Arm® Cortex®-M

32-bit Microcontroller

NuMicro® Family NuMicroPy User Manual

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

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
M487	NuMaker-PFM-M487	362KB	77KB
M487	NuMaker-IOT-M487	373KB	77KB

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.

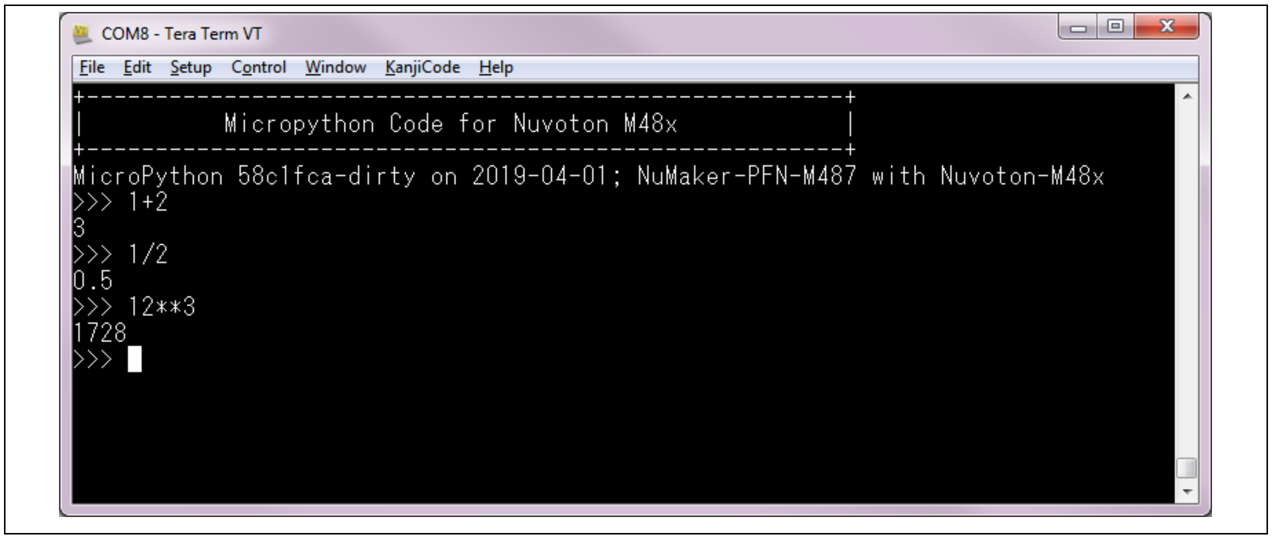
The MicroPython defines the I/O classes associated with the MCU peripheral in the pyb module[3]. NuMicroPy implements these I/O classes according to these definitions.

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 UART0 serial peripheral. The baud rate of the REPL is 115200. In the NuMaker-PFM-M487 board, Nu-Link-Me has a USB-to-serial converter on it then 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.



```
+--------------------------------------------------------+
|          Micropython Code for Nuvoton M48x             |
+--------------------------------------------------------+
MicroPython 58c1fca-dirty on 2019-04-01; NuMaker-PFN-M487 with Nuvoton-M48x
>>> 1+2
3
>>> 1/2
0.5
>>> 12**3
1728
>>>
```

Figure 2-1 REPL Mode

2.2 Embedded Flash Partition

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. The firmware will attempt to mount the data partition 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

[3] pyb module: MicroPython board related module.

force the firmware into USB mass storage mode and then write your python code to these files. Table 2-1 shows the default embedded Flash partition status.

MCU	Embedded Flash Size	Firmware Partition Start Address	Firmware Partition Size	Data Partition Start Address	Data Partition Size
M487	512KB	0x0	384KB	0x60000	128KB

Table 2-1 Default Embedded Flash Partition Status

2.3 Modules and I/O Classes Support List

Table 2-2 and Table 2-3 show the M487 support status on the modules and I/O classes of MicroPython.

Module	Description	M487
array	Arrays of numeric data	✓
cmath	Mathematical function for complex numbers	✓
gc	Control the garbage collector	✓
math	Mathematical functions	✓
sys	System specific functions	✓
ubinascii	Binary/ASCII conversions	✓
ucollections	Collection and container types	✓
uerrno	System error codes	✓
uhashlib	Hashing algorithms	✓
uheapq	Heap queue algorithm	✓
uio	Input/output streams	✓
ujson	JSON encoding and decoding	✓
uos	Basic “operating system” service	✓
ure	Simple regular expressions	✓
uselect	Wait for events on a set of streams	✓
usocket	Socket module	✓
ussl	SSL/TLS module	✓
ustruct	Pack and unpack primitive data types	✓

utime	Time related functions	✓
uzlib	zlib decompression	✓
_thread	Multithreading support	✓
network	Network configuration	✓
uctypes	Access binary data in a structured way	✓
Machine	Functions related to the hardware	✓
ucryptolib	Cryptographic ciphers	✓

Table 2-2 Default Supported Modules

I/O Class	Description	M487
ADC	Analog to digital conversion	✓
CAN	Controller area network communication bus	✓
I^{2}C	A two-wire serial protocol	✓
LED	LED object	✓
Pin	Control I/O pins	✓
PinAF	Pin alternate functions	✓
RTC	Real time clock	✓
SPI	A master-driven serial protocol	✓
Switch	Switch button object	✓
Timer	Control internal timers	✓
TimerChannel	Setup a channel for a timer	✓
UART	Duplex serial communication bus	✓
USB_HID	USB Human Interface Device	✓
USB_VCP	USP virtual com port	✓
PWM	BPWM/EPWM generator and capture timer	✓
WDT	Watchdog timer	✓
Accel	Accelerometer control (Bosch BMX055)	Only for NuMaker-IOT-M487

