QSTM32
SDK Quick Start Guide

Version: 2.1

Date: 2026-01-21

Status: Released

At Quectel, our aim is to provide timely and comprehensive services to our customers. If you require any assistance, please contact our headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai 200233, China
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local offices. For more information, please visit:
http://www.quectel.com/support/sales.htm.

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm.
Or email us at: support@quectel.com.

Legal Notices

We offer information as a service to you. The provided information is based on your requirements and we make every effort to ensure its quality. You agree that you are responsible for using independent analysis and evaluation in designing intended products, and we provide reference designs for illustrative purposes only. Before using any hardware, software or service guided by this document, please read this notice carefully. Even though we employ commercially reasonable efforts to provide the best possible experience, you hereby acknowledge and agree that this document and related services hereunder are provided to you on an "as available" basis. We may revise or restate this document from time to time at our sole discretion without any prior notice to you.

Use and Disclosure Restrictions

License Agreements

Documents and information provided by us shall be kept confidential, unless specific permission is granted. They shall not be accessed or used for any purpose except as expressly provided herein.

Copyright

Trademarks

Except as otherwise set forth herein, nothing in this document shall be construed as conferring any rights to use any trademark, trade name or name, abbreviation, or counterfeit product thereof owned by Quectel or any third party in advertising, publicity, or other aspects.

Third-Party Rights

This document may refer to hardware, software and/or documentation owned by one or more third parties ("third-party materials"). Use of such third-party materials shall be governed by all restrictions and obligations applicable thereto.

We make no warranty or representation, either express or implied, regarding the third-party materials, including but not limited to any implied or statutory, warranties of merchantability or fitness for a particular purpose, quiet enjoyment, system integration, information accuracy, and non-infringement of any third-party intellectual property rights with regard to the licensed technology or use thereof. Nothing herein constitutes a representation or warranty by us to either develop, enhance, modify, distribute, market, sell, offer for sale, or otherwise maintain production of any our products or any other hardware, software, device, tool, information, or product. We moreover disclaim any and all warranties arising from the course of dealing or usage of trade.

Privacy Policy

To implement module functionality, certain device data are uploaded to Quectel's or third-party's servers, including carriers, chipset suppliers or customer-designated servers. Quectel, strictly abiding by the relevant laws and regulations, shall retain, use, disclose or otherwise process relevant data for the purpose of performing the service only or as permitted by applicable laws. Before data interaction with third parties, please be informed of their privacy and data security policy.

Disclaimer

a) We acknowledge no liability for any injury or damage arising from the reliance upon the information.
b) We shall bear no liability resulting from any inaccuracies or omissions, or from the use of the information contained herein.
c) While we have made every effort to ensure that the functions and features under development are free from errors, it is possible that they could contain errors, inaccuracies, and omissions. Unless otherwise provided by valid agreement, we make no warranties of any kind, either implied or express, and exclude all liability for any loss or damage suffered in connection with the use of features and functions under development, to the maximum extent permitted by law, regardless of whether such loss or damage may have been foreseeable.
d) We are not responsible for the accessibility, safety, accuracy, availability, legality, or completeness of information, advertising, commercial offers, products, services, and materials on third-party websites and third-party resources.

About Document

Revision History

Version	Date	Author	Description
-	2025-05-01	Jerry Chen	Creation of the document
1.0	2025-05-15	Jerry Chen	Initial version
2.0	2025-08-08	Jerry Chen	Update Figure 17 in Chapter 5.1
2.1	2025-01-21	Jerry Chen	Rename the project to UniKnect

Contents

Table Index

Figure Index

1 Introduction

Quectel UniKnect Project, one SW framework designed for developer, is capable to implement various functions by calling API. Thus, the developer can focus on working logic alone without paying attention to handle data interaction between MCU and module, making it easier and more friendly to develop.

In this article, it will illustrate how to utilize Quectel UniKnect Project SDK, including SDK directory structure, compilation framework, compilation environment building, HW components preparation, project building & compilation, cleaning, downloading and debugging in development stage.

2 SDK Directory Architecture

See Quectel UniKnect Project SDK directory architecture as shown in ***Figure 1***.

```
├── .vscode            # VSCode debugging environment configuration
│                        (optional, generated by script)
├── apps               # Applications directory, including example and app
│                        entrance of individual function
├── build              # Build output directory, including intermediate file
│                        and executable file generated by compilation
├── quectel            # Quectel-related code adaption directory
├── system             # System directory, including HAL-related codes
│                        adapted to various MCU driver
├── tools              # Tool script directory, including cross-compilation
│                        tool-chain, script and configuration
├── .clang-format      # Clang format file
├── .editorconfig      # Cross-editor format uniform configuration file
├── .gitignore         # Git version control ignore regulation
├── build.bat          # SDK script to build, compile, download and debug
├── CMakeLists.txt     # Build Main configuration in Cmake (generated by script)
├── CMakePresets.json  # Build Preset parameter in Cmake (generated by script)
└── README.md          # Illustration document
```

Figure 1: SDK Directory Architecture

3 SDK Compilation Structure

Corresponding compilation structure of Quectel UniKnect Project SDK supports all types of STM32 micro-processors. Different MCU can share the same SDK, which will differentiate in initialization configuration when building project. Additionally, all operations can be done automatically without user, making it friendly.

3.1. Main Characteristic

- **Build, compile, clean, download and debug**
- Cross-compilation tool-chain is embedded in SDK and no need to build compilation environment
- All relevant parameters and files among various MCUs can adapt automatically
- *Build files such as CMakeLists.txt & CMakePresets.json can generate automatically.*
- Configrue automation script, which can accomplish above operation without configuring manually

3.2. Operation

1. **CLI:** Not rely on any IDE or code editor
2. **GUI:** VSCode + Plugin (Related parameters can be generated automatically)

3.3. Auto-configuration

- HAL driver-related code
- Flash link script .ld file
- cfg/.svd/interface/target file called by OpenOCD
- Compiler, linker, macro, compilation file and include directory
- *CMakeLists.txt / CMakePresets.json / xxx.json*
- Absent chip type and SW version
- Fool-proofing by **build.bat** in terms of exceptional parameters written by user.

3.4. Compilation Architecture

See ***Figure 2*** on Quectel UniKnect Project SDK compilation architecture.

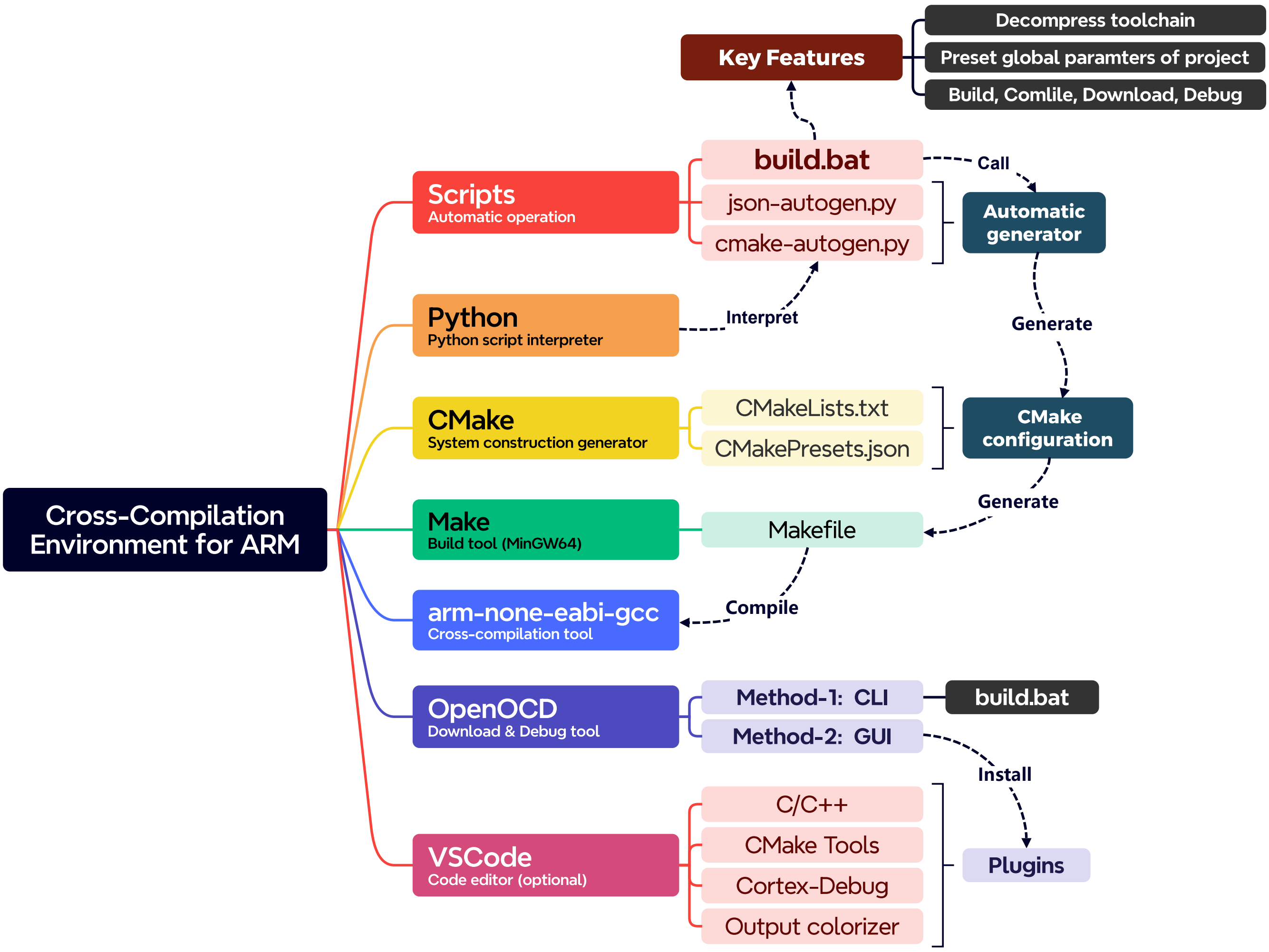

Figure 2: SDK Compilation Architecture

4 SDK Compilation Environment

4.1. Overview

The host with 64-bit Windows10 is suggested when running Quectel UniKnect Project SDK compilation environment.
Intact compilation toolchain is integrated in SDK; the user can deploy it directly or build compilation environment on their own. See difference on above two methods as ***Table 1***.

Table 1: SDK Compilation Environment Building Comparison

Comparison	**SDK-embedded compilation Toolchain (Recommeded)**	**Build on their own**
Difficulty in Building	No need to build ★☆☆☆☆	Build according to **Chapter 4.3** in this article ★★★★☆
Occupied Space	Intact toolchain: 2G. By zipping: 332M Time for unzipping: 4 minutes ★★★★☆	None ☆☆☆☆☆

4.2. SDK-embedded Compilation Tool-chain

In order to simplify development, save effort in building compilation and facilitate actual application, intact compilation tool-chain is embedded in SDK. Upon deploying SDK for the first time, the toolchain will be unzipped automatically when executing **build.bat**. See relevant path: ***tools\toolchain***.
See directory structure after unzipping toolchain as ***Figure 3***.

```
├── 📁 arm-gcc        # Cross-compilation tool
├── 📁 cmake          # Build system generator
├── 📁 mingw64        # Build tool such as Make
├── 📁 openocd        # Downloading and debugging tool
└── 📁 python         # Python script interpreter
```

Figure 3: Embedded Compilation Toolchain Directory Architecture in SDK

4.3. Build Compilation Environment by yourself

For user who does not want to utilize the embedded toolchain in SDK, you can build development environment by yourself. Therefore, following chapters will illustrate how to build it from scratch.

Note

For user who selects embedded compilation tool-chain in SDK, please ignore this chapter.

4.3.1. Install ARM-GCC

1. Download [arm-gnu-toolchain-13.3.rel1-mingw-w64-i686-arm-none-eabi.zip]
2. Unzip it to the path of ***D:\Toolchain\arm-gcc***, see ***Figure 4*** in detail.

