# 最小uboot的在S3C2440中的实现

Uboot的根本目的在于加载内核，下面以三星S3C2440为平台，实现一个最小uboot来加载内核。

涉及到的芯片手册如下：



## Stage one

### 关看门狗



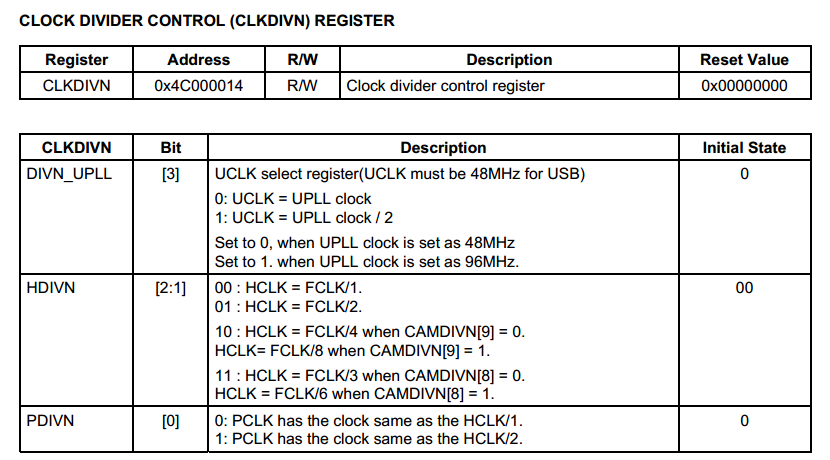
ldr r0, =0x53000000

mov r1, #0

str r1, [r0]

### 设置时钟

为了让boot跑得更快，需要对输入的12Mhz的时钟频率进行倍频操作。



ldr r0, =0x4c000014

mov r1, #0x03; // FCLK:HCLK:PCLK=1:2:4, HDIVN=1,PDIVN=1

str r1, [r0]

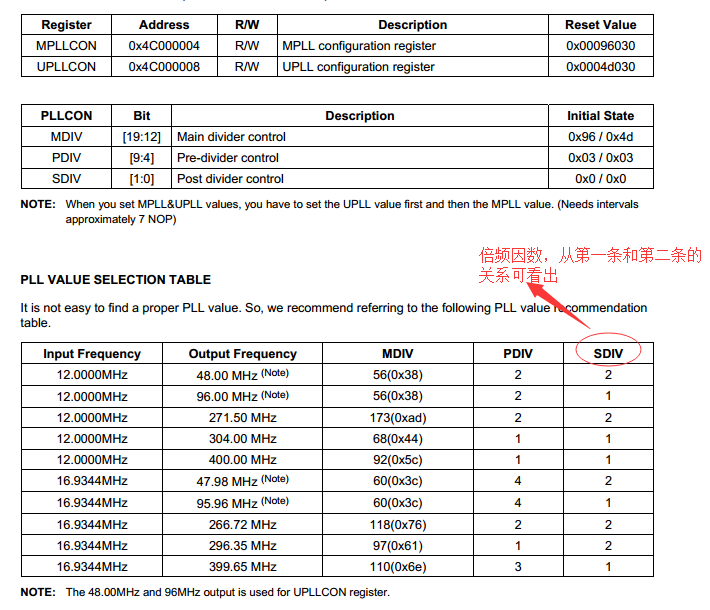


mrc p15, 0, r1, c1, c0, 0 /\* 读出控制寄存器 \*/

orr r1, r1, #0xc0000000 /\* 设置为“asynchronous bus mode” \*/

mcr p15, 0, r1, c1, c0, 0 /\* 写入控制寄存器 \*/

从datasheet中可知，当HDIVN非0时，需要切换CPU的总线模式，执行以上三条指令：Move to ARM Register r0 from Coprocessor p15，对r0寄存器进行操作，设置为“asynchronous bus mode”，Move to Coprocessor p15 from ARM Register r0.



