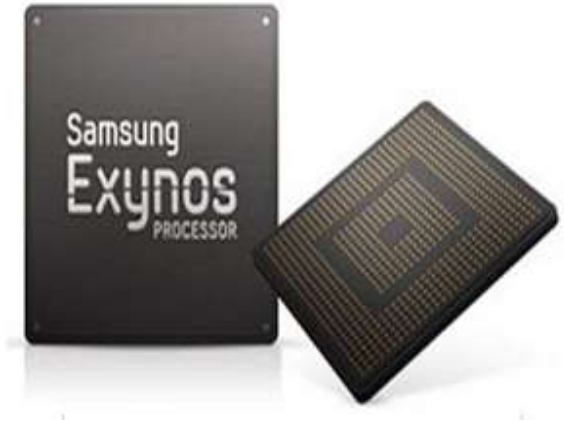
Exynos 4412的GPIO

- **Exynos 4412**处理器总览
- **GPIO**接口
- **GPIO**应用实例

5.1 Exynos 4412处理器

Exynos 4412是三星公司推出的一款基于Cortex-A9的RSIC架构的性价比高、功耗低、性能优越的32位处理器。Exynos 4412的内存系统中有专用DRAM端口和静态存储器端口。其中的DRAM端口支持DDR2、LPDDR2和DDR3，静态存储器端口支持FlexOneNAND、NOR Flash和ROM型的外部存储器。



5.2 GPIO接口

5.2.1 GPIO总览

三星Exynos 4412共有304个GPIO，分为GPA0、GPA1、GPB、GPC0、GPC1等共37组，可以通过设置寄存器来确定某个引脚用于输入、输出还是其他特殊功能，比如可以设置GPC0、GPC1作为一般的输入引脚、输出引脚，或者用于AC97、SPDIF、I^2C、SPI口。

