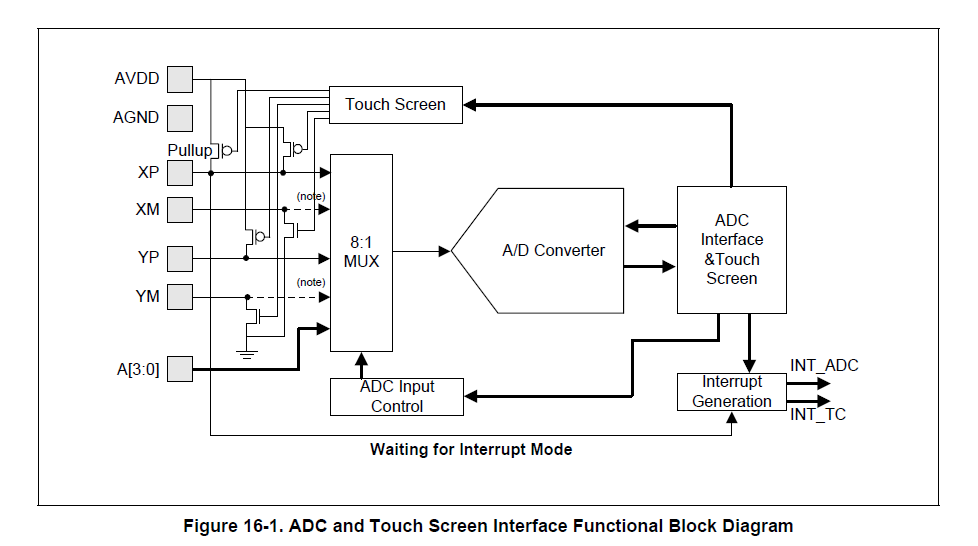
## adc硬件原理

模数转换器即A/D转换器，简称adc，通常是指一个将模拟信号转变为数字信号的电子元件

通常的模数转换器是经过把经过与标准量比较处理后的模拟量转换成以二进制数值表示的离散信号的转换器

故任何一个模数转换器都需要一个参考模拟量作为转换的标准，比较常见的参考标准为最大的可转换信号大小。而输出的数字量则表示输入信号相对于参考信号的大小

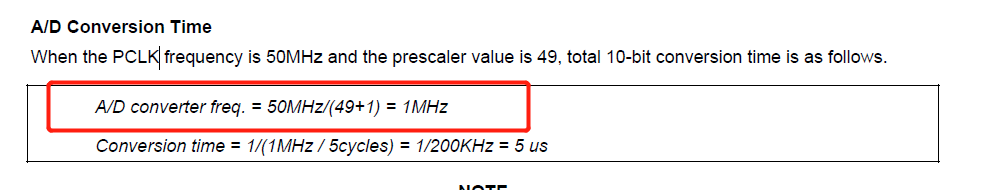


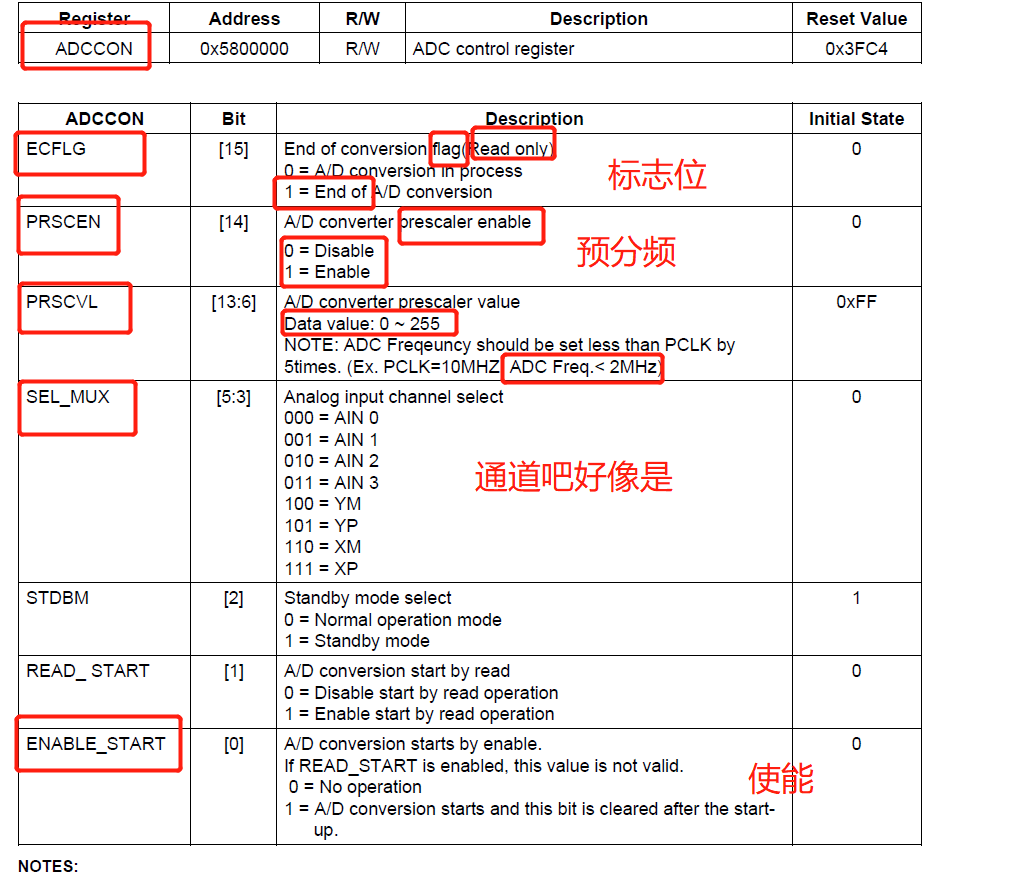
看图有8个多路选择器，编程思路

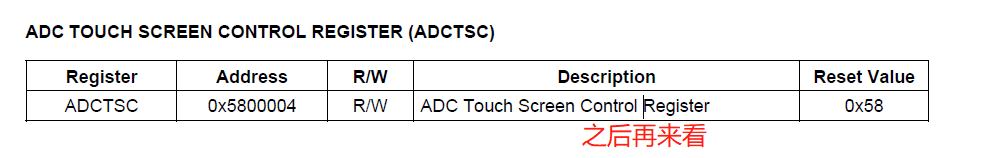
1. 确定是那一路信号：设置8:1MUX，选择要测量哪一个引脚（看原理图）
2. 设置工作时钟（从工作时钟中可以算出转换一次需要多长时间）
3. 启动
4. 读状态，判断adc是否转换成功
5. 读数据

