Arm® Cortex®-M

32-bit Microcontroller

NuMicro® Family
NuMicroPy
User Manual

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 OVERVIEW

NuMicroPy is the port of MicroPython to Nuvoton NuMicro® family microcontrollers (MCUs). The MicroPython[1] project aims to put an implementation of Python 3.x on the microcontrollers and small embedded systems. Refer to Table 1-1 for NuMicroPy support status.

MCU	Board	Firmware ROM Size	Firmware RAM Size
M55M1	NuMaker-M55M1	342KB	350KB
M5531	NuMaker-M5531	342KB	350KB

Table 1-1 NuMicroPy Support Status

The MicroPython implements Python 3.4 and some selected feature of Python 3.5, but there are some conflicting results in MicroPython when compared to standard Python. See details[2].

[1] http://micropython.org/

[2] http://docs.micropython.org/en/latest/genrst/index.html

2 NUMICROPY INTRODUCTION

The MicroPython divides the code into two parts, python interpreter firmware (firmware.bin) and the user's python code. The firmware must be burned into the MCU first. After boot, the firmware executes the user's python code.

The execution of the python code supports the REPL mode and/or performs the python code from storage. The firmware tries to perform the python code from storage first, and finally enter REPL mode. Uses can test their python code in REPL mode, and finally put the python code into storage.

The MicroPython defines the I/O classes associated with the MCU peripheral in the pyb module[3]. NuMicroPy implements these I/O classes according to these definitions.

2.1 REPL

REPL stands for Read Evaluate Print Loop, and is the name given to the interactive MicroPython prompt that you can access on the MCUs. Using the REPL is by far the easiest way to test out your python code (Figure 2-1).

The REPL is always available on the USB1.1 port. You should be able to access the REPL directly from your PC.

To access the prompt over USB-to-serial you need to use a terminal emulator program. On Windows, Tera Term is a good choice; on Mac you can use the built-in screen program, and Linux has picocom and minicom.

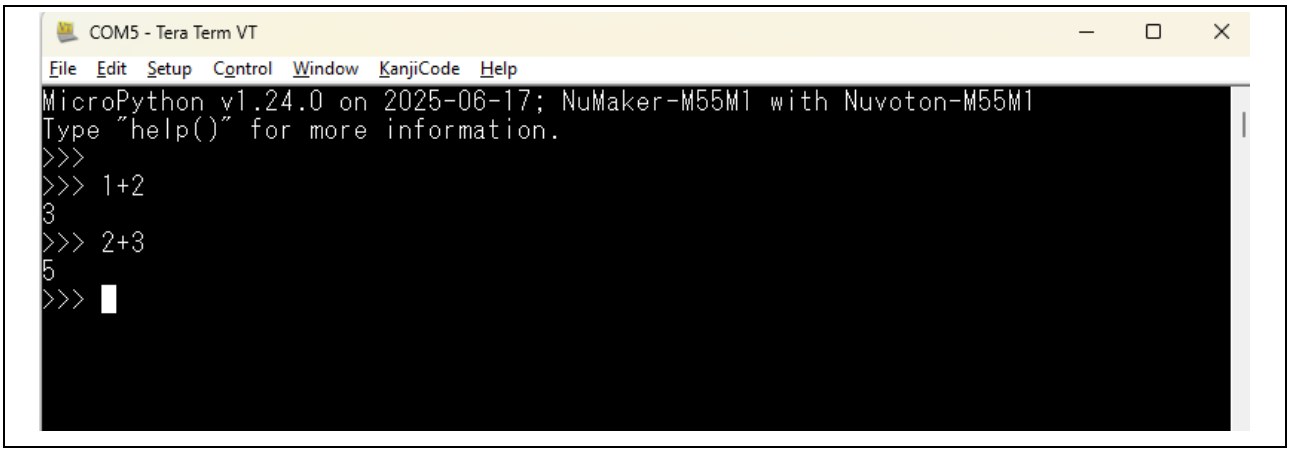


Figure 2-1 REPL Mode

2.2 Flash Layout

Figure 2-2 show the layout for NuMicroPy. The NuMicroPy divides the MCU's embedded flash into two partitions, one is the firmware partition and the other is the data partition.

The firmware partition is used to put the firmware binary code. The data partition will be a partition of FAT file system and exported to "PYBFlash" disk through USB mass storage class.

The firmware will attempt to mount the "PYBFlash" disk at the beginning of execution. If mount fails, it will force the data partition to be formatted into FAT file system and produce two blank python file (main.py and boot.py). You can connect USB1.1 port to access the "PYBFlash" disk and then write your python code to these files.

[3] pyb module: MicroPython board related module.

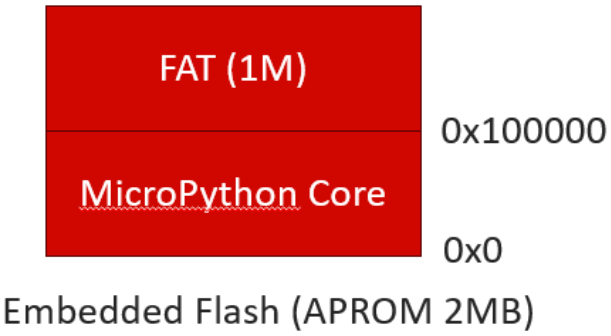


Figure 2-2 Flash Layout

2.3 Modules and I/O Classes Support List

Table 2-1 and Table 2-2 show the M487 and M263 support status on the modules and I/O classes of MicroPython.

Module	Description	M55M1	M5531
array	Arrays of numeric data	✓	✓
cmath	Mathematical function for complex numbers	✓	✓
gc	Control the garbage collector	✓	✓
math	Mathematical functions	✓	✓
binascii	Binary/ASCII conversions	✓	✓
collections	Collection and container types	✓	✓
hashlib	Hashing algorithms	✓	✓
io	Input/output streams	✓	✓
json	JSON encoding and decoding	✓	✓
re	Simple regular expressions	✓	✓
select	Wait for events on a set of streams	✓	✓
struct	Pack and unpack primitive data types	✓	✓
time	Time related functions	✓	✓
_thread	Multithreading support	✓	✓

uctypes	Access binary data in a structured way	✓	✓
machine	Functions related to the hardware	✓	✓

Table 2-1 Default Supported Modules

I/O Class	Description	IPs	M55M1	M5531
ADC	Analog to digital conversion	EADC	✓	✓
CAN	Controller area network communication bus	CANFD	✓	✓
I²C	A two-wire serial protocol	I2C, LPI2C	✓	✓
Pin	Control I/O pins	GPIO	✓	✓
PinAF	Pin alternate functions	GPIO	✓	✓
RTC	Real time clock	RTC	✓	✓
SPI	A master-driven serial protocol	SPI, LPSPI	✓	✓
Timer	Control internal timers	TMR, LPTMR	✓	✓
TimerChannel	Setup a channel for a timer	TMR, LPTMR	✓	✓
UART	Duplex serial communication bus	UART, LPUART	✓	✓
PWM	BPWM/EPWM generator and capture timer	BPWM, EPWM	✓	✓
WDT	Watchdog timer	WDT	✓	✓

Table 2-2 Default Supported I/O Classes

3 HOW TO START NUMICROPY

The following uses the NuMaker-M55M1 board to show how to burn firmware into the NuMicro® MCU and how to update your python code.