雨课堂 Rain Classroom

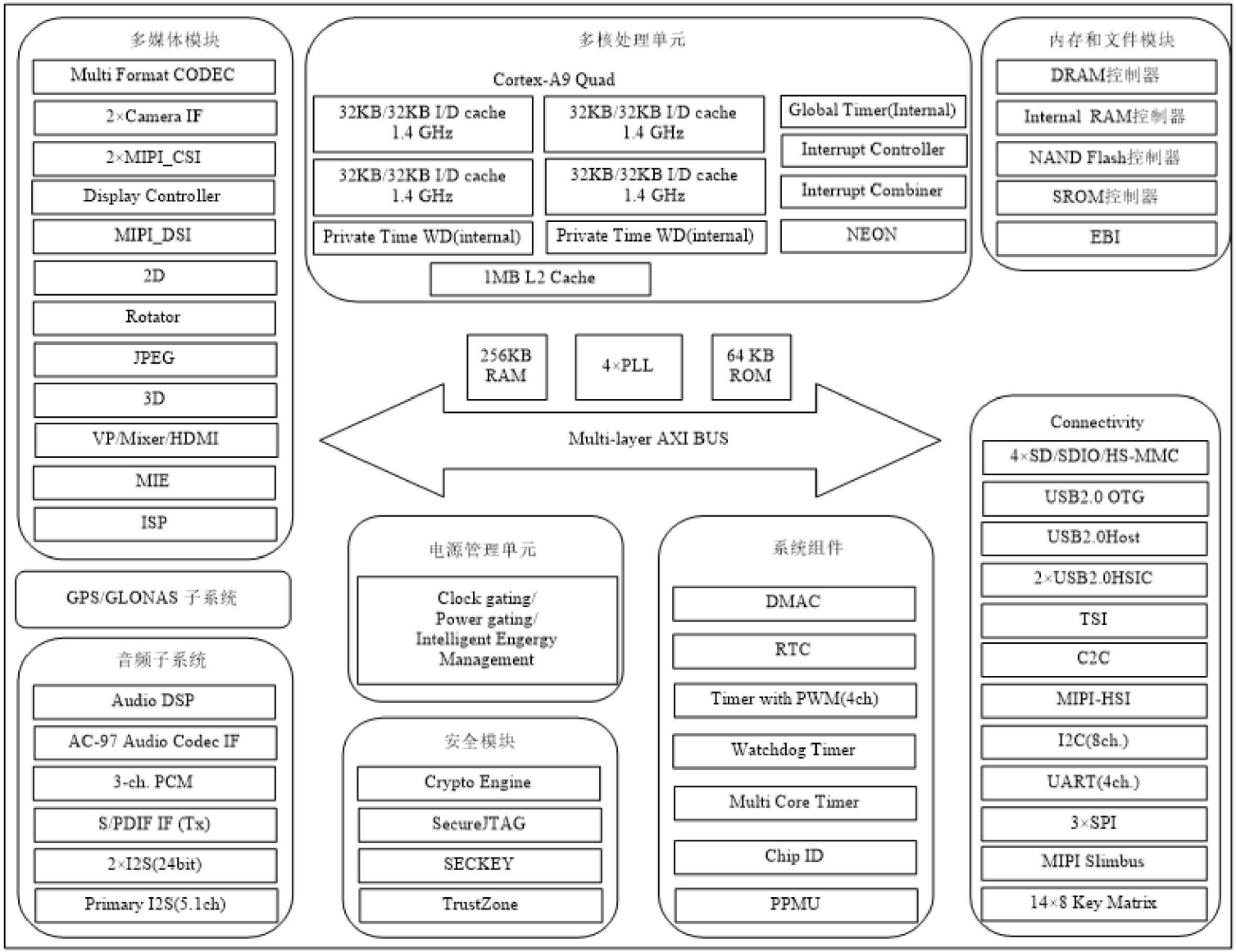


图5.2 Exynos 4412处理器的系统架构和片上硬件资源

37组GPIO分类如下：

(1) GPA0(8个)，GPA1(6个)：共14个I/O口，用于3 × UART(带流控制)和1 × UART(不带流控制)，或2 × I^2C；

(2) GPB：共8个I/O口，用于2 × SPI或2 × I^2C或IEM；

(3) GPC0(5个)，GPC1(5个)：共10个I/O口，用于2 × I^2S或2 × PCM或AC97、S/PDIF、I^2C或SPI；

(4) GPD0(4个)，GPD1(4个)：共8个I/O口，用于PWM、2 × I^2C或LCD I/F、MIPI；

(5) GPF0(8个)，GPF1(8个)，GPF2(8个)，GPF3(6个)：共30个I/O口，用于LCD I/F；

(6) GPJ0(8个)，GPJ1(5个)：共13个I/O口，用于CAM I/F；

(7) GPK0(7个)，GPK1(7个)，GPK2(7个)，GPK3(7个)：共28个I/O口，用于4 × MMC (4-bit MMC)或2 × MMC (8-bit MMC)，或GPS debugging I/F；

(8) GPL0(7个)，GPL1(2个)：共9个I/O口，用于GPS I/F；

(9) GPL2：共8个I/O口，用于GPS debugging I/F或Key pad I/F；

(10) GPM0(8个)，GPM1(7个)，GPM2(5个)，GPM3(8个)，GPM4(8个)：共36个I/O口，用于CAM I/F或TS I/F、HIS或Trace I/F；

(11) GPX0(8个)，GPX1(8个)，GPX2(8个)，GPX3(8个)：共32个I/O口，用于External wake-up或Key pad I/F。

5.2.2 GPIO引脚的配置

在对GPIO引脚配置的时候，一般采用的方法都是对相应的寄存器的值进行配置，从而达到目的。例如，当配置成输入引脚时，可以通过读取某个寄存器来确定引脚的电平是高还是低；当配置成输出引脚时，可以通过写入某个寄存器来让这个引脚输出高或低电平。对于其他特殊功能，则需要有另外的寄存器来配置和控制这些特殊功能。常用的寄存器有GPxxCON寄存器、GPxxDAT寄存器和 GPxxPUD 寄存器等等。

