# GV850 Software Development Guide

Version: 1.02

| | |
|---|---|
| **Document Title** | GV850 Software Development Guide |
| **Version** | 1.02 |
| **Date** | 2024-08-28 |
| **Status** | Released |

## General Notes

Queclink offers this information as a service to its customers, to support application and engineering efforts that use the products designed by Queclink. The information provided is based upon requirements specifically provided to Queclink by the customers. Queclink has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by Queclink within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

## Contents

# 0. Revision History

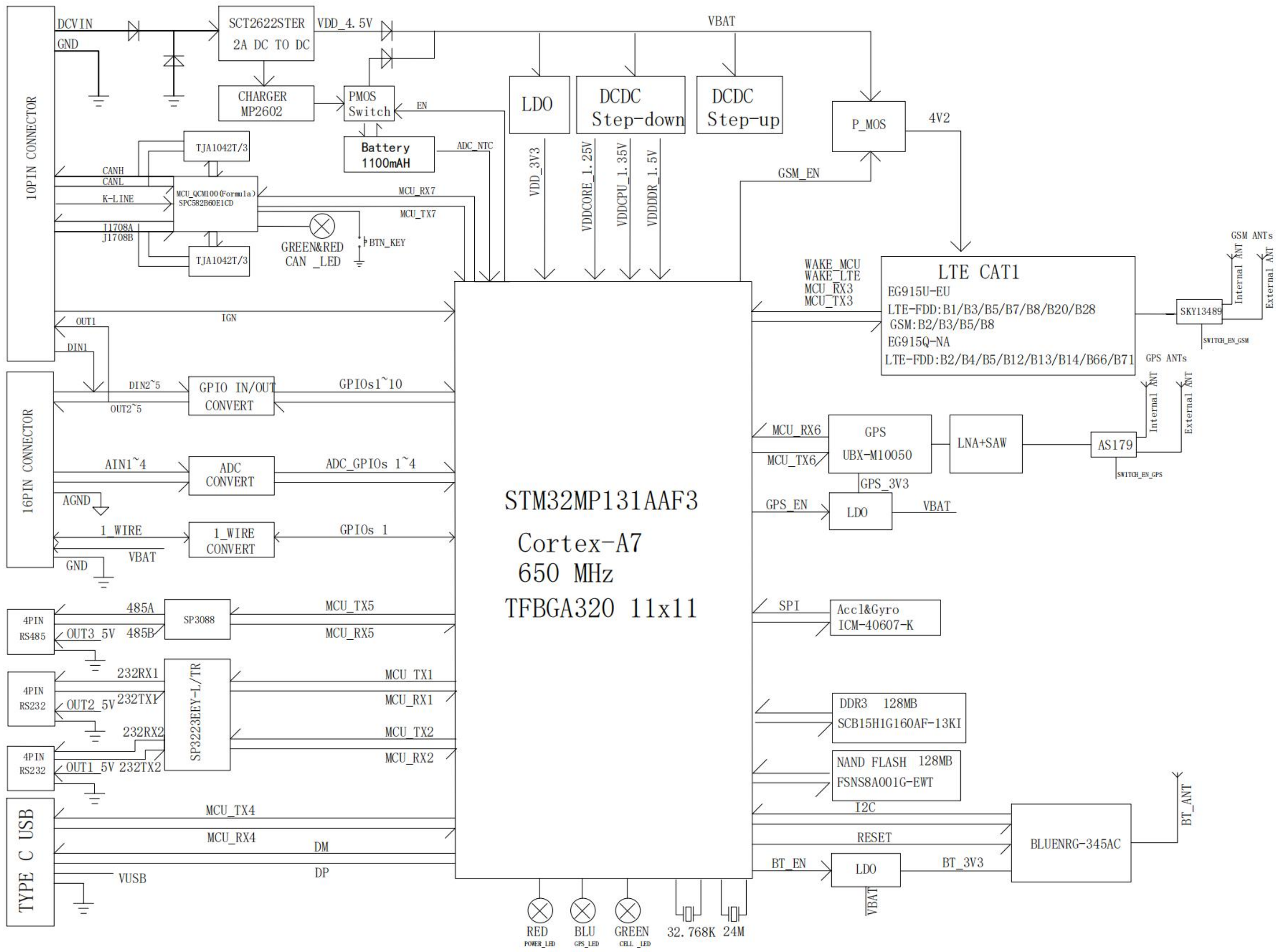
| Version | Date | Author | Description of Change |
|---|---|---|---|
| 1.00 | 2023-09-25 | Alex Liao | Initial |
| 1.01 | 2023-11-21 | Alex Liao | Added more information to make the file more complete. |
| 1.02 | 2024-08-22 | Mundo ,<br>Alex Liao | 1. In accordance with the modifications in hardware version V1.04, update the device's IO resource content.<br><br>A new DDR power control pin with a latching function has been added.<br><br>A new internal power supply enable latching function has been introduced, with the data pin remaining the same.<br><br>The LTE module has been updated with a new USB network card function. Details can be found in the "LTE" section.<br><br>2. Added more description on the application development and debugging process<br><br>3. Added an explanation of the GV851 CAN FD module to this document. Refer to the "Raw CAN FD" section for details.<br><br>4. A new section "Queclink Software Modules" has been added to the document, along with introductions to the "CANOBD," "Batterymgr," "selftask," and "Testcase" software modules.<br><br>5. Optimized the module circuit diagrams in the "Interface and Driver" chapter, removing connections and information that are not related to the corresponding modules. |

# 1. Overview

| | |
|---|---|
| CPU | STM32MP133A |
| RAM | 128MB, DDR3 or above |
| FLASH Memory | 128MB SPI or above |
| Status LEDs | 1 x Power, 1 x CEL, 1 x GNSS LED, 1 x CAN/Tachograph |
| Modem | Support Cat 1<br>LTE-FDD: B1/B3/B5/B7/B8/B20/B28<br>GSM: B2/B3/B5/B8 |
| SIM | 1 x SIM card slot or eSIM |
| Cellular Antenna | Internal or external |
| GNSS | u-box all-in-one GNSS receiver, support GPS, Glonass, Galileo, Beidou |
| GNSS Antenna | Internal or external |
| CAN | CAN1H/CAN1L, support reading CAN bus data in heavy (J1939/FMS) and light vehicle<br>CAN2H/CAN2L, support reading and download tachograph data, support reading CAN data in J1708 and OBDII |
| K-Line | Connect D8 of tachograph for live data reading |
| RS232 or RS485 | 2 x RS232, 300-115200 baud rate<br>1 x RS485, 300-115200 baud rate/Half Duplex (2 wires) |
| I/O | 1 x positive trigger input for ignition detection<br>5 x negative trigger inputs<br>4 x analog input (0-32V)<br>5 x digital output, open drain, 150mA max drive current<br>3 x 5V outputs for external accessories<br>1 x DC 3.3V output for temperature sensor |
| 1-Wire Interface | Support 1-wire temperature sensor and iButton driver ID |
| G-sensor | 6-axis motion, motion detection, harsh driving detection, shock detection |
| BLE | BLE5.2 |
| BLE Antenna | Internal |
| Battery | Li-Polymer, 1100mAh |
| Reset button | Reset button to reset CAN |
| Type-C USB | Used for configuration, upgrade and debug |
| Sleep Current | < 10mA |

| Firmware/configuration | |
|---|---|
| Operating system | Linux OS, Kernel 5.15.67 |
| **Power** | |
| Connector | Pin connector |
| Input voltage range | 8 – 32 VDC, reverse polarity protection; surge protection >31 VDC 10us max |
| Power consumption | 5W (average) |
| **Physical Specification** | |
| Dimensions | 123*80*21mm (L*W*H) |
| Weight | 150g |
| Mounting options | Flat surface placement |
| **Operating Environment** | |
| Operating temperature | -20 °C to 60 °C |
| Operating humidity | 10% to 90% RH non-condensing |
| Ingress Protection Rating | IP30 |

The hardware block diagram is as following:




The software block diagram is as follows:


Two construction methods, buildroot and yocto, and corresponding SDK source codes, are provided. The positioning from Linux of these two construction methods differs (though both are commonly used in embedded systems, but there are differences in efficiency and usage methods):

- Buildroot, which builds a more streamlined and simple system and is suitable for devices with limited hardware resources (mainly flash);
- Yocto, which builds a system with rich features and supports more complete hardware, including UI, audio and video software stacks, requiring a larger flash size.

# 2. Platform Development

At present, source codes for building systems based on buildroot are provided, which can build and package complete system images.

## 2.1. Device Tree

The first step in developing STM32MP1 platform devices is to adapt a device tree based on its hardware. Moreover, because the device tree is used in each module of bootchain, it is a complex and cumbersome operation to ensure that each module obtains the correct device tree during compilation. Therefore, ST has provided the STM32CubeMX tool to provide visualization, assistance, and configuration wizards that can automatically generate the device tree required by each module.
The provided buildroot source codes already contain the adapted devicetree.