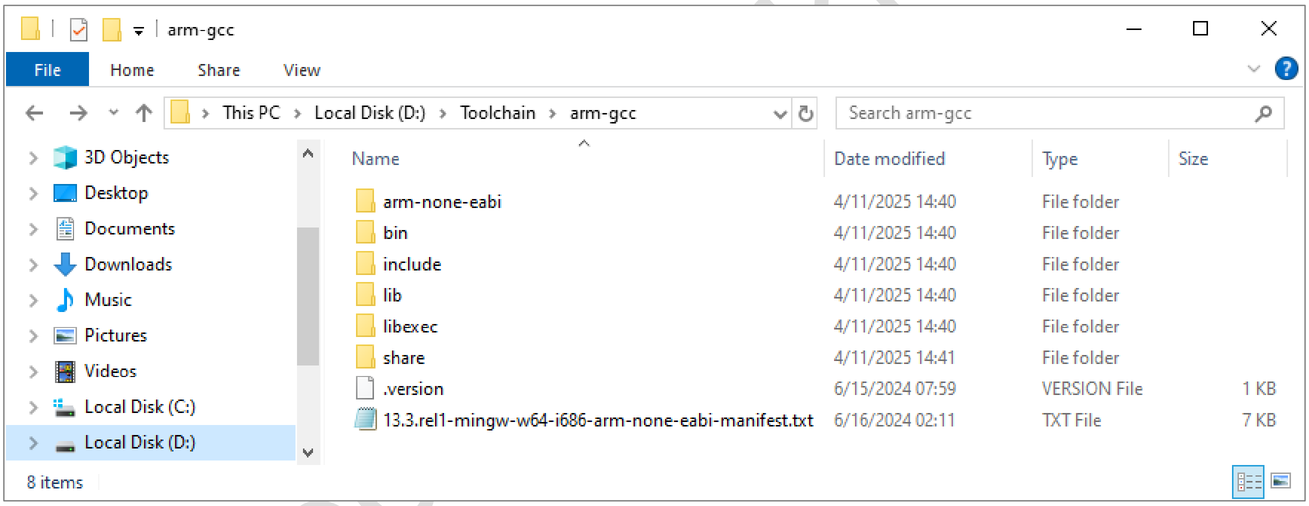

Figure 4: Install ARM-GCC

3. Add ***D:\Toolchain\arm-gcc\bin*** to the **Path** environment variable
   Please right click as following sequence: This PC➔Properties➔Advanced system settings
   Following that, operate according to steps shown in ***Figure 5***.

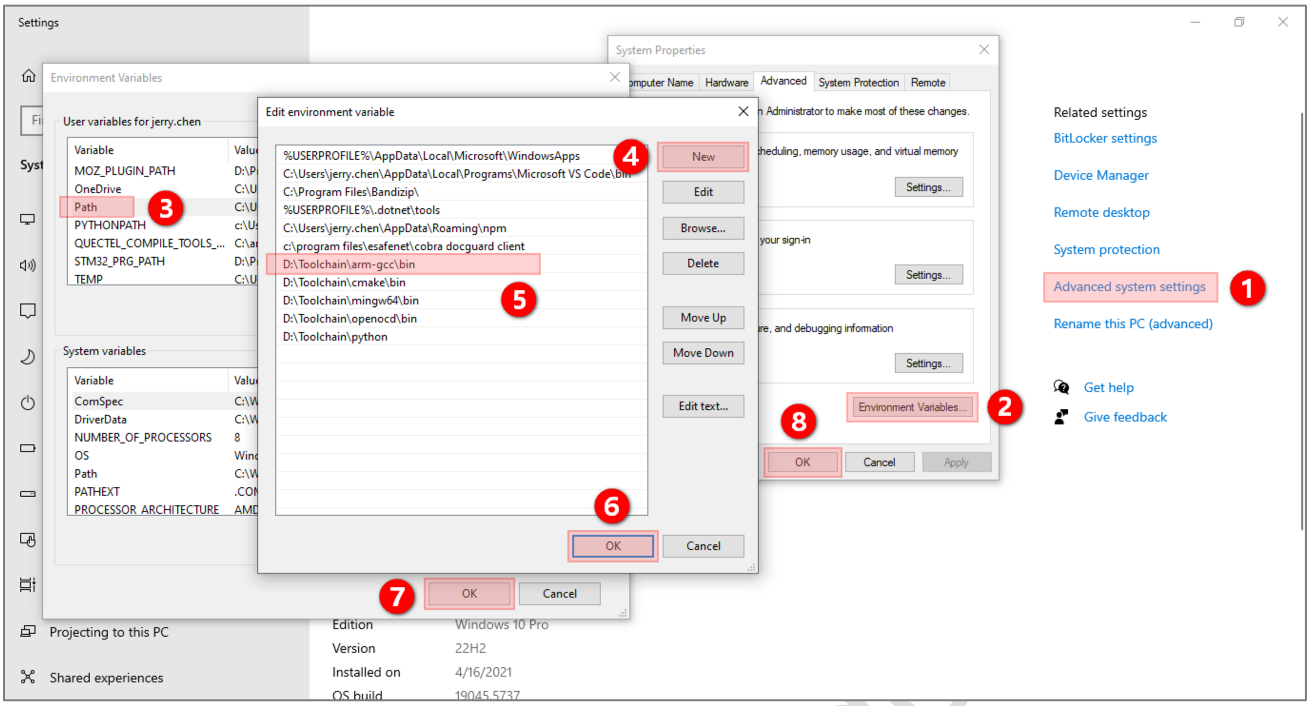

Figure 5: Set Environment Variable

4. Verify the installed ***arm-none-eabi-gcc -v***

If it is shown as ***Figure 6***, which means a success to install CMake and configure environment variable.

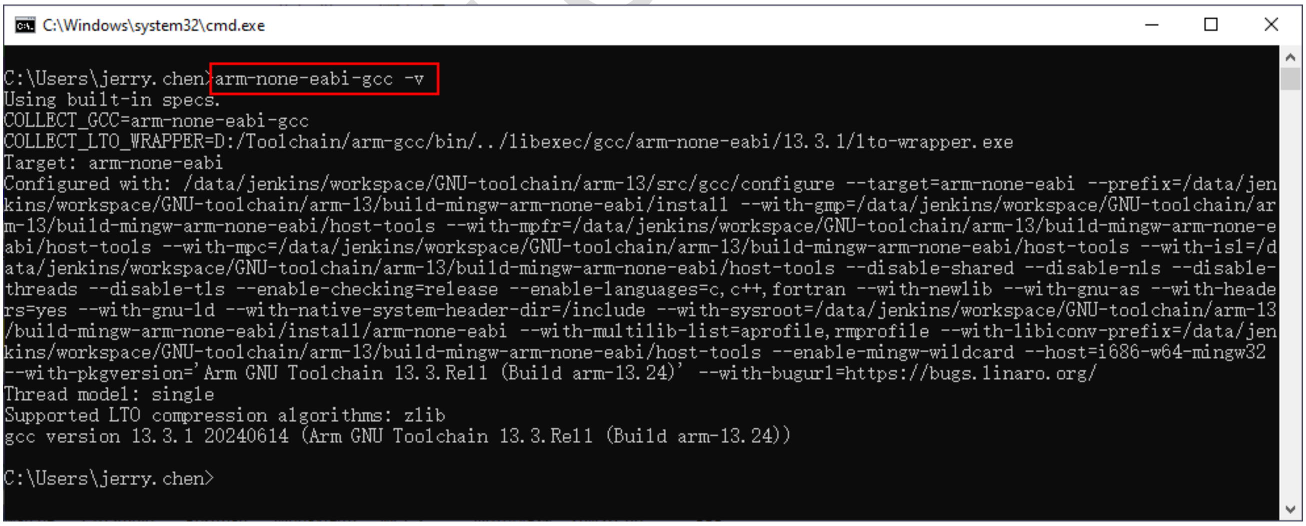

Figure 6: Verify ARM-GCC

4.3.2. Install CMake

1. Download [cmake-3.31.3-windows-x86_64.msi]
2. Install it to the path of ***D:\Toolchain\cmake***

Note: please tick "☑ **Add CMake to the PATH environment variable**", otherwise, you need to add environment variable as shown in ***Figure 7*** manually.

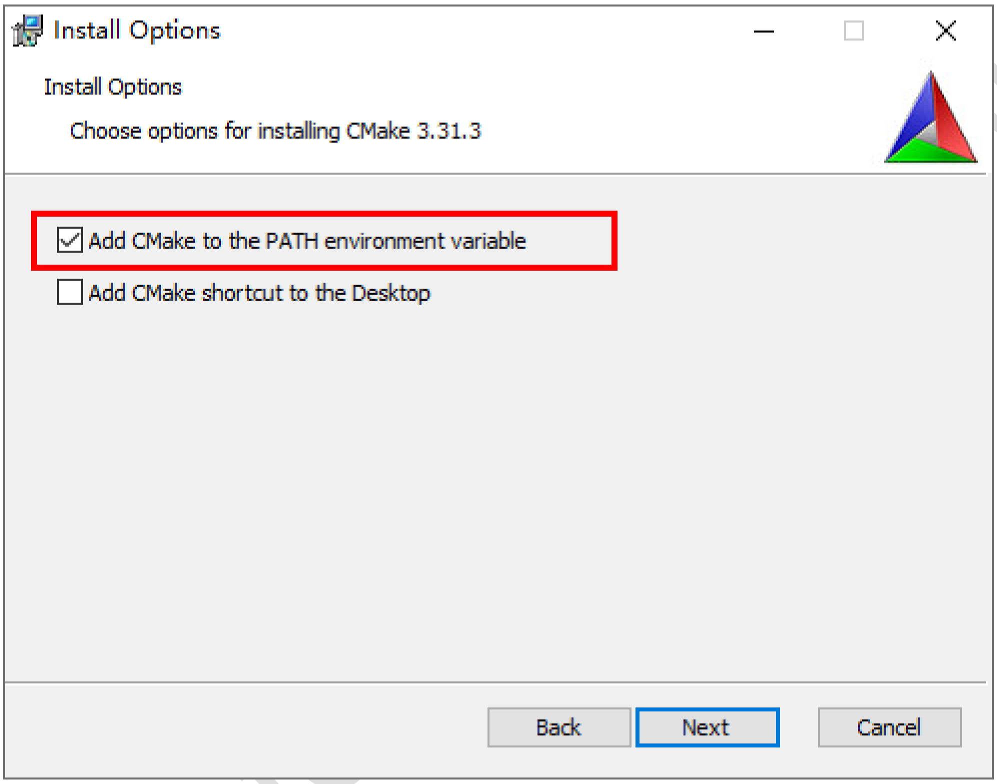

Figure 7: Install CMake

5. Verify the installed ***cmake --version***

If it is shown as ***Figure 8***, which means a success to install CMake and configure environment variable.

Figure 8: Verify CMake

4.3.3. Install MinGW

1. Download via this link -- x86_64-14.2.0-release-posix-seh-ucrt-rt_v12-rev2.7z
2. Unzip it to the path of ***D:\Toolchain\mingw64***，See specific as ***Figure 9***.

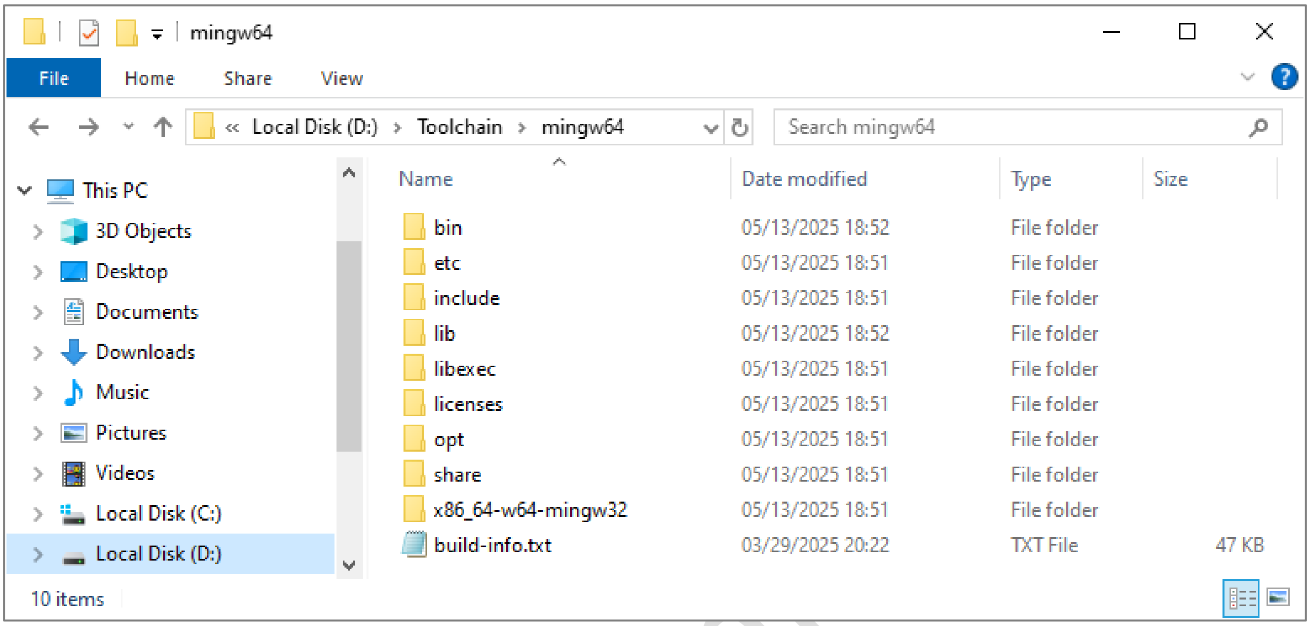

Figure 9: Install MinGW

3. Copy **mingw32-make.exe** in the directory of ***D:\Toolchain\mingw64\bin*** and rename it as **make.exe**. For specific, please refer to ***Figure 10***.

