HT MICRON SEMICONDUTORES S.A.
Av. Unisinos, 1550 | 93022-750 | São Leopoldo – RS | Brasil
www.htmicron.com.br

HT MICRON
semicondutores

iMCP HTNB32L-XXX Getting Started

Getting Started for iMCP HTNB32L-XXX System-in-Package

Classification:	CONFIDENTIAL
Doc. Type:	USER MANUAL
Revision:	v.01
Date:	04/04/2023
Code:	HTNB32L-XXX-UM0001

SUMMARY

DOCUMENT INFO

This document provides technical information about the first steps with the iMCP HTNB32L-XXX. It is intended to contribute only the necessary information to run the basic examples available in its SDK. More specific details can be found at iMCP HTNB32L-XXX GitHub Repository.

1. GENERAL DESCRIPTION

The iMCP HTNB32L-XXX is a highly compact and low-power wireless communication MCO/SiP featuring Qualcomm QCX-212 LTE IoT Modem supporting single-mode 3GPP Release 14 Cat. NB2 IoT connectivity. Its SDK (Software Development Kit) provides OpenCPU solutions based on a FreeRTOS system, where users can embed their own IoT application, as well as AT Commands, used in a master-slave model.

2. SETUP ENVIRONMENT

This section describes the setup environment that users should build in order to start developing and testing their application. It is strongly recommended to follow all instructions pointed out here to reproduce the expected results.

2.1. Operating System

Although the HTNB32L-XXX SDK could be easily adapted to Unix systems, the current version was developed to run only in the following OS:

- Windows 10: 32 or 64 bits.
- Windows 11: 32 or 64 bits.

2.2. iMCP HTNB32L-XXX SDK

Please contact HT Micron's Commercial Team (imcp@htmicron.com.br) to request the iMCP HTNB32L-XXX SDK.

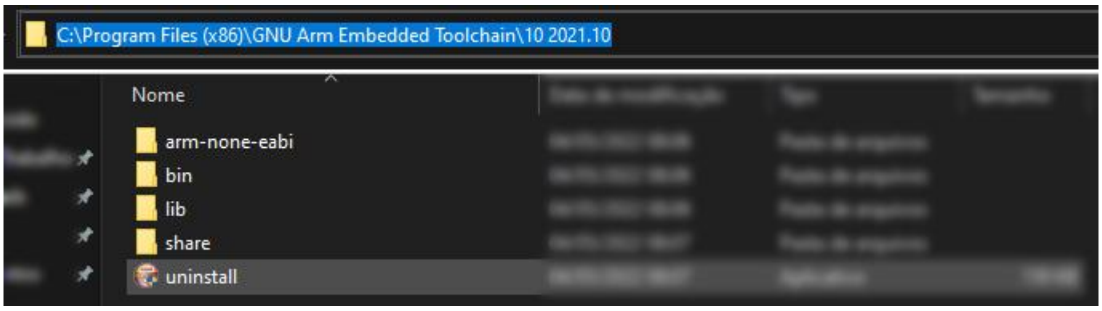
iMCP HTNB32L-XXX SDK is composed by the following resources:

- Visual Studio Code Workspace (with debugging and flashing features).
- Firmware Examples.
- Software Applications
- Flashing tool.
- Application notes and User Guides.

2.3. Toolchain Download and Installation

GNU ARM Embedded Toolchain, which includes GNU Compiler (GCC), is used to compile the HTNB32L-XXX firmware. The following steps show how the compiler should be installed:

1. Go to ARM Developer website and download and install the lasted version of GNU ARM Embedded Toolchain (*gcc-arm-none-eabi-10.3-2021.10-win32.exe*).
2. Open the directory where you have installed it (usually "*C:\Program Files (x86)\GNU Arm Embedded Toolchain*"):


C:\Program Files (x86)\GNU Arm Embedded Toolchain\10 2021.10

```
Nome
arm-none-eabi
bin
lib
share
uninstall
```

Figure 1: Toolchain path.

3. Copy the "*10 2021.10*" (it depends on the toolchain version you had installed) directory and paste it to your "C:\".
4. Rename the directory to "*gcc_arm*":

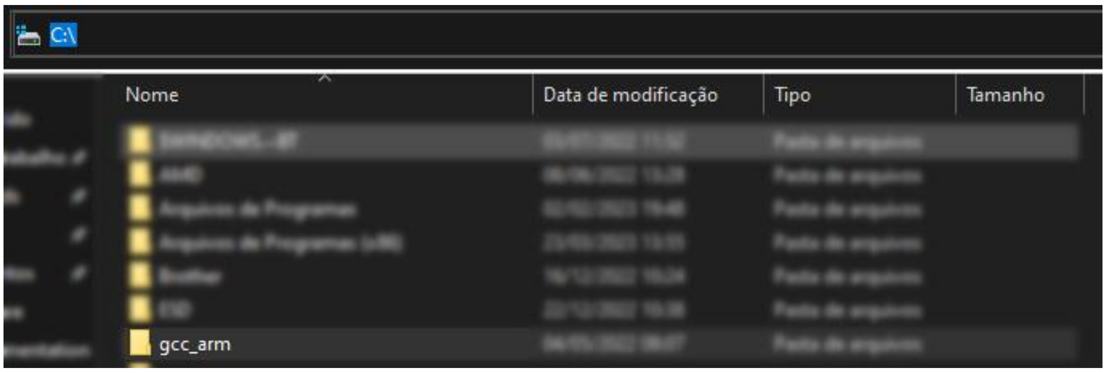


Figure 2: Renaming compiler path to "*gcc_arm*..

NOTE

Steps 3 and 4 are optional. Users could choose any other name for this directory or even keeping it in the

2.4. Make Download

The compilation procedure requires the Make tool installed on Windows. Following instructions indicates how to download and install Make on Windows using Chocolatey:

1. Install Chocolatey: HERE
2. Open a PowerShell terminal and run the following command to install Make on Windows:

choco install make

2.5. Git Download and Installation

The HTNB32L-XXX SDK is stored on a GitHub repository. Users should be familiar with Git tools, such as Git Bash and Git GUI. Visit Git website to download and install Git on your Windows OS.

2.6. Visual Studio Code Workspace

Even though users can choose any other text editor of their preference, Visual Studio Code is the programming tool recommended to utilize with HTNB32L-XXX SDK. A complete and ready-to-use workspace, with debugging and flashing features, was built on VSCODE to support developers in their programming process.

2.6.1. VSCODE Extensions

To benefit from all features available in this workspace, the following list of VSCODE Extension must be installed:

1. C/C++.
2. Cortex-Debug.

3. Embedded Tools.
4. Memory View.
5. Makefile Tools.
6. RTOS Views.
7. Serial Monitor.
8. ARM Assembly.
9. Python.

2.6.2. VSCODE Default Terminal Profile

Due to some specific characteristics of Windows CMD, it is strongly recommended to change the VSCODE default terminal to the Git Bash terminal. Instructions below show how to do it:

1. Open your VSCODE.
2. Click on "Terminal", "New Terminal", as it is shown in Figure 3:

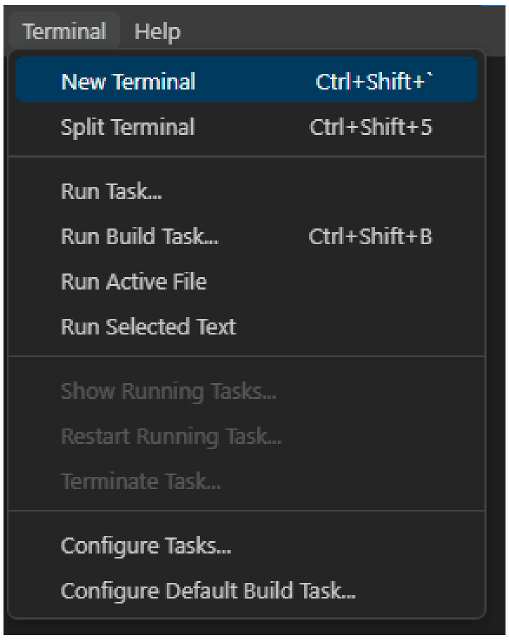


Figure 3: Opening a new terminal on VSCODE.

3. Go to "Select Default Profile" and change the default terminal to "Git Bash". Figure 4 shows where to find the "Select Default Profile" option and Figure 5 shows the CMD terminal being replaced to Git Bash as default terminal profile.

