# Kmbox-Net版开发文档

为什么开发？

1. B板串口通信速度慢。调用速度太慢。且通信过快回导致盒子重启掉效果。
2. 串口容易被检测和扫描，且原来公开协议，导致B板单机被企鹅针对。
3. 很多AI和DMA不需要板载宏功能。只需要简单的调用。

Kmbox-Net具有以下特性：

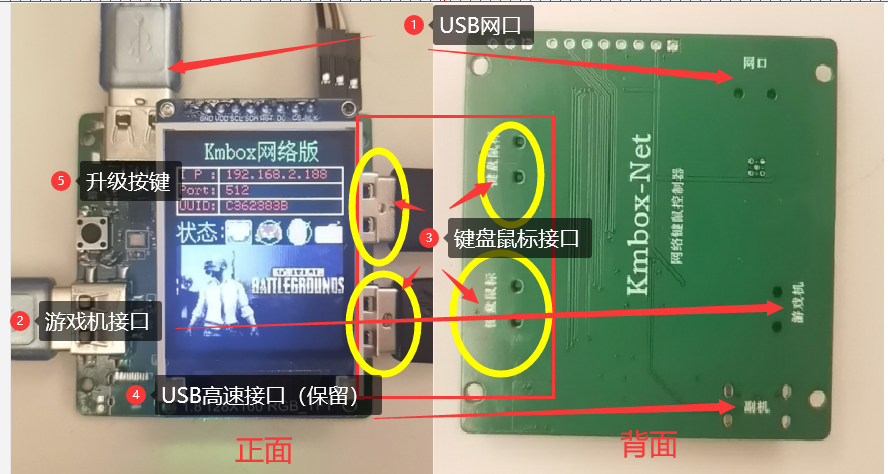
1. 安全，协议不公开，无法被特征。不用装驱动。每个盒子独立IP、端口、硬件编码。

采用阻塞socket通信。无法被扫描。

1. 稳定性强，不会像串口通信那样用着用着就白屏重启。
2. 速度快。100M网络，通信速度是原来B板的100倍（相比115200波特率）。每秒调用 次数接近1000次。原来B板只右每秒300次调用。调用太快会白屏重启。
3. 自动默认人工轨迹，不会出现键鼠数据异常，请参考kmNet\_mouse\_move\_auto函数。
4. 无需适配鼠标。通用性强。
5. Device端支持修改所有USB参数。支持网络更新。每个人都有不同特征值。真正的一人 一固件模式。(TBD)
6. 友好UI模式。哪里出问题一目了然。傻瓜式操作。
7. 支持B板除板载脚本外所有功能。
8. 支持监控物理键鼠。屏蔽物理键鼠功能。方便写软件。
9. 支持物理键鼠学习复制功能。（TBD）
10. 支持kvm键鼠切换功能。（TBD）

# 硬件篇

Kmbox-Net包含4个USB口。（图片只是样品一切以实际发布为准）。



如图所示，kmbox-Net包含以下物理接口：

1. USB网口一个。接上电脑后会有一个USB网卡。用于跟盒子通信，传输控制指令。
2. USB游戏机接口一个。接上电脑后会枚举为标准键盘鼠标。用于控制游戏电脑键盘鼠标。
3. USB键鼠接口两个。用于连接键盘或者鼠标。用来控制游戏电脑。
4. USB高速接口（typeC口）一个。可用于双机同步等操作。（默认不焊接）
5. 升级按钮一个。用于固件更新。