- GPxxCON寄存器：该类寄存器称为配置寄存器，主要用于配置引脚的功能。
- GPxxDAT寄存器：该类寄存器用于读/写引脚。当引脚被设为输入时，读此寄存器可知相应引脚的电平状态；当引脚被设为输出时，写此寄存器相应位可令此引脚输出高电平或低电平。
- GPxxPUD 寄存器：该类寄存器用于确定是否使用内部上拉/下拉电阻，使用两个位来控制一个引脚：其值为0b00 时，相应引脚无内部上拉/下拉电阻；值为0b01时，使用内部下拉电阻；值为0b11时，使用内部上拉电阻；0b10为保留值。

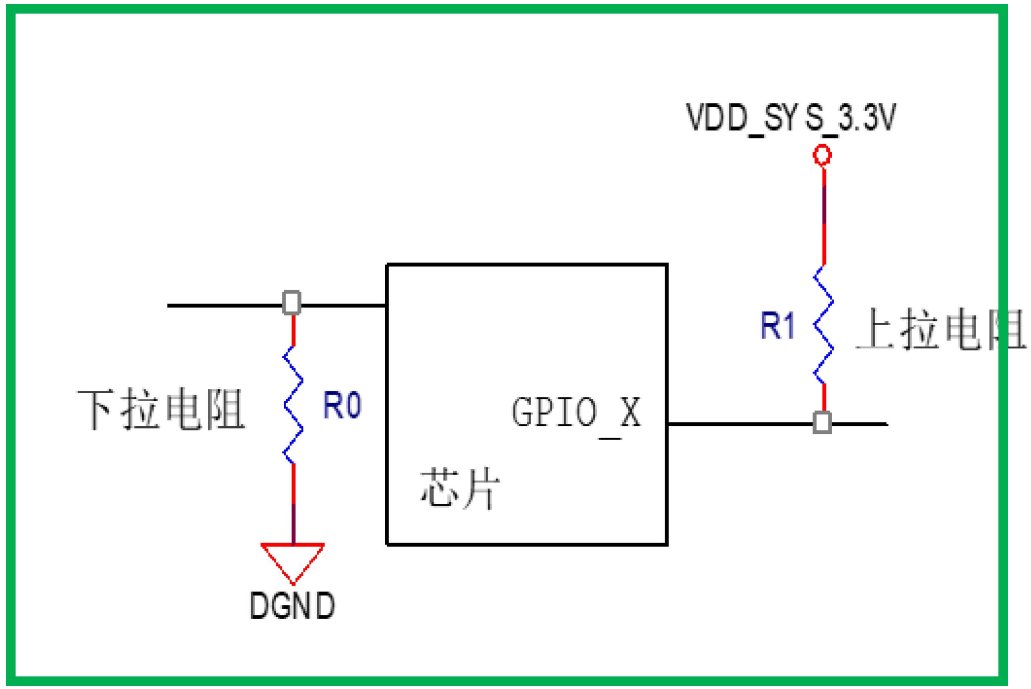


图5.3 上拉电阻和下拉电阻示意图

所谓上拉电阻、下拉电阻，如图5.3所示。上拉电阻、下拉电阻的作用在于，当GPIO引脚处于第三态(既不是输出高电平，也不是输出低电平，而是呈高阻态，即相当于没接芯片)时，它的电平状态由上拉电阻、下拉电阻确定。

5.2.3 GPIO端口寄存器

GPIO端口寄存器多达476个。每个GPIO口组相关的寄存器一般包括配置寄存器GPxxCON、数据寄存器GPxxDAT、上/下拉电阻寄存器GPxxPUD、引脚驱动能力配置寄存器GPxxDRV、省电模式配置寄存器GPxxCONPDN、低功耗模式下上/下拉电阻寄存器GPxxPUDPDN。接下来以GPA0端口为例，给出GPA0口相关寄存器及配置方法，对其他口组也可以进行类似的配置。