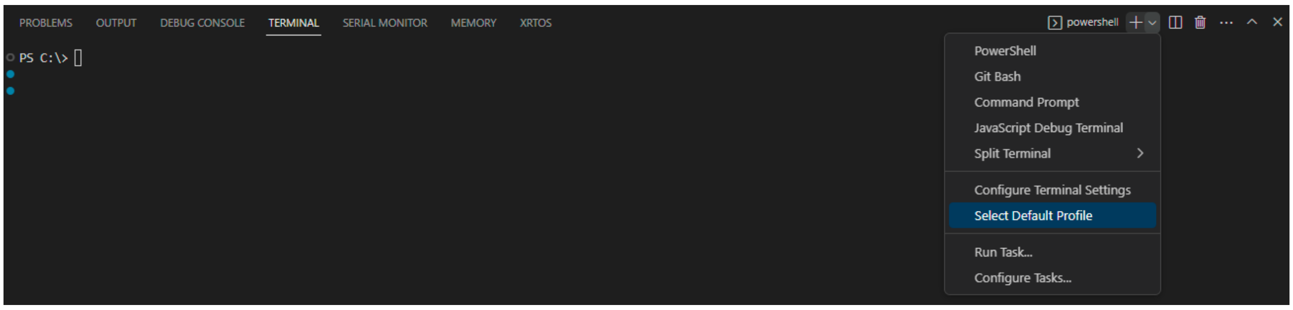
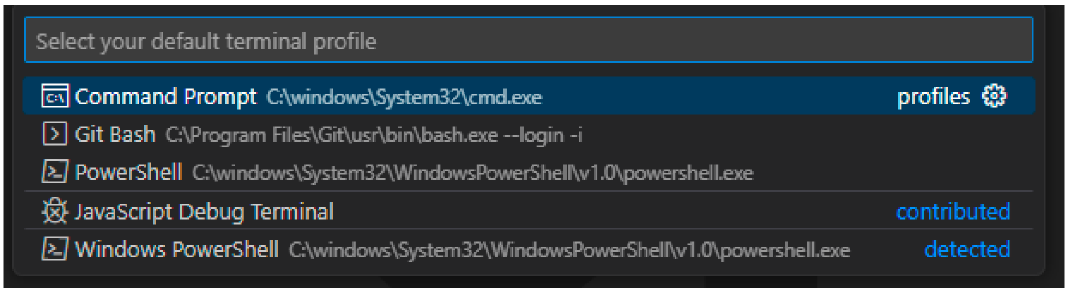

```
PS C:\>
```

Figure 4: "Select Default Profile" location.



Select your default terminal profile

Command Prompt C:\windows\System32\cmd.exe profiles
Git Bash C:\Program Files\Git\usr\bin\bash.exe --login -i
PowerShell C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe
JavaScript Debug Terminal contributed
Windows PowerShell C:\windows\System32\WindowsPowerShell\v1.0\powershell.exe detected

Figure 5: Replacing default terminal to Git Bash.

2.7. Python Packages

Scripts developed in Python are required to run the workspace built as programming tool for HTNB32L-XXX users. **It is mandatory to download and install Python 3.9.8 or later**. Table 1 shows the list of Python packages that must be installed:

Table 1: Python packages.

Package	Command
PyQt5	pip install PyQt5
PyQt5-Tools 5.15.9.3.3	pip install pyqt5-tools
PySerial	pip install pyserial

2.8. Hardware

To start developing an application with the HTNB32L-XXX, users can design their own hardware or request a breakout board (test board) to HT Micron's Commercial Contact: imcp@htmicron.com.br.

Please check out the iMCP HTNB32L-XXX GitHub Page to search for hardware documentation and PCB examples.

2.9. Flashing Setup

The HTNB32L-XXX has an internal bootloader flashed on its ROM that can be used to upload a new firmware through UART. Figure 6 presents a possible flashing setup, using a FTDI as USB-serial converter, and Table 2 describes its connections.

Table 2: HTNB32L-XXX and FTDI connection description.

HTNB32L-XXX Pin	FTDI Pin	Description
UART1_TX (GPIO12)	RX	HTNB32L-XXX UART transmitter connected to the FTDI UART receiver.
UART1_RX (GPIO13)	TX	HTNB32L-XXX UART receiver connected to the FTDI UART transmitter.
GND	GND	Ground of both sides equalized.

3V3	-	HTNB32L-XXX power supplier in 3.3V.
GPIO1	-	HTNB32L-XXX boot pin. Starts the flashing process if the device is booted with this pin at a low logic level. Should be connected to a pull-up resistor.
RST	-	Reset pin. Resets the device when at low logic level. Should be connected to a pull-up resistor.

NOTE

Any other USB-serial converter able to transmit up to 921600 bps (baud rate) can be chosen for this task.

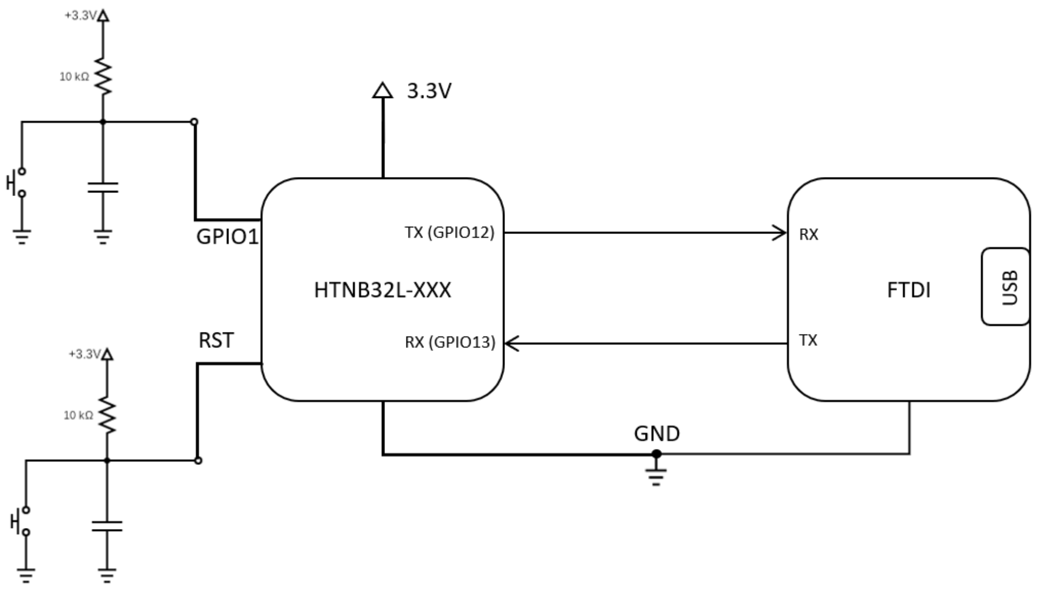


Figure 6: Possible flashing setup.

2.10. Debugging Setup

HTNB32L-XXX SDK has a debugging environment built on VSCODE, which uses J-Link as debugger probe. To take advantage of this feature, it is mandatory to install the latest version of SEGGER J-Link software pack.

- J-link Software and Documentation Pack Download: HERE

There are two possible hardware setups for debugging purposes. The first one uses a SEGGER J-Link device and the second one takes advantage of a STM32 Nucleo. Following subsections describe all these options.

2.10.1. SEGGER J-Link Setup

Figure 7 demonstrates a possible setup to run the VSCODE debugger with a SEGGER J-Link. The SEGGER J-LINK pinout for ARM's SWD can be found HERE. The setup description is provided in Table 3.

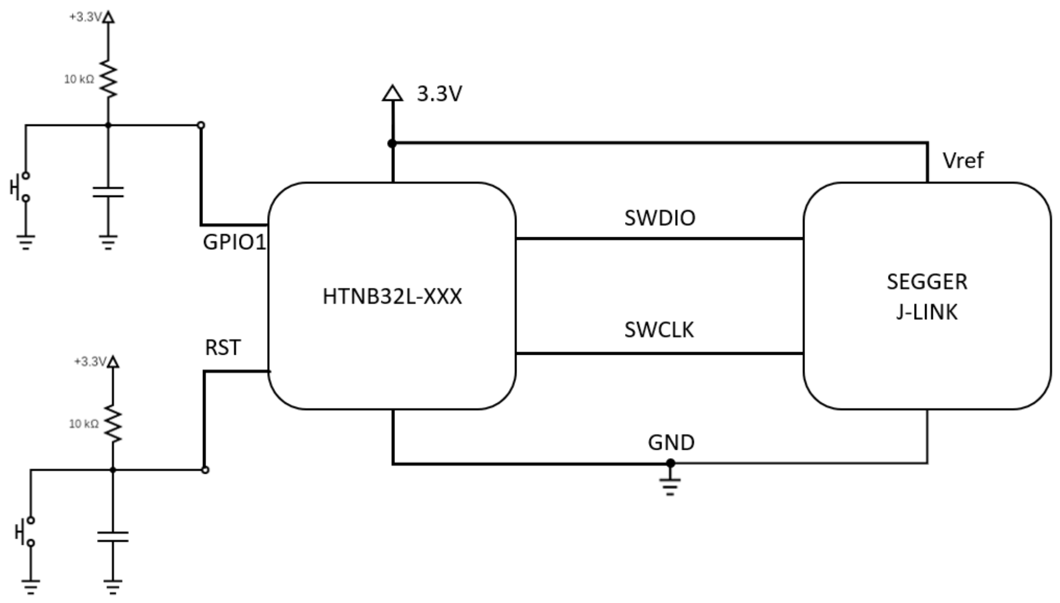


Figure 7: SEGGER J-Link debug setup.