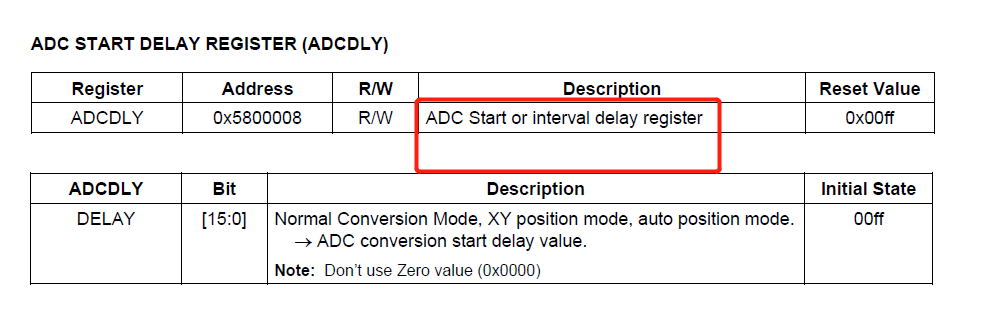
## adc编程

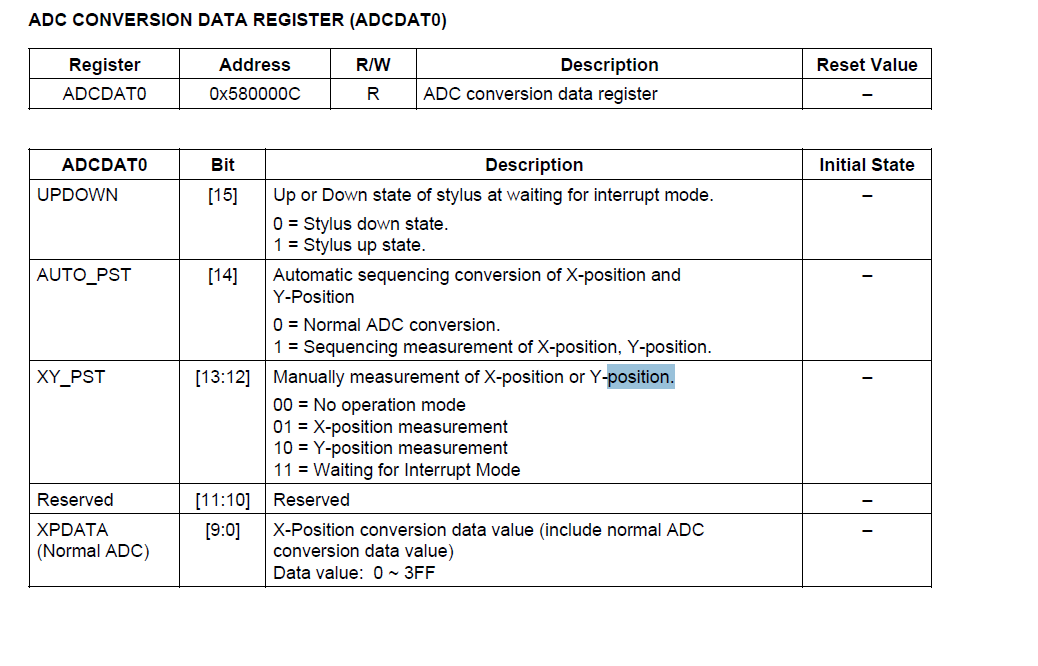
### adc寄存器

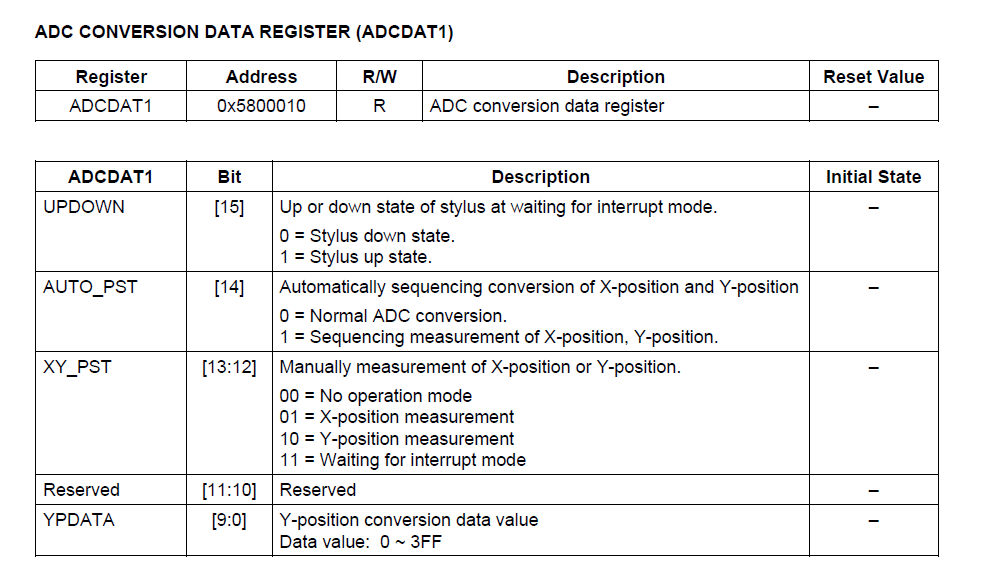


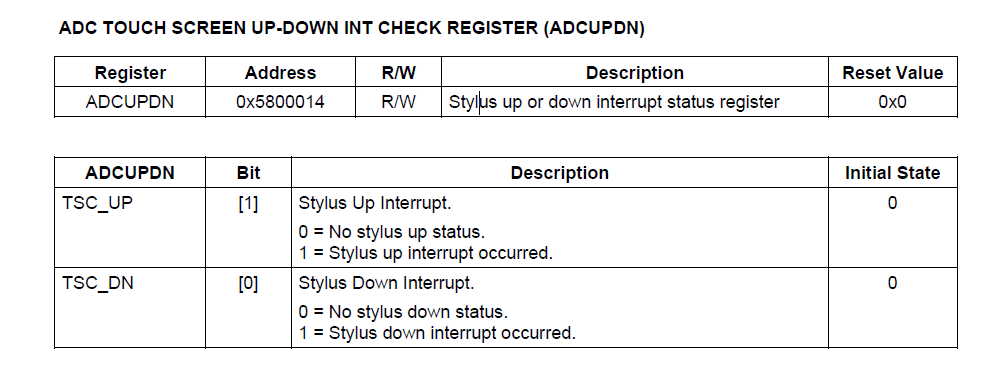












写void adc\_init(void),读数据int adc\_init\_ain0(void)