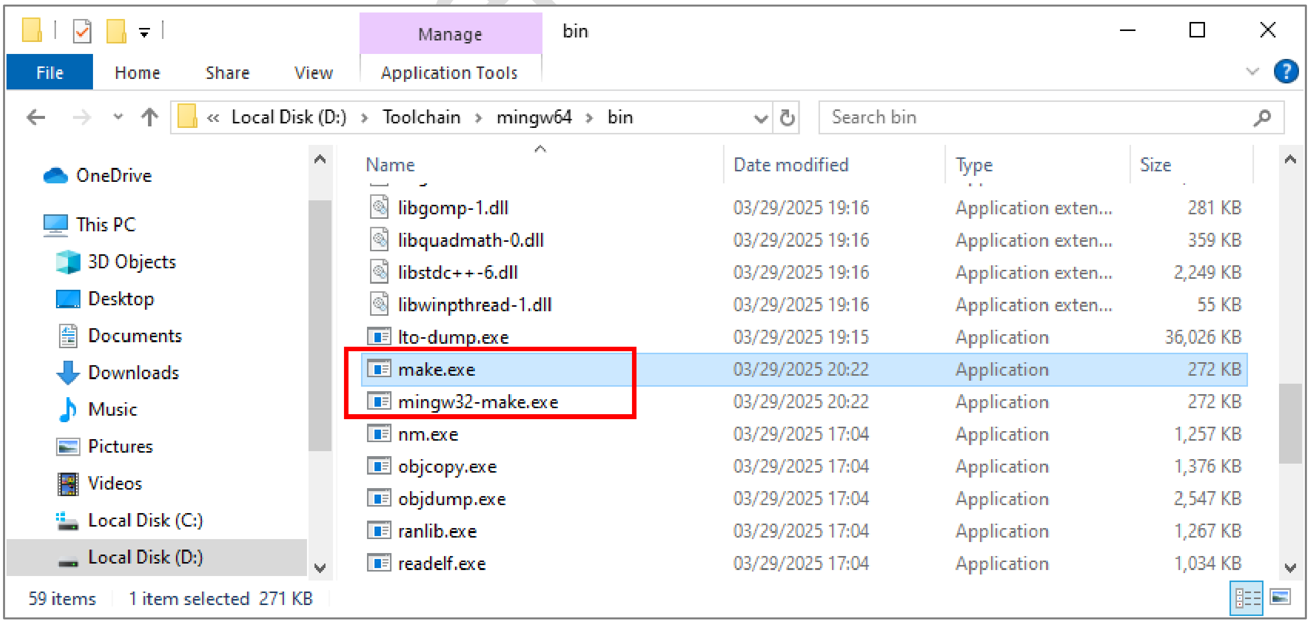

Figure 10: Copy and Rename as make.exe

4. Add ***D:\Toolchain\mingw64\bin*** to **Path** environment variable. See **Section 4.3.1**.
5. Verify the installed ***make -v***

If it is shown as ***Figure 11***, which means a success to install MinGW and configure environment variable.

Figure 11: Verify MinGW

4.3.4. Install OpenOCD

1. Download [openocd-0.12.0-20240916.7z]()
2. Unzip it to the directory of ***D:\Toolchain\openocd***, see ***Figure 12*** in detail.

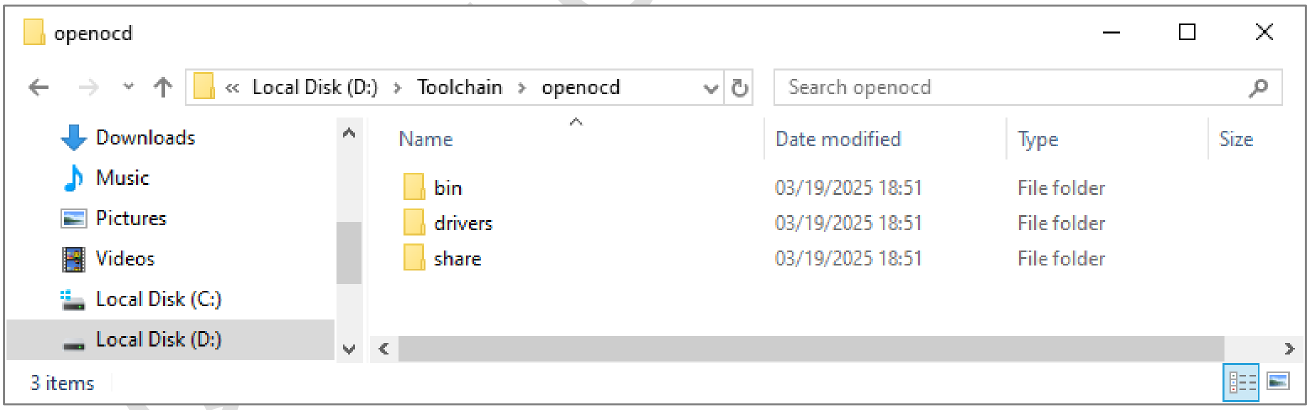

Figure 12: Install OpenOCD

3. Add ***D:\Toolchain\openocd\bin*** to **Path** environment variable. See ***Section 4.3.1***.
4. Verify the installed ***openocd -v***

If it is shown as ***Figure 13***, which means a success to install OpenOCD and configure environment variable.

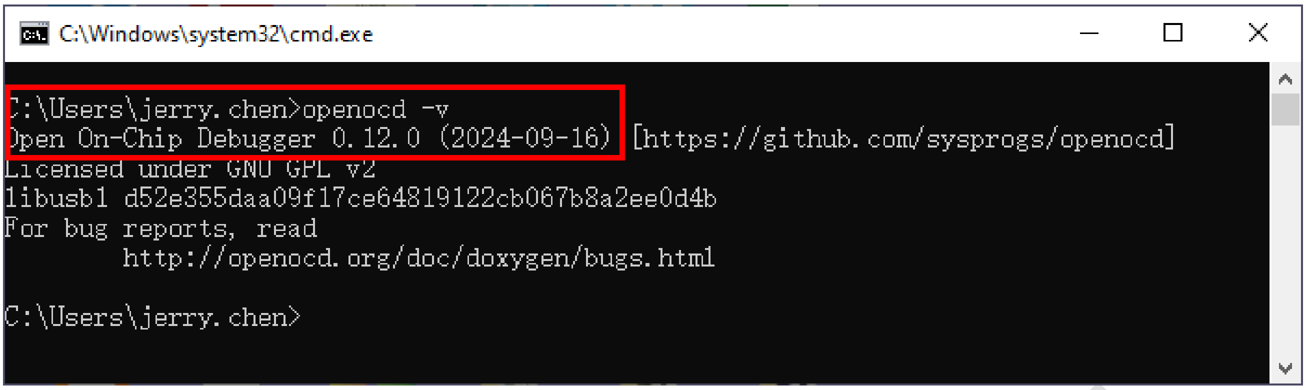

Figure 13: Verify OpenOCD

4.3.5. Install Python

1. Download this link---[python-3.9.6-amd64.exe]
2. Install Python

Note: It is needed to tick "☑ **Add Python 3.9 to PATH"** before clicking "**Install Now**", otherwise, you need to add environment variable as shown in ***Figure 14*** manually.

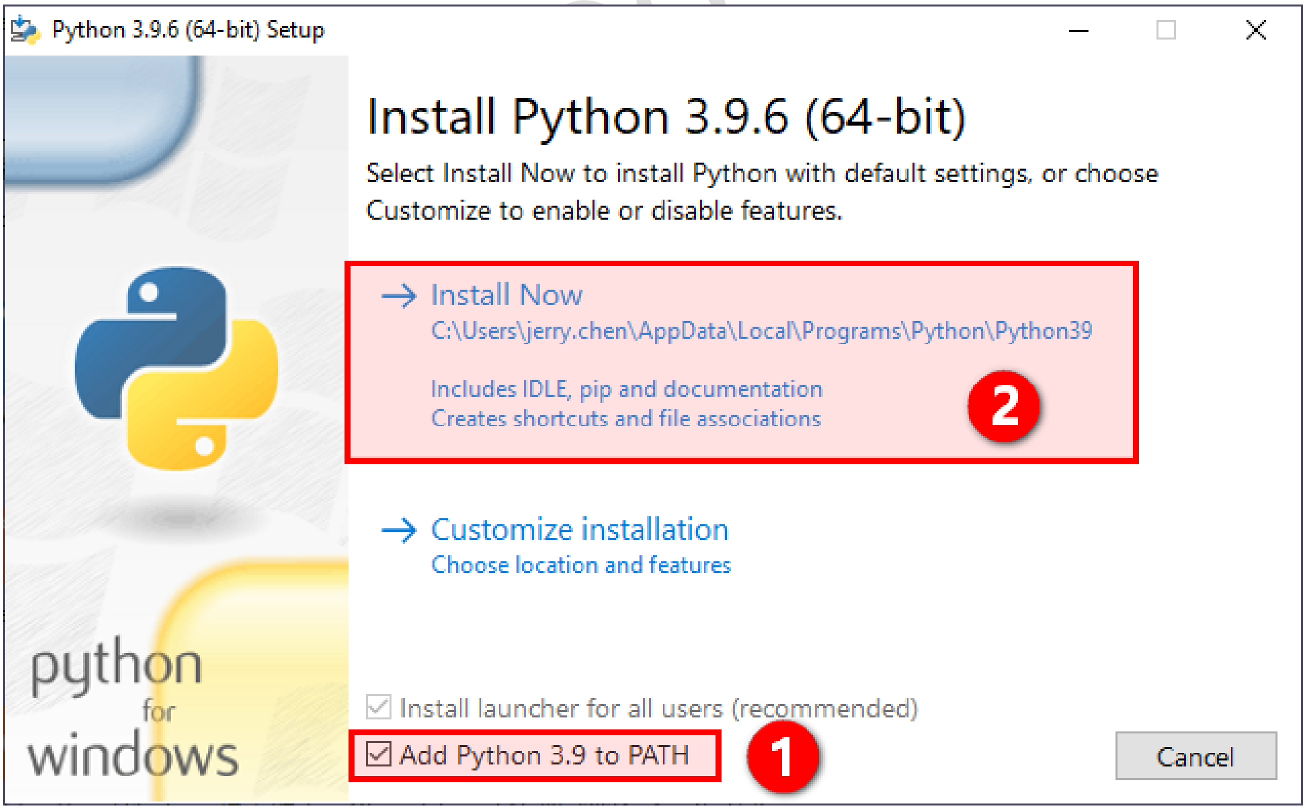

Figure 14: Install Python

3. Verify the installed Python

As displayed in ***Figure 15***, it proves a success to install Python and configure environment variable.

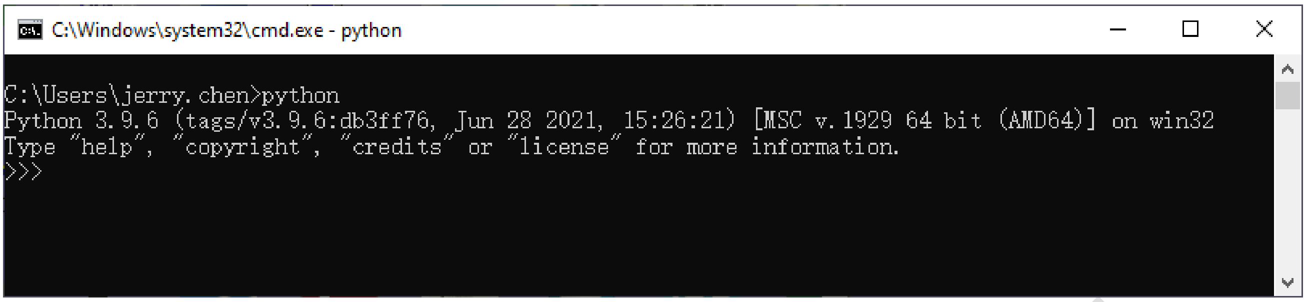

Figure 15: Verify Python

4.3.6. Delete SDK-embedded Compilation Tool-chain

Delete ***toolchian.7z*** package in ***tools*** directory. Once unzipped before, the **tools\toolchain** folder shall be deleted alongside. See ***Figure 16*** for reference.

After that, the embedded toolchain will be invisible in **Build** and **Compile** script. Instead, it will adapt the toolchain in the environment variable of **Path**.