Table 3: HTNB32L-XXX and SEGGER J-Link connection description.

HTNB32L-XXX Pin	SEGGER J-Link Pin	Description
SWDIO	SWDIO	Single bi-directional data pin.
SWCLK	SWCLK	Clock signal to target CPU.
GND	GND	Ground pin.
3V3	VREF	Target reference voltage. Used to create the logic-level reference for the input comparators and to control the output logic level to the target.
GPIO1	-	HTNB32L-XXX boot pin. Starts the flashing process if the device is booted with this pin at a low logic level. Should be connected to a pull-up resistor.
RST	-	Reset pin. Resets the device when at low logic level. Should be connected to a pull-up resistor.

2.10.2. ST-LINK Setup

SEGGER company offers a script that converts ST-LINK devices into a J-LINK. The conversion is done by uploading the ST-LINK firmware to a new code J-LINK compatible. This procedure is possible only in ST-LINK devices embedded on STM32 Nucleo and Discovery Boards. To download the converter script and to check its tutorial, please go to SEGGER Website.

- STM32 Nucleo example: HERE
- Discovery Boards example: HERE

Figure 8 presents a possible setup using a STM32 Nucleo. This setup also includes a USB-serial converter, implemented by the on-board ST-LINK and which can be used as a flashing interface, replacing the FTDI mentioned in Subsection 2.9, and two buttons for GPIO1 bootloader pin and NRESET, both with similar circuits as presented in Figure 6 and Figure 7. Please check UM1724 – STM32 User Manual for more pinout information.

1. Open SB13 and SB14 resistors, on the bottom side of STM32 Nucleo board. Figure 9 shows the location of these two resistors.
2. Open CN2 jumpers. These jumpers are responsible to connect ST-LINK to the on-board microcontroller (STM32 MCU). By opening this connector, it makes possible to use the ST-LINK to access an external device, such as HTNB32L-XXX.
3. Connect the CN4 pins (ST-LINK SWD pins) to HTNB32L-XXX. Table 4 shows the STM32 Nucleo CN4 pinout. The connections between HTNB32L-XXX and ST-LINK are the same as mentioned in Table 3.

Table 4: STM32 Nucleo CN4 pinout.

Pin	CN4	Designation
1	VDD_TARGET	VDD from application.
2	SWCLK	SWD Clock.
3	GND	Ground.
4	SWDIO	SWD data input/output.
5	NRST	RESET of the target.
6	SWO	Reserved.

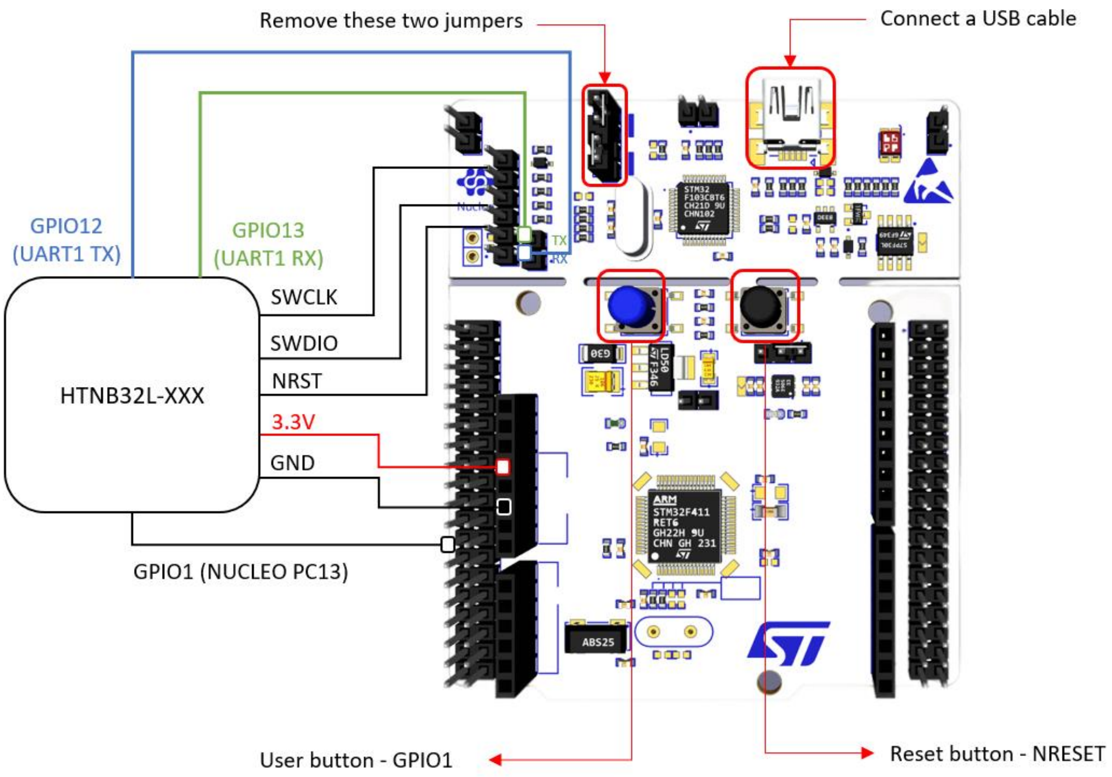


Figure 8: STM32 Nucleo + HTNB32L-XXX possible setup.

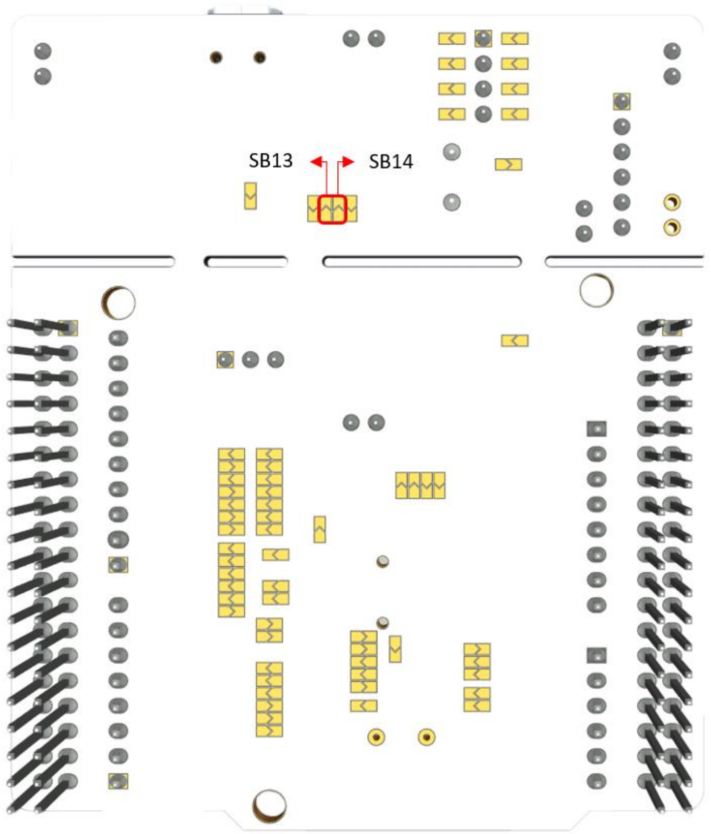


Figure 9: STM32 Nucleo bottom view.

3. IMCP HTNB32L-XXX SDK

The Software Development Kit (SDK) built for HTNB32L-XXX is intended to provide a complete environment for programmers develop their own firmware application. It contains all libraries and examples needed for an OpenCPU solution, where users can embed their custom code. Following sections describe how HTNB32L-XXX SDK is assembled and how to configure and operate it.

3.1. Cloning SDK

After having requested access to the HTNB32L-XXX SDK and having received the proper authorization, the content will be available in a private GitHub repository, where customers would be able to clone and change it according to their needs.