主要用到上面有红圈的寄存器位

ADCCON

ADCDLY

ADCDAT0 只用到低10位

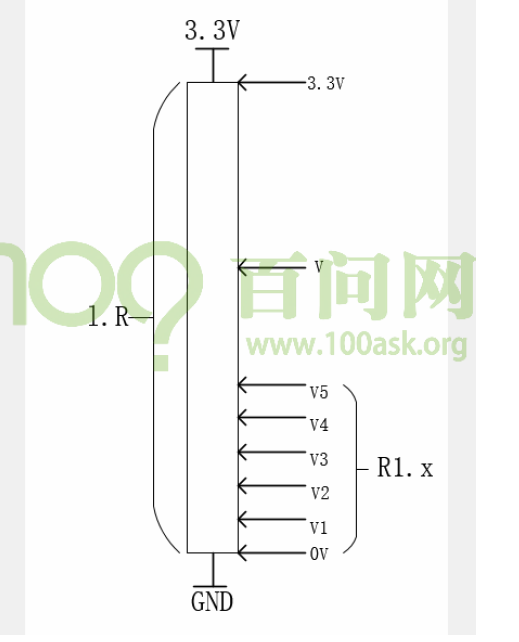
写完

这里因为没有相应模块做不了adc\_test

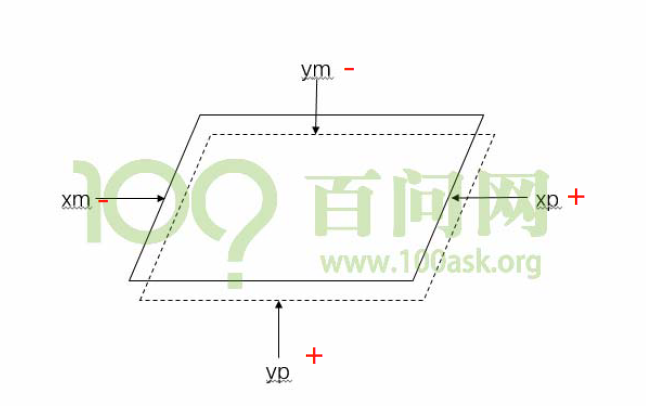
所以就先空着

等后续实现触屏功能的时候顺便做了

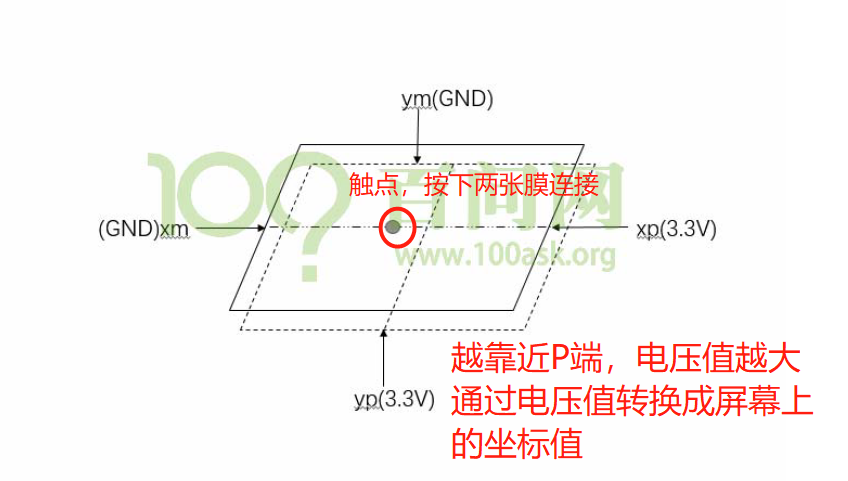
## 电阻触摸屏原理



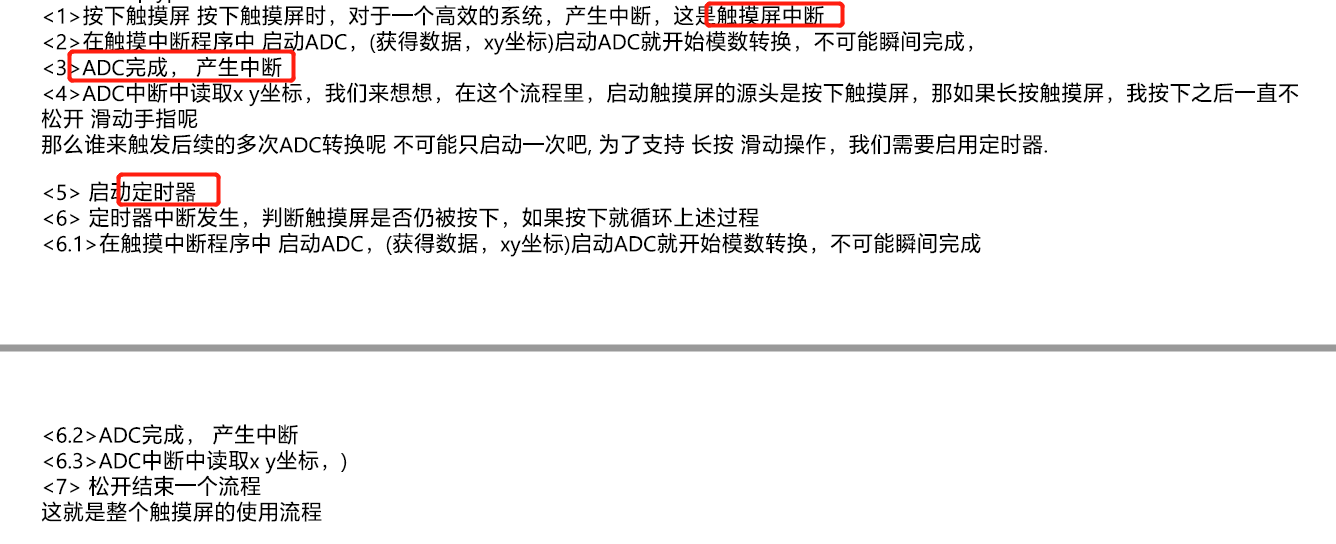
假设电阻是均匀的，通过adc算出电压，间接得出触电x的坐标



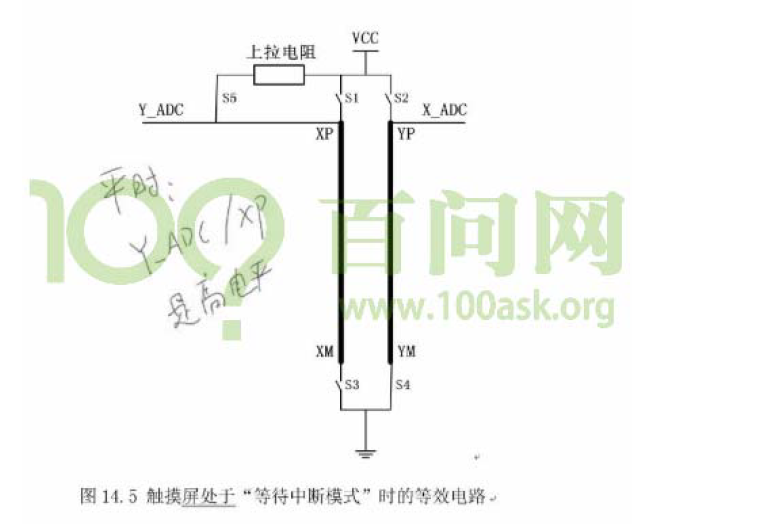