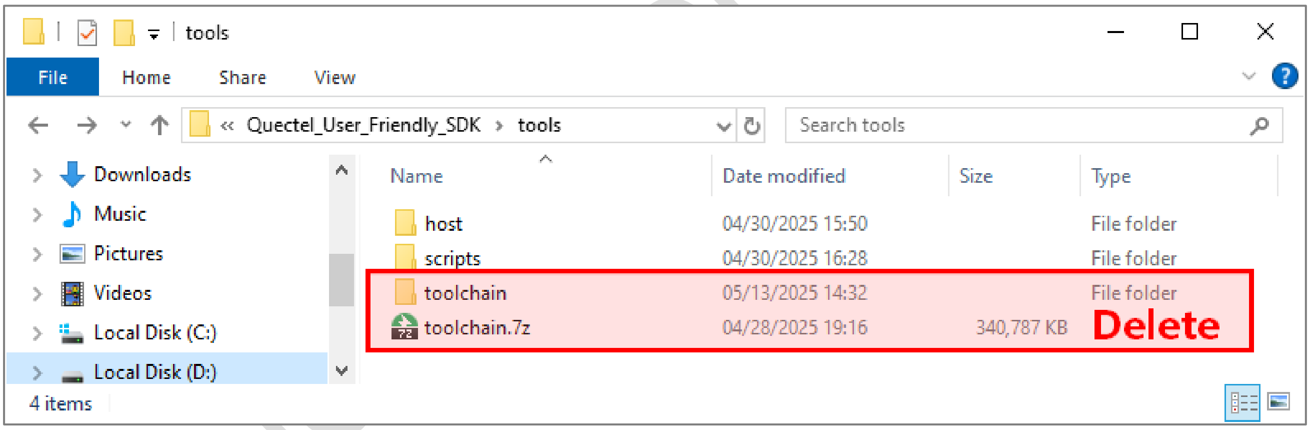

Figure 16: Delete SDK-embedded Compilation Tool-chain

5 Hardware Components

5.1. Component Assembly and Wire Connection

Before SW development, the user shall make all components ready and connected.
Please assemble mandatory components and connect wires in accordance with ***Figure 17***.

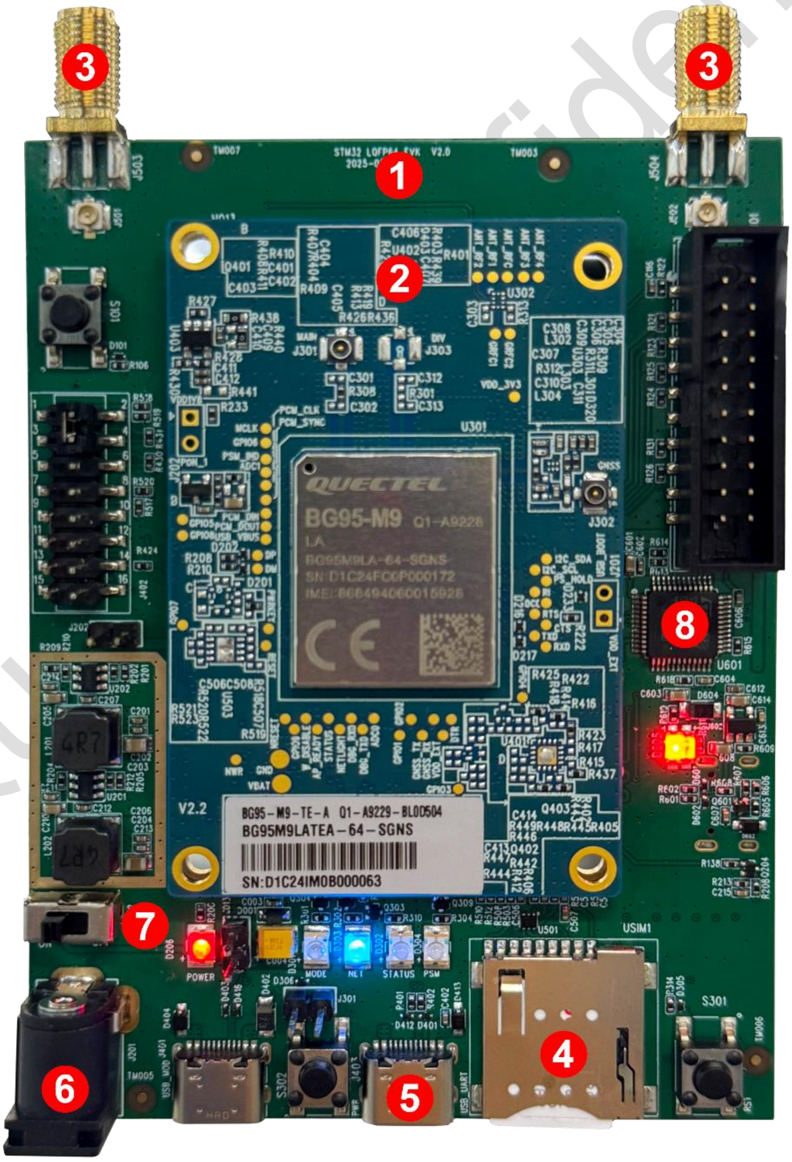

Figure 17: Components Checklist

❶ EVK
❷ TE-A
❸ Antenna
❹ Micro SIM Card
❺ Type-C USB Cable (Debug)
❻ 5V-DC Adapter
❼ Power Switch (Left: ON. Right: OFF)
❽ ST-Link Debugger

Once above connections are ready, switch the ❼ Power Button to the Left to power on.

5.2. Install Driver

5.2.1. Module Driver

If it is necessary for module to send/receive AT command or capture log via USB, please install corresponding driver according to module type embedded on TE-A. If not, the module driver can be ignored. I.e., if the module type is BG95, the corresponding driver will be shown as
Quectel_LTE&5G_Windows_USB_Driver_V2.2.4.zip
Driver for corresponding module is needed, please send your request to support@quectel.com

5.2.2. Debugger Driver

Download and install corresponding the ❽ ST-Link driver.

- ST-Link: *https://www.st.com.cn/zh/development-tools/stsw-link009.html*

After installing driver, the device manager will display STM32 STLink successfully after plugging it into PC with driver installed. See ***Figure 18*** for reference.

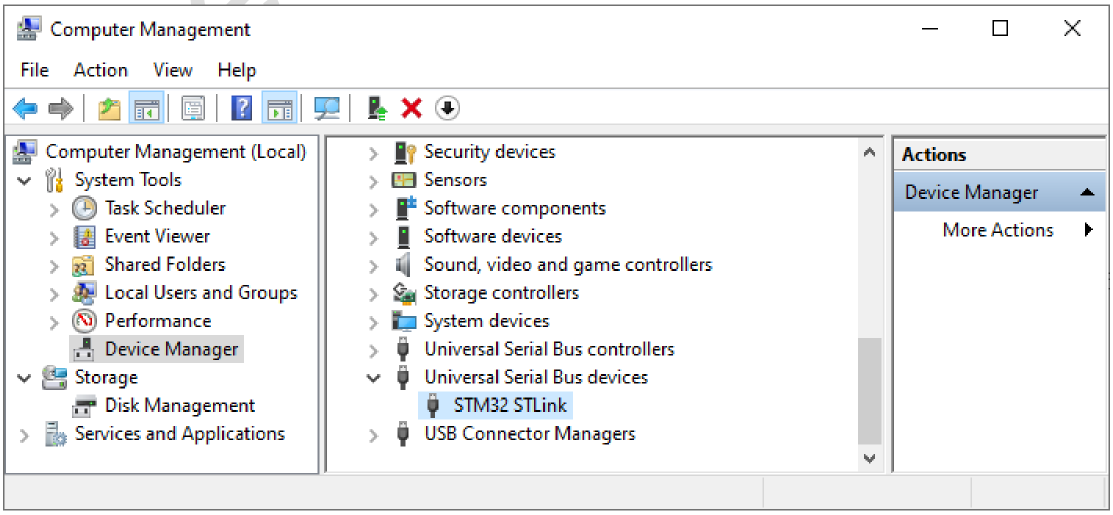

Figure 18: Debugger in Device Manager

5.2.3. Serial Port Driver

1. Download driver: [CP210x Universal Windows Driver]
2. Unzip package-> Right click "**silabser.inf**"-> Click "**Install**". See specific steps in ***Figure 19***.

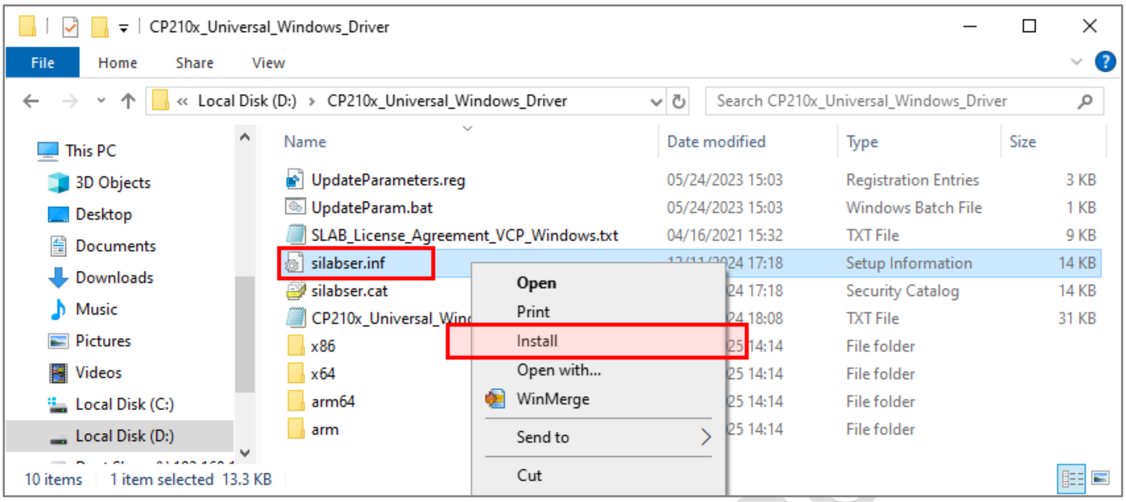

Figure 19: Install Serial Port Driver

After it is a success to install driver, it is available to display **Ports** in device manager.
Among them, the "**Silicon Labs Quad CP2108 USB to UART Bridge: Interface 1 (COM7)**" will be played as the MCU debug port to be used later. See ***Figure 20*** in detail.

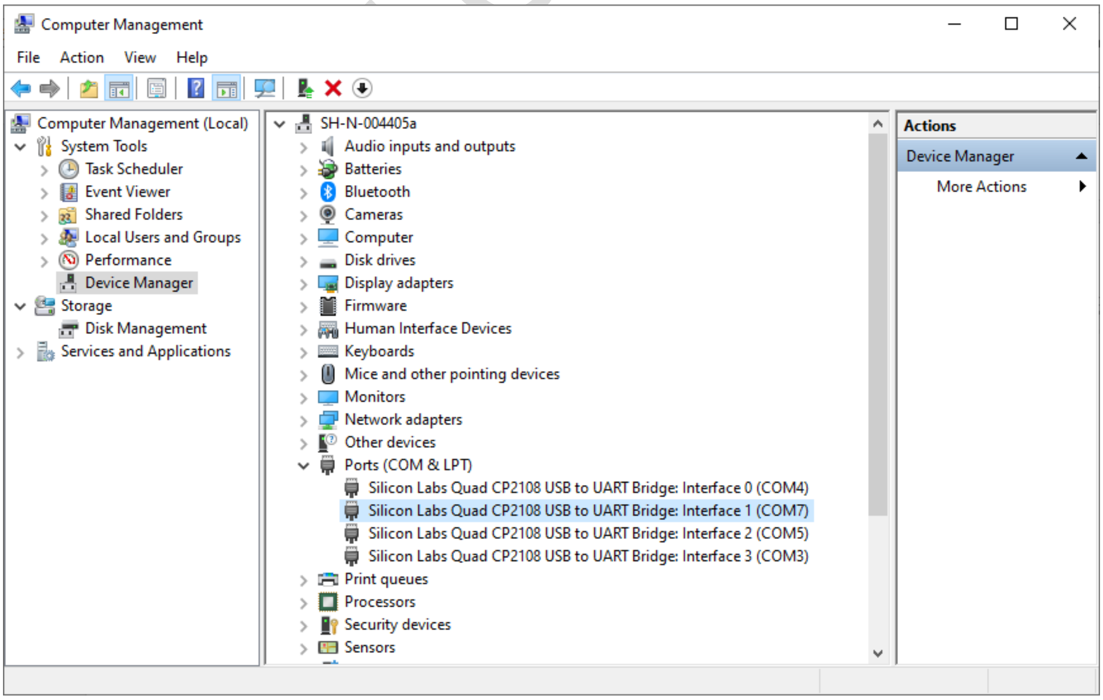

Figure 20: PC Device Manager

Subsequently, open serial port interaction tool like **MobaXterm** and select **COM7** as shown in ***Figure 21***.