HTNB32L-XXX SDK repository will be available here: https://github.com/htmicron/HTNB32L-XXX-SDK

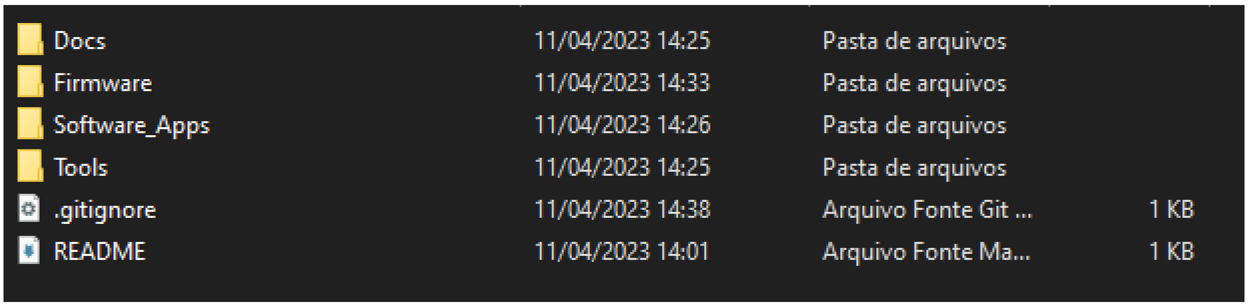
How to clone the HTNB32L-XXX SDK:

1. Download and install Git on your Windows OS.
2. Open Git Bash.
3. Navigate to the directory where the HTNB32L-XXX SDK will be cloned.
4. Finally, paste the following command line to clone the repository:

git clone git@github.com:htmicron/HTNB32L-XXX-SDK.git

3.2. SDK Directories

This subsection describes the main directories available on HTNB32L-XXX SDK. Figure 10 shows the directories available in the SDK and Table 5 describe them:



```
Docs            11/04/2023 14:25   Pasta de arquivos
Firmware        11/04/2023 14:33   Pasta de arquivos
Software_Apps   11/04/2023 14:26   Pasta de arquivos
Tools           11/04/2023 14:25   Pasta de arquivos
.gitignore      11/04/2023 14:38   Arquivo Fonte Git ...   1 KB
README          11/04/2023 14:01   Arquivo Fonte Ma...     1 KB
```

Figure 10: SDK directories.

Table 5: SDK directories description

Name	Description
.Docs	Directory where Application Notes and documents related to firmware and software are stored.
Firmware	Firmware workspace environment and examples.

	This directory also contains binary files that can be directly flashed to a HTNB32L-XXX device.
Software_Apps	Software developed as accessory for some specific firmware applications.
Tools	Development tools required

Table 6 describes the Firmware directory, where the workspace environment for HTNB32L-XXX can be found.

Table 6: Firmware directory description.

Name	Description
.vscode	Directory where all debugging and flashing environment are described (tasks, launch and settings etc).
Applications	Where the application examples for the HTNB32L-XXX are implemented. It is also where users should implement their own firmware.
Build	Directory where the binary and object files generated from an APP compilation will be stored. This directory is created after compiling an application available on SDK.
Debug	Directory that contains every script used in the flashing and debugging process.
SDK	Where driver libraries and their respective header files are stored on.

3.3. OPENING VSCODE WORKSPACE

The following guidelines shows how to properly launch the VSCODE workspace:

1. Go back to Subsection 2.6 to double check if the VSCODE environment was properly configured.
2. Open VSCODE on your Windows OS.
3. Click on “File”, “Open Folder…” as is shown in Figure 11:

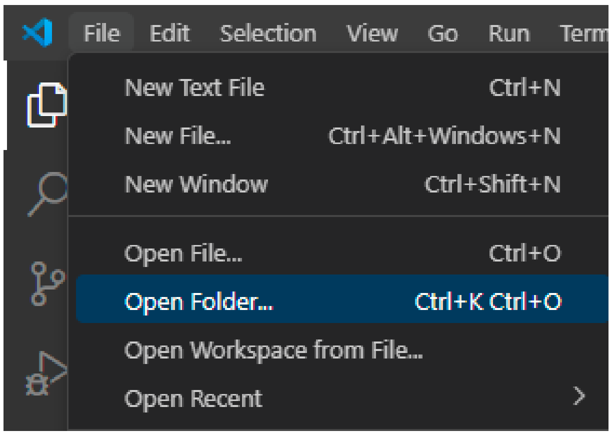


Figure 11: Opening VSCODE environment.

4. Go to where the HTNB32L-XXX SDK repository was cloned and select the "*Firmware/HTNB32L-XXX-SDK*" directory:

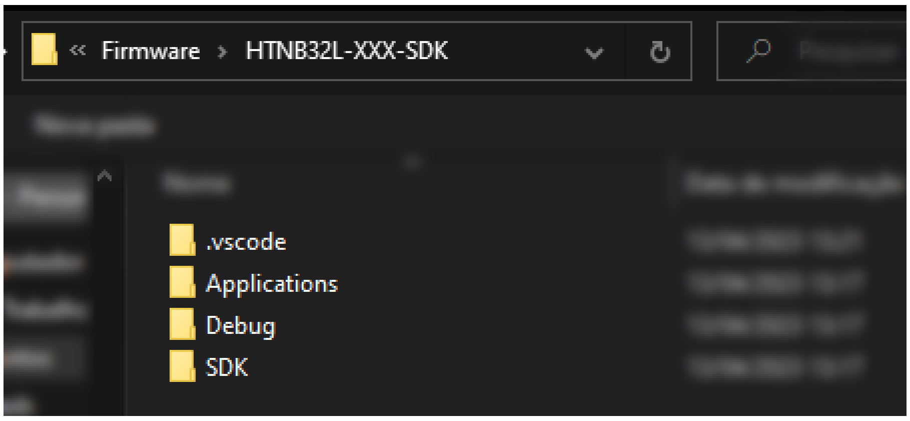


Figure 12: Firmware directory.

5. Figure 13 shows the VSCODE workspace properly launched:

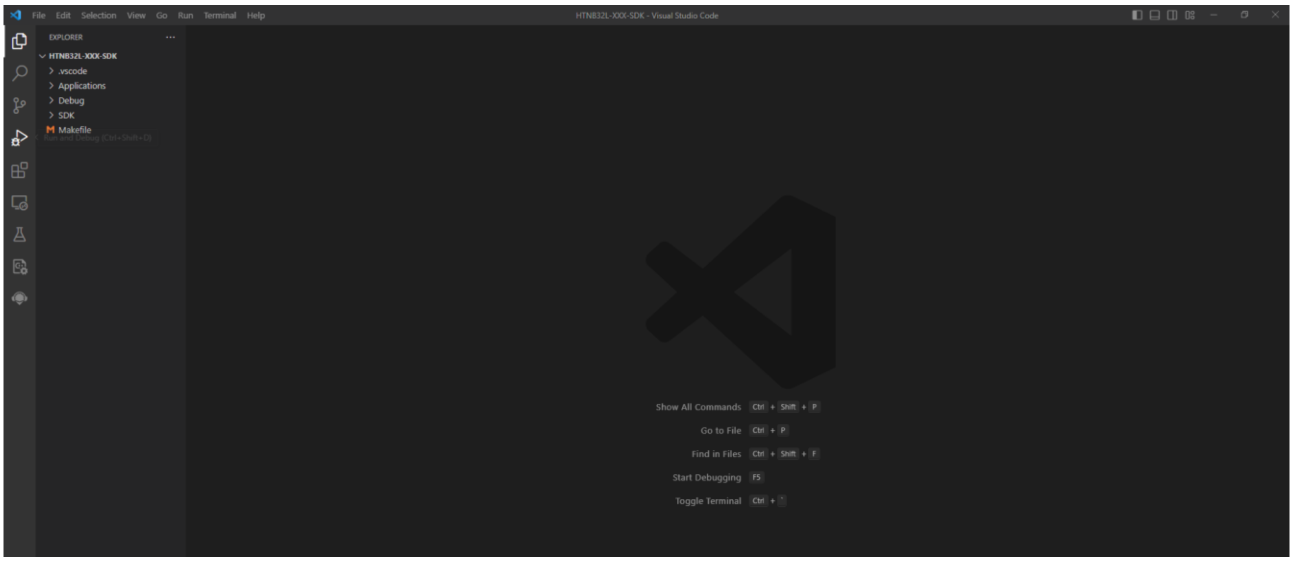


Figure 13: HTNB32L-XXX-SDK opened on VSCODE editor.

3.4. Compiling Procedure

There are two different ways to compile a firmware example available on HTNB32L-XXX SDK. The most direct one uses the VSCODE environment to call the compilation task described on "*.vscode*" directory (Direct Compilation) and the second one executes the compiler by command line (CLI Compilation).

