```
//输入: Controller  -  OneNand 控制器端口数
//      uStartBlkAddr  -  擦除块开始地址
//      uEndBlkAddr  -  擦除块结束地址
#if 0
OneNAND_eINTERROR ONENAND_EraseBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    u32 i;
    u32 uStatus;
    if(uEndBlkAddr < uStartBlkAddr)
        return eOND_NOERROR;
    OneNandT_oIntFlag.IntActInt = 0;
    OneNandT_oIntFlag.ErsFailInt = 0;
    OneNandT_oIntFlag.LockedBlkInt = 0;
    OneNandT_oIntFlag.ErsCmpInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
    }
    else
    {
#if 1
        for(i=uStartBlkAddr ; i<uEndBlkAddr ; i++)
        {
            ONENAND_WriteCmd(Controller, i, 0, 0x01);
        }
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
#else
```

```
        for(i=uStartBlkAddr ; i<=uEndBlkAddr ; i++)
        {
            ONENAND_WriteCmd(Controller, i, 0, 0x03);
            while(!OneNandT_oIntFlag.IntActInt);
        }
#endif
        // 校验擦除块
        for(i=uStartBlkAddr ; i<=uEndBlkAddr ; i++)
        {
            OneNandT_oIntFlag.IntActInt = 0;
            ONENAND_WriteCmd(Controller, i, 0, 0x15);
            while(!OneNandT_oIntFlag.IntActInt);

            if(OneNandT_oIntFlag.ErsFailInt == 1)
            {
                OneNandT_oIntFlag.ErsFailInt = 0;
                if(OneNandT_oIntFlag.LockedBlkInt == 1)
                return (OneNAND_eINTERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
                return eOND_ERSFAIL;
            }
        }
        if(OneNandT_oIntFlag.ErsCmpInt == 1)
        {
            return eOND_NOERROR;
        }
        else
            return eOND_ERSFAIL;
    }
#if 0
```

```
    uError = Inp32(&ONENAND(Controller)->rIntErrStat);
    if (uError & (1<<3))
    {
        //擦除操作失败
        Outp32(&ONENAND(Controller)->rIntErrAck, (1<<3));
        if(uError & (1<<8))
        {
            Outp32(&ONENAND(Controller)->rIntErrAck, (1<<8));
            return (eOND_ERSFAIL | eOND_LOCKEDBLK);
        }
        return eOND_ERSFAIL;
    }
#elif 0
    if(OneNandT_oIntFlag.ErsFailInt == 1)
    {
        if(OneNandT_oIntFlag.LockedBlkInt == 1)
        return (OneNAND_eINTERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
        return eOND_ERSFAIL;
    }
    else if(OneNandT_oIntFlag.BlkRwCmpInt == 1)
    {
        return eOND_NOERROR;
    }
#endif
#if 0
    ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x00);
    ONENAND_ReadCmd(Controller, uEndBlkAddr, 0, &uData);

    // 等待擦除完成
```

```
    while(uData)
    {
        ONENAND_ReadCmd(Controller, uEndBlkAddr, &uData);
    }
    return eOND_ERSCMP;
#endif
    if(OneNandT_oIntFlag.ErsCmpInt == 1)
        return eOND_NOERROR;
    else
        return eOND_ERSFAIL;
}
#else
OneNAND_eINTERROR ONENAND_EraseBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
{
    u32 i, j, uBlockSize, uQuotient, uRemainder;
    u32 uBlock;
    if(uEndBlkAddr < uStartBlkAddr)
