

m5stack / M5Stack_MicroPython

forked from [loboris/MicroPython_ESP32_psRAM_LoBo](#)

MicroPython for ESP32 with psRAM support

 View license

☆ 130 stars  259 forks

☆ Star

👁 Watch ▼

Code

Issues 10

Pull requests

Actions

Projects

Wiki

Security

Insights

 master ▼

...

This branch is 16 commits ahead, 42 commits behind loboris:master.

 Pull request  Compare



EeeeBin ...

on 12 Aug 2019 

[View code](#)

README.md

M5Stack Micropython Base **Lobo** **MicroPython**

Getting Started

- See LoBo MicroPython [WiKi](#)
- Please make sure run `./BUILD.sh menuconfig` choose open **MicroPython** --> **Modules** --> **Use Display module**

Simple Build instructions

This works for Linux or macOS, windows is slightly different. Follow the information above to prepare your build environment.

1. Clone the repository `git clone https://github.com/m5stack/M5Stack_MicroPython`

2. Enter the build `cd M5Stack_MicroPython/MicroPython_BUILD/`
3. Build the firmware `./BUILD.sh`
4. Flash the firmware `./BUILD.sh flash`

Upload code

- Use [Ampy](#)
- Use [Lobo Miropython filesystems](#)

Contents

- [MicroPython API](#)
 - [LCD](#)
 - [Button](#)
 - [SD Card](#)
 - [Speaker](#)
 - [GPIO](#)
 - [PWM](#)
 - [ADC](#)
 - [DAC](#)
 - [I2C](#)
 - [SPI](#)
 - [UART](#)
 - [Timer](#)
 - [Neopixel](#)
 - [RTC](#)
- [LoBo MicroPython Wiki](#)

MicroPython API

Micropython Getting Started

In m5stack.py already init timer 0 as EXTI, lcd pwm use timer 1, speak use timer 2, 8

```
import m5stack
```

LCD

Import M5Stack:

```
from m5stack import lcd  
lcd.print('hello world!')
```

Colors

Color value are given as 24 bit integer numbers, 8-bit per color.

For example: **0xFF0000** represents the RED color. Only upper 6 bits of the color component value is used.

The following color constants are defined and can be used as color arguments:

BLACK, NAVY, DARKGREEN, DARKCYAN, MAROON, PURPLE, OLIVE, LIGHTGREY, DARKGREY, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, WHITE, ORANGE, GREENYELLOW, PINK

Drawing

All **drawings** coordinates are **relative** to the **display window**.

Initially, the display window is set to full screen, and there are methods to set the window to the part of the full screen.

Fonts

9 bit-mapped fonts and one vector 7-segment font are included. Unlimited number of fonts from file can also be used.

The following font constants are defined and can be used as font arguments:

FONT_Default, FONT_DefaultSmall, FONT_DejaVu18, FONT_Dejavu24, FONT_Ubuntu, FONT_Comic, FONT_Minya, FONT_Tooney, FONT_Small, FONT_7seg

Methods

lcd.pixel(x, y [,color])

Draw the pixel at position (x,y).

If *color* is not given, current foreground color is used.

lcd.readPixel(x, y)

Get the pixel color value at position (x,y).

lcd.line(x, y, x1, y1 [,color])

Draw the line from point (x,y) to point (x1,y1)

If *color* is not given, current foreground color is used.

lcd.lineByAngle(x, y, start, length, angle [,color])

Draw the line from point (x,y) with length *length* starting at distance *start* from center.

If *color* is not given, current foreground color is used.

The angle is given in degrees (0~359).

lcd.triangle(x, y, x1, y1, x2, y2 [,color, fillcolor])

Draw the triangle between points (x,y), (x1,y1) and (x2,y2).

If *color* is not given, current foreground color is used.

If *fillcolor* is given, filled triangle will be drawn.

lcd.circle(x, y, r [,color, fillcolor])

Draw the circle with center at (x,y) and radius *r*.

If *color* is not given, current foreground color is used.

If *fillcolor* is given, filled circle will be drawn.

lcd.ellipse(x, y, rx, ry [opt, color, fillcolor])

Draw the circle with center at (x,y) and radius *r*.

If *color* is not given, current foreground color is used.

**opt* argument defines the ellipse segment to be drawn, default is 15, all ellipse segments.

Multiple segments can be drawn, combine (logical or) the values.