# 如何接线

用蓝色USB线连上盒子的游戏机口到电脑即可给盒子上电。此时盒子显示屏如下。

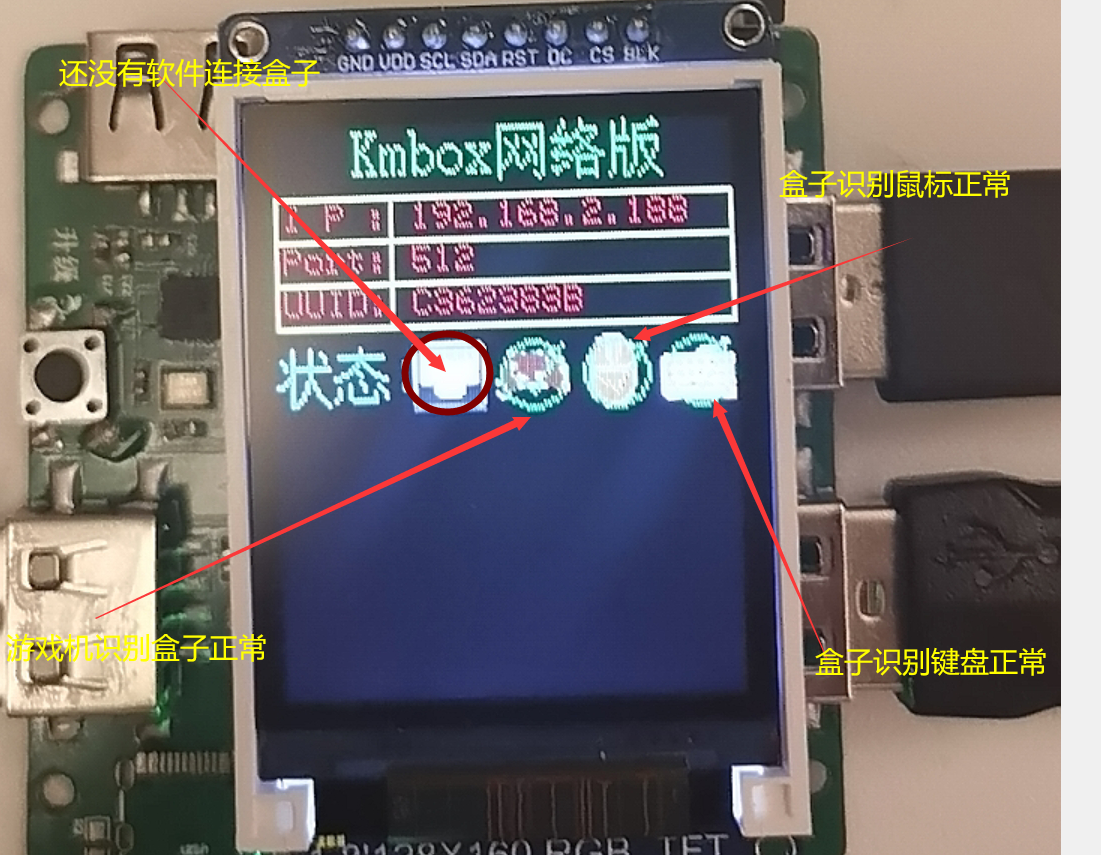


状态栏第二个图标是一个手柄。表示游戏机。仔细看会发现第二个图标上电后先打叉最后打钩。此图标的意义如下：

|  |  |  |
| --- | --- | --- |
| 等待连接 | 游戏机识别到盒子 | 游戏机断开与盒子的连接 |
| 游戏机、 |  |  |
| 一般上电枚举过程出现。 | 正常识别盒子键鼠设备出现 | 主机休眠或者断开连接时出现 |