        return eOND_NOERROR;
    OneNandT_oIntFlag.IntActInt = 0;
    OneNandT_oIntFlag.ErsFailInt = 0;
    OneNandT_oIntFlag.LockedBlkInt = 0;
    OneNandT_oIntFlag.ErsCmpInt = 0;
    if(uStartBlkAddr == uEndBlkAddr)
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);
        OneNandT_oIntFlag.IntActInt = 0;
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x15);
        while(!OneNandT_oIntFlag.IntActInt);
```

```
    if(OneNandT_oIntFlag.ErsFailInt == 1)
    {
        OneNandT_oIntFlag.ErsFailInt = 0;
        if(OneNandT_oIntFlag.LockedBlkInt == 1)
        return (OneNAND_eINTERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
        return eOND_ERSFAIL;
    }
}
else
{
    uBlockSize = uEndBlkAddr - uStartBlkAddr + 1;
    uQuotient = uBlockSize/OND_MULTIERASE_SIZE;
    uRemainder = uBlockSize%OND_MULTIERASE_SIZE;
    uBlock = uStartBlkAddr;

    for(i=0 ; i<uQuotient ; i++)
    {
        for(j=uBlock ; j<(uBlock+OND_MULTIERASE_SIZE-1) ; j++)
        {
            OneNandT_oIntFlag.IntActInt = 0;
            ONENAND_WriteCmd(Controller, j, 0, 0x01);
            while(!OneNandT_oIntFlag.IntActInt);
        }
        ONENAND_WriteCmd(Controller, uBlock+OND_MULTIERASE_SIZE-1, 0, 0x03);
        while(!OneNandT_oIntFlag.IntActInt);

        //校验擦除块
        for(j=uBlock ; j<=(uBlock+OND_MULTIERASE_SIZE-1) ; j++)
```

```
    {
        OneNandT_oIntFlag.IntActInt = 0;
        ONENAND_WriteCmd(Controller, j, 0, 0x15);
        while(!OneNandT_oIntFlag.IntActInt);
        if(OneNandT_oIntFlag.ErsFailInt == 1)
        {
            OneNandT_oIntFlag.ErsFailInt = 0;
            if(OneNandT_oIntFlag.LockedBlkInt == 1)
            return (OneNAND_eINTERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
            return eOND_ERSFAIL;
        }
    }
    uBlock +=  OND_MULTIERASE_SIZE;
}
if(uRemainder > 0)
{
    for(i=uBlock ; i<(uBlock+uRemainder-1) ; i++)
    {
        OneNandT_oIntFlag.IntActInt = 0;
        ONENAND_WriteCmd(Controller, i, 0, 0x01);
        while(!OneNandT_oIntFlag.IntActInt);
    }
    ONENAND_WriteCmd(Controller, uBlock+uRemainder-1, 0, 0x03);
    while(!OneNandT_oIntFlag.IntActInt);

    //校验擦除块
    for(i=uBlock ; i<=(uBlock+uRemainder-1) ; i++)
    {
        OneNandT_oIntFlag.IntActInt = 0;
```

```
                ONENAND_WriteCmd(Controller, i, 0, 0x15);
                while(!OneNandT_oIntFlag.IntActInt);

                if(OneNandT_oIntFlag.ErsFailInt == 1)
                {
                    OneNandT_oIntFlag.ErsFailInt = 0;
                    if(OneNandT_oIntFlag.LockedBlkInt == 1)
                    return (OneNAND_eINTERROR)(eOND_ERSFAIL | eOND_LOCKEDBLK);
                    return eOND_ERSFAIL;
                }
            }
        }
    }
    if(OneNandT_oIntFlag.ErsCmpInt == 1)
        return eOND_NOERROR;
    else
        return eOND_ERSFAIL;
}
#endif
```