3.1 Nuvoton Nu-Link Driver Download and Install

Please visit the Nuvoton software download website to download “NuMicro ICP Programming Tool” file. When the NuMicro ICP Programming Tool has been downloaded successfully, please unzip the file and execute the “NuMicro ICP Programming Tool.exe” to install the driver.

3.2 Hardware Setup Steps

1. Turn on ICE function switch pin 1 and 2.
2. Connect ICE USB port to PC (Figure 3-1).

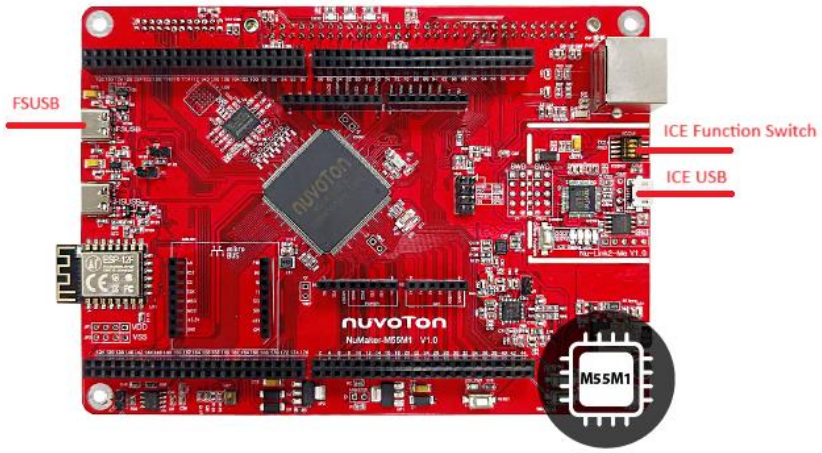


Figure 3-1 NuMaker-M55M1 Board

3.3 Burn Firmware

The Nu-Link-Me exports a “NuMicro MCU” disk, just Copy and Paste your firmware.bin into “NuMicro MCU” disk (Figure 3-2). After firmware burning, you can see MicroPython prompt on your terminal screen as Figure 2-1.

You can get prebuilt firmware from NuMicroPy repository[4].