ldr r0, =0x4c000004

ldr r1, = ((0x5c<<12)|(0x01<<4)|(0x02))

str r1, [r0]

设置输出的时钟频率为200MHZ。

### 初始化SDRAM

#define MEM\_CTL\_BASE 0x48000000

ldr r0, =MEM\_CTL\_BASE

adr r1, sdram\_config /\* sdram\_config的当前地址 \*/

add r3, r0, #(13\*4)

1:

ldr r2, [r1], #4 /\* loads R2 from 4 bytes above the address in R1, and

increments R1 by 4.\*/

str r2, [r0], #4 /\* 把r2的值存到r0所指的地方，r0+4 \*/

cmp r0, r3

bne 1b

sdram\_config:

.long 0x22011110 //BWSCON

.long 0x00000700 //BANKCON0

.long 0x00000700 //BANKCON1

.long 0x00000700 //BANKCON2

.long 0x00000700 //BANKCON3

.long 0x00000700 //BANKCON4

.long 0x00000700 //BANKCON5

.long 0x00018005 //BANKCON6

.long 0x00018005 //BANKCON7

.long 0x008C04F4 //REFRESH

.long 0x000000B1 //BANKSIZE

.long 0x00000030 //MRSRB6

.long 0x00000030 //MRSRB7

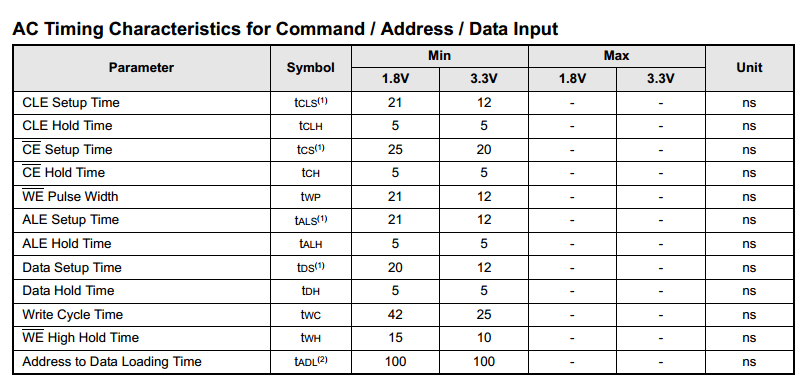
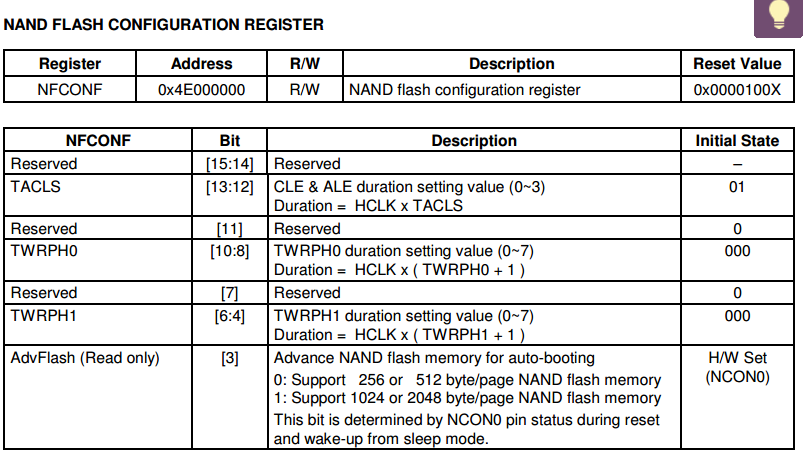
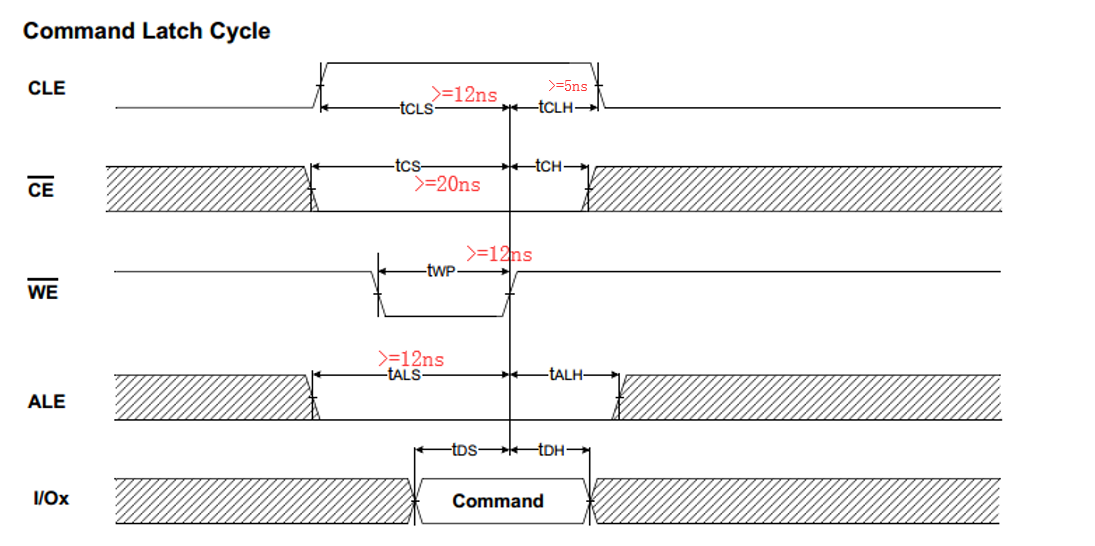
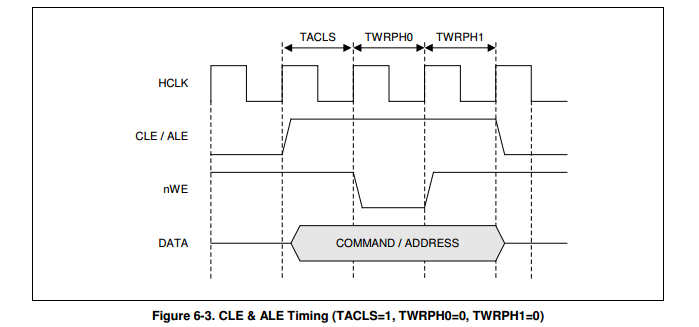
将各寄存器的信息复制到MEM\_CTL\_BASE所对应的内存地址中。

