# Kmbox-Net version development documentation

Why is it developed?

1. The serial port communication speed of board B is slow. The call speed is too slow. And the communication too fast causes the box to restart the effect.
2. The serial port is easy to be detected and scanned, and the original public protocol led to the B board being targeted by the penguin.
3. Many AI and DMAs do not require onboard macros. Just need a simple call.

The Kmbox-Net has the following properties:

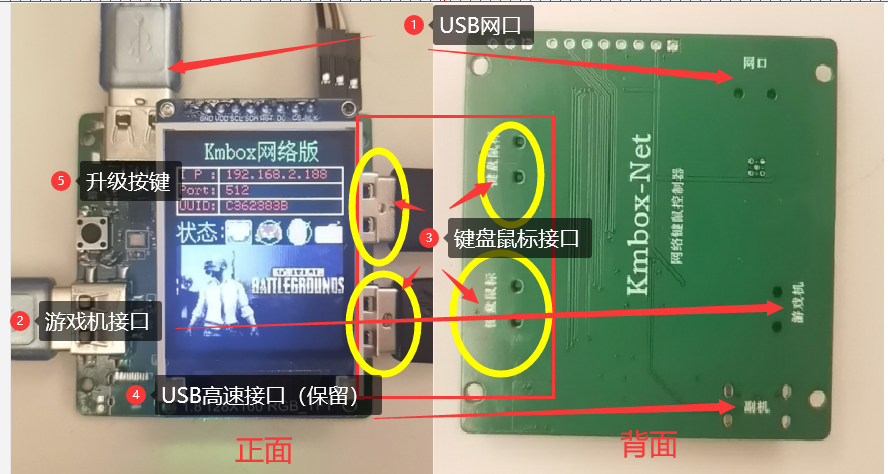
1. Security, the agreement is not open, and cannot be characterized. No drive. Each box is independent IP, port, hardware coding.

A blocked socket communication was used. Unable to be scanned.

1. Strong stability, not like serial communication with the white screen restart.
2. fast speed. For the 100M network, the communication speed is 100 times higher than the original B board (compared to 115,200 baud rate). Call per second Nearly 1,000 times. Board B is only 300 calls per second. A call made too quickly causes a white screen to restart.
3. Automatic default manual track, no key and mouse data exception, please refer to kmNet \_ mouse \_ move \_ auto function.
4. No mouse required. Strong versatility.
5. The Device end supports the modification of all USB parameters. Support for network updates. Each individual has different feature values. A real person A firmware mode.(TBD)
6. Friendly UI mode. What went wrong is clear. Silly operation.
7. Support all functions of board B except the onboard script.
8. Support monitoring physical mice. Shield the physical key and mouse functions. Easy to write software.
9. Support the physical key and mouse learning replication function.（TBD）
10. Support the kvm key and mouse switch function.（TBD）

# Hardware

Kmbox-Net contains four USB ports.(Pictures are just samples and all that are subject to the actual release).



As shown, the kmbox-Net contains the following physical interfaces:

1. A USB network port. Connect the computer will have a USB network card. For communicating with the box and transmitting control instructions.
2. The USB game console interface is one. The computer is enumerated as a standard keyboard and mouse. Used to control the game computer keyboard and mouse.
3. Two USB bond and mouse interfaces. Use to connect a keyboard or mouse. Used to control the game computers.
4. USB high-speed interface (typeC port) one. Can be used for dual-machine synchronization and other operations.(No welding by default)
5. Upgrade the button is one. For the firmware updates.

# How to connect

Connect the blue USB cable to the game port of the box to the computer to power the box. The box display screen is as follows.

