K210-MicroPython &OpenMV
User Manual

canaan | KENDRYTE

Super Computing is what we do,
Social Enrichment is why we do it.

About this manual

This document provides a manual that how to K210-MicroPython & OpenMV for users.

Release Notes

Date	Version	Release Notes
2020-12-20	V 1.0	Initial version
2021-01-15	V 1.0.1	Modified OTA、IDE sections
2021-03-15	V1.0.2	Modified content and formalized text
2021-04-13	V1.0.2-En	English version

Disclaimer

The Information in this document, including the reference URL address, is subject to change without prior notice.the document is provided "as is" and is not liable for any warranty, including any warranty for merchantability, application to a particular purpose or non-infringement, and any warranty mentioned elsewhere by any proposal, specification or sample.This document is not responsible for any infringement of any patent rights arising from the use of the information in this document. this document here does not grant any intellectual property use license, either express or implied, by estoppel or otherwise.It is hereby declared that all trademark names, trademarks and registered trademarks mentioned in the text are the property of their respective owners.

Copyright Notice

Directory

1. Overview

The document mainly tells that the features and advantages of K210-MicroPython, scenes to be applied, and various of interfaces.

Users can learn and use quickly with examples provided of each module, develop prototypes of hardware products without too much hardware knowledge.

1.1 Audiences

- Software/Hardware product manager
- Software/Hardware engineer
- Project manager
- Hardware Beginner
- Python engineer
- ...

1.2 Application scenes

MicroPython running well on K210, which can bring convenience to developers at all stages of product development, is developed based on Python3.

The developer can quickly achieve various of applications through libraries provided such as the LED、liquid crystal, steering gear、a variety of sensors、SD cards、UART、I2C and so on, rather than study the use of the underlying hardware modules with reading the register manual. So, it reduces the difficulty of development, decreases the duplication of development work, speeds up the development process and improves the development efficiency. Therefore, it only takes a few hours for junior embedded developers that a task was spent days or even weeks in the past.

With the development of semiconductor technology, the chip is more capable of computing resource and less cost. Consequently, MicroPython can be applied for more and more business purposes.

- Prototype validation

As we known, it is an essential step to validate prototype for new products, which needs to design the approximate model of the product in the shortest time and reviews the business process or feedback from market. In contract to traditional development methods, Using MicroPython makes the verification process easier and

faster.

MicroPython is good at the development of network communication on developing IoT applications, which can save development time instead of C/C++.

- Test hardware rapidly

On developing embedded products, it generally divided into hardware development and software development. As a hardware engineer is not good at software development, which results in that software engineers may spend a lot of time helping hardware engineers find designing or welding problems. For these problems, it is done easily with simple python commands by the hardware engineer alone.

- DIY

Using MicroPython can code with any text editor, without other complex settings such as special software environments and additional hardware. It's very suitable for makers to develop some creative projects with MicroPython.

- Education

MicroPython, as a language used easily, is very suitable for programming beginners to quickly develop some fun projects , such as college students or amateurs, who learn programming ideas and improve own hands-on ability in the process of development.

1.3 Advantages

As MicroPython using easily with powerful functions, elegant and simple grammar, it not only can reduce the threshold of embedded development, but also let more people experience programming fun.

Using MicroPython can easily control the hardware without understanding registers, data manual and manufacturer's library functions.

Meanwhile, Peripheral devices and common functions with corresponding modules, not only can reduce the difficulty of development but also can make development and transplantation easy and fast.

1.4 Features

- Support OpenMV graphics algorithm
- Support OpenMV IDE debug

- Support OTA upgrade
- Support AT (esp8266/w600), w5500(ethernet), RW007(wifi)
- Support SD card
- Support MicroPython standard libraries and most specific libraries
- Support littleVGL mpy module
- Support for audio capture/play

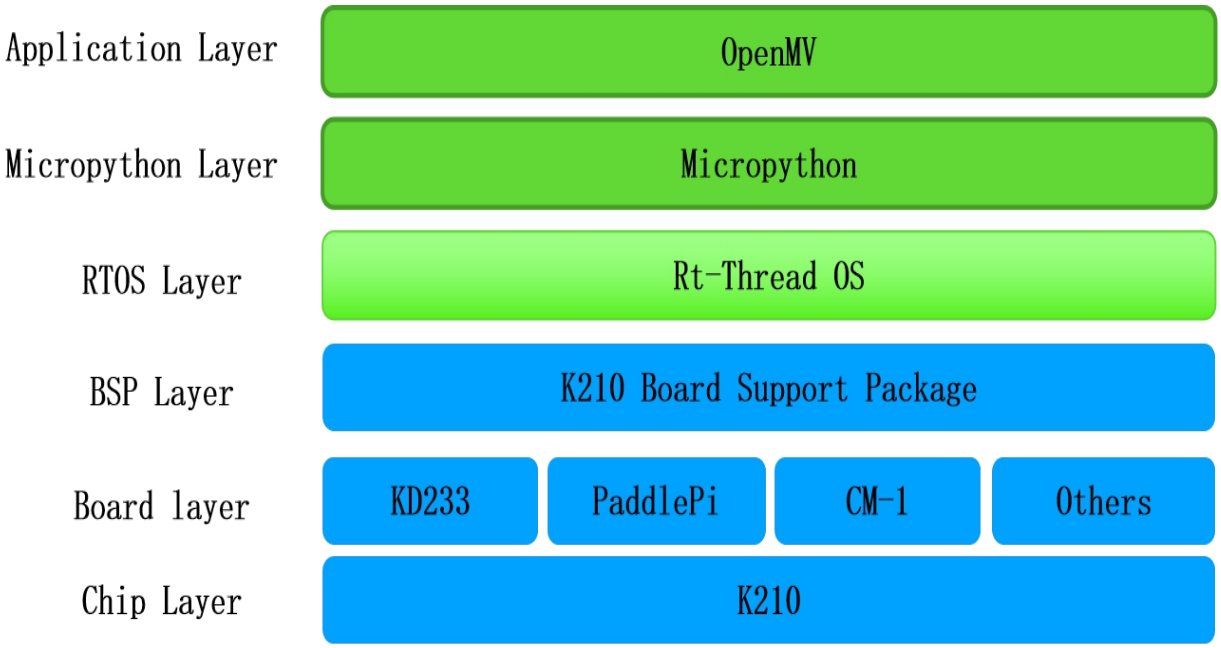
2. Development environment

2.1 Warm-up

- Support development environment and IDE tool in Windows and Linux
- When MicroPython or OMV is turned on, the available SRAM is about 2.9MB;or not, the available SRAM is about 4.5MB;
- Operating system version: Rt-Thread 4.0.3
- Compiler version: xpack-riscv-none-embed-gcc-8.3.0-1.9;
- Development tools may choose the MicroPython IDE provided by the Rt-Thread, downloading address as follows:

  Rt-Thread MicroPython Develop Environment

2.2 Framework



2.3 Environment building

Windows:

1) Download omv-k210(this development package downloaded has integrated with Rt-Thread 4.0.3. compiler: xpack-riscv-none-embed-gcc-8.3.0-1.9 windows)
   Refer to https://github.com/kendryte/K210-Micropython-OpenMV
2) Install the compiler and add the installation path **\ xpack-riscv-none-embed-gcc -8.3.0-1.9\bin** to the environment variable;
3) Install the Rt-Thread configuration tool **ENV**, which has the functions of configuring and calling the compiler.
   Configuration Reference: Getting Started of QEMU (Windows)

Linux:

1) Download omv-k210(this development package downloaded has integrated with Rt-Thread 4.0.3. compiler: xpack-riscv-none-embed-gcc-8.3.0-1.9 Linux)
   Refer to https://github.com/kendryte/K210-Micropython-OpenMV
2) Install the compiler and add the installation path **/opt/ xpack-riscv-none-embed-gcc -8.3.0-1.9/bin** to the environment variable;
4) Install the Rt-Thread configuration tool ENV, which has the functions of

configuring and calling the compiler.

Configuration Reference: Getting Started of QEMU (Ubuntu)

2.4 SDK structure

```
SDK directory structure:
--doc                        # user manual
--src                        # The SDK project source code
--toolchain                  # compilation tool chain


src folder:
--bsp                      # board support package
   --driver                # Peripheral Driver
   --kendryte-sdk          # Chip vendor support package
--extmods                  # MicroPython Extension Module for User
   --lvgl                  # lvgl module
   --k210                  # K210 on-chip or on-board peripheral MPY module
   --cameras               # camera drivers
   --moddefsmisc.h         # Module Declaration
   --qstrdefsmisc.h        # String declaration
--OpenMV
   --omv/boards            # Board - related function configuration
   --omv/caremas           # camera drivers
   --omv/hal               # File system, peripherals and other underlying
interfaces
   --omv/img               # Image Algorithm
   --omv/py                # MPY module of application interface
   --omv/rt_thread_dbg.c # the communication thread with OpenMV IDE
--projects                 # User project code, when the user has more than one
project,can be uniformly placed in this directory
   --boot_k210             # bootloader reference code
   --app_k210              # Application project reference code
--rtthread                 # rtthread kernel code
--test
boot_k210 directory structure:
--boot_k210
```

```
    --applications        # Application layer code
    --packages            # Online software packages, this directory is
automatically produced by the RTThread ENV tool
    --rtconfig.h          # Function profile, generated by the configuration
tool
    --rtconfig.py         # Toolchain configuration scripts
app_k210 directory structure:
--app_k210
  --applications
  --packages
  --rtconfig.h
  --rtconfig.py
  --moddefs.user.extmods.h  # Register the user module into the MicroPython-
v1.13 transplanted based on RTThread
  --qstrdefs.user.extmods.h   # Register string
```

2.5 Menuconfig

Note: When a configuration item is closed and saved,the menuconfig will be reopen and be restored to the default, which needs to be reset.

- Turn AT module (w600) on

```
Rt-Thread online packages  --->
  IoT - internet of things  --->
     [*] AT DEVICE: Rt-Thread AT component porting ... --->
        [*]   WinnerMicro W60X  --->
          [ ]   Enable initialize by thread  (NEW)
          [*]   Enable sample  (NEW)
          (rtthread) WIFI ssid (NEW)
          (12345678) WIFI password (NEW)
          (uart1) AT client device name  (NEW)
          (512)   The maximum length of receive line buffer (NEW)
```

- Turn W5500 on

```
Rt-Thread online packages  --->
  IoT - internet of things  --->
    [*] WIZnet: WIZnet TCP/IP chips SAL  framework implement  --->
      WIZnet: WIZnet TCP/IP chips SAL framework  implement
        WIZnet device type (W5500)  --->
        WIZnet device configure  --->
          (spi30) SPI device name (NEW)
          (10) Reset PIN number (NEW)
          (11) IRQ PIN number (NEW)
        [*]   Enable alloc IP address through DHCP (NEW)
        [*]   Enable Ping utility  (NEW)
        [ ]   Enable debug log output (NEW)
        Version (latest)  --->
```

- Turn RW007 on

```
Step 1:
Hardware Drivers  --->
  [*] Enable RW007 wifi  module
    (spi11) the SPIDEV rw007 driver  on (NEW)      # spi11 means cs1 of spi1 bus
    (8) reset PIN (NEW)                            # modify according to connection
    (7)  INT and BUSY status  pin (NEW)            # modify according to connection
```

```
Step 2:
Rt-Thread online packages  --->
  IoT - internet  of things  --->
    Wi-Fi -->
      -*- rw007: SPI WIFI  rw007 driver  --->
        version (latest)  --->
        example driver port (not use example driver, porting  by myself)  --->
        (30000000) SPI Max Hz  (NEW)
```

- Turn MicroPython on

```
Rt-Thread online packages  --->
  language packages  --->
    [*] MicroPython: ... -->
      ...
      User Extended Module  --->
        [*] modules define in your  project
      (1500000) Heap size for python run environment
      Version (latest)  --->
```

- Turn OpenMV on

```
OpenMV --->
  [*]  Enable  OpenMV(Cross-Platform)
  Board type (Kendryte KD233) ---> [*]
  Hal K210
  (230400) Total Framebuffer Size      # the image size up to 320*240*3

  [ ]  Enable LAB LUT                  # static data list of  LAB algorithm,  off optimization
  [ ]  Enable YUV LUT                  volume
```

- Turn on peripheral drivers

```
Note: After selecting a configuration item with "-- ", you need to enter
the sub-menu for detailed configuration
Hardware Drivers  --->
    IO Groups Power Supply Settings --->   # set IO BANK voltage
  [*] Enable High Speed UART
    General Purpose UARTs --->
    [*] Enable I2C0
    (35)  I2C0 SCL pin number      # Pins specify pins numbered as the
                                    number after the chip name IOXX
    (34)  I2C0 SDA pin number
  [ ] Enable I2C1
  [ ] Enable I2C2
  [*] Enable SPI1  --->
  [*] Enable LCD on SPI0  --->
  [ ] Enable I2S0(Play Only)
  [ ] Enable I2S1(Record Only)
  [ ] Enable I2S2




  [ ] Enable PWM   ----
  [ ] Enable Timer0 ____
  [ ] Enable Timer1 ____









  [ ] Enable DVP Camera
  [ ] Enable bridge module          # Configure the communication
  conversion module between OpenMV IDE and OpenMV IDE
  [ ] Enable RW007 wifi module
```

- Turn littleVGL on

Note: in the latest version, the screen size obtained from the driver, which is not configurable

```
Rt-Thread online packages  --->
  system packages  --->
    [*] LittlevGL2RTT: The LittlevGl gui ...  --->
      version (latest)  --->
      LittlevGL2RTT Options  --->
        Color depth (32bit)  --->  # Choose 32bit color when using LVGL MPY Bindings
        Garbage Collector (enable GC)  --->  # Choose the item when using LVGL MPY Bindings
    [ ]  LittlevGL2RTT demo example     # Close the item when using LVGL MPY
```

- Turn littleVGL MPY Bindings on

```
Note: If not configured before this function is turned on littleVGL, please
configure it

  extmods  --->
    [*]Enable MPY extmods
    [*]   Enable lvgl mpy bindings (NEW)   ----
```

- Turn K210 specific mpy extension module on

```
Note: DVP module is closed by default

  extmods  --->
    [*]   Enable K210extmods (NEW)    --->
```

- Turn Configure network function on

```
Note: network modules such as W5500 or RW007 are selected to open network
related functions by default, but a small number of configurations are required

  Rt-Thread Components  --->
     Network  --->
      Socket abstraction layer  --->
        [*]Enable socket abstraction layer
        [*]   ...Enable BSD socket operated by file
        light weight TCP /IP stack          --->    # don't need to
                                                    configure this item
                                                    without RW007
        (4096) the stack size of lwIP thread
        ...    # Other tips
```

- Turn SD card on

```
Step 1: Configure SPI appropriate driver

Step 2:
  Rt-Thread Components  --->
     Device Drivers  --->
       -*-Using SPI Bus /Device device drivers
         [*]   Using SD /TF card driver with spi

Step 3:
  Rt-Thread Components  --->
    Device virtual file system  --->
      [*]   Enable elm -chan fatfs
```

- Turn OTA Downloading on

```
  Rt-Thread online packages  --->
     IoT -internet of things  --->
      [*]ota_downloader : The firmware downloader...    --->
        [*]   Enable HTTP /HTTPS OTA
         (http : //xxx/xxx/rtthread.rbl) HTTP OTA Download ...(NEW)# Modify on Demand
        [*]   Enable Ymodem OTA
        Version (latest)  --->
```

3. MicroPython Standards Library

- **Built functions**

• abs()	• hash()	• range()
• all()	• hex()	• repr()
• any()	• id()	• reversed()
• bin()	• input()	• round()
• class bool	• class int	• class set
• class bytearray	• classmethod from_bytes(bytes, byteorder) In MicroPython, byteorder parameter must be positional (this is compatible with CPython).	• setattr()
• class bytes	• to_bytes(size, byteorder) In MicroPython, byteorder parameter must be positional (this is compatible with CPython).	• class slice The slice builtin is the type that slice objects have.
• callable()	• isinstance()	• sorted()
• chr()	• issubclass()	• staticmethod()
• classmethod()	• iter()	• class str
• compile()	• len()	• sum()
• class complex	• class list	• super()
• delattr(obj, name)	• locals()	• class tuple
• class dict	• map()	• type()
• dir()	• max()	• zip()
• divmod()	• class memoryview	
• enumerate()	• min()	
• eval()	• next()	
• exec()	• class object	
• filter()	• oct()	
• class float	• open()	
• class frozenset	• ord()	
• getattr()	• pow()	
• globals()	• print()	
• hasattr()	• property()	

- **Exceptions**

• exception AssertionError	• exception OSError *See CPython documentation: OSError. MicroPython doesn't implement errno attribute, instead use the standard way to access* exception arguments: exc.args[0].*
• exception AttributeError	• exception RuntimeError
• exception Exception	• exception StopIteration
• exception ImportError	• exception SyntaxError
• exception IndexError	• exception SystemExit *See CPython documentation: SystemExit.*
• exception KeyboardInterrupt	• exception TypeError *See CPython documentation: TypeError.*
• exception KeyError	• exception ValueError
• exception MemoryError	• exception ZeroDivisionError
• exception NameError	
• exception NotImplementedError	

3.1 uarray

uarray module defines an object type, It can succinctly represent basic values,such as array, ocharacters, integers and floating-point. Supported format codes: b,B,h, H,i,I,l, L,q, Q,f, d (the last 2 depending on the floating-point support).