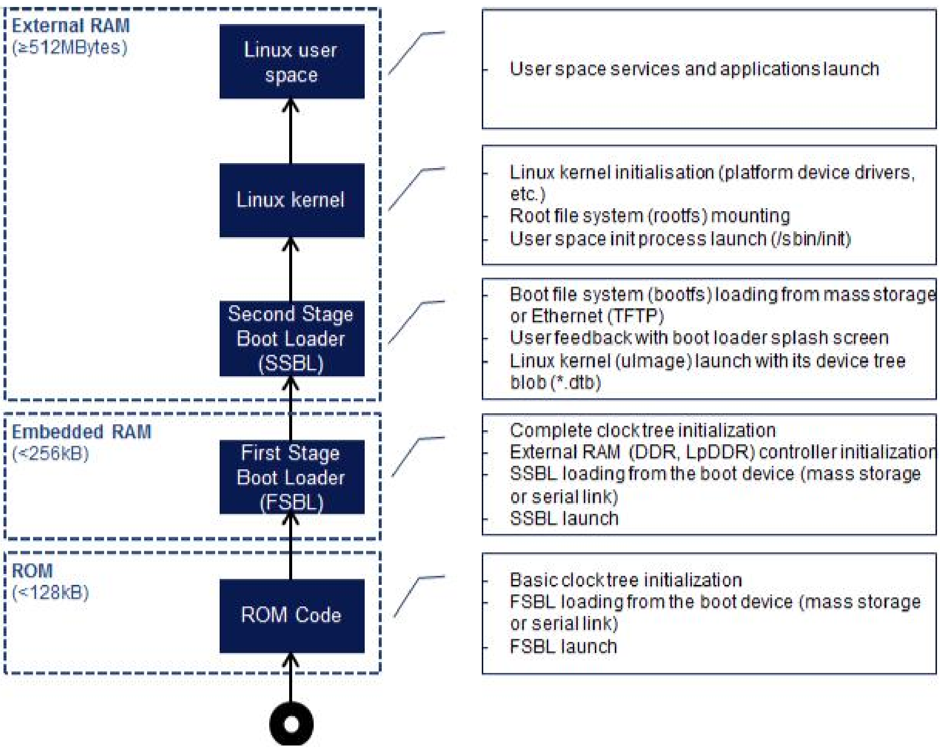
## 2.2. Boot Chain

The STM32MP133 platform is based on the ARM Cortex-A7 architecture, and the boot process is similar to other ARM architectures. It is mainly divided into the following stages:

ROM code
FSBL (First stage bootloader) TF-A
SSBL (Second-stage bootloader), u-boot

Please search “boot chain” in following link to get more information.

- [https://wiki.stmicroelectronics.cn/stm32CPU/wiki/Main_Page](https://wiki.stmicroelectronics.cn/stm32CPU/wiki/Main_Page)



After understanding the startup process of STM32 ARM, it's known that the actual module relationship is:

TF-A->OP-TEE->U-BOOT->Linux Kernel

After initially adjusting the device tree according to the actual hardware configuration, copy the device tree to the source code directory of the above-mentioned modules for compilation, flashing, and running. If any errors occur, refer to the error messages and resolve them by consulting the "STM32MP135x_bringup" documentation.

## 2.3. Compilation Method

To use the Buildroot, there must be a Linux distribution installed on the workstation. Any reasonably recent Linux distribution (Ubuntu, Debian, Fedora, Redhat, OpenSuse, etc.) will work fine. Then, a small set of packages needs to be installed as described in the System Requirements section of Buildroot Manual.

For Debian/Ubuntu distributions, use the following command to install the necessary packages:

```
$ sudo apt-get install -y debianutils sed make binutils build-essential gcc g++ \
                          bash patch gzip bzip2 perl tar cpio unzip rsync file bc git \
                          wget python3 libssl-dev libncurses-dev
```

After finishing installation, extract source tarball that is provided:

```
$ tar zxvf GV850_buildroot_dd981da1.tar.gz
```

Go to the Buildroot directory:

```
$ cd GV850_buildroot_dd981da1/buildroot/
```

And then, configure the system you want to build by using the defconfigs provided in this BR2_EXTERNAL tree.

```
$ make BR2_EXTERNAL=../buildroot-external-st st_stm32mp133a_queclink_GV850CEU_defconfig
```

For model GV851:

```
$ make BR2_EXTERNAL=../buildroot-external-st st_stm32mp133a_queclink_GV851CEU_defconfig
```

There are two pieces of information are provided:

1. The path to BR2_EXTERNAL tree, which is provided side-by-side to the Buildroot repository.
2. The name of the Buildroot configuration.

If there is the need to further customize the Buildroot configuration, please run 'make menuconfig', but for the first build, it is recommended to keep the configuration unchanged so that it can be verified that everything is working.

Start the build:

```
$ make V=s
```

It might take between 30 and 60 minutes depending on the configuration that is chosen and how powerful the machine is. All software packages for building the entire Linux system for the STM32MP1 platform (e.g. cross-compilation toolchain, firmware, bootloader, Linux kernel, root filesystem) are already included, no downloading is needed unless default configuration is customized.

Buildroot might need to be authorized to root (or sudo) in order to compile some packages (related to Python 3) properly. If some permission failures are met, please retry:

```
$ sudo make V=s
```

When the building is done, it will output images in the directory below, including u-boot, kernel, rootfs binary files.

```
$ cd output/images
```

Following files in this directory are necessary for flashing, please copy and prepare for flashing.

```
├── fip.bin
├── flash.tsv
├── metadata.bin
├── rootfs.ubi
└── tf-a-stm32mp133a-gv850ceu-mx.stm32
```

## 2.4. Programming

The device supports both USB OTG programming and OTA firmware updating.

**USB OTG**

After successfully building with Buildroot, the complete files required for programming can be obtained.

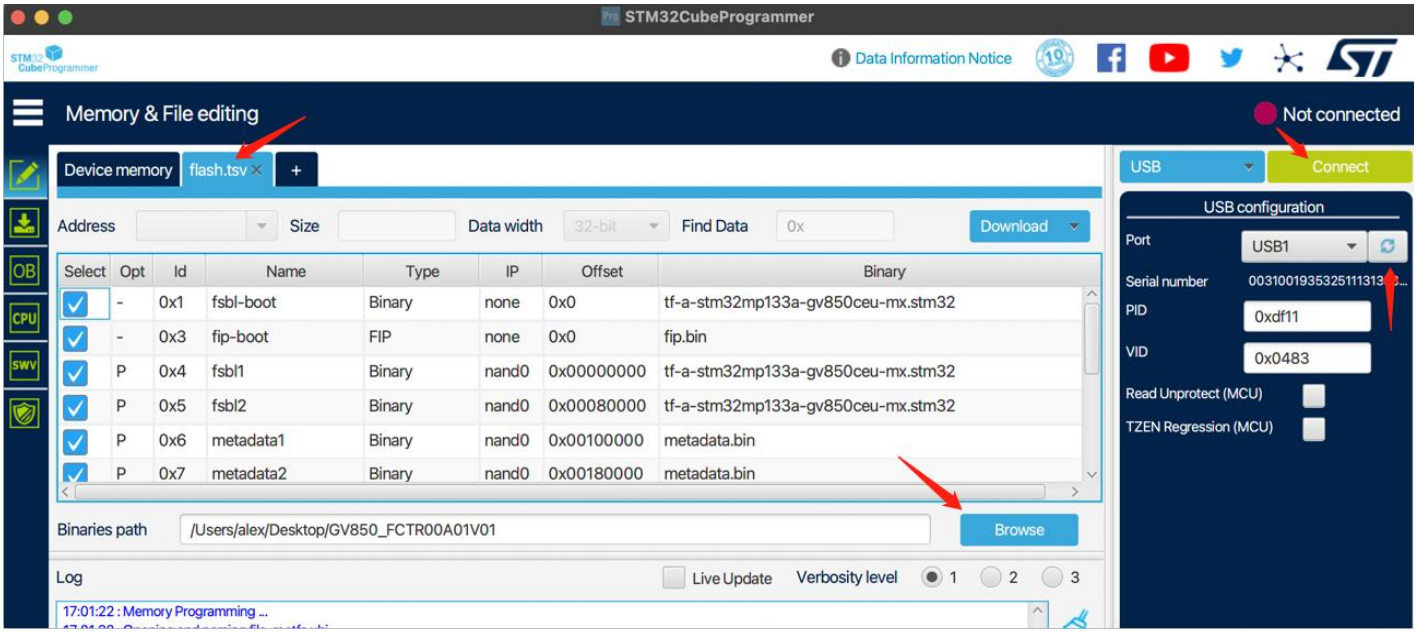
```
├── fip.bin                                // FIP
├── flash.tsv                              // Program partitions configuration table
├── metadata.bin
├── rootfs.ubi                             // Including kernel and file system rootfs
└── tf-a-stm32mp133a-gv850ceu-mx.stm32     // TF-A
```

The device first enters DFU mode. And then use the STM32CubeProgrammer tool to erase and programm the device. The method and steps are as follows:
1. Use the USB+UART 2-in-1 cable provided along with the device, open the COM device on a PC using the UART tool, and the baud rate is 115200bps;
2. Power on the device, the COM starts printing the startup log, and then quickly press any key on the keyboard. The startup process will be interrupted and it requires to enter the u-Boot command. Then, enter the following command to enter DFU mode;

```
STM32MP> stm32prog usb 0
```

```
Net:   No ethernet found.
Hit any key to stop autoboot:  0
STM32MP>
STM32MP>
STM32MP>
STM32MP>
STM32MP>
STM32MP> stm32prog usb 0
DFU alt info setting: done
#
UPLOAD ... done
Ctrl+C to exit ...
#
FlashLayout received, size = 653
DOWNLOAD ... OK
Ctrl+C to exit ...
```

3. Connect the USB of the cable to the PC, click the right button of the mouse to click refresh, after automatically scanning and finding the device that has entered DFU mode, and then click "Connect"
4. Select "Open file" to load the flash. tsv file from the released firmware, and note that select the correct path for "Browse";
3. Click "Download" to start programming. After successful programming, power off the device, unplug and reinsert the USB Type-C cable, power on the device, and the device enters the boot process.