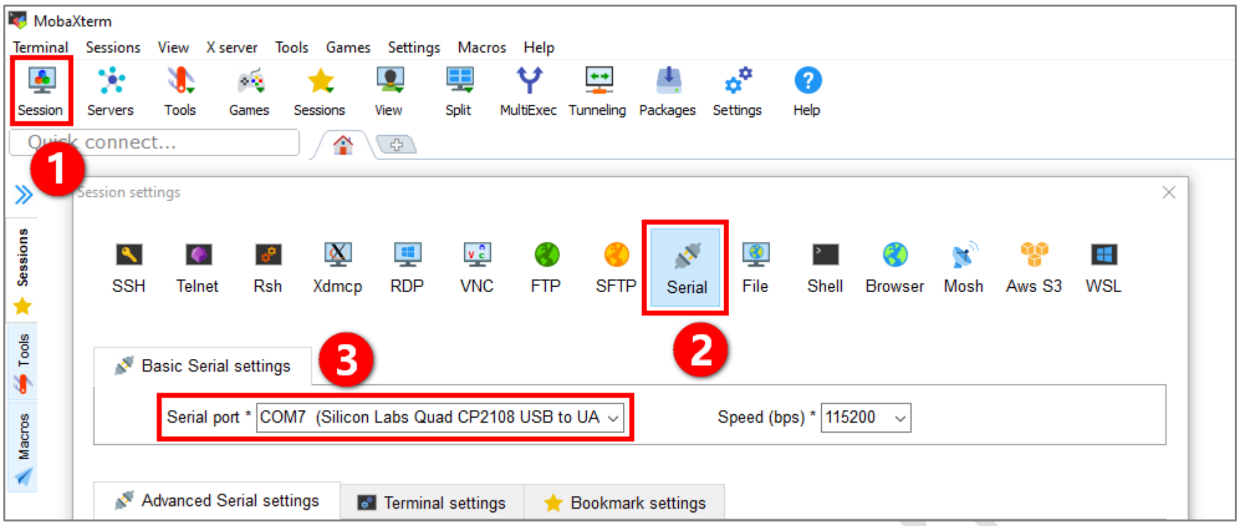

Figure 21: Serial Port Interaction Tool

Note

1) The port number will be varied in different PC. As a result, **COM7** is just a reference instead. However, **Interface 1** is mandatory to select.
2) For specific HW application note, please see ***Document [1]***.

6 Build Project

For Quectel UniKnect Project SDK, it supports various MCU. However, before developing based on this SDK, it is necessary to execute **build** command for sake of configuring designated MCU type, version number, compiling link dependencies and generating mandatory **Build** file such as **CMakeLists.txt / xxx.json**. For specific log, please refer to ***Chapter 3***.

6.1. Build Command

```
build.bat config [ChipType] [Version]
```

Note

build.bat config

This command can carry with two parameters **[Chip type] [Version]**
E.g. build.bat config STM32F413RGT6 your_firmware_version

Once both parameters **[Chip type] [Version]** are not given, former chip type and version shall be deployed instead.
When you use this SDK firstly, without former configuration, the **STM32F413RGT6** will be utilized by default. Upon version format, **Quectl_UFP_Chip_Date** will be selected.

E.g. Quectel_UFP_STM32F413RGT6_20250430

See Cmdline log in ***Figure 22.***
When it builds successfully, ***.vscode*** and ***build*** directories will be generated in the root directory of SDK. Additionally, two files- ***CMakeLists.txt*** and ***CMakePresets.json*** will be displayed as well in ***Figure 23***.

Figure 22: Build Command Execution Log

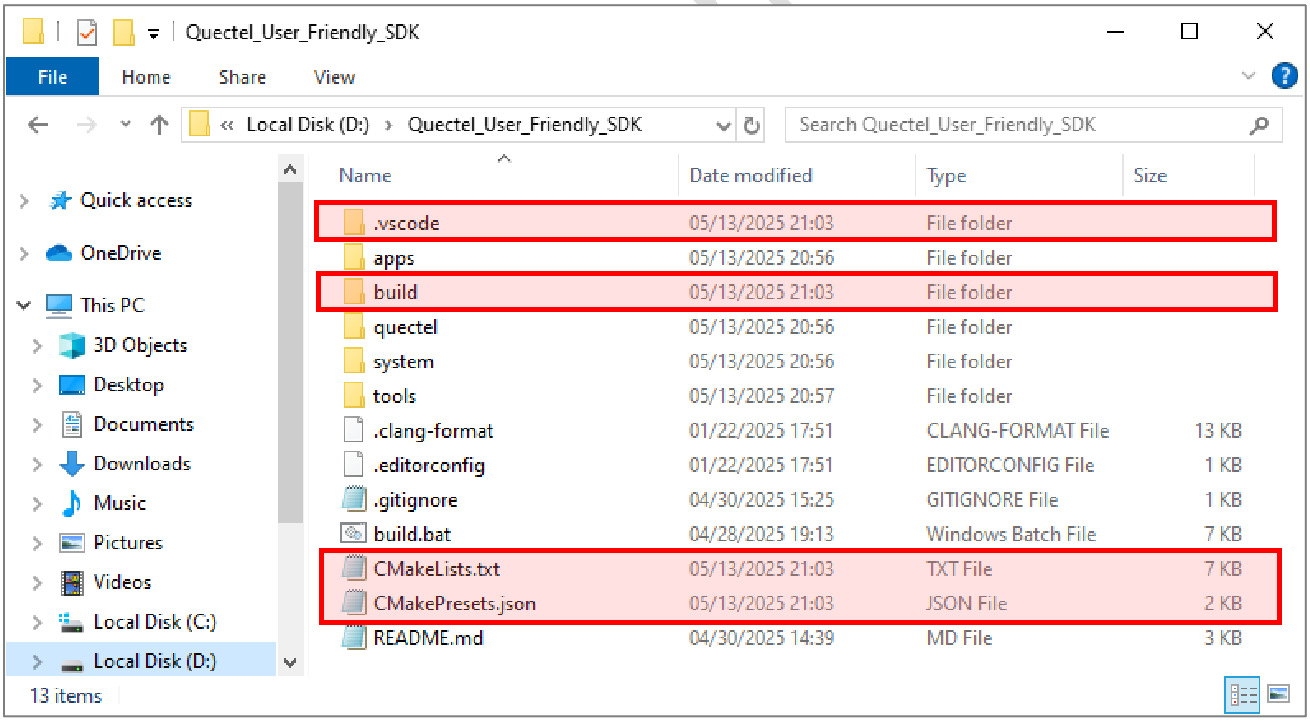

Figure 23: File Generated by Build Command Automatically

7 Develop & Debug

Two kinds of operation are used to develop & debug by Quectel UniKnect Project SDK, in this situation, the user can select either or both of them according to actual scenario.

1. **CLI:** Not rely on any IDE or code editor
2. **GUI:** VSCode + Plugin (Related parameters can be generated automatically)

Note

For user who selects Cmdline, please refer to Chapter 7.1;
For user who selects GUI, please refer to Chapter 7.2;
For user who selects both, Chapter 7.1 and Chapter 7.2 shall be taken into consideration both.

7.1. Cmdline Operation

7.1.1. Compile

See relevant command.

```
build.bat all
```

After executing ***build.bat all*** command, if it displays log as shown in ***Figure 24***, which means the compilation is successful.
Moreover, after it is a success to compile, target files such as ***elf / hex / bin / map*** will be generated in ***build*** directory. See ***Figure 25*** in detail.

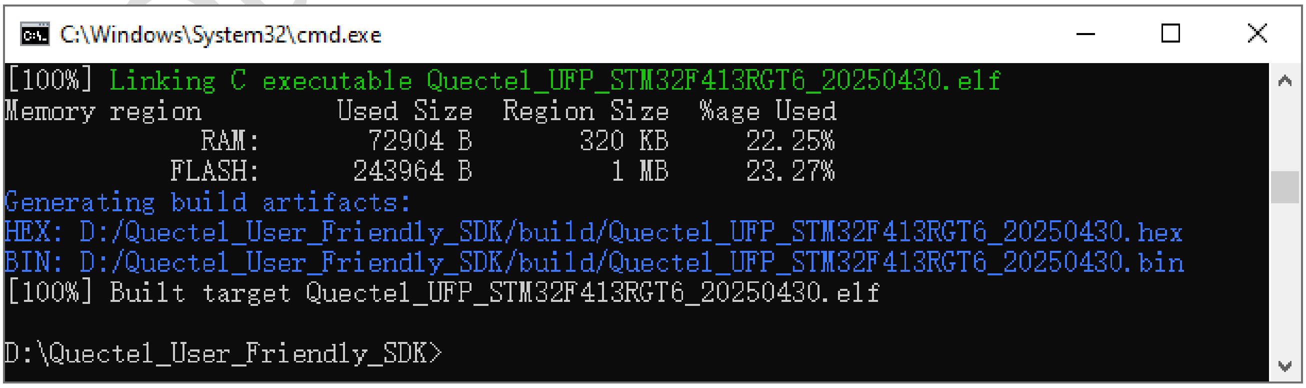

Figure 24: Compilation Log

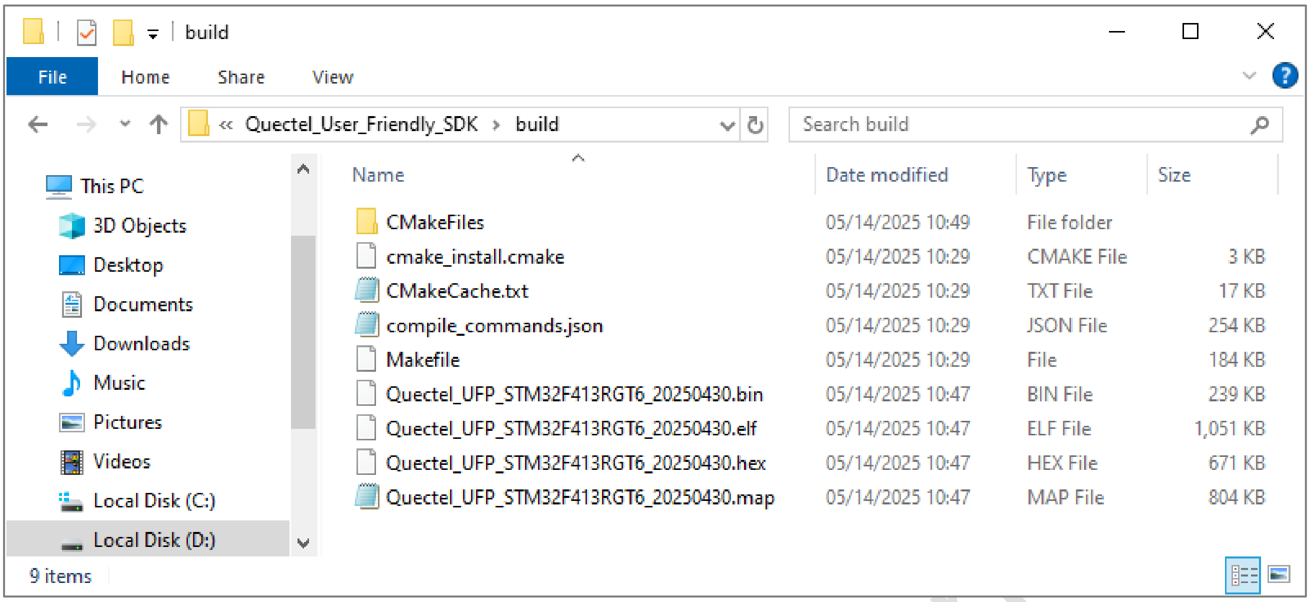

Figure 25: Compile Newly-born Target File

7.1.2. Clean

See relevant command.

```
build.bat clean
```

After executing ***build.bat clean*** command, it succeeds in cleaning if log as ***Figure 26*** displays.
If it fails to execute command or demands cleaning fundamentally, please delete ***build*** directory in the root directory of SDK.

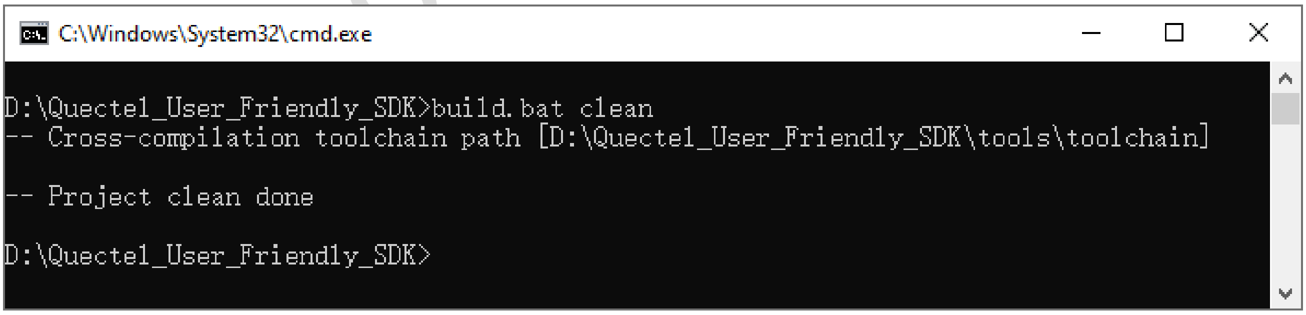

Figure 26: Log Related to Clean

7.1.3. Download

See relevant command.

```
build.bat download
```

After executing ***build.bat download*** command, it illustrates the download is a success if log as ***Figure 27*** displays.

For MCU boot log, please refer to ***Figure 28***.