Ps:如果出现打叉情况，请重新拔插游戏机USB线缆。

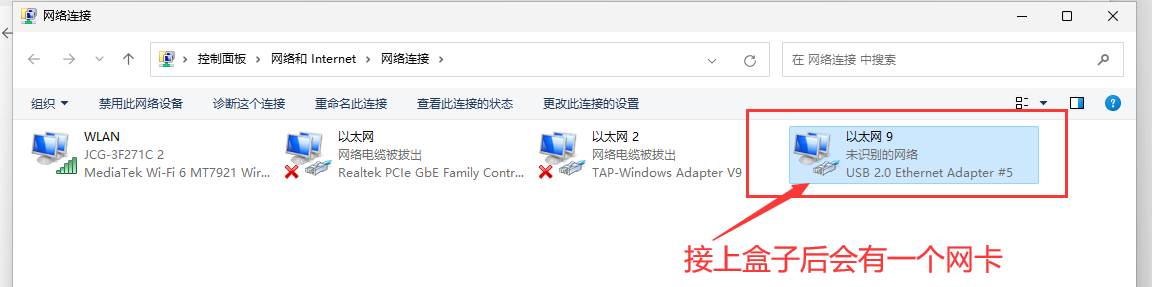
将打游戏的键盘鼠标接到盒子上。如果识别正常。盒子显示屏显示如下：



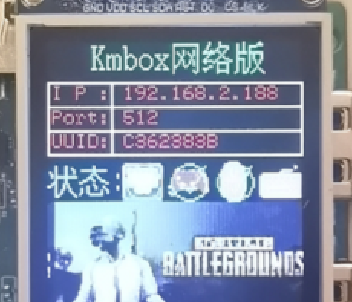
如果接入的键盘鼠标能正常控制游戏机，那么盒子就连接就OK了。

# 软件篇

软件可以通USB网卡连接盒子并控制所有键鼠数据。屏蔽真实物理按键。检测物理按键是否按下。移动鼠标。点击键盘等操作。盒子提供API供您完全掌控物理键鼠。在使用软件前需线将网口USB线连接到电脑。



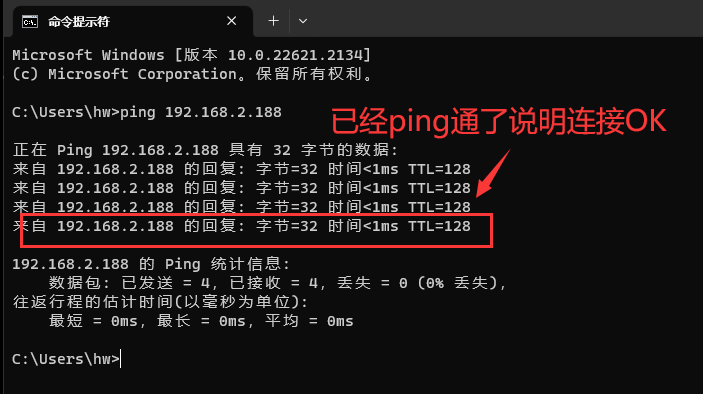
首次接入盒子会自动弹出一个CD-ROM设备。打开双击后即可出现USB网卡。出现网卡后我们需要配置一下网卡，使网卡的IP与盒子IP在同一网段。这样电脑才能跟盒子通信成功。如下图所示，盒子IP是192.168.2.188（PS每个盒子IP不一样。一切以显示屏上的IP为准）



注意：网口USB线一定是插开挂的电脑。因为外挂是通过网口与盒子通信的。



修改完主机IP后。我们可以ping一下盒子。用来验证主机与盒子网络是否联通：



接下来就可以开始软件调用了。

# 连接盒子 kmNet\_init

首先必须调用连接盒子：int kmNet\_init(char\* ip, char\* port, char\* uuid);//ok



第一个参数ip :盒子的IP地址。显示屏上有。

第二个参数port :盒子的端口号。显示屏上有。

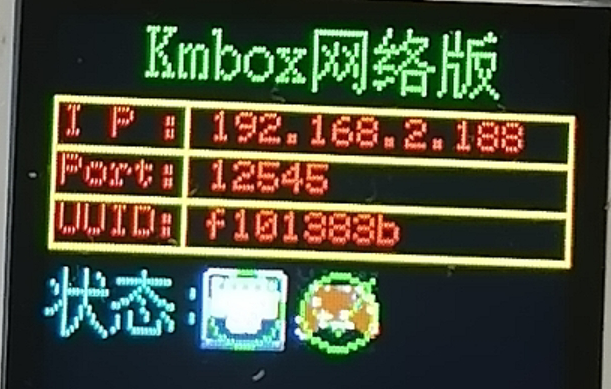
第三个参数UUID :硬件唯一标识码。显示屏上有。

以上三个参数默认出厂的盒子，每个设备都不一样。一切以显示屏上显示的为准。由于每个设备都不一样，因此源头上杜绝被批量针对问题。

状态栏第一个图标是网络连接状态。详见下表：

|  |  |  |
| --- | --- | --- |
| 没接网线时 | 等待被连接 | 连接盒子成功后 |
|  | 网口 |  |
| 没接网线时是网口打叉图标 | 接入网线后打叉消失。此时表示盒子正在等待被连接 | 盒子已经被上位机（辅助软件）连接。调用kmNet\_init后此图标出现。 |