**OTA**

Still under development.

# 3. Application Development

## 3.1. Programming Languages

The current GV850 and GV851 device support the development languages C/C++/Python. The provided Buildroot SDK source package includes a cross-toolchain, which can compile C/C++ source code and link it into executable files that can run on the ARM platform.

The device currently only supports Python 3, and comes pre-installed with the pip tool. After connecting to the Internet, you are free to download and install various Python modules. This greatly enhances the efficiency of embedded software development.

## 3.2. Queclink Software Package Compilation

In order to facilitate developers to familiarize themselves with the platform, example code and software packages of testing programs are provided. Please Compile it using the following command:

```
$ make queclink-dirclean
$ make queclink
```

In the output/build/queclink-X.X/modules/ directory, you will find the target files generated from the compilation. These target files are also copied to the corresponding output/target/ directory. During the firmware compilation process, they are collectively packaged into the root file system. Taking the example_ble tool as an example, the executable file path is as follows:

```
./output/build/queclink-1.0/tools/example_ble
./output/target/usr/sbin/example_ble
```

## 3.3. Debugging Methods

Once an executable file is compiled, it needs to be copied to the device for running and debugging. This process is both frequent and crucial. We offer very convenient debugging methods, namely UART Console and USB Ethernet/RNDIS Net SSH.

The product package includes a 2-in-1 cable, with one end being a USB Type-C connector that connects to the USB port of the GV850 device; the other end consists of two USB Type-A connectors. One of the USB Type-A connectors, marked with "DATA_CABLE_M," has a built-in USB-to-Serial chip, and the other USB Type-A connector serves as a general-purpose USB port. In the subsequent content, USB Type-A #1 is the USB-to-Serial USB connector, while USB Type-A #2 is the USB Ethernet/RNDIS USB connector.


### 3.3.1. UART Console Debugging

UART Console debugging is essential during the development of embedded devices. It is the only way to check the device status when the system experiences severe failures. The UART Console port is often used for configuring the system, viewing logs, entering commands, and transferring files.

The GV850 device also offers this debugging method. Initially, install the appropriate USB-to-Serial port driver on the development coCPUter. Subsequently, connect the Type-C end of the USB cable that comes with the GV850 device to the GV850, and connect the USB Type-A #1 connector to the development coCPUter. Once the connection is established and the device is powered on, if the drivers are correctly installed and the connections are secure, a COM port will be visible in the Windows Device Manager of the development coCPUter, or a /dev/ttyUSBx device node will be present in the Linux environment.

Use a command terminal tool, such as putty, to establish a connection with the UART Console port, setting the parameters to a baud rate of 115200, with 1 start bit, 8 bits of data, 1 stop bit, and no parity bit. Upon successful connection, authenticate using the username "root" and the password "root".

The USB-to-Serial driver file for Windows is named "CH341SerSetup.zip".

### 3.3.2. USB Port Debugging

The UART Console port allows for one connection, and the speed of file transfer is comparatively slow. The GV850 device is equipped with the USB Ethernet/RNDIS virtual network card capability, enabling SSH connections to the device for debugging purposes.

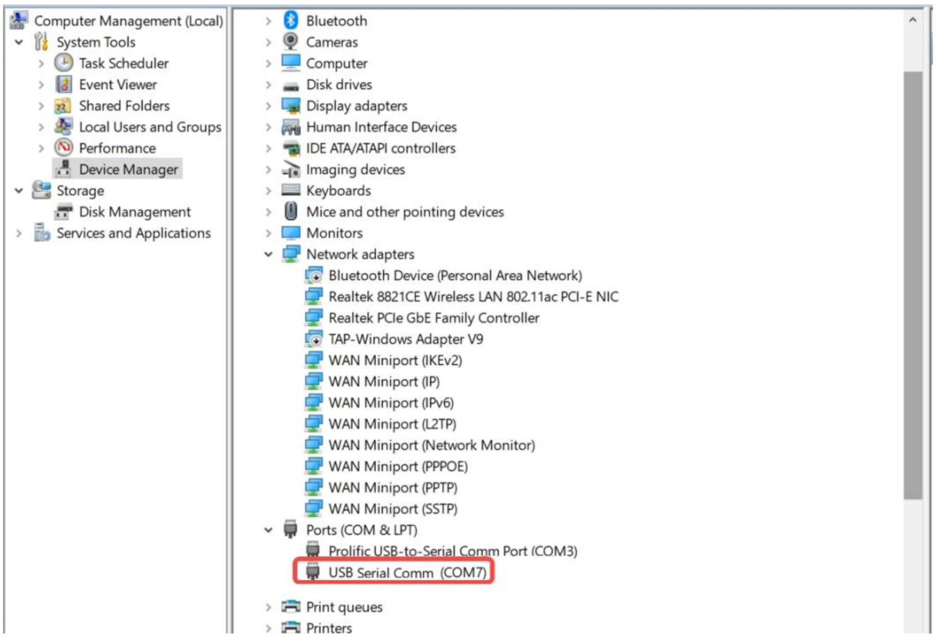
Initially, install the USB Ethernet/RNDIS virtual network card driver on the coCPUter to enable it to recognize the GV850 device's USB port as a virtual network card. Next, connect the Type-C end of the USB cable that comes with the GV850 device to the device itself, and connect the USB Type-A #2 connector to the development coCPUter. Provided that the driver installation is successful and the connection is secure, a USB Ethernet adapter will be visible in the Windows Device Manager, and a usbx network node will be present in the Linux environment.

The steps for installing the USB Ethernet/RNDIS virtual network card driver on the development coCPUter are as follows:

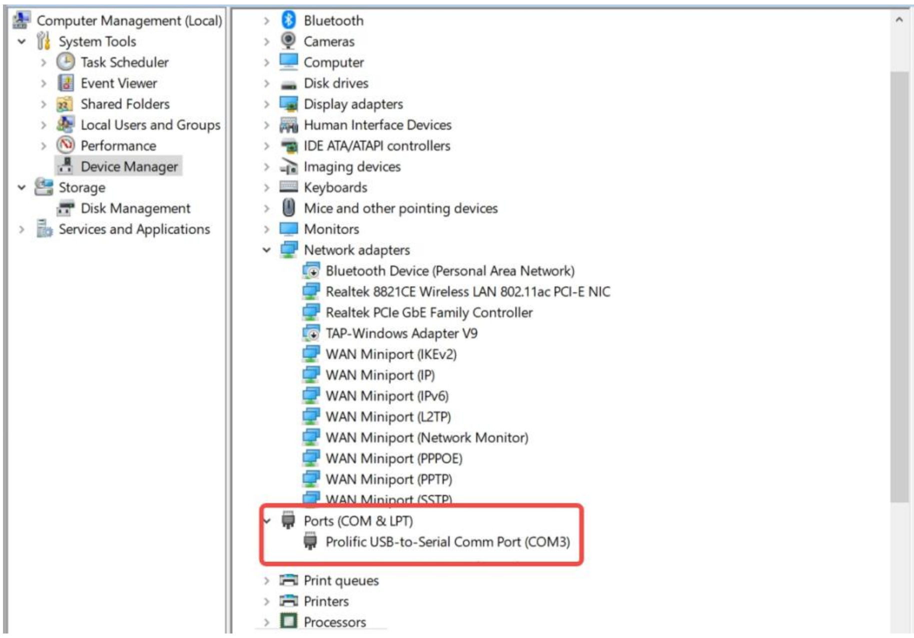
1 Step: Power up the device and connect the Type-C end of the USB cable. Initially, do not attach the USB Type-A #1 and USB Type-A #2 connectors to the coCPUter.

2 Step: Open the Device Manager on your coCPUter, and then connect the USB Type-A #2 connector to the coCPUter. Look for the new device that has been added in the Device Manager. The USB Type-A #2 connector may be recognized as a COM port, it might be identified as a different driver, or it could appear as an unrecognized device. If you are unable to recognize which device is the newly connected one, move on to Step 3.