3.4.1. Compiler Path

Follow the instructions mentioned on Section 2.3 to download and install GCC ARM toolchain. In order to change the compiler path, two different parameters must be replaced:

1. "*GCCLIB_PATH*", located at "*Firmware/HTNB32L-XXX-SDK/SDK/PLATtools/scrips/Makefile.vars*" file.
2. "*cortex-debug.gdbPath*", located at "*Firmware/HTNB32L-XXX-SDK/.vscode/settings.json*" file.

3.4.2. Direct Compilation

The following steps show how to start a Direct Compilation, utilizing the VSCODE workspace built specially for HTNB32L-XXX SDK:

1. Open the HTNB32L-XXX-SDK directory in your VSCODE editor (please refer to Subsection 3.3 for more details).
2. Press "*CTRL+SHIFT+B*" to open the task list with all tasks available in this workspace:

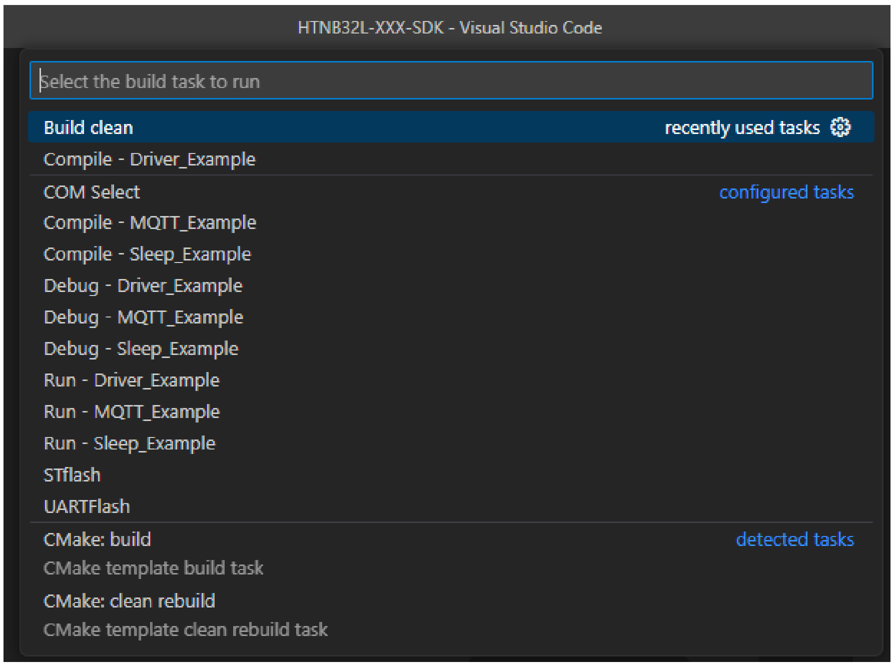
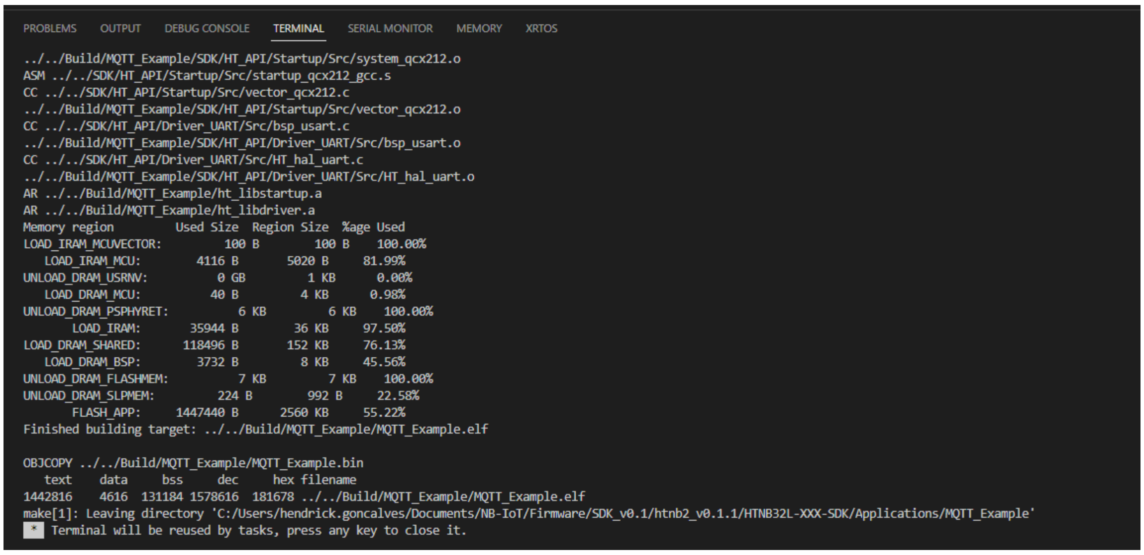


Figure 14: HTNB32L-XXX SDK task list.

3. Select the example that is supposed to be compile and press "Enter".
4. A terminal should be opened showing the compilation logs. Figure 15 shows the logs generated from MQTT_Example compilation:



```
../../Build/MQTT_Example/SDK/HT_API/Startup/Src/system_qcx212.o
ASM ../../SDK/HT_API/Startup/Src/startup_qcx212_gcc.s
CC ../../SDK/HT_API/Startup/Src/vector_qcx212.c
../../Build/MQTT_Example/SDK/HT_API/Startup/Src/vector_qcx212.o
CC ../../SDK/HT_API/Driver_UART/Src/bsp_usart.c
../../Build/MQTT_Example/SDK/HT_API/Driver_UART/Src/bsp_usart.o
CC ../../SDK/HT_API/Driver_UART/Src/HT_hal_uart.c
../../Build/MQTT_Example/SDK/HT_API/Driver_UART/Src/HT_hal_uart.o
AR ../../Build/MQTT_Example/ht_libstartup.a
AR ../../Build/MQTT_Example/ht_libdriver.a
Memory region         Used Size  Region Size  %age Used
LOAD_IRAM_MCUVECTOR:         100 B        100 B    100.00%
   LOAD_IRAM_MCU:        4116 B       5020 B     81.99%
UNLOAD_DRAM_USRNV:          0 GB         1 KB      0.00%
   LOAD_DRAM_MCU:          40 B         4 KB      0.98%
UNLOAD_DRAM_PSPHYRET:         6 KB         6 KB    100.00%
       LOAD_IRAM:       35944 B        36 KB     97.50%
LOAD_DRAM_SHARED:      118496 B       152 KB     76.13%
   LOAD_DRAM_BSP:        3732 B         8 KB     45.56%
UNLOAD_DRAM_FLASHMEM:         7 KB         7 KB    100.00%
UNLOAD_DRAM_SLPMEM:         224 B        992 B     22.58%
       FLASH_APP:     1447440 B      2560 KB     55.22%
Finished building target: ../../Build/MQTT_Example/MQTT_Example.elf

OBJCOPY ../../Build/MQTT_Example/MQTT_Example.bin
   text    data     bss     dec     hex filename
1442816    4616  131184 1578616  181678 ../../Build/MQTT_Example/MQTT_Example.elf
make[1]: Leaving directory 'C:/Users/hendrick.goncalves/Documents/NB-IoT/Firmware/SDK_v0.1/htnb2_v0.1.1/HTNB32L-XXX-SDK/Applications/MQTT_Example'
 *  Terminal will be reused by tasks, press any key to close it.
```

Figure 15: MQTT_Example compilation logs.

5. The binary file will be available on "Build" directory,

3.4.3. CLI Compilation

Above steps show how to compile the firmware using command line:

1. Open the HTNB32L-XXX-SDK directory in your VSCODE editor (Figure 13).
2. Click on "Terminal" and select "New Terminal" option to open a new PowerShell (or bash, depending on your VSCODE settings) terminal:

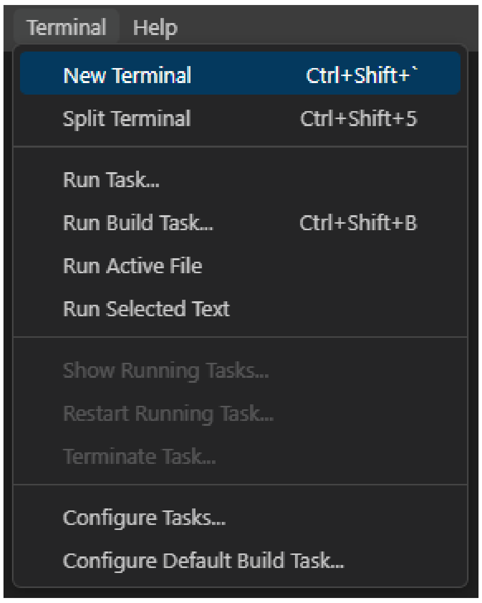


Figure 16: Opening a new PowerShell terminal.