Gyro	Gyroscope control (Bosch BMX055)	Only for NuMaker-IOT-M487
Mag	Magnetometer control (Bosch BMX055)	Only for NuMaker-IOT-M487

Table 2-3 Default Supported I/O Classes

3 HOW TO START NUMICROPY

The following uses the NuMaker-PFM-M487 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 “Nu-Link_Command_Tool” file. When the Nu-Link command tool has been downloaded successfully, please unzip the file and execute the “NuMicro NuLink Command Tool.exe” to install the driver.

3.2 Hardware Setup Steps

1. Turn on ICE function switch pin 1, 2, 3 and 4.
2. Connect USB ICE and USB1.1 to PC (Figure 3-1).
3. Set up your terminal program (Figure 3-2 and Figure 3-3).

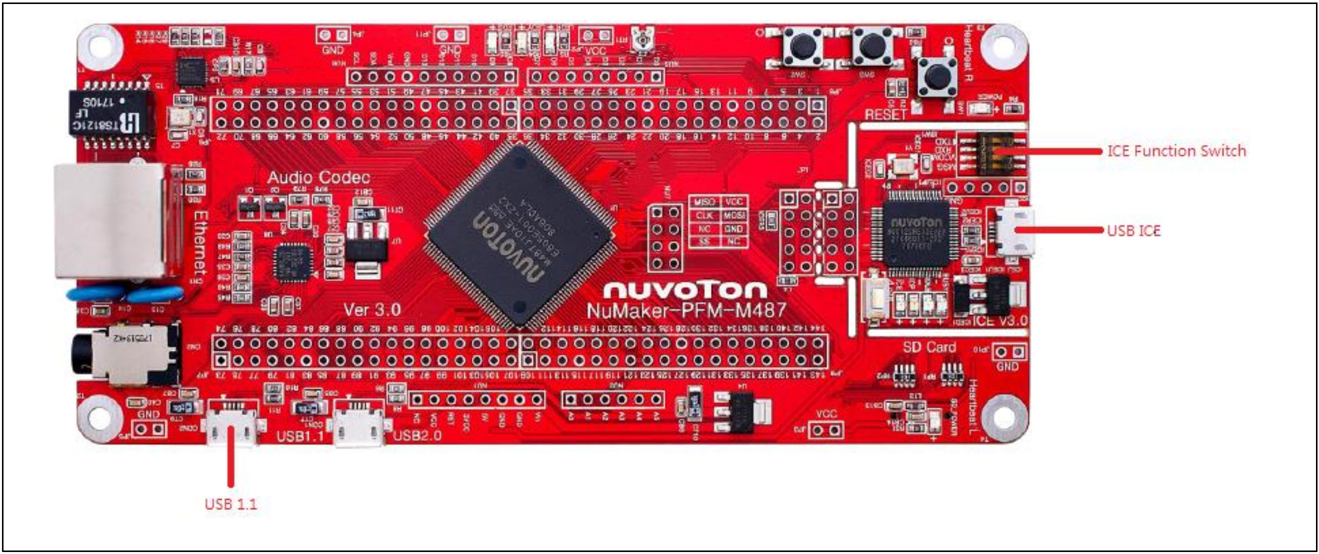
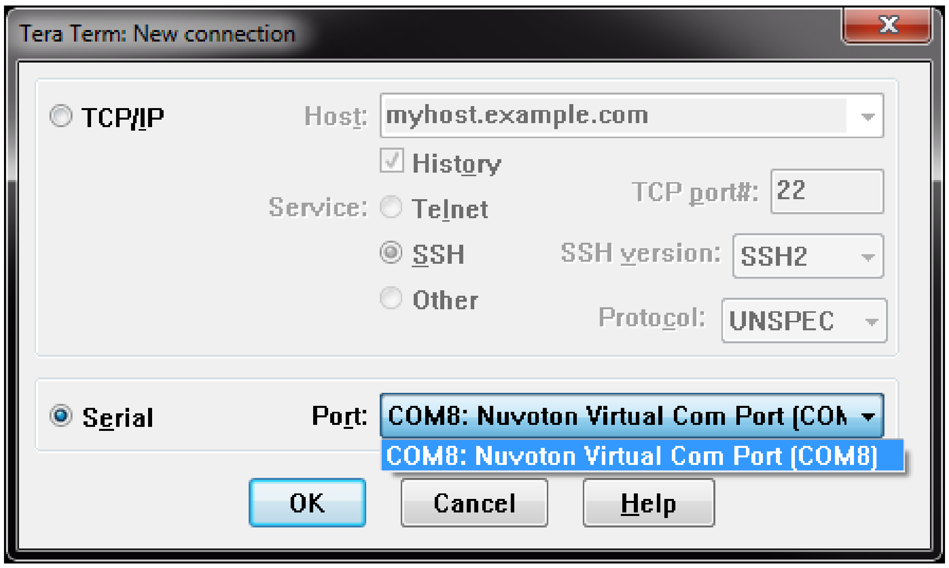
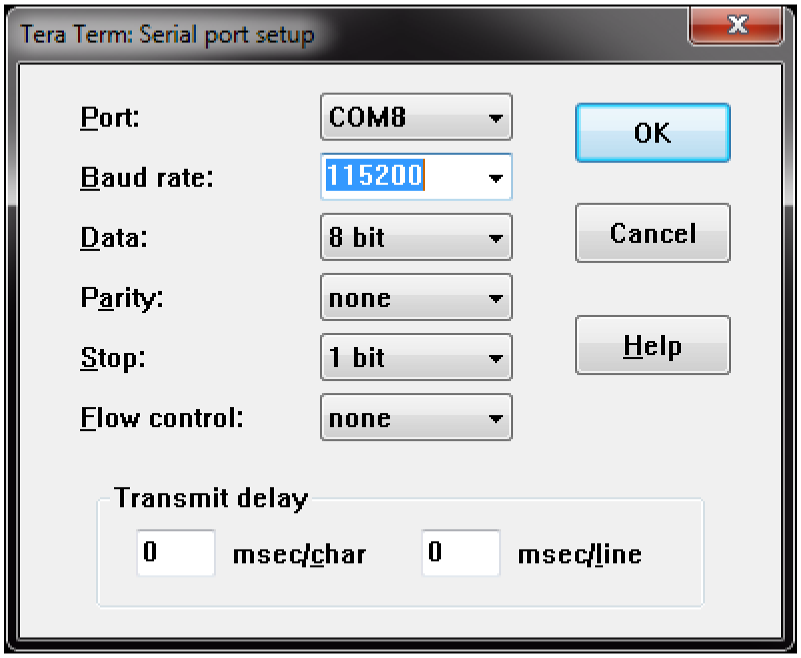