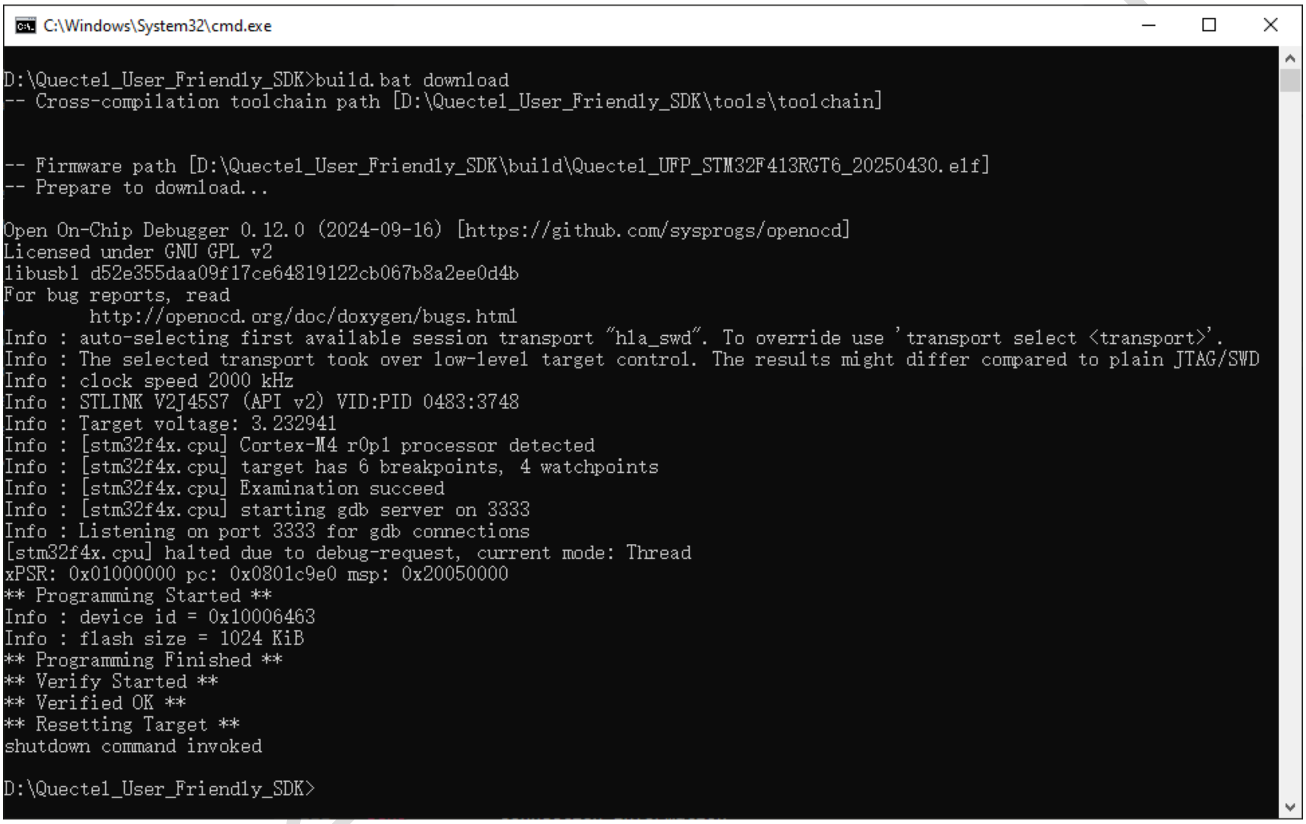

Figure 27: Log Related to Download

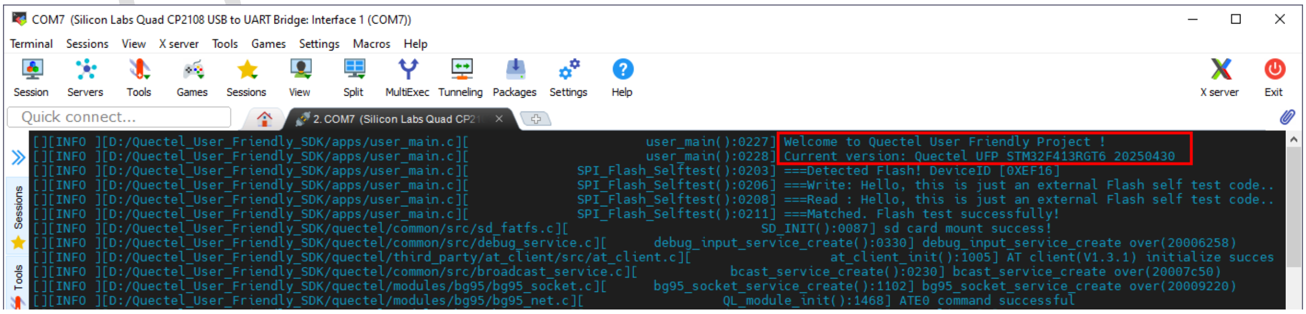

Figure 28: Boot Log

7.1.4. Debug

See relevant command

```
build.bat debug
```

After executing ***build.bat debug*** command, the system will initiate one GDB service process named OpenOCD. Once log panel displays as ***Figure 29***, it proves a success to enter debugging mode.

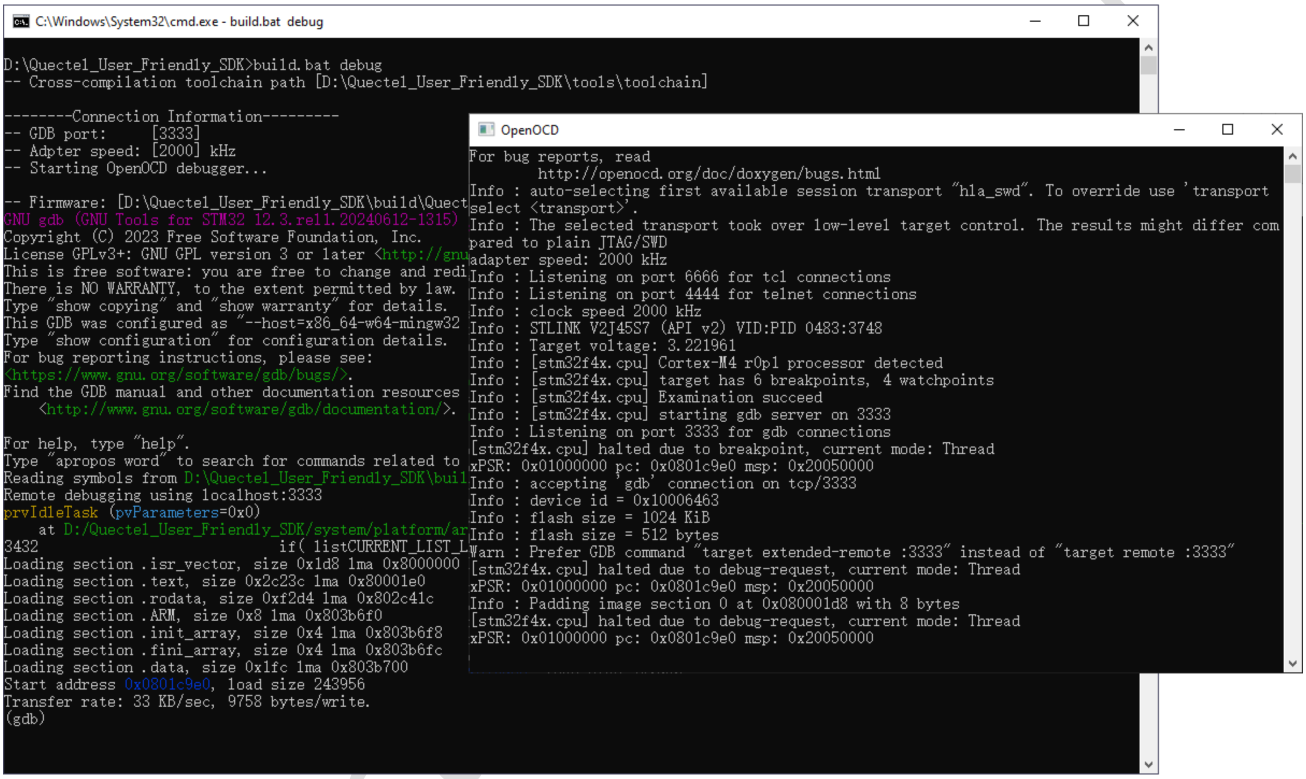

Figure 29: GDB Debugging Surface

For common commands in GDB debugging, please see ***Table 2*** below.

Table 2: Common Commands in GDB Debugging

Function	Command	Description	Sample
Set breakpoint	b <Location> or break	Set breakpoint in function or address	b main b *0x08001234
Skip over	n or next	Execute next-line codes (Skip over function)	n
Step in	s or step	Execute next-line codes (Step in function)	s
Continue running	c or continue	Continue executing till next break-point or end	c
Check Memory	x/<format> <address>	Check memory Format: Hex	x/4x 0x20000000 (Check four 32-bit values)
Check variable value	p <variable> or print	Print variable or Expression	p cnt p (uint32_t*)0x20000000
Check register	p/x $r0 ~ p/x $r15	Check ARM Register	p/x $sp (Check stack pointer)
Check all registers	info reg	Check all registers	info reg
Check call stack	bt or backtrace	Check call stack	bt
Switch stack frame	f <No.> or frame	Switch to stack frame in designated layer	frame 1
Delete break-point	delete <No.>	Delete designated break-point	delete 2
List source-code	l or list	Check source codes in current or designated location	list 20,30 (Display Line 20 & 30)
Reset MCU	monitor reset	Reset MCU	monitor reset
Quit debugging	q or quit	Quit GDB	q

7.2. GUI Operation

7.2.1. Install VSCode and Plugin

1. Download via this link-- https://code.visualstudio.com
2. Install VSCode according to prompt
3. Install VSCode plugin
   Open VSCode, click "**Extensions**" on the left or display plugin manager via "**Ctrl+Shift+X**". Subsequently, search and install following 4 plugins in searching box as ***Figure 30***.

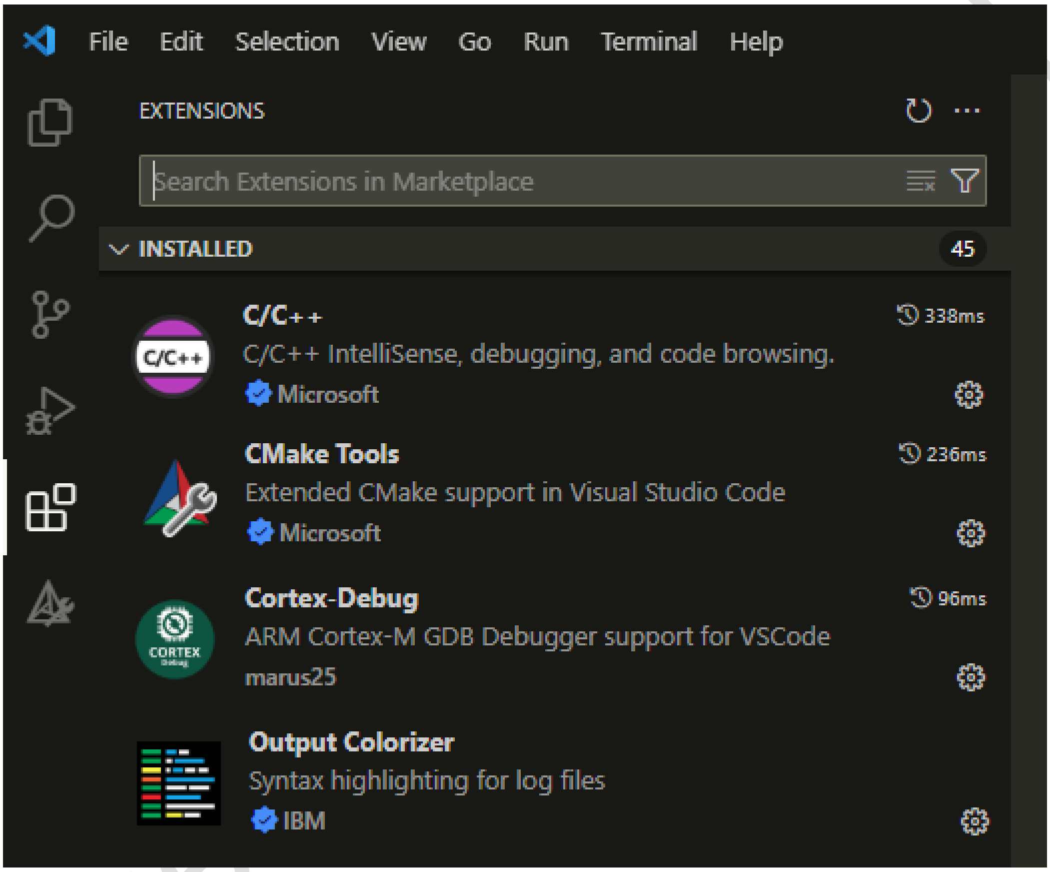

Figure 30: Install VSCode Plugins

7.2.2. Configure

Initially, please open Quectel_UniKnect_SDK Project.
As shown in ***Figure 31***, it is available to select SDK Project folder via shortcut "**Ctrl+K Ctrl+O**" or click "**Open Folder…**" in the drop-list of "**File**".

