专题13-NandFlash操作

一、NandFlash原理

1.1、NOR与NAND的区别

性能比较

flash闪存是非易失存储器，可以对称为块的存储器单元块进行擦写和再编程。任何flash器件的写入操作只能在空或已擦除的单元内进行，所以大多数情况下，在进行写入操作之前必须先执行擦除。NAND器件执行擦除 操作是十分简单的，而NOR则要求在进行擦除前先要将目标块内所有的位都写为0。

由于擦除NOR器件时是以64～128KB的块进行的，执行一个写入/擦除操作的时间为5s ，与此相反，擦除NAND器件是以8～32KB的块进行的，执行相同的操作最多只需要4ms。

执行擦除时块尺寸的不同进一步拉大了NOR和NAND之间的性能差距，统计表明，对于给定的一套写入操作（尤其是更新小文件时），更多的擦除操作必须在基于NOR的单元中进行。这样，当选择存储解决方案时，设计师必 须权衡以下的各项因素。

- NOR的读速度比NAND稍快一些。
- NAND的写入速度比NOR快很多。
- NAND的擦除速度远比NOR快。
- NAND的擦除单元更小，相应的擦除电路更加简单。
- NAND的实际应用方式要比NOR复杂的多。
- NOR可以直接使用，并在上面直接运行代码，而NAND需要I/O接口，因此使用时需要驱动。

接口差别

NORflash带有SRAM接口，有足够的地址引脚来寻址，可以很容易地存取其内部的每 一个字节。

NAND器件使用复杂的I/O口来串行地存取数据，各个产品或厂商的方法可能各不相同 。8个引脚用来传送控制、地址和数据信息。

NAND读和写操作采用512字节的块，这一点有点像硬盘管理此类操作，很自然地，基于NAND的存储器就可以取代硬盘或其他块设备。

NOR的特点是芯片内执行(XIP, eXecute In Place),这样应用程序可以直接在flash闪存内运行,不必再把代码读到系统RAM中。

NOR的传输效率很高,在1～4MB的小容量时具有很高的成本效益,但是很低的写入和擦除速度大大影响了它的性能。

NAND结构能提供极高的单元密度,可以达到高存储密度,并且写入和擦除的速度也很快。应用NAND的困难在于flash的管理需要特殊的系统接口。

1.2、NandFlash特点

物理构成

NAND Flash 的数据是以bit的方式保存在memory cell，一般来说，一个cell 中只能存储一个bit。这些cell 以8个或者16个为单位，连成bit line，形成所谓的byte(x8)/word(x16)，这就是NAND Device的位宽。这些Line会再组成Page，(NAND Flash 有多种结构，我使用的NAND Flash 是K9F1208,下面内容针对三星的K9F1208U0M)，每页528Bytes(512byte(Main Area)+16byte(Spare Area))，每32个page形成一个Block(32*528B)。具体一片flash上有多少个Block视需要所定。我所使用的三星k9f1208U0M具有4096个block，故总容量为4096*（32*528B）=66MB，但是其中的2MB是用来保存ECC校验码等额外数据的，故实际中可使用的为64MB。

NAND flash以页为单位读写数据，而以块为单位擦除数据。按照这样的组织方式可以形成所谓的三类地址：

Column Address：Starting Address of the Register. 翻成中文为列地址，地址的低8位

Page Address ：页地址

Block Address ：块地址

对于NAND Flash来讲，地址和命令只能在I/O[7:0]上传递，数据宽度是8位。

位交换

所有flash器件都受位交换现象的困扰。在某些情况下（很少见，NAND发生的次数要比NOR多），一个比特位会发生反转或被报告反转了。

一位的变化可能不很明显，但是如果发生在一个关键文件上，这个小小的故障可能 导致系统停机。如果只是报告有问题，多读几次就可能解决了。

当然，如果这个位真的改变了，就必须采用错误探测/错误更正(EDC/ECC)算法。位 反转的问题更多见于NAND闪存，NAND的供应商建议使用NAND闪存的时候，同时使用EDC/ECC算法。

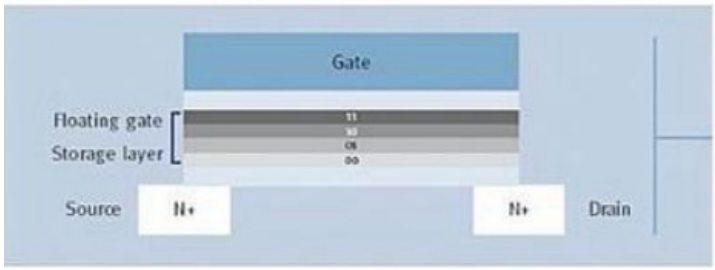
这个问题对于用NAND存储多媒体信息时倒不是致命的。当然，如果用本地存储设备来存储操作系统、配置文件或其他敏感信息时， 必须使用EDC/ECC系统以确保可靠性。

1.3、根据物理结构上的区别,NandFlash主要分为如下两类:

- SLC (Single Level Cell): 单层式存储
- MLC (Multi Level Cell): 多层式存储

SLC在存储格上只存一位数据，而MLC则存放两位数据。

(a)SLC



(b)MLC

1.3.1、2.2 MLC对比SLC

1. 价格：

由于MLC采用了更高密度的存储方式，因此同容量的MLC价格上远低于SLC.

