RAS 延迟(行预加电时间) (最小:22.5ns, 4@133MHz, 极限:1clock)
    uPara_RRD = (((g_HCLK /1000 *eDDR_RRD) -1)/1000000 + 1);
    Outp32(&DMC(uController)->rT_RRD, uPara_RRD);      // 激活 bank x 到激活 bank y 延迟(最小:15ns, 3@133MHz, 极限:1clock)
    uPara_WR = (((g_HCLK /1000 *eDDR_WR) -1)/1000000 + 1);
```

```
    Outp32(&DMC(uController)->rT_WR, uPara_WR);         // 写入预加电延迟(最小:15ns, 3@133MHz,极限:1clock)
//  uPara_WTR= (((g_HCLK /1000 *eDDR_RP) -1)/1000000 + 1) + 2;
    uPara_WTR = 2;
    Outp32(&DMC(uController)->rT_WTR, uPara_WTR);           // 写到读延迟
    Outp32(&DMC(uController)->rT_XP, 2);                    // 脱离掉电 cmd 时间
    uPara_XSR= (((g_HCLK /1000 *eDDR_XSR) -1)/1000000 + 1) ;
    Outp32(&DMC(uController)->rT_XSR,uPara_XSR);        // 脱离自刷新 cmd 时间
    Outp32(&DMC(uController)->rT_ESR,uPara_XSR);        // 自刷新 cmd 时间

        if(uController == 0)
    {
        // 存储配置寄存器
        Outp32(&DMC(uController)->rMEMCFG,
            (0<<21) |       // 1 片
            (0<<18) |       // 通过 ARID[3:0]进行 Qos 主控器选择
            (2<<15) |       // 突发脉冲 4
            (eStopClk<<14) |        // 停止存储器时钟无效
            (eAutoPD<<13) |         // 自动掉电无效
            (uPDprd<<7)     |       // 自动掉电周期
            (eAP_bit<<6) |          // 位 10 自动加电位
            (eRow_bit<<3)       |           // 13 位列位
            (eCol_bit<<0) );        // 10 位列位

        // 存储配置寄存器 2
        Outp32(&DMC(uController)->rMEMCFG2,
            (1<<11) |       // Read delay 1 cycle from the Pad I/F ( 0x1 => mDDR)
            (3<<8)      |           // Memory Type (mDDR)
            (0<<6) |                // Width ( 16bit)
```

```
            (0<<4) |            // Bank bits = 2bit
            (0<<2) |            // DQM state
            (1<<0) );       // Sync. clock scheme

        // CHIP0 配置
        Outp32(&DMC(uController)->rCHIP_0_CFG,   0x140fc);
0x4000_0000 ~ 0x43ff_ffff ( 64MB)

    }
    else if(uController == 1)
    {
        //  存储配置寄存器
        Outp32(&DMC(uController)->rMEMCFG,
            (0<<31) |       //  无效个别 CKE 控制
            (0<<21) |       // 1 片
            (0<<18) |       // ARID[3:0]的 Qos 主控器选择
            (2<<15) |       //  脉冲 4
            (eStopClk<<14) |        //  无效存储器时钟( 0)
            (eAutoPD<<13) |         //  无效自动掉电(0)
            (uPDprd<<7)     |       //  自动掉电周期
            (eAP_bit<<6) |          //  位 10 的自动预加电位
            (eRow_bit<<3)|          // 13 位行位
            (eCol_bit<<0) );        // 10 位列位

        //  存储配置寄存器 2
        Outp32(&DMC(uController)->rMEMCFG2,
            (1<<11) |                       //  读取从 Pad I/F ( 0x1 => mDDR)延迟一个周期
            (3<<8) |                    //  存储器类型(mDDR)
            (1<<6) |                        //  宽度  ( 32 位)
```

```
            (0<<4) |                    // Bank 位 = 2 位
            (0<<2) |                    // DQM 状态
            (1<<0) );                   // 同步时钟配置