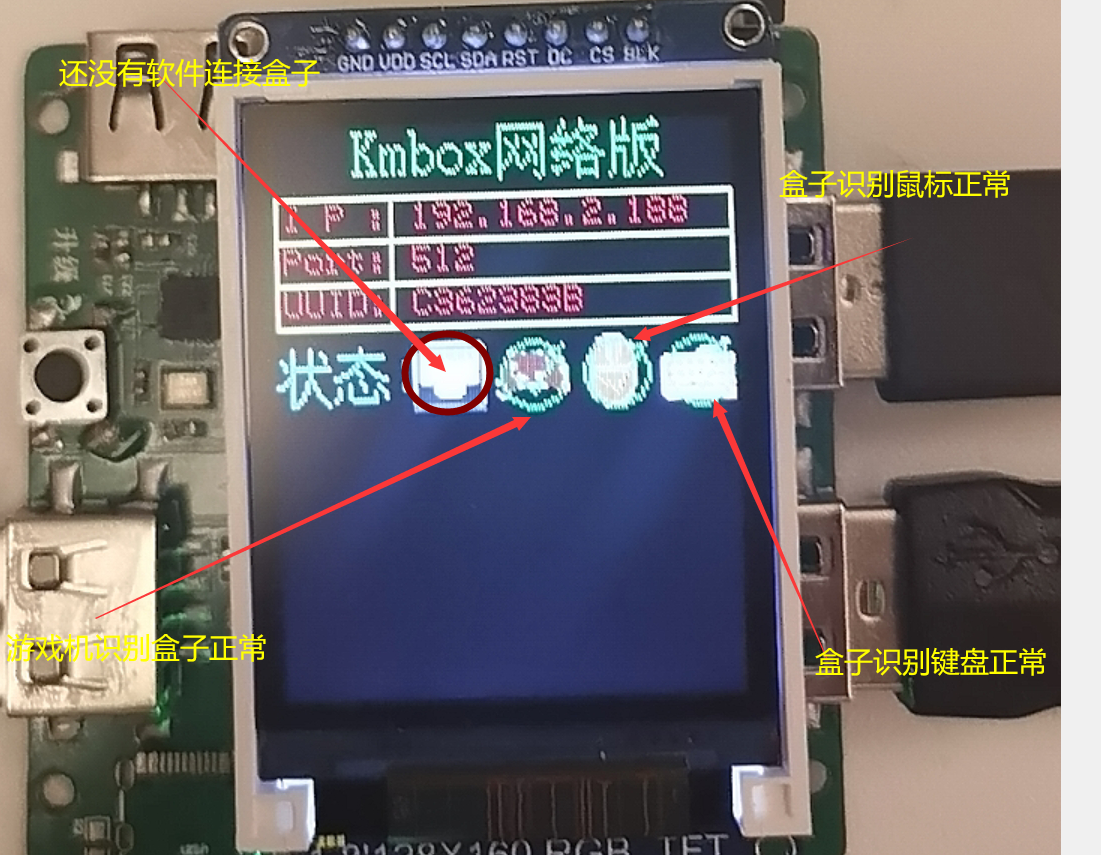


The second icon in the status bar is a handle. Indicates the game console. If you look carefully, you will fork the second icon and then hook it. The significance of this icon is as follows:

|  |  |  |
| --- | --- | --- |
| queued for connection | The game console is identified to the box | The game console is disconnected from the box |
| 游戏机、 |  |  |
| Generally, the power-on enumeration process occurs. | Normal identification of the box mouse device appears | Appear when the host is dormant or disconnected |

Ps: If a fork occurs, restart the USB cable.

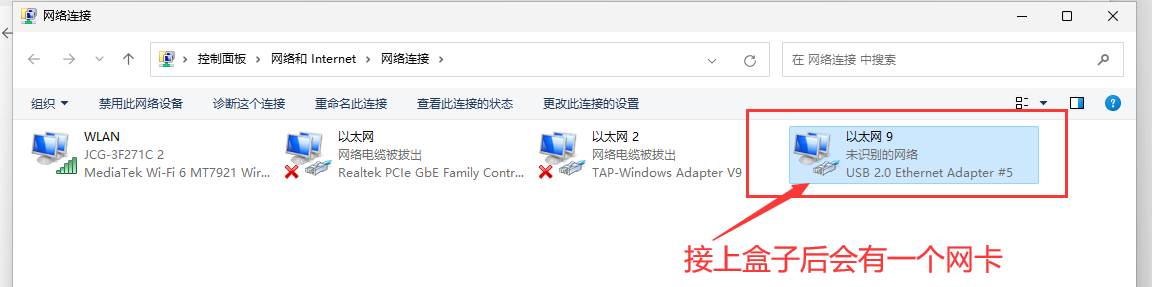
Connect the keyboard and mouse of the game to the box. If the identification is normal. The box display is as follows:



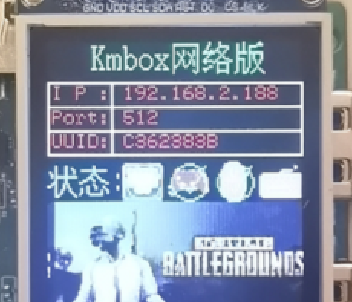
If the keyboard and mouse can control the game, the box will be OK.