2. 访问速度：

SLC的访问速度一般要比MLC快3倍以上.

3. 使用寿命：

SLC能进行10万次的擦写，MLC能进行1万次

4. 功耗：

MLC功耗比SLC高15%左右

1.4、访问方式

1.4.1、编址方式

地址+命令+数据

Table 1: Array Addressing for Logical Unit (LUN)

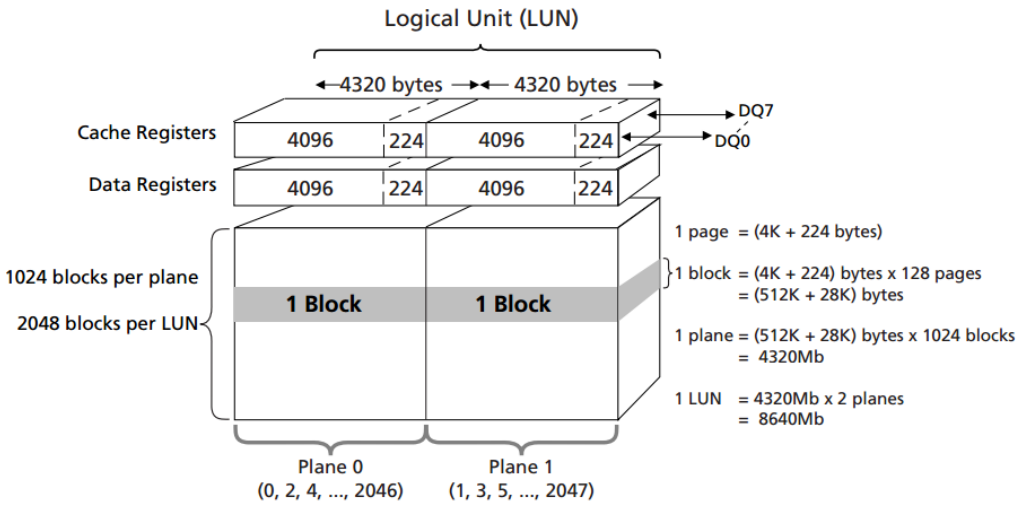
Cycle	DQ7	DQ6	DQ5	DQ4	DQ3	DQ2	DQ1	DQ0
First	CA7	CA6	CA5	CA4	CA3	CA2	CA1	CA0[2]
Second	LOW	LOW	LOW	CA12[3]	CA11	CA10	CA9	CA8
Third	BA7[4]	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Fourth	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8
Fifth	LOW	LOW	LOW	LOW	LOW	LA0[5]	BA17	BA16

Notes:
1. CAx = column address, PAx = page address, BAx = block address, LAx = LUN address; the page address, block address, and LUN address are collectively called the row address.
2. When using the synchronous interface, CA0 is forced to 0 internally; one data cycle always returns one even byte and one odd byte.
3. Column addresses 4320 (10E0h) through 8191 (1FFFh) are invalid, out of bounds, do not exist in the device, and cannot be addressed.
4. BA[7] is the plane-select bit:
   Plane 0: BA[7] = 0
   Plane 1: BA[7] = 1
5. LA0 is the LUN-select bit. It is present only when two LUNs are shared on the target; otherwise, it should be held LOW.
   LUN 0: LA0 = 0
   LUN 1: LA0 = 1

1.4.2、地址结构

行地址+列地址：

Figure 8: Array Organization per Logical Unit (LUN)



1.4.3、信号引脚

1. CLE(Command Latch Enable): 命令锁存允许
2. ALE(Address Lactch Enable): 地址锁存允许
3. CE:芯片选择
4. RE:读允许
5. WE:写允许
6. WP:在写或擦除期间，提供写保护
7. R/B:读/忙