1. GPA0配置寄存器GPA0CON

名 称	位域	类型	功 能 描 述	复位值
GPA0CON[0]	[3:0]	RW	0x0=Input；0x1=Output；0x2=UART_0_RXD； 0x3-0xE=保留；0xF=EXT_INT1[0]	0x00
GPA0CON[1]	[7:4]	RW	0x0=Input；0x1=Output；0x2=UART_0_TXD； 0x3-0xE=保留；0xF=EXT_INT1[1]	0x00
GPA0CON[2]	[11:8]	RW	0x0=Input；0x1=Output；0x2=UART_0_CTSn； 0x3-0xE=保留；0xF=EXT_INT1[2]	0x00
GPA0CON[3]	[15:12]	RW	0x0=Input；0x1=Output；0x2=UART_0_RTSn； 0x3-0xE=Reserved；0xF=EXT_INT1[3]	0x00
GPA0CON[4]	[19:16]	RW	0x0=Input；0x1=Output；0x2=UART_1_RXD； 0x3-0xE=保留；0xF=EXT_INT1[4]	0x00
GPA0CON[5]	[23:20]	RW	0x0=Input；0x1=Output；0x2=UART_1_TXD； 0x3-0xE=保留；0xF=EXT_INT1[5]	0x00
GPA0CON[6]	[27:24]	RW	0x0=Input；0x1=Output；0x2=UART_1_CTSn； 0x3=I2C_2_SDA；0x4-0xE=保留；0xF=EXT_INT1[6]	0x00
GPA0CON[7]	[31:28]	RW	0x0=Input；0x1=Output；0x2=UART_1_RTSn； 0x3=I2C_2_SCL；0x4-0xE=保留；0xF=EXT_INT1[7]	0x00

表5.1 GPA0配置寄存器GPA0CON

2. GPA0数据寄存器GPA0DAT

名 称	位域	类型	功 能 描 述	复位值
GPA0DAT[7:0]	[7:0]	RWX	当配置成输入引脚时，相应的位是引脚的状态(高/低电平)；当配置成输出引脚时，引脚的状态(高/低电平)是相应位的值；当配置成其他功能时，读到的是不确定的值	0x00

表5.2 GPA0数据寄存器GPA0DAT

雨课堂 Rain Classroom

3. GPA0上/下拉电阻寄存器GPA0PUD

名 称	位域	类型	功 能 描 述	复位值
GPA0PUD[n]	[2n+1:2n] n = 0~7	RW	0x0 = 禁止启用上/下拉电阻；0x1 = 启用下拉电阻； 0x2 = 保留；0x3 = 启用上拉电阻	0x5555

表5.3 GPA0上/下拉电阻寄存器GPA0PUD

雨课堂 Rain Classroom

4. GPA0引脚驱动能力配置寄存器GPA0DRV

名 称	位域	类型	功 能 描 述	复位值
GPA0DRV[n]	[2n+1:2n] n=0~7	RW	0x0=1x;0x2=2x;0x1=3x;0x3=4x	0x000000

表5.4 GPA0驱动寄存器GPA0DRV

5. GPA0的省电模式配置寄存器GPA0CONPDN

名 称	位域	类型	功 能 描 述	复位值
GPA0PUD[n]	[2n+1:2n] N=0~7	RW	0x0=Output 0；0x1=Output 1； 0x2=Input；0x3=前一电平状态	0x0000

表5.5 GPA0的省电模式配置寄存器GPA0CONPDN

6. GPA0的省电模式上/下拉电阻寄存器GPA0PUDPDN

名称	位域	类型	功 能 描 述	复位值
GPA0[n]	[2n+1:2n] n=0~7	RW	0x0=禁止启用上/下拉电阻；0x1=启用下拉电阻； 0x2=保留；0x3=启用上拉电阻	0x00

表5.6 GPA0的省电模式上/下拉电阻寄存器GPA0PUDPDN

5.3 GPIO应用

本书设计的开发平台和编译环境如下：

- 电路连接：各例程中用到的电路源自三星公司的评估板，并根据需要进行了修改。
- 代码编译环境：UltraEdit。
- 宿主机：VMWare和Fedora 9。
- 编译工具：arm-linux-gcc4.5.1。

