图 3-17　睡眠模式下的电源域

睡眠模式进入顺序如下：

1） 在 SLEEP 模式下，用户软件设置 PWR_CFG[6:5]。

2） 通过 MCR 指令（MCR p15, 0, Rd, c7, c0, 4），用户软件生成 STANDBYWFI 信号。

3） SYSCON 请求总线控制器，以完成当前 AHB 总线的处理。

4） 当前总线的处理完成后，AHB 总线控制器发送确认信息到 SYSCON。

5） SYSCON 请求 DOMAIN-V 去完成当前 AXI 总线的处理。

6） 当前总线处理完成后，AXI 总线控制器发送确认信息到 SYSCON。

7） SYSCON 请求外部存储控制器进入到自刷新模式，在 SLEEP 模式期间外部内存的内容必须要保存起来。

8） 自刷新模式时，存储控制器发送确认信息。

9） 如果 PLL 被使用， 则 SYSCON 改变时钟源从 PLL 输出到外部振荡器。

10）SYSCON 禁用 PLL 操作和晶体振荡器。

11）最后，通过声明 XPWRRGTON 引脚到低电平状态，SYSCON 禁用外部时钟源用于内部逻辑。XPWRRGTON 信号控制外部调节器。

其退出顺序如下：

1） SYSCON 启动外部电源，通过声明 XPWRRGTON 引脚到高电平状态，通过 PWR_STABLE 配置等待稳定时间。

2） SYSCON 生成系统时钟，包括 HCLK，PCLK 和 ARMCLK。

3） SYSCON 释放系统复位信号，包括 HRESETn 和 PRESETn。

4） 如果启动设备是 NAND，则 NFCON 从外部 NAND 设备到 stepping stone 复制启动代码。(XOM[4:3] == 2’b00)

5） SYSCON 释放 ARM 复位信号。

6. 唤醒

如表 3-4 所示，说明了从低功耗状态，IDLE，(DEEP)-STOP 和 SLEEP 模式中的各种唤醒源。根据低功耗状态，可用不同的唤醒源。

表 3-4 唤醒源的电源模式

电源模式	唤醒源
	所有中断源

			MMC0, MM1, MMC2
			TS ADC
闲置模式	停止模式	睡眠模式	外部中断源
			RTC 警告
			TICK
			键盘中断
			MSM (MODEM)
			电池故障
			HSI
			温复位

7. 复位

S3C6410 有五种类型的复位信号，SYSCON 可以把系统的五分之一进行复位。

- 硬件复位：它是通过声明 XnRESET 产生的。它可以完全初始化所有系统。
- 温复位：它是通过 XnWRESET 产生的。当需要初始化 S3C6410 和保存当前硬件状态时，XnWRESET 被使用。
- 看门狗复位：它是通过一个特殊的硬件模块产生的，也就是看门狗定时器。当系统发生一个不可预测的软件错误时，硬件模块监控内部硬件状态，同时产生复位信号来脱离该状态。
- 软件复位：它是通过设置 SW_RESET 产生的。
- 唤醒复位：它是当 S3C6410 从睡眠模式唤醒时产生的。睡眠模式后，内部硬件状态在任何时候都不可用，必须对其进行初始化。

硬件复位

当 XnRESET 引脚被声明，系统内的所有单元（除了 RTC 之外）复位到预先定义好的状态时，硬件复位被调用。在这段期间，将发生下面的动作：

- 所有内部寄存器和 ARM1176 内核都到预先定义好的复位状态。
- 所有引脚都得到它们的复位状态。
- 当 XnRESET 被声明的同时，XnRSTOUT 引脚就被声明了。

XnRESET 是不被屏蔽的，始终保持使能状态。XnRESET 的声明，无论先前为何模式，S3C6410 都进入复位状态。XnRSET 必须持有足够长的时间允许内部稳定和传播。

S3C6410 的电源调节器必须预先稳定到 XnRESET.的 deassertion 状态。否则，它可能会损害 S3C6410，发生不可预测的操作。

温复位

当在正常，闲置和停止模式下，为了超过 100ns，XnWRESET 引脚被声明时，温复位被调用。在睡眠模式下，它是作为一个唤醒事件被处理的。如果 XnBATFLT 保持低电平或系统处于唤醒时期，则 XnWRESET 被忽略。如图 3-18 所示，所有寄存器除了 SYSCON，RTC 和 GPIO 都被初始化。