- **Constructor**

class uarray.array (typecode [, iterable])

Description: Create an array with elements of a given type, which initial contents of the array are provided by the iterable. if not, an empty array is created.

- **Arguments**

```
typecode : Type of array
iterable : initial content of array
```

- **Demo**

```
>>>import uarray
>>>a =uarray.array('i',[2,4,1,5])
>>>b =uarray.array('f')
>>>print (a)
array('i',[2,4,1,5])
>>>print (b)
array ('f')
```

- **Function**

uarray.append (val)

Append a new element val to the end of the array,growing it

Example:

```
>>>a =uarray.array('f',[3,6])
>>>print(a)
array('f',[3.0,6.0])
>>>a.append(7.0)
>>>print(a)
array('f',[3.0,6.0,7.0])
```

uarray.extend (iterable)

Append new elements as contained in iterable to the end of array, growing it

Example:

```
>>>a =uarray.array('i',[1,2,3])
>>>b =uarray.array('i',[4,5])
>>>a .extend(b)
>>>print(a)
array ('i',[1,2,3,4,5])
```

more information refer to uarray

3.2 cmath

cmath module provides some methods for complex number operation,which can accept integers, floating-point, or complex numbers as arguments, which also accept any Python objects with complex or floating-point methods. These methods are used to respectively convert objects to complex or floating-point , and then apply to the result of the conversion.

- **Function**

cmath.cos (z)

Return the cosine of z

cmath.exp (z)

Return the exponential of z

cmath.log (z)

Return the natural logarithm of z. The branch cut is along the negative real axis

cmath.log10(z)

Return the base-10 logarithm of z The branch cut is along the negative real axis

cmath.phase (z)

Return the phase of the number z, in the range (- pi,+pi]

cmath.polar (z)

Return, as a tuple, the polar form of z

cmath.rect (r, phi)

Returns the complex number with modulus r and phase phi

cmath.sin (z)

Return the sine of z

cmath.sqrt (z)

Return the square root of z

- **Constant**

cmath.e

base of the natural logarithm

cmath.pi

the ratio of a circle's circumference to its diameter

more information refer to cmath

3.3 gc

gc module provides control interfaces of garbage collector

- **Function**

gc.enable()

Enable automatic garbage collection

gc.disable()

Disable automatic garbage collection. Heap memory can still be allocated, and garbage collection can still be initiated manually using gc.collect()

gc.collect()

Run a garbage collection

gc.mem_alloc()

Return the number of bytes of heap RAM allocated

gc.mem_free()

Return the number of bytes of available heap RAM, or -1 if this amount is not known.

more information refer to gc

3.4 math

- **Function**

math.acos (x)

Return the inverse cosine of x

math.acosh (x)

Return the inverse hyperbolic cosine of x

math.asin (x)

Rerurn the inverse sine of x

Example:

```
>>>x =math .asin (0.5)
>>>print (x)
0.5235988
```

math.asinh (x)

Return the inverse hyperbolic sine of x

math.atan (x)

Return the inverse tangent of x

math.atan2(y, x)

Return the principal value of the inverse tangent of y/x

math.atanh (x)

Return the inverse hyperbolic tangent of x

math.ceil (x)

Return an integer, being x rounded towards positive infinity

Example:

```
>>>x =math .ceil (5.6454)
>>>print (x)
6
```

math.copysign (x, y)

Returns x with the sign of y

math.cos (x)

Return the cosine of x.

Example: calculate cos60°

```
>>>math .cos (math.radians (60))
0.5
```

math.cosh (x)

Return the hyperbolic cosine of x

math.degrees (x)

Return radians x converted to degrees

Example:

```
>>>x =math .degrees (1.047198)
>>>print (x)
60.00002
```

math.erf (x)

Return the error function of x

math.erfc (x)

Return the complementary error function of x

math.exp (x)

Return the exponential of x

Example:

```
>>>x =math .exp (2)
>>>print (x)
7.389056
```

math.expm1(x)

Return exp(x) - 1

math.fabs (x)

Return the absolute value of x

Example:

```
>>>x =math .fabs (-5)
>>>print (x)
5.0
>>>y =math .fabs (5.0)
>>>print (y)
5.0
```

math.floor (x)

Return an integer, being x rounded towards negative infinity

Example:

```
>>>x =math .floor (2.99)
>>>print (x)
2
>>>y =math .floor (-2.34)
>>>print (y)
-3
```

math.fmod (x, y)

Return the remainder of x/y

Example:

```
>>>x =math .fmod (4,5)
>>>print (x)
4.0
```

math.frexp (x)

Decompose a floating-point into its mantissa and exponent. The return value is a tuple (m,e) so that the x ==m *2**e exactly. The function returns (0.0,0) if x ==0, otherwise the return relation 0.5<x ==abs (m)<1 holds.

math.gamma (x)

Return the gamma function of x

Example:

```
>>>x =math .gamma (5.21)
>>>print (x)
33.08715
```

math.isfinite (x)

Return True if x is finite

math.isinf (x)

Return True if x is infinite

math.isnan (x)

Return True if x are not a-number

math.ldexp (x, exp)

Return x *(2**exp)

math.lgamma (x)

Return the natural logarithm of the gamma function of x

Example:

```
>>>x =math.lgamma (5.21)
>>>print (x)
3.499145
```

math.log (x)

Return the natural logarithm of x

Example:

```
>>>x =math .log (10)
>>>print (x)
2.302585
```

math.log10(x)

Return the base-10 logarithm of x

Example:

```
>>>x =math .log10(10)
>>>print (x)
1.0
```

math.log2(x)

Return the base-2 logarithm of x

Example:

```
>>>x =math .log2(8)
>>>print (x)
3.0
```

math.modf (x)

Return a tuple of two floats, being the fractional and integral parts of x. Both return values have the same sign as x

math.pow (x, y)

Return x to the power of y

Example:

```
>>>x =math .pow (2,3)
>>>print (x)
8.0
```

math.radians (x)

Return degrees x converted to radians

Example:

```
>>>x =math .radians (60)
>>>print
(x)1.047198
```

math.sin (x)

Return the sine of x

Example: calculate sin90°

```
>>>math .sin (math.radians (90))
1.0
```

math.sinh (x)

Return the hyperbolic sine of x

math.sqrt (x)

Return the square root of x

Example:

```
>>>x =math .sqrt (9)
>>>print (x)
3.0
```

math.tan (x)

Return the tangent of x

Example: Calculate tan60°

```
>>>math .tan (math.radians (60))
1.732051
```

math.tanh (x)

Return the hyperbolic tangent of x

math.trunc (x)

Return an integer, being x rounded towards 0

Example:

```
>>>x =math .trunc (5.12)
>>>print (x)
5
>>>y =math .trunc (-6.8)
>>>print (y)
-6
```

- **Constant**

math.e

The base of the natural logarithm

Example:

```
>>>import math
>>>print (math.e)
2.718282
```

math.pi

The ratio of a circle’s circumference to its diameter

Example:

```
>>>print (math.pi)
3.141593
```

more information refer to math

3.5 sys

sys module provides system-specific functions.

- **Function**

sys.exit (retval =0)

Terminate the exit code given with the current program. The method throws a SystemExit exception.

sys.print_exception (exc, ftle =sys.stdout)

Print exception with a traceback to a file-like object file(or sys.stdout by default)

Tips: This is simplified version of a function which appears in the traceback module in CPython. Unlike traceback.print_exception(), this function takes just exception value instead of exception type, exception value, and traceback object; file argument should be positional; further arguments are not supported. CPython-compatible traceback module can be found in micropython-lib.

- **Constant**

sys.argv

A mutable list of arguments with which the current program was started.

sys.byteorder

The byte order of the system ("little" or "big")

sys.implementation

Object with information about the current Python implementation. For MicroPython, it has following attributes:

- Name: string “MicroPython "

- Version: tuple (primary, secondary, small), such as (1,9,3)

sys.modules

Dictionary of the loaded module. It may not contain built-in modules

sys.path

A mutable list of directories to search for imported modules

sys.platform

Return the information of the current platform.

sys.stderr

Standard Error Stream

sys.stdin

Standard input stream

sys.stdout

standard output stream

sys.version

Python language version that this implementation conforms to, as a string

sys.version_info

Python language version that this implementation conforms to, as a tuple of ints.

Example:

```
>>>import sys
>>>sys .version'3
.4.0'
>>>sys .version_info
```

```
(3, 4, 0)
>>>sys .path
['','/ libs/mpy/']
>>>sys .__name__
sys'
>>>sys .platform'
Rt-Thread'
>>>sys .byteorder'little
'
```

more information refer to sys

3.6 ubinascii

ubinascii module implements many methods of converting between binary and binary representations encoded by various ascii.

- **Function**

ubinascii.hexlify (data [, sep])

Convert binary data to a hexadecimal representation,return bytes string

Example:

```
>>>ubinascii .hexlify ('hello Rt-Thread')
b'68656c6c6f2052542d546872656164'
>>>ubinascii .hexlify ('summer')
b'73756d6d6572'
```

If additional argument, sep is supplied, it is used as a separator between hexadecimal values.

Example:

```
>>>ubinascii .hexlify ('hello Rt-Thread','')
b'68 65 c 6c 6f 20 6 52 54 d 54 2 68 72 65 61
64'
>>>ubinascii .hexlify ('hello Rt-Thread',',')
```

ubinascii.unhexlify (data)

Convert hexadecimal data to binary representation and return bytes string. Example:

```
>>>ubinascii .unhexlify ('73756d6d6572')
b'summer'
```

ubinascii.a2b_base64(data)

Decode base64-encoded data, ignoring invalid characters in the input. Conforms to RFC 2045 s.6.8. Returns a bytes object.

ubinascii.b2a_base64(data)

Encode binary data in base64 format, as in RFC 3548. Returns the encoded data followed by a newline character, as a bytes object.

More information refer to ubinascii

3.7 ucollections

ucollections module implements specialized container types that provide alternatives to Python generic built-in containers, including dictionaries, lists, collections, and tuples.

- **Function**

ucollections.namedtuple (name, fields)

This is factory function to create a new namedtuple type with a specific name and set of fields. A namedtuple is a subclass of tuple which allows to access its fields not just by numeric index, but also with an attribute access syntax using symbolic field names. Fields is a sequence of strings specifying field names. For compatibility with CPython it can also be a a string with space-separated field named (but this is less efficient).

example:

```
from ucollections import namedtuple

MyTuple =namedtuple ('MyTuple',('id','name')
MyTuple =namedtuple MyTuple (1,'foo')
t2=MyTuple (2,'bar')
print (t1.name)
assert t2.name ==t2[1]
```

ucollections.OrderedDict(…)

dict type subclass which remembers and preserves the order of keys added. When ordered dict is iterated over, keys/items are returned in the order they were added:

```
from ucollections import OrderedDict

# To make benefit of ordered keys,OrderedDict should be initialized
#from sequence of (key, value) pairs.
d =OrderedDict ([('z',1,('a',2)])
# More items can be added d as usual
d['w']5
d['b']=3
for k,v in d.items ():
    print (k, v)
```

Output:

```
z 1a 2w 5b 3
```

More information refer to ucollections

3.8 uerrno

uerrno module implements a standard errno system symbol, each symbol has a corresponding integer value.

Example:

```
try :
    uos .mkdir ('my_dir')
except OSError as exc :
    if exc.args [0]==uerrno.EEXIST :
        print ('Directory already
exists')
```

uerrno. error code

Dictionary mapping numeric error codes to strings with symbolic error code (see above):

```
>>>print (uerrno. error code
[uerrno. EEXIST])
EEXIST
```

more information refer to uerrno

3.9 uhashlib

uhashlib module implements binary data hash algorithm.

- **Algorithm Function**

SHA256

The current generation, modern hashing algorithm (of SHA2 series). It is suitable for cryptographically-secure purposes. Included in the MicroPython core and any board is recommended to provide this, unless it has particular code size constraints.

SHA1

A previous generation algorithm. Not recommended for new usages, but SHA1 is a part of number of Internet standards and existing applications, so boards targeting network connectivity and interoperability will try to provide this.

MD5

A legacy algorithm, not considered cryptographically secure. Only selected boards, targeting interoperability with legacy applications, will offer this.

- **Constructor**

class uhashlib.sha256([data])

Create an SHA256 hasher object and optionally feed data into it.

class uhashlib.sha1([data])

Create an SHA1 hasher object and optionally feed data into it.

class uhashlib.md5([data])

Create an MD5 hasher object and optionally feed data into it.

- **Function**

hash.update (data)

Feed more binary data into hash

hash.digest()

Return hash for all data passed through hash, as a bytes object. After this method is called, more data cannot be fed into the hash any longer.

hash.hexdigest()

This method is not implemented, using ubinascii.hexlify (hash.digest () to achieve similar effect more information refer to uhashlib

3.10 uheapq

This module implements the heap queue algorithm.

- **Function**

uheapq.heappush (heap, item)

Push the item onto the heap

uheapq.heappop (heap)

pop up the first element from the heap and return it. Raises IndexError if heap is empty.

uheapq.heapify (x)

Convert the list x into a heap. This is an in-place operation.

more information refer to uheapq

3.11 uio

This module contains additional types of stream (file-like) objects and helper functions.

- **Function**

uio.open (name, mode =' r',kwargs)

Open a file. Builtin open() function is aliased to this function. All ports (which provide access to file system) are required to support mode parameter, but support for other arguments vary by port.

- **Class**

class uio.FileIO(…)

This is type of a file open in binary mode, e.g. using open(name, "rb"). You should not instantiate this class directly.

class uio.TextIOWrapper(…)

This is type of a file open in text mode, e.g. using open(name, "rt"). You should not instantiate this class directly.

class uio.StringIO ([string])

class uio.BytesIO ([string])

In-memory file-like objects for input/output. StringIO is used for text-mode I/O (similar to a normal file opened with “t” modifier). BytesIO is used for binary-mode I/O (similar to a normal file opened with “b” modifier). Initial contents of file-like objects can be specified with stringparameter

(should be normal string for StringIO or bytes object for BytesIO). All the usual file methods like read(), write(), seek(), flush(), close() are available on these objects, and additionally, a following method:

- getvalue()

  Get the current contents of the underlying buffer which holds data.

  more information refer to uio

3.12 ujson

This modules allows to convert between Python objects and the JSON data format.