# Software article

The software can connect the box to the USB network card and control all the key and mouse data. Block the real physical buttons. Check whether the physical button is pressed.rolling mouse. Click on the keyboard and so on. The box provides an API for you to take complete control of the physical key and mouse. Connect the network port USB cable to the computer before using the software.



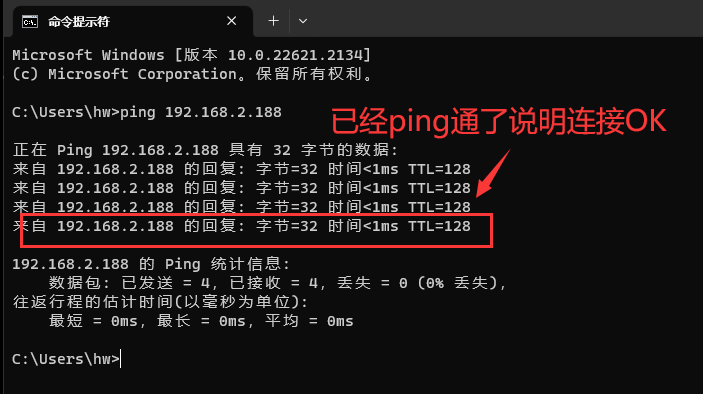
A CD-ROM device is automatically pop-up. Double click the USB network card. After the network card appears, we need to configure the network card, so that the IP of the network card and the box IP are in the same network segment. So the computer can communicate with the box. As shown in the figure below, the box IP is 192.168.2.188 (PS each box IP is different. IP on the display)



Note: the network port USB cable must be plugged in the computer. Because the plug-in communicates with the box through the network port.



After modifying the host IP. We can ping the box. Used to verify whether the host machine is connected to the box network:



Then you can start the software call.

# Connect the box to kmNet \_ init

First the connection box must be called: int kmNet \_ init (char \* ip, char \* port, char \* uuid); / / ok



The first parameter, ip : The IP address of the box. There are on the display.

The second parameter, the port : Port number of the box. There are on the display.

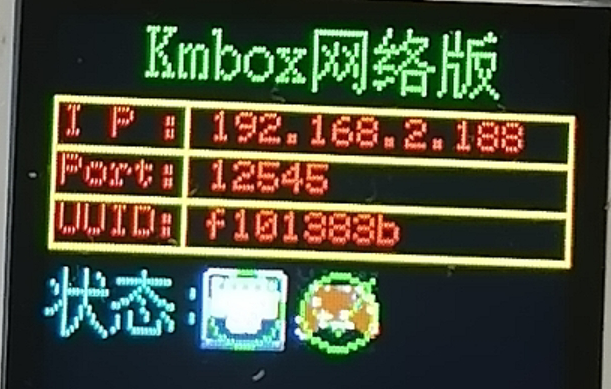
The third parameter, the UUID : Hardware unique identification code. There are on the display.

The above three parameters default to the factory box, and each device is different. Everything is displayed on the display screen. Because each device is different, the source is the targeted in targeted.