下图是调用kmNet\_init成功连接盒子后的显示：



# 鼠标相对移动kmNet\_mouse\_move

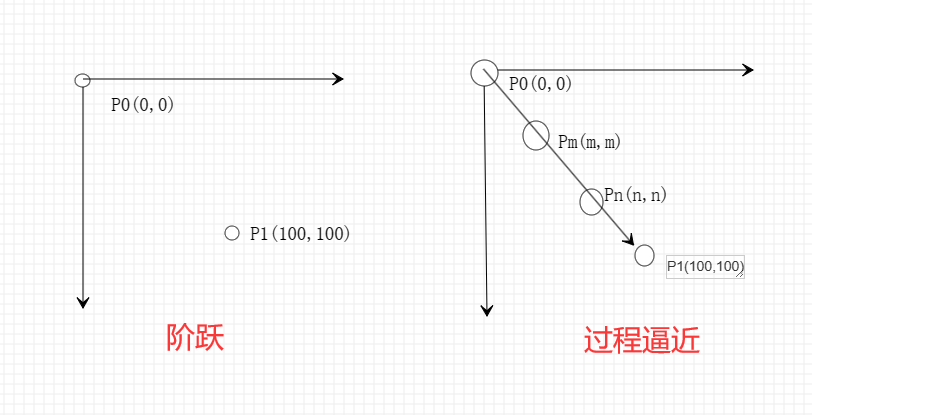
int kmNet\_mouse\_move(short x, short y);

此函数用来控制鼠标移动，耗时约为1ms。此移动是立即移动，不存在中间状态（跳变）。

如下图所示：

鼠标当前在坐标（0,0）点，调用kmNet\_mouse\_move（100,100）后。鼠标会直接移动到坐标（100,100）。

中间没有任何的过渡。这种移动很容易被游戏检测到键鼠异常。因为正常用物理鼠标操作，从（0,0）移动到（100,100）。会有很多过渡点（过程逼近）。正常鼠标每次移动的最小单位为1.理论上从P0（0,0）点移动到P1(100,100).至少要有100\*=144个过渡点。如果一次性从（0,0）阶跃到（100,100）那么可能就会被游戏认为鼠标数据异常。因为你省略了中间143个过渡点。kmNet\_mouse\_move函数用作最基本的移动。适合自己写移动曲线。





如图所示，调用10000次鼠标移动函数，一共耗时10188ms。相当于每次调用耗时约为1.0188ms。平均每秒可调用919次。PS:原来A,B板速度约为每秒300次。

样板速度为每秒1000次。

# 鼠标模拟人工移动 kmNet\_mouse\_move\_auto

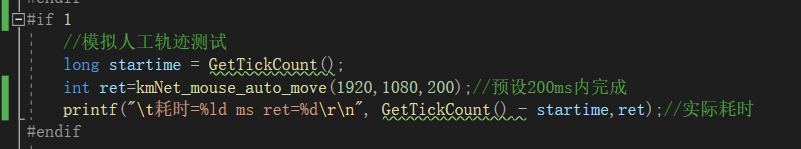
int kmNet\_mouse\_move\_auto(int x, int y,int time\_ms);//ok

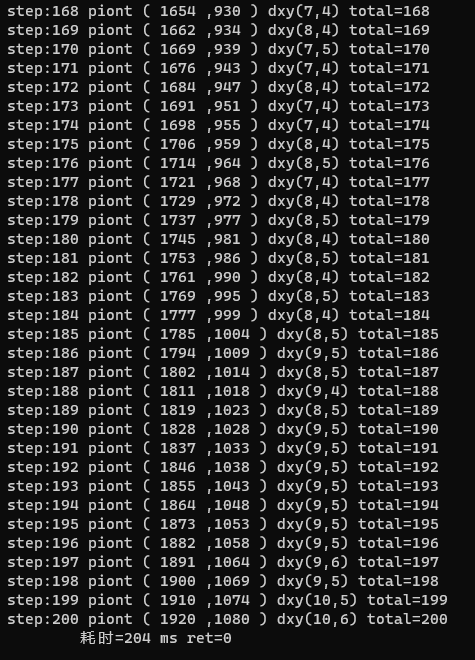
与kmNet\_mouse\_move的不同之处在于第三个参数。kmNet\_mouse\_move是阶跃的一步到位，速度最快，而 kmNet\_mouse\_move\_auto是自动带中间过程的移动。适用于自动模拟人工轨迹。第三个参数是指定此次移动在多少个毫秒内完成。例如：

