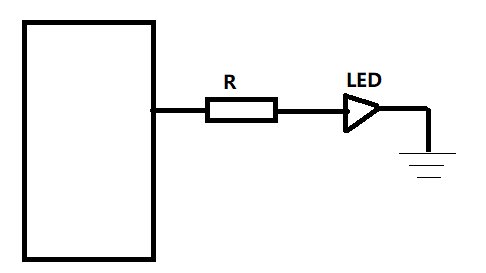
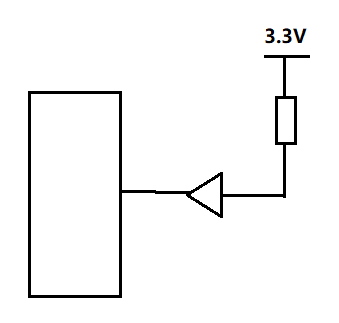
# 点亮LED的电路原理图

## 主芯片GPIO（General Purpose Input Output ）电压足够点亮LED

### 高电平点亮

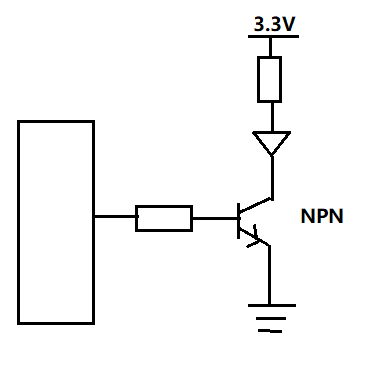


### 低电平点亮

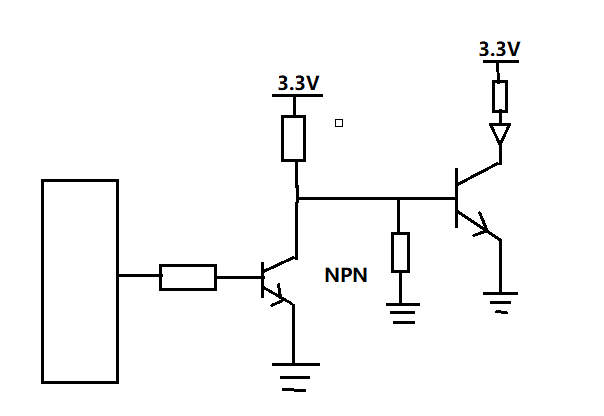


## 主芯片GPIO电压不足以点亮LED

### 高电平点亮



### 低电平点亮



三极管中，P指向N

# 怎么让GPIO输出高低电平？

查看原理图和芯片手册

* 原理图

R58 
R59 
R63 
1K 
1K 
1K 
1K 
LEDI 
LED 
LED 
LED3 
LED4 
>XED4/GPA26 
[21 
[21 
[21 
[21 

所以需要设置GPB5,GPB6,GPA25,GPA26为低电平，能将对应的LED点亮

* 芯片手册

GPA26 
GPA25 
Output only 
Output only 
DQM3 
DQM2 

GPB6 
GPB5 
Input/output 
Input/output 
nXBREQ 
nXBACK 
XBREQ 
XBACK 
RTCK 

对于GPA25和26来说，只能最为输出引脚，所以直接设置GPADAT即可，

而GPB5和6，既能作为输出引脚，又能作为输入引脚，所以需要先设置GPBCON让其变为输出引脚，再设置GPBDAT来控制引脚的输出

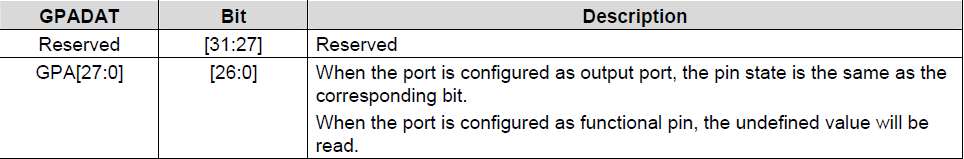
对于GPA来说：

Register 
G pACON 
GPADAT 
Address 
ox56000004 
ox5600000c 
Description 
R/W Configures the pins of port A 
R/W The data register for port A 
Reset Value 
Ox0fffffff 
Oxo 