        // CHIP0 配置
        Outp32(&DMC(uController)->rCHIP_0_CFG,  0x150f8);
0x5000_0000 ~ 0x57ff_ffff ( 128MB)
    }

    // 外部存储器初始化

        // NOP
        Outp32(&DMC(uController)->rDIRECTCMD,
            (0<<20) |       //片地址 - 片 0
            (3<<18) );          //指令- NOP

        // 预加电
        Outp32(&DMC(uController)->rDIRECTCMD,
            (0<<20) |       // 片地址 - Chip 0
            (0<<18)  );         // 指令- PALL

        // 自动刷新两次
        Outp32(&DMC(uController)->rDIRECTCMD,
            (0<<20) |       //片地址 - 片 0
            (1<<18) );          // 指令 – 自动刷新

        Outp32(&DMC(uController)->rDIRECTCMD,
            (0<<20) |       //片地址 - 片 0
            (1<<18) );          // 指令- 自动刷新
```

```
    // MRS
    Outp32(&DMC(uController)->rDIRECTCMD,
        (0<<20) |       // 片地址 – 片 0
        (2<<18) |       // 指令- MRS/EMRS
        (0<<16) |       // [17:16] - MRS ( 0 )
        (0x32<<0));         //   [6:4]- CAS 等待 - 3,
                            // [3] –脉冲类型 (连续), [2:0] – 脉冲长度(4) -> 8

    // EMRS
    Outp32(&DMC(uController)->rDIRECTCMD,
        (0<<20) |       // 片地址   - 片 0
        (2<<18) |       // 指令- MRS/EMRS
        (2<<16) |       // [17:16] - EMRS (2)
        (0x0<<0));


    //'GO' 模式
    Outp32(&DMC(uController)->rMEMCCMD, (0x0<<0));   // Go 指令

    // 检查控制器状态
    while((Inp32(&DMC(uController)->rMEMSTAT))&0x3 == 1    );