kmNet\_mouse\_auto\_move(1920,1080,200);//预设200ms内完成

是移动到（1920,1080）点，要求在200ms内完成。盒子收到此指令后会自动填充中间人工移动轨迹。

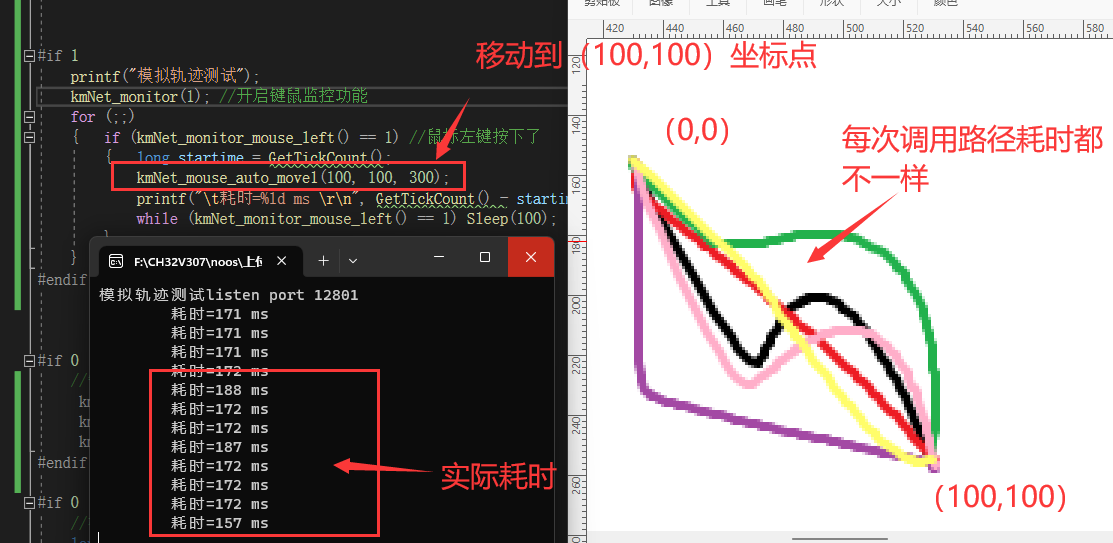
在200ms内移动到（1920,1080）的坐标点。





如上图所示。在给定200ms的参数后，盒子经过204ms后将（0,0）到（1920,1080）的中间所有轨迹模拟出来了。此函数适用于没有轨迹模拟的软件调用。盒子可以自动帮助生成中间状态的轨迹。另外第三个参数很重要，一切请以实际人工操作耗时为参考。例如：

kmNet\_mouse\_auto\_move(1920,1080,1);第三个参数给1ms.这一定会有问题。没人能1ms将鼠标指针移动（1920,1080）。第三个参数尽量以人工操作耗时为参考。参数异常导致盒子被检测。不要说我硬件问题谢谢。（为了防止键鼠异常，做硬件的真是操碎了心啊！）。并且每次的移动路径都是随机不一样的。详细可参考demo调用代码