模块	寄存器	XnRESET	XnWRESET	Watchdog	Wakeup from SLEEP	Software
SYSCON	PWR_CFG, EINT_MASK, NORMAL_CFG, STOP_CFG, SLEEP_CFG, OSC_FREQ, OSC_STABLE, PWR_STABLE, FPC_STABLE, MTC_STABLE, OTHERS, RST_STAT, WAKEUP_STAT, BLK_PWR_STAT, INFORM0, INFORM1, INFORM2, INFORM3	X	X	X	X	O
RTC	RTCCON, TICCNT, RTCALM, ALMSEC, ALMMIN, ALMHOUR, ALMDAY, ALMMON, ALMYEAR, RTCRST	X	X	X	X	O
GPIO	GPICONSLP, GPIPUDSLP, GPJCONSLP, GPJPUDSLP, GPKCON0, GPKCON1, GPKDAT, GPKPUD, GPLCON0, GPLCON1, GPLDAT, GPLPUD, GPMCON, GPMDAT, GPMPUD, GPNCON, GPNDAT,GPNPUD, GPOCON, GPOPUD, GPPCON, GPPPUD, GPQCON, GPQPUD, EINT0CON0, EINT0CON1, EINT0FLTCON0, EINT0FLTCON1, EINT0FLTCON2, EINT0FLTCON3, EINT0MASK, EINT0PEND, SPCONSLP, SLPEN	X	X	X	X	O
其他	-	O	O	O	O	O

图 3-18　寄存器初始化的各种复位

在温复位期间，将发生以下的动作：

- 所有模块除了 ALIVE 和 RTC 模块之外，都到预先定义好的复位状态。
- 所有引脚进入复位状态。
- 在看门狗复位期间，nRSTOUT 引脚被声明。

当 XnWRESET 信号被声明为‘0’，下列依次发生：

1） SYSCON 请求 AHB 总线控制器，以完成当前 AHB 总线的处理。

2） 在当前总线处理完成之后，AHB 总线控制器发送确认信息到 SYSCON。

3） SYSCON 请求 DOMAIN-V，以完成当前 AXI 总线处理。

4） 在当前总线处理完成后，AXI 总线控制器发送确认信息到 SYSCON。

5） SYSCON 请求外部存储控制器进入到自刷新模式，当温复位被声明时，外部内存的内容必须被保存。

6） 当自刷新模式时，存储控制器发送确认信息。

7） SYSCON 声明内部复位信号和 XnRSTOUT.。

温复位模块，在具体 ARM11 处理器中，代码实现如下:

```
void Test_WarmReset(void)
{
    u32 uRstId;
    u32 uInform0, uInform1;
    uRstId = SYSC_RdRSTSTAT(1);

    // Check Alive Reg.
    // Alive Register
    uInform0 = 0x01234567;
    uInform1 = 0x6400ABCD;

    // For Test
    //WDT_operate(1,0,0,1,100,15625,15625);
    if( ( uRstId == 1 ) && !(g_OnTest) )
    {
        printf("Warm Reset- Memory data check \n");
        CheckData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    //Check Information Register Value
        if( (uInform0 !=Inp32Inform(0) )||(uInform1 != Inp32Inform(1)))
        {
            printf(" Information Register Value is wrong!!! \n");
        }
        else
```

```
    {
        printf(" Information Register Value is correct!!! \n");
    }
    printf("Warm Reset test is done\n");
    g_OnTest = 1;
    SYSC_BLKPwrONAll();
    Delay(10);
    SYSC_RdBLKPWR();
}
else
{
    printf("[WarmReset Test]\n");
    InitData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    // Alive Register Write
    Outp32Inform(0, uInform0);
    Outp32Inform(1, uInform1);

   //Added case : bus power down
      SYSC_BLKPwrOffAll();
    printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
    printf("Now, Push Warm Reset Botton. \n");
    while(1)
    {
    // test case
    DMAC_InitCh(DMA0, DMA_ALL, &oDmac_0);
    DMAC_InitCh(DMA1, DMA_ALL, &oDmac_1);
    INTC_SetVectAddr(NUM_DMA0,  Dma0Done_Test);
    INTC_SetVectAddr(NUM_DMA1,  Dma1Done_Test);
```

```
            INTC_Enable(NUM_DMA0);
            INTC_Enable(NUM_DMA1);

            g_DmaDone0=0;
            g_DmaDone1=0;
            printf("DMA Start \n");
            // 16MB
            DMACH_Setup(DMA_A, 0x0, 0x51f00000, 0, 0x51f01000, 0, WORD, 0x1000000, DEMAND, MEM,
                        MEM, BURST4, &oDmac_0);
            DMACH_Setup(DMA_A, 0x0, 0x52000000, 0, 0x52001000, 0, WORD, 0x1000000, DEMAND, MEM,
                        MEM, BURST4, &oDmac_1);

            // Enable DMA
            DMACH_Start(&oDmac_0);
            DMACH_Start(&oDmac_1);
            while((g_DmaDone0==0)||(g_DmaDone1==0))     // Int.
            {
                Copy(0x51000000, 0x51800000, 0x1000000);
            }
        }
    }
}
```