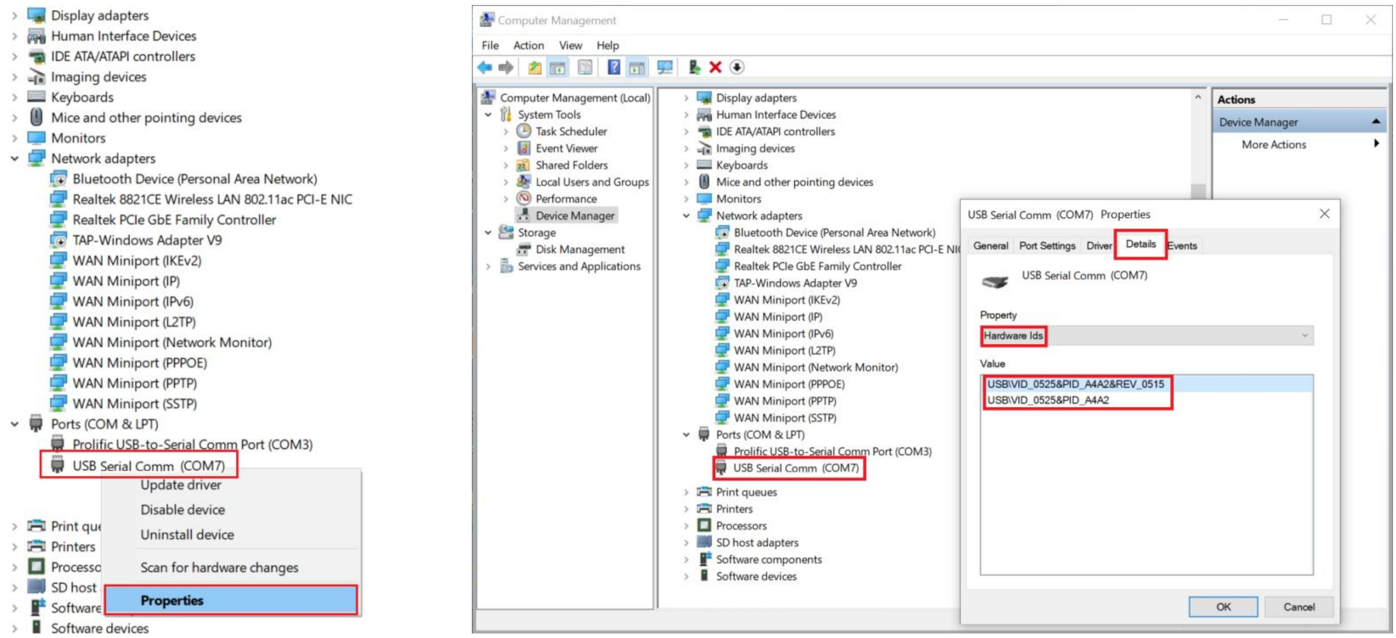
Upon connecting the USB Type-A #2 connector, a new COM7 interface has been added in the coCPUter, as depicted in the figure below.



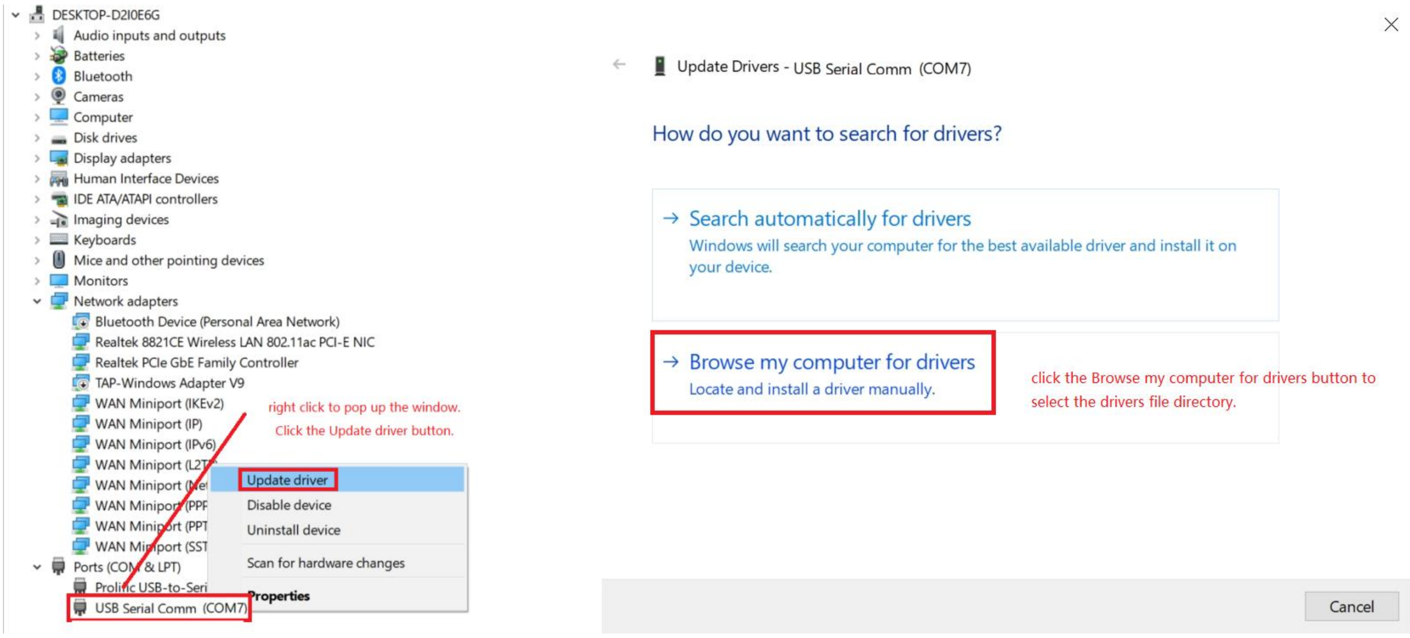

3 Step: Remove the USB Type-A #2 connector from the coCPUter and examine the Device Manager. Find out which device has been removed. If you are unable to identify which device has been removed, perform Step 2 again. If there is no new or missing device in the Device Manager during Steps 2 and 3, it is necessary to verify whether the coCPUter's USB ports are working correctly and to test with another USB port on the coCPUter. After you have determined the device node created by the USB Type-A #2 connector on the coCPUter, move on to Step 4.

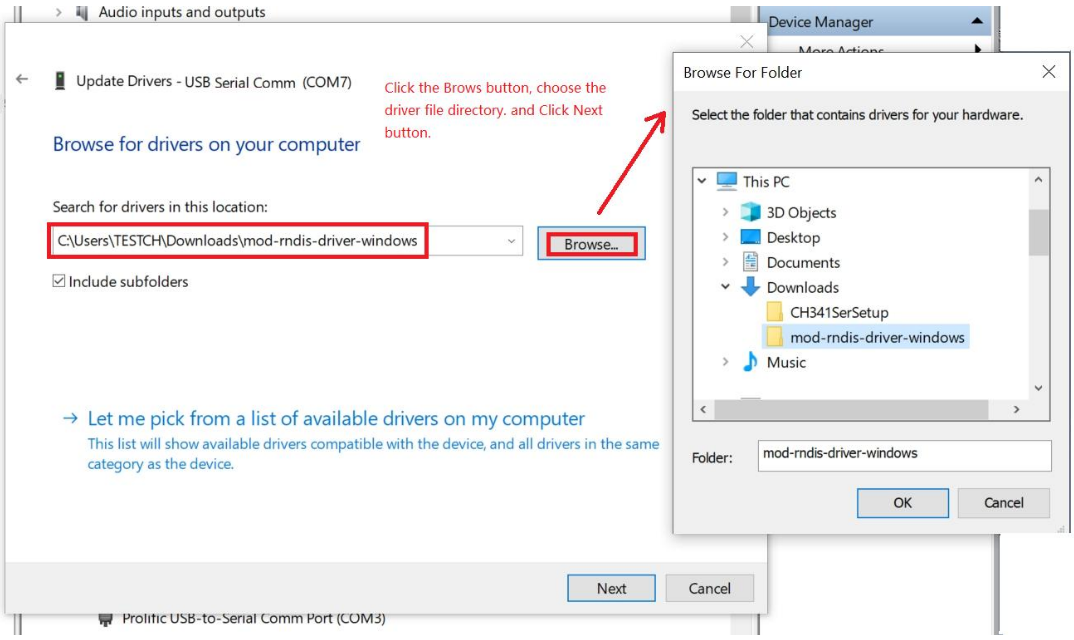



4 Step: Examine the Vendor ID (VID) and Product ID (PID) of the device to confirm that the device node originated from the GV850/1 device. The VID for the USB Type-A #2 connector is 0525, and the PID is A4A2. The way to check is illustrated in the figure below:

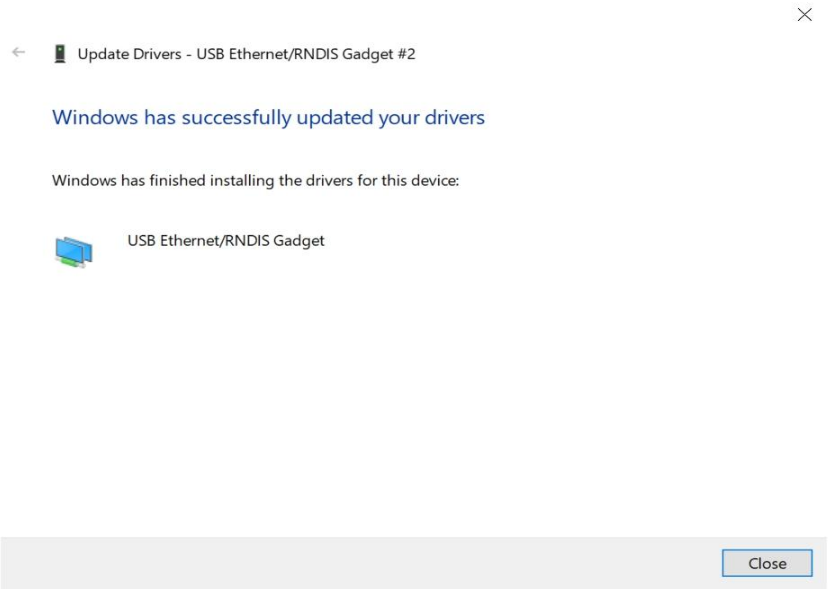



5 Step: Right-click the device and choose "Update Driver Software." Continue by selecting "Let me pick from a list of available drivers on my coCPUter" and then "Browse." Select the folder where the driver files are located. Click "Next" to initiate the driver update process.

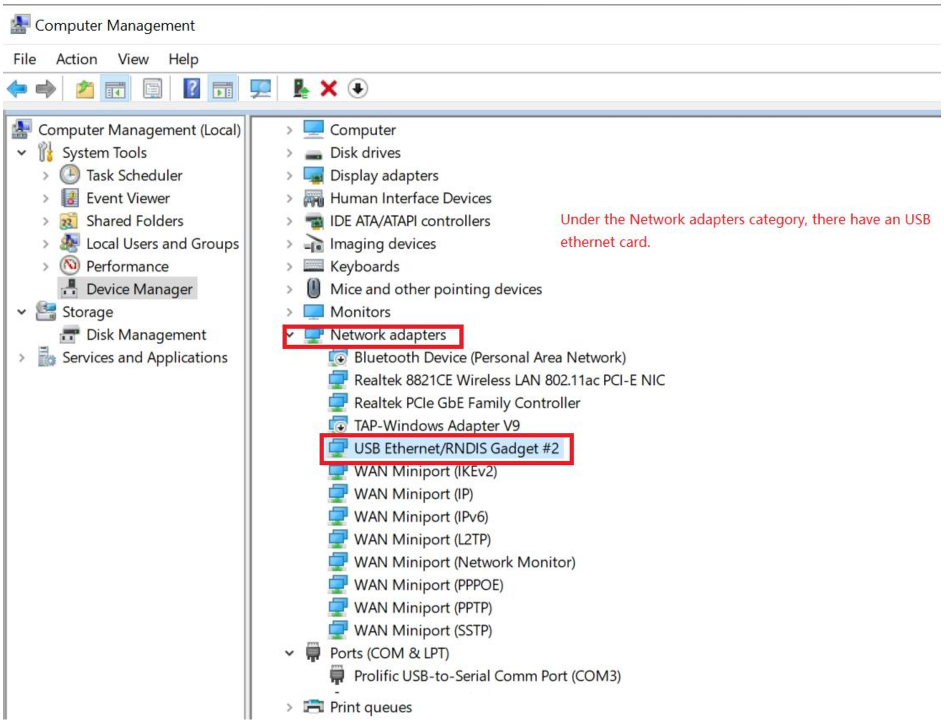


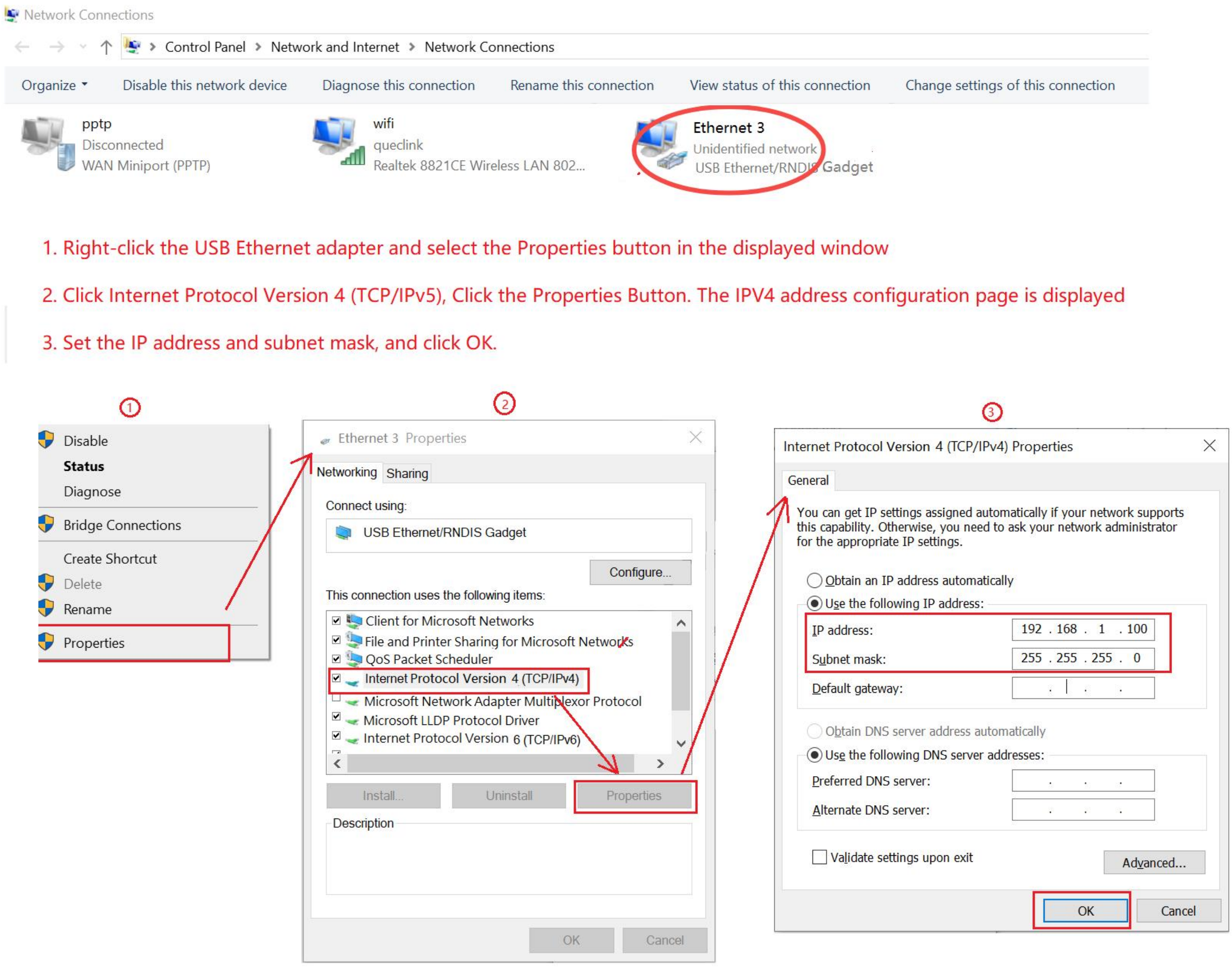

The page indicating successful driver update is as follows.



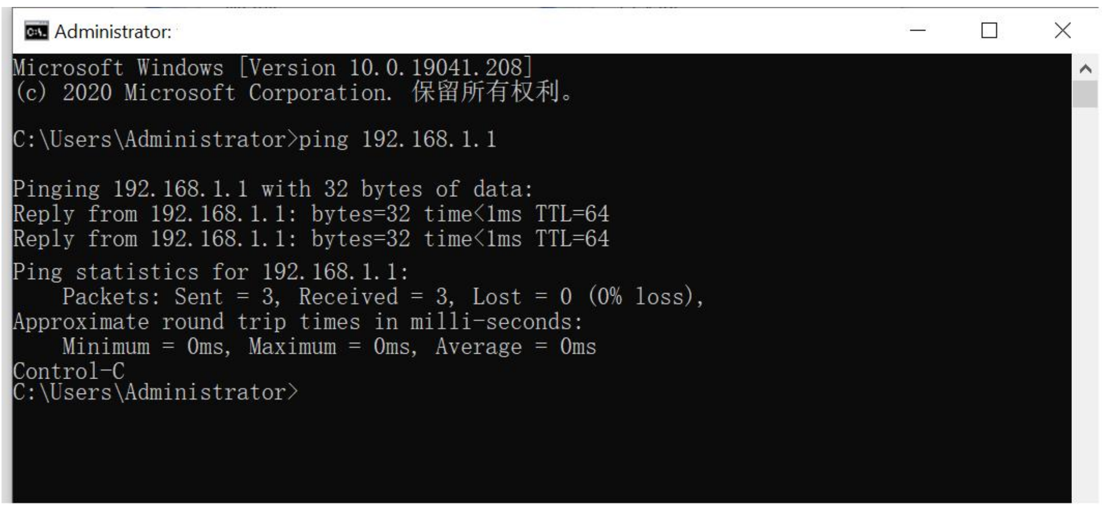

6 Step: Within the Device Manager page, under the Ethernet adapters category, check whether the device is identified as a USB Ethernet/RNDIS Gadget# device. If yes, this indicates that the driver installation has been successful; if not, the installation has failed, and you may attempt the process again. If subsequent attempts to install the driver are unsuccessful, please seek help from technical support.