- 1 - upper left segment
- 2 - upper right segment
- 4 - lower left segment
- 8 - lower right segment

If *fillcolor* is given, filled ellipse will be drawn.

lcd.arc(x, y, r, thick, start, end [color, fillcolor])

Draw the arc with center at (x,y) and radius *r*, starting at angle *start* and ending at angle *end*

The thickness of the arc outline is set by the *thick* argument

If *fillcolor* is given, filled arc will be drawn.

lcd.poly(x, y, r, sides, thick, [color, fillcolor, rotate])

Draw the polygon with center at (x,y) and radius *r*, with number of sides *sides*

The thickness of the polygon outline is set by the *thick* argument

If *fillcolor* is given, filled polygon will be drawn.

If *rotate* is given, the polygon is rotated by the given angle (0~359)

lcd.rect(x, y, width, height, [color, fillcolor])

Draw the rectangle from the upper left point at (x,y) and width *width* and height *height*

If *fillcolor* is given, filled rectangle will be drawn.

lcd.roundrect(x, y, width, height, r [color, fillcolor])

Draw the rectangle with rounded corners from the upper left point at (x,y) and width **width** and height **height**

Corner radius is given by *r* argument.

If **fillcolor** is given, filled rectangle will be drawn.

lcd.clear([color])

Clear the screen with default background color or specific color if given.

lcd.clearWin([color])

Clear the current display window with default background color or specific color if given.

lcd.orient(orient)

Set the display orientation.

Use one of predefined constants:

lcd.PORTRAIT, lcd.LANDSCAPE, lcd.PORTRAIT_FLIP, lcd.LANDSCAPE_FLIP

lcd.font(font [,rotate, transparent, fixedwidth, dist, width, outline, color])

Set the active font and its characteristics.

| Argument | Description |
|-------------|--|
| font | required, use font name constant or font file name |
| rotate | optional, set font rotation angle (0~360) |
| transparent | only draw font's foreground pixels |
| fixedwidth | draw proportional font with fixed character width, max character width from the font is used |

| Argument | Description |
|----------|---|
| dist | only for 7-seg font, the distance between bars |
| width | only for 7-seg font, the width of the bar |
| outline | only for 7-seg font, draw the outline |
| color | font color, if not given the current foreground color is used |

lcd.attrib7seg(dist, width, outline, color)

Set characteristics of the 7-segment font

| Argument | Description |
|----------|---------------------------|
| dist | the distance between bars |
| width | the width of the bar |
| outline | outline color |
| color | fill color |

lcd.fontSize()

Return width and height of the active font

lcd.print(text[,x, y, color, rotate, transparent, fixedwidth, wrap])

Display the string *text* at position (x,y).

If *color* is not given, current foreground color is used.

- **x**: horizontal position of the upper left point in pixels, special values can be given:
 - CENTER, centers the text
 - RIGHT, right justifies the text
 - LASTX, continues from last X position; offset can be used: LASTX+n
- **y**: vertical position of the upper left point in pixels, special values can be given:
 - CENTER, centers the text
 - BOTTOM, bottom justifies the text
 - LASTY, continues from last Y position; offset can be used: LASTY+n
- **text**: string to be displayed. Two special characters are allowed in strings:
 - '\r' CR (0x0D), clears the display to EOL
 - '\n' LF (0x0A), continues to the new line, x=0

lcd.text(x, y, text [, color])

Display the string *text* at position (x,y).

If *color* is not given, current foreground color is used.

- **x**: horizontal position of the upper left point in pixels, special values can be given:
 - CENTER, centers the text
 - RIGHT, right justifies the text
 - LASTX, continues from last X position; offset can be used: LASTX+n
- **y**: vertical position of the upper left point in pixels, special values can be given:
 - CENTER, centers the text
 - BOTTOM, bottom justifies the text
 - LASTY, continues from last Y position; offset can be used: LASTY+n
- **text**: string to be displayed. Two special characters are allowed in strings:
 - '\r' CR (0x0D), clears the display to EOL
 - '\n' LF (0x0A), continues to the new line, x=0

lcd.textWidth(text)

Return the width of the string *text* using the active font fontSize

lcd.textClear(x, y, text [, color])

Clear the the screen area used by string *text* at position (x,y) using the bacckground color *color*.