lcd和lcd触摸屏是两个东西，触摸屏是由两层膜组成



使用触摸屏流程：



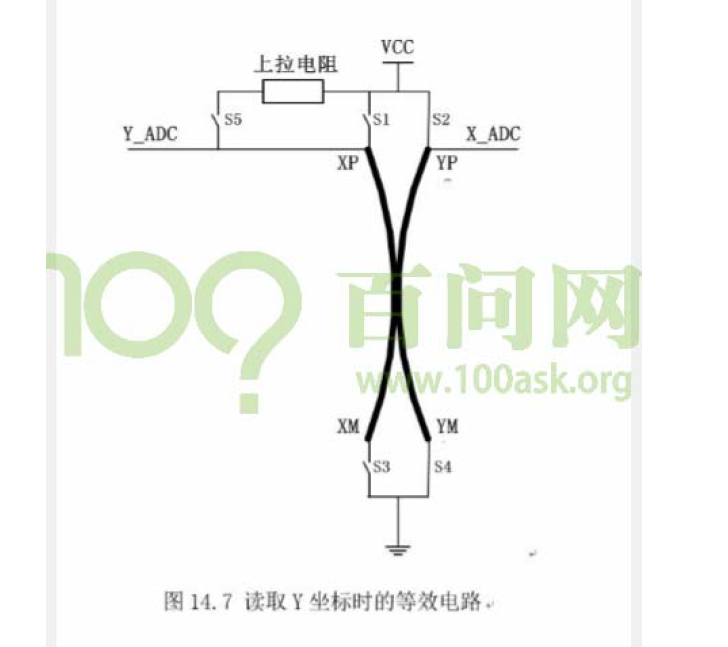
这里用到了触摸屏中断，adc中断，定时器，较为复杂

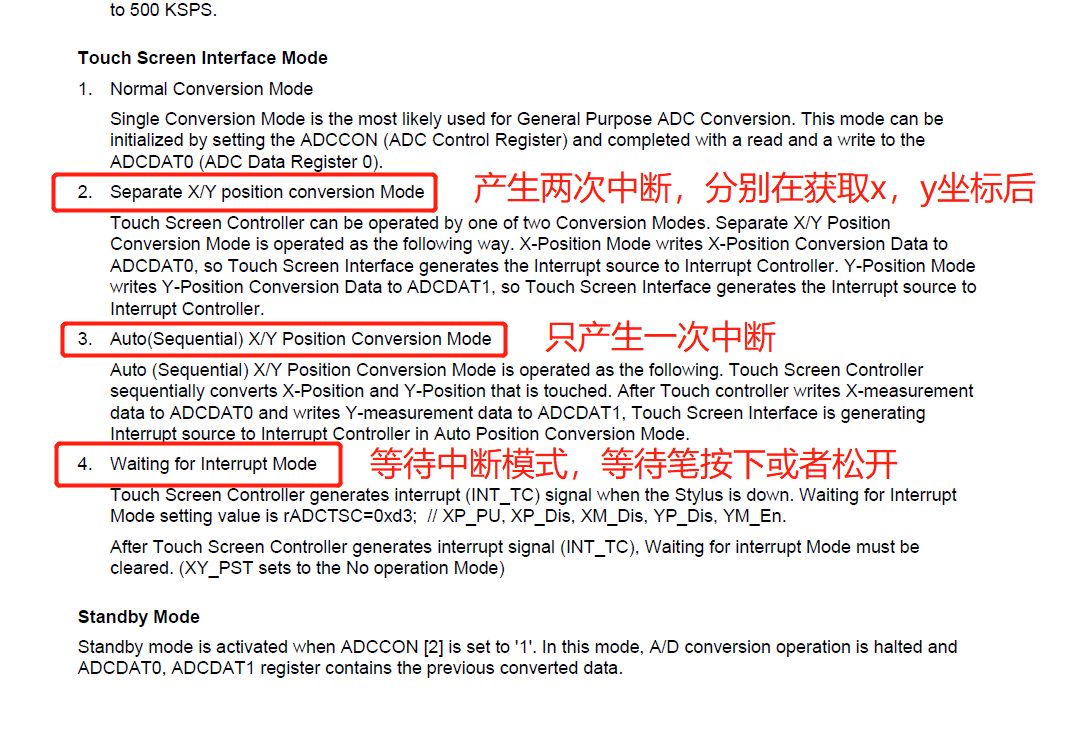


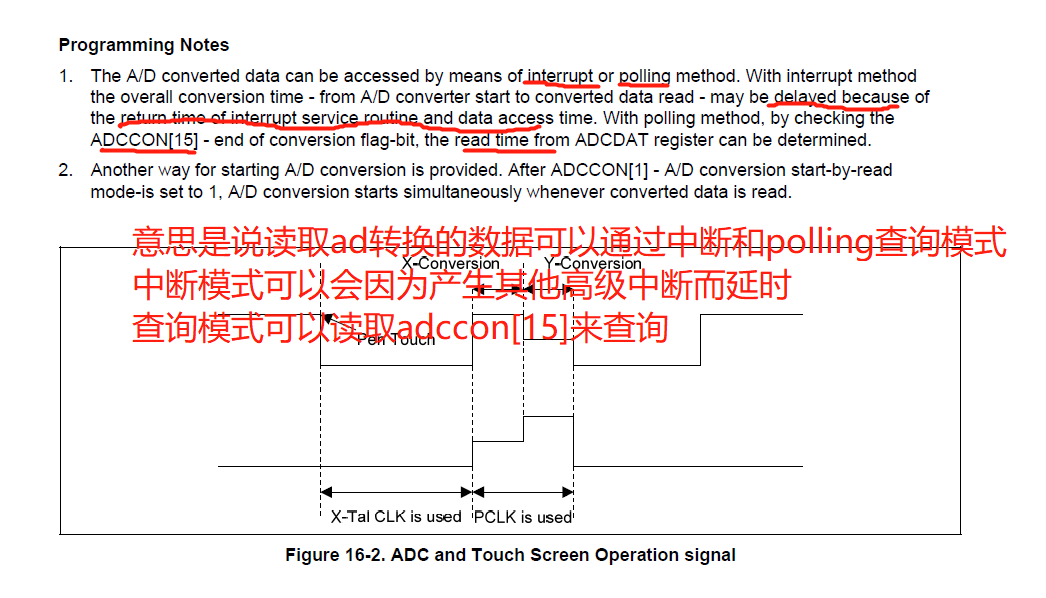
按下，读x坐标



y点同理



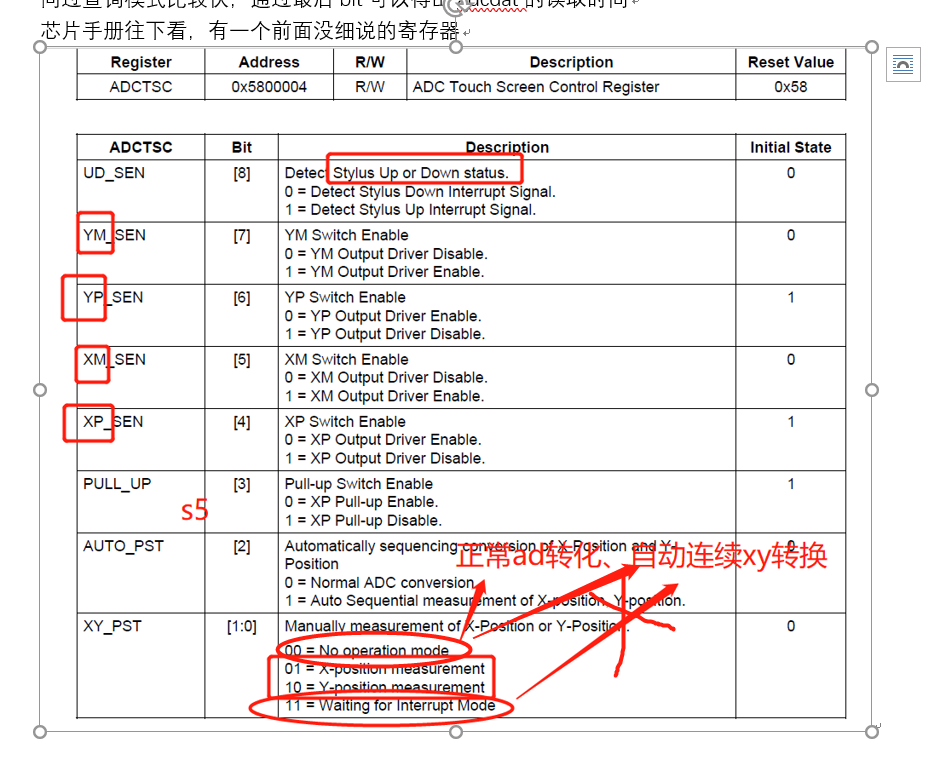


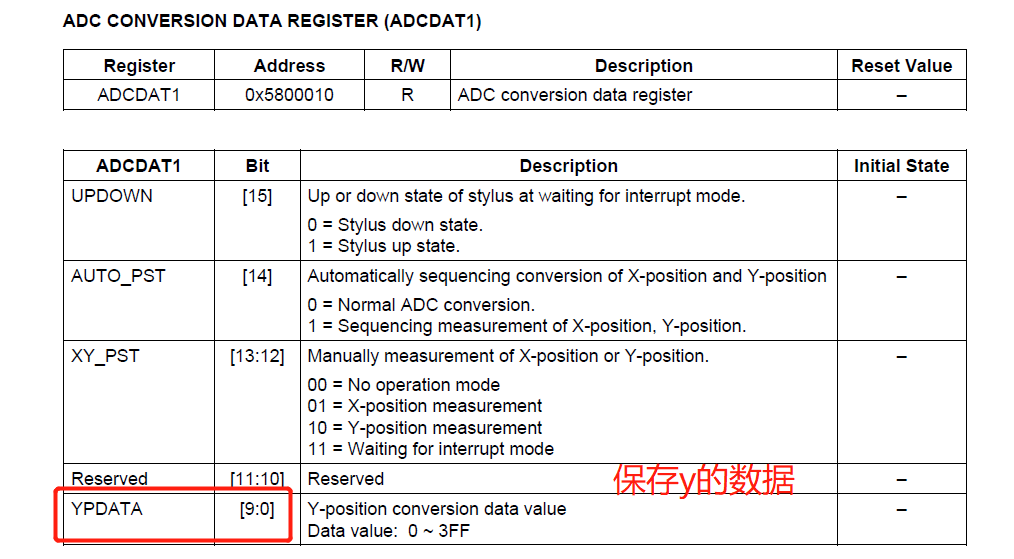
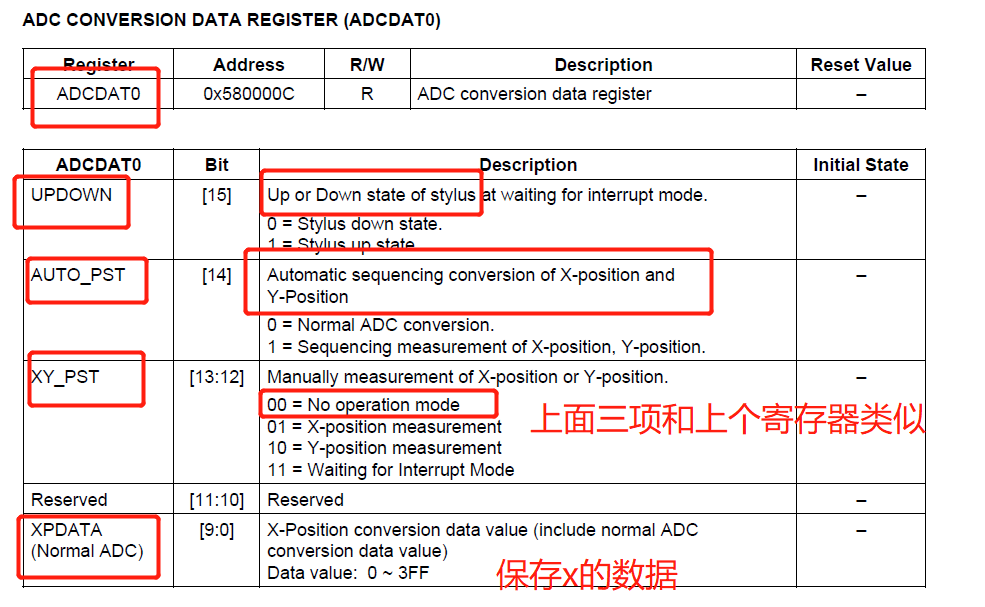