[4] https://github.com/OpenNuvoton/NuMicroPy

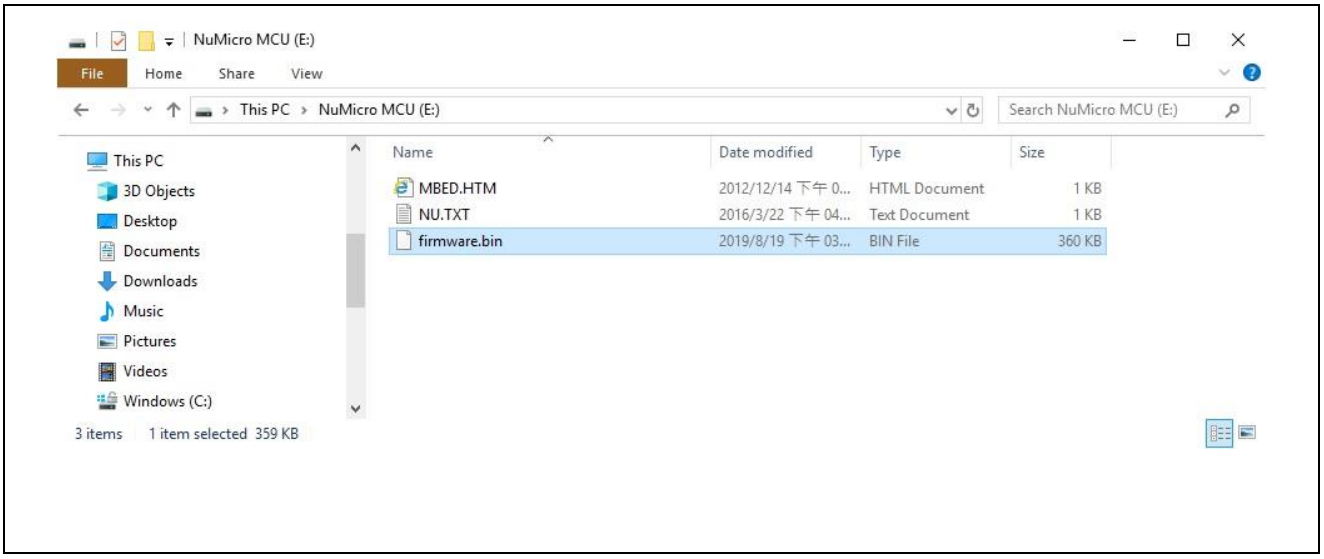


Figure 3-2 Copy and Paste Firmware

3.4 Python Code Update Steps

1. Connect FSUSB port to PC.
2. Set up your terminal program (Figure 3-3 and Figure 3-4).

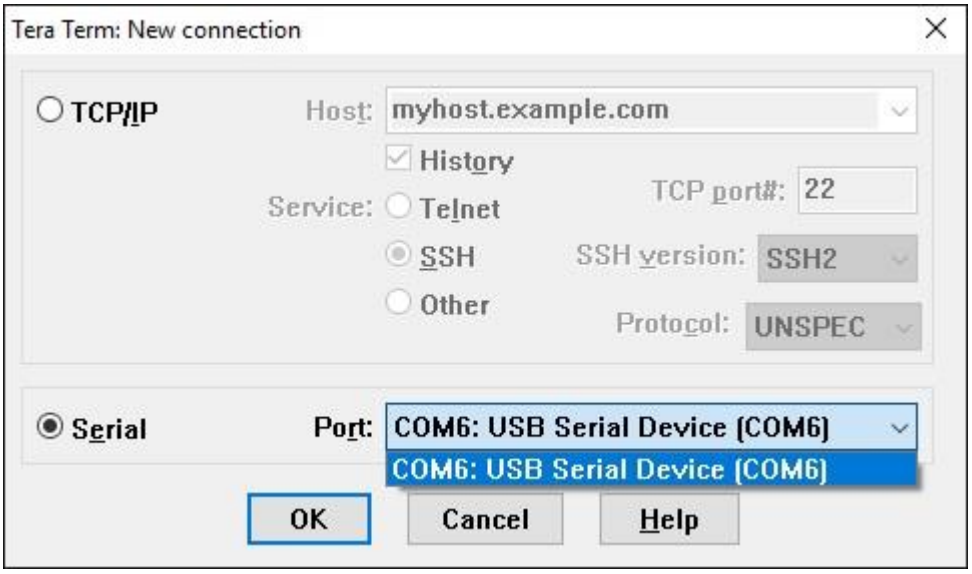


Figure 3-3 Create a Serial Connection

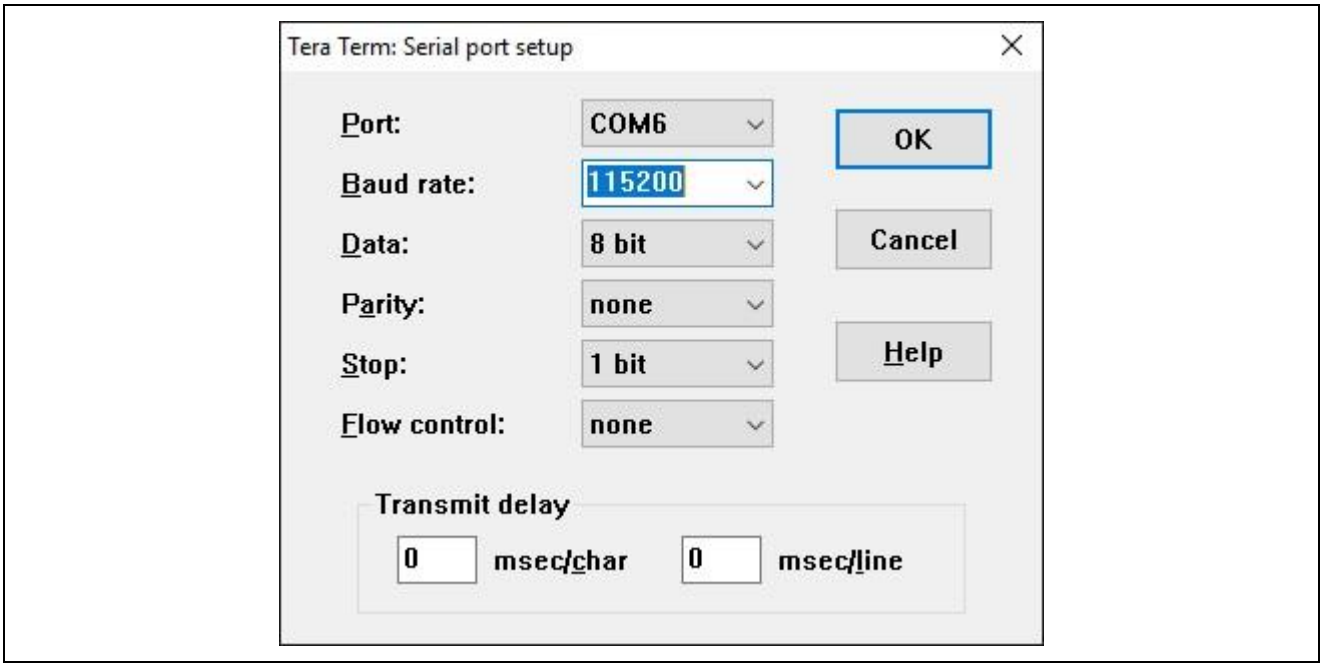


Figure 3-4 Serial Port Setup

3. Enable USB mass storage mode. The firmware will export a PYBFLASH disk (Figure 3-5).
   - Press BTN0 + RESET button.
4. Update your python code to boot.py or main.py (Figure 3-6).
5. Press the RESET button (Figure 3-7).

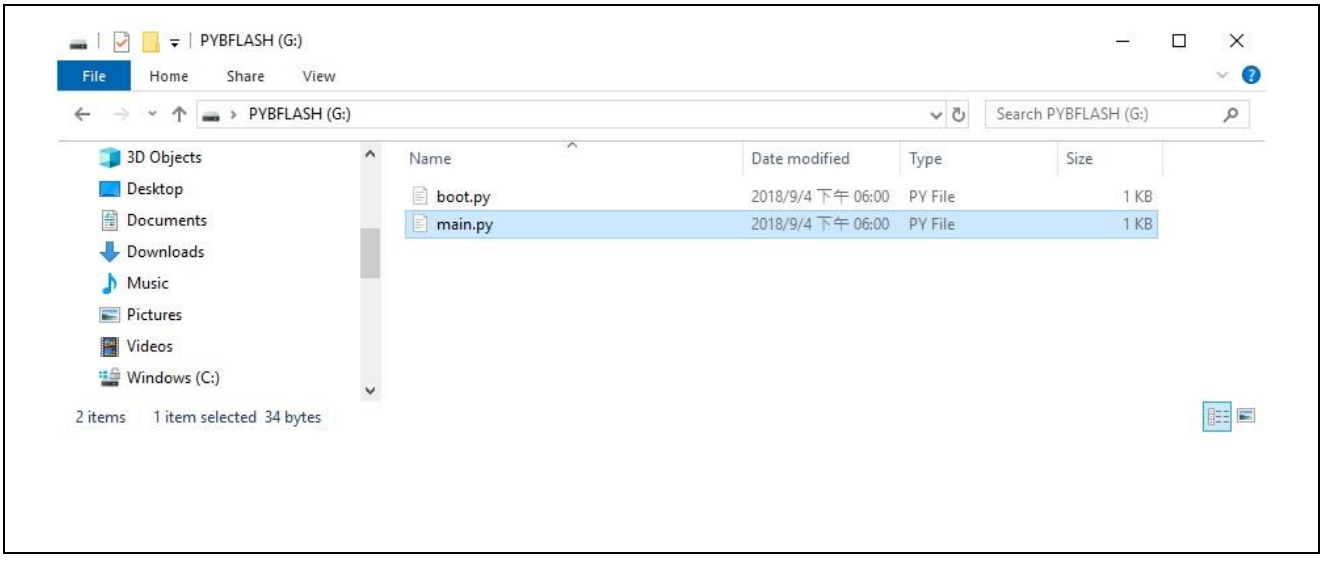


Figure 3-5 PYBFLASH Disk

```python
# main.py -- put your code here!
import time
from pyb import Pin

def btn1_callback(pin):
    print(pin)

btn1 = Pin.board.BTN1
btn1_af = btn1.af_list()

#btn1 already in input mode
print(btn1)
print(btn1.af_list())


btn1.irq(handler=btn1_callback, trigger=Pin.IRQ_FALLING)

#change ledr to output mode
r = Pin('LEDR', Pin.OUT)
#r = Pin('LEDR', Pin.OUT, pull = None, value = 0)   #Defalut output low

print(r)
print(r.af_list())

while True:
    pin_value = btn1.value()
    if pin_value == 0:
        print('Button press')
    time.sleep_ms(1000)

print('demo done')
```

Figure 3-6 Write Switch Button Example Code

```
>>>
>>> Pin(Pin.cpu.H1, mode=Pin.IN)
[Pin.AF_PH1_EBIO_ADR6(index: 1), Pin.AF_PH1_SPI3_MOSI(index: 2), Pin.AF_PH1_UART
5_RXD(index: 3), Pin.AF_PH1_SC1_CLK(index: 4), Pin.AF_PH1_I2C3_SDA(index: 5), Pi
n.AF_PH1_TM1_EXT(index: 6)]
Pin(Pin.cpu.H4, mode=Pin.OUT)
[Pin.AF_PH4_EBIO_ADR3(index: 1), Pin.AF_PH4_SPI1_MISO(index: 2), Pin.AF_PH4_UART
7_nRTS(index: 3), Pin.AF_PH4_UART6_TXD(index: 4), Pin.AF_PH4_SPI3_I2SMCLK(index:
 5), Pin.AF_PH4_EPWM0_CH5(index: 6)]
Pin(Pin.cpu.H1, mode=Pin.IN)
Button press
Pin(Pin.cpu.H1, mode=Pin.IN)
Pin(Pin.cpu.H1, mode=Pin.IN)
```

Figure 3-7 Execute Switch Button Example Code

4 HOW TO CUSTOMIZE MICROPYTHON FIRMWARE

The development of MicroPython firmware is in the Unix-like environment. The description below uses Ubuntu 22.04.