### 重定位

把bootloader本身的C部分代码从flash复制到它的链接地址去。

void copy\_code\_to\_sdram(unsigned char \*src, unsigned char \*dest, unsigned int len)

#### 初始化NAND

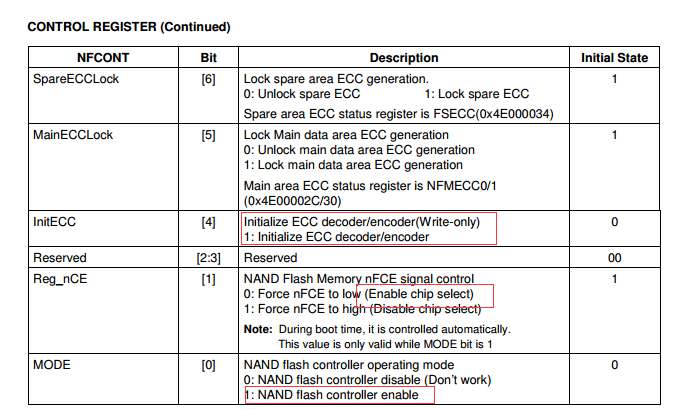
  

根据以上时序图、寄存器表、参数表，依次设置寄存器。

TACLS：Duration = 1/100mhz \* TACLS >= tcls – twp = 0， 得TACLS >= 0，此处设置为0

TWRPH0：Duration = 1/100mhz \* (TWRPH0 + 1) >= 12ns 得TWRPH0 >=0.2 ，此处设置为1

TWRPH1：Duration = 1/100mhz \* (TWRPH0 + 1) >= 5ns 得TWRPH0 >=0，此处设置为0



使能NAND Flash控制器, 初始化ECC, 禁止片选。

对应代码如下：

void nand\_init(void)