（2）封锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作

OneNAND Flash 控制器支持所有的闪存加锁操作。然而，内存设备对这个功能的支持可能有所限制。如果不支持锁功能，所有有关命令都会忽略。如果不支持“unlock all”功能，将会触发中断。

一块内存区域能被加锁也能被“locked-tight”。一旦一个区域被“locked-tight”，解锁则需要复位。

如果存储设备支持区段锁定，那么通过使用多条指令可以加锁或者解锁多个范围。这种特性通过 lock_bit_per_block 寄存器来控制。当使用时，如果开始和最终地址能指定范围，那么任何“unlock”命令都被转化为“unlock all”命令。实际上，使用 lock/unlock/lock-tight 命令是由 AHB 总线处理（读或写）和 datain 总线的低字节决定的。如表 7-7 所示。

表 7-7　锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作

| 地址 | 命令类型 | 数据输入 | 功能 |
| --- | --- | --- | --- |
| [23:22] =10<br>DFS_DBS和 FBA被用<br>FPA 和FSA 不用于<br>锁操作，必须清除。 | 写 | 0x08 | 为解锁保存开始地址 |
| | 写 | 0x09 | 为解锁和初始化解锁保存终止地址 |
| | 写 | 0x0A | 为加锁保存起始地址 |
| | 写 | 0x0B | 为加锁和初始化加锁保存终止地址 |
| | 写 | 0x0C | 为 lock-tight 保存起始地址<br>注：存储控制器发送“lock-tight”指令之前不会校验指定区段是否加锁，但是“lock-tight”只能在加锁区段成功执行 |
| | 写 | 0x0D | 为 lock-tight 和初始化 lock-tight 保存终止地址。<br>注：存储控制器发送“lock-tight”指令之前不会校验指定区段是否加锁，但是“lock-tight”只能在加锁区段成功执行 |
| | 写 | 0x0E | 解锁整个存储器列阵。如果不支持这个功能，将产生一个指令错误中断 |

下面是 ARM11 处理器中，OneNand Flash 控制器锁（Lock），解锁（Unlock）和锁住（Lock-Tight）操作的部分代码，参考如下：

1.锁 Lock 操作的代码实现：

```
// ONENAND_LockBlock 函数：主要功能实现锁存储器块
// 输入: Controller  -  OneNand 控制器端口数
//       uStartBlkAddr  -  锁开始块
//       uEndBlkAddr  -  锁结束块
bool ONENAND_LockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
    {
        //u32 uData;
        OneNandT_oIntFlag.IntActInt = 0;
        if(uStartBlkAddr == uEndBlkAddr)
```

```
    {
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }
    else
    {
        ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x0A);
        ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x0B);
    }


    //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
    //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
    while(!OneNandT_oIntFlag.IntActInt);
#if 0
    ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);
    if ((uData&0xFFFF) != 0x2)
    {
    //UART_Printf(" Failed lock block, locked status (0x%x)\n", uData);
        return TRUE;
    }
#endif
    return FALSE;
}
```

2.解锁 Unlock 操作的代码实现：

```
// ONENAND_UnlockBlock 函数：主要功能实现解锁存储器块
// 输入: Controller - OneNand 控制器端口数
//      uStartBlkAddr  -  解锁开始块
//      uEndBlkAddr  -  解锁结束块
```

```
bool ONENAND_UnlockBlock(u32 Controller, u32 uStartBlkAddr, u32 uEndBlkAddr)
    {
        //u32 uData;
        OneNandT_oIntFlag.IntActInt = 0;
        if(uStartBlkAddr == uEndBlkAddr)
        {
            ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
        }
        else
        {
            ONENAND_WriteCmd(Controller, uStartBlkAddr, 0, 0x08);
            ONENAND_WriteCmd(Controller, uEndBlkAddr, 0, 0x09);
        }

        //while(!(ONENAND(Controller)->rIntErrStat & (1<<10)));
        //Outp32(&ONENAND(Controller)->rIntErrAck, (1<<10));
        while(!OneNandT_oIntFlag.IntActInt);
    #if 0
        ONENAND_DirectRead(Controller, OND_WPROT_STATUS, &uData);
        if ((uData&0xFFFF) != 0x4)
        {
            //UART_Printf(" Failed unlock block, locked status (0x%x)\n", uData);
            return TRUE;
        }
    #endif
        return FALSE;
    }
```

3. 锁住 Lock-Tight 操作的代码实现：