另外后续会加入更多高级算法。鼠标合速度，分速度，加速度，一阶导，二阶导，N阶导，方向向量等。杜绝鼠标数据异常问题。

# 鼠标按键控制

鼠标按键控制包括以下几个函数：

int kmNet\_mouse\_left(int isdown);//左键控制ok

int kmNet\_mouse\_right(int isdown);//右键控制ok

int kmNet\_mouse\_middle(int isdown);//中键控制ok

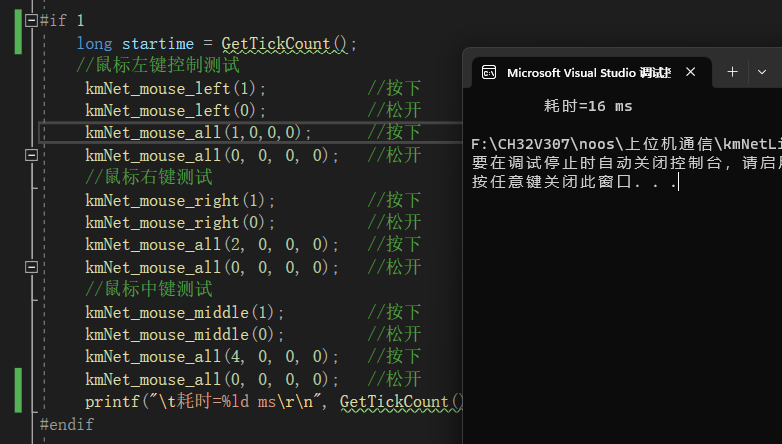
int kmNet\_mouse\_wheel(int wheel);//滚轮控制ok

int kmNet\_mouse\_all(int button, int x, int y, int wheel);//鼠标全部数据一次性控制ok

Isdown=0时表示抬起，1时表示按下。

Wheel为正时表示下滑滚轮，负时表示上滑滚轮。

Button是鼠标按键，x,y是坐标，wheel是滚轮。



# 键盘控制类函数

键盘控制类函数主要有以下几个：

int kmNet\_keydown(int vkey);// ok

int kmNet\_keyup(int vkey); // ok

其中vkey是按键的HID键码表，键盘按键按下调用kmNet\_keydown函数，松开调用kmNet\_keyup函数即可。

# 物理键鼠状态获取

当需要知道盒子上的键盘或者鼠标按键是否按下时可以使用此类函数。这些函数是直接读取盒子硬件。不会调用系统的API，也能在一定程度上防止hook检测。

//监控系列

int kmNet\_monitor(short enable); //打开关闭物理键鼠状态监测

int kmNet\_monitor\_mouse\_left(); //鼠标左键是否按下

int kmNet\_monitor\_mouse\_middle();//鼠标中键是否按下

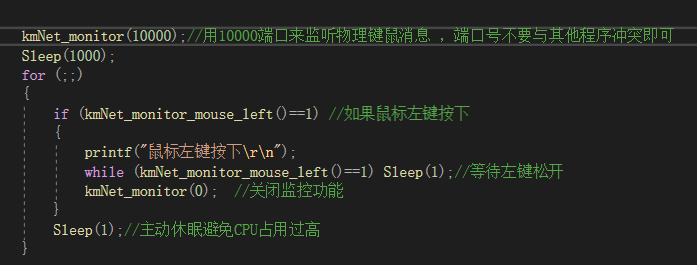
int kmNet\_monitor\_mouse\_right();//鼠标右键是否按下

int kmNet\_monitor\_mouse\_side1();//鼠标侧键1是否按下

int kmNet\_monitor\_mouse\_side2();//鼠标侧键2是否按下

int kmNet\_monitor\_keyboard(int vk\_key);//键盘指定按键vk\_key是否按下。

注意，以上函数均为实时状态，即调用时刻检测当前鼠标状态，不会阻塞。不影响与上位机发送其他指令。因为传输指令和接收监控信息是在不同的端口进行。调用监控类函数前请先调用一遍kmNet\_monitor（port），使能监控。port为socket的端口号。注意Port不要与本机其他应用程序端口冲突。Port=0时为关闭监听。