- **Function**

ujson.dumps (obj)

Return obj represented as a JSON string.

Example:

```
>>>obj ={1:2,3:4," a": 6}
>>>print (type (obj), obj)  # used to be dict type
<class'dict'>{3:4,1:2} a': 6}
>>>jsObj =json .dumps (obj)  # Converts a dict type to a str
>>>print (type (jsObj), jsObj)
<class'str'>{3:4,1:2," a": 6}
```

ujson.loads (str)

Parse the JSON str and return an object. Raises ValueError if the string is not correctly formed.

Example:

```
>>>obj ={1:2,3:4," a": 6}
>>>jsDumps =json .dumps (obj)
>>>jsLoads =json .loads (jsDums)
>>>print (type (obj), obj)
<class'dict'>{3:4,1:2} a': 6}
```

more information refer to ujson

3.13 uos

The uos module contains functions for filesystem access and mounting, terminal redirection and duplication, and the uname and urandom functions.

- **Function**

  uos.chdir (path)

  Change the current directory

  uos.getcwd()

  Gets the current directory.

  uos.listdir ([dir])

  Without argument, list the current directory. Otherwise list the given directory.

  uos.mkdir (path)

  Create a new directory.

  uos.remove (path)

  Remove a file

  uos.rmdir (path)

  Remove a directory.

  uos.rename (old_path, new_path)

  Rename files or folders.

  uos.stat (path)

  Gets the status of a file or directory.

  uos.sync()

  Synchronize all file systems.

  Example:

```
>>> import uos
>>> uos.                        # Tab
__name__        uname           chdir           getcwd
listdir         mkdir           remove          rmdir
stat            unlink          mount           umount
>>> uos.mkdir("rtthread")
>>> uos.getcwd()
'/'
>>> uos.chdir("rtthread")
>>> uos.getcwd()
'/rtthread'
>>> uos.listdir()
['web_root', 'rtthread', '11']
>>> uos.rmdir("11")
>>> uos.listdir()
['web_root', 'rtthread']
>>>
```

more information refer to uos

3.14 ure

ure module is used to test a pattern of strings and perform regular expression operations.

-- Matches any character: '.'

-- Matches a collection of characters, supporting a single character and a range: '[]'

-- Support for multiple matching metacharacters: **‘ ^ ’ ‘ $ ’ ‘ ? ’ ‘*’ ‘+’ ‘? ?’ ‘ *?’ ‘+?’ ‘{m,n}’**

- **Function**

ure.compile (regex)

Compile regular expressions and return regex objects.

ure.match (regex, string)

Compile regex and match against string. Match always happens from starting position in a string.

ure.search (regex, string)

Compile regex and search it in a string. Unlike match, this will search string for first position which matches regex (which still may be 0 if regex is anchored).

ure.DEBUG

Flag value, display debug information about compiled expression.

--Regular object:

Compiled regular expression. Instances of this class are created using ure.compile()

regex.match (string)

regex.search (string)

regex.split (string, max_split =-1)

--Matching objects:

Match objects as returned by match() and search() methods.

match.group ([index])

Return matching (sub)string. index is 0 for entire match, 1 and above for each capturing group. Only numeric groups are supported.

more information refer to ure

3.15 uselect

This module provides functions to efficiently wait for events on multiple streams (select streams which are ready for operations).

- **Constant**

  uselect.POLLIN

  data available for reading

  uselect.POLLOUT

  more data can be written

  uselect.POLLERR

  Error happened

  uselect.POLLHUP

  Stream-Ending / Connection Termination Detection

- **Function**

  uselect.select (rlist, wlist, xlist [, timeout])

  Monitor when the object is readable or writable and return the result once the monitored object state changes (blocking thread). This method is for compatibility, efficiency is not high, recommended poll method.

  -- rlist： file descriptor array waiting for read

  -- wlist： file descriptor array waiting to be written xlist： array waiting for exceptions

  -- timeout： waiting time (in seconds)

  Example:

```
def selectTest ():
  global s
  rs ,ws ,es =uselect .select ([s ,],[],[])
  # The program waits until the object s readable
  print
  (rs) for
  i in rs:
    if i ==s :
      print (" can read now ")
      data, addr =s.recvfrom (1024)print
      ('received :', data ,'from', addr)
```

- **Poll category**

uselect.poll()

Create poll instances. Example:

```
>>>poller =uselect .poll ()
>>>print (poller)
<poll >
```

poll.register (obj [, eventmask])

Register an object for monitoring and set the monitoring flag bit eventmask. of the monitored object

--obj： object monitored

--eventmask：sign monitored

- ✓ uselect.POLLIN — readable
- ✓ uselect.POLLHUP — hang up
- ✓ uselect.POLLERR — error
- ✓ uselect.POLLOUT — written

poll.unregister (obj)

Unregister obj from polling.

--obj: objects registered

Examples:

```
>>>READ_ONLY =uselect .POLLIN |select .POLLHUP |select .POLLERR
>>>READ_WRITE =uselect .POLLOUT |READ_ONLY
>>>poller.register (s, READ_WRITE)
>>>poller.unregister (s)
```

poll.modify (obj, eventmask)

Modify the registered object monitoring flag.

--obj： registered monitored objects

--eventmask: monitoring flags eventmask modified

Examples :

```
>>>READ_ONLY =uselect.POLLIN | uselect .POLLHUP | uselect .POLLERR
>>>READ_WRITE =uselect.POLLOUT |READ_ONLY
>>>poller.register (s, READ_WRITE)
>>>poller.modify (s, READ_ONLY)
```

poll.poll ([timeout])

Wait for at least one of the registered objects to become ready or have an exceptional condition, with optional timeout in milliseconds (if timeout arg is not specified or -1, there is no timeout).Returns list of (obj, event, …) tuples. There may be other elements in tuple, depending on a platform and version, so don't assume that its size is 2. The event element specifies which events happened with a stream and is a combination of uselect.POLL* constants described above. Note that flags uselect.POLLHUP and uselect.POLLERR can be returned at any time (even if were not asked

for), and must be acted on accordingly (the corresponding stream unregistered from poll and likely closed), because otherwise all further invocations of poll() may return immediately with these flags set for this stream again. In case of timeout, an empty list is returned.

more information refer to uselect

3.16 usocket

usocket module provides access to the BSD socket interface.

- **Constant**

Address family

- ✓ usocket.AF_INET =2 --TCP/IP – IPv4
- ✓ usocket.AF_INET6=10 --TCP/IP –IPv6

Socket type

- ✓ usocket.SOCK_STREAM =1 -- TCP flow
- ✓ usocket.SOCK_DGRAM =2 -- UDP Datagram
- ✓ usocket.SOCK_RAW =3 -- Original socket
- ✓ usocket.SO_REUSEADDR =4 -- socket reusable

IP Agreement Number

- ✓ usocket.IPPROTO_TCP =16
- ✓ usocket.IPPROTO_UDP =17

Socket option level

- ✓ usocket.SOL_SOCKET =409

● **Function**

usocket.socket

```
usocket .socket (usocket.AF_INET, usocket.SOCK_STREAM, usocket.IPPROTO_TCP)
```

Create a new socket using the specified address, type, and protocol number.

usocket.getaddrinfo (host, port)

Convert host domain name (host) and port (port) to a 5-tuples for creating sockets. The tuple list is structured as follows:

```
(family ,type ,proto ,canonname ,sockaddr)
```

Example:

```
>>>info =usocket.getaddrinfo (" Rt-Thread.org ",10000)
>>>print (info)
[(2, 1, 0, '', ('118.31.15.152', 10000))]
```

usocket.close()

Mark the socket closed and release all resources. Once that happens, all future operations on the socket object will fail. The remote end will receive EOF indication if supported by protocol.

Sockets are automatically closed when they are garbage-collected, but it is recommended to close() them explicitly as soon you finished working with them.

usocket.bind (address)

Bind the socket to address. The socket must not already be bound.

usocket.listen ([backlog])

Enable a server to accept connections. If backlog is specified, it must be at least 0 (if it's lower, it will be set to 0); and specifies the number of unaccepted connections that the system will allow before refusing new connections. If not specified, a default reasonable value is chosen.

--backlog： accept the maximum number of sockets, at least 0, if not specified, a reasonable value is default.

usocket.accept()

Accept a connection. The socket must be bound to an address and listening for connections. The return value is a pair (conn, address) where conn is a new socket object usable to send and receive data on the connection, and address is the address bound to the socket on the other end of the connection.

-- conn： a new socket object that can be used to send and receive messages

-- address：client address connected to the server

usocket.connect (address)

Connect to a remote socket at address.

-- address: tuples or lists of server addresses and port numbers

usocket.send (bytes)

Send data to the socket. The socket must be connected to a remote socket. Returns number of bytes sent, which may be smaller than the length of data ("short write").

-- bytes：bytes types of data

usocket.recv (bufsize)

Receive data from the socket. The return value is a bytes object representing the data received.

The maximum amount of data to be received at once is specified by bufsize.

-- bufsize: specify the maximum amount of data received once

Example:

```
data =conn .recv (1024)
```

usocket.sendto (bytes, address)

Send data, the target is determined by the address, often used to UDP communication, return the sent data size.

-- bytes：bytes types of data

-- address： tuples of destination address and port number

Example:

```
data =sendto (" hello Rt-Thread ",("192.168.10.110",100))
```

usocket.recvfrom (bufsize)

Receive data from the socket. The return value is a pair (bytes, address) where bytes is a bytes object representing the data received and address is the address of the socket sending the data.

-- bufsize： specify the maximum amount of data received once

Example:

```
data,addr =fd.recvfrom (1024)
```

usocket.setsockopt (level, optname, value)

Set the value of the given socket option. The needed symbolic constants are defined in the socket module (SO_* etc.). The value can be an integer or a bytes-like object representing a buffer.

-- level： socket option level

-- optname： socket options

-- value： can be an integer or a bytes class object representing a buffer.

Example:

```
s.setsockopt (usocket.SOL_SOCKET, usocket.SO_REUSEADDR ,1)
```

usocket.settimeout (value)

Set a timeout on blocking socket operations. The value argument can be a nonnegative floating point number expressing seconds, or None. If a non-zero value is given, subsequent socket operations will raise an OSError exception if the timeout period value has elapsed before the operation has completed. If zero is given, the socket is put in non-blocking mode. If None is given, the socket is put in blocking mode.

Not every MicroPython port supports this method. A more portable and generic solution is to use uselect.poll object. This allows to wait on multiple objects at the same time (and not just on sockets,

but on generic stream objects which support polling).

Example:

```
s.settimeout(2)
usocket.setblocking (flag)
```

Set blocking or non-blocking mode: if flag is false, set non-blocking mode.

usocket.read ([size])

Read up to size bytes from the socket. Return a bytes object. If size is not given, it reads all data available from the socket until EOF; as such the method will not return until the socket is closed. This function tries to read as much data as requested (no “short reads”). This may be not possible with nonblocking socket though, and then less data will be returned.

usocket.readinto (buf [, nbytes])

Read bytes into the buf. If nbytes is specified then read at most that many bytes. Otherwise, read at most len(buf) bytes. Just as read(), this method follows “no short reads” policy. Return value: number of bytes read and stored into buf.

usocket.readline()

Read a line, ending in a newline character.

Return value: the line read.

usocket.write (buf)

Write the buffer of bytes to the socket. This function will try to write all data to a socket (no “short writes”). This may be not possible with a non-blocking socket though, and returned value will be less than the length of buf.

Return value: number of bytes written.

Example: TCP Server

```
>>>import usocket
>>>s=usocket.socket (usocket.AF_INET, usocket.SOCK_STREAM)    # Create STREAM TCP
    socket
>>>s.bind (('192.168.12.32',6001))
>>>s.listen (5)
>>>s.setblocking (True)
>>>sock,addr =s.accept ()
>>>sock.recv (10)
b'Rt-Thread \r'
>>>s.close ()
TCP Client example
>>>import usocket
>>>s=usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
>>>s.connect (("192.168.10.110",6000))
>>>s.send (" MicroPython
")11
>>>s .close ()
```

Connecting to a web server

```
s=usocket.socket ()
s.connect (usocket.getaddrinfo ('www.MicroPython.org',80)[0][1])
```

more information refer to usocket

3.17 ussl

This module provides access to transport layer security (formerly known as the "secure socket layer "), as well as peer authentication for network sockets (client and server). This module provides access to security.

- **Function**

ussl.wrap_socket

ussl.wrap_socket (lock, server_side =False,key=None,cert=None)

Takes a stream sock (usually usocket.socket instance of SOCK_STREAM type), and returns an instance of ssl.SSLSocket, which wraps the underlying stream in an SSL context. Returned object has the usual stream interface methods like read(), write(), etc. In MicroPython, the returned object does not expose socket interface and methods like recv(), send(). In particular, a server-side SSL socket should be created from a normal socket returned fromaccept() on a non-SSL listening server socket. Depending on the underlying module implementation in a particular MicroPython port, some or all keyword arguments above may be not supported.

Warning: Some implementations of ussl module do NOT validate server certificates, which makes an SSL connection established prone to man-in-the-middle attacks.

- **Exception**

ssl.SSLError

This exception does NOT exist. Instead its base class, OSError, is used.

- **Constant**

ussl.CERT_NONE

ussl.CERT_OPTIONAL

ussl.CERT_REQUIRED

Support values for cert_reqs parameters

more information refer to ussl

3.18 ustruct

This module implements a subset of the corresponding CPython module, as described below. For more information, refer to the original CPython documentation: struct.

--Supported size/byte order prefixes: @, <, >, !.

--Supported format codes: b, B, h, H, i, I, l, L, q, Q, s, P, f, d (the latter 2 depending on the floating-point support).

- **Function**

ustruct.calcsize (fmt)

Return the number of bytes required fmt store a certain type of data.

```
fmt ：  data type
    b — bytes
    B — unsigned bytes
    h — short integer
    H — unsigned short integer
    i — integer
    I — unsigned integer
    l — integer
    L — unsigned integer
    q — long integral
    Q — unsigned long integer
    f — floating point
    d — Double Precision Floating Point
    P — unsigned
```

Example:

```
>>>print (struct.calcsize (" i ")
4
>>>print (struct.calcsize (" B ")
1
```

ustruct.pack (fmt, v1, v2,...)

Pack the values v1, v2, … according to the format string fmt. The return value is a bytes object encoding the values.

```
fmt ：  Ibid
```

Example:

```
>>>struct .pack ("ii",3,2' )
b'\\x03\\x00\\x00\\x00\\x02\\x00\\x0
```

ustruct.unpack (fmt, data)

Unpack from the data according to the format string fmt. The return value is a tuple of the

unpacked values.

```
data : byte object to extract
```

Example:

```
>>>buf =struct .pack (" bb ",1,2)
>>>print (buf)
b'\x01\x02'
>>>print (struct. unpack (" bb ", buf))
(1,2)
```

ustruct.pack_into (fmt, buffer, offset, v1, v2,...)

Pack the values v1, v2, … according to the format string fmt into a buffer starting at offset. offsetmay be negative to count from the end of buffer.

ustruct.unpack_from (fmt, data, offset =0)

Unpack from the data starting at offset according to the format string fmt. offset may be negative to count from the end of buffer. The return value is a tuple of the unpacked values.

```
>>>buf =struct .pack (" bb ",1,2)
>>>print (struct. unpack (" bb ", buf))
(1,2)
>>>print (struct.unpack_from (" b ", buf ,1))
(2,)
```

more information refer to ustruct

3.19 utime

The utime module provides functions for getting the current time and date, measuring time intervals, and for delays.

Time Epoch: Unix port uses standard for POSIX systems epoch of 1970-01-01 00:00:00 UTC. However, embedded ports use epoch of 2000-01-01 00:00:00 UTC.