7 Step: Open the Ethernet adapter configuration page, and you will find a USB Ethernet/RNDIS Gadget adapter. On this page, you can adjust the settings for the adapter. As depicted in the figure below:




8 Step: Verify the network connectivity between the coCPUter and the device. Use the ping tool to do the test, as illustrated in the figure below:



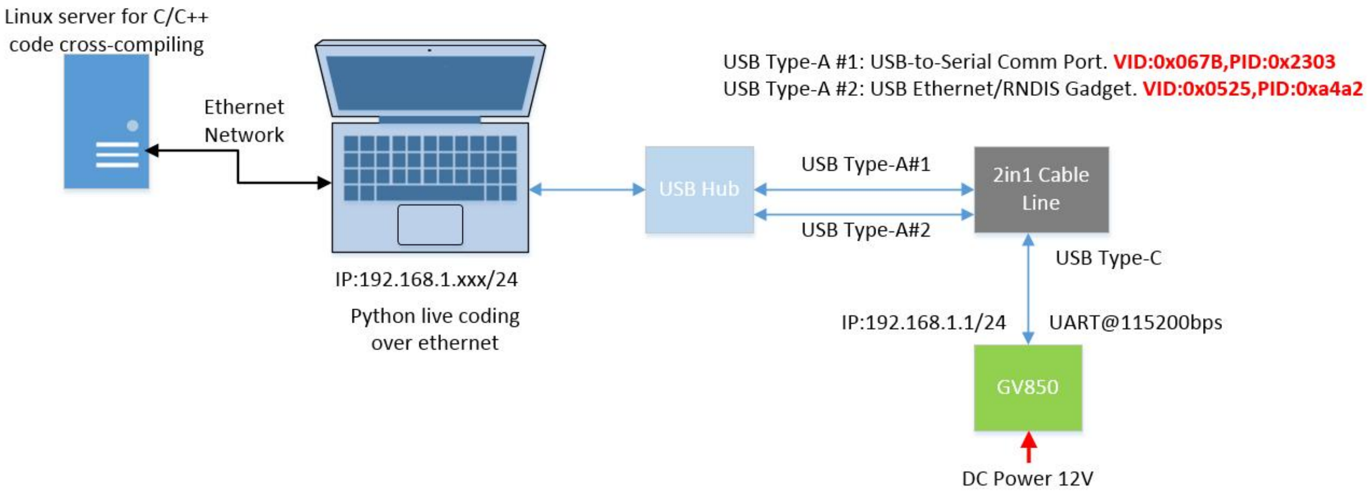
The USB virtual network interface in the GV850 device has a default IPv4 address of 192.168.1.1, with a subnet mask of 255.255.255.0. The coCPUter must configure an address for the new USB Ethernet adapter, ensuring that the IPv4 address is within the same subnet as the device. After completing this step, the development coCPUter and the device will be able to communicate via the network. An example of how to set the address on the development coCPUter is provided below:

IPv4 Address 192.168.1.100， Subnet 255.255.255.0

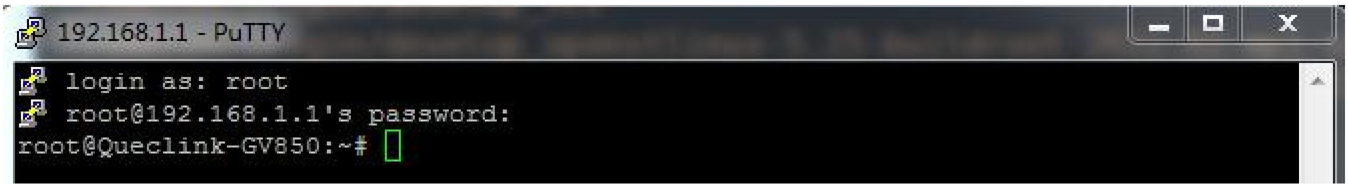
On the coCPUter, use the Ping tool to test whether the network configuration is correct.

```
$ ping 192.168.1.1
```

After the coCPUter and the GV850 device communicate properly, you can use a SSH tool to log in to the device's backend for debugging. The network topology is shown below.




SSH Login Success Example:




The USB Ethernet/RNDIS driver file for Windows is named "mod-rndis-driver-windows.zip".

### 3.3.3. Debugging Tools

The GV850 device supports the rz and sz commands and file transfer protocols, such as ZMODEM/YMODEM/XMODEM , enabling the upload and download of files to the device via tools.

Additionally, the GV850 device supports the SCP and SSH commands, and it initiates the SSHD service at startup, facilitating connections from multiple clients.

### 3.3.4. Terminal Login

The default username/password for the Linux system of the GV850 is: root/root.

#### 3.3.5. Internet Access to the Internet

There are two ways to access the Internet on the device: via LTE Cellular or USB Ethernet/RNDIS Gadget.

##### 3.3.5.1. LTE Cellular Network

For details of how to use cellular to access the Internet, refer to the "LTE" section.

##### 3.3.5.2. USB Ethernet/RNDIS Gadget Network

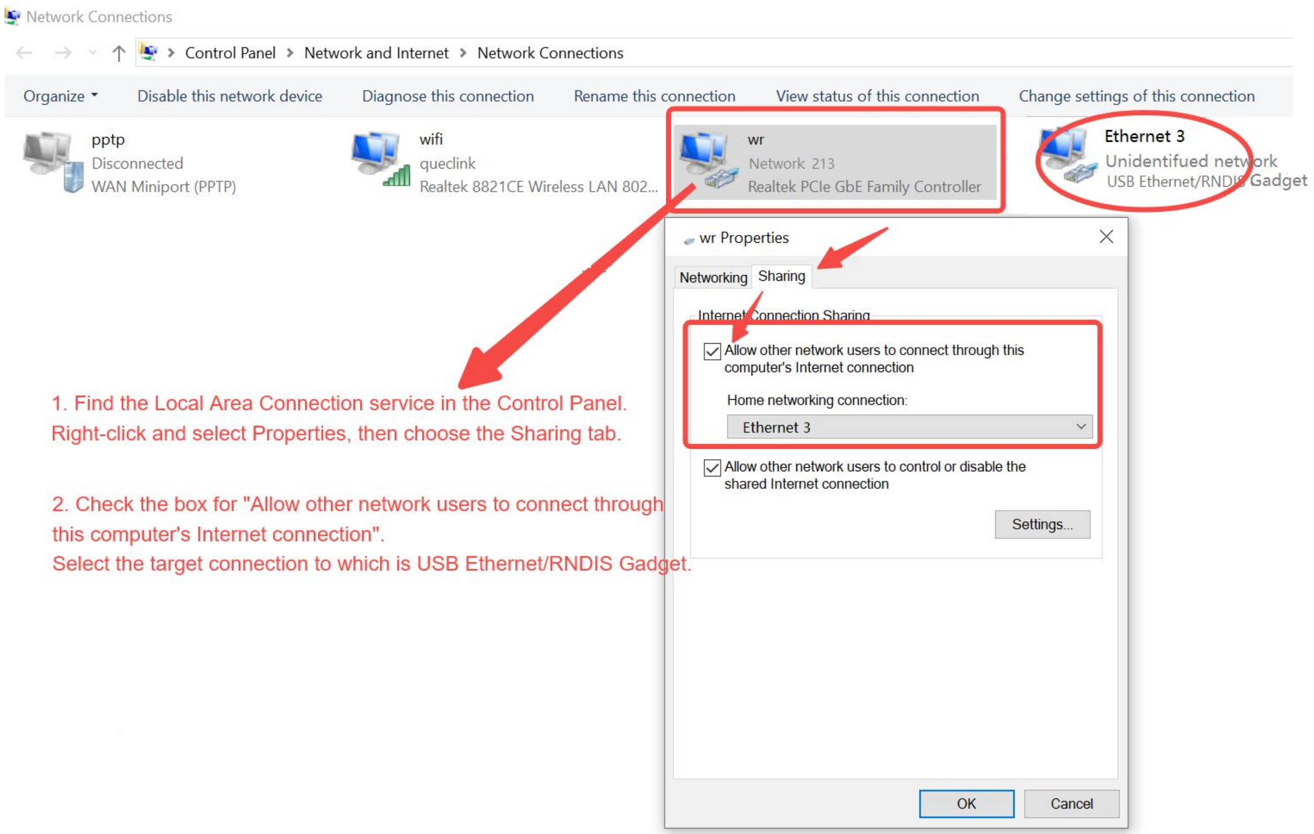
The previous chapters covered using the device's USB port to allow the coCPUter to connect to the device via a TCP/IP network for backend debugging. Based on this, further configuring the network settings of the coCPUter and the GV850 device can enable the GV850 to access the Internet through the USB port connected to the development coCPUter. The prerequisite is that the development coCPUter must be able to access the Internet. The steps are as follows:

Initially, connect the coCPUter to the Internet. Subsequently, connect the USB Type-A #2 connector to the coCPUter. Attach the Typc-C end to the GV850 device and power on the device.