Figure 3-1 NuMaker-PFM-M487 Board



TCP/IP
Host: myhost.example.com
History
Service: Telnet / [✓ SSH] / Other
TCP port#: 22
SSH version: SSH2
Protocol: UNSPEC
[✓ Serial] Port: COM8: Nuvoton Virtual Com Port (COM8)

Figure 3-2 Create a Serial Connection



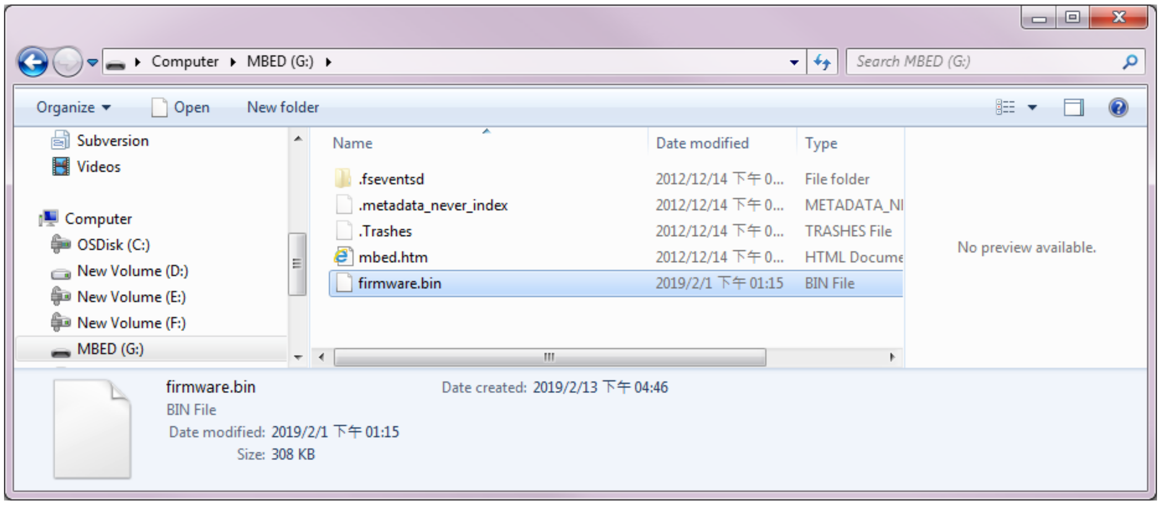
Port: COM8
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay
0 msec/char 0 msec/line

Figure 3-3 Serial Port Setup

3.3 Burn Firmware

The Nu-Link-Me exports a MBED disk, just Copy and Paste your firmware.bin into MBED disk (Figure 3-4). 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].



Name	Date modified	Type
.fseventsd	2012/12/14 下午 0...	File folder
.metadata_never_index	2012/12/14 下午 0...	METADATA_NI
.Trashes	2012/12/14 下午 0...	TRASHES File
mbed.htm	2012/12/14 下午 0...	HTML Docume
firmware.bin	2019/2/1 下午 01:15	BIN File