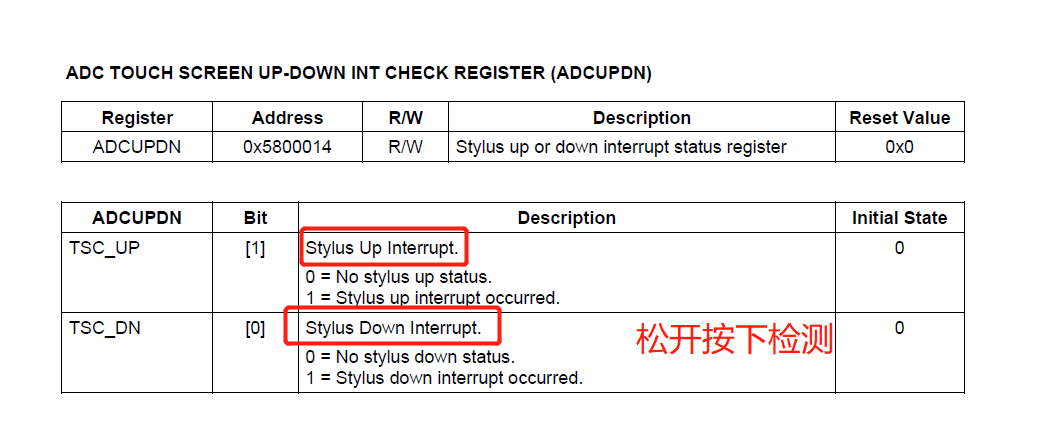
这里说的不是很对，中断模式是因为中断服务的返回时间和数据访问时间会有延时

同过查询模式比较快，通过最后bit可以得出adcdat的读取时间

### ts寄存器 芯片手册往下看，有一个前面没细说的寄存器

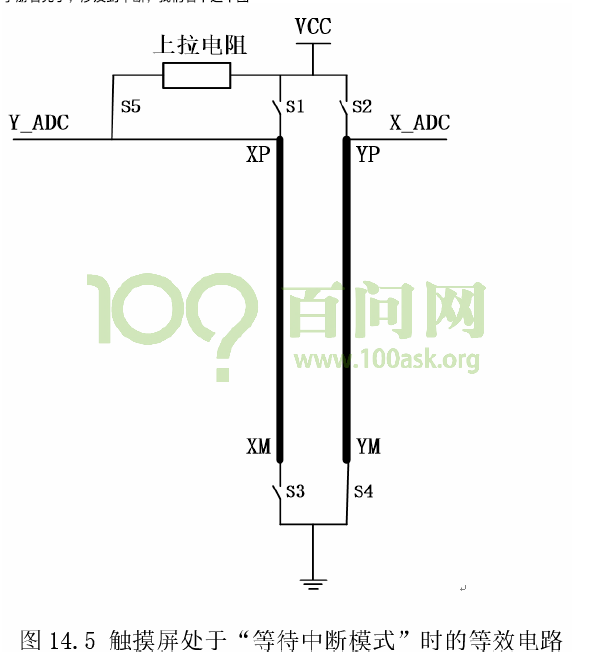






adc寄存器基本就这几个

下面看中断寄存器



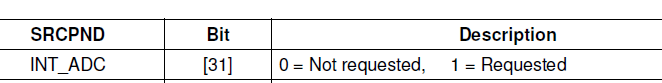
它会涉及两个中断，触摸笔按下、松开中断，adc启动，完成中断

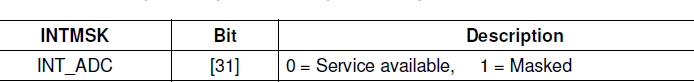
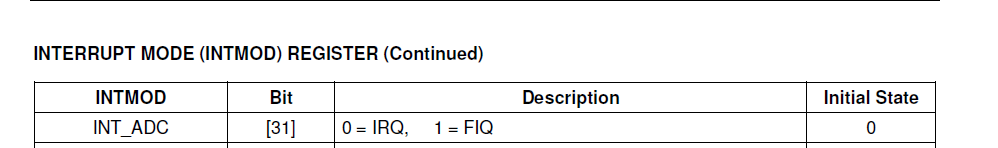
手册并没找到相应的寄存器，猜测可能是自动产生中断不用设置