4.1 Packages Requirement

The following packages will need to be installed before you can compile and run MicroPython.

- build-essential
- libreadline-dev
- libffi-dev
- git
- pkg-config
- python3-pip

To install these packages, use the following command.

```
1. sudo apt-get install build-essential libreadline-dev libffi-dev git pkg-config
   python3-pip
2. pip3 install gitpython
3. pip3 install rich
```

4.2 Install GNU Arm Toolchain

Download GNU Arm toolchain linux 64-bit version arm-gnu-toolchain-13.2.Rel1-x86_64-arm-none-eabi.tar.xz from Arm Developer [5] website.

Next, use the tar command to extract the file to your favor directory (ex. /usr/local)

```
1. mv arm-gnu-toolchain-13.2.Rel1-x86_64-arm-none-eabi.tar.xz /usr/local/
2. cd /usr/local
3. tar -xvf arm-gnu-toolchain-13.2.Rel1-x86_64-arm-none-eabi.tar.xz
```

Now, modify your PATH environment variable to access the bin directory of toolchain

4.3 Build Firmware

Get source code from GitHub.

```
1. Python3 download_src.py --board NuMaker-M55M1
```

To build NuMaker-M55M1 firmware, use the following command.

```
2. cd ../M55M1
3. make V=1
```

To build NuMaker-M5531 firmware, use the following command.

```
4. cd ../M5531
5. make V=1
```

4.4 Enable/Disable Module

Default modules support list as Table 2-1. You can enable/disable the built-in modules of MicroPython by modifying the options of mpconfigport.h. It should be noted that some options may have dependencies, which may cause compiling failure. Below is part of M55M1/mpconfigport.h.

[5] https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads

```
// Python internal features
#define MICROPY_ENABLE_GC           (1)
#define MICROPY_READER_VFS          (1)
#define MICROPY_HELPER_REPL         (1)
#include <stdint.h>

#include "mpconfigboard.h"
#include "mpconfigboard_common.h"
#include "NuMicro.h"
// options to control how MicroPython is built

// You can disable the built-in MicroPython compiler by setting the following
// config option to 0.  If you do this then you won't get a REPL prompt, but you
// will still be able to execute pre-compiled scripts, compiled with mpy-cross.
#define MICROPY_ENABLE_COMPILER     (1)

#define MICROPY_QSTR_BYTES_IN_HASH  (1)
#define MICROPY_QSTR_EXTRA_POOL     mp_qstr_frozen_const_pool
#define MICROPY_ALLOC_PATH_MAX      (256)
#define MICROPY_ALLOC_PARSE_CHUNK_INIT (16)
#define MICROPY_EMIT_X64            (0)
#define MICROPY_EMIT_THUMB          (0)
#define MICROPY_EMIT_INLINE_THUMB   (0)
#define MICROPY_COMP_MODULE_CONST   (0)
#define MICROPY_COMP_CONST          (0)
#define MICROPY_COMP_DOUBLE_TUPLE_ASSIGN (0)
#define MICROPY_COMP_TRIPLE_TUPLE_ASSIGN (0)
#define MICROPY_MEM_STATS           (0)
#define MICROPY_DEBUG_PRINTERS      (0)
#define MICROPY_GC_ALLOC_THRESHOLD  (0)
#define MICROPY_REPL_EVENT_DRIVEN   (0)
#define MICROPY_HELPER_LEXER_UNIX   (0)
#define MICROPY_ENABLE_SOURCE_LINE  (1)
#define MICROPY_ENABLE_DOC_STRING   (0)
#define MICROPY_ERROR_REPORTING     (MICROPY_ERROR_REPORTING_TERSE)
#define MICROPY_BUILTIN_METHOD_CHECK_SELF_ARG (0)
#define MICROPY_PY_ASYNC_AWAIT      (0)
#define MICROPY_PY___FILE__         (0)
#define MICROPY_PY_GC               (1)
#define MICROPY_PY_ARRAY            (1)
#define MICROPY_PY_ARRAY_SLICE_ASSIGN (1)
#define MICROPY_PY_ATTRTUPLE        (1)
#define MICROPY_PY_COLLECTIONS      (1)
#define MICROPY_PY_COLLECTIONS_DEQUE (1)
#define MICROPY_PY_COLLECTIONS_ORDEREDDICT (1)
#define MICROPY_PY_MATH             (1)
#define MICROPY_PY_CMATH            (1)
#define MICROPY_PY_IO               (1)
#define MICROPY_PY_IO_FILEIO        (1)
#define MICROPY_PY_STRUCT           (1)
//#define MICROPY_PY_SYS              (0)
#define MICROPY_PY_SYS_MAXSIZE      (1)
#define MICROPY_PY_SYS_EXIT         (1)
#define MICROPY_PY_SYS_STDFILES     (1)
#define MICROPY_PY_SYS_STDIO_BUFFER (1)
#ifndef MICROPY_PY_SYS_PLATFORM     // let boards override it if they want
#define MICROPY_PY_SYS_PLATFORM     "pyboard"
```

4.5 Enable/Disable I/O Class

The MicroPython defines I/O classes in pyb and machine modules. User can modify

mpconfigboard.h to enable/disable I/O classes or modify the pin definitions for individual I/O class. The I/O class support is disabled if the definition of I/O pin is deleted. When modifying the individual I/O pin, it needs to be consistent with the pin alternatives function definition file (xxx_af.csv). Below are some contents of M55M1/mpconfigboard.h and M55M1/board/m55m1_af.csv.