If *color* is not given, current background color is used.

lcd.image(x, y, file [,scale, type])

Display the image from the file *file* on position (x,y)

- **JPG** and **BMP** can be displayed.
- Constants **lcd.CENTER**, **lcd.BOTTOM**, **lcd.RIGHT** can be used for x&y
- **x** and **y** values can be negative

scale (jpg): image scale factor: 0 to 3; if scale>0, image is scaled by factor $1/(2^{\text{scale}})$ (1/2, 1/4 or 1/8)

scale (bmp): image scale factor: 0 to 7; if scale>0, image is scaled by factor $1/(\text{scale}+1)$

type: optional, set the image type, constants *lcd.JPG* or *lcd.BMP* can be used. If not set, file extension and/or file content will be used to determine the image type.

lcd.setwin(x, y, x1, y1)

Set active display window to screen rectangle (x,y) - (x1,y1)

lcd.resetwin()

Reset active display window to full screen size.

lcd.savewin()

Save active display window dimensions.

lcd.restorewin()

Restore active display window dimensions previously saved with savewin().

lcd.screenSize()

Return the display size, (width, height)

lcd.winSize()

Return the active display window size, (width, height)

lcd.hsb2rgb(hue, saturation, brightness)

Converts the components of a color, as specified by the HSB model, to an equivalent set of values for the default RGB model.

Returns 24-bit integer value suitable to be used as color argument

Arguments

- **hue**: float: any number, the floor of this number is subtracted from it to create a fraction between 0 and 1. This fractional number is then multiplied by 360 to produce the hue angle in the HSB color model.
- **saturation**: float; 0 ~ 1.0
- **brightness**: float; 0 ~ 1.0

lcd.compileFont(file_name [,debug])

Compile the source font file (must have .c extension) to the binary font file (same name, .fon extension) which can be used as external font.

If *debug=True* the information about compiled font will be printed.

You can create the **c** source file from any **tft** font using the included [ttf2c_vc2003.exe](#) program. See [README](#) for instructions.

You can create **c** source file or **.fon** file from **tft** font using [online converter](#).

Button

Method

```
buttonA.isPressed()
buttonA.isReleased()
buttonA.pressedFor(timeout)

# if set the callback param, it will interrupt callback function
# or if not set param it will return result at once
buttonA.wasPressed(callback=None)
buttonA.wasReleased(callback=None)
buttonA.releasedFor(timeout, callback=None)
```

Example

Loop:

```
from m5stack import *
import utime

while True:
    if buttonA.wasPressed():
        lcd.print('Button A was Pressed\n')

    if buttonA.wasReleased():
        lcd.print('Button A was Released\n')

    if buttonA.pressedFor(1.5):
        lcd.print('Button A pressed for 1.5s\n')

    if buttonA.releasedFor(2):
        lcd.print('Button A released for 2s press hold\n')

    utime.sleep(0.1)
```

Callback:

```
from m5stack import *

def on_wasPressed():
    lcd.print('Button B was Pressed\n')

def on_wasReleased():
    lcd.print('Button B was Released\n')

def on_releasedFor():
    lcd.print('Button B released for 1.2s press hold\n')

buttonB.wasPressed(on_wasPressed)
buttonB.wasReleased(on_wasReleased)
buttonB.releasedFor(1.2, on_releasedFor)
```

SD Card

```
import uos

uos.mountsd()
uos.listdir('/sd')
```

Speaker

```
from m5stack import *

speaker.volume(2)
speaker.tone(freq=1800)
speaker.tone(freq=1800, duration=200) # Non-blocking
```

GPIO

```
import machine

pinout = machine.Pin(0, machine.Pin.OUT)
pinout.value(1)

pinin = machine.Pin(2, machine.Pin.IN)
val = pinin.value()
```

PWM

```
pwm = machine.PWM(pin [, freq=f] [, duty=d] [, timer=tm])
pwm.init([ freq=f] [, duty=d] [, timer=tm])
```

| Arg | Description |
|-----|---|
| pin | esp32 GPIO number to be used as pwm output can be given as integer value or machine.Pin object |

| Arg | Description |
|-------|---|
| freq | optional , default 5 kHz; pwm frequency in Hz (1 - 40000000) |
| duty | optional , default 50% kHz; pwm duty cycle in % (0 - 100) |
| timer | optional , default 0; pwm timer (0 - 3) |