二、NandFlash读操作

2.1、NandFlash读操作的类型

2.1.1、页读

读取一页所有的数据，只需要提供行地址。

2.1.2、随机读

指定存储单元的数据，需要提供行地址+页地址。

2.2、实现NandFlash读操作

8.11.1 NAND FLASH CONTROLLER REGISTER MAP

Address	R/W	Reset value	Name	Description
Base + 0x00	R/W	0xX000_100X	NFCONF	Configuration register
Base + 0x04	R/W	0x0001_00C6	NFCONT	Control register
Base + 0x08	R/W	0x0000_0000	NFCMMD	Command register
Base + 0x0c	R/W	0x0000_0000	NFADDR	Address register
Base + 0x10	R/W	0xXXXX_XXXX	NFDATA	Data register
Base + 0x14	R/W	0x0000_0000	NFMECCD0	1st and 2nd main ECC data register
Base + 0x18	R/W	0x0000_0000	NFMECCD1	3rd and 4th main ECC data register
Base + 0x1c	R/W	0x0000_0000	NFSECCD	Spare ECC read register
Base + 0x20	R/W	0x0000_0000	NFSBLK	Programmable start block address register
Base + 0x24	R/W	0x0000_0000	NFEBLK	Programmable end block address register
Base + 0x28	R/W	0x0080_001D	NFSTAT	NAND status registet
Base + 0x2C	R	0xXXXX_XXXX	NFECCERR0	ECC error status0 register
Base + 0x30	R	0x0000_0000	NFECCERR1	ECC error status1 register
Base + 0x34	R	0xXXXX_XXXX	NFMECC0	Generated ECC status0 register
Base + 0x38	R	0xXXXX_XXXX	NFMECC1	Generated ECC status1 register
Base + 0x3C	R	0xXXXX_XXXX	NFSECC	Generated Spare area ECC status register
Base + 0x40	R	0x0000_0000	NFMLCBITPT	4-bit ECC error bit pattern register
Base + 0x44	R	0x4000_0000	NF8ECCERR0	8bit ECC error status0 register
Base + 0x48	R	0x0000_0000	NF8ECCERR1	8bit ECC error status1 register
Base + 0x4C	R	0x0000_0000	NF8ECCERR2	8bit ECC error status2 register
Base + 0x50	R	0xXXXX_XXXX	NFM8ECC0	Generated 8-bit ECC status0 register
Base + 0x54	R	0xXXXX_XXXX	NFM8ECC1	Generated 8-bit ECC status1 register
Base + 0x58	R	0xXXXX_XXXX	NFM8ECC2	Generated 8-bit ECC status2 register
Base + 0x5C	R	0xXXXX_XXXX	NFM8ECC3	Generated 8-bit ECC status3 register
Base + 0x60	R	0x0000_0000	NFMLC8BITPT 0	8-bit ECC error bit pattern 0 register
Base + 0x64	R	0x0000_0000	NFMLC8BITPT 1	8-bit ECC error bit pattern 1 register

Base = 0x7020_0000
Stepping STON : 0x0C00_0000 ~ 0x0C00_1FFF (8K)

0x0000_0000 ~ 0x0000_1FFF (8K)*

*In 6410 memory map, stepping stone memory is in the area between 0x0C00_0000 and 0x0C00_1FFF.

2.2.1、NandFlash初始化

2.2.1.1、NandFlash 芯片时序图

Figure 9: Asynchronous Command Latch Cycle

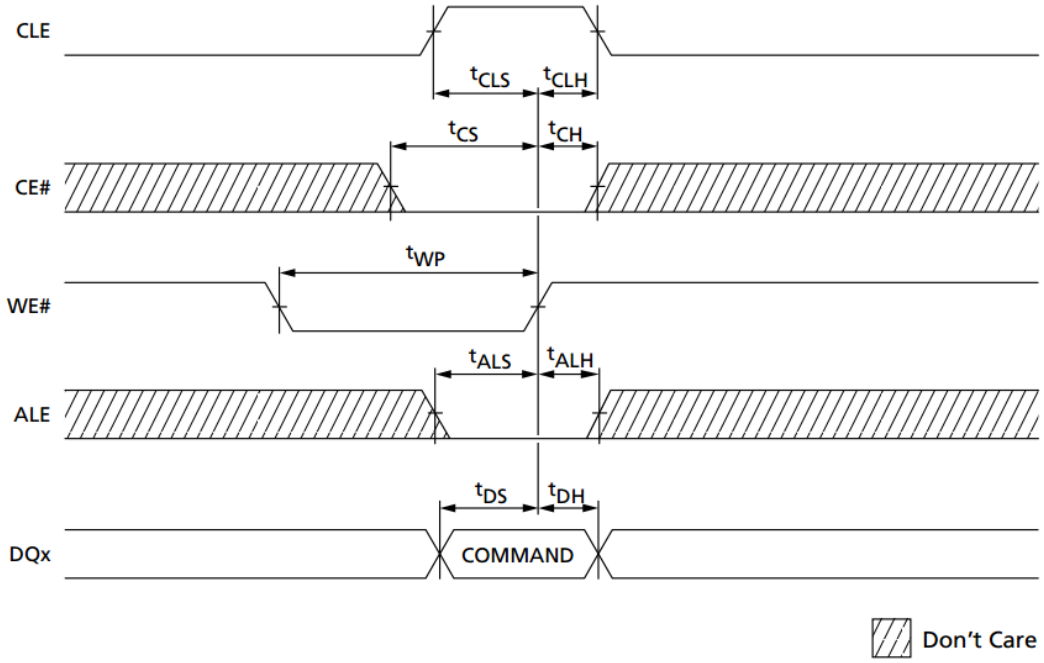


Table 37: AC Characteristics: Asynchronous Command, Address, and Data (Continued)

Parameter	Symbol	Mode 0		Mode 1		Mode 2		Mode 3		Mode 4		Mode 5		Unit	Notes
		Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max		
WE# pulse width	tWP	50	–	25	–	17	–	15	–	12	–	10	–	ns	
WP# transition to WE# LOW	tWW	100	–	100	–	100	–	100	–	100	–	100	–	ns	

Notes:
1. Timing for tADL begins in the address cycle, on the final rising edge of WE# and ends with the first rising edge of WE# for data input.
2. Data transition is measured ±200mV from steady-steady voltage with load. This parameter is sampled and not 100 percent tested.
3. AC characteristics may need to be relaxed if output drive strength is not set to at least nominal.
4. Do not issue a new command during tWB, even if R/B# or RDY is ready.