- mpconfigboard.h

```
#define MICROPY_HW_BOARD_NAME "NuMaker-M55M1"
#define MICROPY_HW_BOARD_NUMAKER_M55M1

// I2C busses
#define MICROPY_HW_I2C0_SCL (pin_H2) //CCAP connector
#define MICROPY_HW_I2C0_SDA (pin_H3) //CCAP connector

#define MICROPY_HW_I2C3_SCL (pin_G0)        //UNO_SCL
#define MICROPY_HW_I2C3_SDA (pin_G1)        //UNO_SDA

#define MICROPY_HW_I2C4_SCL (pin_C12)       //D5
#define MICROPY_HW_I2C4_SDA (pin_C11)       //D4

//UART
//#define MICROPY_HW_UART1_RXD  (pin_B2) //D0
//#define MICROPY_HW_UART1_TXD  (pin_B3) //D1
//#define MICROPY_HW_UART1_CTS  (pin_A1) //D12
//#define MICROPY_HW_UART1_RTS  (pin_A0) //D11

//#define MICROPY_HW_UART5_RXD  (pin_B4) //D6
//#define MICROPY_HW_UART5_TXD  (pin_B5) //D7
//#define MICROPY_HW_UART5_CTS  (pin_B2) //D0
//#define MICROPY_HW_UART5_RTS  (pin_B3) //D1

#define MICROPY_HW_UART6_RXD  (pin_C11) //D4
#define MICROPY_HW_UART6_TXD  (pin_C12) //D5
#define MICROPY_HW_UART6_CTS  (pin_C9)  //D2
```

- m55m1_af.csv

```
#Port,Pin,MFPL/MFPH,SPIM,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,EVENTOUT,
PortA,PA0,MFP0,GPIO0_GPIO,QSPI0_MOSI0,SPI0_MOSI,SD1_DAT0,SC0_CLK,UART0_RXD,UART1_nRTS,I2C2_SDA,CCAP0_DATA6,I3C0_SDA,BPWM0_CH0,EPWM0_CH5,EQEI3_B,DAC0_ST,PSIO0_CH7,UTCPD0_VCNEN1,LPSPI0_MOSI,LPUART0_RXD,LPIO0_LPIO,EVENTOUT,
PortA,PA1,MFP0,GPIO0_GPIO,QSPI0_MISO0,SPI0_MISO,SD1_DAT1,SC0_DAT,UART0_TXD,UART1_nCTS,I2C2_SCL,CCAP0_DATA7,I3C0_SCL,BPWM0_CH1,EPWM0_CH4,EQEI3_A,DAC1_ST,DMIC0_CH1_CLK,PSIO0_CH6,UTCPD0_DISCHG,UTCPD0_FRSTX1,LPSPI0_MISO,LPUART0_TXD,LPIO1_LPIO,EVENTOUT,
```

4. PortA,PA2,MFP0,GPIO0_GPIO,QSPI0_CLK,SPI0_CLK,SD1_DAT2,SC0_RST,UART4_RXD,UART1_RXD,I2C1_SDA,I2C0_SMBSUS,CCAP0_DATA2,BPWM0_CH2,EPWM0_CH3,EQEI3_INDEX,DMIC0_CH1_DAT,PSIO0_CH5,I3C0_PUPEN,UTCPD0_VBSRCEN,LPSPI0_CLK,EVENTOUT,
5. PortA,PA3,MFP0,GPIO0_GPIO,QSPI0_SS,SPI0_SS,SD1_DAT3,SC0_PWR,UART4_TXD,UART1_TXD,I2C1_SCL,I2C0_SMBAL,CCAP0_DATA3,BPWM0_CH3,EPWM0_CH2,EQEI0_B,EPWM1_BRAKE1,DMIC0_CH0_CLKLP,PSIO0_CH4,UTCPD0_VBSNKEN,LPSPI0_SS,EVENTOUT,
6. PortA,PA4,MFP1,GPIO0_GPIO,EBI0_AD1,QSPI0_MOSI1,SPI0_I2SMCLK,SD1_CLK,SC0_nCD,UART0_nRTS,UART5_RXD,I2C0_SDA,CANFD0_RXD,UART0_RXD,BPWM0_CH4,EPWM0_CH1,EQEI0_A,CCAP0_SCLK,DMIC0_CH0_CLK,I3C0_SDA,UTCPD0_VBSRCEN,LPUART0_RXD,LPUART0_nRTS,LPI2C0_SDA,EVENTOUT,
7. PortA,PA5,MFP1,GPIO0_GPIO,EBI0_AD0,QSPI0_MISO1,SPI1_I2SMCLK,SD1_CMD,SC2_nCD,UART0_nCTS,UART5_TXD,I2C0_SCL,CANFD0_TXD,UART0_TXD,BPWM0_CH5,EPWM0_CH0,EQEI0_INDEX,CCAP0_PIXCLK,DMIC0_CH0_DAT,I3C0_SCL,UTCPD0_VBSNKEN,LPUART0_TXD,LPUART0_nCTS,LPI2C0_SCL,EVENTOUT,
8. PortA,PA6,MFP1,GPIO0_GPIO,UTCPD0_DISCHG,EMAC0_RMII_RXERR,SPI1_SS,SD1_nCD,SC2_CLK,UART0_RXD,I2C1_SDA,QSPI1_MOSI1,EPWM1_CH5,BPWM1_CH3,ACMP1_WLAT,TM3_TM,INT0_INT,UTCPD0_VBSRCEN,KPI0_COL0,LPUART0_RXD,LPIO4_LPIO,EVENTOUT,
9. PortA,PA7,MFP1,GPIO0_GPIO,EBI0_AD7,EMAC0_RMII_CRSDV,SPI1_CLK,SC2_DAT,UART0_TXD,I2C1_SCL,QSPI1_MISO1,EPWM1_CH4,BPWM1_CH2,ACMP0_WLAT,TM2_TM,INT1_INT,UTCPD0_VBSNKEN,KPI0_COL1,LPUART0_TXD,LPIO5_LPIO,EVENTOUT,
10. PortA,PA8,MFP2,GPIO0_GPIO,EADC0_CH20,LPADC0_CH20,EBI0_ALE,SC2_CLK,SPI2_MOSI,SD1_DAT0,USCI0_CTL1,UART1_RXD,UART7_RXD,BPWM0_CH3,EQEI1_B,ECAP0_IC2,I2S1_DO,TM3_EXT,INT4_INT,EVENTOUT,
11. PortA,PA9,MFP2,GPIO0_GPIO,EADC0_CH21,LPADC0_CH21,EBI0_MCLK,SC2_DAT,SPI2_MISO,SD1_DAT1,USCI0_DAT1,UART1_TXD,UART7_TXD,BPWM0_CH2,EQEI1_A,ECAP0_IC1,I2S1_DI,TM2_EXT,SWDH0_DAT,EVENTOUT,
12. PortA,PA10,MFP2,GPIO0_GPIO,ACMP1_P0,EADC0_CH22,LPADC0_CH22,EBI0_nWR,SC2_RST,SPI2_CLK,SD1_DAT2,USCI0_DAT0,I2C2_SDA,UART6_RXD,BPWM0_CH1,EQEI1_INDEX,ECAP0_IC0,I2S1_MCLK,TM1_EXT,DAC0_ST,SWDH0_CLK,KPI0_ROW5,LPTM1_EXT,EVENTOUT,
13. PortA,PA11,MFP2,GPIO0_GPIO,ACMP0_P0,EADC0_CH23,LPADC0_CH23,EBI0_nRD,SC2_PWR,SPI2_SS,SD1_DAT3,USCI0_CLK,I2C2_SCL,UART6_TXD,BPWM0_CH0,EPWM0_SYNC_OUT,EPWM0_BRAKE1,I2S1_BCLK,TM0_EXT,DAC1_ST,KPI0_ROW4,LPTM0_EXT,EVENTOUT,

5 I/O CLASS QUICK REFERENCE

The MicroPython provides the document[6] to describe how to access MCU's I/O by python language, and, has description[7] to introduce how to add a module written in C. This document only provides the NuMicro® MCU's I/O quick reference for you reference.