}
```

6　SROM 控制器

本小节主要介绍 SROM 控制器的功能及使用。S3C6410 SROM 控制器(SROMC)支持外部 8,16 位 NOR Flash，PROM，SRAM 存储器。

6.1 SROM 控制器的特性

S3C6410 SROM 控制器的特性包括：

（1）支持 SRAM，不同的 ROM 和 NOR flash 存储器。

（2）支持仅 8 或 16 位数据总线。

（3）地址空间：每页高达 128MB。

（4）支持 6 页。

（5）固定内存页开始地址。

（6）外部等待扩展总线周期。

（7）支持字节和半字存取的外部存储器。

如图 6-1 所示，显示了 SROM 控制器的结构框图。

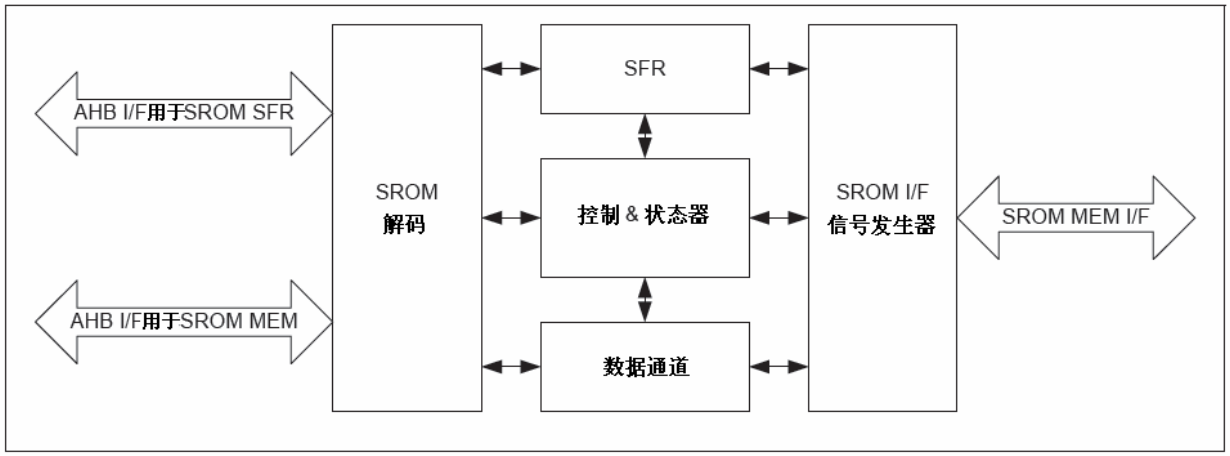

图 6-1 SROM 控制器框图

6.2 SROM 控制器功能描述

SROM 控制器支持从 Bank0～Bank5 的 SROM 接口。在 OneNAND 启动情况中，SROM 控制器不能控制 Bank2 和 Bank3，因为它的主要身份是在 OneNAND 控制器上。在 NAND 引号例子中，SROM 控制器同样也不能控制 Bank2 和 Bank3，因为它的控制是在 NAND Flash 控制器上。