5.3.1 电路连接

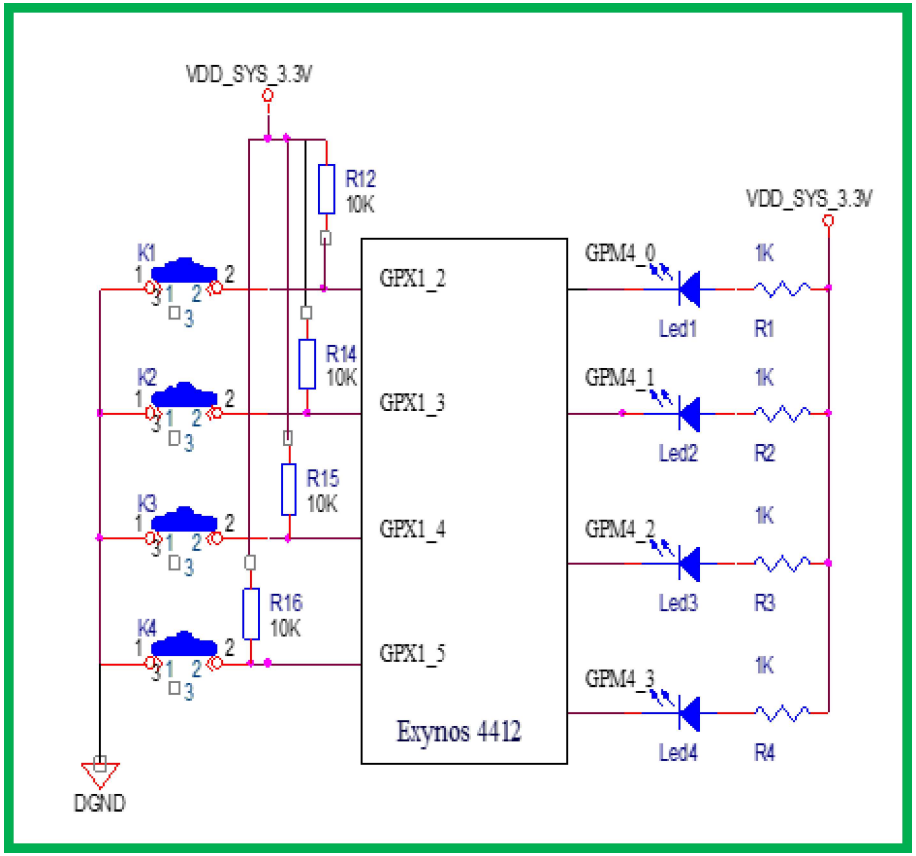


图5.4 按键和LED电路图

任务介绍：

GPM4_0～GPM4_3分别控制四个LED的状态,当这4个I/O引脚为高电平时，LED灯熄灭；当I/O引脚为低电平时LED灯点亮。设置的4个按键K1、K2、K3、K4分别连接GPX1_2、GPX1_3、GPX1_4、GPX1_5。

要做的工作就是设置GPM4_0～GPM4_3为输出功能，且控制其输出电平高低即可。配置GPX1_2～GPX1_5为输入功能，并在相应的数据寄存器中得到相应位的电平状态。

5.3.2 实例1：汇编代码点亮LED灯

程序实现的步骤:

第一步：设置GPM4_0～GPM4_3相对应的配置寄存器GPM4CON，使GPM4_0～GPM4_3四个引脚为输出功能。

第二步：设置其对应的数据寄存器GPM4DAT对应的4个bit位为0，使GPM4_0～GPM4_3引脚为低电平，4个LED灯全亮。等待一段时间后将第0位设置为0，其余位设置为1，这样只有第一个灯亮；其他三个灯类似,只需将相应的位置0即可。