Maintaining actual calendar date/time: This requires a Real Time Clock (RTC). On systems with underlying OS (including some RTOS), an RTC may be implicit. Setting and maintaining actual calendar time is responsibility of OS/RTOS and is done outside of MicroPython, it just uses OS API to query date/time. On baremetal ports however system time depends on machine.RTC() object. The current calendar time may be set using machine.RTC().datetime(tuple) function, and maintained by following means:

-- By a backup battery (which may be an additional, optional component for a particular board).

-- Using networked time protocol (requires setup by a port/user).

-- Set manually by a user on each power-up (many boards then maintain RTC time across hard resets, though some may require setting it again in such case).

If actual calendar time is not maintained with a system/MicroPython RTC, functions below which require reference to current absolute time may behave not as expected.

- **Function**

utime.localtime ([secs])

Convert a time expressed in seconds since the Epoch (see above) into an 8-tuple which contains: (year, month, mday, hour, minute, second, weekday, yearday) If secs is not provided or None, then the current time from the RTC is used.

-- year includes the century (e.g .2014).

-- month is 1-12

-- mday is 1-31

-- hour is 0-23

-- minute is 0-59

-- second is 0-59

-- weekday is 0-6 for Mon to Sun

-- yearday is 1-366

utime.mktime()

This is inverse function of localtime. It's argument is a full 8-tuple which expresses a time as per localtime. It returns an integer which is the number of seconds since Jan 1, 2000.

utime.sleep (seconds)

Sleep for the given number of seconds. Some boards may accept seconds as a floating-point number to sleep for a fractional number of seconds. Note that other boards may not accept a floating-point argument, for compatibility with them use sleep_ms() and sleep_us() functions.

utime.sleep_ms (ms)

Delay for given number of milliseconds, should be positive or 0.

utime.sleep_us (us)

Delay for given number of microseconds, should be positive or 0.

utime.ticks_ms()

Returns an increasing millisecond counter with an arbitrary reference point, that wraps around after some value.

The wrap-around value is not explicitly exposed, but we will refer to it as TICKS_MAX to simplify discussion. Period of the values is TICKS_PERIOD = TICKS_MAX + 1. TICKS_PERIOD is guaranteed to be a power of two, but otherwise may differ from port to port. The same period value is used for all of ticks_ms(), ticks_us(), ticks_cpu() functions (for simplicity). Thus, these functions will return a value in range [0 .. TICKS_MAX], inclusive, total TICKS_PERIODvalues. Note that only non-negative values are used. For the most part, you should treat values returned by these functions as opaque. The only operations available for them are ticks_diff()and ticks_add() functions described below.

Note: Performing standard mathematical operations (+, -) or relational operators (<, <=, >, >=) directly on these value will lead to invalid result. Performing mathematical operations and then passing their results as arguments to ticks_diff() or ticks_add() will also lead to invalid results from the latter functions.

utime.ticks_us()

Just like ticks_ms() above, but in microseconds.

utime.ticks_cpu()

Similar to ticks_ms() and ticks_us(), but with the highest possible resolution in the system. This is usually CPU clocks, and that's why the function is named that way. But it doesn't have to be a CPU clock, some other timing source available in a system (e.g. high-resolution timer) can be used instead. The exact timing unit (resolution) of this function is not specified on utimemodule level, but documentation for a specific port may provide more specific information. This function is intended for very fine benchmarking or very tight real-time loops. Avoid using it in portable code.

Availability: Not every port implements this function.

utime.ticks_add (ticks, delta)

Offset ticks value by a given number, which can be either positive or negative. Given a ticksvalue, this function allows to calculate ticks value delta ticks before or after it, following modular-arithmetic definition of tick values (see ticks_ms() above). ticks parameter must be a direct result of call to ticks_ms(), ticks_us(), or ticks_cpu() functions (or from previous call to ticks_add()). However, delta can be an arbitrary integer number or numeric expression. ticks_add() is useful for calculating deadlines for events/tasks. (Note: you must use ticks_diff() function to work with deadlines.)

Code example:

```
## search beat values before 100 ms
print (utime.ticks_add (utime.ticks_ms(),-100))
## Calculate the cut-off time for the operation and then test it
deadline =utime.ticks_add (utime.ticks_ms(),200)
while utime.ticks_diff (deadline, utime.ticks_ms ())>0:
    do_a_little_of_something ()
## Find the maximum value of this transplant beat
print (time.ticks_add (0,-1))
```

utime.ticks_diff (ticks1, ticks2)

Measure ticks difference between values returned from ticks_ms(), ticks_us(), or ticks_cpu()functions, as a signed value which may wrap around.

Code example:

```
## Wait for GPIO pin to be valid, but at most 500 us
start =utime.ticks_us ()
while pin .value ()==0:
    if utime.ticks_diff(utime.ticks_us(),start )>500:
        raise TimeoutError
```

utime.time()

Returns the number of seconds, as an integer, since the Epoch, assuming that underlying RTC is set and maintained as described above. If an RTC is not set, this function returns number of seconds since a port-specific reference point in time (for embedded boards without a battery-backed RTC, usually since power up or reset). If you want to develop portable MicroPython application, you should not rely on this function to provide higher than second precision. If you need higher precision, use ticks_ms() and ticks_us() functions, if you need calendar time,localtime() without an argument is a better choice.

Difference to CPython

In CPython, this function returns number of seconds since Unix epoch, 1970-01-01 00:00 UTC, as a floating-point, usually having microsecond precision. With MicroPython, only Unix port uses the same Epoch, and if floating-point precision allows, returns sub-second precision. Embedded hardware usually doesn't have floating-point precision to represent both long time ranges and subsecond precision, so they use integer value with second precision. Some embedded hardware also lacks battery-powered RTC, so returns number of seconds since last power-up or from other relative, hardware-specific point (e.g. reset).

Example:

```
>>>import utime
>>>utime.sleep (1)              # sleep for 1second
>>>utime.sleep_ms (500)         # sleep for 500milliseconds
>>>utime.sleep_us (10)          #sleep for 10microseconds

>>>start=utime .ticks_ms ()# get value of millisecond counter
>>>delta=utime .ticks_diff ((), start )# compute time difference
>>>delta
6928
>>>print(utime.ticks_add (utime.ticks_ms (),-100))
1140718
>>>print(utime.ticks_add (0,-1))
1073741823
```

more information refer to utime

3.20 uzlib

This module allows to decompress binary data compressed with DEFLATE algorithm (commonly used in zlib library and gzip archiver). Compression is not yet implemented.

- **Method**

uzlib.decompress (data)

Return decompressed data as bytes. wbits is DEFLATE dictionary window size used during compression (8-15, the dictionary size is power of 2 of that value). Additionally, if value is positive, data is assumed to be zlib stream (with zlib header). Otherwise, if it's negative, it's assumed to be raw DEFLATE stream. bufsize parameter is for compatibility with CPython and is ignored.

more information refer to uzlib

3.21 _thread

The thread module provides a basic way to handle multithreading — multiple control threads share their global data space. To achieve synchronization, simple locks are provided (known as mutexes or binary semaphores).

Example:

```
import _thread
import utime
def testThread ():
    while True :
        print (" Hello from thread ")
        utime.sleep (2)
_thread.start_new_thread (testThread ,())
while True :
    pass
```

Output (start new thread, print characters every two seconds):

```
Hello from thread Hello from thread Hello from thread Hello from thread Hello from thread
```

more information, refer to _thread

4. MicroPython specific libraries

4.1 machine

Most functions in this module allow to achieve direct and unrestricted access to and control of hardware blocks on a system (like CPU, timers, buses, etc.). Used incorrectly, this can lead to malfunction, lockups, crashes of your board, and in extreme cases, hardware damage.

However, it is not the same way that the resource can be controlled by MicroPython for the different hardware resources on the different development boards. Therefore, before using, the configuration needs to be modified to fit different development boards, Or directly run MicroPython sample programs fitted to a development board.

- **Function**

1) Reset function

machine.info()

Return information about system introduction and memory usage.

machine.reset() Note: Not yet achieved

Restart the device in a manner similar to pushing the external reset button.

machine.reset_cause() Note: Not yet achieved

Return the reason for the reset and check the constants of possible return values.

2) Interrupt function

machine.disable_irq()

Disable interrupt requests. Returns the previous IRQ state which should be considered an opaque value. This return value should be passed to the enable_irq() function to restore interrupts to their original state, before disable_irq() was called.

machine.enable_irq (state)

Re-enable interrupt requests. The state parameter should be the value that was returned from the most recent call to the disable_irq() function.

3) Power Function

machine.freq()

Return CPU frequency in hertz.

machine.idle() Note: Not yet achieved

Gates the clock to the CPU, useful to reduce power consumption at any time during short or long periods. Peripherals continue working and execution resumes as soon as any interrupt is triggered (on many ports this includes system timer interrupt occurring at regular intervals on the order of millisecond).

machine.sleep() Note: Not yet achieved

Stop CPU running and prohibit all peripherals except WLAN. To ensure that wake-up does occur, the interrupt source should be configured first.

machine.deepsleep() Note: Not yet achieved

Stops execution in an attempt to enter a low power state.

If time_ms is specified then this will be the maximum time in milliseconds that the sleep will last for. Otherwise the sleep can last indefinitely.

With or without a timout, execution may resume at any time if there are events that require processing. Such events, or wake sources, should be configured before sleeping, like Pin change or RTC timeout.

4.2 machine.Pin

machine.Pin class is a hardware handling class in the machine module, which is used to configure and control pins and provide operation methods for pin devices.

Pin object is used to control input/output pins (also called GPIO), which is usually associated with a physical pin that can drive the output voltage and read the input voltage. A pin class has some methods to set pin modes (input/output) in Pin class, and some methods to get and set digital logic (0 or 1).

A Pin object is constructed by an identifier, which specifies a specific input and output. The mapping between form of identifiers and physical pins is for a single porting. Identifiers can be integer, string, or a tuple with a port and a pin number. A pin identifier in the K210-MicroPython is a tuple consisting of a code name and a pin number, such as Pin (" PB15",31"), Pin. OUT_PP).

- **Constructor**

The constructor of the Pin object in the Rt-Thread-MicroPython is as follows:

class machine.Pin (id, mode =-1, pull =-1, value)

--id： consisting of user-defined pin names and pin numbers of pin device, such as (" PB15, ", 31), " PB15" for the user-defined pin name, and 31 for the pin number driven by the Rt-Thread Pin device in this transplant.

--mode： specify pin mode, it can be:

- ✓ Pin.IN： Input Mode
- ✓ Pin.OUT_PP： output mode
- ✓ Pin.OUT_OD： open and drain mode

-- pull： if the specified pin is connected to the pull-down resistor, it can be configured as follows:

- ✓ None： no pull-up or pull-down resistance.
- ✓ Pin.PULL_UP： enable pull-up resistance.
- ✓ Pin.PULL_DOWN： enable pull-down resistance.

-- value： The value is valid only for output mode and open-drain output mode and is used to set initial output values.

- **Method**

Pin.init (mode =-1, pull=-1,*, value,drive,alt)

According to the input parameters to reinitialize the pin, which only specified will be set. Detailed parameters can refer to the constructor above.

Pin.value ([x])

Without given parameter x, this method can get the value of the pin. If a parameter x is given, such as 0 or 1, then the value of the pin can be set to logic 0 or logic 1.

Pin.name()

Return the pin name user-defined

Pin.irq (handler =None,trigger=(Pin.IRQ_RISING))

Configure an interrupt handler to be called when the trigger source of the pin is active. If the pin mode is Pin.IN then the trigger source is the external value on the pin. If the pin mode is Pin.OUT then the trigger source is the output buffer of the pin. Otherwise, if the pin mode is Pin.OPEN_DRAIN then the trigger source is the output buffer for state '0' and the external pin value for state '1'.

Parameters:

-- handler: an optional function, which is called when an interruption is triggered

-- trigger: configurate the event that can generate an interruption. Possible values are:

- ✓ Pin.IRQ_FALLING: interrupt on falling edge
- ✓ Pin.IRQ_RISING: interrupt on rising edge
- ✓ Pin.IRQ_RISING_FALLING: interrupt on rising or falling edge
- ✓ Pin.IRQ_LOW_LEVEL: interrupt on low level
- ✓ Pin.IRQ_HIGH_LEVEL: interrupt on high level

- **Constants**

Below constants are used to configure Pin objects.

1） Select pin mode:

- ✓ Pin.IN
- ✓ Pin.OUT_PP
- ✓ Pin.OUT_OD

2） Select up/down mode:

- ✓ Pin.PULL_UP
- ✓ Pin.PULL_DOWN
- ✓ None: None means not up and down

3） Select interruption trigger mode:

- ✓ Pin.IRQ_FALLING
- ✓ Pin.IRQ_RISING
- ✓ Pin.IRQ_RISING_FALLING
- ✓ Pin.IRQ_LOW_LEVEL
- ✓ Pin.IRQ_HIGH_LEVEL

- **Example 1**

Control pin output for high or low level signal, and read key pin level signal.

```
from machine import Pin

PIN_OUT=31
PIN_IN=58
p_out=Pin (" PB15",PIN_OUT,Pin.OUT_PP)
p_out.value (1)                 # set io high
p_out.value (0)                 # set io low
p_in=Pin(" key_0", PIN_IN),Pin.IN, Pin.PULL_UP)
print(p_in.value ())            # get value, 0 or 1
```

- **Example 2**

```
from machine import Pin

PIN_KEY0=58        # PD10
key_0=Pin ("key_0", PIN_KEY0, Pin.IN, Pin.PULL_UP)
def func (v):
    print (" Hello Rt-Thread! ")
key_0.irq (trigger=Pin.IRQ_RISING ,handler=func)
```

The processing function of interruption is executed when pin interruption is triggered by a rising edge signal.

More information refer to pin

4.3 machine.UART

machine.UART class is a hardware handling class in the machine module for UART configuration and control, which provides operation methods for UART devices. UART implements the standard uart/usart duplex serial communication protocol, at the physical layer, which consists of two data channels: RX and TX. The communication block is a character, which can be 8 or 9 bit wide.

- **Constructor**

The constructor of the UART object in the Rt-Thread-MicroPython is as follows:

class machine.UART (id,...)

Construct a UART object on a given bus, id depending on the particular porting. Parameters initialization can refer to the UART.init method below.

Using hardware UART, only initializes the number of the UART device, such as 1 for uart1 device. Initialization can refer to the example.

- **Function**

UART.init (baudrate =9600, bits=8, parity=None,stop=1)

-- baudrate： the clock rate.

-- bits： the number of bits per character.

-- parity： parity. Including None, 0 (even) or 1 (odd).

-- stop： the number of the stop bits, 1 or 2

UART.deinit()

Turn off the UART bus.

UART.read ([nbytes])

Read characters. If nbytes is specified then read at most that many bytes, otherwise read as much data as possible. It may return sooner if a timeout is reached. The timeout is configurable in the constructor.

Return value: a bytes object containing the bytes read in. Returns None on timeout.

UART.readinto (buf [, nbytes])

Read bytes into the buf. If nbytes is specified then read at most that many bytes. Otherwise, read at most len(buf) bytes. It may return sooner if a timeout is reached. The timeout is configurable in the constructor.

Return value: number of bytes read and stored into buf or None on timeout.

UART.readline()

Read a line, ending in a newline character. It may return sooner if a timeout is reached. The timeout is configurable in the constructor.

Return value: the line read or None on timeout.

UART.write (buf)

Write the buffer of bytes to the bus.

Return value: number of bytes written or None on timeout.

