		> 200ns)	

OSC_STABLE	位	描述	初始状态
RESERVED	[31:4]	保留。	0x0000_000
OSC_CNT_VALUE	[3:0]	振荡器pad稳定计数器的值。 值 周期 值 周期 0x0 2^4 0x1 2^9 0x2 2^{10} 0x3 2^{11} 0x4 2^{12} 0x5 2^{13} 0x6 2^{14} 0x7 2^{15} 0x8 2^{16} 其它 2^4	0x1

PWM_STABLE	位	描述	初始状态
RESERVED	[31:4]	保留。	0x0000_000
PWR_CNT_VALUE	[3:0]	电源稳定计数器的值。 值 周期 值 周期 0x0 2^4 0x1 2^{12} 0x2 2^{13} 0x3 2^{14} 0x4 2^{15} 0x5 2^{16} 0x6 2^{17} 0x7 2^{18} 0x8 2^{19} 其它 2^4	0x1

MTC_STABLE	位	描述	初始状态
RESERVED	[31:28]	保留。	0xF
DOMAIN_ETM	[27:24]	内存电源稳定计数器，用于domain ETM。	0xF
DOMAIN_S	[23:20]	内存电源稳定计数器，用于domain S。	0xF
DOMAIN_F	[19:16]	内存电源稳定计数器，用于domain F。	0xF
DOMAIN_P	[15:12]	内存电源稳定计数器，用于domain P。	0xF

DOMAIN_I	[11:8]	内存电源稳定计数器，用于domain I。	0xF
DOMAIN_V	[7:4]	内存电源稳定计数器，用于domain V。	0xF
DOMAIN_TOP	[3:0]	内存电源稳定计数器，用于domain TOP。	0xF

MTC_STABLE 代表若干外部振荡器（或时钟）周期。当子块从 MTC 模式返回到正常操作模式时，内部电源的稳定需要一段时间。这期间必须大于 200ns，并可以估计出使用 MTC_STABLE 和 OSC_FREQ 寄存器的值。

3.4.13 其他控制寄存器

寄存器	地址	读/写	描述	复位值
OTHERS	0x7E00_F900	读/写	其他控制寄存器。	0x0000_801E

OTHERS	位	描述	初始状态
RESERVED	[31:17]	保留。	0x0000
USB_SIG_MASK	[16]	屏蔽USB信号。 （在USB PHY被使用前，该位必须进行设置）	0
USB_STATE	[15]	USB PHY电源状态，只读位（1：正常，0：掉电）	1
RESERVED	[14]	不改变。	0
CLEAR_DBGACK	[13]	当该区域为1时，清除DBGACK信号。ARM1176声明DBGACK信号，以说明系统进入调试状态。如果DBGACK信号被声明，该状态被保留在SYSCON里，直到软件使用该区域而被清除。	0
CLEAR_BATF_INT	[12]	该位被设置时，清除中断所引起的电池故障。	0
RESERVED	[11:7]	不改变。	0x00
SYNCMUXSEL	[6]	SYNCMUX选择（0：MOUTMPLL，1：DOUTAPLL）。	0
RESEVED	[5:3]	不改变。	0x3
SPNIDEN	[2]	安全特权的非入侵式使能调试。在 ARM1176 的安全区中，该区域的使能和禁用非入侵式调试。如果该位为‘1’，非入侵式调试在所有非安全模式下被允许。否则，非入侵式调试在所有安	1

		全特权的模式下不被允许。非入侵式调试被允许，是在安全使用模式下，根据 ARM1176 的 SUNIDEN 位决定的。	
SPIDEN	[1]	安全特权的入侵式使能调试。在ARM1176的安全区中，该区域的使能和禁用入侵式调试。如果该位为‘1’，入侵式调试在所有安全模式下被允许。 否则，入侵式调试在任何安全特权的模式下不被允许。入侵式调试被允许，是在安全使用模式下，根据ARM1176的SUNIDEN位决定的。	1
CP15DISABLE	[0]	禁用写评到一些系统控制处理器ARM1176的寄存器。（0：使能，1：禁用）	0

3.4.14 状态寄存器

寄存器	地址	读/写	描述	复位值
RST_STAT	0x7E00_F904	读	复位状态寄存器。	0x0000_0001
WAKEUP_STAT	0x7E00_F908	读/写	唤醒状态寄存器。	0x0000_0000
BLK_PWR_STAT	0x7E00_F90C	读	块电源状态寄存器。	0x0000_007F

