# 问题分析及解决方案

## 1、CPU不行

这个不能解决，呵呵

## 2、QT问题(消息，合成)

触摸屏EVENT处理占用CPU过多：

1. 接收INPUT数据时，使用QSocketNotifier来导致产生很多event。

解决：取消QSocketNotifier，直接监听

1. 触摸屏event过多。触摸屏event会同时分发给QTUI和GPU合成模块

* GPU合成模块：实时刷新鼠标绘制
* QTUI：分发给界面UI对象

过多的event导致触摸过程中CPU飙升，解决方法：

* 没有接鼠标时，不发送event给GPU合成模块(需要InputMgr配合)
* 对event重采样，减少发给QTUI的event

1. 滚动时界面快速变化，消耗过多CPU

QT原因，滚动就是图像拷贝。解决方法：

* 滚动时使用GPU协助拷贝，降低CPU负担

QT原因，从QWidget派生的对象无法准确判断脏区域，解决方法：

* 从QLabel派生(原因正在分析)

应用原因。应用错误的在滚动区叠加图层，导致QT无法判断准确判断脏区域，滚动时，需要更新整个滚动视口，无法使用QT的滚动优化机制，解决方法：

* 更正及优化应用

## 3、驱动问题

InputMgr占用过多CPU，当操作触摸屏时，CPU飙升到7%~8%。解决方法：

* 禁止周期性LOG(写文件)，CPU降到2%~3%
* 优化InputMgr(还未实施：作为一个低速设备2%~3%的CPU还是高)

## 4、应用问题

### 4.1、QImage赋值操作问题

QT帮助中，说明QImage赋值操作是浅拷贝(和QString一样)，所以我们应用中转移QImage都是通过=号处理。但是如果当前有QPainter正在操作QImage，那么=号的实现就是深拷贝。QImage怎么判断是否有QPainter在操作呢？就是QImage被指派给QPainter对象，QPainter没有析构，或者没有调end方法。

### 4.2、图层叠加问题

图层叠加问题。我们系统设计中存在很多无必要图片叠加，比如先画一张图，然后再把图转绘叠加到另外一张图上，这样做从某种意义上说有利于对象封装或者说编程，但产生过多的拷贝。推荐的更改方法是：

QImage image;

QRect rect1;//方法1操作区域

QRect rect2;//方法2操作区域

Draw1(image, rect1); //绘制操作1

Draw2(image, rect2); //绘制操作2

void Draw1(QImage &image, QRect &rect)

{

//绘制限制在rect里

}

void Draw2(QImage &image, QRect &rect)

{

//绘制限制在rect里

}

### 4.3、event处理优化问题

当用户连续快速操作时，因为UI的反应慢，导致event堆积，如果应用不对堆积的event做一个处理，就会导致CPU长时间在100%以上，降低操作体验。解决方案：

* 降低event处理周期
* 智能识别event堆积，合并处理一些event

### 4.4、滚动视口叠加导致每次滚动都重新绘制

拿事件回顾说，目前实现是，在滚动区域叠加了3个透明的图层实现的：

波形

标尺

卡尺

这样的实现将导致滚动每次重新绘制整个视口。解决方案：



* 标尺画在波形上
* 像卡尺这种应用，没法画到波形上，解决方法是触摸时把卡尺层掩藏

## 5、请求及处理数据较慢，导致UI长时间冻结

这部分的解决方案是增加后台异步处理，当界面移到某个位置时，判断是否准备好数据(图像)，如果没准备好，向后台发出准备数据命令，后台准备好数据后向QTUI发送更新通知。

这一块，应该有一个统一的架构去实现。

## 6、QT绘制性能

关于QT绘制性能，实验了不使用QT，直接操作QImage位图缓存，的确会比QT性能高。40道波形32 秒的情况下，可以节约0.9秒左右。

但如果我们缩小图片，这个提升就不会很明显。