5.1 Pin Class

All board pins are predefined as pyb.Pin.board.name. CPU pins corresponding to the board pins are available as pyb.Pin.cpu.Name. For example, on the NuMaker-M55M1 board, pyb.Pin.board.D0 corresponds to pyb.Pin.cpu.B2. Table 5-1 is the board pin name and CPU pin name mapping table.

Board	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	D0	B2
	D1	B3
	D2	C9
	D3	C10
	D4	C11
	D5	C12
	D6	B4
	D7	B5
	D8	I13
	D9	I12
	D10	A3
	D11	A0
	D12	A1
	D13	A2
	SCL	G0
	SDA	G1
	A0	B6

[6] http://docs.micropython.org/en/latest/library/index.html

[7] https://micropython-dev-docs.readthedocs.io/en/latest/adding-module.html

	A1	B7
	A2	B8
	A3	B9
	A4	B0
	A5	B1
	SS	A11
	CLK	A10
	MISO	A9
	MOSI	A8
	CCAP_SCL	H2
	CCAP_SDA	H3
	BTN0	I11
	BTN1	H1
	LEDR	H4
	LEDY	D6
	LEDG	D5
	CANFD_RX	J11
	CANFD_TX	J10
	DAC0_OUT	B12
	DAC1_OUT	B13

Table 5-1 Board Pin Name and CPU Pin Name Mapping

```
1.  from pyb import Pin
2.
3.  p_d0 = Pin(Pin.board.D0, Pin.OUT)     # create output pin on GPIO B2
4.  p_d0.value(1)                         # set pin to on/high
5.
6.  p_d1 = Pin(Pin.board.D1, Pin.IN)      # create input pin on GPIO B3
7.  print(p_d1.value())                   # get value, 0 or 1
8.
9.  Pin.board.D2.af_list()                # list available alternate functions on GPIO C9
10.
11. def btn1_callback(pin):               # define btn1 (button 1) callback
```

```
12.     print(pin)
13.
14. btn1 = Pin.board.BTN1
15. btn1.irq(handler=btn1_callback, trigger=Pin.IRQ_RISING)  # configure btn1 to interru
    pt
```

Pin(id, mode, [pull=Pin.PUL_NONE, alt=-1])

- mode can be one of:
  - Pin.IN – configure the pin for input
  - Pin.OUT – configure the pin for output, with push-pull control
  - Pin.OPEN_DRAIN – configure the pin for output, with open-drain control
  - Pin.QUASI – configure the pin for output, with quasi control
  - Pin.ALT – configure the pin for alternate function, push-pull
  - Pin.ALT_OPEN_DRAIN – configure the pin for alternate function, open-drain
- pull can be one of:
  - Pin.PULL_NONE – no pull up or down resistors
  - Pin.PULL_UP – enable the pull-up resistor
  - Pin.PULL – enable the pull-down resister
- When the mode is Pin.ALT or Pin.ALT_OPEN_DRAIN, alt can be the name of one of the alternate functions associated with a pin. You can use af_list() to query available alternate functions for this pin.

pin.irq([handler=None, trigger=Pin.IRQ_FALLING|Pin.IRQ_RIGING])

Configure GPIO pins to interrupt on external. If a falling or rising edge seen on this pin, the handler will be called.

5.2 ADC Class

The NuMicroPy uses EADC0 to implement ADC class. Table 5-2 is ADC channel support list by the board.

Board	Channel Number	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	CH0	A4	B0
	CH1	A5	B1
	CH2	D0	B2
	CH3	D1	B3
	CH4	D6	B4
	CH5	D7	B5
	CH6	A0	B6
	CH7	A1	B7
	CH8	A2	B8
	CH9	A3	B9
	CH20	MOSI	A8

	CH21	MISO	A9
	CH22	CLK	A10
	CH23	SS	A11

Table 5-2 ADC Support List

```
from pyb import ADC, Pin, ADCALL

adc0 = ADC(Pin.board.A4)             # create an analog object from a pin
val = adc0.read()                    # read an analog value

adc_all = ADCALL(12, 0x6000000)      # 12 bit resolution, internal channel
25(core_temp) and channel 26(VBAT)
adc_all.read_core_temp()           # read MCU's temperature
adc_all.read_core_vbat()           # read MCU's VBAT
```

5.3 SPI Class

In the MicroPython, there are two SPI drivers. One is implemented in software and works on all pins, and is accessed via the machine.SPI class. The other is implemented in hardware and accessed via the pyb.SPI class. Table 5-3 is hardware SPI support list by the board.

Board	SPI Pin Name	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	SPI0_NSS	D10	A3
	SPI0_SCK	D13	A2
	SPI0_MISO	D12	A1
	SPI0_MOSI	D11	A0
	SPI2_NSS	SS	A11
	SPI2_SCK	CLK	A10
	SPI2_MISO	D4	A9
	SPI2_MOSI	D5	A8

Table 5-3 SPI Support List

```
from pyb import SPI

# construct an SPI bus on the SPI0
# mode is Master
# polarity is the idle state of SCK
# phase=0 means sample on the first edge of SCK, phase=1 means the second
spi = SPI(0, SPI.MASTER, baudrate=100000, polarity=1, phase=0)

spi.read(10)            # read 10 bytes on MISO
spi.read(10, 0xff)      # read 10 bytes while outputing 0xff on MOSI

buf = bytearray(50)     # create a buffer
spi.readinto(buf)       # read into the given buffer (reads 50 bytes in this case)
```

```
spi.readinto(buf, 0xff) # read into the given buffer and output 0xff on MOSI

spi.write(b'12345')     # write 5 bytes on MOSI

buf = bytearray(4)      # create a buffer
spi.write_readinto(b'1234', buf) # write 4 bytes to MOSI and read from MISO into the buffer
```

SPI(bus, mode, [baudrate=328125, polarity=1, phase=0, bits=8, firstbit=SPI.MSB])

- mode must be either SPI.MASTER ore SPI.SLAVE
- baudrate is the SCK clock rate(only sensible for master)
- polarity can be 0 or 1, and is the level the idle clock line sits at.
- pahse can be 0 or 1 to sample date on the first or second clock edge respectively.
- bits can be 8 or 16 or 32
- firstbit can be SPI.MSB or SPI.LSB

5.4 I²C Class

Table 5-4 is hardware I²C support list by the board.