6.2.1.nWAIT 引脚操作

如果 WAIT 操作相应的每个存储块都被使能，nOE 持续时间将通过外部 nWAIT 引脚被延长，存储块被激活。从 tacc-1 中检测 nWAIT。在取样 nWAIT 为高后，下一个时钟 nOE 将低有效。new 信号和 nOE 信号有同样的关系。SROM 控制器 nWAIT 的时序框图，如图 6-2 所示。

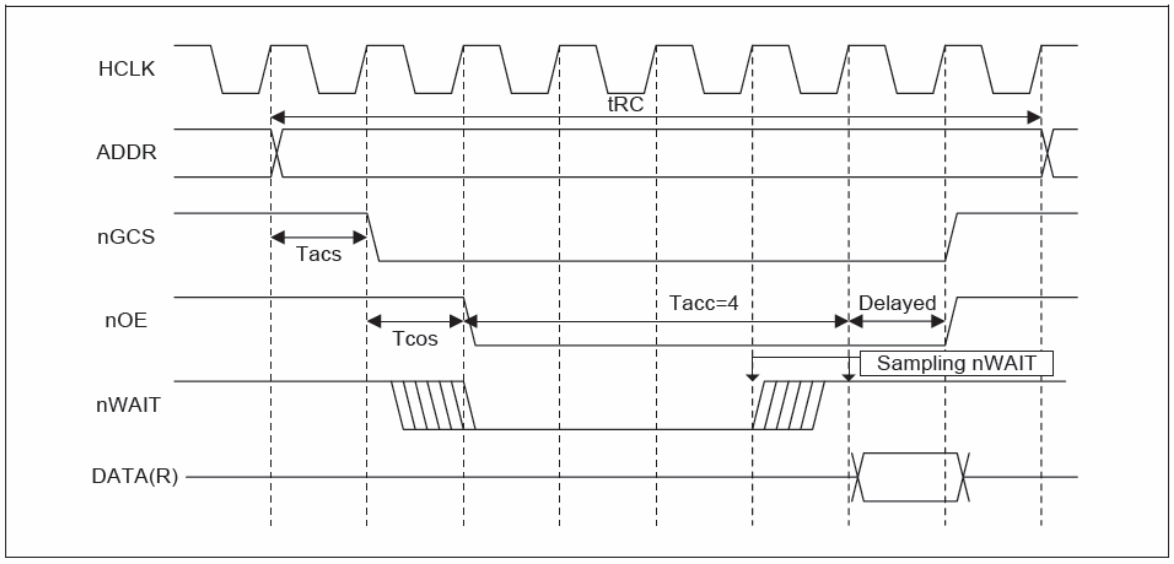

图 6-2 SROM 控制器 nWAIT 时序框图

6.2.2.可编程访问周期

如图 6-3 所示，描述的是 SROM 控制器读时序的周期框图，如图 6-4 所示，描述的是 SROM 控制器写时序的周期框图。

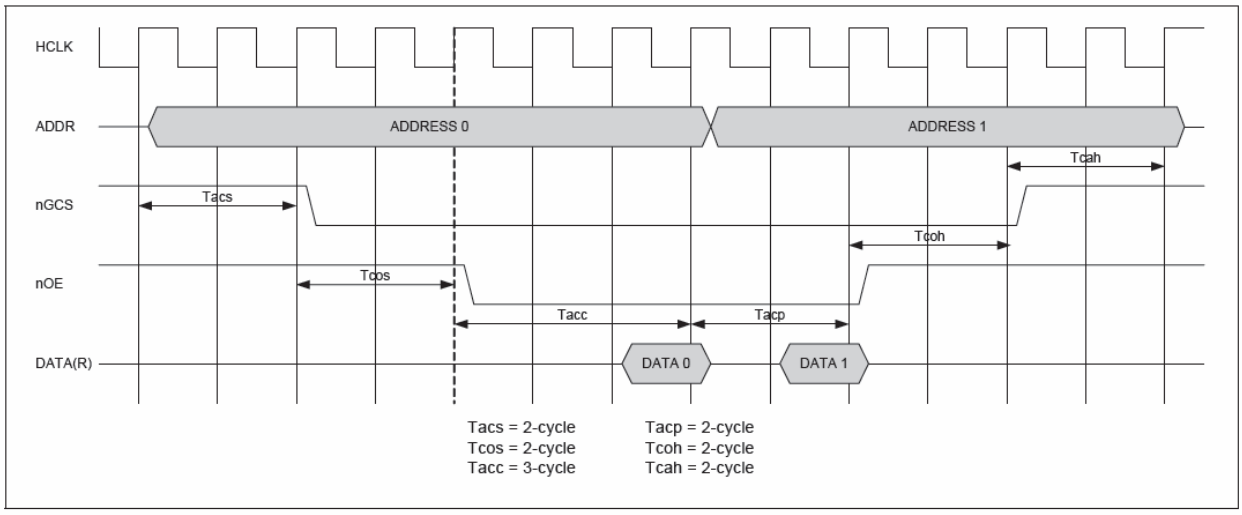

图 6-3 SROM 控制器读时序框图

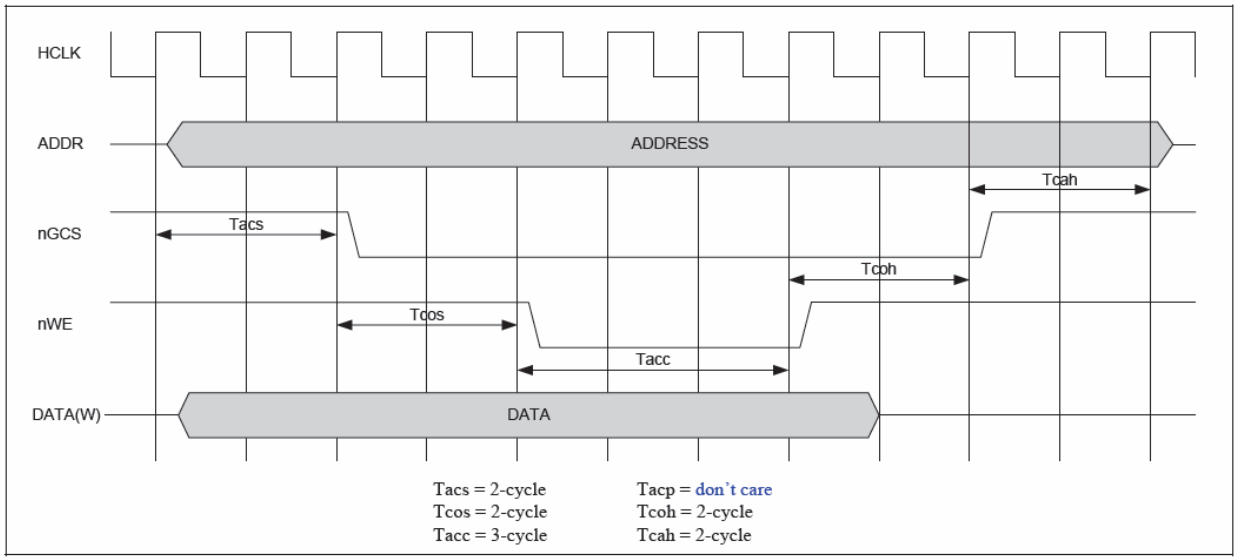

图 6-4 SROM 控制器写时序框图

注：页模式仅支持读周期。

6.2.3.SROMC 块设置代码实现

以下是 S3C6410 SROM 控制器（SROMC）在 ARM11 中的代码定义，参照如下：