2.2.1.2、NandFlash 控制器时序图

8.4 NAND FLASH MEMORY TIMING

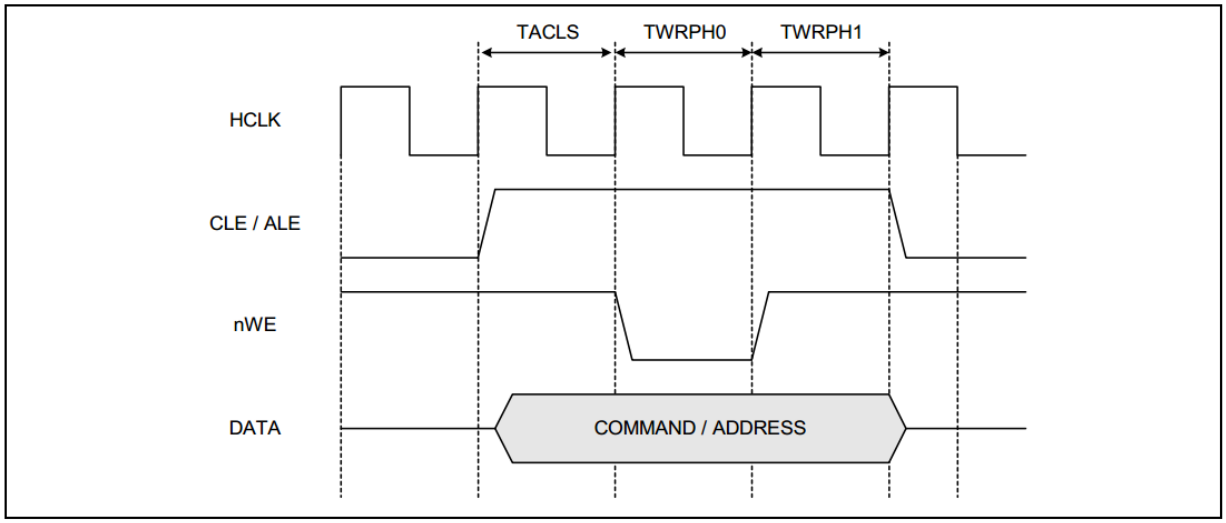


Figure 8-2. CLE & ALE Timing (TACLS=1, TWRPH0=0, TWRPH1=0) Block Diagram

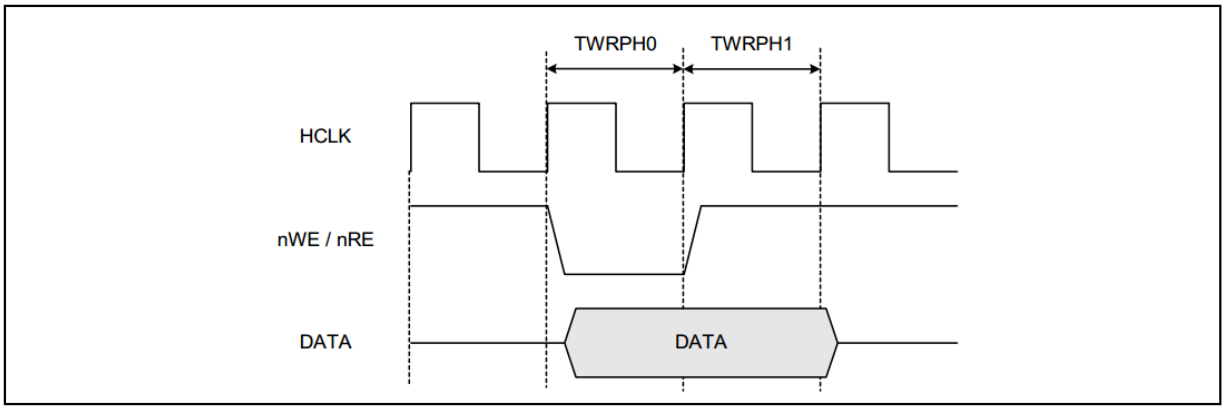


Figure 8-3. nWE & nRE Timing (TWRPH0=0, TWRPH1=0) Block Diagram

2.2.1.3、NandFlash 时序配置

8.11.2 NAND FLASH CONFIGURATION REGISTER

Register	Address	R/W	Description	Reset Value
NFCONF	0x70200000	R/W	NAND Flash Configuration register	0xX000100X

NFCONF	Bit	Description	Initial State
Reserved			
MLCClkCtrl	[30]	Clock control for 4-bit ECC & 8-bit ECC engine.(Hidden Spec.) 0: Recommended when system clock is more than 66MHz. 1: Recommended when system clock is less than 66MHz	0
Reserved	[29:26]	Reserved	0000
MsgLength	[25]	Message (Data) length for 4/8 bit ECC 0: 512-byte 1: 24-byte	0
ECCType	[24:23]	This bit indicates what kind of ECC should be used. 00: 1-bit ECC 10: 4-bit ECC 01 : 8-bit ECC Note. Don't confuse the value of 4-bit ECC and 8-bit ECC.	H/W Set (CfgBootEcc)
Reserved	[22:15]	Reserved	000000000
TACLS	[14:12]	CLE & ALE duration setting value (0~7) Duration = HCLK x TACLS	001
Reserved	[11]	Reserved	0
TWRPH0	[10:8]	TWRPH0 duration setting value (0~7) Duration = HCLK x (TWRPH0 + 1)	000
Reserved	[7]	Reserved	0
TWRPH1	[6:4]	TWRPH1 duration setting value (0~7) Duration = HCLK x (TWRPH1 + 1)	000
Reserved			
Reserved	[2]	Reserved. Must be written 1	1
Reserved			
Reserved	[0]	Reserved. Must be written 0.	0

```
52 #define TACLS   1
53 #define TWRPH0  2
54 #define TWRPH1  1
55
56 void nand_init(void)
57 {
58      //init NFCONF
59      NFCONF &= ~((7 << 12) | (7 << 8) | (7 << 4));
60      NFCONF |= ((TACLS << 12) | (TWRPH0 << 8) | (TWRPH1 << 4));
61
```

```
62      //init_NFCONT
63      NFCONT = (1 | (1 << 1));
64
65      //reset
66      nand_reset();
67 }
```