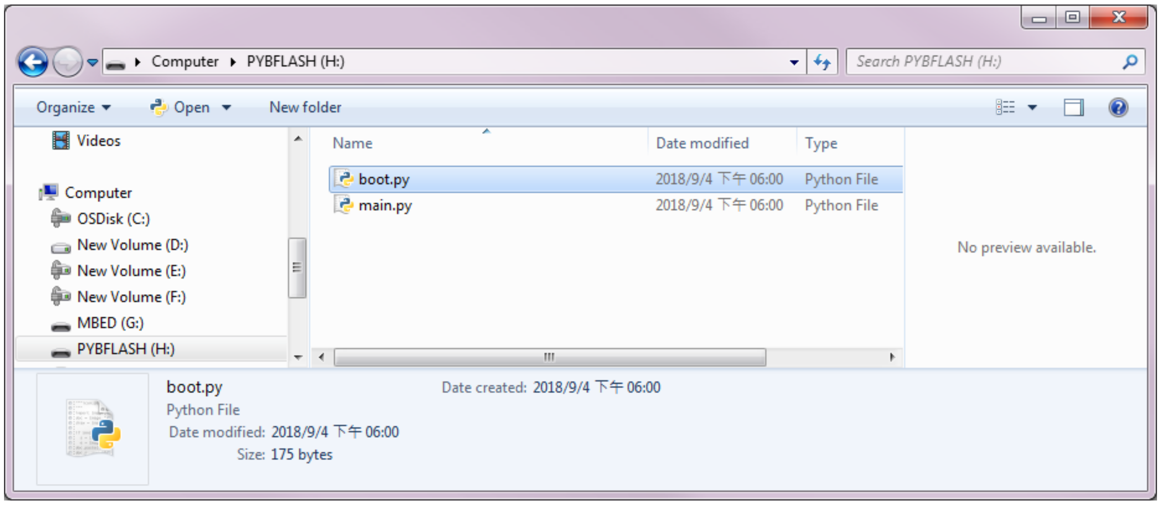
firmware.bin
BIN File
Date modified: 2019/2/1 下午 01:15
Size: 308 KB
Date created: 2019/2/13 下午 04:46

Figure 3-4 Copy and Paste Firmware

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

3.4 Python Code Update Steps

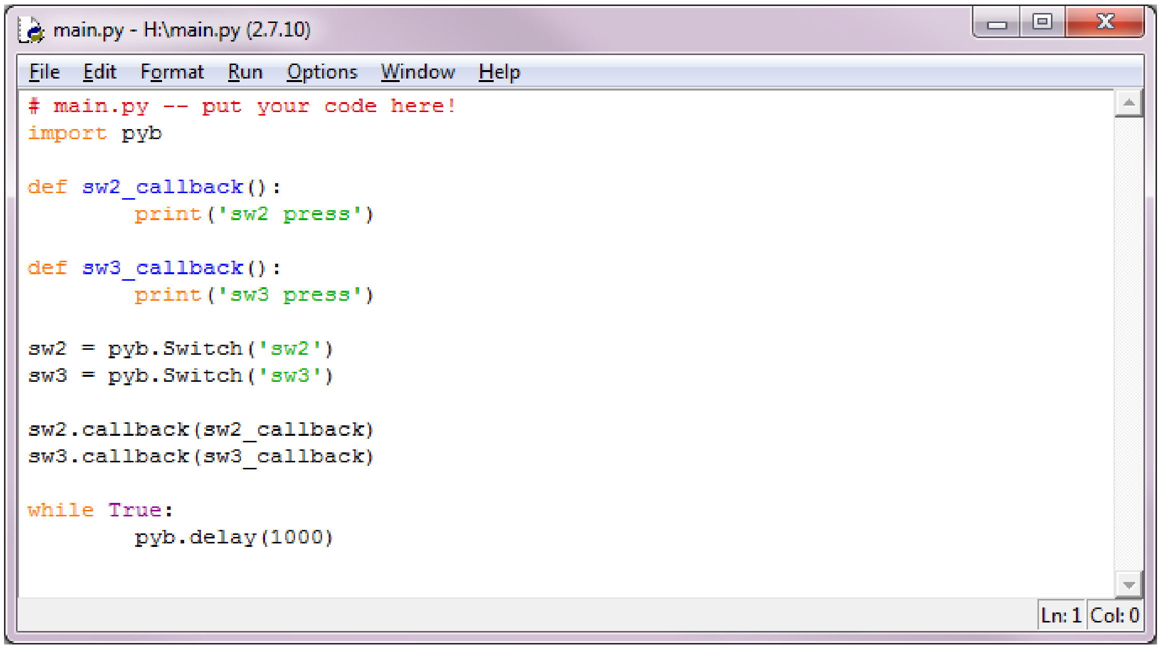
1. Connect USB1.1 to PC
2. Press the SW2 and RESET button at the same time. The firmware will export a PYBFLASH disk (Figure 3-5).
3. Update your python code to boot.py or main.py (Figure 3-6).
4. Press the RESET button (Figure 3-7).



Name	Date modified	Type
boot.py	2018/9/4 下午 06:00	Python File
main.py	2018/9/4 下午 06:00	Python File