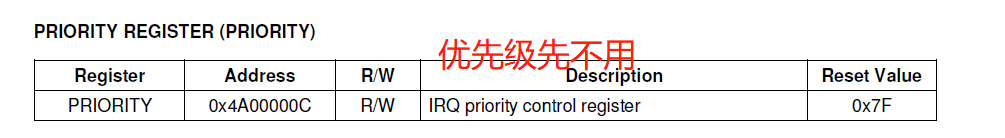
adc，触摸屏合用一个中断，可能有一个寄存器用来分辨具体是哪个中断产生

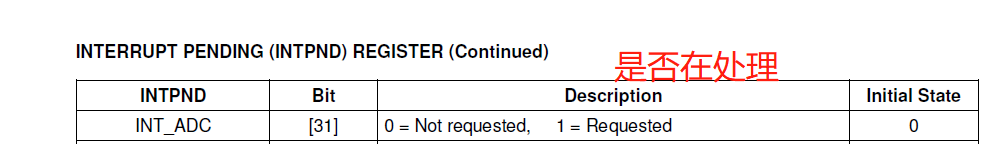
### interrupt寄存器

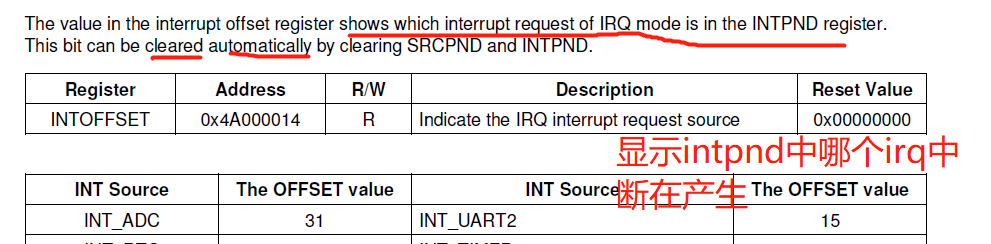
下面找到中断章节，一个一个看

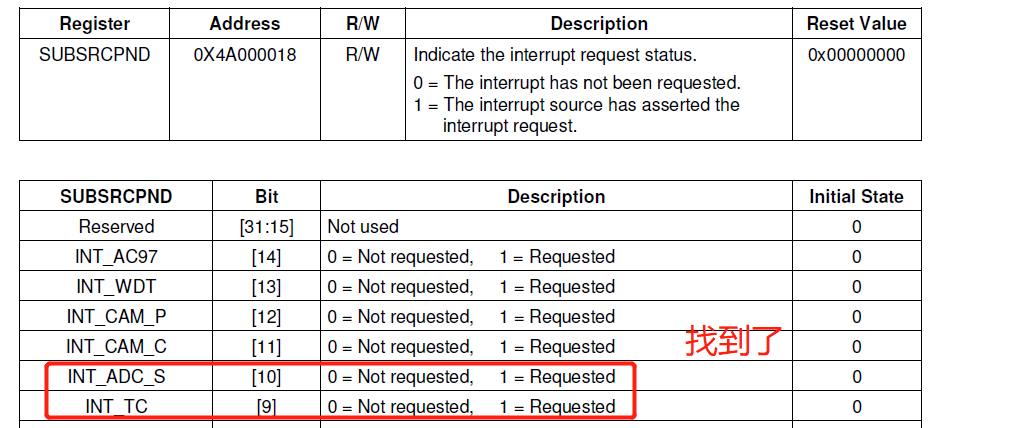


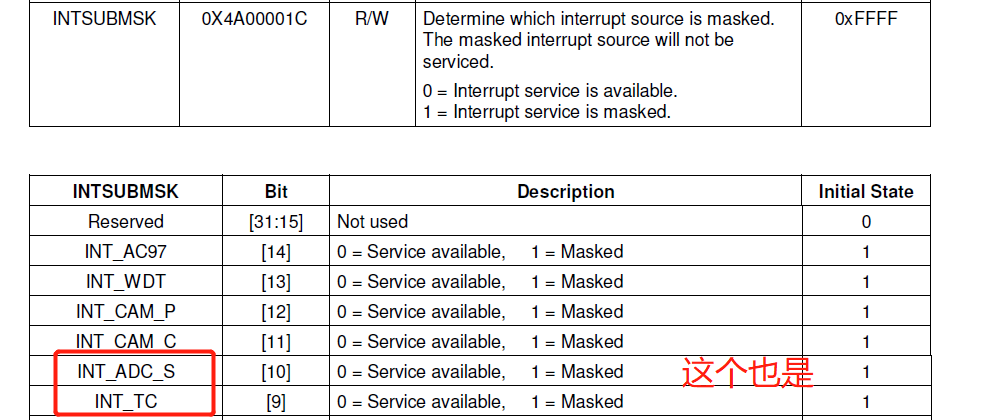












最后两个寄存器就是要找的

用intsubmsk来屏蔽中断

用subsrcpnd来查询哪个中断在发生

这两个寄存器会汇集一起到一个叫INT\_ADC的中断爱给CPU

### 程序框架:

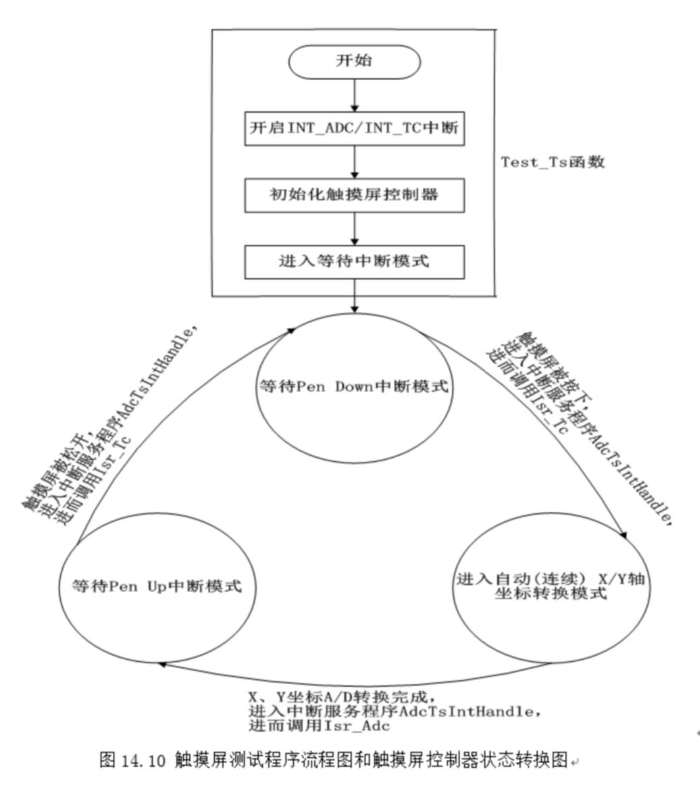
1. 初始化adc/touchscreen接口，adccon时钟接口
2. 一开始触摸屏是松开的，设置TS处于等待中断模式
3. 设置中断

intsubmsk使能acd和ts中断，

intmsk是ANT\_ADC发送给CPU

1. 按下触摸屏，进入ts中断
2. 自动xy左边转换
3. 启动adc
4. 转换完成后产生adc中断
5. 读数据
6. 再次进入等待中断模式
7. 启动定时器，处理长按或者滑动操作
8. 定时器中断
9. 判断是否松开，若松开则结束
10. 若重新按下则执行4.2启动adc

## 触摸屏编程\_按下松开检测



参考这个流程图

可以看到有个总中断服务程序AdcTsTntHandle，里面会分辨是adc中断还是ts中断

按下那么进入Isr\_Tc();

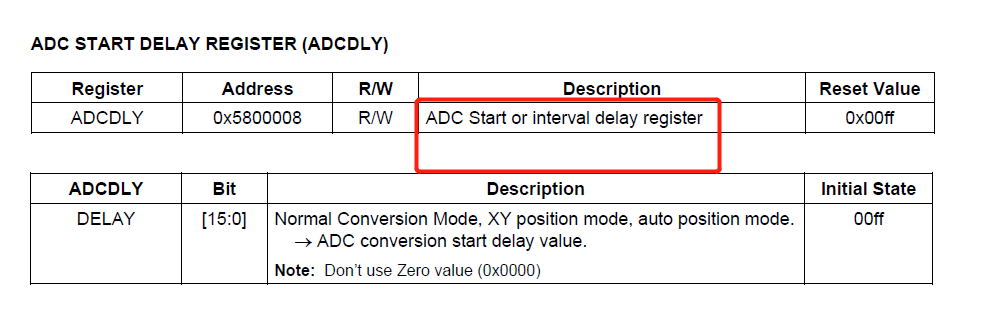
Isr\_Tc()判断是否按下，

如果是就进入测量模式enter\_auto\_measure\_mode()并启动adc，

测量完毕产生adc中断，回到AdcTsTntHandle，进入 Isr\_Adc();

发现输出不线性

添加delay函数



有一个之前被忽略的寄存器 延迟时间 = ADCDLY \* D(晶振周期) =ADCDLY \* 1 / 12000000 = 5ms

ADCDLY = 60000;