2.2.2、NAND FLASH芯片片选

8.11.3 CONTROL REGISTER

Register	Address	R/W	Description	Reset Value
NFCONT	0x70200004	R/W	NAND Flash control register	0x000100C6

NFCONT	Bit	Description	Initial State
Reserved	[31:19]	Reserved	0
ECC Direction	[18]	4-bit, 8-bitECC encoding / decoding control 0: Decoding 4-bit, 8bit ECC, It is used for page read 1: Encoding 4-bit, 8-bit ECC, It is be used for page program	0
Lock-tight	[17]	Lock-tight configuration 0: Disable lock-tight 1: Enable lock-tight, Once this bit is set to 1, you cannot clear. Only reset or wake up from sleep mode can make this bit disable (cannot cleared by software). When it is set to 1, the area setting in NFSBLK (0x70200020) to NFEBLK (0x70200024) is unlocked, and except this area, write or erase command will be invalid and only read command is valid. When you try to write or erase locked area, the illegal access will be occurred (NFSTAT [5] bit will be set). If the value of NFSBLK is bigger than that of NFEBLK, entire area will be locked.	0
Soft Lock	[16]	Soft Lock configuration 0: Disable lock 1: Enable lock Soft lock area can be modified at any time by software. When it is set to 1, the area setting in NFSBLK (0x70200020) to NFEBLK (0x70200024) is unlocked, and except this area, write or erase command will be invalid and only read command is valid. When you try to write or erase locked area, the illegal access will be occurred (NFSTAT [5] bit will be set). If the NFSBLK and NFEBLK are same, entire area will be locked.	1
Reserved	[15:13]	Reserved. Should be written to 0.	000
EnbECCDecINT	[12]	4-bit, 8-bit ECC decoding completion interrupt control 0: Disable interrupt 1: Enable interrupt	0
8bitStop	[11]	8-bit ECC encoding/decoding operation initialization. 8bit ECC module generates parity code for 512/24 byte data. If you want to stop generating ECC parity before completing current work, you must set this value to “1” for initializing 8bit ECC module. This bit will be cleared automatically.	0

NFCONT	Bit	Description	Initial State
EnbIllegalAccINT	[10]	Illegal access interrupt control 0: Disable interrupt 1: Enable interrupt Illegal access interrupt will occurs when CPU tries to program or erase locking area (the area setting in NFSBLK (0x70200020) to NFEBLK (0x70200024).	0
EnbRnBINT	[9]	RnB status input signal transition interrupt control 0: Disable RnB interrupt 1: Enable RnB interrupt	0
RnB_TransMode	[8]	RnB transition detection configuration 0: Detect rising edge 1: Detect falling edge	0
MainECCLock	[7]	Lock Main area ECC generation 0: Unlock Main area ECC 1: Lock Main area ECC Main area ECC status register is NFMECC0/1(0x70200034/38),	1
SpareECCLock	[6]	Lock Spare area ECC generation. 0: Unlock Spare ECC 1: Lock Spare ECC Spare area ECC status register is NFSECC(0x7020003C),	1
InitMECC	[5]	1: Initialize main area ECC decoder/encoder (write-only) Caution : In case of 8bit ECC, you must set this bit carefully. If you set this bit before completing current encoding/decoding, it cause some trouble to 8bit ECC module. If you want to stop current work and start encoding/decoding for new data, you must set 8bitStop(NFCONT[11]) before this bit.	0
InitSECC	[4]	1: Initialize spare area ECC decoder/encoder (write-only)	0
Reserved	[3]	Reserved	0
Reg_nCE1	[2]	NAND Flash Memory Xm0CSn3 signal control 0: Force Xm0CSn3 to low(Enable chip select) 1: Force Xm0CSn3 to High(Disable chip select) **Note**: Even Reg_nCE1 and Reg_nCE0 are set to zero simultaneously, only one of them is asserted.	1
Reg_nCE0	[1]	NAND Flash Memory Xm0CSn2 signal control 0: Force Xm0CSn2 to low(Enable chip select) 1: Force Xm0CSn2 to High(Disable chip select) **Note**: This value is only valid while MODE bit is 1	1
MODE	[0]	NAND Flash controller operating mode 0: NAND Flash Controller Disable (Don't work) 1: NAND Flash Controller Enable	0

```
 9 void select_chip(void)
10 {
11         NFCONT &= ~(1 << 1);
12 }
13
14 void delselect_chip(void)
15 {
16         NFCONT |= (1 << 1);
17 }
```

2.2.3、清除和检测RnB信号（发送命令前需要清除RnB信号）

8.11.10 NFCON STATUS REGISTER

Register	Address	R/W	Description	Reset Value
NFSTAT	0x70200028	R/W	NAND Flash operation status register	0x0080001D