RST_STAT	位	描述	初始状态
RESERVED	[31:6]	保留。	0x0000_000
SW_RESET	[5]	通过SWRESET进行软件复位。	0
RESERVED	[4]	保留。	0
SLEEP_WAKEUP	[3]	通过睡眠模式复位唤醒。	0
WDT_RESET	[2]	通过WDTRST进行看门狗定时器复位。	0
WARM_RESET	[1]	通过XnWRESET进行温复位。当睡眠模式的唤醒源是XnWRESET时，该区域不被设置。	0
HW_RESET	[0]	通过XnRESET进行外部复位。	1

WAKEUP_STAT	位	描述	初始状态
RESERVED	[31:12]	保留。	0x0000_0
MMC2_WAKEUP	[11]	通过MMC2唤醒，通过写1被清除。	0
MMC1_WAKEUP	[10]	通过MMC1唤醒，通过写1被清除。	0
MMC0_WAKEUP	[9]	通过MMC0唤醒，通过写1被清除。	0
HSI_WAKEUP	[8]	通过HSI唤醒，通过写1被清除。	0
WRESET_WAKEUP	[7]	通过温复位唤醒，通过写1被清除。	0
BATFLT_WAKEUP	[6]	通过电池故障唤醒，通过写1被清除	0
MSM_WAKEUP	[5]	通过MSM 调制解调器唤醒，通过写1被清除。	0
KEY_WAKEUP	[4]	通过键盘唤醒，通过写1被清除。	0
TS_WAKEUP	[3]	通过触摸屏唤醒，通过写1被清除。	0
RTC_TICK_WAKEUP	[2]	通过嘀嗒信号中断唤醒，通过写1被清除。	0
RTC_ALARM_WAKEUP	[1]	通过RTC警告唤醒，通过写1被清除。	0
EINT_WAKEUP	[0]	通过外部中断唤醒，通过写1被清除。	0

BLK_PWR_STAT	位	描述	初始状态
RESERVED	[31:7]	保留。	0x0000_000
BLK_ETM	[6]	块ETM电源准备就绪。	1
BLK_S	[5]	块 S 电源准备就绪。	1
BLK_F	[4]	块 F 电源准备就绪。	1
BLK_P	[3]	块 P 电源准备就绪。	1
BLK_I	[2]	块 I 电源准备就绪。	1
BLK_V	[1]	块 V 电源准备就绪。	1
BLK_TOP	[0]	块 TOP 电源准备就绪	1

3.4.15 消息寄存器

寄存器	地址	读/写	描述	复位值
INFORM0	0x7E00_FA00	读/写	消息寄存器 0。	0x0000_0000
INFORM1	0x7E00_FA04	读/写	消息寄存器 1。	0x0000_0000
INFORM2	0x7E00_FA08	读/写	消息寄存器 2。	0x0000_0000
INFORM3	0x7E00_FA0C	读/写	消息寄存器 3。	0x0000_0000

INFORMn	位	描述	初始状态
INFORM	[31:0]	用户定义消息。通过声明 XnRESET 引脚，INFORM0～3 寄存器被清除。	0x0000_0000