The first icon in the status bar is the network connection status. See the following table for details:

|  |  |  |
| --- | --- | --- |
| When the network cable is not connected | Wait to be connected | After successful connecting the box |
|  | 网口 |  |
| When the network cable is not connected, the network port is a fork icon | After accessing the network cable, the fork disappeared. This point indicates that the box is waiting to be connected | The box has been connected by the upper computer (auxiliary software). This icon appears after calling kmNet \_ init. |

The following figure is the display after a successful box connection by calling kmNet \_ init:



# The mouse moves relative to the kmNet \_ mouse \_ move

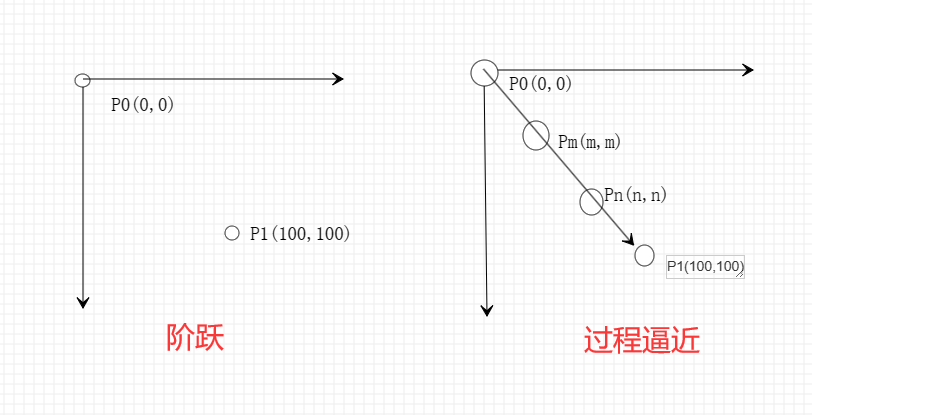
int kmNet\_mouse\_move(short x , short y );

This function is used to control the mouse movement and takes about 1ms. This move is an immediate move, and there is no intermediate state (jump).

As shown in the figure below:

The mouse is currently at the coordinate (0,0) point, after calling kmNet \_ mouse \_ move (100,100). The mouse is moved directly to the coordinates (100,100).

There is no transition in the middle. This movement is easily detected by the game. Normal physical mouse operation moves from (0,0) to (100,100). There will be many transition points (process approaching). The minimum unit of normal mouse movement is 1. Theoretically move from P0 (0,0) to P1 (100,100). There must be at least 100 \* =144 transition points. If you step from (0,0) to (100,100), the mouse data may be considered abnormal by the game. Because you omit the middle 143 transition points. The kmNet \_ mouse \_ move function serves as the most basic move. Suitable for writing your own moving curve.





The 10000 mouse movement function calls took 10188ms as shown. This is equivalent to about 1.0188ms per call. The average call rate was 919 calls per second. PS: Originally A, B board speed is about 300 times per second.

The template speed is 1,000 times per second.

# Mouse simulated manual movement kmNet \_ mouse \_ move \_ auto

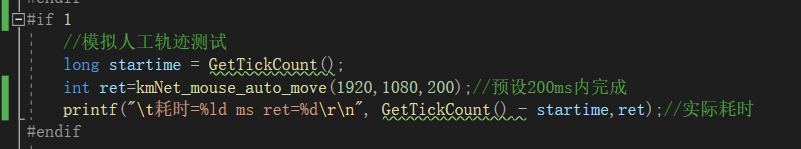
int kmNet\_mouse\_move \_auto (int x , int y ,int time\_ms );//ok

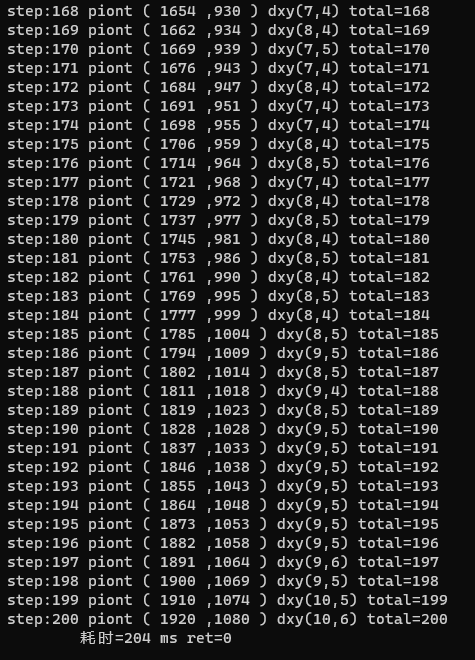
Differers from kmNet \_ mouse \_ move is the third parameter. The kmNet \_ mouse \_ move is the fastest step, while the kmNet \_ mouse \_ mouse \_ move \_ auto is an automatic movement with intermediate processes. Suitable for automatic simulation of manual trajectories. The third parameter specifies how many milliseconds the move takes.for instance:

The kmNet \_ mouse \_ auto \_ move (1920,1080,200); / / the preset is completed within 200ms

Is moved to the (1920,1080) point, required within 200ms. After receiving this instruction, the box will automatically fill in the middle manual movement track.