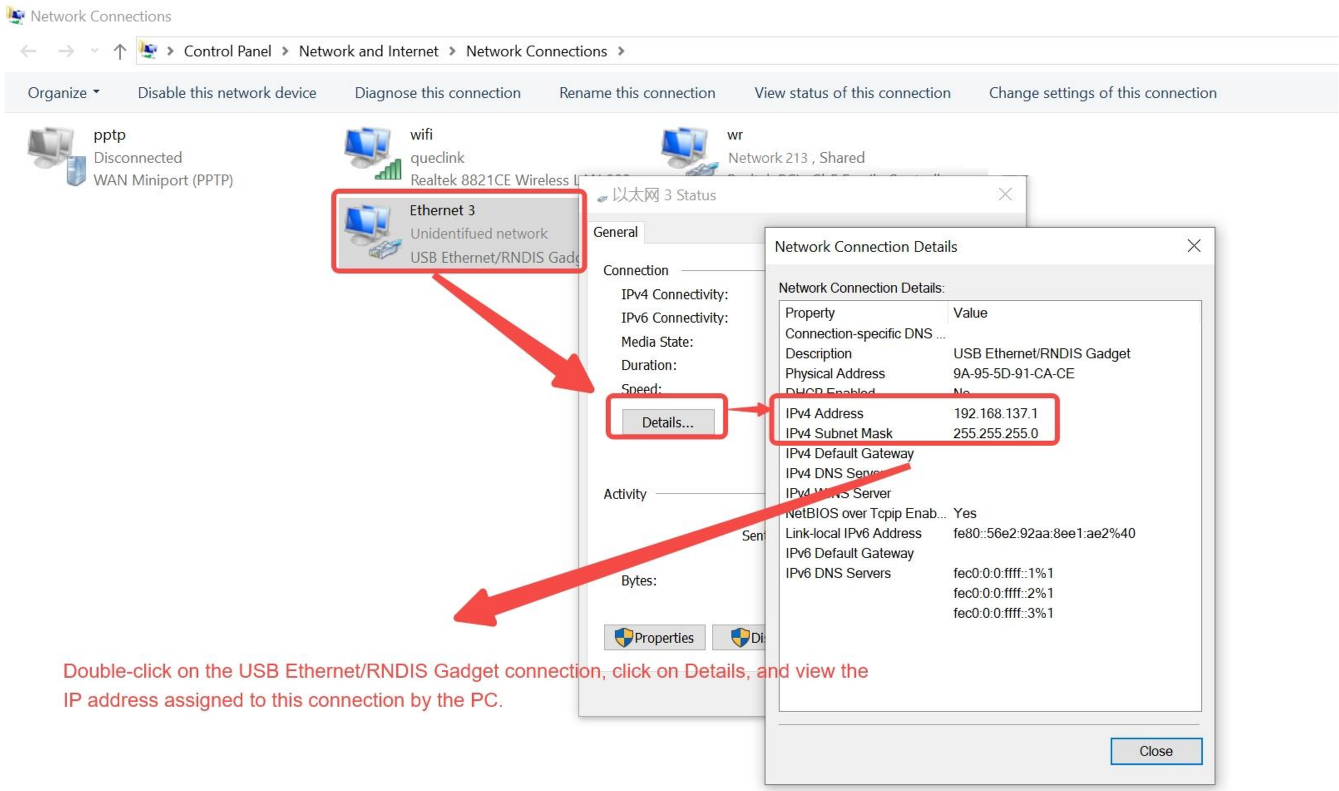
Follow the configuration method described in the "USB Debugging" section to properly network the device with the coCPUter, ensuring that the GV850 device is reachable via Ping from the developer's coCPUter. Then, share the network adapter that provides Internet access on the coCPUter with the USB Ethernet/RNDIS Gadget Adapter generated by the GV850. For the topology diagram, refer to the "USB Debugging" section.

Example:
Taking a Windows 7 PC as an example,

1 Step: In Windows, sharing the local Internet connection from the adapter to the USB Ethernet adapter that is created by the GV850.




2 Step: On the Windows system, check the IP address of the USB Ethernet adapter that has been created by the GV850 device.



3 Step: according to the information obtained in Step 2 to configure the IPv4 address and default gateway on the GV850 device.

In our example, the network segment is 192.168.137.0/24, so configure the IPv4 address of the GV850 device to 192.168.137.10. The command is as follows:

```
$ ifconfig usb0 192.168.137.10 netmask 255.255.255.0
```

The default gateway should be configured as 192.168.137.1:

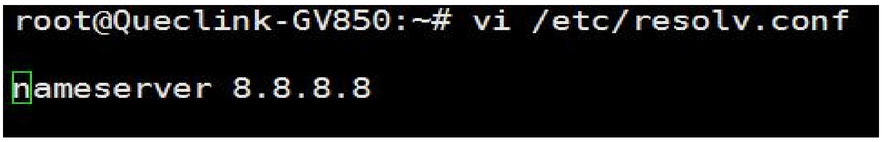
```
$ ip route add default via 192.168.137.1
```

```
root@Queclink-GV850:~# ifconfig usb0 192.168.137.10 netmask 255.255.255.0
ute add default via 192.168.137.1
root@Queclink-GV850:~# ip route add default via 192.168.137.1
root@Queclink-GV850:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.137.1   0.0.0.0         UG    0      0        0 usb0
192.168.137.0   0.0.0.0         255.255.255.0   U     0      0        0 usb0
root@Queclink-GV850:~# ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

usb0      Link encap:Ethernet  HWaddr 4E:BF:AF:13:BB:DD
          inet addr:192.168.137.10  Bcast:192.168.137.255  Mask:255.255.255.0
          inet6 addr: fe80::4cbf:afff:fe13:bbdd/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:531 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:100580 (98.2 KiB)  TX bytes:1236 (1.2 KiB)

root@Queclink-GV850:~#
```

Set the device's IP address and gateway. The IP address should be in the same network segment as the PC's address, with the gateway being the same as the IP address....

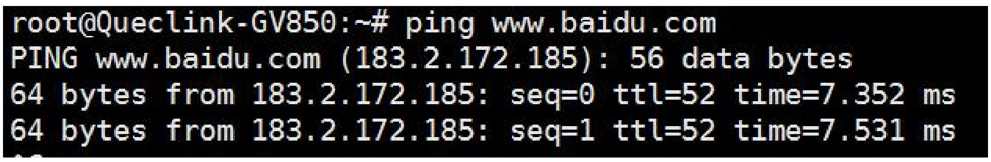
4 Step: Configure the DNS server settings on the GV850 .

```
$ vi /etc/resolv.conf
```



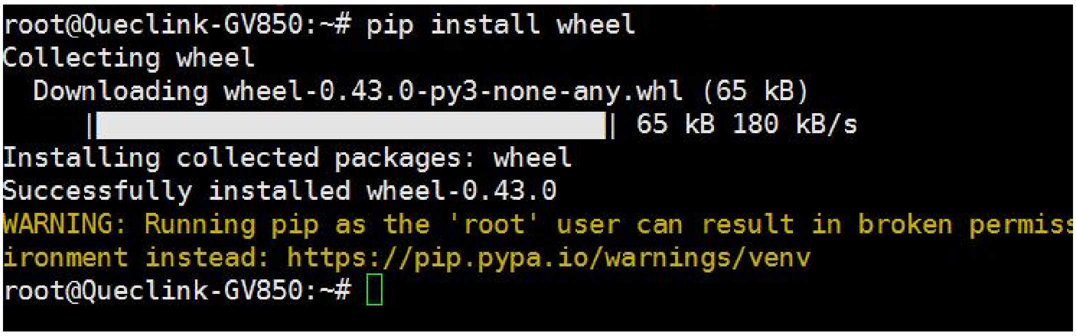
```
root@Queclink-GV850:~# vi /etc/resolv.conf

nameserver 8.8.8.8
```

5 Step: In the GV850 , test if the Internet network services are normal.



```
root@Queclink-GV850:~# ping www.baidu.com
PING www.baidu.com (183.2.172.185): 56 data bytes
64 bytes from 183.2.172.185: seq=0 ttl=52 time=7.352 ms
64 bytes from 183.2.172.185: seq=1 ttl=52 time=7.531 ms
```

6 Step: If the network is working properly on the GV850, you can then use pip to install Python packages.




Caution: After the device is rebooted, the network configurations within the device will be lost. To access the Internet, you will need to repeat the aforementioned steps.