{

#define TACLS 0

#define TWRPH0 1

#define TWRPH1 0

/\* 设置时序 \*/

NFCONF = (TACLS<<12)|(TWRPH0<<8)|(TWRPH1<<4);

/\* 使能NAND Flash控制器, 初始化ECC, 禁止片选 \*/

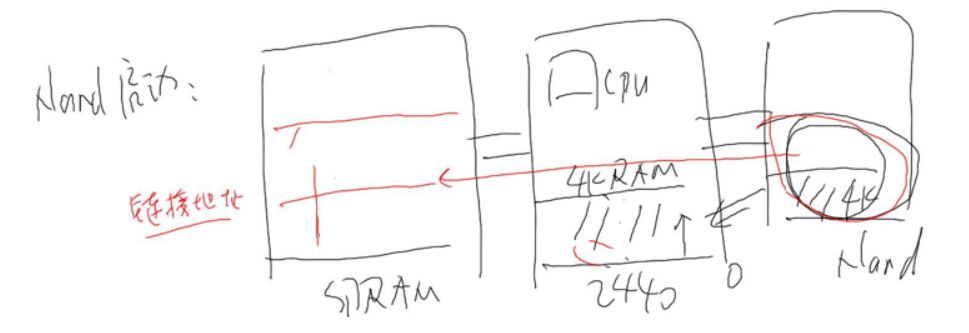
NFCONT = (1<<4)|(1<<1)|(1<<0);