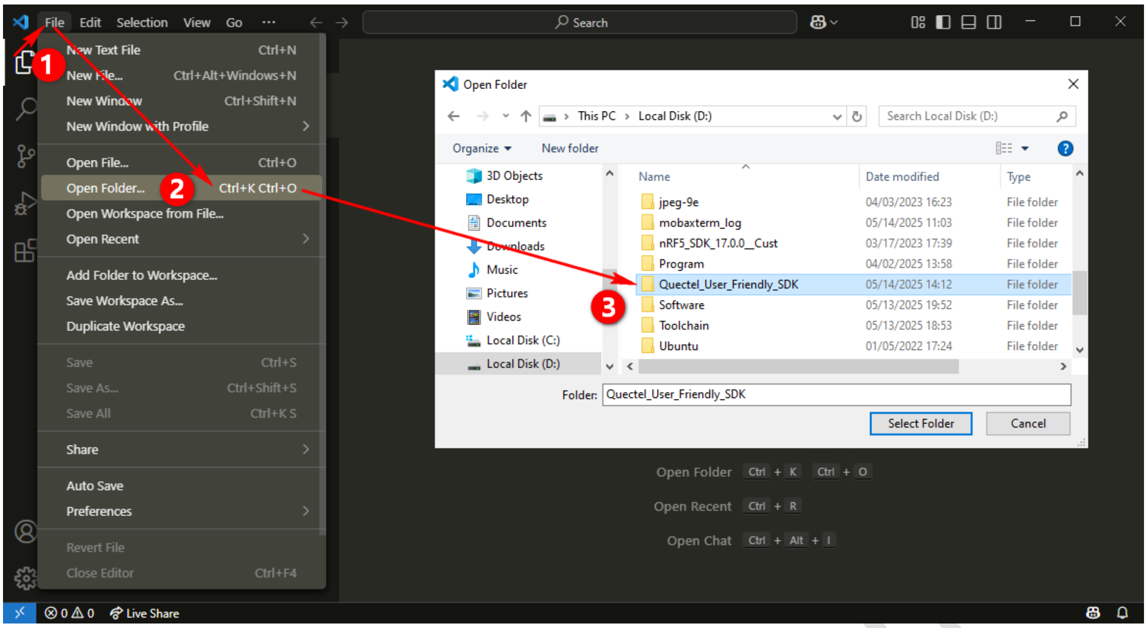

Figure 31: Open Project Folder in VSCode

Once a success to open project folder, VSCode will load CMake Tools plugin and configure automatically. Without automatic configuration or **build** deletion, it is available to re-configure by clicking “**Delete Cache and Reconfigure**” button in CMake Tools. For specific, please refer to ***Figure 32***.

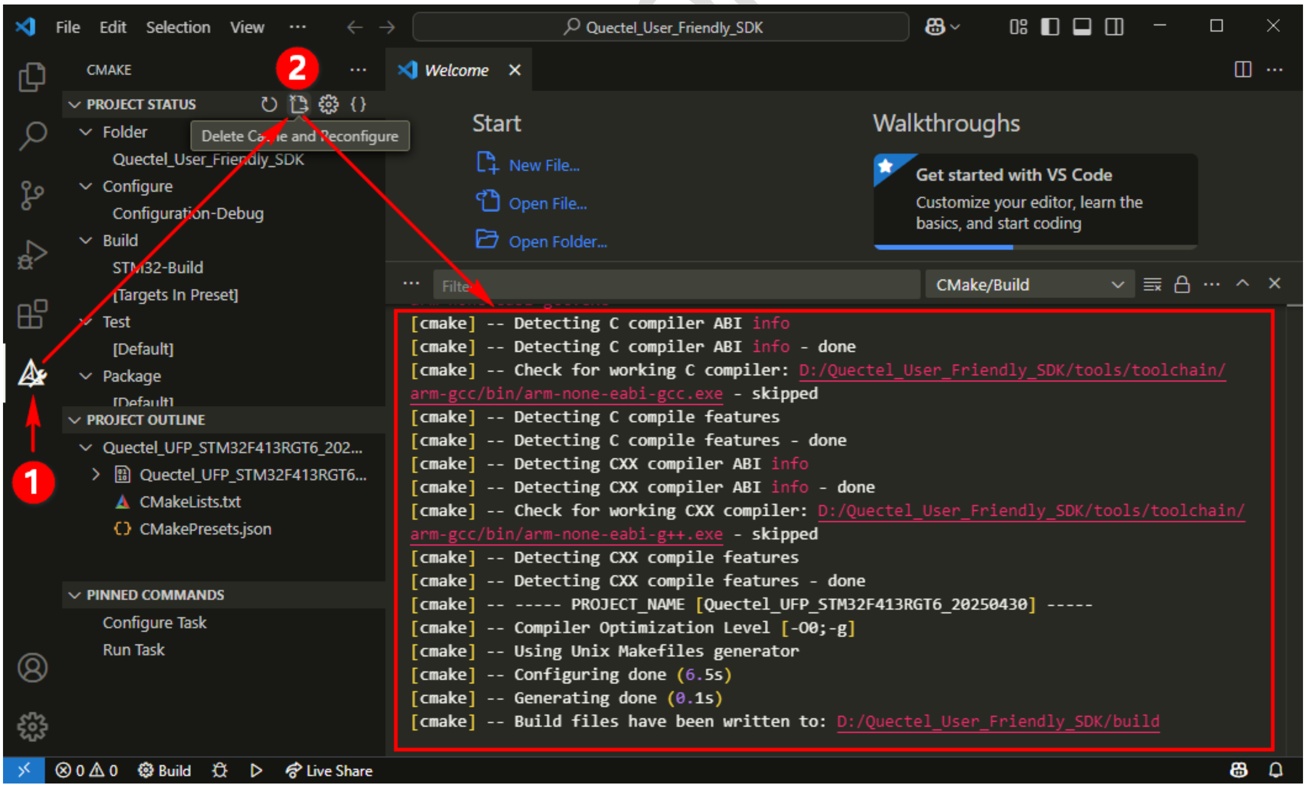

Figure 32: CMake Configuration in VSCode

7.2.3. Compile

As illustrated in ***Figure 33***, it is available to compile via shortcut “**F7**” or “**Build all projects**” in CMake Tools.

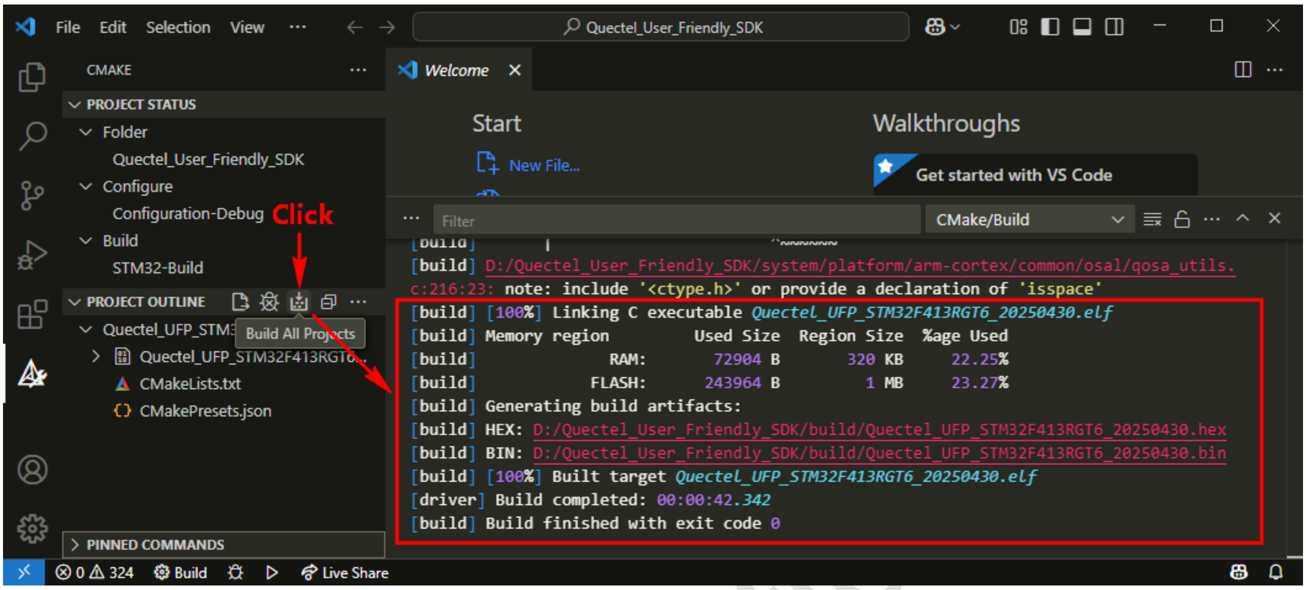

Figure 33: Compile in VSCode

7.2.4. Clean

Click “**Clean all projects**” in CMake Tools displayed as ***Figure 34***.
If it fails to execute build command or demands cleaning fundamentally, please delete build directory in the root directory of SDK.

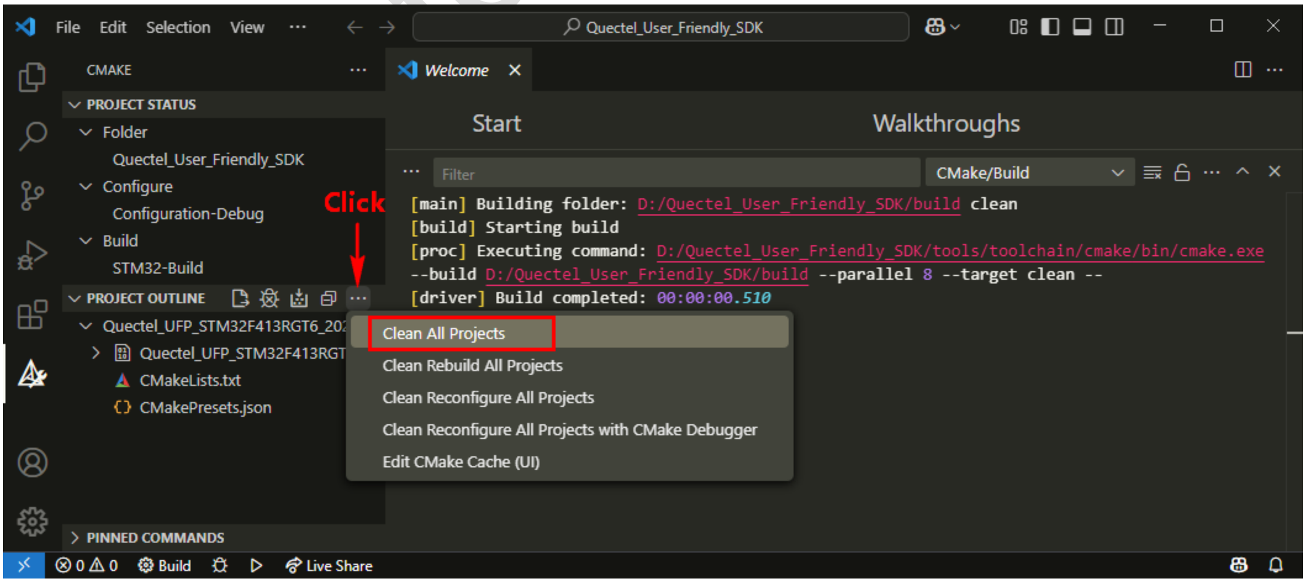

Figure 34: Clean in VSCode

7.2.5. Download

By clicking Shortcut "**Ctrl +Shift+B**" or "**Run Task…**" in the drop-list of "**Terminal**", you can click "**Download**" to start downloading in Task Panel. For details, please refer to following ***Figure 35*** & ***Figure 36***.

