BG95M3-QPython EVB (Start-Up and Data Call)

Huanchen Chen (Erik), tekmodul GmbH

Differences from MicroPython

QuecPython is essentially MicroPython running on Quectel modules. Due to the lack of a comprehensive standard specification in MicroPython, the built-in libraries and the usage of various functions may vary depending on the developers and hardware platforms. Some QuecPython users may have prior experience developing with MicroPython on modules such as ESP32, ESP8266, and STM32. To facilitate the migration for these users, the known differences between QuecPython and MicroPython are listed below:

- Some MicroPython standard or dedicated libraries, such as framebuf and network, are not implemented or built-in in QuecPython.
- Some MicroPython standard libraries, such as utime, may have different implementations and feature completeness in QuecPython compared to modules like ESP32, resulting in differences in performance or other detailed characteristics.
- The organization of certain features in QuecPython may differ from MicroPython. For example, ADC functionality is generally included in the machine library in MicroPython, but in QuecPython, it is included in the misc library.
- APIs related to specific hardware interfaces such as UART, I2C, and SPI have significant differences between MicroPython and QuecPython and cannot be used interchangeably.
- QuecPython currently does not include the upip functionality, so quick online installation of libraries is not possible. Manual porting is required.
- Compatibility with MicroPython IDE tools such as Thonny and uPyCraft is not guaranteed.

In summary, MicroPython code that runs successfully on modules like ESP32 usually **cannot be directly copied and run in the QuecPython** environment without any modifications. Therefore, it is advisable to **avoid directly applying documentation and development experience from other MicroPython** hardware modules to QuecPython development.

Differences from CPython (normale Python)

- Unlike traditional CPython development, QuecPython has a much **smaller number of built-in libraries** (standard libraries). Although QuecPython does include basic libraries for tasks such as time setting and file management, the quantity is significantly lower compared to CPython. The names and usage methods of these standard libraries also have many differences and are not fully compatible.
- QuecPython **does not have built-in pip functionality**, so quick **online installation of libraries is not possible**. Manual porting is required.
- Due to the syntax differences between QuecPython and CPython, as well as the fact that most QuecPython libraries cannot run on desktop computers, tools such as VSCode and PyCharm on the desktop can only be used for simple code editing. The completed **scripts need to be manually downloaded to the module** for execution and debugging.
- **The syntax highlighting and code completion features provided by tools like VSCode and PyCharm are based on CPython and may not be fully applicable to QuecPython**. Therefore, for beginners who have no prior experience with the Python language, **it is not recommended to use overly intelligent IDE tools while writing code**, as the built-in suggestions may be misleading.

EVB Connection

- **Step 1: Connect EVB**

Connect the EVB Type-C port to your PC USB port with a USB Type-C cable for power supply.

- **Step 2: Power on EVB**

Short the two PWK_AUTO pins with a jumper cap to power on the board and it will turn on automatically, or long press the PWK button after power-on. It is recommended that the time interval between power-on and pressing the PWK button should be at least 30 ms.

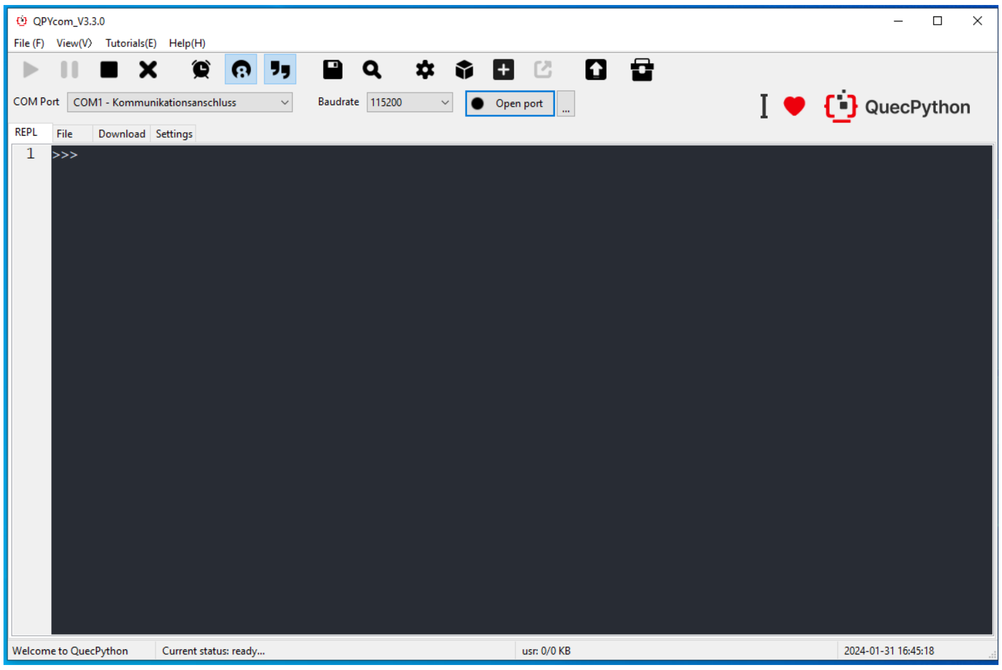
After performing the above operations, wait for the power indicator on the main board to light up (LED PWR lights up blue, PWM lights up green, then EVB should be recognized in Device Manager.).