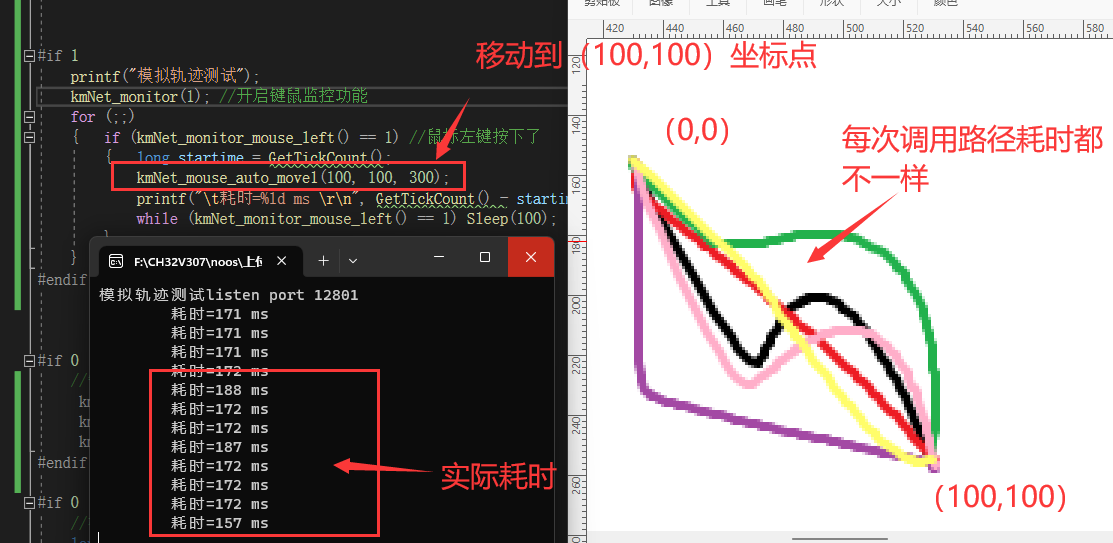
Coordinate point moved to (1920,1080) within 200ms.





As shown above. After giving 200ms parameters, the box is simulated from 204ms (0,0) to (1920,1080). This function works for software calls without trajectory simulation. Boxes can automatically help generate trajectories of intermediate states. In addition, the third parameter is very important, all please refer to the actual manual operation time as a reference.for instance:

The kmNet \_ mouse \_ auto \_ move (1920,1080,1); the third parameter is given to 1ms. This is bound to be problematic. No one can move the mouse pointer in 1ms (1920,1080). The third parameter tries to use the manual operation time as a reference. Parnormal parameters causes the box to be detected. Don't say my hardware problem Thank you.(In order to prevent the key and mouse abnormal, do the hardware is really worry broken heart ah!）。And each time the movement path is randomly different. Refrefer to the demo call code



More advanced algorithms will also be added later. Mouse closing speed, split speed, acceleration, first order, second order, N order, direction vector, etc. Eliminate mouse data abnormalities.

# Mouse button control

Mouse button control includes the following functions:

int kmNet \_ mouse \_ left (int isdown); / / left key control ok

int kmNet\_mouse\_right (int isdown); / / Right-control ok

int kmNet\_mouse\_middle (int isdown); / / medium key control ok

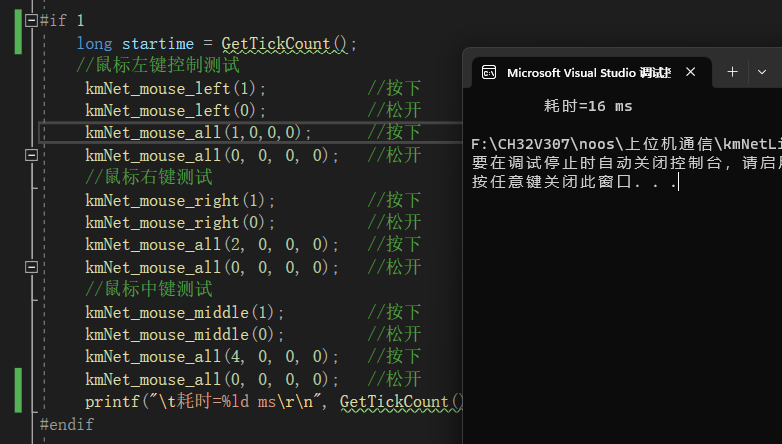
int kmNet\_mouse\_wheel (int wheel); / / roller control ok

int kmNet\_mouse\_all (int button, int x, int y, int wheel); / / mouse all data disposable control ok

Isdown =0 for lift and 1 for press.

Wheel is the downward roller, and negative is the upper roller.

Button Is the mouse button, x, y is the coordinate, while is the roller wheel.



# Keyboard control class functions

Keyboard control class functions mainly have the following several:

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

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

The vkey is the HID code table of the key to call the kmNet \_ keydown function and release the call kmNet \_ keyup function.

# Physical key-mouse state acquisition

You can be used when you need to know whether the keyboard or mouse button on the box is pressed. These functions are directly read the box hardware. Will not call the API of the system, can also prevent hook detection to some extent.

/ / Monitoring series

int kmNet\_monitor (short enable); / / Turn on and off physical mouse status monitoring

int kmNet \_ monitor \_ mouse \_ lift (); / / Press the left mouse button

int kmNet \_ monitor \_ mouse \_ middle(); / / Press the middle mouse button

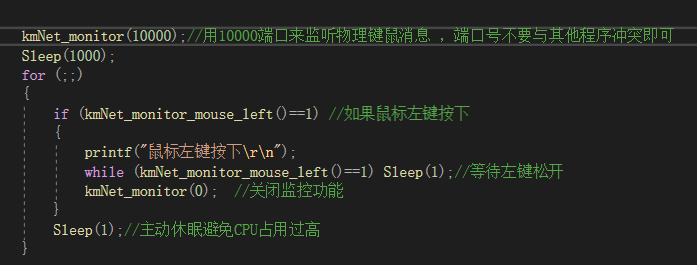
int kmNet\_monitor\_mouse\_right (); / / Press on the right mouse button

int kmNet\_monitor\_mouse\_side1 (); / / Mouse button 1 is pressed

int kmNet\_monitor\_mouse\_side2 (); / / Mouse button 2 is pressed

int kmNet\_monitor\_keyboard (int vk\_key); / / The keyboard specifies whether the key vk \_ key is pressed.

Note that the above functions are in real-time state, that is, the current mouse state is detected at the call time and is not blocked. Does not affect the sending of other instructions with the upper computer. Because the transmitting instructions and receiving monitoring information are conducted on different ports. Please call the kmNet \_ monitor (port) again before calling the monitoring class function to enable the monitoring. The port is the port number of the socket. Note that the Port does not conflict with other native application ports. Port=0 for shutdown monitoring.



# Physical hamster shielding

/ / Physical key-mouse shielding series

int kmNet\_mask\_mouse\_left(int enable ); / / Shield the left mouse button

int kmNet\_mask\_mouse\_right(int enable ); / / Shield the right mouse button

int kmNet\_mask\_mouse\_middle (int enable); / / shield the middle mouse button

int kmNet\_mask\_mouse\_side1(int enable ); / / Shield the mouse side button 1

int kmNet\_mask\_mouse\_side2(int enable ); / / Shield side button 2

int kmNet\_mask\_mouse\_x(int enable ); / / Shield the mouse X-axis coordinates

int kmNet\_mask\_mouse\_y(int enable ); / / Shield the mouse y-axis coordinates

int kmNet\_mask\_mouse\_wheel(int enable ); / / Shield the mouse wheel

int kmNet\_mask\_keyboard(short vkey ); / / Shield the keyboard for the specified keys