}

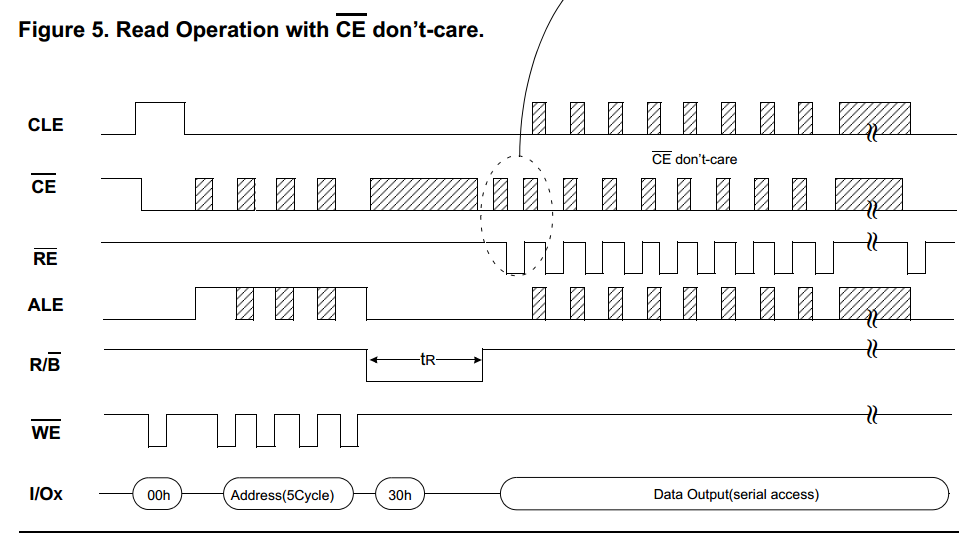
#### copy\_code\_to\_sdram

##### 若是从nand flash启动

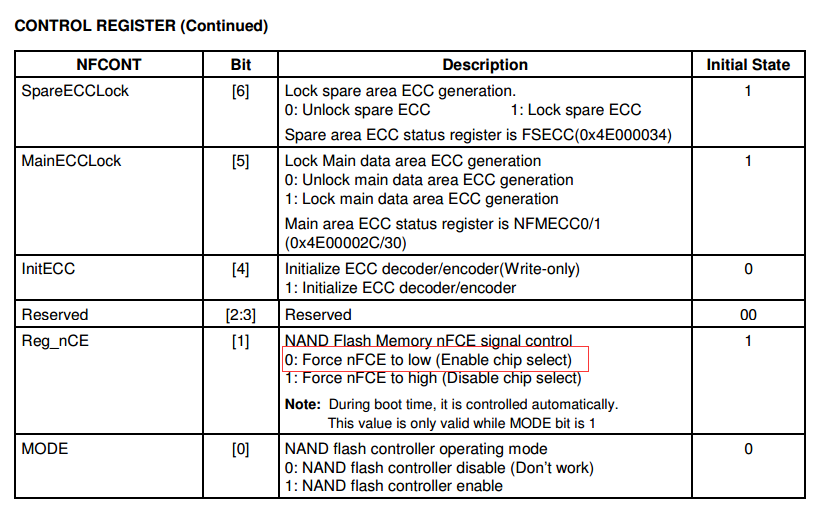
当uboot从nand flash启动时，简要框图如下所示。CPU从NAND中读取4k数据到本芯片上执行，此时CPU的0地址指向CPU上的4kRAM上，若要将NAND上BootLoader的C部分代码复制到SDRAM的链接地址上，需要对NAND进行读操作，然后赋值至链接地址处的相应内存中。



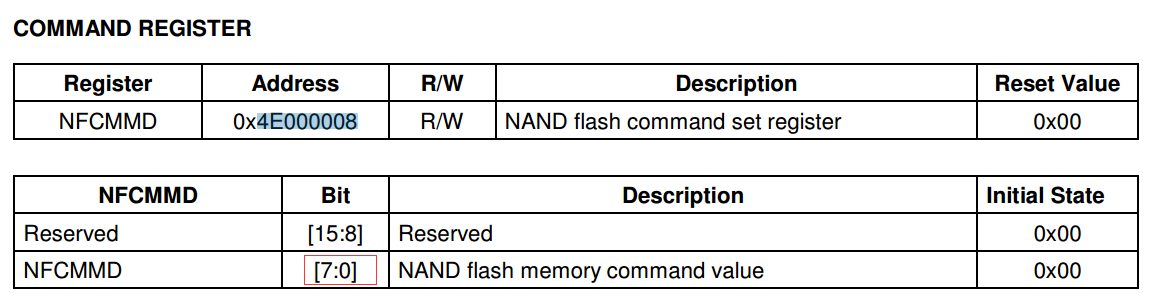
1. 时序



1. 选中

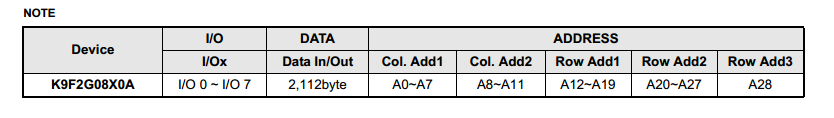


1. 发00h命令



1. 发地址

地址格式如下所示：



12位列地址，17位页地址，总共29位地址，需要用四个字节表示。

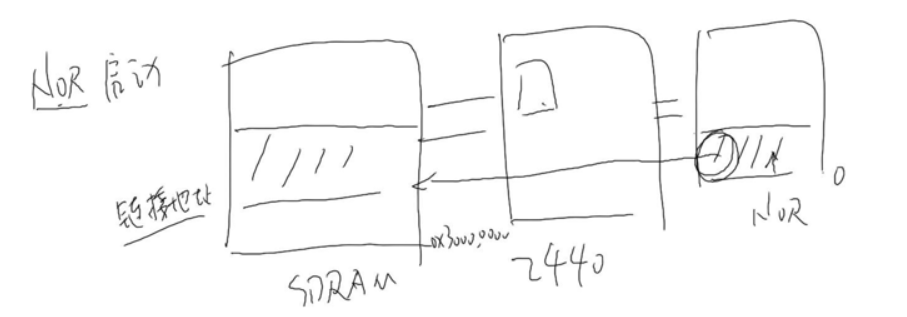
从下图标可见，地址和地址之间需要延时。



1. 发出读命令，可读寄存器就绪之后，循环读取。
2. 取消选中，做2的逆操作。

##### 若是从nor flash启动

其简要框图如下所示，此时CPU运行的0地址即在NOR上，可直接拷贝。



while (i < len)