代码实现:

```
.text
.globl _start
_start:
    /* 将GPM4配置称为输出 */
    ldr r0, =0x110002E0   ；GPM4CON寄存器的地址是0x110002E0
    ldr r1, [r0]          ；先读出GPM4CON寄存器的原值
    bic r1, r1, #0xff00       ；清除bit[15:8]
    bic r1, r1, #0xff     ；清除bit[7:0]
    orr r1, r1, #0x1100   ；设置bit[15:8] 为0b00010001
    orr r1, r1, #0x11     ；设置bit[7:0] 为0b00010001
    str r1, [r0]        ；写入GPM4CON
```

```
/* 将GPM4DAT低4位设置为低电平，点亮4个LED灯 */
    ldr r0, =0x110002E4   ；GPM4DAT的地址是0x110002E4
    ldr r1, [r0]          ；读出原值
leds_loop:
    bic r1, r1, #0xf      ；清除bit[3:0]为0，此时bit[3:0] = 0000
    str r1, [r0]          ；写入GPM4DAT，4个LED全亮
    ldr r2,=0xffffff      ；设定延时时间，在delay子程序中使用
    bl delay              ；调用延时子程序delay
```

```
/* 将GPM4DAT位0设置为低电平，点亮LED1灯 */
    orr r1, r1, #0xe          ；0b1110，设置bit[0] 为0, 此时bit[3:0] = 1110
    str r1, [r0]              ；写入GPM4DAT，LED1亮
    ldr r2,=0xffffff
    bl delay
```

```
/* 将GPM4DAT位1设置为低电平，点亮LED2灯 */
    bic r1, r1, #0x3          ；清除r1的bit[1:0] 位，此时bit[3:0] = 1100
    orr r1, r1, #1            ；设置bit[0] 为1，即只有bit[1] 为0，此时  bit[3:0] = 1101
    str r1, [r0]              ；写入GPM4DAT，LED2亮
    ldr r2,=0xffffff
    bl delay
```

```
/* 将GPM4DAT位2设置为低电平，点亮LED3灯 */
    bic r1, r1, #0x6          ；清除r1的bit[1] 和bit[2] 位，此时
bit[3:0] = 1001
    orr r1, r1, #2            ；设置bit[1] 为1，只有bit[2] 为0，此时
bit[3:0] = 1011
    str r1, [r0]              ；写入GPM4DAT，LED3亮
    ldr r2,=0xffffff
    bl delay
```

```
/* 将GPM4DAT位3设置为低电平，点亮LED4灯 */
    bic r1, r1, #0xc          ；清除r1的bit[3] 和bit[2] 位，此时
bit[3:0] = 0011
    orr r1, r1, #4            ；设置bit[2] 为1，只有bit[3] = 0，此时
bit[3:0] = 0111
    str r1, [r0]              ；写入GPM4DAT，LED4亮
    ldr r2,=0xffffff
    bl delay
```

```
/* 将GPM4DAT低4位设置为高电平，熄灭4个LED灯 */
    orr r1, r1, #0xf          ；bit[3:0] = 1111
    str r1, [r0]              ；写入GPM4DAT，4个LED全灭
    ldr r2,=0xffffff
    bl delay
    b leds_loop               ；跳至leds_loop，开始下一次循环
halt_loop:
    b halt_loop
delay:                        ；跑马灯延时程序

     sub r2,r2,#1       ；sub减法
     cmp r2,#0x0              ；将r0与0比较
     bne delay                ；bne是不相等跳转到delay
```

程序编译与烧写:

1) 编译。通过FTP或者其他工具将led.s、Makefile、led.lds三个文件上传到服务器上，输入make命令进行编译，将得到led.bin文件。

(2) 烧写。将SD卡插入电脑，并让VMware里的Fedora识别出来，然后执行如下命令：

```
sudo ./sd_fusing.sh /dev/sdb ../01.led/led.bin
```

5.3.3 实例2：C语言按键控制LED灯

由于汇编语言可读性较差，如果要实现复杂的功能，最好采用C语言进行编程。在这一节我们用C语言来实现按键对LED灯的控制，按键检测采用轮询方式。一上电，4个LED灯全亮，若某个按键被按下，则对应的LED熄灭(可同时按下多个键)。按键和LED灯的对应关系为：KEY1-LED1，KEY2-LED2，KEY3-LED3，KEY4-LED4。

C语言程序就不在此展开了，具体可见课本中的代码分析。

问题与思考：

1. 使用汇编语言，编写按键检测程序。

2. 理解本节中的两个简单Makefile文件。

3. 将本节实例中的按键和LED电路连接互换，重新编写按键控制LED灯的C语言程序。