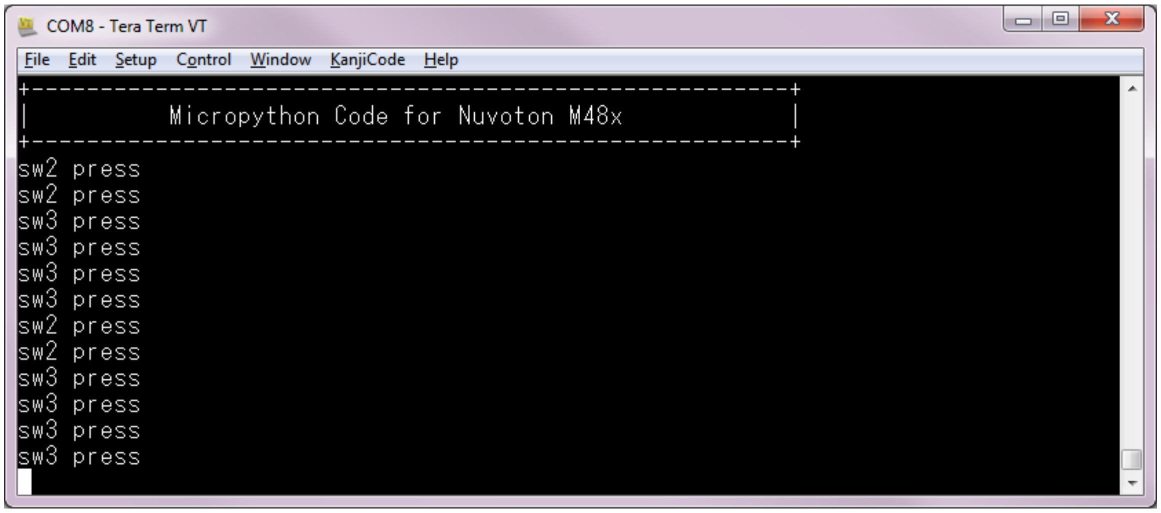
boot.py
Python File
Date modified: 2018/9/4 下午 06:00
Size: 175 bytes
Date created: 2018/9/4 下午 06:00

Figure 3-5 PYBFLASH Disk


main.py - H:\main.py (2.7.10)
File Edit Format Run Options Window Help
Ln: 1 Col: 0

```
# main.py -- put your code here!
import pyb

def sw2_callback():
        print('sw2 press')

def sw3_callback():
        print('sw3 press')

sw2 = pyb.Switch('sw2')
sw3 = pyb.Switch('sw3')

sw2.callback(sw2_callback)
sw3.callback(sw3_callback)

while True:
        pyb.delay(1000)
```

Figure 3-6 Write Switch Button Example Code



```
+----------------------------------------------------------+
|          Micropython Code for Nuvoton M48x               |
+----------------------------------------------------------+
sw2 press
sw2 press
sw3 press
sw3 press
sw3 press
sw3 press
sw2 press
sw2 press
sw3 press
sw3 press
sw3 press
sw3 press
```

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 16.04.

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

To install these packages, use the following command.

```
sudo apt-get install build-essential libreadline-dev libffi-dev git pkg-config
```

4.2 Install GNU Arm Toolchain

Download GNU Arm toolchain linux 64-bit version 7-2018-q2 update from Arm Developer [5] website.

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

```
mv gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2 /usr/local/
cd /usr/local
tar -xjvf gcc-arm-none-eabi-7-2018-q2-update-linux.tar.bz2
```

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

4.3 Build Firmware

To build MicroPython firmware for M487, use the following command.

```
git clone --recursive https://github.com/OpenNuvoton/NuMicroPy.git
cd patch
./run_patch.sh
cd ../M48x
make V=1
```

4.4 Enable/Disable Module

Default modules support list as Table 2-2. 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 M48x/mpconfigport.h.

```
#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)

```

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

```
#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  (0)
#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_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"
#endif

#define MICROPY_MODULE_FROZEN_MPY   (1)
#define MICROPY_CPYTHON_COMPAT      (0)
#define MICROPY_LONGINT_IMPL        (MICROPY_LONGINT_IMPL_NONE)
#define MICROPY_FLOAT_IMPL          (MICROPY_FLOAT_IMPL_FLOAT)
#define MICROPY_PY_UERRNO           (1)

// control over Python builtins
#define MICROPY_PY_BUILTINS_BYTEARRAY (1)
#define MICROPY_PY_BUILTINS_MEMORYVIEW (1)
#define MICROPY_PY_BUILTINS_ENUMERATE (0)
#define MICROPY_PY_BUILTINS_FILTER  (0)
#define MICROPY_PY_BUILTINS_FROZENSET (0)
#define MICROPY_PY_BUILTINS_REVERSED (0)
#define MICROPY_PY_BUILTINS_SET     (0)
#define MICROPY_PY_BUILTINS_SLICE   (0)
#define MICROPY_PY_BUILTINS_PROPERTY (0)
#define MICROPY_PY_BUILTINS_MIN_MAX (0)



```

```
// Python internal features
#define MICROPY_ENABLE_GC           (1)
#define MICROPY_READER_VFS          (1)
#define MICROPY_HELPER_REPL         (1)
```

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 M48x/mpconfigboard.h and M48x/board/m487_af.csv.