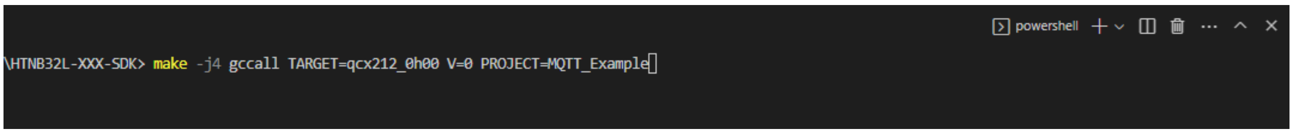
3. Type the command related to the Application Example that is supposed to be compiled. The command structure is shown below:

make -j4 gccall TARGET=qcx212_0h00 V=0 PROJECT=<APPLICATION_EXAMPLE>

Replace the "Application Example" field to the name of the respective application. Table 7 shows command examples to trigger a compilation process and Figure 17 shows how it should be used in PowerShell terminal:

Table 7: CLI Compilation examples.

Application	Command
MQTT_Example	make -j4 gccall TARGET=qcx212_0h00 V=0 PROJECT=MQTT_Example
Driver_Example	make -j4 gccall TARGET=qcx212_0h00 V=0 PROJECT=Driver_Example
Sleep_Example	make -j4 gccall TARGET=qcx212_0h00 V=0 PROJECT=Sleep_Example



```
\HTNB32L-XXX-SDK> make -j4 gccall TARGET=qcx212_0h00 V=0 PROJECT=MQTT_Example
```

Figure 17: MQTT_Example CLI compilation.

3.4.4. Build Clean

A "Build Clean" task was also included in the VSCODE workspace. To run it, just press "*CTRL+SHIFT+B*", to open the task list, and select "Build clean". It will erase the "Build" directory from the HTNB32L-XXX SDK root.

3.5. Flashing Procedure

The HTNB32L-XXX is uploaded through UART. It uses an internal bootloader to receive the new firmware through UART1 RX pin.

3.5.1. VSCODE Flashing Procedure

1. Considering the same circuit for GPIO1 (BOOT pin) and NRST, both drawn in Figure 7: Press and hold the GPIO1 button, then press the NRST button. Release the NRST button and release the GPIO1.

NOTE

GPIO1 must be at a low-logic level during the reset in order to trigger the flashing process.

2. Open the HTNB32L-XXX-SDK directory in your VSCODE editor:
3. Press "*CTRL+SHIFT+B*" to open the task list.
4. Select "Run - <Application Example>" to start uploading the new firmware.
5. If the COM port was not previously configured, a window called "COM Selection" will appear. User should just select the right com port and press "Ok" or "Enter".

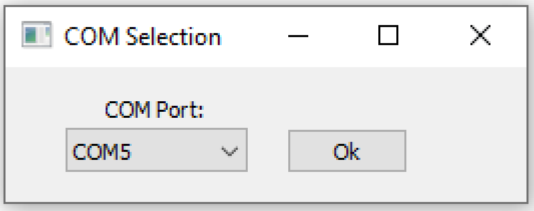


Figure 18: COM port selection.

NOTE

Users can change the COM port manually by executing the "COM select" task available on the task list.

3.5.2. HTTools Flashing Procedure

HTTools is a software developed to furnish a complete and user-friendly flash tool for HTNB32L-XXX customers. It transmits a binary file to the HTNB32L-XXX device through serial interface, based on UART protocol. To download learn how to use HTTools, please check HTNB32L-XXX-UM0003-HTTools document.

3.6. Debug Procedure

The HTNB32L-XXX SDK has a complete debug environment based on SWD protocol. Users can debug their code step by step, by connecting a J-Link device to the HTNB32L-XXX SWD pins. To take full advantage of this environment, it is mandatory to be in possession of a SEGGER J-Link or a ST-LINK device. Please refer to Subsection 2.10 to build the debugging setup.

3.6.1. VSCODE Debugger

Follow below steps to start the VSCODE debugger:

1. Open the HTNB32L-XXX-SDK directory in your VSCODE editor:
2. Click on "Run and Debug" button or press "*CTRL+SHIFT+D*" to open the debug view.
3. Select the application and click on "play" (or just F5) to start the debugger. The scripts programmed to execute the debugging task will first compile and flash the firmware.
4. Figure 20 shows the VSCODE debugger running.

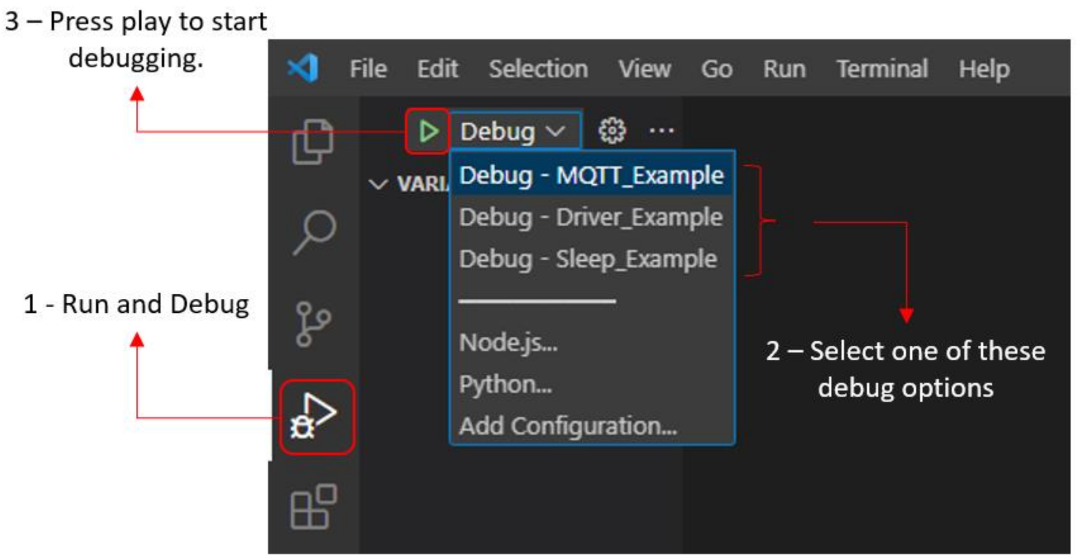
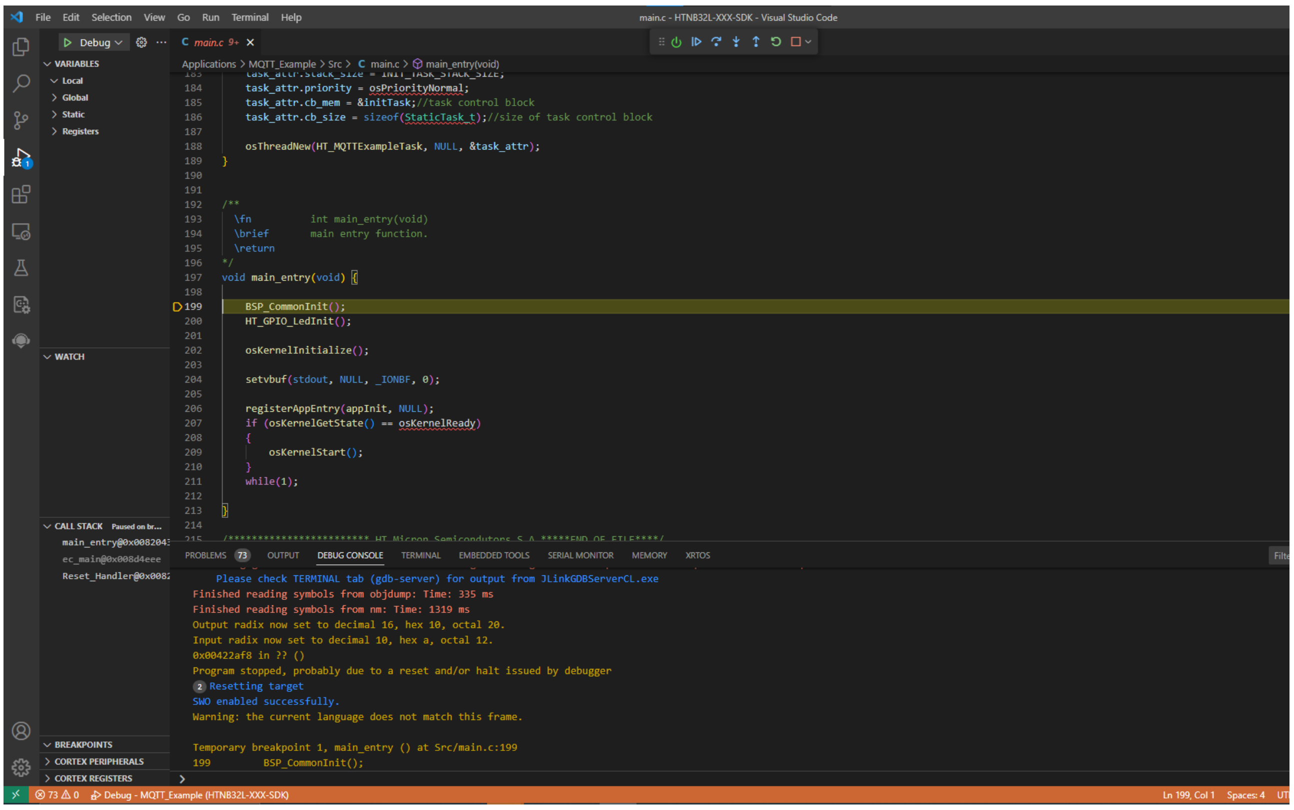


Figure 19: Debugger steps.