3.5 部分应用程序

下面是系统控制器应用的一部分程序。有分别是系统时钟控制的程序，也有系统电源管理控制的部分程序。对这些程序进行分析将有助于更进一步理解系统控制器。

```
//用于获得指令时钟信息
//输入 : NONE
//输出 : NONE
void SYSC_GetClkInform( void)
{
    u8 muxApll, muxMpll, muxSync;
    u8 divApll, divHclkx2, divHclk, divPclk;
    u16 pllM, pllP, pllS;
    u32 temp;

    ////
    // 时钟分频频率
    temp = Inp32(rCLK_DIV0);
```

```
divApll = temp & 0xf;
divHclkx2 = (temp>>9) & 0x7;
divHclk = (temp>>8) & 0x1;
divPclk = (temp>>12) & 0xf;

////
// 操作模式
temp = Inp32(rOTHERS);
temp = (temp>>8)&0xf;
if(temp)
{
    g_SYNCACK = 1;
}
else
{
    g_SYNCACK = 0;
}

////
// ARMCLK
muxApll = Inp32(rCLK_SRC) & 0x1;
if(muxApll) //FOUT
{
    temp = Inp32(rAPLL_CON);
    pllM = (temp>>16)&0x3ff;
    pllP = (temp>>8)&0x3f;
    pllS = (temp&0x7);

    g_APLL = ((FIN>>pllS)/pllP)*pllM;
```

```
}
else    //FIN
{
    g_APLL = FIN;
}

g_ARMCLK = g_APLL/(divApll+1);

////
// HCLK
muxSync = (Inp32(rOTHERS)>>7) & 0x1;
if(muxSync) //synchronous mode
{
    g_HCLKx2 = g_APLL/(divHclkx2+1);

    temp = Inp32(rMPLL_CON);
    pllM = (temp>>16)&0x3ff;
    pllP = (temp>>8)&0x3f;
    pllS = (temp&0x7);

    g_MPLL = ((FIN>>pllS)/pllP)*pllM;
}
else
{
    muxMpll = (Inp32(rCLK_SRC)>>1) & 0x1;
    if(muxMpll) //FOUT
    {
        temp = Inp32(rMPLL_CON);
        pllM = (temp>>16)&0x3ff;
```

```
            pllP = (temp>>8)&0x3f;
            pllS = (temp&0x7);

            g_MPLL = ((FIN>>pllS)/pllP)*pllM;
        }
        else    //FIN
        {
            g_MPLL = FIN;
        }
        g_HCLKx2 = g_MPLL/(divHclkx2+1);
    }

    g_HCLK = g_HCLKx2/(divHclk+1);

    ////
    // PCLK
    g_PCLK = g_HCLKx2/(divPclk+1);

    return;
}
```

/*控制 PLL 输出频率（APLL, MPLL, EPLL）：Fout = (mdiv * Fin) / (pdiv x 2^sdiv), Fout = ((mdiv+k/2^16) * Fin) / (pdiv x 2^sdiv)*/

//输入： ePLL : APLL, MPLL, EPLL
// uMdiv : Mdiv 值 (56 ~ 1023), (13 ~ 255)
// uPdiv : Pdiv 值 (1~63)
// uSdiv : Sdiv 值 (0~5)
// uKdiv : PLL9025X (Not Used, 0), PLL9024X(0~65535)
//输出： NONE

```
void SYSC_SetPLL(PLL_eTYPE    ePLL, u32 uMdiv, u32 uPdiv, u32 uSdiv, u32 uKdiv)
{
    u32 temp, uRegValue;
    switch(ePLL)
    {
        case eAPLL:
            // 检测分频器的值
            if( uMdiv < 56 || uMdiv > 1023)
            {
                printf(" Wrong    Mdiv Value, Correct Value Range = (56 ~ 1023) (%d)\n", uMdiv);
            }
            if( uPdiv < 1 || uPdiv > 63)
            {
                printf(" Wrong    Pdiv Value, Correct Value Range = (1 ~ 63) (%d)\n", uPdiv);
            }
            if( uSdiv > 5)
            {
                printf(" Wrong    Sdiv Value, Correct Value Range = (0 ~ 5) (%d)\n", uSdiv);
            }
                //检测 Fvco 范围

                temp = ((FIN/uPdiv)*uMdiv)/1000000;
            if( temp <1000 || temp > 2000)
            {
#if 0 //EVT1
                printf(" Please select the proper M,P,S divider value\n");
                printf(" Fvco Range = (1000MHz ~ 2000MHz),    Current Value is (%d)MHz\n", temp);
#endif
            }
```

```
            uRegValue = (u32)(((u32)(0x1<<31))|(uMdiv<<16)|(uPdiv<<8)|(uSdiv<<0));
            Outp32(rAPLL_CON, uRegValue);
            break;

        case eMPLL:
            // 检测分频器的值
            if( uMdiv < 56 || uMdiv > 1023)
            {
                printf(" Wrong    Mdiv Value, Correct Value Range = (56 ~ 1023) (%d)\n", uMdiv);
            }
            if( uPdiv < 1 || uPdiv > 63)
            {
                printf(" Wrong    Pdiv Value, Correct Value Range = (1 ~ 63) (%d)\n", uPdiv);
            }
            if( uSdiv > 5)
            {
                printf(" Wrong    Sdiv Value, Correct Value Range = (0 ~ 5) (%d)\n", uSdiv);
            }
                // 检测 Fvco 范围

                temp = ((FIN/uPdiv)*uMdiv)/1000000;
            if( temp <1000 || temp > 2000)
            {
#if 0 //EVT1
                printf(" Please select the proper M,P,S divider value\n");
                printf(" Fvco Range = (1000MHz ~ 2000MHz),    Current Value is (%d)MHz\n", temp);
#endif
            }
```