- mpconfigboard.h

```
// I2C busses
#define MICROPY_HW_I2C0_SCL (pin_A5)
#define MICROPY_HW_I2C0_SDA (pin_A4)
#define MICROPY_HW_I2C1_SCL (pin_A3)
#define MICROPY_HW_I2C1_SDA (pin_A2)


// SPI busses
#define MICROPY_HW_SPI0_NSS  (pin_A3) //D10
#define MICROPY_HW_SPI0_SCK  (pin_A2) //D13
#define MICROPY_HW_SPI0_MISO (pin_A1) //D12
#define MICROPY_HW_SPI0_MOSI (pin_A0) //D11

#define MICROPY_HW_SPI3_NSS  (pin_C9) //D2
#define MICROPY_HW_SPI3_SCK  (pin_C10) //D3
#define MICROPY_HW_SPI3_MISO (pin_B9) //A3
#define MICROPY_HW_SPI3_MOSI (pin_B8) //A2

//ADC(using EADC pin to implement ADC class)
#define MICROPY_HW_EADC0_CH0  (pin_B0) //10
#define MICROPY_HW_EADC0_CH1  (pin_B1) // 9
#define MICROPY_HW_EADC0_CH2  (pin_B2) // 4
#define MICROPY_HW_EADC0_CH3  (pin_B3) // 3
//#define MICROPY_HW_EADC0_CH4  (pin_B0)//not implement yet
//#define MICROPY_HW_EADC0_CH5  (pin_B0)//not implement yet
#define MICROPY_HW_EADC0_CH6  (pin_B6)//144
#define MICROPY_HW_EADC0_CH7  (pin_B7)//143
#define MICROPY_HW_EADC0_CH8  (pin_B8)//142 //A2
#define MICROPY_HW_EADC0_CH9  (pin_B9)//141 //A3
//#define MICROPY_HW_EADC0_CH10 (pin_B0)//
//#define MICROPY_HW_EADC0_CH11 (pin_B0)//
//#define MICROPY_HW_EADC0_CH12 (pin_B0)//
//#define MICROPY_HW_EADC0_CH13 (pin_B0)//
//#define MICROPY_HW_EADC0_CH14 (pin_B0)//
```

- m487_af.csv

```
#Port,Pin,MFPL/MFPH,SPIM,,,,,,,,,,,,,,,,,,,,,,EVENTOUT,
PortA,PA0,MFPL,SPIM0_MOSI,QSPI0_MOSI0,SPI0_MOSI,SD1_DAT0,SC0_CLK,UART0_RXD,UART1_nRT
S,I2C2_SDA,BPWM0_CH0,EPWM0_CH5,DAC0_ST,EVENTOUT,
PortA,PA1,MFPL,SPIM0_MISO,QSPI0_MISO0,SPI0_MISO,SD1_DAT1,SC0_DAT,UART0_TXD,UART1_nCT
S,I2C2_SCL,BPWM0_CH1,EPWM0_CH4,DAC1_ST,EVENTOUT,
PortA,PA2,MFPL,SPIM0_CLK,QSPI0_CLK,SPI0_CLK,SD1_DAT2,SC0_RST,UART4_RXD,UART1_RXD,I2C
1_SDA,BPWM0_CH2,EPWM0_CH3,EVENTOUT,
PortA,PA3,MFPL,SPIM0_SS,QSPI0_SS,SPI0_SS,SD1_DAT3,SC0_PWR,UART4_TXD,UART1_TXD,I2C1_S
CL,BPWM0_CH3,EPWM0_CH2,QEI0_B,EVENTOUT,
PortA,PA4,MFPL,SPIM0_D3,QSPI0_MOSI1,SPI0_I2SMCLK,SD1_CLK,SC0_nCD,UART0_nRTS,UART5_RX
D,I2C0_SDA,CAN0_RXD,BPWM0_CH4,EPWM0_CH1,QEI0_A,EVENTOUT,
PortA,PA5,MFPL,SPIM0_D2,QSPI0_MISO1,SPI1_I2SMCLK,SD1_CMD,SC2_nCD,UART0_nCTS,UART5_TXD,I2C0_SCL,CAN0_TXD,BPWM0_CH5,EPWM0_CH0,QEI0_INDEX,EVENTOUT,

PortB,PB0,MFPL,SPI0_I2SMCLK,SD0_CMD,UART2_RXD,I2C1_SDA,EPWM0_CH5,EPWM1_CH5,EPWM0_BRAKE1,OPA0_P,EBI0_ADR9,EADC0_CH0,EVENTOUT,
PortB,PB1,MFPL,EADC0_CH1,OPA0_N,EBI0_ADR8,SD0_CLK,EMAC0_RMII_RXERR,SPI1_I2SMCLK,SPI3_I2SMCLK,UART2_TXD,USCI1_CLK,I2C1_SCL,I2S0_LRCK,EPWM0_CH4,EPWM1_CH4,EPWM0_BRAKE0,EVENTOUT,
```

NUMICROPY USER MANUAL

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 LAN Class

The NuMicroPy implements a LAN class to network module according to the network class definitions of MicroPython. The LAN class is based on lwIP TCP/IP stack and associated with EMAC hardware.

Once the network is established, the usocket module can be used to create and use TCP/UDP sockets as usual.

```python
import network

lan = network.LAN()     # create lan interface
lan.isconnected()       # check if the lan is connected and up
lan.active(True)        # Activate the lan interface
lan.ifconfig(('192.168.0.4', '255.255.255.0', '192.168.0.1', '8.8.8.8')) #set IP address, subnet mask, gateway and DNS
lan.ifconfig()          # get IP address, subnet mask, gateway and DNS
lan.ifconfig('dhcp')    # set dynamic IP
```

5.2 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_PFM_M487 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_PFM_M487	D0	B2
	D1	B3
	D2	C9
	D3	C10
	D4	C11
	D5	C12
	D6	E4
	D7	E5
	D8	A5
	D9	A4

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

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

	D10	A3
	D11	A0
	D12	A1
	D13	A2
	A0	B6
	A1	B7
	A2	B8
	A3	B9
	A4	B0
	A5	B1
	SW2	G15
	SW3	F11
	LEDR	H0
	LEDY	H1
	LEDG	H2

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

```python
from pyb import Pin

p_d0 = Pin(Pin.board.D0, Pin.OUT)    # create output pin on GPIO B2
p_d0.value(1)                        # set pin to on/high

p_d1 = Pin(Pin.board.D1, Pin.IN)     # create input pin on GPIO B3
print(p_d1.value())                  # get value, 0 or 1

Pin.board.D2.af_list()               # list available alternate functions on GPIO C9

def sw2_callback(pin):               # define sw2 (switch button 2) callback
    print(pin)

sw2 = Pin.board.SW2
sw2.irq(handler=sw2_callback, trigger=Pin.IRQ_RISING)  # configure sw2 to interrupt
```

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.