SIM Choice

For BG95 EVB, an NB-IoT SIM card should be used. For test we used 1NCE SIM (It should recognize the Telekom network, send SMS, and support National Roaming).

Tools

QPYcom. This tool is used for debugging code, analyzing logs, uploading python scripts to module, downloading firmware, merging firmware, and more. Can not modify Python scripts!



VSCode: To write Python scripts. The python scripts written by VSCode must later be imported into QPYcom.

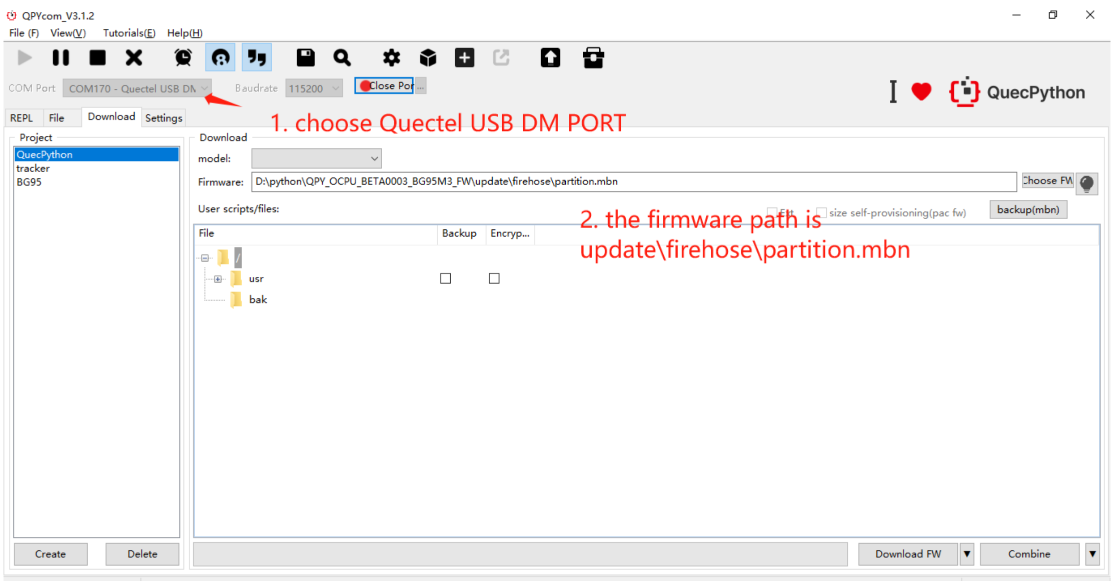
Firmware Flashing

Standard AT firmware or QuecOpen firmware is usually downloaded into a module when the module leaves the factory. If you want to develop the module based on QuecPython, you need to manually re-download the dedicated QuecPython firmware into it.

https://python.quectel.com/en/download

Note: QuecPython requires a **special firmware**! It needs to be flashed separately. After flashing this firmware, the board CANNOT be used for the AT command environment. If development with AT commands is desired, the corresponding firmware must be flashed again.