Example:

When the first parameter of the constructor passes in 1, the system searches for a device named uart1 and uses it to build UART object:

```
from machine import UART
uart =UART (1,115200)                              # init with given baudrate
uart .init (115200, bits =8, parity =None ,stop =1) # init with given parameters
uart .read (10)      # read 10characters,returns a bytes object
uart .read ()        # read all available characters
uart .readline ()    # read a line
uart .readinto (buf)# read and store into the given buffer
uart .write ('abc') # write the 3characters
```

More information refer to uart

4.4 machine.SPI

machine.SPI class is a hardware handling class in the machine module, which is used to configure and control the SPI, which provides operation methods for the SPI device.

SPI is a synchronous serial protocol that is driven by a master. At the physical level, a bus consists of 3 lines: SCK, MOSI, MISO. Multiple devices can share the same bus. Each device should have a separate, 4th signal, SS (Slave Select), to select a particular device on a bus with which communication takes place. Management of an SS signal should happen in user code (via machine.Pin class).

- **Constructor**

The constructor of the SPI object in the Rt-Thread MicroPython is as follows:

class machine.SPI (id,...)

Construct an SPI object on the given bus, id. Values of id depend on a particular port and its hardware. Values 0, 1, etc. are commonly used to select hardware SPI block #0, #1, etc. Value -1 can be used for bitbanging (software) implementation of SPI (if supported by a port).

With no additional parameters, the SPI object is created but not initialised (it has the settings from the last initialisation of the bus, if any). If extra arguments are given, the bus is initialised. See init for parameters of initialisation.

- **Function**

SPI.init SPI.init (baudrate =1000000,*, polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck=None, mosi=None, miso=None)

Initialize the SPI bus with the given parameters:

-- baudrate： clock rate.

-- polarity： polarity can be 0 or 1, the level the idle clock line sits at.

-- phase： phase can be 0 or 1, samples data at the first or second clock edges, respectively.

-- bits： the width in bits of each transfer. Only 8 is guaranteed to be supported by all hardware.

-- firstbit： data can be SPI.MSB or SPI.LSB. from high or low

-- sck： machine.Pin objects for sck.

-- mosi： machine.Pin objects for mosi.

-- miso： machine.Pin objects for miso.

SPI.deinit()

Turn off the SPI bus.

SPI.read (nbytes, write =0x00)

Read a number of bytes specified by nbytes while continuously writing the single byte given by write. Returns a bytes object with the data that was read.

SPI.readinto (buf, write =0x00)

Read into the buffer specified by buf while continuously writing the single byte given by write. Returns None.

Note: on WiPy this function returns the number of bytes read.

SPI.write (buf)

write the bytes contained in the buffer. Return None.

SPI.write_readinto (write_buf, read_buf)

write data from the write_buf while reading data into read_buf. Buffers can be the same or different, but both buffers must have the same length. Return None.

- **Constant**

SPI.MSB

Set to transfer data from high position.

SPI.LSB

Set to transfer data from low position.

- **Example**

Software simulation SPI

```
>>>from machine import Pin ,SPI
>>>clk =Pin (" clk ",26), Pin. OUT_PP)
>>>mosi =Pin (" mosi ",27), Pin. OUT_PP)
>>>miso =Pin (" miso ",28, Pin. IN)
>>>spi =SPI (-1500000,=0, phase =0, bits =8, firstbit =0,
mosi =mosi ,miso =miso)
>>>print (spi)
SoftSPI (baudrate =500000, polarity =0, phase =0, sck =clk ,mosi =mosi ,miso =miso)
>>>spi .write (" hello RT-Thread! ")
>>>spi .read (10)b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

Hardware SPI

Before Using, the SPI device driver first needs to be turned on. For finding the device, enter **list_device** command in the msh. For example: When the first parameter of the constructor is 50, the system will search a device named spi50 and use it to build SPI object:

```
>>>from machine import SPI
>>>spi =SPI (50)
>>>print (spi)
SPI (device port: spi50)
>>>spi .write (b'\x9f')
>>>spi .read (5)
b'\xff\xff\xff\xff\xff'
>>> buf = bytearray(1)
>>> spi.write_readinto(b"\x9f",buf)
>>> buf
bytearray(b'\xef')
>>> spi.init(100000,0,0,8,1)     # Resetting SPI parameter
```

More information refer to spi

4.5 machine.I2C

machine.I2C class is a hardware handling class in the machine module, which is used to configure and control the I2C, which provides operation methods for the I2C device.

I2C is a two-wire protocol for communication between devices. At the physical layer, it consists of two lines: SCL and SDA, that are instant clocks and data lines.

I2C objects are created attached to a specific bus, which can be initialized when created or initialised later on. Printing the I2C object gives you information about its configuration.

- **Constructor**

The constructor of the I2C object in the Rt-Thread-MicroPython is as follows:

class machine.I2C (id =-1, scl,sda,freq=400000)

Construct and return a new I2C object using the following parameters:

--id ： identifies a particular I2C peripheral. The default value of -1 selects a software implementation of I2C which can work (in most cases) with arbitrary pins for SCL and SDA. If id is -1 then scl and sda must be specified. Other allowed values for id depend on the particular port/board, and specifying scl and sda may or may not be required or allowed in this case.

--scl： A Pin object, specified as a Pin object for scl.

--sda： A Pin object, specified as a Pin object for sda.

--freq：The maximum frequency setting for scl.

- **Function**

I2C.init (scl, sda, freq =400000)

Initialize the I2C bus, parameter introduction can refer to the parameters in the constructor.

I2C.deinit()

Turn off the I2C bus.

I2C.scan()

Scan all I2C addresses between 0x08 and 0x77 inclusive and return a list of those that respond. A device responds if it pulls the SDA line low after its address (including a write bit) is sent on the bus.

- **primitive function**

The following methods implement primitive I2C master bus operations, which can be combined to make any I2C transaction. They are provided if you need more control over the bus, otherwise the standard methods (see below) can be used..

I2C.start()

Generate a START condition on the bus (SDA transitions to low while SCL is high).

I2C.stop()

Generate a STOP condition on the bus (SDA transitions to high while SCL is high).

I2C.readinto (buf, nack =True)

Reads bytes from the bus and stores them into buf. The number of bytes read is the length of buf. An ACK will be sent on the bus after receiving all but the last byte. After the last byte is received, if nack is true then a NACK will be sent, otherwise an ACK will be sent (and in this case the slave assumes more bytes are going to be read in a later call).

I2C.write (buf)

Write the bytes from buf to the bus. Checks that an ACK is received after each byte and stops transmitting the remaining bytes if a NACK is received. The function returns the number of ACKs that were received.

- **standard bus operation**

the following methods implement the standard I2C master read and write operation that target a given slave device.

I2C.readfrom (addr, nbytes, stop =True)

Read nbytes from the slave specified by addr. If stop is true then a STOP condition is generated at the end of the transfer. Returns a bytes object with the data read.

I2C.readfrom_into (addr, buf, stop =True)

Read into buf from the slave specified by addr. The number of bytes read will be the length of buf. If stop is true then a STOP condition is generated at the end of the transfer.

The method returns None.

I2C.writeto (addr, buf, stop =True)

Write the bytes from buf to the slave specified by addr. If a NACK is received following the write of a byte from buf then the remaining bytes are not sent. If stop is true then a STOP condition is generated at the end of the transfer, even if a NACK is received. The function returns the number of ACKs that were received.

- **Memory operation**

Some I2C devices act as a memory device that can be read and written into. In this case there are two addresses associated with an I2C transaction: the slave address and the memory address. The following methods are convenience functions to communicate with such devices.

I2C.readfrom_mem (addr, memaddr, nbytes ,*, addrsize =8)

Read nbytes from the slave specified by addr starting from the memory address specified by memaddr. The argument addrsize specifies the address size in bits. Returns a bytes object with the

data read.

I2C.readfrom_mem_into (addr, memaddr, buf ,*, addrsize =8)

Read into buf from the slave specified by addr starting from the memory address specified by memaddr. The number of bytes read is the length of buf. The argument addrsize specifies the address size in bits (on ESP8266 this argument is not recognised and the address size is always 8 bits).

The method returns None.

I2C.writeto_mem (addr, memaddr, buf ,*, addrsize =8)

Write buf to the slave specified by addr starting from the memory address specified by memaddr. The argument addrsize specifies the address size in bits (on ESP8266 this argument is not recognised and the address size is always 8 bits).

The method returns None.

- **Example**

Software simulation I2C

```
>>>from machine import Pin ,I2C
>>>clk =Pin (" clk ",29), Pin. OUT_OD)    # Select the 29pin device as the clock
>>>sda =Pin (" sda ",30, Pin. OUT_OD)     # Select the 30pin device as the data line
>>>i2c =I2C (-1, clk, sda, freq =100000)# create I2C peripheral at frequency of 100kHz

>>>i2c .scan ()                          # scan for slaves,returning a list of 7-bit
    addresses
[81]                                     # Decimal representation
>>>i2c .writeto (0x51, b'123')3          # write 3bytes to slave with 7-bit address
>>>i2c .readfrom (0x51,4)
    42                                   42#read 4bytes from slave with 7-bit address
b'\xf8\xc0\xc0\xc0'
>>>i2c .readfrom_mem (0x51,0x02,1)
    ),                                   # read 1bytes from memory of slave 0x51:7-bit
b'\x12'
                                         # starting at memory-address 8in the slave
>>>i2c .writeto_mem (0x51,2, b'\x10')    # write1byte to memory of slave 42,
                                         # starting at address 2in the slave
```

Hardware I2C

Using it that needs to turn on the I2C device driver first, finding the device with entering list_device command in the msh. When the first parameter of the constructor passes in 0, the system searches for a device named i2c0 and uses it to build I2C object:

```
>>>from machine import Pin ,I2C
>>>i2c =I2C (0)                         # create I2C peripheral at frequency of 100kHz
>>>i2c .scan ()                         # scan for slaves,returning a list of 7-bit
addresses
[81]                                     # Decimal representation
```

More information refer to ISC

4.6 FPIOA

- **Constructor**

  class k210.FPIOA()

- **Function**

  bool set_function (int pin, int func, int set_sl, int set_st, int set_io_driving)

  Set pin function

  -- pin： pin number, range 0 to 48

  -- func： pin function, see attached F1 for value range

  - set_sl： (optional parameters)

  - set_st： (optional parameters)

  - set_io_driving： (optional parameters)

  int get_io_by_function (int func)

  View the pin of a function

  - The range of func： values is F1 attached

F1 :

```
JTAG_TCLK JTAG_TDI JTAG_TMS JTAG_TDO
SPI0_D0 SPI0_D1 SPI0_D2 SPI0_D3
SPI0_D4  SPI0_D5 SPI0_D6  SPI0_D7
SPI0_SS0 SPI0_SS1 SPI0_SS2 SPI0_SS3
SPI0_ARB SPI0_SCLK UARTHS_RX UARTHS_TX
CLK_SPI1 CLK_I2C1 GPIOHS0 GPIOHS1 GPIOHS2
GPIOHS3 GPIOHS4 GPIOHS5 GPIOHS6
GPIOHS7 GPIOHS8 GPIOHS9 GPIOHS10
GPIOHS11 GPIOHS12 GPIOHS13 GPIOHS14
GPIOHS15 GPIOHS16 GPIOHS17 GPIOHS18
GPIOHS19 GPIOHS20 GPIOHS21 GPIOHS22
GPIOHS23 GPIOHS24 GPIOHS25 GPIOHS26
GPIOHS27 GPIOHS28 GPIOHS29 GPIOHS30
GPIOHS31 GPIO0 GPIO1 GPIO2
GPIO3 GPIO4 GPIO5 GPIO6
GPIO7 UART1_RX UART1_TX UART2_RX
UART2_TX UART3_RX UART3_TX SPI1_D0
SPI1_D1 SPI1_D2 SPI1_D3 SPI1_D4
SPI1_D5 SPI1_D6 SPI1_D7 SPI1_SS0
SPI1_SS1 SPI1_SS2 SPI1_SS3 SPI1_ARB
SPI1_SCLK SPI_SLAVE_D0 SPI_SLAVE_SS SPI_SLAVE_SCLK
```

```
I2S0_MCLK I2S0_SCLK I2S0_WS I2S0_IN_D0
I2S0_IN_D1 I2S0_IN_D2 I2S0_IN_D3 I2S0_OUT_D0
I2S0_OUT_D1 I2S0_OUT_D2 I2S0_OUT_D3 I2S1_MCLK
I2S1_SCLK I2S1_WS I2S1_IN_D0 I2S1_IN_D1
I2S1_IN_D2 I2S1_IN_D3 I2S1_OUT_D0 I2S1_OUT_D1
I2S1_OUT_D2 I2S1_OUT_D3 I2S2_MCLK I2S2_SCLK
I2S2_WS I2S2_IN_D0 I2S2_IN_D1 I2S2_IN_D2
I2S2_IN_D3 I2S2_OUT_D0 I2S2_OUT_D1 I2S2_OUT_D2
I2S2_OUT_D3 I2C0_SCLK I2C0_SDA I2C1_SCLK
I2C1_SDA I2C2_SCLK I2C2_SDA CMOS_XCLK
CMOS_RST CMOS_PWDN CMOS_VSYNC CMOS_HREF
CMOS_PCLK CMOS_D0 CMOS_D1 CMOS_D2
CMOS_D3 CMOS_D4 CMOS_D5 CMOS_D6
CMOS_D7 SCCB_SCLK SCCB_SDA UART1_CTS
UART1_DSR UART1_DCD UART1_RI UART1_SIR_IN
UART1_DTR UART1_RTS UART1_OUT2 UART1_OUT1
UART1_SIR_OUT UART1_BAUD UART1_RE UART1_DE
UART1_RS485_EN UART2_CTS UART2_DSR UART2_DCD
UART2_RI
UART2_SIR_IN
```

```
UART2_DTR
UART2_RTS
UART2_OUT2
UART2_OUT1
UART2_SIR_OUT
UART2_BAUD
UART2_RE
UART2_DE
UART2_RS485_EN
UART3_CTS
UART3_DSR
UART3_DCD
UART3_RI
UART3_SIR_IN
UART3_DTR
UART3_RTS
UART3_OUT2
UART3_OUT1
UART3_SIR_OUT
UART3_BAUD
UART3_RE
UART3_DE
UART3_RS485_EN
TIMER0_TOGGLE1
TIMER0_TOGGLE2
TIMER0_TOGGLE3
TIMER0_TOGGLE4
TIMER1_TOGGLE1
TIMER1_TOGGLE2
```

```
TIMER1_TOGGLE3
TIMER1_TOGGLE4
TIMER2_TOGGLE1
TIMER2_TOGGLE2
TIMER2_TOGGLE3
TIMER2_TOGGLE4
CLK_SPI2
CLK_I2C2
```

- **Example**

```
from k210 import FPIOA

fpioa = FPIOA()
fpioa.set_function(1, fpioa.GPIOHS0)
fpioa.get_io_by_function(fpioa.GPIOHS0)
```

4.7 I2S

- **Constructor**

  class k210.I2S (int bus, int mode)

  -- bus： I2S bus number, range 0 to 2

  -- mode： receive/send mode, value [TRANSMITTER(play), RECEIVER(recording)]

- **Function**

  void init()

  Initialize I2S

  void set_param (int sample_rate, int bps, int track_num)

  Set parameters

  -- sample_rate： sampling rate

  -- bps： sampling bits

  -- track_num： number of channels ,1 or 2

  void play (bytearray pcm)

  Play sound

  - Audio data in pcm： PCM format

  bytearray record (void)

  Record sound

- **Example**

  Real-time playback of recordings

```
from k210 import I2S

i2sp = I2S(0, I2S.TRANSMITTER)
i2sp.init()
i2sp.set_param(8000, bps = 16, track_num = 2)

i2sr = I2S(1, I2S.RECEIVER)
i2sr.init()
i2sr.set_param(8000)

while (True):
pcm = i2sr.record()
i2sp.play(pcm)
```

  Play sound with the trigonometric function