Board	I2C Pin Name	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	I2C0_SCL	CCAP_SCL	H2
	I2C0_SDA	CCAP_SDA	H3
	I2C3_SCL	SCL	G0
	I2C3_SDA	SDA	G1
	LPI2C0_SCL	D5	C12
	LPI2C0_SDA	D4	C11

Table 5-4 I²C Support List

```
from pyb import I2C
# master mode
i2c = I2C(3, I2C.MASTER)     # create and initiate I2C3 as a master
i2c.scan()        # scan for slaves on the bus, returning a list of valid addresses. Only valid when in master mode.
i2c.is_ready(0x42)           # check if slave 0x42 is ready
i2c.send('123', 0x42)        # send 3 bytes to slave with address 0x42
data = bytearray(3)          # create a buffer
i2c.recv(data, 0x42)         # receive 3 bytes from slave with address 0x42, writing them into data
i2c.deinit()                 # turn off the peripheral

# slave mode. Only for M480 series
i2c = I2C(3, I2C.SLAVE, addr=0x12)  # create and initiate I2C3 as a slave
i2c.send('123')             # send 3 bytes to master
i2c.recv(data)              # receive data from master
```

I2C(bus, mode, [addr=-0x12, baudrate=100000, gencall=False])

- mode must be either I2C.MASTER or I2C.SLAVE
- addr is the 7-bit address (only sensible for a slave)
- baudrate is the SCL clock rate (only sensible for a master)
- gencall is whether to support general call mode.

5.5 RTC Class

```
from pyb import RTC

rtc = RTC()
rtc.datetime((2025, 8, 23, 1, 12, 48, 0, 0)) # set a specific date and time
rtc.datetime() # get date and time

def rtc_cb()   # define RTC wakeup callback function
    print(“wake up)

rtc.wakeup(rtc.WAKEUP_1_SEC, rtc_cb) # set the RTC wakeup timer to trigger repeatedly at every timeout
```

RTC.wakeup(timeout, callback)

- timeout can be one of:
  - RTC.WAKEUP_1_SEC: 1 second
  - RTC.WAKEUP_1_2_SEC: 1/2 second
  - RTC.WAKEUP_1_4_SEC: 1/4 second
  - RTC.WAKEUP_1_8_SEC: 1/8 second
  - RTC.WAKEUP_1_16_SEC: 1/16 second
  - RTC.WAKEUP_1_32_SEC: 1/32 second
  - RTC.WAKEUP_1_64_SEC: 1/64 second
  - RTC.WAKEUP_1_128_SEC: 1/128 second

5.6 UART Class

Table 5-5 is UART support list by the board.

Board	UART Pin Name	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	UART6_RXD	D4	C11
	UART6_TXD	D5	C12
	UART6_CTS	D2	C9
	UART6_RTS	D3	C10

Table 5-5 UART Support List

```
from pyb import UART

uart = UART(6, 9600, bits=8, parity=None, stop=1) # initiate UART6
uart.read(10)       # read 10 characters, 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 3 characters
uart.readchar()     # read 1 character and returns it as an integer
uart.writechar(42)  # write 1 character
uart.any()          # returns the number of characters waiting
uart.deinit()       # turn off the UART bus
```

UART(bus, buadrate, [bits=8, parity=None, stop=1, timeout=2000, flow=0, timeout_char=0, read_buf_len=64])

- baudrate is the clock rate.

- bits is the number of bits per character, 5, 6, 7 or 8.
- parity is the parity, None, 0(even) or 1(odd).
- stop is the number of stop bits, 1 or 2.
- flow sets the flow control type. Can be 0, UART.RTS, UART.CTS or UART.RTS | UART.CTS.
- timeout is the timeout in milliseconds to wait for writing/reading the first character.
- timeout_char is the timeout in milliseconds to wait between characters while writing or reading.
- read_buf_len is the character length of the read buffer(0 to disable).

5.7 CAN Class

Table 5-6 is CAN support list by the board.

Board	CAN Pin Name	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	CANFD0_RXD	CANFD_RX	J11
	CANFD0_TXD	CANFD_TX	J10

Table 5-6 CAN Support List

```
1.  from pyb import CAN
2.
3.  can = CAN(0, mode=CAN.NORMAL, extframe=True, baudrate=500000) # create and initiate
    an object on CAN1
4.  can.setfilter(id=0x55, mask=0xf0)  # set a filter to receive messages with id=0x55 a
    nd mask is 0xf0
5.  can.send('message!', id=0x50)   # send a message with id 0x50
6.
7.  buf = bytearray(8)
8.  data_lst = [0, 0, 0, memoryview(buf)]
9.
10. def can_cb1(bus, reason, fifo_num):
11.     if reason == CAN.CB_REASON_RX:
12.         bus.recv(list = data_lst)
13.         print(data_lst)
14.     if reason == CAN.CB_REASON_ERROR_WARNING:
15.         print('Error Warning')
16.     if reason == CAN.CB_REASON_ERROR_PASSIVE:
17.         print('Error Passive')
18.     if reason == CB_REASON_ERROR_BUS_OFF:
19.         print('Bus off')
20.
21. can.rxcallback(can_cb1) # register a callback when a message is accepted into FIFO
```

CAN(bus, [mode=CAN.NORMAL, extframe=True, baudrate=500000])

- mode is one of: CAN.NORMAL, CAN.LOOPBACK, CAN.SILENT and CAN.SILENT_LOOPBACK.
- If extframe is True then the bus uses extended identifiers in the frames.
- baudrate is the clock rate

can.setfilter(id=0, mask=0)

- id is the identifier of the frame that will be received.
- mask is the identifier mask used for a acceptance filtering.

can.recv(list=None, timeout=5000)

- list is an option list object to be used as the return value.
- timeout is the timeout in milliseconds to wait for receive.

Return value: A tuple containing four fields.

- The id of the message.

- A boolean that indicates if the message is an RTP message.
- Always 0
- An array containing the data

5.8 Timer Class

Each timer consists of a counter that counts up at a certain rate. When the counter reaches the timer period it triggers an event, and the counter reset back to zero. By using the callback method, the timer event can call a Python function.

```
1.  from pyb import Pin
2.  from pyb import Timer
3.
4.  def tick(timer):
5.      print(timer.counter())
6.
7.  tim = Timer(3, freq = 2)  # create a timer object using timer 3 and trigger at 2Hz
8.  tim.callback(tick)  # set the function to be called when the timer triggers
9.
10. # configure timer to be a PWM, output compare, or input capture channel
11. chan = tim.channel(Timer.PWM, pin = Pin.board.D0, pulse_width_percent = 20)
12. chan.callback(tick)  # set the function to be called when the timer channel triggers

13.
14. chan.capture()  # get the capture associated with a input capture channel
15.
16. chan.compare(100)  # set compare value associcated with a channel
17. chan.compare()  # get the compare value associated with a channel
18.
19. chan.pulse_width_percent(50)  # set the pulse width percentage associated with a PWM
     channel
20. chan.pulse_width_percent()  # get the pulse width percentage associated with a PWM
    channel
```

Timer(id, freq=0, [prescaler=-1, period=-1, mode=Timer.PERIODIC, callback=None])

- freq specifies the periodic frequency of timer.
- prescaler specifies the value to be loaded into the timer's prescale counter register.
- period specifies the value to be loaded into the timer's comparator register.
- mode specifies the timer's operation mode. It can be one of Timer.ONESHOT, Timer.PERIODIC or Timer.CONTINUOUS.
- callback specifies the function to be called when the timer triggers.

timer.channel(mode, pin=None, [callback=None, pulse_width_percent=50, polarity=-1])

- mode can be one of:
  - Timer.PWM: configure the timer in PWM mode.
  - Timer.OC_TOGGLE: the pin will be toggled when a compare match occurs.
  - Timer.IC: configure the timer in Input Capture mode.
- pin is a Pin object. If specified this will cause the alternate function of the indicated pin to be configured for this timer channel.
- callback specifies the function to be called when the timer channel triggers.