5.3 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_PFM_M487	CH0	A4	B0
	CH1	A5	B1
	CH2	D0	B2
	CH3	D1	B3
	CH6	A0	B6
	CH7	A1	B7
	CH8	A2	B8
	CH9	A3	B9

Table 5-2 ADC Support List

```
from pyb import ADC, Pin

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

adc_all = ADCALL(12, 0x00003)       # 12 bit resolution, internal channels 0 and 1
val = adc.read_channel(1)           # read analog value of channel 1
val = adc.read_core_temp()          # read MCU's temperature
val = adc.read_core_vbat()          # read MCU's VBAT
```

5.4 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_PFM_M487	SPI0_NSS	D10	A3
	SPI0_SCK	D13	A2

	SPI0_MISO	D12	A1
	SPI0_MOSI	D11	A0
	SPI3_NSS	D2	C9
	SPI3_SCK	D3	C10
	SPI3_MISO	A3	B9
	SPI3_MOSI	A2	B8

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

5.5 I^{2}C Class

Table 5-4 is hardware I^{2}C support list by the board.

Board	I2C Pin Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	I2C0_SCL	D8	A5
	I2C0_SDA	D9	A4
	I2C1_SCL	D10	A3
	I2C1_SDA	D13	A2

Table 5-4 I^{2}C Support List

```
from pyb import I2C

i2c = I2C(1, I2C.MASTER)     # create and initiate I2C1 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)               # receive 3 bytes, writing them into data
i2c.deinit()                 # turn off the peripheral
```

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.

5.6 RTC Class

```
from pyb import RTC

rtc = RTC()
rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # set a specific date and time
rtc.datetime() # get date and time
```

5.7 UART Class

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

Board	UART Pin Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	UART1_RXD	D0	B2
	UART1_TXD	D1	B3
	UART1_CTS	D12	A1
	UART1_RTS	D11	A0
	UART5_RXD	D9	A4
	UART5_TXD	D8	A5
	UART5_CTS	D0	B2
	UART5_RTS	D1	B3

Table 5-5 UART Support List

```
from pyb import UART

uart = UART(1, 9600, bits=8, parity=None, stop=1) # initiate UART1
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).

5.8 CAN Class

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

Board	CAN Pin Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	CAN0_RXD	D9	A4
	CAN0_TXD	D8	A5
	CAN1_RXD	D2	C9
	CAN1_TXD	D3	C10

Table 5-6 CAN Support List

```
from pyb import CAN

can = CAN(1, mode=CAN.NORMAL, extframe=True, baudrate=500000) # create and initiate an object on CAN1
can.setfilter(id=0x55, fifo=10, mask=0xf0)  # set a filter to receive messages with id=0x55 and mask is 0xf0 on FIFO 10
can.send('message!', id=0x50)   # send a message with id 0x50
can.recv(fifo=10)               # receive message on FIFO 10

buf = bytearray(8)
data_lst = [0, 0, 0, memoryview(buf)]

def can_cb1(bus, reason, fifo_num):
    if reason == CAN.CB_REASON_RX:
        bus.recv(fifo = fifo_num, list = data_lst)
        print(data_lst)
    if reason == CAN.CB_REASON_ERROR_WARNING:
        print('Error Warning')
    if reason == CAN.CB_REASON_ERROR_PASSIVE:
        print('Error Passive')
    if reason == CB_REASON_ERROR_BUS_OFF:
        print('Bus off')

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, fifo=0, mask=0)

- id is the identifier of the frame that will be received.
- fifo is the FIFO number, from 0 to 31.
- mask is the identifier mask used for a acceptance filtering.

can.recv(fifo=0, list=None, timeout=5000)

- fifo is an integer, which is the FIFO to receive on.
- 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

5.9 USB HID Class

The USB_HID class allows creation of an object representing the USB 1.1 Human Interface Device.

```
import pyb

pyb.usb_mode('HID')   # set USB device mode to HID
hid = pyb.USB_HID()   # create a new USB HID object

#prepare multiple of 64 bytes buffer for sending or receiving, beacuse each HID
 transaction is 64 bytes
send_pakcet_size = hid.send_packet_size()
send_buf = bytearray(send_pakcet_size)
recv_pakcet_size = hid.recv_packet_size()
recv_buf = bytearray(recv_pakcet_size)

send_buf[0] = 1
hid.send(send_buf)    # send data on the bus
hid.recv(recv_buf)    # receive data on the bus
```

5.10 USB VCP Class

The USB_VCP class allows creation of an object representing the USB 1.1 virtual com port. It can be used to read and write data over USB to the connected host.

```
import pyb

pyb.usb_mode('VCP')        # set USB device mode to VCP
vcp = pyb.USB_VCP()        # create a new USB VCP object

# prepare sending or receiving buffer
send_pakcet_size = 64
send_buf = bytearray(send_pakcet_size)
recv_pakcet_size = 64
recv_buf = bytearray(recv_pakcet_size)

vcp.isconnected()         # return True if USB is connected as a serial device
vcp.any()                 # return True if any characters waiting
vcp.read(64)              # read at most 64 from the serial device
vcp.readline()            # read a whole line from the serial device
vcp.readinto(recv_buf)    # read bytes from the serial device and store them into
buffer
vcp.recv(recv_buf, timeout = 100)     # receive data with timeout
vcp.write(send_buf)       # write the bytes from buffer to the serial device
vcp.send(send_buf)        # send data over the USB VCP
```

5.11 LED Class

The LED object controls an individual LED. Table 5-7 is LED support list by the board.