```
import math
from k210 import I2S

sample_rate = 8000

i2s=I2S(0, I2S.TRANSMITTER)
i2s.init()
i2s.set_param(sample_rate, bps = 16, track_num = 1)

def sin_sound(freq, len, samrate):
pcm = bytearray()

step = (2 * math.pi * freq)/samrate

for i in range(0, len):
val = 5000 * math.sin(i * step)
iv = int(val)
pcm += iv.to_bytes(2, "little", True)


    return pcm
```

4.8 camera

- **Constructor**

  class k210.camera()

- **Function**

  void reset()

  Initialization

  void set_pixformat (int fmt)

  Set Image Format

  -- fmt： format, value [RGB565]

  void set_framesize (int width, int height)

  Set image size

  picture snapshot()

  Acquire single image

  return： picture object customized

- **Example**

```
from k210 import camera
from k210 import picture

cam = camera()

cam.reset()
cam.set_pixformat(cam.RGB565)
cam.set_framesize(320, 240)

for i in range(0, 1000):
img = cam.snapshot()
img.show()



cam.set_pixformat(cam.YUV422)
for i in range(0, 1000):
img = cam.snapshot()
img.show()
```

4.9 FFT

- **Constructor**

  class k210.FFT()

- **Function**

  list<int real,int imag>run (int [] input, int shift, int direction)

  Calculate fft

  Parameter Description:

  -- input： data to be transformed, type: int array

  -- shift： default value is 0

  -- direction： value [DIR_BACKWARD/DIR_FORWARD]

  Description of return value:

  -- return： return a list that contains real and virtual parts, which the virtual/real is type of int.

- **Example**

```
from k210 import FFT
import math

fft = FFT()
```

4.10 KPU

- **Constructor**

  class k210.KPU()

- **Function**

  load_kmodel (path, size)

  Load kmodel model

  Parameter Description:

  -- path： file system path or flash address, parameter type is string/int

  -- size： when path passes flash address, the model size needs to be Specified.

  run (input, dma)

  Run model reasoning

  Parameter Description:

  -- input： picture object or file path of omv

  -- dma： specify dma channel, the type is integer and default value is 5

  list get_output (index, getlist)

  Get the result of the model reasoning

  Parameter Description:

  -- index： the type is integer ,0 for all results (default); >0 for corresponding results

  -- getlist： the type is Boolean, Whether it returns result as list, default: FALSE

  Description of return value:

  -- return： list of reasoning results of models

  regionlayer_init (anchor_num), crood_num, landm_num, cls_num, in_w, in_h, obj_thresh, nms_thresh, variances, max_num, anchor)

  For related api of yolo, initialize related layers

parameters	description	Values(default)
anchor_num	–	3160
crood_num	–	4
landm_num	–	5
cls_num	–	1
in_w	–	320
in_h	–	240
obj_thresh	–	0.7
nms_thresh	–	0.4
variances	–	[0.1, 0.2]
max_num	–	200
anchor	–	See prior.h

regionlayer()

process yolo output

return： return coordinates of box boundary detected

- **Example 1: mnist handwritten digit recognition**

Mnist handwritten digit recognition program outputs 10 digit probabilities, and the result is the index with the largest probability value.

1) Model preparation

model：uint8_mnist_cnn_model.kmodel

mnist dataset：infer.bin

2) Example code

```
#demo : Mnist_cnn
import lcd,sensor,image,k210, gc
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
lcd.init()
m = k210.KPU()
m.load_kmodel("/uint8_mnist_cnn_model.kmodel")
# m.load_kmodel(0xb00000,540*1024) #load from flash
m.run("/infer.bin")
out=m.get_output(getlist=True)
out[0].index(max(out[0]))
```

- **Example 2: face dection**

The face detection demo, which implements the NMS processing of the prediction boundary box. The output result is the effective prediction frame coordinates of face detection;

1) Model preparation

model：ulffd_landmark.kmodel

2) Example code

```
#demo : face_landmark
import lcd,sensor,image,k210, gc
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
lcd.init()
m = k210.KPU()
m.load_kmodel("/ulffd_landmark.kmodel")
m.regionlayer_init(anchor_num=3160,crood_num=4,max_num=200)
m

while(True):
    gc.collect()
    img=sensor.snapshot()
    m.run(img)
    m.get_output(0)
    dect = m.regionlayer()
    print("dect:",dect)
    for l in dect :
        img.draw_rectangle(l[0],l[1],l[2]-l[0],l[3]-l[1])
    lcd.display(img)
```

4.11 machine.Timer

machine.Timer class is a hardware handling class in the machine module for configuration and control of the Timer equipment, which provides the operation methods for the Timer device.

-- Timer(Hardware Timer) is a device that used to handle periodic and timing events.

-- Timer hardware timer mainly counts the pulse signal through the internal counter module to implement the function of periodic equipment control.

-- Timer hardware timer can customize timeout and timeout-callback functions, provide two timer modes:

- ONE_SHOT： The timer runs once until the configured period of the channel expires.
- PERIOD：The timer runs periodically at the configured frequency of the channel.

-- printing Timer object can print out the configuration information.

- **Constructor**

The constructor of the Timer object in the Rt-Thread-MicroPython is as follows:

class machine.Timer (id)

Construct a new timer object of the given id. Id of -1 constructs a virtual timer (if supported by a board).

Parameter Description:

--id: The Timer device number used, for id =1 represents the Timer device number is 1, or the name of the timer device used, such as "timer" means the device named timer; this function is mainly used to create the device object through the device number.

- **Function**

Timer.init (mode =Timer.PERIODIC, period =0, callback =None)

Parameter Description:

--mode: set timer mode, it can set two modes: ONE_SHOT(once), PERIOD(periodically), the default mode is PERIOD mode;

--period： set timing period, unit is milliseconds (ms)

--callback：set the callback function of the timeout, the default function is None function, the function format setting as follows:

```
def callback_test (device):                    # the callback function has only one entry parameter for the Timer object created
    print (" Timer callback test
    ")
    print (device)
```

As the following example:

```
timer.init (wdt.PERIOD ,5000, callback_test)   # Set timer mode periodically executing, timeou: 5 seconds, timeout function: callback_test
```

Timer.deinit()

Deinitialises the timer. Stops the timer, and disables the timer peripheral.

- **Constant**

  below constants are used to configure Timer objects.

  Select Timer Mode:

  -- Timer.PERIODIC

  -- Timer.ONE_SHOT

- **Example**

```
>>> from machine import Timer
>>> timer = Timer(15)

>>>
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== def callback_test(device):
===     print("Timer callback test")
>>> timer.init(timer.PERIODIC, 5000, callback_test)

>>> Timer callback test

>>> Timer callback test
>>> Timer callback test
>>> timer.init(timer.ONE_SHOT, 5000, callback_test)

>>> Timer callback test

>>> timer.deinit()
```

More information refer to Timer

4.12 machine.PWM

machine.PWM class is a hardware handling class in the machine module, which is used to specify the configuration and control of the PWM device, which provides operation methods.

--PWM (Pulse Width Modulation) is a way of digit encoding for analog signal levels;

--Though Adjusting the proportional time of the effective level in one cycle to operate the device;

--PWM device has two important parameters: frequency (freq) and duty cycle (duty) [frequency: the time period from one rising edge (falling edge) to the next rising edge (falling edge), the unit is Hz; duty cycle: the proportional time of the effective level in a periodic signal];

- **Constructor**

The constructor of the PWM object in the Rt-Thread-MicroPython is as follows:

class machine.PWM (id, channel, freq, duty)

Construct a PWM object at a given bus with the following parameters:

--id: the PWM device number, such as, id =1 means the number is 1;

Or the name of the device used, such as "pwm" means the device is named pwm;

--channel: the PWM device channel number, each of device contains multiple channels with a range of [0,4];

--freq: initialization frequency, range is [1,156250];

--duty: initialization duty cycle value, range is [0,255];

for example, PWM(1,4,100,100), which means that PWM device number is 1, the channel is 4, the initialization frequency is 1000 Hz and duty cycle initialization value is 100.

- **Function**

PWM.init (channel, freq, duty)

Initialize the PWM object according to the input parameters, the parameter description is the same as above.

PWM.deinit()

Turn off a PWM object.

PWM.freq (freq)

Get or set the frequency of PWM objects, the range of which is [1,156250].

If the parameter is None, returns the frequency of the current PWM object; if not, sets the frequency of the current PWM object with [1,156250].

PWM.duty (duty)

Get or set duty cycle value of PWM object, the range of which is [0,255], such as duty =100, means that the duty cycle value is 100/255=39.22.

If the parameter is none, returns the duty cycle value of the current PWM object; if not, sets the duty cycle value of the current PWM object with [0,255].

- **Example**

```
>>>from machine import PWM
>>>pwm =PWM (3,3,1000,100)
>>>pwm .freq (2000)
>>>pwm .freq ()
2000
>>>pwm .duty (200)
>>>pwm .duty ()
200
>>>pwm .deinit ()
>>>pwm .init (3,1000,100)
```

4.13 machine.RTC

machine.RTC class is a hardware handling class in the machine module, which is used to configure and control the specified RTC device, which provides operation methods.

RTC (Real-Time Clock) can provide accurate real-time time, which can be used to generate year, month, day, time, minutes, seconds and other information.

- **Constructor**

The constructor of the RTC object in the Rt-Thread-MicroPython is as follows:

class machine.RTC()

Construct a RTC object without input parameter at a given bus. the example as follows.

- **Function**

RTC.init (datetime)

Initialise the RTC. Datetime is a tuple of the form.

```
(year ,month ,day ,wday ,hour ,minute ,second ,yday)
```

The parameters are described below:

-- year: year;

-- month: month , range is [1,12] ;

-- day: date, range is [1,31];

-- wday: week, range is [0,6], 0 means Monday, and so on;

-- hour: hours , range is [0,23];

-- minute: minutes , range is [0,59];

-- second: seconds, range is [0,59];

-- yday: days from January 1 of the current year, the range is [0,365].

The way using can refer to the example.

RTC.deinit()

Resets the RTC to the time of January 1, 2015 and starts running it again.

RTC.now()

Get get the current datetime tuple.

- **Example**

```
>>>from machine import RTC
>>>rtc=RTC ()                                    # Create RTC device
>>>rtc.init ((2019,6,5,2,10,22,30,0))
>>>rtc.now ()
(2019, 6, 5, 2, 10, 22, 40, 0)
>>>rtc.deinit ()
>>>rtc.now ()
(2015, 1, 1, 3, 0, 0, 1, 0)
```

More information refer to RTC

4.14 machine.LCD

machine.LCD class is a hardware handling class in the machine module, which is used to the configuration and control of the LCD devices, which provides the operation methods.

- **Constructor**

The constructor of the LCD object in the Rt-Thread-MicroPython is as follows:

class machine.LCD()

Construct a LCD object at a given bus without input parameters. The initialized object depends on the specific hardware. The initialization method can refer to the example.

- **Function**

LCD.light (value)

Control whether to turn on the LCD backlight. The LCD backlight is turned on or off according to that the input parameter is True or False.

LCD.ftll (color)

Fill the entire screen according to given color, which supports a variety of colors.

Parameters available are as follows:

WHITE BLACK BLUE BRED GRED GBLUE RED MAGENTA GREEN CYAN YELLOW BROWN BRRED GRAY GRAY175 GRAY151 GRAY240

The detailed methods can refer to the example.

LCD.pixel (x, y, color)

Draw a point to the specified position (x,y), which is specified by color argument.

Note: (x,y) coordinates do not exceed the actual range. Using the following method to operate on coordinates also need to follow this limit.

LCD.text (str, x, y, size)

Write a string at the specified location (x,y), which is specified by the str argument. And the font size of the string can be 16/24/32.

LCD.line (x1, y1, x2, y2)

Draw a straight line on the LCD, starting with (x1,y1) and ending with (x2,y2).

LCD.rectangle (x1, y1, x2, y2)

Draw a rectangle on the LCD, the upper left corner of which is (x1,y1), and the lower right corner is (x2,y2).

LCD.circle (x1, y1, r)

Draw a circle on the LCD, the center of which is (x1,y1), and the radius length of which is r.

LCD.show_bmp (x, y, pathname)

Output to display a picture information in 32- bit bmp format on the LCD position specified. When the bmp picture is displayed, the (x,y) coordinate is at the lower left corner of the picture.

- **Example**

```
from machine import LCD
lcd =LCD ()
lcd.light (False)
lcd.light (True)
lcd.fill (lcd.BLACK)
lcd.fill (lcd.RED)
lcd.fill (lcd.GRAY)
lcd.Fill (lcd.WHITE)
Lcd.pixel(50,50,lcd.BLUE)
lcd .text ("hello Rt-Thread",0,0,16)
lcd .text ("hello RT-Thread",0,16,24)
lcd .text ("hello RT-Thread",0,48,32)
lcd .line (0,50,239,50)
lcd .line (0,50,239,50)
lcd .rectangle (100,100,200,200)
lcd .circle (150,150,80)
lcd .show_bmp (180,50," sun.bmp ")
```

4.15 micropython

- **Function**

micropython.const (expr)

Used to represent an expression as a constant, so that the code can be optimized, when compiled. The function should be used as described below:

```
from micropython import const

CONST_X =const (123)
CONST_Y =const (2*CONST_X +1)
```

Constants declared in this way can be as global variables, which is accessed from outside. On the other hand, if a constant's name starts with underline, which is hidden, which can not be accessed as a global variable and occupy any memory at running time.

const function is directly identified by the MicroPython parser and provided as part of the MicroPython module, hence, what to code through the above pattern all can be performed on CPython and MicroPython.

micropython.opt_level ([level])

When level is given, the function sets the optimization level for subsequent script compilation and returns None; otherwise, which returns the current optimization level. The optimization level has the following compilation features:

-- Assertions: For level 0, the assertion statement is enabled and compiled into bytecode; for levels 1 and higher, which can not be compiled.

-- Built-in debug variable: For level 0, the variable can be extended to True ; for level 1 and higher, which can be extended to False.

-- Code line numbers: For levels 0, 1 and 2, the code line numbers are stored with bytecode so that the code line numbers of throwing exceptions can be caught; others are not stored.The default level is 0.

micropython.alloc_emergency_exception_buf (size)

The size bytes of the RAM are allocated for the emergency exception buffer, which is suitable for 100 bytes. The buffer is used to perform an exception when a normal RAM assignment is failed (such as in an interrupt handler), therefore, in this case, providing some useful backtracking information is needed.

Using this function, the better way is to put it at the beginning of the main script, and then all subsequent code can be had it.

micropython.mem_info ([verbose])

Print the current memory information. If the verbose parameter is given, the additional information is printed.

The printed information is related to the implementation of the function, but which currently contains the usage of the storage stack and heap. The all of heap information is printed in detail, which

indicates which of them is occupied and which of them is available.

micropython.qstr_info ([verbose])

Print the current interned string information. If the verbose parameter is given, the additional information is printed. The printed information is related to the implementation of the function, which currently contains interned strings and RAM usage. All RAM-interned string names are printed in detail mode.

micropython.stack_use()

Return a number of current heap in using.

micropython.heap_lock() / micropython.heap_unlock()

Lock or unlock the heap. Can not allocate memory when locked. In this case operation attempted to allocate heap can cause MemoryError. These functions can be nested to use, which heap_lock() can be called many times in a code block, so that the locking depth will increase. So heap_unlockheap_lock() must call the same number of times to make the heap available again.

If REPL becomes active when the heap is locked, it is forced to unlock.

micropython.kbd_intr (chr)

Set characters causing KeyboardInterrupt exceptions. During script execution, the default value is 3, corresponding to the Ctrl-C. When passing -1 to this function, the exception capture will be disabled; for 3, it will be recovered.

The function can be used to capture ctrl-c in an inputting character stream (which is often used for REPL) to prevent the stream from being used for other purposes.

micropython.schedule (func, arg)

Schedule the function func to be executed “very soon”. The function is passed the value arg as its single argument. “Very soon” means that the MicroPython runtime will do its best to execute the function at the earliest possible time, given that it is also trying to be efficient, and that the following conditions hold:

A scheduled function will never preempt another scheduled function.

Scheduled functions are always executed “between opcodes” which means that all fundamental Python operations (such as appending to a list) are guaranteed to be atomic.

A given port may define “critical regions” within which scheduled functions will never be executed. Functions may be scheduled within a critical region but they will not be executed until that region is exited. An example of a critical region is a preempting interrupt handler (an IRQ).

A use for this function is to schedule a callback from a preempting IRQ. Such an IRQ puts restrictions on the code that runs in the IRQ (for example the heap may be locked) and scheduling a function to call later will lift those restrictions.

Note: If schedule() is called from a preempting IRQ, when memory allocation is not allowed and the callback to be passed to schedule() is a bound method, passing this directly will fail. This is because creating a reference to a bound method causes memory allocation. A solution is to create a reference to the method in the class constructor and to pass that reference to schedule(). This is discussed in detail here reference documentation under “Creation of Python objects”.

There is a finite queue to hold the scheduled functions and schedule() will raise a RuntimeError if

the queue is full.

4.16 network

This module provides network drivers and routing configuration. To use this module, a MicroPython variant/build with network capabilities must be installed. Network drivers for specific hardware are available within this module and are used to configure hardware network interface(s). Network services provided by configured interfaces are then available for use via the usocket module.

- **Network class configuration**

The following specific classes implement the interface of abstract network card and provide various interfaces for controlling network.

class WLAN

This class provides a driver for WiFi network processors. Refer to the example below:

```
import network
# enable station interface and connect to WiFi access point
nic=network.WLAN (network.SA_IF)
nic.active (True)
nic.connect ('your-ssid','your-password')
# now use sockets as u mutual
```

- **Constructor**

The constructor of the WLAN object in the Rt-Thread-MicroPython is as follows:

class network.WLAN (interface_id)

Create a WLAN network interface object. Supported interfaces are network.STA_IF (station aka client, connects to upstream WiFi access points) and network.AP_IF (access point, allows other WiFi clients to connect). Availability of the methods below depends on interface type. For example, only STA interface may WLAN.connect() to an access point.

- **Function**

WLAN.active ([is_active]

Activate (“up”) or deactivate (“down”) network interface, if boolean argument is passed. Otherwise, query current state if no argument is provided. Most other methods require active interface.

WLAN.connect (ssid, password)

Connect to the specified wireless network, using the specified account and password.

WLAN.disconnect()

disconnected from the currently connected wireless network.

WLAN.scan()

Scan for the available wireless networks.

Scanning is only possible on STA interface. Return list of tuples with the information about WiFi access points:

```
(ssid ,bssid ,channel ,rssi ,authmode ,hidden)
```

WLAN.status ([param])

Return the current status of the wireless connection.

When called with no argument the return value describes the network link status. The possible statuses are defined as constants:

- STAT_IDLE – no connection and no activity,
- STAT_CONNECTING – connecting in progress,
- STAT_WRONG_PASSWORD – failed due to incorrect password,
- STAT_NO_AP_FOUND – failed because no access point replied,
- STAT_CONNECT_FAIL – failed due to other problems,
- STAT_GOT_IP – connection successful.

When called with one argument param should be a string naming the status parameter to retrieve. Supported parameters in WiFI STA mode are: 'rssi'.

WLAN.isconnected()

In case of STA mode, which returns True if connected to a WiFi access point and has a valid IP address. In AP mode returns True when a station is connected. Otherwise returns False .

WLAN.ifconfig ([(ip, subnet, gateway, dns)]

Get/set IP-level network interface parameters: IP address, subnet mask, gateway and DNS server. When called with no arguments, this method returns a 4-tuple with the above information. To set the above values, pass a 4-tuple with the required information. For example::

```
nic.ifconfig (('192.168.0.4','255.255.255.0','192.168.0.1,'8.8.8.8))
```

WLAN.config (param=value, ...)

Get or set general network interface parameters. These methods allow to work with additional parameters beyond standard IP configuration (as dealt with by WLAN.ifconfig()). These include network-specific and hardware-specific parameters. For setting parameters, keyword argument syntax should be used, multiple parameters can be set at once. For querying, parameters name should be quoted as a string, and only one parameter can be queries at time:

```
# Set WiFi access point name (formally known as ESSID) and WiFi password
ap.config (essid =' My_AP',password ="88888888")

# Query params one by one
print (ap.config ('essid'))
Print(ap.config('essid'channel'essid'))
```

The following parameters are currently supported:

-- mac： MAC address (bytes)

-- essid： WiFi access point name (string)

-- channel： WiFi channel (integer)

-- hidden： Whether ESSID is hidden (boolean)

-- password： Access password (string)

- **Example -: STA pattern**

```
import network
wlan=network .WLAN (network.SA_IF)
wlan.scan ()
wlan.connect("rtthread","021888888")
wlan.isconnected()
```

- **Example 2: AP mode**

```
import network
ap=network.WLAN (network.AP_IF)
ap.config (essid ="hello_Rt-Thread",password ="88888888)
ap.active(True)
ap.config("essid")
```

4.17 rtthread

rtthread module provides functions related to the Rt-Thread operating system, such as checking stack usage.

- **Function**

  rtthread.current_tid()

  Return the current thread id.

  rtthread.is_preempt_thread()

  Return whether it is a preemptable thread.

- **Example:**

```
>>>import rtthread
>>>
>>>rtthread .is_preempt_thread ()          # determine if it's a preemptible thread
True
>>>rtthread .current_tid ()                # current thread id
268464956
```

5. OpenMV

- **Support image processing API**

  1)Reference Image Processing

  2)reference OpenMV official

- **Support UI-API (LVGL)**

  1) API interface (support C language)

- **Example**

```
import lvgl as lv
import lvdrv

lvdrv.init()   # must be called

scr=lv.obj()
btn=lv.btn(scr)
btn.align(lv.scr_act (), lv.ALIGN.CENTER,0,0)
label =lv.label (btn)
label.set_text(" Hello World! ")
lv.scr_load
```

6. OpenMV IDE

OpenMV IDE is a programming tool for OpenMV Camera with a powerful text editor supported by QtCreator, a frame buffer viewer, a histogram display, and an integrated serial terminal for debugging.

Please refer to OpenMV for use.

- **Communication bridge**

Since K210 having no USB interface, OpenMV IDE requires additional transfer bridge chips for the connection. Note: what is provided in the SDK is the firmware based on the STM32F411.

1) Hardware connection (STM32 Pin function schematic)

```
PB12-SPI_CS
PB13-SPI_SCK
PB14-SPI_MISO
PB15-SPI_MOSI
PA8 -SPI_INT
PA9 -SPI_READY_PIN
PA10-DATA_READY_PIN
```

2) Compilation

Enter the directory src/bridge/stm32f411 through the Rt-Thread ENV, execute the following instructions to update the package and compile.

```
pkgs --update
scons
```

7. Rt-Thread MicroPython IDE

- **Overview**

It is the best MicroPython plug-ins for MicroPython development, which is a powerful tool based on VSCode, the features as follows:

- ✓ Easy development board connection (serial port, network, USB)
- ✓ Support MicroPython - based code intelligent completion and syntax checking
- ✓ Support MicroPython REPL interactive environments
- ✓ Support code examples and demo programs
- ✓ Support engineering synchronization
- ✓ Support for downloading individual files or folders to the development board
- ✓ Support for fast running code files in memory
- ✓ Support for running code snippets
- ✓ Support multiple mainstream MicroPython development boards
- ✓ Support Windows、Ubuntu、Mac operating system

More information, please refer to Rt-Thread MicroPython Develop Environmet

8. OTA

- **Bootloader**

boot burning, please use kflash provided by the Canaan official.

1) install kflash

- install pip

```
sudo pip3 install kflash
```

- Download source code

```
wget https://raw.githubusercontent.com /kendryte/kflash.py/master/kflash.py
```

2) Burn boot.bin [there are different burning commands on the hardware platform]

burning commands as follows:

```
# Linux or macOS
# Using pip
kflash -B dan boot.bin
kflash -B dan -t boot.bin # Open a Serial Terminal After Finish
# Using source code
python3 kflash.py -B dan boot.bin
python3 kflash.py -B dan -t boot.bin # Open a Serial Terminal After Finish

# Windows CMD or PowerShell
# Using pip
kflash -B dan boot.bin
kflash -B dan -t boot.bin # Open a Serial Terminal After Finish
kflash -B dan -n -t boot.bin # Open a Serial Terminal After Finish, do not use ANSI
    colors
# Using source code
python kflash.py -B dan boot.bin
python kflash.py -B dan -t boot.bin # Open a Serial Terminal After Finish
python kflash.py -B dan -n -t boot.bin # Open a Serial Terminal After Finish, do not
     use ANSI colors

# Windows Subsystem for Linux
# Using pip
sudo kflash -B dan -p /dev/ttyS13 boot.bin # ttyS13 Stands for the COM13 in Device
    Manager
sudo kflash -B dan -p /dev/ttyS13 -t boot.bin # Open a Serial Terminal After Finish
# Using source code
sudo python3 kflash.py -B dan -p /dev/ttyS13 boot.bin # ttyS13 Stands for the COM13
    in Device Manager
sudo python3 kflash.py -B dan -p /dev/ttyS13 -t boot.bin # Open a Serial Terminal
    After Finish
```

As completing the burning, open debug serial port terminal, then the following output can be seen:

```
heap:  [0x80587d99 - 0x80600000]
initialize rti_board_start: 0 done
initialize io_config_init: 2 done
initialize rt_hw_pin_init: 0 done

 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Dec 28 2020
 2006 - 2020 Copyright by rt-thread team
do components initialization.
initialize rti_board_end: 0 done
initialize dfs_init: 0 done
initialize ulog_init: 0 done
initialize ulog_console_backend_init: 0 done
initialize k210_hwtimer_init: 0 done
initialize libc_system_init: 0 done
initialize cplusplus_system_init: 0 done
initialize load_application[D/FAL] (fal_flash_init: 63) Flash device |
                    spi_nor | addr:  0x00000000 | len:  0x01000000 | blk_size:  0
    x00001000 |initialized finish.
[I/FAL] ==================== FAL partition table ====================
[I/FAL] | name     | flash_dev |   offset   |    length  |
[I/FAL] -------------------------------------------------------------
[I/FAL] | bl       | spi_nor   | 0x00000000 | 0x00100000 |
[I/FAL] | app      | spi_nor   | 0x00100000 | 0x00300000 |
[I/FAL] | download | spi_nor   | 0x00400000 | 0x00200000 |
[I/FAL] | fs       | spi_nor   | 0x00600000 | 0x00200000 |
[I/FAL] =============================================================
[I/FAL] RT-Thread Flash Abstraction Layer (V0.5.0) initialize success.
[I/OTA] RT-Thread OTA package(V0.2.3) initialize success.
```

```
[E/OTA] (get_fw_hdr: 150) Get firmware header occur CRC32(calc.crc:
    ffffffffc933e758 != hdr.info_crc32:  fffffffaaab66c3) error on 'download'
    partition!
[142] I/OTA:  check upgrade...
[E/OTA] (get_fw_hdr: 150) Get firmware header occur CRC32(calc.crc:
    ffffffffc933e758 != hdr.info_crc32:  fffffffaaab66c3) error on 'download'
    partition!
[E/OTA] (rt_ota_check_upgrade: 464) Get OTA download partition firmware header
    failed!
[168] I/OTA:  No firmware upgrade!
[E/OTA] (get_fw_hdr: 150) Get firmware header occur CRC32(calc.crc:  7e61bad8 != hdr
    .info_crc32:  773d95aa) error on 'app' partition!
[185] E/OTA:  App verify failed! Need to recovery factory firmware.
: 0 done
initialize finsh_system_init: 0 done
msh />world

msh />
```

- **Customize OTA firmware**

1) Add downloader

This section describes how to add the downloader into firmware.

To add this feature needs to use the env tool. Please follow the following steps to operate:

-- Download the ota_downloader package and select the Enable Ymodem OTA.(highlighted with red box)

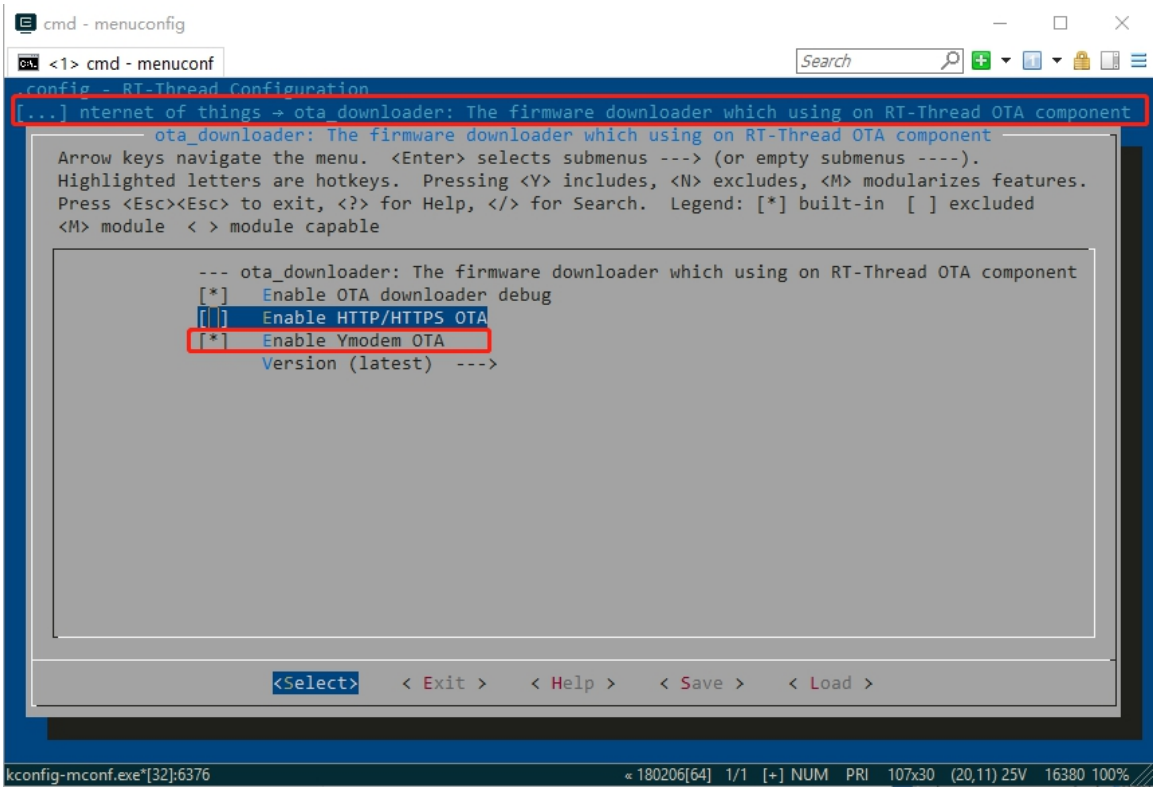


Figure 7.1: ota_downloader package configuration

--Enable HTTP/HTTPS OTA as required.

--Make sure that the fal driver is turned on correctly, that the fal configuration provided in the SDK use the same header file with the bootloader. Please try to avoid to modify fal_cfg.h, so that partition exceptions will not be caused.

--Recompile the firmware.

2) Release package

OTA firmware is based on ordinary firmware for secondary packaging, supporting encryption, compression and other functions. The packaging methods are as follows:

```
Usage:
rt_ota_packaging_tool_cli -f BIN -v VERSION -p PARTNAME [-o OUTFILE] [-c CMPRS_TYPE] [-
s CRYPT_TYPE] [-i IV] [-k KEY] [-h]
        -f bin file.
        -v firmware's version.
        -p firmware's target part name.
        -o output rbl file path.(optional)
        -c compress type allow [quicklz|gzip|lzma|none](optional)
        -s crypt type allow [aes|none](optional)
        -i iv for aes-256-cbc
        -k key for aes-256-cbc
        -h show this help information
```

Refer to this example

```
./rt_ota_packaging_tool_cli-x64 -f app.bin  -v 2.0.0  -p  app  -o app.rbl  -c quicklz  -s aes  -
i  usguJdWWUugadheA -k 0123456789ABCDEF0123456789ABCDEF
```

The firmware packager offers three forms of firmware compression: fastlz、 quicklz and gzip, which can be chosen according to the actual requirements.

As follows are the corresponding encryption keys provided by the SDK for bootloader:

Encryption key:

```
qnbGbxjnxtwaPqjuntaZyQvZepywimdg
```

IV：

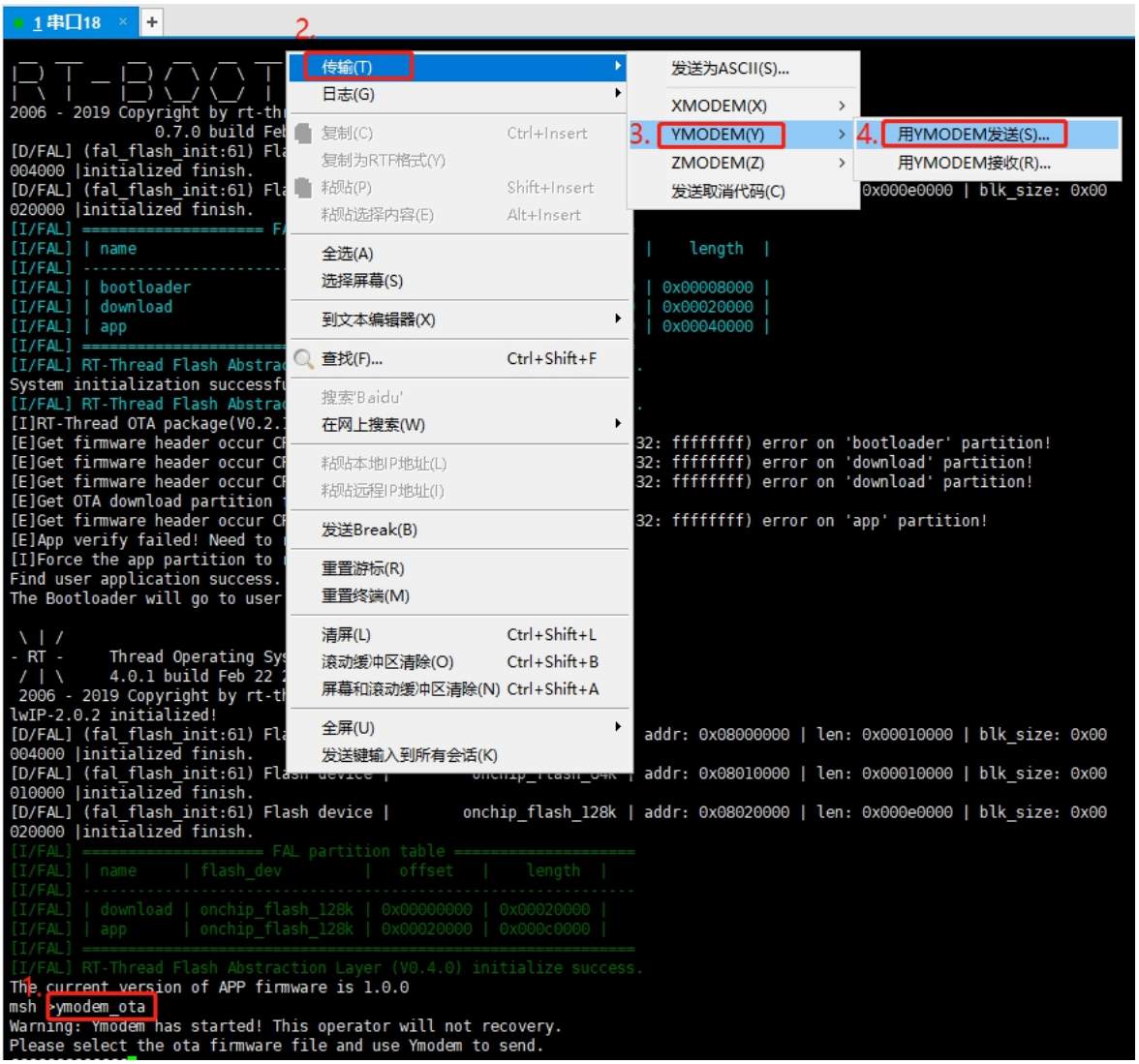
```
usguJdWWUugadheA
```

3) Update firmware through ymodem under bootloader

Recommendation: Xshell terminal (upgrade firmware on Ymodem protocol).

After entering the ymodem_ota command on the msh command line, right-click and find the YMODEM sending option in the menu bar, as shown below:

-- Choose Ymodem mode to send upgrade firmware:



-- check the **rtthread.rbl** file generated previously by OTA firmware packaging tool.

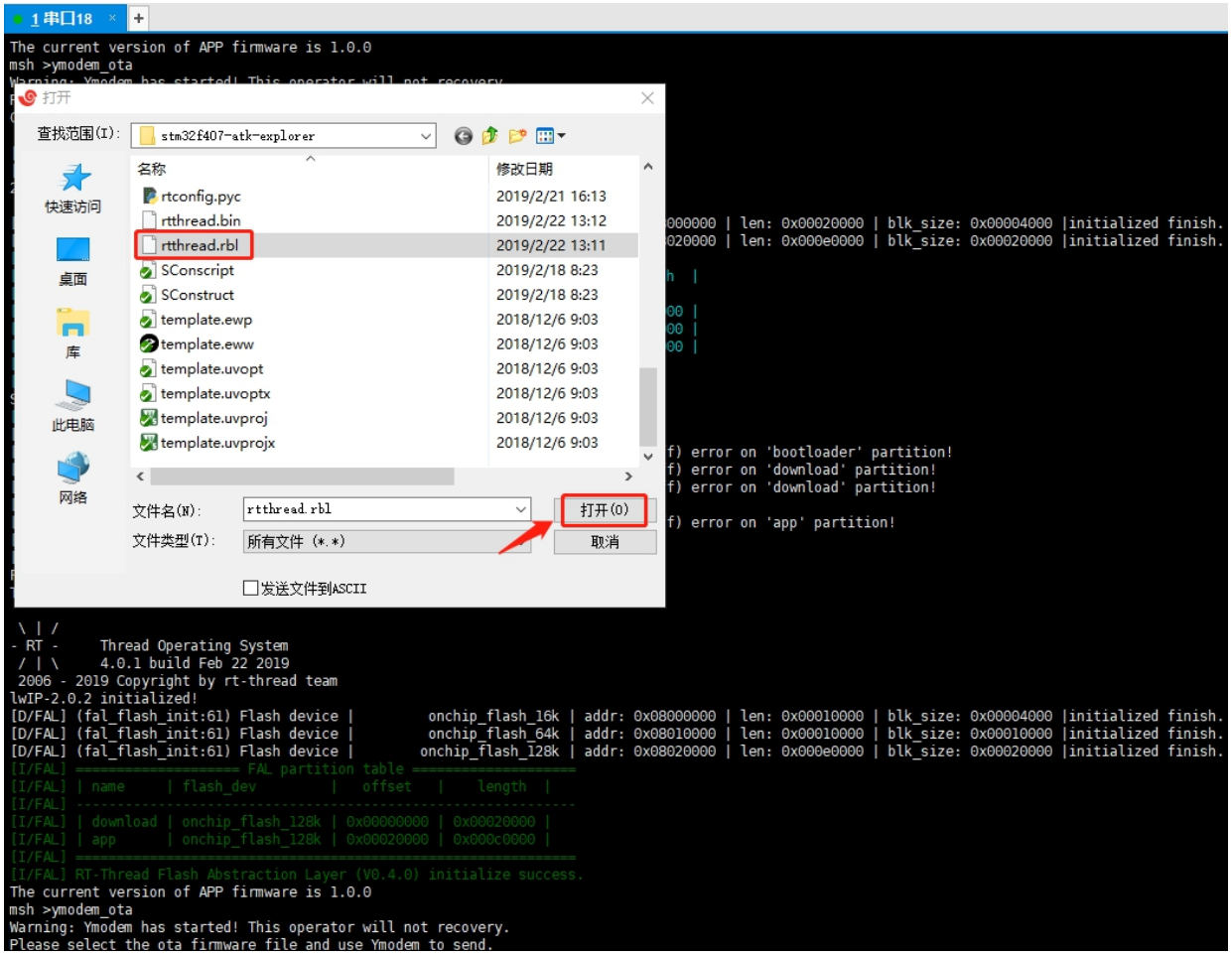


Figure 7.4: Ymodem selection file

--Next, the upgrade firmware is downloaded to the downloading partition by Ymodem. Then, the system will restart automatically and perform OTA upgrades.

4) OTA upgrade firmware under app

It is the same as the bootloader that the way upgrading the firmware with Ymodem method in the app. And the upgrade firmware command is as shown bellow:

```
http_ota [url]
```

5) Customized OTA Downloader

The working principle of the OTA downloader is to write the content of the rbl file into the downloading partition.

-- Initialize FAL components

-- Get downloading partition handle

-- Erase downloading partition

-- write data of data of rbl file into downloading partition

Example

Initialize FAL components

```
/**
 * FAL (Flash Abstract Layer) initialization.
 * It will initialize all flash device and all flash partition.
 *
 * @ return >=0: partitions total number
 */
int fal_init (void);
```

Get a partition handle

```
/**
 * find the partition by name
 *
 * param name partition name @
 *
 * return !@ =NULL: partition
 *             NULL: not found
 */
const struct fal_partition *fal_partition_find (const char * name);
```

Erasure partition

```
/**
 * erase partition data
 *
 * param part partition @
 * param addr relative address for partition @
 * param size erase size @
 *
 * @ return >=0: successful erased data size
 *            - error 1
 */
int fal_partition_erase: conststruct fal_partition* part, uint32_t addr, size_t size
    );

/**
 * erase partition all data
 *
 * param part partition @
 *
 * @ return >=0: successful erased data size
 *            - error 1
 */
int fal_partition_erase_all (construct fal_partition* part);
```

Write data into partition

```
/**
 * write data to partition
 *
 * param part partition @
 * param addr relative address for partition @
 * param buf write buffer @
 * param size write size @
 *
 * @ return >=0: successful write data size
 *           - error 1
 */
int fal_partition_write (conststruct fal_partition* part, uint32_t addr, const
    uint8_t *buf ,size_t size);
```

The Example for ymodem OTA Downloader

```
/*
 * Copyright (c) 2006-2018, RT-Thread Development Team
 *
 * SPDX-License-Identifier: Apache-2.0
 *
 * Change Logs:
 * Date           Author       Notes
 * 2018-01-30     armink       the first version
 * 2018-08-27     Murphy       update log
 */

#include <rtthread.h>
#include <stdio.h>
#include <stdbool.h>
#include <finsh.h>
#include <fal.h>
#include <ymodem.h>

#define DBG_ENABLE
#define DBG_SECTION_NAME               "ymodem"
#ifdef OTA_DOWNLOADER_DEBUG
#define DBG_LEVEL                      DBG_LOG
#else
#define DBG_LEVEL                      DBG_INFO
#endif
#define DBG_COLOR
```

```c
#include <rtdbg.h>

#ifdef PKG_USING_YMODEM_OTA

#define DEFAULT_DOWNLOAD_PART "download"

static char* recv_partition = DEFAULT_DOWNLOAD_PART;
static size_t update_file_total_size, update_file_cur_size;
static const struct fal_partition * dl_part = RT_NULL;

static enum rym_code ymodem_on_begin(struct rym_ctx *ctx, rt_uint8_t *buf, rt_size_t
     len)
{
    char *file_name, *file_size;

    /* calculate and store file size */
    file_name = (char *)&buf[0];
    file_size = (char *)&buf[rt_strlen(file_name) + 1];
    update_file_total_size = atol(file_size);
    rt_kprintf("Ymodem file_size:%d\n", update_file_total_size);

    update_file_cur_size = 0;

    /* Get download partition information and erase download partition data */
    if ((dl_part = fal_partition_find(recv_partition)) == RT_NULL)
    {
        LOG_E("Firmware download failed! Partition (%s) find error!", recv_partition
            );
        return RYM_CODE_CAN;
    }
```

```c
    if (update_file_total_size > dl_part->len)
    {
        LOG_E("Firmware is too large! File size (%d), '%s' partition size (%d)",
            update_file_total_size, recv_partition, dl_part->len);
        return RYM_CODE_CAN;
    }

    LOG_I("Start erase. Size (%d)", update_file_total_size);

    /* erase DL section */
    if (fal_partition_erase(dl_part, 0, update_file_total_size) < 0)
    {
        LOG_E("Firmware download failed! Partition (%s) erase error!", dl_part->name
            );
        return RYM_CODE_CAN;
    }

    return RYM_CODE_ACK;
```

```
}

static enum rym_code ymodem_on_data(struct rym_ctx *ctx, rt_uint8_t *buf, rt_size_t
    len)
{
    /* write data of application to DL partition  */
    if (fal_partition_write(dl_part, update_file_cur_size, buf, len) < 0)
    {
        LOG_E("Firmware download failed! Partition (%s) write data error!", dl_part
            ->name);
        return RYM_CODE_CAN;
    }

    update_file_cur_size += len;

    return RYM_CODE_ACK;
}

void ymodem_ota(uint8_t argc, char **argv)
{
    struct rym_ctx rctx;

    if (argc < 2)
    {
        recv_partition = DEFAULT_DOWNLOAD_PART;
        rt_kprintf("Default save firmware on download partition.\n");
    }
    else
    {
        const char *operator = argv[1];
        if (!strcmp(operator, "-p")) {
            if (argc < 3) {
                rt_kprintf("Usage: ymodem_ota -p <partiton name>.\n");
                return;
            } else {
                /* change default partition to save firmware */
                recv_partition = argv[2];
            }
        }else{
            rt_kprintf("Usage: ymodem_ota -p <partiton name>.\n");
            return;
        }
    }

    rt_kprintf("Warning: Ymodem has started! This operator will not recovery.\n");
    rt_kprintf("Please select the ota firmware file and use Ymodem to send.\n");

    if (!rym_recv_on_device(&rctx, rt_console_get_device(), RT_DEVICE_OFLAG_RDWR |
        RT_DEVICE_FLAG_INT_RX,
```

```
                            ymodem_on_begin, ymodem_on_data, NULL,
                                RT_TICK_PER_SECOND))
    {
        rt_kprintf("Download firmware to flash success.\n");
        rt_kprintf("System now will restart...\r\n");

        /* wait some time for terminal response finish */
        rt_thread_delay(rt_tick_from_millisecond(200));

        /* Reset the device, Start new firmware */
        extern void rt_hw_cpu_reset(void);
        rt_hw_cpu_reset();
        /* wait some time for terminal response finish */
        rt_thread_delay(rt_tick_from_millisecond(200));
    }
    else
    {
        /* wait some time for terminal response finish */
        rt_thread_delay(RT_TICK_PER_SECOND);
        rt_kprintf("Update firmware fail.\n");
    }

    return;
}
/**
 * msh />ymodem_ota
*/
MSH_CMD_EXPORT(ymodem_ota, Use Y-MODEM to download the firmware);

#endif /* PKG_USING_YMODEM_OTA */
```