NFSTAT	Bit	Description	Initial State
Reserved	[31:24]	Read undefined	0x00
Reserved			
Reserved	[22:7]	Reserved	0x00
ECCDecDone	[6]	When 4-bit ECC or 8-bit ECC decoding is finished, this value set and issue interrupt if enabled. The NFMLCBITPT, NFMLCL0 and NFMLCEL1 have valid values. To clear this, write '1'. 1: 4-bit ECC or 8-bit ECC decoding is completed	0
IllegalAccess	[5]	Once Soft Lock or Lock-tight is enabled, The illegal access (program, erase) to the memory makes this bit set. 0: illegal access is not detected 1: illegal access is detected	0
RnB_TransDetect	[4]	When RnB low to high transition is occurred, this value set and issue interrupt if enabled. To clear this write '1'. 0: RnB transition is not detected 1: RnB transition is detected Transition configuration is set in RnB_TransMode(NFCONT[8]).	1
NCE[1] (Read-only)	[3]	The status of Xm0CSn3 output pin	1
NCE[0] (Read-only)	[2]	The status of Xm0CSn2 output pin	1
Reserved	[1]	Reserved	0
RnB (Read-only)	[0]	The status of RnB input pin. 0: NAND Flash memory busy 1: NAND Flash memory ready to operate	1

```
19 void clear_RnB(void)
20 {
21        NFSTAT |= (1 << 4);
22 }

29 void wait_RnB(void)
30 {
31        while (!(NFSTAT & 0x1)) ;
32 }
```

2.2.4、发送命令

8.11.4 COMMAND REGISTER

Register	Address	R/W	Description	Reset Value
NFCMMD	0x70200008	R/W	NAND Flash command set register	0x00

NFCMMD	Bit	Description	Initial State
Reserved	[31:8]	Reserved	0x00
NFCMMD	[7:0]	NAND Flash memory command value	0x00

```
24 void send_command(unsigned long cmd)
25 {
26       NFCMMD = cmd;
27 }
```

2.2.5、发送地址

8.11.5 ADDRESS REGISTER

Register	Address	R/W	Description	Reset Value
NFADDR	0x7020000C	R/W	NAND Flash address set register	0x0000XX00

REG_ADDR	Bit	Description	Initial State
Reserved	[31:8]	Reserved	0x00
NFADDR	[7:0]	NAND Flash memory address value	0x00

```
34 void send_addr(unsigned long addr)
35 {
36      NFADDR = addr;
37 }
```

2.2.6、存取数据

8.11.6 DATA REGISTER

Register	Address	R/W	Description	Reset Value
NFDATA	0x70200010	R/W	NAND Flash data register	0xXXXX

NFDATA	Bit	Description	Initial State
NFDATA	[31:0]	NAND Flash read/program data value for I/O (Note) Refer to **DATA REGISTER CONFIGURATION**.	0xXXXX

8.6 DATA REGISTER CONFIGURATION

1. 8-bit NAND Flash Memory Interface

A. Word Access

Register	Bit [31:24]	Bit [23:16]	Bit [15:8]	Bit [7:0]
NFDATA	4th I/O[7:0]	3rd I/O[7:0]	2nd I/O[7:0]	1st I/O[7:0]

B. Half-word Access

Register	Bit [31:24]	Bit [23:16]	Bit [15:8]	Bit [7:0]
NFDATA	Invalid value	Invalid value	2nd I/O[7:0]	1st I/O[7:0]

C. Byte Access

Register	Bit [31:24]	Bit [23:16]	Bit [15:8]	Bit [7:0]
NFDATA	Invalid value	Invalid value	Invalid value	1st I/O[7:0]

因为NAND FLASH的数据口为8位，所以存取是会根据定义的数据类型进行操作，但每次操作只能存取8bit。因为内存每个地址为8bit，所以建议使用unsigned char来定义数据类型，便于操作内存。

```
69 void NandFlash_PageRead(unsigned long addr, unsigned char * buff)
70 {
71      int i;
72      //select nand flash
73      select_chip();
74
75      //clear R/B signal
76      clear_RnB();
77      //send command
78      send_command(0x00);
79
80      //send column address (2 cycle)
81      send_addr(0x00);
82
83      send_addr(0x00);
84
85      //send row address (3 cycle)
86      send_addr(addr & 0xff);
```

```
87
88      send_addr((addr >> 8) & (0xff));
89
90      send_addr((addr >> 16) & (0xff));
91
92      //send command
93      send_command(0x30);
94
95      //wait R/B signal
96      wait_RnB();
97
98      //read data
99      for( i = 0; i < 1024 * 4; i++) {
100             buff[i] = NFDATA;
101     }
102
103     //cancel select nand flash
104     delselect_chip();
105
106 }
```

2.2.7、从NAND FLASH复制U-boot到内存中

2.2.7.1、汇编中调用拷贝函数

```
42 reset:
43      bl set_svc
44      bl peri_port_setup
45      bl disable_watchdog
46      bl disable_interrupt
47      bl flush_cache
48      bl disable_mmu_cache
49      bl system_clock_init
50      bl men_ctrl_asm_init
51      bl stack_init
52      bl nand_init
53
54      bl copy_to_ram
55 @      bl iram_to_ram
56      bl clear_bss
57      ldr pc, _uboot_main
58 @      bl led_on_asm
59
60 _uboot_main:
61      .word uboot_main
```