# 物理键鼠屏蔽

//物理键鼠屏蔽系列

int kmNet\_mask\_mouse\_left(int enable); //屏蔽鼠标左键

int kmNet\_mask\_mouse\_right(int enable); //屏蔽鼠标右键

int kmNet\_mask\_mouse\_middle(int enable);//屏蔽鼠标中键

int kmNet\_mask\_mouse\_side1(int enable); //屏蔽鼠标侧键键1

int kmNet\_mask\_mouse\_side2(int enable); //屏蔽鼠标侧键键2

int kmNet\_mask\_mouse\_x(int enable); //屏蔽鼠标X轴坐标

int kmNet\_mask\_mouse\_y(int enable); //屏蔽鼠标y轴坐标

int kmNet\_mask\_mouse\_wheel(int enable); //屏蔽鼠标滚轮

int kmNet\_mask\_keyboard(short vkey); //屏蔽键盘指定按键

int kmNet\_unmask\_all(); //解除屏蔽所有已经设置的物理屏蔽

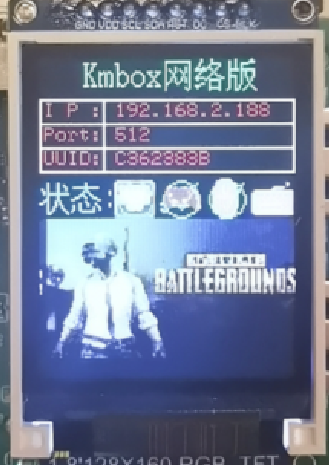
# 显示屏控制函数

int kmNet\_lcd\_color(unsigned short rgb565);//将整个LCD屏幕用指定颜色填充。

注意参数是rgb565格式。可以填充黑色来实现请清屏的目的。

int kmNet\_lcd\_picture\_bottom(unsigned char\* buff\_128\_80); //下半部分显示128x80图片

注意buff\_128\_80是分辨率为128x80的图片取模数组。可以参考取模软件设置。

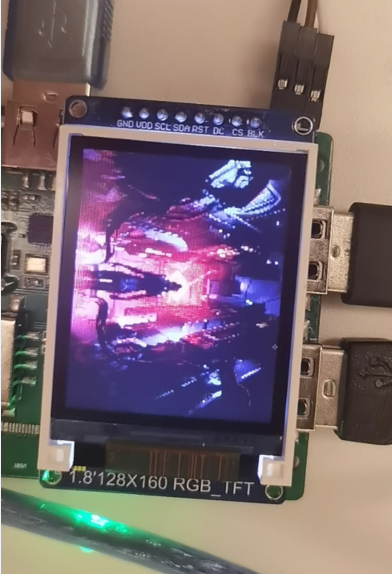


注意取模数组大小，一定是128x80x2字节。

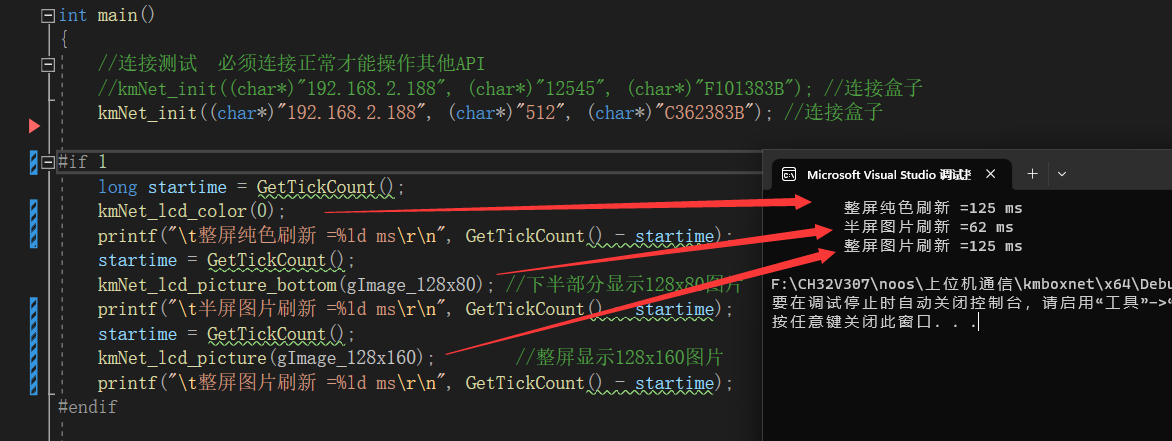
int kmNet\_lcd\_picture(unsigned char\* buff\_128\_160); //整屏显示128x160图片

此函数用于刷新整个屏幕。图片大小固定为128x160。





下面是测试代码耗时情况。全屏耗时125ms.半屏62ms



# 关于升级

盒子支持固件升级功能。升级工具源码见doc文件夹下升级工具中。您可以自己修改升级工具。以官方demo为例。后续如何升级盒子固件。