按下触摸笔，延时后才能产生中断

最后发现xy轴硬件接反了



后期解决

## 触摸屏编程\_定时器

前面只是单次按下才有坐标数据

用定时器解决长按滑动的问题

在按下器件启动定时器

定时器每过10ms就中断一次

在中断函数里测量触电的坐标

这样就可以得到连续的数据

首先打开timer.c

添加注册函数register\_timer

注销unregister\_timer

执行函数timer\_irq

顺便把之前的led点灯注册一下

还有原来定时器的设置改为10ms调用一次

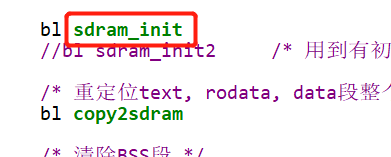
烧写到nand无输出,NOR有输出，是重定位前的代码超出了4K

移动重定位及之前所必须的代码放到最前面，确保在4k范围内

修改makefile

看start.S

copy2sdram函数有用到nand\_init



所以这两个都要放在前面，保证他们在4k前

之后都是在sdram执行的就不用管了

所以修改Makefile start.o nand\_flash.o 和 init.o放前面

将定时器放在屏幕按下后开启

touchscreen\_init注册touchscreen\_timer\_irq 每10ms启动一次

touchscreen\_timer\_irq检测新定义的ts\_timer\_enable标志位是否

根据ts\_timer\_enable来决定是否自动测量坐标

如果ts\_timer\_enable为零，直接返回不执行

如果松开，把ts\_timer\_enable设置零，等待笔按下，返回

如果按下，自动测量启动adc

其实这个函数做的事情跟tc\_irq是相似的

就是10ms一次的频率执行ts中断

重新屡一下流程

初始化，注册定时器中断touchscreen\_timer\_irq

等待笔按下

如果这时候按下了进入总中断服务器，判断是ts还是adc中断

ts中断会启动自动测量 ，启动adc

完后产生adc中断

adc中断判断是否还是按下

不是就ts\_timer\_enable为0，

是就打印坐标，并ts\_timer\_enable为1，以供每10ms启动一次的touchscreen\_timer\_irq去判断是否继续测量

编译烧写

main函数记得加timer\_init

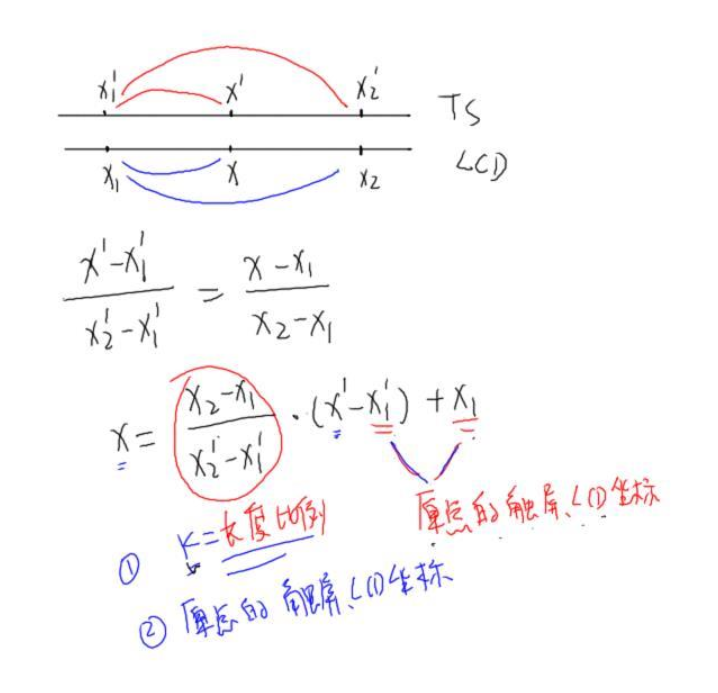
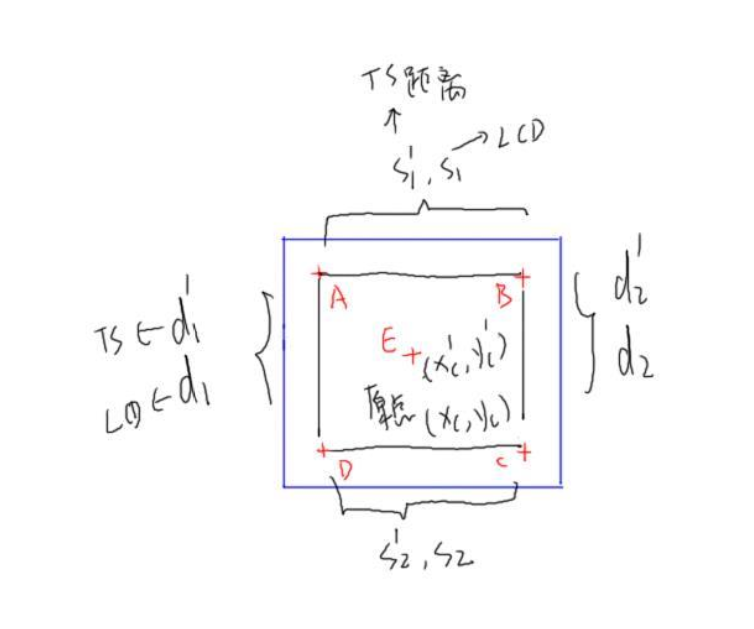
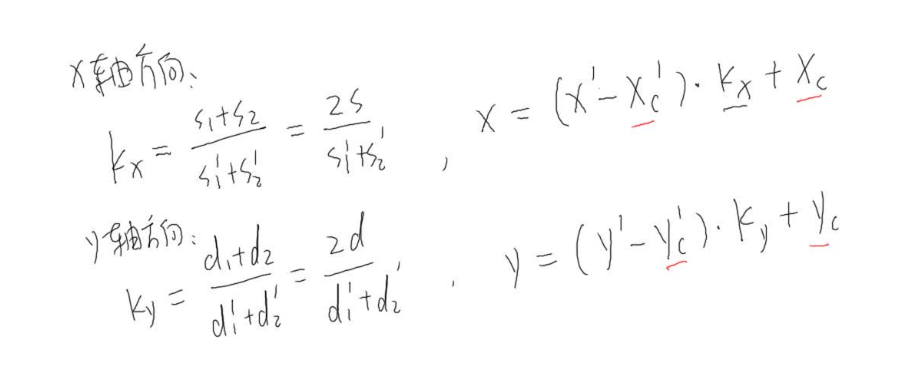
不然定时器无法工作

## 触摸屏编程\_屏幕校准

所谓校准就是找一个公式把电压值转换为坐标值

上面在adc中断时打印了触点的x，y

怎么转换为lcd的x，y

程序流程

在A点显示+

点击+

记录触摸屏的坐标

在bcde上循环操作，显示点击读取操作

得到的数据，确定公式

以后得到ts触点是，课转换出lcd坐标

构造几个函数

画+ fb\_disp\_cross(int x,int y) 在geometry.c

记录ts坐标 ts\_read\_raw

公式 ts\_calibrate

转换lcd坐标 ts\_read

touchscreen.c将原来的打印printf去掉，换上report\_ts\_xy函数 打印 x’ ， y’ ，

和 ts\_read\_raw函数 用来获取上图公式里的x’ ， y’

新建tslib.c校准函数极其相关

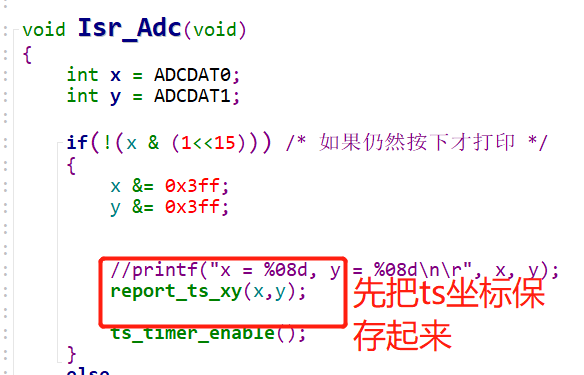
在framebuff里新建清屏函数

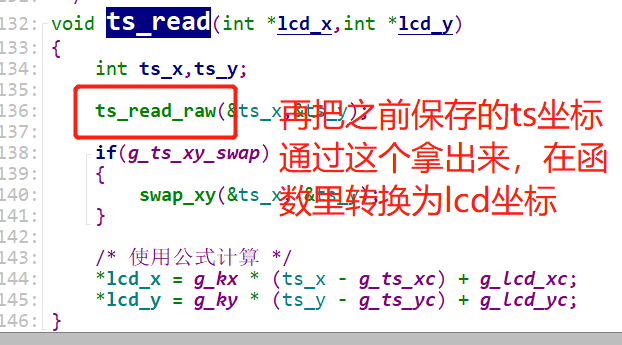
最后在touchscreen\_test里面，先一个一个显示，然后提醒开始校准，校准完毕提示