File Edit Selection View Go Run Terminal Help
main.c - HTNB32L-XXX-SDK - Visual Studio Code
Debug
main.c 9+
Applications > MQTT_Example > Src > main.c > main_entry(void)
VARIABLES
Local
Global
Static
Registers
WATCH
CALL STACK Paused on br...
BREAKPOINTS
CORTEX PERIPHERALS
CORTEX REGISTERS
PROBLEMS 73
OUTPUT
DEBUG CONSOLE
TERMINAL
EMBEDDED TOOLS
SERIAL MONITOR
MEMORY
XRTOS
73 0
Debug - MQTT_Example (HTNB32L-XXX-SDK)
Ln 199, Col 1
Spaces: 4

```
main_entry@0x008204
ec_main@0x008d4eee
Reset_Handler@0x0082
```

```c
    task_attr.stack_size = INIT_TASK_STACK_SIZE;
    task_attr.priority = osPriorityNormal;
    task_attr.cb_mem = &initTask;//task control block
    task_attr.cb_size = sizeof(StaticTask_t);//size of task control block

    osThreadNew(HT_MQTTExampleTask, NULL, &task_attr);
}


/**
  \fn          int main_entry(void)
  \brief       main entry function.
  \return
*/
void main_entry(void) {

    BSP_CommonInit();
    HT_GPIO_LedInit();

    osKernelInitialize();

    setvbuf(stdout, NULL, _IONBF, 0);

    registerAppEntry(appInit, NULL);
    if (osKernelGetState() == osKernelReady)
    {
        osKernelStart();
    }
    while(1);

}

/************************ HT Micron Semicondutors S.A *****END OF FILE****/
```

```
    Please check TERMINAL tab (gdb-server) for output from JLinkGDBServerCL.exe
Finished reading symbols from objdump: Time: 335 ms
Finished reading symbols from nm: Time: 1319 ms
Output radix now set to decimal 16, hex 10, octal 20.
Input radix now set to decimal 10, hex a, octal 12.
0x00422af8 in ?? ()
Program stopped, probably due to a reset and/or halt issued by debugger
2 Resetting target
SWO enabled successfully.
Warning: the current language does not match this frame.

Temporary breakpoint 1, main_entry () at Src/main.c:199
199         BSP_CommonInit();
```

Figure 20: VSCODE debugger running.

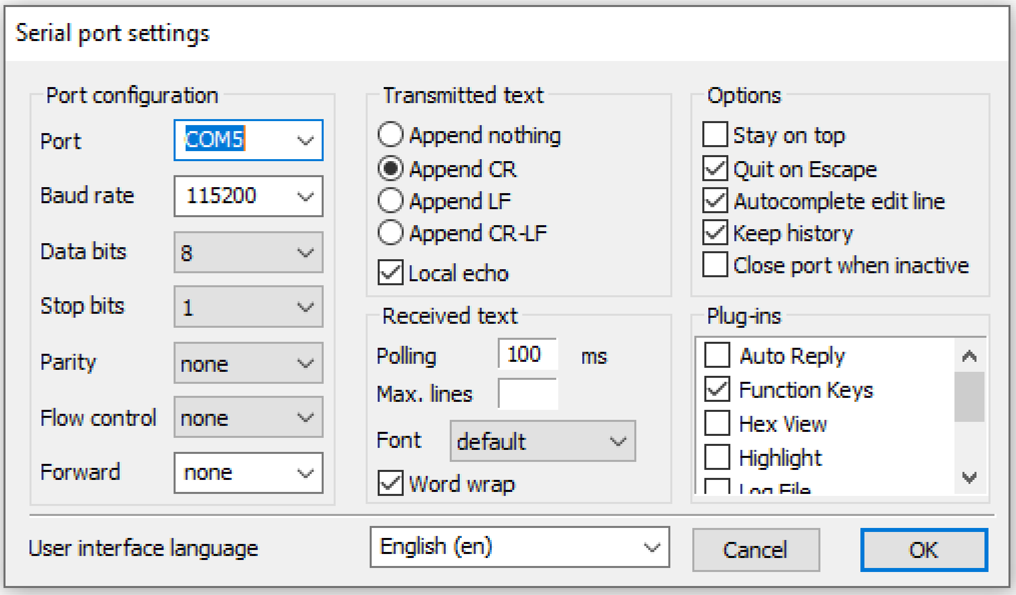
4. AT COMMANDS

A firmware of AT Commands is also available for applications that do not require embedding a custom code on HTNB32L-XXX. The binary file related to this firmware can be found at the "Firmware" directory, at the root of the HTNB32L-XXX SDK. This section describes how to use the AT Commands to connect a HTNB32L-XXX device to a NB-IoT network. Please refer to **HTNB32L-XXX-UM0001-AT_Commands** document to check the complete list of AT commands available.

4.1. Serial Terminal Settings

The AT commands can be tested using a serial terminal (RS232 terminal). Although users can choose any terminal of their preference, Figure 21 shows the Termite settings that must be followed.

- Baud rate: 115200.
- Data bits: 8.
- Stop bits: 1.
- Parity: none.
- Flow control: none.
- **Transmitted text: APPEND CR.**


Serial port settings

Port configuration
Port: COM5
Baud rate: 115200
Data bits: 8
Stop bits: 1
Parity: none
Flow control: none
Forward: none

Transmitted text
Append nothing [✓ Append CR] Append LF Append CR-LF
[✓] Local echo

Received text
Polling: 100 ms
Max. lines:
Font: default
[✓] Word wrap

Options
[ ] Stay on top
[✓] Quit on Escape
[✓] Autocomplete edit line
[✓] Keep history
[ ] Close port when inactive

Plug-ins
[ ] Auto Reply
[✓] Function Keys
[ ] Hex View
[ ] Highlight
[ ] Log File

User interface language: English (en)

Figure 21: Termite settings.

NOTE

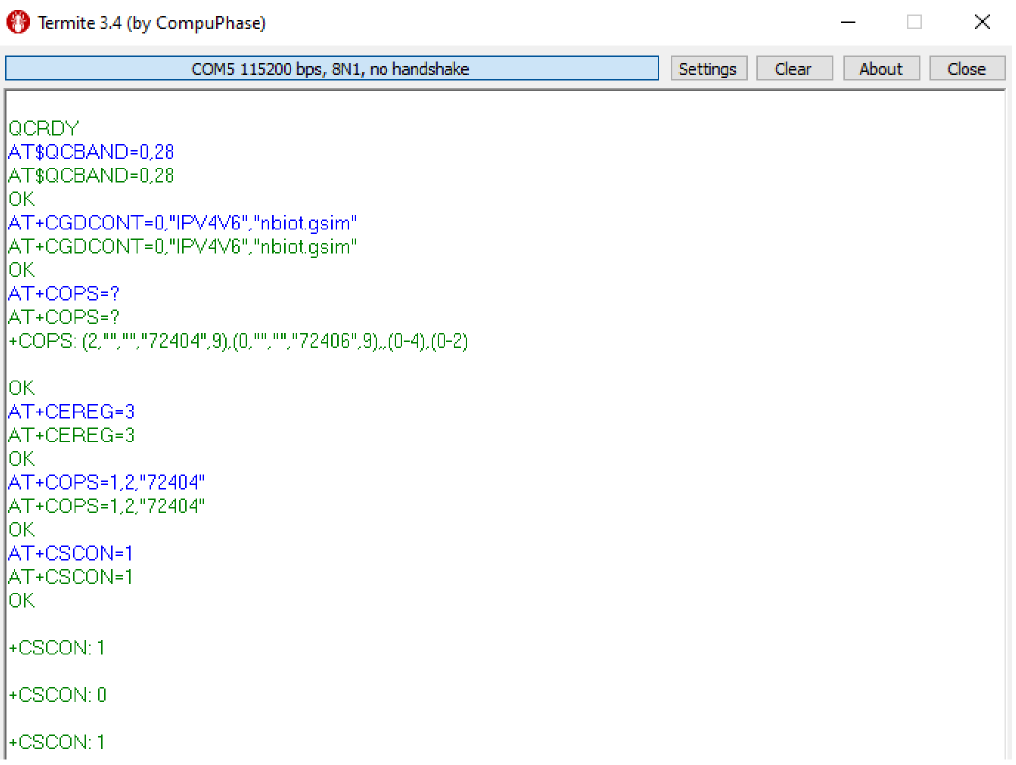
It is important to set the "Transmitted text" field correctly before transmitting any command to the device.