软件复位

当利用软件将 0x6410 写入到 SW_RST 时，软件复位被调用。行为与温复位的情况相同。

软件复位在 ARM11 处理器中，代码实现如下:

```
void Test_SoftReset(void)
{
    u32 uRstId;
```

```
u32 uInform0, uInform1;

printf("rINFORM0: 0x%x\n", Inp32Inform(0));
printf("rINFORM1: 0x%x\n", Inp32Inform(1));
uInform0 = 0xABCD6400;
uInform1 = 0x6400ABCD;
uRstId = SYSC_RdRSTSTAT(1);
SYSC_RdBLKPWR();

if( ( uRstId == 5 ) && !(g_OnTest) )
{
    printf("Software Reset- Memory data check \n");
    CheckData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

    //Check Information Register Value
    if( (uInform0 !=Inp32Inform(0) )||(uInform1 != Inp32Inform(1)))
    {
        printf(" Information Register Value is wrong!!! \n");
    }
    else
    {
        printf(" Information Register Value is correct!!! \n");
    }
    printf("software reset test is done\n");
    g_OnTest = 1;
    SYSC_BLKPwrONAll();
    Delay(10);
    SYSC_RdBLKPWR();
}
```

```
    else
    {
        printf("[SoftReset Test]\n");
        InitData_SDRAM(_DRAM_BaseAddress+0x1000000, 0x10000);

           //Added case : bus power down
           SYSC_BLKPwrOffAll();

           // Added case : Clock Off Case
           // Outp32SYSC(0x30, 0xFDDFFFFE); //IROM, MEM0, MFC
           // Outp32SYSC(0x30, 0xFFFFFFFE); // MFC, MFC Block OFF OK
           // Outp32SYSC(0x30, 0xFDFFFFFF); // IROM  OK
           // Outp32SYSC(0x30, 0xFFDFFFFF); // MEM0
        printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
        // Alive Register Write
        Outp32Inform(0, uInform0);
        Outp32Inform(1, uInform1);
        //Outp32(0x7F008880, 0x1000);

        UART_TxEmpty();
        printf("Now, Soft Reset causes reset on 6410 except SDRAM. \n");
        SYSC_SWRST();
        while(!UART_GetKey());
    }
}
```

看门狗复位

当软件挂起时，看门狗复位被调用。因此，在用于看门狗复位的 WDT 和 WDT 超时信号里，软件不能初始化寄存器。在看门狗复位期间，有以下动作发生：

- 除了 ALIVE 和 RTC 模块，所有模块进入预先定义好的复位状态。
- 所有引脚都进入复位状态。
- 在看门狗复位期间，nRSTOUT 引脚被声明。

在正常模式和闲置模式下，看门狗可被激活，并可产生超时信号。当看门狗定时器和复位使能时，其被调用。因此，下列依次发生：

1） WDT 产生超时信号。

2） SYSCON 调用复位信号，初始化内部 IP。

3） 包括 nRSTOUT 的复位被声明，直到复位计数器 RST_STABLE 被终止。

看门狗复位在具体 ARM11 处理器中，代码实现如下:

```
void Test_WDTReset(void)
{
    printf("[WatchDog Timer Reset Test]\n");
    INTC_Enable(NUM_WDT);

    // 1. Clock division_factor 128
    printf("\nClock Division Factor: 1(dec), Prescaler: 100(dec)\n");
    // WDT reset enable
     printf("\nI will restart after 2 sec.\n");
    WDT_operate(1,1,0,1,100,15625,15625);

    //Test Case - add SUB Block Off
    SYSC_BLKPwrOffAll();

    //Added case : Clock Off Case
       Outp32(0x7E00F030, 0xFDDFFFFE);
    // Outp32SYSC(0x30, 0xFFFFFFFE);    // MFC, MFC Block OFF OK
    // Outp32SYSC(0x30, 0xFDFFFFFF);    // IROM  OK
    // Outp32SYSC(0x30, 0xFFDFFFFF);    // MEMO
    printf("HCLKGATE: 0x%x\n", Inp32(0x7E00F030));
```

```
//while(!UART_GetKey());
// Test Case - add Bus operation
while(1)
{
    // test case
    DMAC_InitCh(DMA0, DMA_ALL, &oDmac_0);
    DMAC_InitCh(DMA1, DMA_ALL, &oDmac_1);
    INTC_SetVectAddr(NUM_DMA0,  Dma0Done_Test);
    INTC_SetVectAddr(NUM_DMA1,  Dma1Done_Test);
    INTC_Enable(NUM_DMA0);
    INTC_Enable(NUM_DMA1);

    g_DmaDone0=0;
    g_DmaDone1=0;

    printf("DMA Start \n");
    // 16MB
    DMACH_Setup(DMA_A, 0x0, 0x51f00000, 0, 0x51f01000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_0);
    DMACH_Setup(DMA_A, 0x0, 0x52000000, 0, 0x52001000, 0, WORD, 0x1000000, DEMAND, MEM,
                MEM, BURST4, &oDmac_1);

    // Enable DMA
    DMACH_Start(&oDmac_0);
    DMACH_Start(&oDmac_1);

    while((g_DmaDone0==0)||(g_DmaDone1==0))     // Int.
    {
```