GPACON：

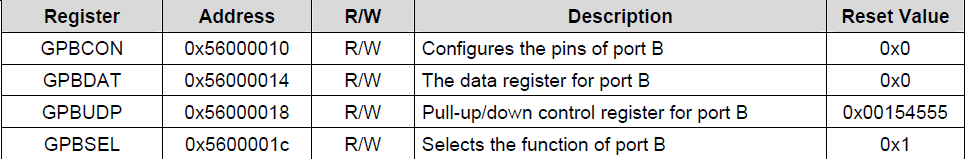


GPADAT：

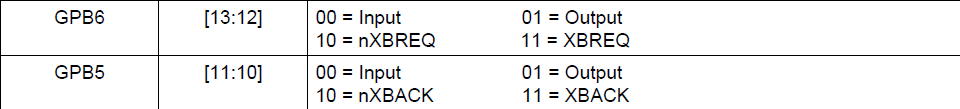


所以，我们需要向(\*0x5600 0000) & 0xf9ffffff，可以将GPA25和26设为OUTPUT； （\*0x5600 0004）&0xf9ffffff，将输出高电平

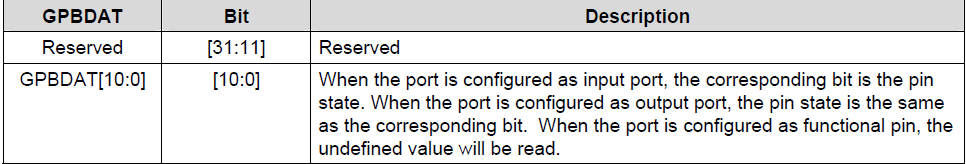
对于GPB来说：



GPBCON：

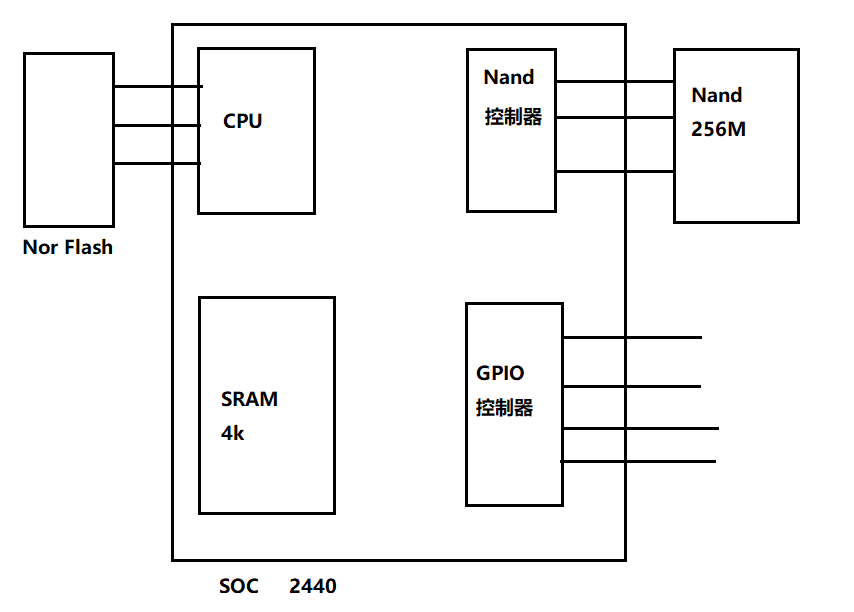


GPBDAT：



所以，我们将（\*0x5600 0010 ）& 0xffffd7ff，将（\*0x5600 0014）& 0xffffff9f

# 2440框架和启动过程



Soc表示，不仅有CPU，还有其他一些外设，都集成在一起，是一个小系统

GPFCON和CPFDAT，不是CPU的寄存器，是GPIO控制器的寄存器，所以不能直接访问，只能用地址来访问

## 启动过程 ？

1. Nor启动

Nor基地址位0，片内Ram地址为0x4000 0000

CPU先读出Nor上第一个指令（前4字节），并开始执行，然后读出其他指令

1. Nand启动

片内4K RAM，基地址为0，此时无法访问Nor Flash

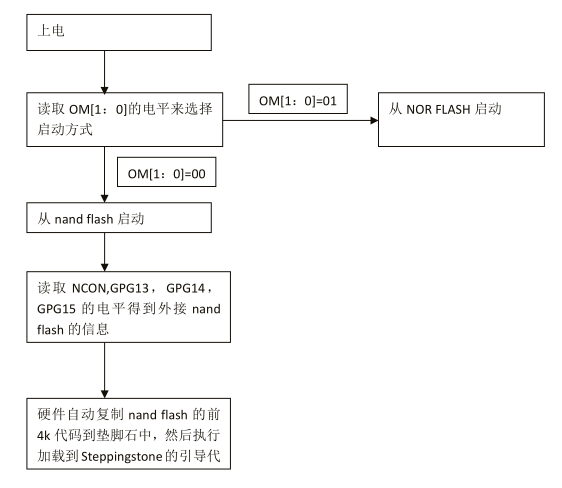
2440硬件，将Nand前4k内容，copy到片内内存上，然后从0地址去除第一条指令开始执行

S3C2440A引导代码可以在外部NAND Flash存储器上存储。为了支持从NAND Flash启动，S3C2440A配备了一个内置的SRAM缓冲器，叫做“Steppingstone(垫脚石)”。从nand flash启动时，NAND Flash存储器的前4K字节将被加载到Steppingstone中并且执行加载到Steppingstone的引导代码。

复制大小为多少：4K。

复制到哪里去：Steppingstone(垫脚石)。

复制完接着做什么：执行加载到Steppingstone的引导代码。



# 一些汇编指令

1. LDR（load） 例： LDR R0,[R1] #设R1值为x，将x读入R0中（4字节）
2. STR（store） 例： STR R0,[R1] #设R1值为x，将R0的值，写到地址x所指空间（4字节）
3. B（跳转指令）
4. MOV（move） MOVE R0, R1 #等价于 R0 = R1

MOVE R0，#0x100 #等价于 R0=0x100

（5） LDR R0，=0x1345678