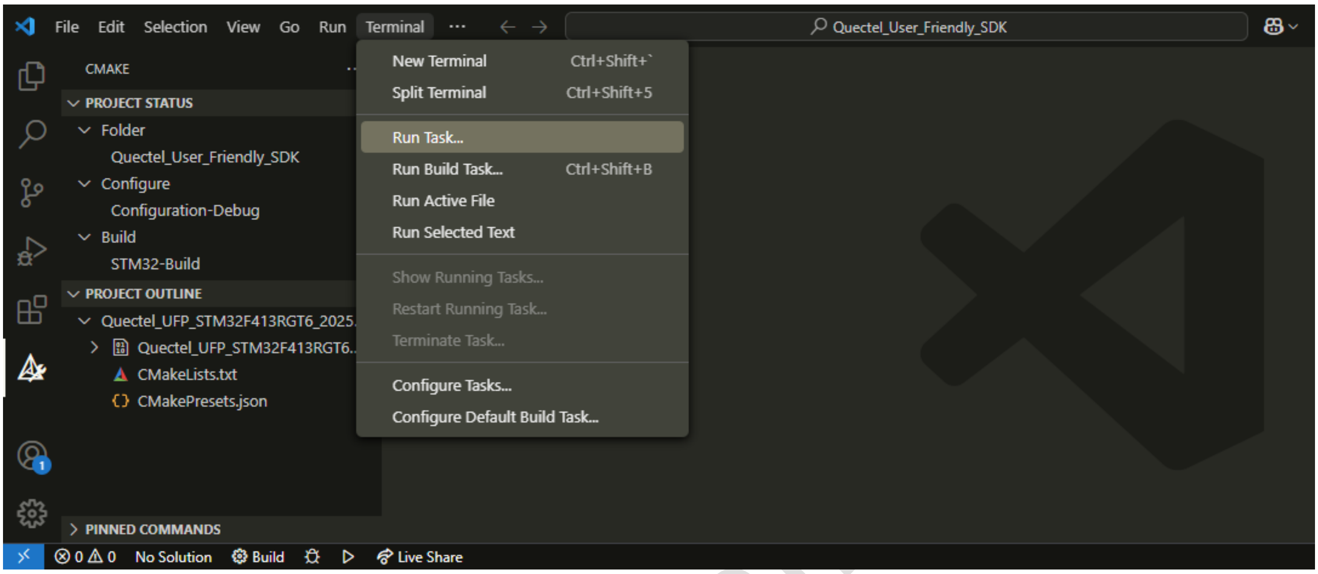

Figure 35: Open Task Panel in VSCode

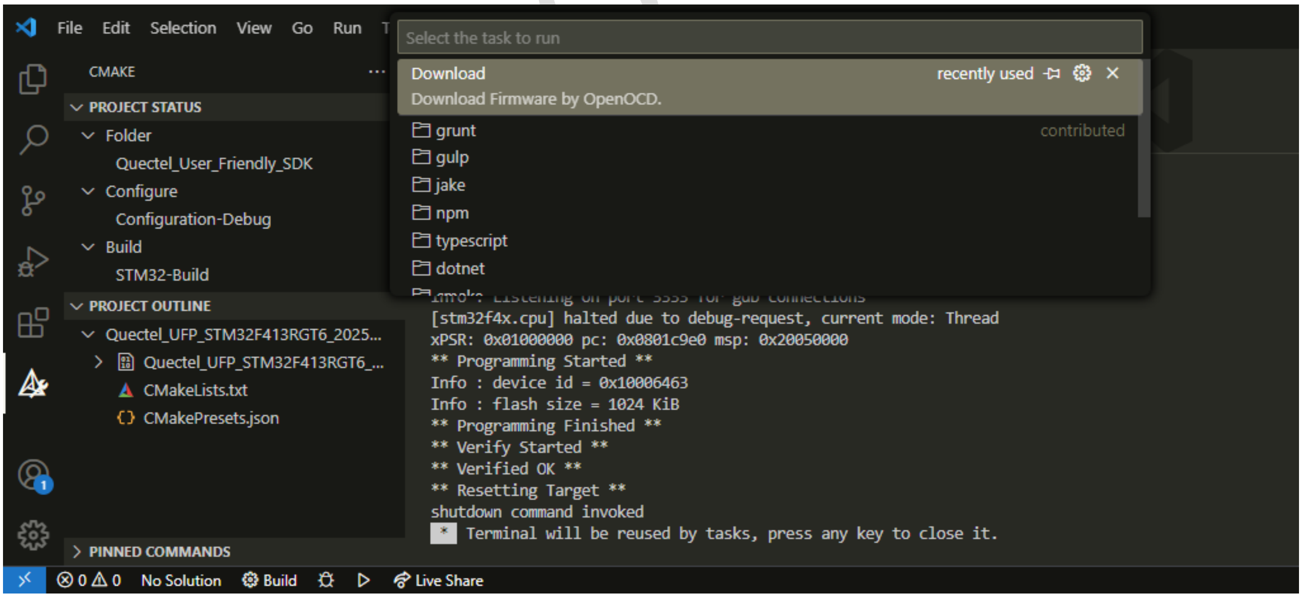

Figure 36: Download in VSCode

7.2.6. Debug

After clicking “**Configure All Projects with CMake Debugger**” in CMake Tools, the debugging button below will turn to “**Debug with OpenOCD**” once the configuration is done. See ***Figure 37*** in detail.

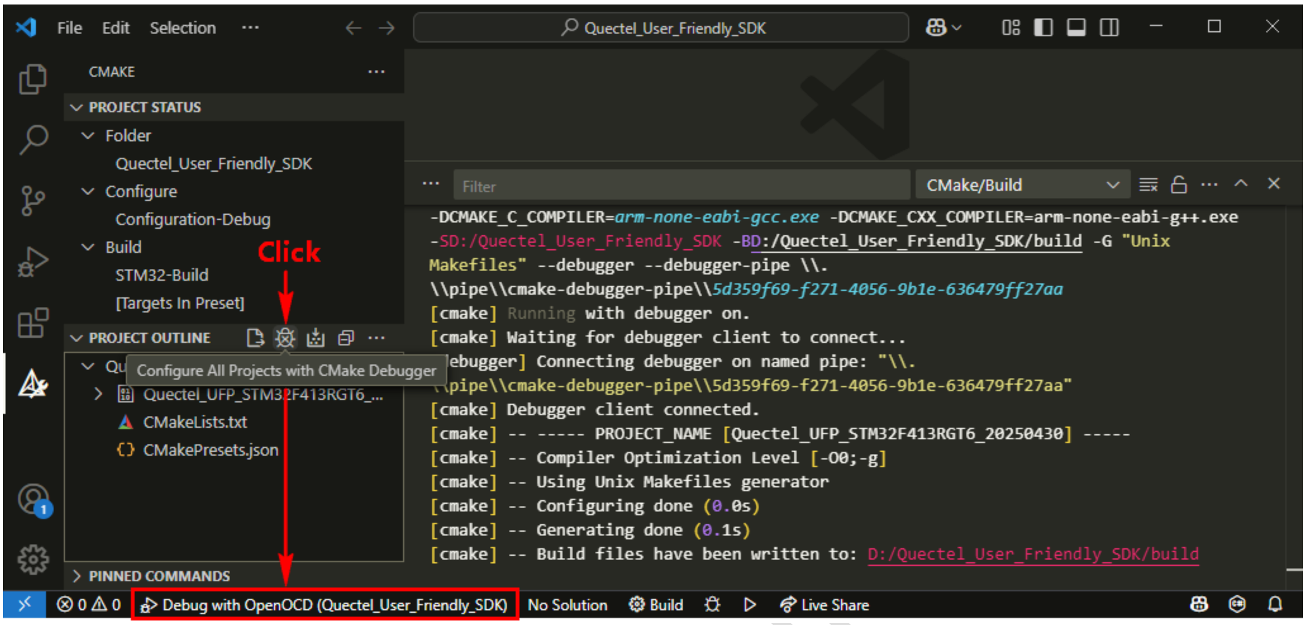

Figure 37: Configure Debugger in VSCode

Following that, by clicking “**Debug with OpenOCD**” below, corresponding configuration panel will display. In this situation, please initiate debugging by clicking it.

Once a success to initiate, one “**Debug panel**” as shown in ***Figure 38*** will occur. Additionally, the default break-point is located in the entrance of **main()**.

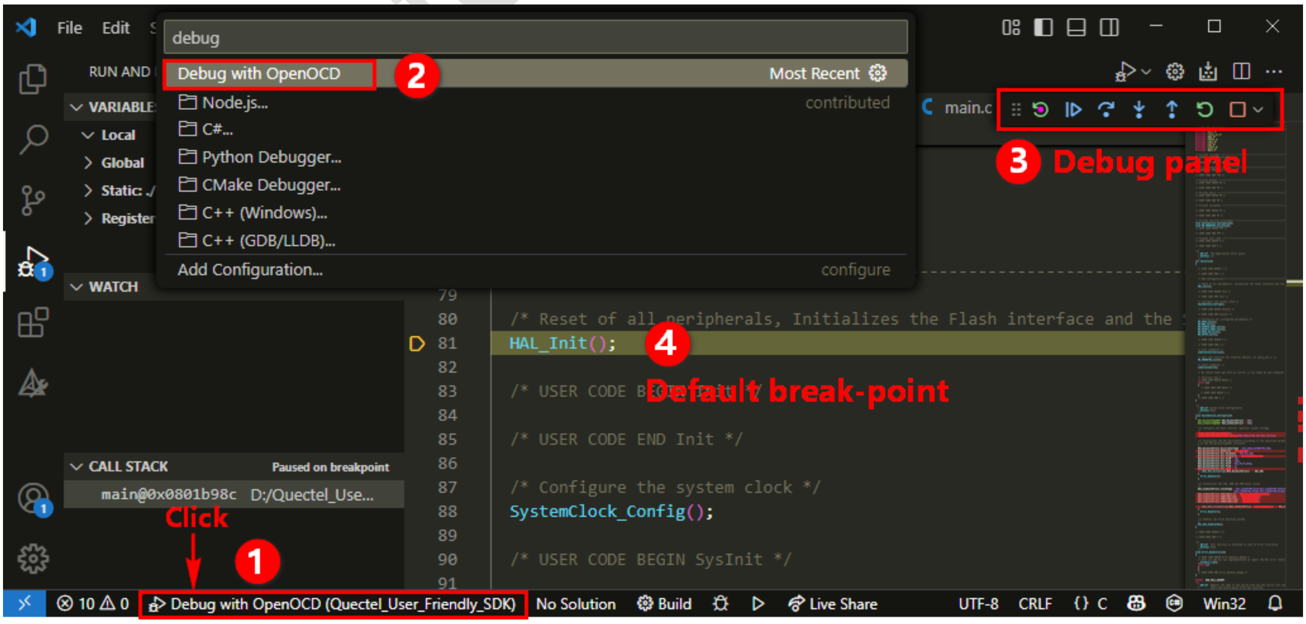

Figure 38: Debug in VSCode

Apart from above methods, you can also initiate debugging process via shortcut “**F5**”.
Subsequently, it is available to set break-point, step-in, check variable & call stack and reset. For specific, please refer to ***Figure 39***.

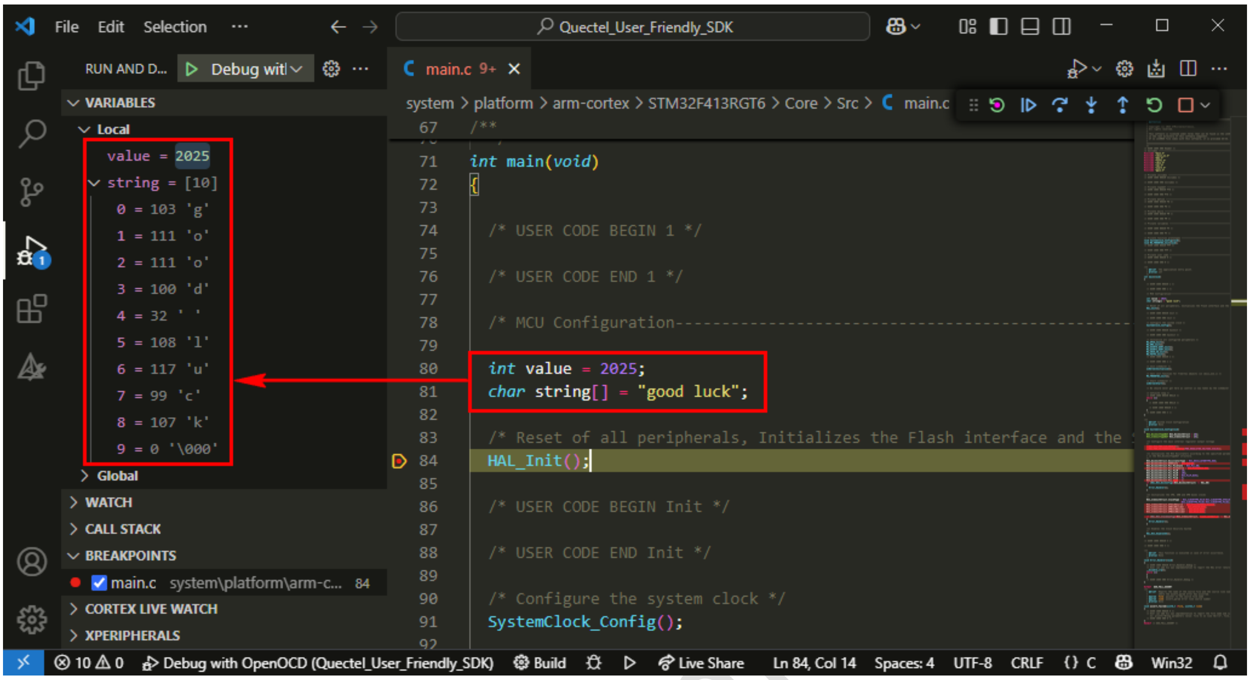

Figure 39: Debug & Check Variable in VSCode

8 Appendix Referential Documentation and Term Abbreviation

Table 3: Referential Documentation

Documentation
[1] STM32 LQFP64 EVK V2.0 User Guide V1.0-0605
[2] Quectel_QSTM32_Test_Guide_V2.0
[3] Quectel_LTE_Standard(U) Series_AT Command Manual_V1.1

Table 4: Term Abbreviation

Abbr.	Full English Name
API	Application Programming Interface
AT	Attention Command
EVK	Evaluation Kit
GCC	GNU Compiler Collection
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LED	Light Emitting Diode
MCU	Micro Controller Unit

RTOS	Real-Time Operating System
SDK	Software Development Kit
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
(U)SIM	(Universal) Subscriber Identity Module