PWM channel is selected automatically from 8 available pwm channels.

```
import machine
pwm = machine.PWM(26)
pwm.freq(5000)
pwm.duty(66) # 0.0 ~ 100.0
```

ADC

```
import machine

adc = machine.ADC(35)
adc.read()
```

DAC

```
import machine

dac = machine.DAC(machine.Pin(26))
dac.write(128)
```

I2C

```
from machine import I2C

i2c = I2C(freq=400000, sda=21, scl=22)
# create I2C peripheral at frequency of 400kHz
# depending on the port, extra parameters may be requ
# to select the peripheral and/or pins to use

i2c.scan() # scan for slaves, returning a list of 7-bit addresse
```

```

i2c.writeto(42, b'123')      # write 3 bytes to slave with 7-bit address 42
i2c.readfrom(42, 4)         # read 4 bytes from slave with 7-bit address 42

i2c.readfrom_mem(42, 8, 3)   # read 3 bytes from memory of slave 42,
                             # starting at memory-address 8 in the slave
i2c.writeto_mem(42, 2, b'\x10') # write 1 byte to memory of slave 42
                             # starting at address 2 in the slave

```

SPI

```

from machine import SPI, Pin

spi = SPI(
    spihost=SPI.HSPI,
    baudrate=2600000
    sck=Pin(18),
    mosi=Pin(23),
    miso=Pin(19),
    cs=Pin(4)
)

spi.write(buf) #NOHEAP
spi.read(nbytes, *, write=0x00) #write is the byte to ?output on MOSI for each byte r
spi.readinto(buf, *, write=0x00) #NOHEAP
spi.write_readinto(write_buf, read_buf) #NOHEAP; write_buf and read_buf can be the sa

```

UART

```

from machine import UART

uart2 = UART(2, tx=17, rx=16)
uart2.init(115200, bits=8, parity=None, stop=1)
uart2.read(10)      # read 10 characters, returns a bytes object
uart2.read()        # read all available characters
uart2.readline()    # read a line
uart2.readinto(buf) # read and store into the given buffer
uart2.write('abc')  # write the 3 characters

```

Timer

```
tm = machine.Timer(timer_no)
```

timer_no argument is the timer number to be used for the timer. It can be 0 - 3 for 4 hardware timers or 4 - 11 for extended timers. If extended timer is selected, timer 0 must already be configured in EXTBASE mode.

```
import machine

tcounter = 0

p1 = machine.Pin(27)
p1.init(p1.OUT)
p1.value(1)

def tcb(timer):
    global tcounter
    if tcounter & 1:
        p1.value(0)
    else:
        p1.value(1)
    tcounter += 1
    if (tcounter % 10000) == 0:
        print("[tcb] timer: {} counter: {}".format(timer.timernum(), tcounter))

t1 = machine.Timer(2)
t1.init(period=20, mode=t1.PERIODIC, callback=tcb)
```

Neopixel

```
import machine, time

np = machine.Neopixel(machine.Pin(22), 24)

def rainbow(loops=120, delay=1, sat=1.0, bri=0.2):
    for pos in range(0, loops):
        for i in range(0, 24):
            dHue = 360.0/24*(pos+i);
            hue = dHue % 360;
            np.setHSB(i, hue, sat, bri, 1, False)
        np.show()
        if delay > 0:
            time.sleep_ms(delay)

def blinkRainbow(loops=10, delay=250):
    for pos in range(0, loops):
        for i in range(0, 24):
            dHue = 360.0/24*(pos+i);
            hue = dHue % 360;
```

```
np.setHSB(i, hue, 1.0, 0.1, 1, False)
np.show()
time.sleep_ms(delay)
np.clear()
time.sleep_ms(delay)
```

RTC

```
import machine
import utime

rtc = machine.RTC()
rtc.ntp_sync(server="hr.pool.ntp.org", tz="CET-1CEST")
rtc.synced()
True
utime.gmtime()
(2018, 1, 29, 16, 3, 18, 2, 29)
utime.localtime()
(2018, 1, 29, 17, 3, 30, 2, 29)
```

Releases

No releases published

Packages

No packages published

Languages

● C 89.4% ● C++ 5.4% ● Python 2.7% ● Makefile 0.8% ● Shell 0.4% ● CMake 0.4%
● Other 0.9%