```
// SROMC_SetBank 函数：主要功能初始化 SROMC 块
//输入：uBank：选择块数
//        eByteCTL：选择块 UL/BL 控制
//        eWAITCTL：选择块 WAIT 控制
//         eDWidth：选择块数据宽度控制
//         ePage：选择块页模式控制
//         eTacs：选择块 Tacs 控制
//         eTcos：选择块 Tcos 控制
//         eTacc：选择块 Tacc 控制
//         eTcoh：选择块 Tcoh 控制
//         eTcah：选择块 Tcah 控制
//         eTacp：选择块 Tacp 控制
// 输出 ：
void SROMC_SetBank(u8 uBank, Byte_eCTL eByteCTL, WAIT_eCTL eWAITCTL, Data_eWidth eDWidth,
                   Page_eMode ePage,Bank_eTiming eTacs, Bank_eTiming eTcos,Bank_eTiming eTacc,
                   Bank_eTiming eTcoh, Bank_eTiming eTcah, Bank_eTiming eTacp)
{
    u32 uBaseAddress = 0;
    u32 uConValue    = 0;
    volatile u32 *pSROMC_BC_Addr = NULL;
    volatile u32 *pSROMC_BW_Addr = NULL;

    uBaseAddress = SROM_BASE;
    g_SROMCBase = (void *)uBaseAddress;
```

```
    // 总线宽度& 等待控制
    pSROMC_BW_Addr = &(SROMC->rSROM_BW);
    uConValue = Inp32(&SROMC->rSROM_BW);
    uConValue = (uConValue & ~(0xF<<(uBank*4))) | (eDWidth<<(uBank*4))|
                (eWAITCTL<<(uBank*4 + 2))|(eByteCTL<<(uBank*4+3));
    *pSROMC_BW_Addr = uConValue;

    //块控制寄存器
    pSROMC_BC_Addr = &(SROMC->rSROM_BC0);
    pSROMC_BC_Addr = pSROMC_BC_Addr + uBank;
    uConValue = (eTacs<<28)|(eTcos<<24)|(eTacc<<16)|(eTcoh<<12)|(eTcah<<8)|(eTacp<<4)
                |(ePage<<0);
    *pSROMC_BC_Addr = uConValue;
}
```

6.3 特殊功能寄存器描述

6.3.1. SROM 总线宽度&等待控制寄存器(SROM_BW)

寄存器	地址	读/写	描述	复位值
SROM_BW	0x70000000	读/写	SROM 总线宽度和等待控制	0x0000_000x

SROM_BW	位	描述	初始状态
Reserved	[31:24]	保留	0
ByteEnable5	[23]	nWBE / nBE(用于UB/LB) 控制，用于存储器Bank5 0 = 不使用UB/LB (XrnWBE[1:0] 专门的nWBE[1:0]) 1 = 使用 UB/LB (XrnWBE[1:0] 专门的 nBE[1:0]	0