4.2. Network Connection

The following commands should be used in order to establish a connection between HTNB32L-XXX and a NB-IoT network. It is important to note that the parameters used in these commands will vary according to the network provider. All commands described in Table 8 have its respective parameters related to the Brazilian NB-IoT provider TIM.

Table 8: Network connection commands.

Step	Command	Description
1	AT$QCBAND=0,28	Set the network band of the respective provider.
2	AT+CGDCONT=0,"IPV4V6","nbiot.gsim"	Set the PDP type and the network APN for NB-IoT. These information must be shared by your network provider.
3	AT+COPS=?	Search for network cells. It returns a set of five parameters, each representing an operator present in the network. These parameters must be used in the Step 5.
4	AT+CEREG=3	Configures the presentation of unsolicited result codes for EPS Network Registration Status.
5	AT+COPS=1,2,"72404"	Forces an attempt to select and register the EPS network operator using the USIM card installed in the currently selected card slot. The parameters selected here must be related to the returned values received on Step 3.
6	AT+CSCON=1	Establish a connection to the NB-IoT network.


Termite 3.4 (by CompuPhase)
COM5 115200 bps, 8N1, no handshake
Settings
Clear
About
Close

```
QCRDY
AT$QCBAND=0,28
AT$QCBAND=0,28
OK
AT+CGDCONT=0,"IPV4V6","nbiot.gsim"
AT+CGDCONT=0,"IPV4V6","nbiot.gsim"
OK
AT+COPS=?
AT+COPS=?
+COPS: (2,"","","72404",9),(0,"","","72406",9),,(0-4),(0-2)

OK
AT+CEREG=3
AT+CEREG=3
OK
AT+COPS=1,2,"72404"
AT+COPS=1,2,"72404"
OK
AT+CSCON=1
AT+CSCON=1
OK

+CSCON: 1

+CSCON: 0

+CSCON: 1
```

Figure 22: Command examples.

NOTE

After knowing what must be returned from +COPS, there is no need to run AT+COPS=? again.

5. RUNNING A FIRST EXAMPLE

The HTNB32L-XXX SDK provide some basic firmware examples that can be used as start point of new OpenCPU projects. This section describes the first steps with the MQTT Example.

5.1. MQTT EXAMPLE

The MQTT Example is a basic application that provides a complete usage example for solutions based on the MQTT protocol. It uses a digital twin, developed in Python, to publish and subscribe commands through a public MQTT broker.

MQTT Example was developed based on the MQTT Demo Board. Please check schematic and gerber files on iMCP HTNB32L-XXX GitHub repository.

5.1.1. MQTT Details

- **Broker address:** broker.hivemq.com (Please check HiveMQ website for more details)
- **Password:** HTmicron
- **TCP port:** 1883

5.1.2. How to Use

1. Run the HTNB32L-XXX-MQTT-Demo-SW software, available on the Software_Apps directory, by double clicking the "*backend.py*" file.

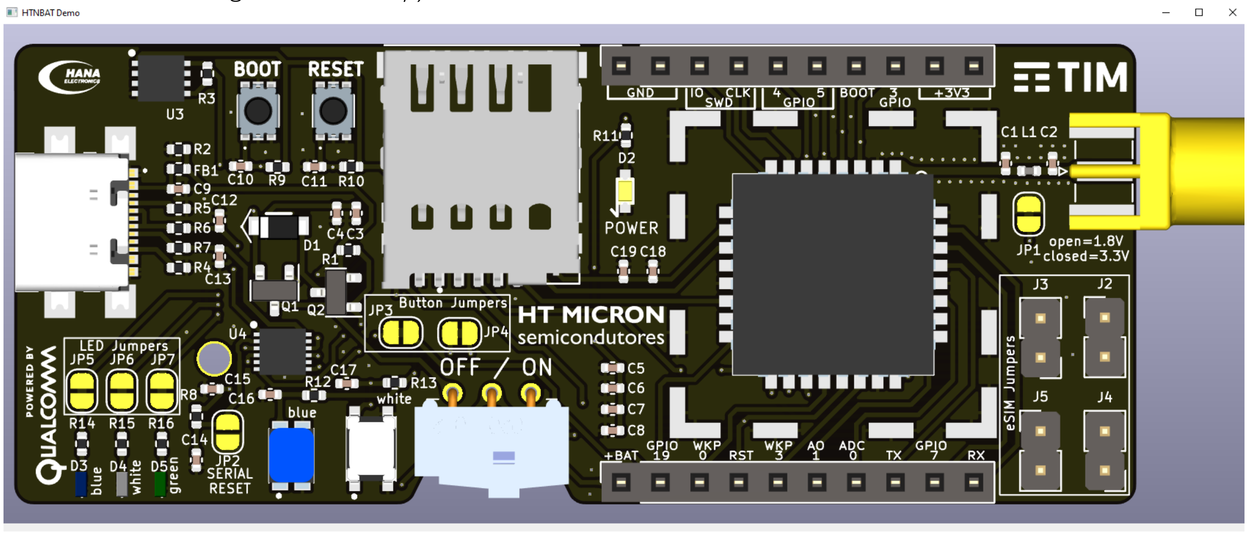


Figure 23: HTNB32L-XXX-MQTT-Demo-SW software.

NOTE

This software is implemented in Python. Any package issue can be easily fixed by installing its dependencies.

2. Wait until the green LED of the digital twin starts blinking. The green LED is used to inform the connection status between the MQTT Demo software and the MQTT broker. It will blink if the Python application was successfully connected to the host.

3. Compile and flash the MQTT_Example firmware on your HTNB32L-XXX device. Please refer to Sections 3.4 and 3.5 for more details about the compiling and flashing procedures.
4. Connect your HTNB32L-XXX device to the local NB-IoT network by following the steps detailed at Subsection 4.2.
5. Wait until the green LED embedded on the physical MQTT Demo board starts blinking. Similar as its digital-twin, this LED is responsible to signalize if the connection was successfully established between the HTNB32L-XXX, the NB-IoT network, the MQTT broker and also the MQTT Demo software.
6. The idea of the digital twin is to replicate everything that happens in the MQTT Demo board. For example:
   a. If the blue button of the MQTT Demo board is pressed, the blue LED of both PCBs will turn on or turn off, depending on its previous state.
   b. If the white button of the digital twin board is pressed, the white LED of both PCBs will turn on or turn off, depending on its previous state.

ABBREVIATIONS

Table 9: Abbreviations

Acronym	Description
GPIO	General Purpose Input Output
PCB	Printed-Circuit Board
SDK	Software Development Kit
VSCODE	Visual Studio Code
SWD	Serial Wire Debug
CPU	Central Processing Unit
MQTT	Message Queuing Telemetry Transport
LIB	Library
COM	Communication
TCP	Transmission Control Protocol
APP	Application
DEMO	Demonstration
UART	Universal Asynchronous Receiver-Transmitter
ROM	Read-only Memory
CMD	(Command) Windows Command Prompt

LIST OF FIGURES

LIST OF TABLES

REVISION HISTORY

Version	Date	Changes	Authors
00	12/04/2023	- Initial draft	HBG

CONTACT

HT MICRON SEMICONDUTORES S.A.
Av. Unisinos, 1550 | 93022-750 | São Leopoldo | RS | Brasil
www.htmicron.com.br

DOCUMENT INFORMATION

Document Title:	iMCP HTNB32L-XXX Getting Started
Document Subtitle:	Getting Started for iMCP HTNB32L-XXX System-in-Package
Classification:	CONFIDENTIAL
Doc. Type:	USER MANUAL
Revision:	v.01
Date:	04/04/2023
Code:	HTNB32L-XXX-UM0001

DISCLAIMER

This document is a property of HT Micron and cannot be reproduced, disseminated or edited without its consent.
HT Micron does not assume any responsibility for use what is described.
This document is subject to change without notice.