int kmNet\_unmask\_all(); / / Unmask all physical shields that have been set

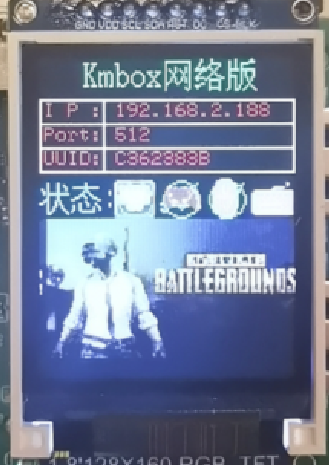
# Display screen control function

int kmNet \_ lcd \_ color (unsigned short rgb565); / / populate the entire LCD screen with the specified color.

Note that the parameter is in the rgb565 format. It can be filled with black to achieve the purpose of clearing the screen.

int kmNet \_ lcd \_ picture \_ bottom (unsigned char \* buff \_ 128 \_ 80); / / Lower half displays 128x80 image

Note that the buff \_ 128 \_ 80 is a picture-taking modulus array with a resolution of 128x80. You can refer to the die acquisition software settings.

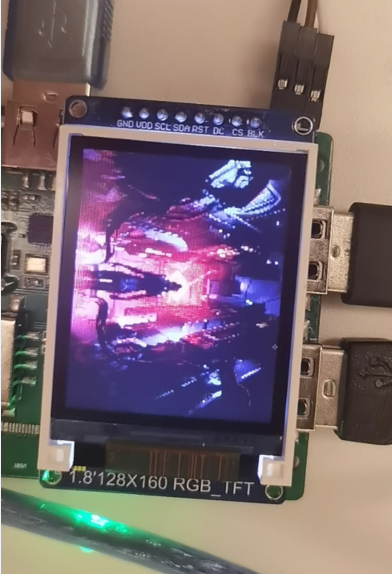


Note that the modulus array size, it must be 128x80x2 bytes.

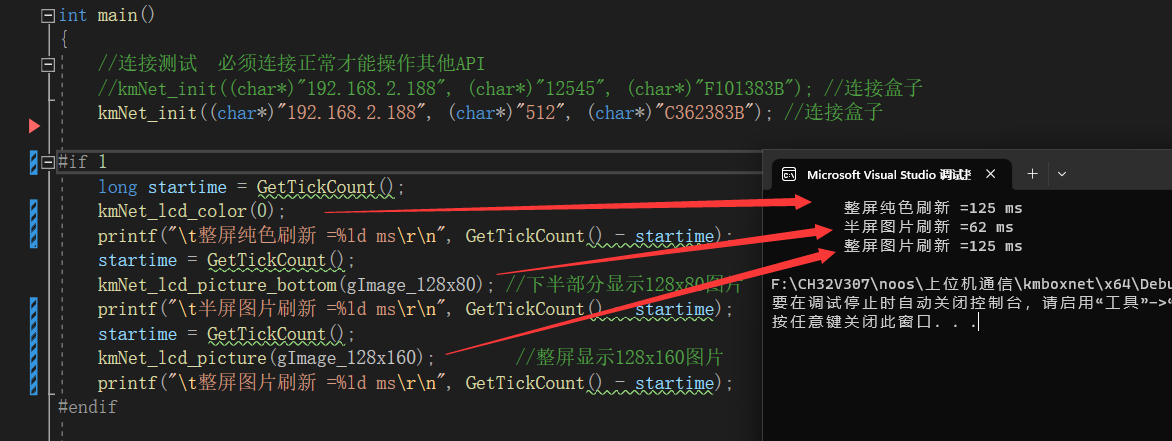
int kmNet\_lcd\_picture(unsigned char \* buff\_128\_160); / / The whole screen displays the 128x160 picture

This function is used to refresh the entire screen. Picture size is fixed to 128x160.





Below, the test code is time-consuming. Full screen takes 125ms. Half screen 62ms



# About the upgrade

The box supports firmware upgrades. The upgrade tool source code is found in the upgrade tool under the doc folder. You can modify the upgrade tool by yourself. Take the official demo, for example. How to upgrade the box firmware later.