调用前必须先初始化栈，对NAND FLASH初始化后进行拷贝。

```
119 void nand_to_ram(unsigned long start_addr, unsigned char* sdram_addr, int size)
120 {
121     int i;
122
123     for (i = 0; i < 4; i++, sdram_addr += 2048) {
124         NandFlash_PageRead(i,sdram_addr);
125     }
126
127     size -=1024*8;
128
129     for ( i=4; size > 0; ) {
130         NandFlash_PageRead(i,sdram_addr);
131         size -= 4096;
132         sdram_addr += 4096;
133         i++;
134     }
135 }
```

注意：

当系统以Nand方式启动时，硬件将Nand Flash的前8KB拷贝到Steppingstone，然后从0地址开始运行程序，在这8KB以内代码

中，我们需要完成必要的硬件初始化，如果代码超过8K，我们还需要将剩余代码的搬移到链接地址处，一般在SDRAM/DDR中。其中，硬件部分需要初始化系统时钟、DDR和NAND Flash三部分。

S3C6410启动时拷贝的8K代码不是存储在Nand flash的第一页上，而是存储在Nand flash的前4页上，每页2K，总共8K，这是S3C6410芯片的硬件结构决定的！也就是说，虽然我们的Nand flash的页大小是8K，但是S3C6410为了适应各种型号的Nand flash并保证硬件能正确的拷贝Nand flash中的前8K到SRAM中，硬性的加上这一规定。

问题探究：

我使用如下拷贝函数从NAND FLASH复制U-boot到内存中，也能正常实现按键中断。

```
/*void nand_to_ram(unsigned long start_addr, unsigned char* sdram_addr, int size)
{
        int i;
        for (i = start_addr >> 12; size > 0; i++) {
                NandFlash_PageRead(i,sdram_addr);
                size -= 4096;
                sdram_addr += 4096;
        }
}*/
```

分析原因为：

根据对程序反汇编，可以得到下面的地址分布。

root@localhost:/home/S3-ARM/Part13

File Edit View Search Terminal Help

```
500086d0:       e2822602        add     r2, r2, #2097152        ; 0x200000
500086d4:       e2822028        add     r2, r2, #40     ; 0x28
500086d8:       e3a03207        mov     r3, #1879048192 ; 0x70000000
500086dc:       e2833602        add     r3, r3, #2097152        ; 0x200000
500086e0:       e2833028        add     r3, r3, #40     ; 0x28
500086e4:       e5933000        ldr     r3, [r3]
500086e8:       e3833010        orr     r3, r3, #16     ; 0x10
500086ec:       e5823000        str     r3, [r2]
}
500086f0:       e28bd000        add     sp, fp, #0      ; 0x0
500086f4:       e8bd0800        pop     {fp}
500086f8:       e12fff1e        bx      lr

500086fc <send_command>:

void send_command(unsigned long cmd)
{
500086fc:       e52db004        push    {fp}            ; (str fp, [sp, #-4]!)
50008700:       e28db000        add     fp, sp, #0      ; 0x0
50008704:       e24dd00c        sub     sp, sp, #12     ; 0xc
50008708:       e50b0008        str     r0, [fp, #-8]
        NFCMMD = cmd;
5000870c:       e3a03287        mov     r3, #1879048200 ; 0x70000008
50008710:       e2833602        add     r3, r3, #2097152        ; 0x200000
50008714:       e51b2008        ldr     r2, [fp, #-8]
50008718:       e5832000        str     r2, [r3]
}
5000871c:       e28bd000        add     sp, fp, #0      ; 0x0
50008720:       e8bd0800        pop     {fp}
50008724:       e12fff1e        bx      lr

50008728 <wait_RnB>:

void wait_RnB(void)
{
50008728:       e52db004        push    {fp}            ; (str fp, [sp, #-4]!)
5000872c:       e28db000        add     fp, sp, #0      ; 0x0
        while (!(NFSTAT & 0x1)) ;
```

```
root@localhost:/home/S3-ARM/Part13
File  Edit  View  Search  Terminal  Help
278 5000891c:       e1a00003        mov     r0, r3
279 50008920:       e51b1014        ldr     r1, [fp, #-20]
280 50008924:       ebffffbb        bl      50008818 <NandFlash_PageRead>
281                 size -= 4096;
282 50008928:       e51b3018        ldr     r3, [fp, #-24]
283 5000892c:       e2433a01        sub     r3, r3, #4096   ; 0x1000
284 50008930:       e50b3018        str     r3, [fp, #-24]
285                 sdram_addr += 4096;
286 50008934:       e51b3014        ldr     r3, [fp, #-20]
287 50008938:       e2833a01        add     r3, r3, #4096   ; 0x1000
288 5000893c:       e50b3014        str     r3, [fp, #-20]
289 }
290
291 void nand_to_ram(unsigned long start_addr, unsigned char* sdram_addr, int size)
292 {
293         int i;
294         for (i = start_addr >> 12; size > 0; i++) {
295 50008940:       e51b3008        ldr     r3, [fp, #-8]
296 50008944:       e2833001        add     r3, r3, #1      ; 0x1
297 50008948:       e50b3008        str     r3, [fp, #-8]
298 5000894c:       e51b3018        ldr     r3, [fp, #-24]
299 50008950:       e3530000        cmp     r3, #0  ; 0x0
300 50008954:       cafffffef       bgt     50008918 <nand_to_ram+0x28>
301                 NandFlash_PageRead(i,sdram_addr);
302                 size -= 4096;
303                 sdram_addr += 4096;
304         }
305 }
306 50008958:       e24bd004        sub     sp, fp, #4      ; 0x4
307 5000895c:       e8bd4800        pop     {fp, lr}
308 50008960:       e12fff1e        bx      lr
309 Disassembly of section .ARM.attributes:
310
311 00000000 <.ARM.attributes>:
312    0:   00002541        andeq   r2, r0, r1, asr #10
313    4:   61656100        cmnvs   r5, r0, lsl #2
314    8:   01006962        tsteq   r0, r2, ror #18
315    c:   0000001b        andeq   r0, r0, fp, lsl r0
```