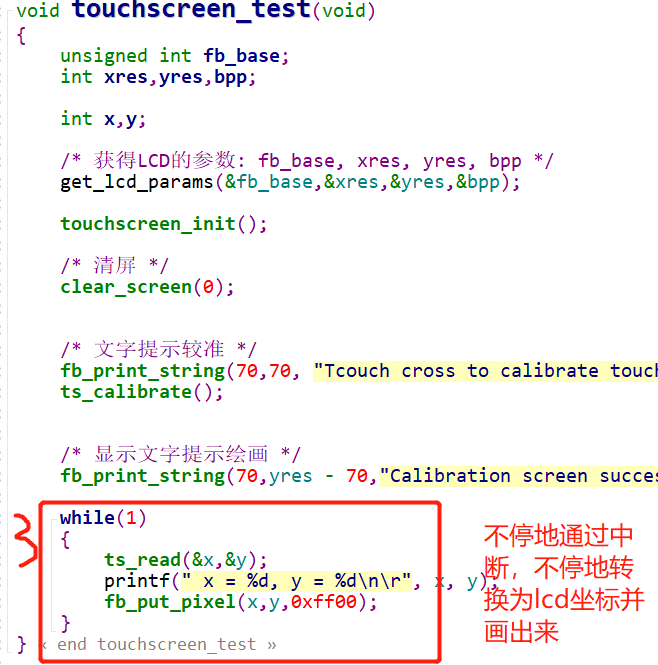
屡一下流程，



看一下adc中断发生了什么，数据怎么来的







### debug和完善

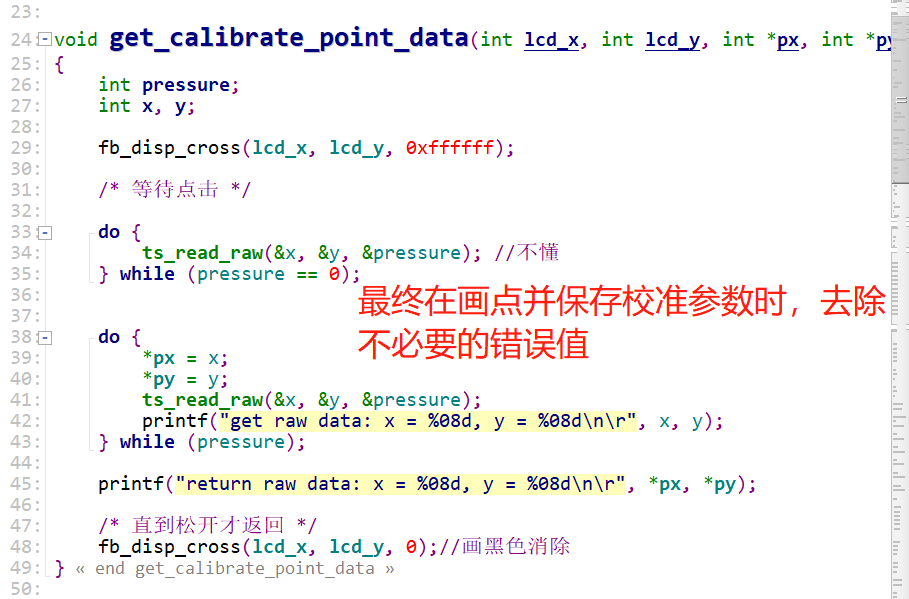
report\_ts\_xy

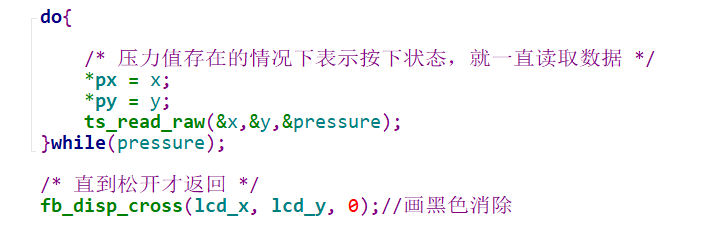
ts\_read\_raw

中新添形参 压力值pressure

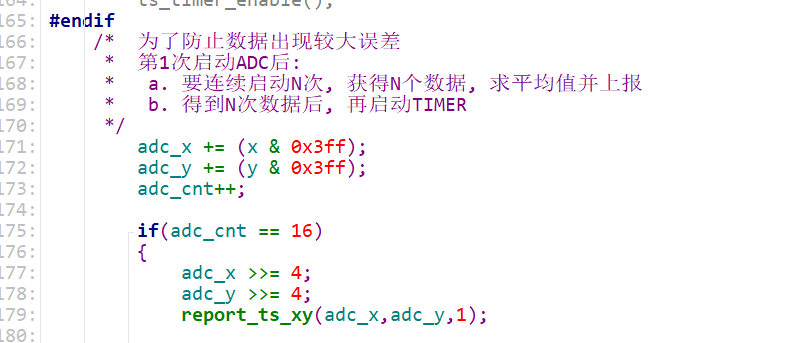
在ts中断的时候report\_ts\_xy（0，0，0）

在adc中断的时候report\_ts\_xy（x,y,1）

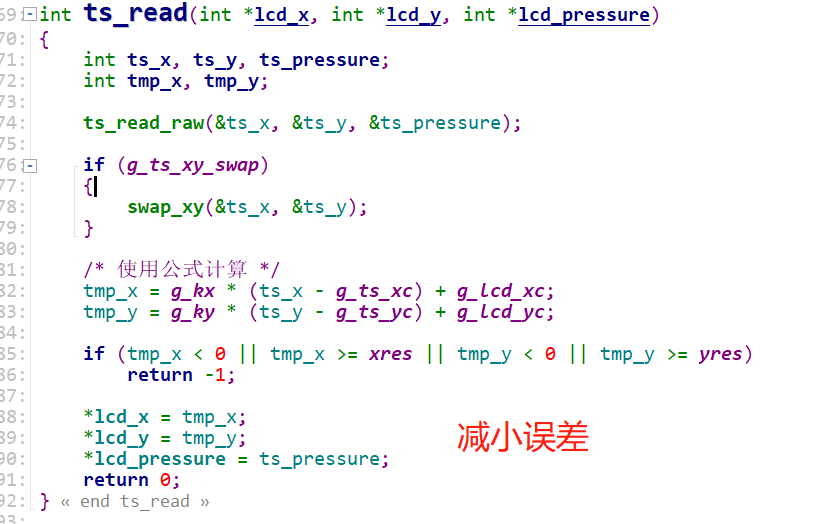


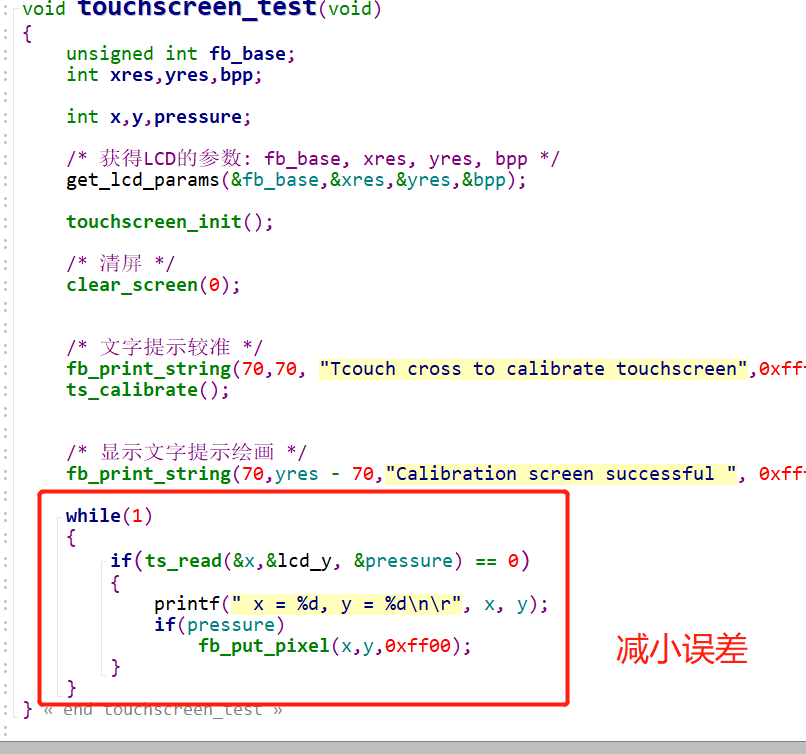


adc中断改进

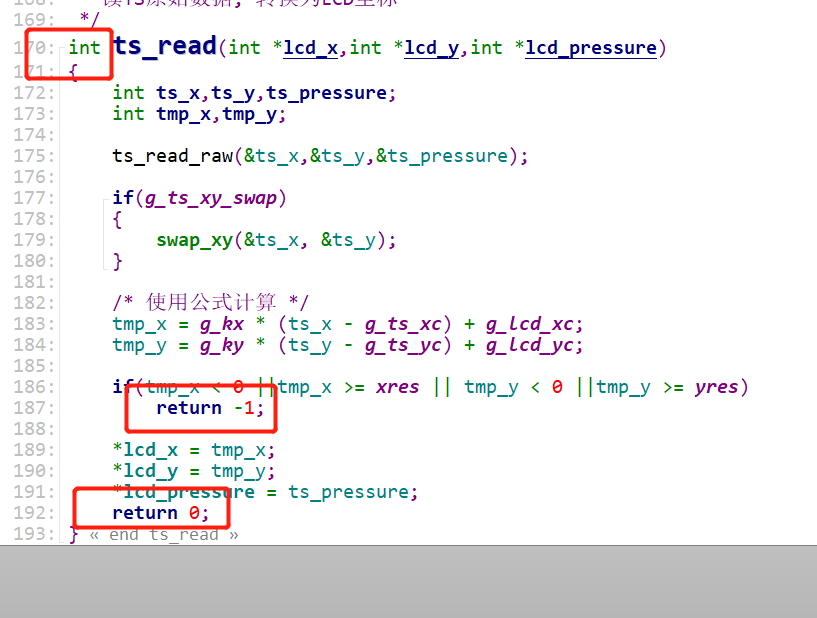


为了确保数据准确性





编译烧写



之前int写成了void，导致校准完后无法继续操作

debug后屏幕画写操作依旧无法实现

存在很多误差

继续完善

## 末章