Example: BG95_M3 QPython Firmware Flash:


QPYcom_V3.1.2

COM Port: COM170 - Quectel USB DM
Baudrate: 115200

1. choose Quectel USB DM PORT

Project: QuecPython, tracker, BG95

Download
model:
Firmware: D:\python\QPY_OCPU_BETA0003_BG95M3_FW\update\firehose\partition.mbn

2. the firmware path is update\firehose\partition.mbn

User scripts/files:

File	Backup	Encryp...
/		
usr	☐	☐
bak		

Port Connection BG95M3

REPL Port: REPL stands for Read-Eval-Print-Loop (interactive interpreter). You can debug QuecPython programs in REPL. Kann python Befehle eingeben (z.B. 5+3)

REPL, short for Read-Eval-Print Loop, is a simple interactive programming environment. REPL typically provides a Command-Line Interface (CLI) that receives user input, parses and executes it, and then returns the results to the user. In terms of functionality and usage, it is similar to the Command Prompt (CMD) in Windows or the Shell in macOS/Linux.

DM Port: Digital Media Port. For Firmware Flashing.

Start-Up Qpython Functions and Commands

Network Registration (in QPYcom, REPL port)

Import net

net.operatorName() gets the operator information of the current network registration.

net.getModemFun() This method is used to obtain the current functional mode of the module.

net.getState() gets the network registration information. （AT+CREG=?

Establish DataCall

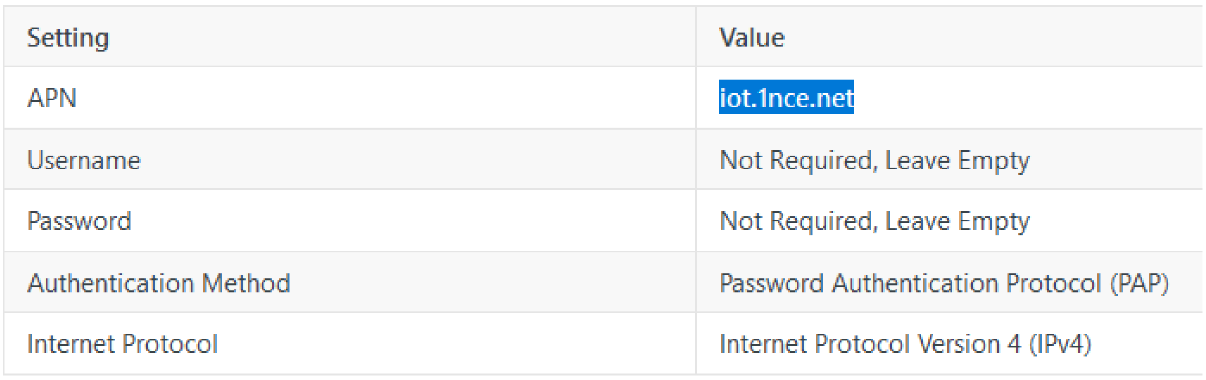
Import dataCall

dataCall.getInfo(profileID, ipType)

# profileID – PDP ID， range 1~3。
# ipType – IP typ， 0： IPV4 1： IPV6 2： IPV4&IPV6
If the output is (1, 0, [0, 0, '0.0.0.0', '0.0.0.0', '0.0.0.0']), it means that the network is already registered, but the data call is not set up.

dataCall.setPDPContext(1,0,'iot.1nce.net','','',1) -> return: 0 = successful

The PDP context with the appropriate APN must be set according to the SIM card manufacturer. (In our case, the SIM card manufacturer is 1NCE, so you need to look for the APN settings for 1NCE).



Setting	Value
APN	iot.1nce.net
Username	Not Required, Leave Empty
Password	Not Required, Leave Empty
Authentication Method	Password Authentication Protocol (PAP)
Internet Protocol	Internet Protocol Version 4 (IPv4)

dataCall.getPDPContext(1) -> return: 0 = successful

dataCall.activate(1) -> return: 0 = successful

dataCall.getInfo(1,0) -> return: (1, 0, [1, 0, '100.69.60.50', '8.8.8.8', '8.8.4.4'])

import checkNet

checkNet.waitNetworkReady(60) -> return: (3,1) network ready

References

https://python.quectel.com/doc/Getting_started/en/index.html

https://python.quectel.com/doc/Application_guide/en/background/iot-and-low-code.html

https://python.quectel.com/doc/API_reference/en/iotlib/dataCall.html