{

dest[i] = src[i];

i++;

}

#### 调用clear\_bss

将bss段清零。

void clear\_bss(void)

{

extern int \_\_bss\_start, \_\_bss\_end;

int \*p = &\_\_bss\_start;

for (; p < &\_\_bss\_end; p++)

\*p = 0;

}

### 执行main

在这里进入第二阶段。

## Stage two

### 从NAND FLASH里把内核读入内存

nand\_read(0x60000+64, (unsigned char \*)0x30008000, 0x200000);

源地址：0x60000+64，uImage存在NAND FLASH的0x60000地址处，zImage为内核的一种映像压缩文件。

目的地址：内存中划给内核的地址。

长度：64\*1024\*1024

### 设置参数

setup\_start\_tag();

setup\_memory\_tags();

setup\_commandline\_tag("noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0");

setup\_end\_tag();

通过对static struct tag \*params一一赋值，将uboot需要告知内核的启动参数按约定传递给内核。

### 跳转执行

theKernel = (void (\*)(int, int, unsigned int))0x30008000;

theKernel(0, 362, 0x30000100);

以上启动内核的c代码等价于以下的汇编代码：

/\*

\* mov r0, #0

\* ldr r1, =362

\* ldr r2, =0x30000100

\* mov pc, #0x30008000

\*/

参数1代表从内核0地址处开始执行。

参数2代表machine ID，用来标识单板相关信息。

参数3代表以上设置参数后的参数表起始地址\*params。

## 调试

自己写strcpy和strlen，编译选项使用CPPFLAGS := -nostdinc -nostdlib **-fno-builtin**，否则会提示warning。

在内核起不来之后，定位到问题可能出现在读NAND的操作中。

对比正确读取后内存中存放的内核镜像的值，将通过现有的NAND\_READ函数读到的值取出，写一个puthex函数来验证这个地址中存放的hex值是否与正确读取后内存中存放的内核镜像的第一个十六进制值相等，若不等，则证明NAND\_READ函数确实有问题。后来发现Addr的/和%弄反，导致地址处理错误，没有读出正确的值。

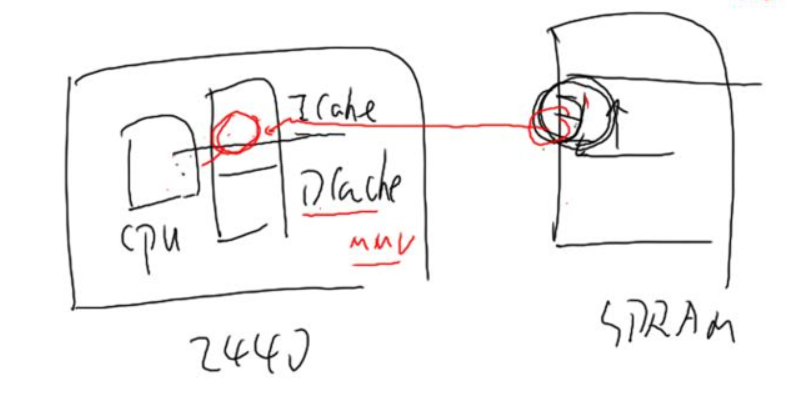
## 改进

### 提高CPU频率

200MHZ ==> 400MHZ

### 启动ICACHE

ICACHE原理框图见下：



将内存中常用的一段命令放到ICACHE中，不用频繁进行内存读写。

/\* 启动ICACHE \*/

mrc p15, 0, r0, c1, c0, 0 @ read control reg

orr r0, r0, #(1<<12)

mcr p15, 0, r0, c1, c0, 0 @ write it back