Board	LED Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	led0	LEDR	H0
	led1	LEDY	H1
	led2	LEDG	H2

Table 5-7 LED Support List

```python
from pyb import LED

led = LED('led0')      # create an LED object
led.off()              # turn the LED off
led.on()               # turn the LED on
led.toggle()           # toggle the LED between on an off
```

5.12 Switch Class

A Switch object is used to control a push-button switch. Table 5-8 is switch support list by the board.

Board	Switch Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	sw2	SW2	G15
	sw3	SW3	F11

Table 5-8 Switch Support List

```python
from pyb import Switch

sw = Switch('sw2')         # create a switch object
sw.value()                 # get state (True if pressed, False otherwise)
sw()                       # shorthand notation to get the switch state

def sw2_callback():
    print('sw2 press')

sw.callback(sw2_callback) # register a callback to be called when the switch is
pressed down
sw.callback(None)          # remove the callback
```

5.13 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.

```python
from pyb import Pin
from pyb import Timer

def tick(timer):
    print(timer.counter())

tim = Timer(3, freq = 2)  # create a timer object using timer 3 and trigger at 2Hz
```

```
tim.callback(tick)  # set the function to be called when the timer triggers

# configure timer to be a PWM, output compare, or input capture channel
chan = tim.channel(Timer.PWM, pin = Pin.board.D0, pulse_width_percent = 20)
chan.callback(tick)  # set the function to be called when the timer channel triggers

chan.capture()  # get the capture associated with a input capture channel

chan.compare(100)  # set compare value assocated with a channel
chan.compare()  # get the compare value associated with a channel

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

5.14 PWM Class

The NuMicroPy supports BPWM and EPWM for PWM class. Table 5-9 is PWM pin support list by the board.

Board	PWM Pin Name	Board Pin Name	CPU Pin Name
NuMaker_PFM_M487	BPWM0_CH0	D11	A0
	BPWM0_CH1	D12	A1
	BPWM0_CH2	D13	A2
		D6	E4
	BPWM0_CH3	D10	A3
		D7	E5

	BPWM0_CH4	D9	A4
	BPWM0_CH5	D8	A5
	BPWM1_CH2	A3	B9
	BPWM1_CH3	A2	B8
	BPWM1_CH4	A1	B7
	BPWM1_CH5	A0	B6
	EPWM0_CH0	D8	A5
	EPWM0_CH1	D9	A4
	EPWM0_CH2	D10	A3
		D1	B3
		D7	E5
	EPWM0_CH3	D13	A2
		D0	B2
		D6	E4
	EPWM0_CH4	D12	A1
		A5	B1
	EPWM0_CH5	D11	A0
		A4	B0
	EPWM1_CH0	D5	C12
	EPWM1_CH1	D4	C11
	EPWM1_CH2	D3	C10
	EPWM1_CH3	D2	C9
	EPWM1_CH4	A5	B1
		A1	B7
	EPWM1_CH5	A4	B0
		A0	B6

Table 5-9 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 D9, CPU pin A4
epwm0ch1 = epwm0.channel(mode = PWM.CAPTURE, capture_edge = PWM.RISING, pin = Pin.board.D9, 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()  # 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.

5.15 WDT Class

```python
from pyb import WDT

wdt = WDT(timeout = 1000)   # start watchdog timer
wdt.feed()        # reset watchdog timer counter
```

WDT(timeout=5000)

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

5.16 Accel Class

```python
from pyb import Accel

a = Accel(range = Accel.RANGE_4G)   # create and return an accelerometer object
ax = a.x()    # get x axis value
ay = a.y()    # get y axis value
az = a.z()    # get z axis value
a_reg = a.read(0x00)   # read register value
a.write(0x0F, 0x08)    # write register value
```

Accel([range=Accel.RANGE_8G])

- range can be one of RANGE_2G, RANGE_4G, RANGE_8G and RANGE_16G

5.17 Gyro Class

```python
from pyb import Gyro

g = Gyro(range = Gyro.RANGE_2000DPS)   # create and return a gyroscope object
gx = g.x()    # get x axis value
gy = g.y()    # get y axis value
gz = g.z()    # get z axis value
g_reg = g.read(0x00)   # read register value
g.write(0x0F, 0x08)    # write register value
```

Gyro([range=Gyro.RANGE_2000DPS])

- range can be one of RANGE_125DPS, RANGE_250DPS, RANGE_500DPS, RANGE_1000DPS and RANGE_2000DPS

5.18 Mag Class

```python
from pyb import Mag

m = Mag()     # create and return a magnetometer object
mx = m.x()    # get x axis value
my = m.y()    # get y axis value
mz = m.z()    # get z axis value
m_reg = m.read(0x00)   # read register value
m.write(0x0F, 0x08)    # write register value
```

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 REVISION HISTORY

Date	Revision	Description
2019.03.29	1.00	1. Initially issued.

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.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.