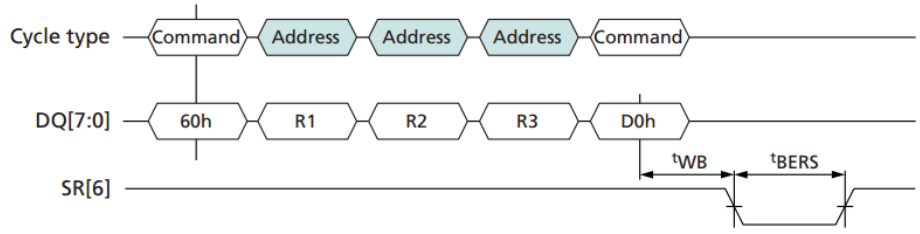
说明处于2K之后的代码都是NAND FLASH操作程序，而这些代码都在IRAM中执行完毕了，通过LDR跳转到uboot_main后的代码都处于2K内，所以运行不会出现问题。

三、NandFlash写操作

NandFlash写操作前需要先擦除。

3.1、NandFlash擦除（块）

Figure 50: ERASE BLOCK (60h-D0h) Operation



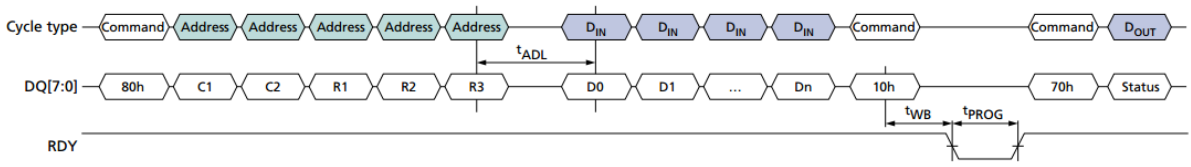
OK6410的NandFlash每块包含128页。

```
137 int Erase_NandFlash(unsigned long addr)
138 {
139     int status;
140
141     select_chip();
142
143     clear_RnB();
144
145     //send command 0x60
146     send_command(0x60);
147
148     //send row address (3 cycle)
149     send_addr(addr & 0xff);
```

```
150
151         send_addr((addr >> 8) & (0xff));
152
153         send_addr((addr >> 16) & (0xff));
154
155         //send command 0xD0
156         send_command(0xd0);
157
158         //wait RnB
159         wait_RnB();
160
161         //send command 0x70
162         send_command(0x70);
163
164         //read status
165         status = NFDATA;
166         //
167         delselect_chip();
168
169         return status;
170 }
```

3.2、NandFlash写操作时序配置

Figure 46: PROGRAM PAGE (80h-10h) Operation




```
172 int NandFlash_PageWrite(unsigned long addr, unsigned char * buff)
173 {
174         int i,status;
175
176         //select chip
177         select_chip();
178
179         //clean RnB
180         clear_RnB();
181
182         //send command 0x80
183         send_command(0x80);
184
185         //send column address (2 cycle)
186         send_addr(0x00);
187
188         send_addr(0x00);
189
190         //send row address (3 cycle)
191         send_addr(addr & 0xff);
192
193         send_addr((addr >> 8) & (0xff));
194
195         send_addr((addr >> 16) & (0xff));
196
197         //write data
198         for( i = 0; i < 1024 * 4; i++) {
199                 NFDATA = buff[i];
200         }
201
202         //send command 0x10
203         send_command(0x10);
204
```

```
205     //wait RnB signal
206     wait_RnB();
207
208     //send read status command 0x70
209     send_command(0x70);
210
211     //read status
212     status = NFDATA;
213     //
214     //delselect chip
215     delselect_chip();
216
217     return status;
218 }
```

3.2、验证读写成功

```
 1 int uboot_main()
 2 {
 3      unsigned char buff[1024*4];
 4
 5 #ifdef MMU_ON
 6      mmu_init();
 7 #endif
 8
 9      led_init();
10
11      button_init_ext_int();
12
13      irq_init();
14
15      led_on();
16
17      Erase_NandFlash(128*1+1);
18
19      buff[0] = 100;
20
21      NandFlash_PageWrite(128*1+1,buff);
22
23      buff[0] = 10;
24
25      NandFlash_PageRead(128*1+1,buff);
26
27      if (buff[0] == 100) {
28          led_off();
29      }
30
31      while(1) ;
32
33      return 0;
34 }
```