For Timer.PWM mode:

- pulse_width_percent determines the initial pulse width percentage to use.

For Timer.IC mode:

- polarity can be one of:
  - Timer.RISING: captures on rising edge.
  - Timer.FALLING: captures on falling edge.
  - Timer.BOTH: captures on both edge.

5.9 PWM Class

The NuMicroPy supports BPWM and EPWM for PWM class. Table 5-7 is PWM pin support list by the board.

Board	PWM Pin Name	Board Pin Name	CPU Pin Name
NuMaker-M55M1 NuMaker-M5531	BPWM0_CH0	D11	A0
		SS	A11
	BPWM0_CH1	D12	A1
		CLK	A10
	BPWM0_CH2	D13	A2
		MISO	A9
	BPWM0_CH3	D10	A3
		MOSI	A8
	BPWM1_CH2	A3	B9
	BPWM1_CH3	A2	B8
	BPWM1_CH4	A1	B7
	BPWM1_CH5	A0	B6
	EPWM0_CH0	D7	B5
		SDA	G1
	EPWM0_CH1	D6	B4
		SCL	G0
	EPWM0_CH2	D10	A3
		D1	B3
	EPWM0_CH3	D13	A2
		D0	B2
	EPWM0_CH4	D12	A1
		A5	B1
	EPWM0_CH5	D11	A0
		A4	B0
	EPWM1_CH0	D5	C12
		D9	I12
	EPWM1_CH1	D4	C11
		D8	I13
	EPWM1_CH2	D3	C10

	EPWM1_CH3	D2	C9
	EPWM1_CH4	A5	B1
		A1	B7
	EPWM1_CH5	A4	B0
		A0	B6

Table 5-7 PWM Pin Support List

```
from pyb import PWM
from pyb import Pin

def capture_cb(chan, reason):
    if reason == PWM.RISING:
        print('rising edge')
    elif reason == PWM.FALLING:
        print('falling edge')
    else:
        print('both edge')


bpwm1 = PWM(1, freq = 2)           # create BPWM1 object
epwm0 = PWM(0, PWM.EPWM, freq = 8) #create EPWM0 object

# configure bpwm 1 channel 4 to be a output channel. Board pin A1, CPU pin B7
bpwm1ch4 = bpwm1.channel(mode = PWM.OUTPUT, pulse_width_percent = 50, pin = Pin.board.A1)

# configure epwm 0 channel 1 to be a capture channel. Board pin D6, CPU pin B4
epwm0ch1 = epwm0.channel(mode = PWM.CAPTURE, capture_edge = PWM.RISING, pin = Pin.board.D6, callback = capture_cb)

bpwm1ch4.pulse_width_percent(50)  # set the pulse width percentage associated with a PWM channel
bpwm1ch4.pulse_width_percent()    # get the pulse width percentage associated with a PWM  channel

epwm0ch1.capture(PWM.RISING)  # get the capture data associated with a input capture channel
epwm0ch1.disable()  # disable channel
```

PWM(id, type, freq=0)

- type can be on of:
  - PWM.BPWM: create a BPWM object
  - PWM.EPWM: create a EPWM object
- freq specifies the periodic frequency of PWM.

pwm.channel(mode =0, pin= None, [callback=None, pulse_width_percent=50, capture_edge=PWM.RISING, freq=0, complementary=False])

- mode can be on of:
  - PWM.OUTPUT: configure the channel to PWM output mode.
  - PWM.CAPTURE: configure the channel to input capture mode.
- pin is a Pin object. If specified this will cause the alternate function of the indicated pin to be configured for this PWM channel.

For capture mode:

- callback specifies the function to be called when the PWM capture channel triggered by rising or falling edge.

- capture_edge can be one of:
  - PWM.RISING: captures on rising edge.
  - PWM.FALLING: captures on falling edge.
  - PWM.RISING_FALLING: captures on both edge.

For output mode:

- pulse_width_percent determines the initial pulse width percentage to use for PWM output channel.

For EPWM channel:

- freq specifies the periodic frequency of EPWM individual channel.
- complementary enable/disable EPWM channel complementary mode.

5.10 WDT Class

```
1.  from pyb import WDT
2.
3.  wdt = WDT(timeout = 1000)   # start watchdog timer
4.  wdt.feed()       # reset watchdog timer counter
```

WDT(timeout=5000)

- timeout specifies the time-out interval and the interval is 2~ 26000ms

6 SUMMARY

The MicroPython is a Python programming language interpreter that runs on the small embedded system. With MicroPython you can write clean and simple Python code to control hardware instead of having to use complex low-level languages like C or C++.

The MicroPython is only a programming language interpreter and does not include an IDE, but you can write code in your desired text editor and then use Copy and Paste (file access) to upload and run the code on a board.

One disadvantage of interpreted code is less performance and sometimes more memory usage when interpreting code.

7 TROUBLESHOOTING

Correct some problems that maybe encountered in use.

7.1 Meet “MemoryError: memory allocation failed” message

It caused by the default heap size of MicroPython is not enough for your python code. Open main.c and modify the value of “MP_TASK_HEAP_SIZE”.

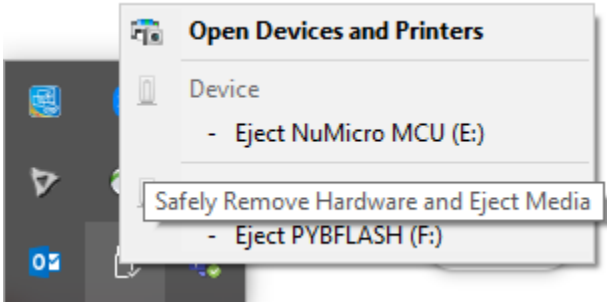
```
#define MP_TASK_PRIORITY        (configMAX_PRIORITIES - 1)
#define MP_TASK_STACK_SIZE      (4 * 1024)
#define MP_TASK_STACK_LEN       (MP_TASK_STACK_SIZE / sizeof(StackType_t))

#ifndef MP_TASK_HEAP_SIZE
#define MP_TASK_HEAP_SIZE     (28 * 1024)
#endif
```

Rebuild and update firmware.

7.2 After updating the file with editor or tool in PYBFLASH disk, restarting and opening the file again will fail.

Maybe your editor or tool did not flush out the file internal cache buffer when file archiving. Please using windows “Safely remove hardware” to safely remove PYBFLASH disk.



8 REVISION HISTORY

Date	Revision	Description
2025.06.24	2.01	1. Support M55M1/M5531

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.