#伪指令，并不存在该指令，且最后会被拆分为几条真正的ARM指令，最后使得R0=0X12345678

因为对于32位系统来说，指令的数据为并没有32位，无法满足长度（32位中肯定有一些控制位）

所以MOVE用于简单数的赋值，即立即数

注意，该处的LDR只是名字和（1）中的相同，但是含义不同

# Coding与Compiling

## Led.S

注意

（1）文件名是大写的S，这样在用arm-linux-gcc 编译时，可以识别C语言风格的注释

（2）第一行代码不能用 #来注释

.text //表示代码段

.global \_start

\_start:

#将 GPA25，25设置位输出引脚

ldr r1,=0x56000000

ldr r0,=0xf9ffffff

str r0,[r1]

#点亮LED

ldr r1,=0x56000004

ldr r0,=0

str r0,[r1]

#while(1)

halt:

b halt

## Makefile

all:

arm-linux-gcc -c -o led\_on.o led\_on.S

arm-linux-ld -Ttext 0 led\_on.o -o led\_on.elf

arm-linux-objcopy -O binary -S led\_on.elf led\_on.bin

clean:

rm -rf \*.bin \*.o \*.elf

## 反汇编

arm-linux-objdump -D led\_on.elf > led\_on.dis

# 2440中的register

2440中一共有16个寄存器：R0~R15

|  |  |  |
| --- | --- | --- |
| 寄存器名称 | 别名 | 作用 |
| R13 | SP | Stack pointer 栈指针 |
| R14 | LR | Link register 保存返回地址 |
| R15 | PC | Program counter，PC指针 |

PC指针指向哪，就去读取那个位置的指令

因为流水线的原因，PC指针的位置 = 当前指令位置 + 8

## 2440的流水线

（以下三个操做，同时进行，所以至少三级流水线）

执行当前地址A的指令

对A+4指令进行译码

对A+8指令进行读取（PC指针的位置）

# 解读反汇编与二进制文件

## 反汇编指令

arm-linux-objdump -D led\_on.elf > led\_on.dis

## led\_on.dis中的内容

led\_on.elf: file format elf32-littlearm

Disassembly of section .text:

00000000 <\_start>:

0: e3a01456 mov r1, #1442840576 ; 0x56000000

4: e3e00406 mvn r0, #100663296 ; 0x6000000

8: e5810000 str r0, [r1]

c: e59f1008 ldr r1, [pc, #8] ; 1c <halt+0x4>

10: e3a00000 mov r0, #0

14: e5810000 str r0, [r1]

00000018 <halt>:

18: eafffffe b 18 <halt>

1c: 56000004 strpl r0, [r0], -r4

Disassembly of section .ARM.attributes:

00000000 <.ARM.attributes>:

0: 00001941 andeq r1, r0, r1, asr #18

4: 61656100 cmnvs r5, r0, lsl #2

8: 01006962 tsteq r0, r2, ror #18

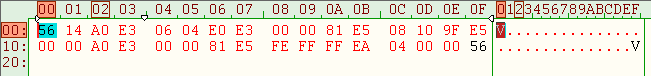
c: 0000000f andeq r0, r0, pc

10: 00543405 subseq r3, r4, r5, lsl #8

14: 01080206 tsteq r8, r6, lsl #4

18: Address 0x00000018 is out of bounds.

## Led\_on.bin 二进制文件中的内容



## 解读

c: e59f1008 ldr r1, [pc, #8]

[pc,#8] pc指向的的内容再+8,

现在当前的地址为c，即12，所以地址为 12+8+8 = 28，即0x1c

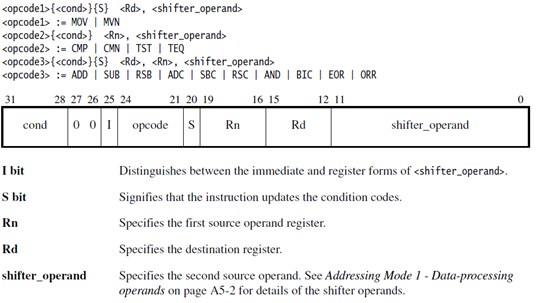
而0x1c的内容为： 1c: 56000004 strpl r0, [r0], -r4

所以与之前的 ldr r1,=0x56000004 相同

0: e3a01456 mov r1, #1442840576

10: e3a00000 mov r0, #0

查看arm机器码手册，得到mov指令格式：



Move的opcode为 1101

12—15位，表示R0目的寄存器

0—11位，表示数据，其中高四位，是移位数，低八位是数字

最终立即数 = 第八位 循环右移 （2 \* 移位数）

例如：0x5600 0000

0x456 == 0100 0101 0110 == （0101 0110） >> (2 \* 0100) == (0101 0110)循环右移8位

---> 0101 0110 0000 0000 0000 0000 0000 00000 == 0x5600 0000

所以 mov r1, #1442840576 的机器码为： e3a01456

如果想用其他值，可以自己计算后，直接修改二进制码

## 总结

GPFCON/GPFDAT在CPU的角度，把他们当作内存来使用

# 为何使用二进制？

进踢馆只有两种状态，1和0

所以数据使用多个晶体管来表示

# 使用C语言点亮LED

## Led.c

int main()

{

unsigned int \*pGPACON = (unsigned int \*)0x56000000;

unsigned int \*pGPADAT = (unsigned int \*)0x56000004;

\*pGPACON &= 0xf9ffffff;

\*pGPADAT &= 0xf9ffffff;

return 0;

}

## Start.s 启动文件

.text

.global \_start

\_start:

/\*设置内存：sp 栈 \*/

ldr sp,=4096 /\*Nand 启动，栈设为片内RAM顶部\*/

# ldr sp,=0x40000000+4096 #Nor启动

/\*调用main\*/

bl main

halt:

b halt

## Makefile

all:

arm-linux-gcc -c -o led.o led.c

arm-linux-gcc -c -o start.o start.S

arm-linux-ld -Ttext 0 start.o led.o -o led.elf

arm-linux-objcopy -O binary -S led.elf led.bin

arm-linux-objdump -D led.elf > led.dis

clean:

rm -rf \*.o \*.elf \*.dis \*.bin