#### 3.3.6. Modify the default IPv4 address of the USB network

When we need to modify the default USB network address of the device, we must log in to the device's backend and modify the DEFAULT_IP_ADDR variable in the /etc/init.d/S40network file. After modifying the file, run the sync command to write the changes to the flash memory. Then restart the device or run the following command:

```
$/etc/init.d/S40network restart
```

This is helpful for debugging multiple devices on one coCPUter at the same time.

### 3.4. Custom Packages

Adding a new package to the Buildroot compilation suite is quite straightforward; simply follow the official instructions provided by Buildroot. Referring to the Queclink software package, here is a brief description of the process:

Package Path:

```
package/queclink
```

Package Files:

```
package/queclink/Config.in
package/queclink/queclink.mk
package/linux-tracker-app -> ../../queclink_custom/linux-tracker-app
```

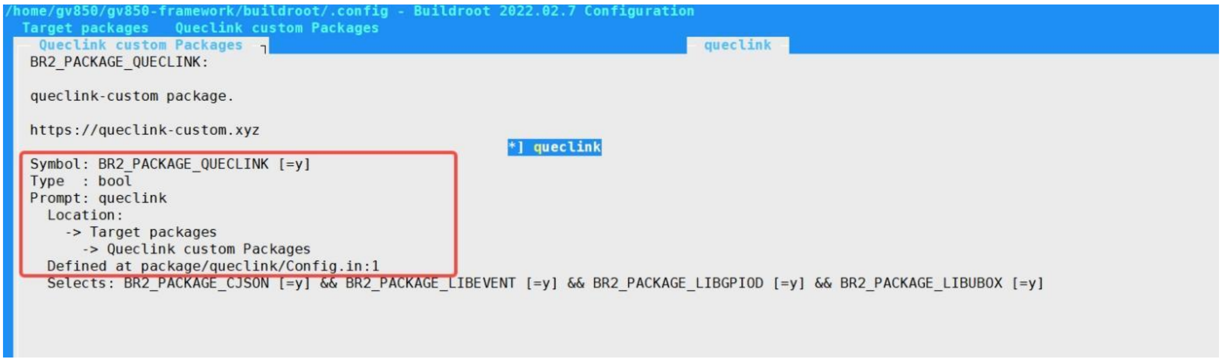
Config.in is used by the menuconfig tool to configure and manage software packages; queclink.mk is used to compile software packages. The Config.in of a custom package needs to be referenced in the upper-level Config.in.

The source code path needs to be specified in the queclink.mk file. queclink.mk specifies the package source code as package/linux-tracker-app. package/linux-tracker-app is a soft link file that connects to the real source code path. The content of Config.in, the previous level of queclink, is as follows:

```
$ cat package/Config.in
menu "Queclink custom Packages"
```

```
    source "package/queclink/Config.in"
endmenu
```

After the above configuration file, you can use make menuconfig to select the custom software package and then compile it.




```
$ make queclink-rebuild
```

### 3.5. Device logs

Logs are an essential part of development and debugging. The device environment provides two log systems, system logs and application logs. Currently, logs are stored in RAM memory files and cannot be saved permanently. They will be lost after the device is restarted. Automatic cycle overwriting is supported.

#### 3.5.1. System log

The device has integrated the syslogd log service to collect system logs. It collects logs generated by the kernel and various system services. Logs are stored in the RAM file /var/log/messages and do not support persistent storage. They will be lost after the device is restarted.

Check the logs,

```
$ dmesg
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 5.15.67 (root@31f3ee1efbea) (arm-linux-gcc.br_real (Buildroot toolchains.bootlin.com-2021.11-1) 10.3.0, GNU ld (GNU Binutils) 2.36.1) #1 SMP PREEMPT Tue Mar 19 17:43:45 CST 2024
[    0.000000] CPU: ARMv7 Processor [410fc075] revision 5 (ARMv7), cr=10c5387d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
[    0.000000] OF: fdt: Machine model: STMicroelectronics custom STM32CubeMX board -openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23
```

You can also use commands such as cat and tail to view:

```
$ tail -f /var/log/messages
Jan  1 00:00:43 Queclink-GV850 kern.err kernel: [   34.954559] usb 2-2: device descriptor read/64, error -62
Jan  1 00:00:44 Queclink-GV850 kern.info kernel: [   35.284660] usb 2-2: new low-speed USB device number 4 using ohci-platform
Jan  1 00:00:44 Queclink-GV850 kern.err kernel: [   35.504566] usb 2-2: device descriptor read/64, error -62
```

```
Jan  1 00:00:44 Queclink-GV850 kern.err kernel: [   35.834551] usb 2-2: device descriptor read/64, error -62
Jan  1 00:00:44 Queclink-GV850 kern.info kernel: [   35.954691] usb usb2-port2: attempt power cycle
```

#### 3.5.2.Application log

Currently, the Queclink application software uses the zlog log system. It is used to collect logs generated by the Queclink application. The logs are stored in the RAM file /var/log/ubus_app.log and do not support persistent storage. They will be lost after the device is restarted.

```
$ tail -f /var/log/ubus_app.log
2000-01-01 01:34:28 187 <WARN> <batterymgr> get real percent:98.4615 from 4144  --  batt_order_tab_lookup() batt_per.c:78
2000-01-01 01:34:32 187 <WARN> <batterymgr> get real percent:98.4615 from 4144  --  batt_order_tab_lookup() batt_per.c:78
2000-01-01 01:34:36 187 <WARN> <batterymgr> get real percent:98.4615 from 4144  --  batt_order_tab_lookup() batt_per.c:78
2000-01-01 01:34:40 187 <WARN> <batterymgr> get real percent:98.4615 from 4144  --  batt_order_tab_lookup() batt_per.c:78
2000-01-01 01:34:44 187 <WARN> <batterymgr> get real percent:98.4615 from 4144  --  batt_order_tab_lookup() batt_per.c:78
```

You can modify the log configuration file and then trigger a reload of the log configuration to dynamically change the log settings. The command is as follows:

The command format is ubus call module name set_logconf '{"file":"log configuration file"}'

Example:

```
$ ubus call canobd set_logconf '{"file":"/etc/ubus_app_log_debug.conf"}'
```

Configuration file contents:

```
$ cat /etc/ubus_app_log_debug.conf
[formats]

default_format = "%d(%F %T) %p <%V> %m -- %U() %f:%L%n"

[rules]

*.DEBUG "/var/log/ubus_app.log", 1MB*1; default_format
```

# 4. Interface and Driver

## 4.1. LED

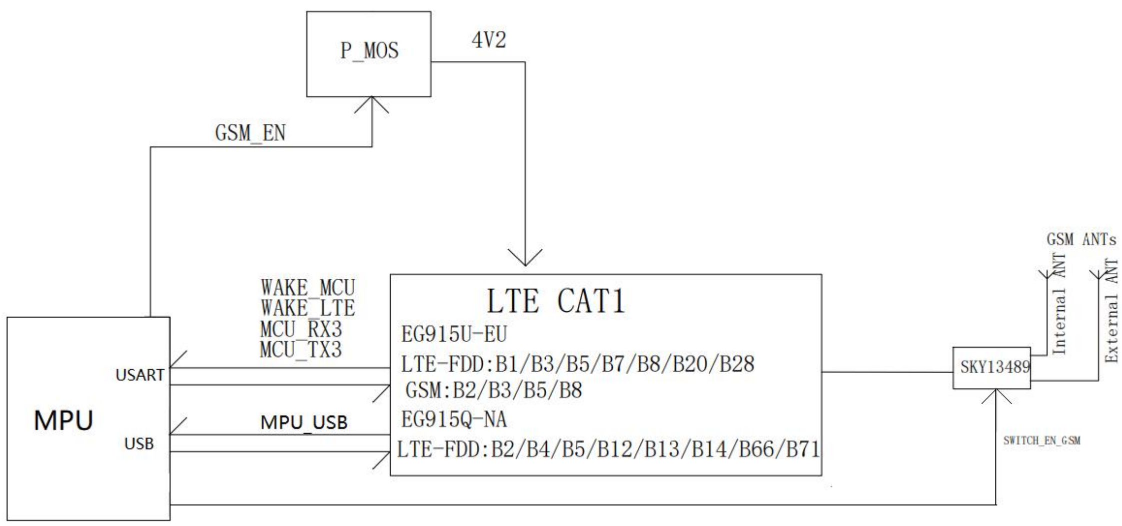
There are 3 LEDs, which are controlled by GPIO.

Hardware resource list:

| Pin Name | LED Name | Description |
|---|---|---|
| PG11 | Power supply status LED | On, PG11 output high, read led<br>$ gpioset 6 11=1<br>Off, PG11 output low<br>$ gpioset 6 11=0 |
| PA8 | GPS signal LED | On, PA8 output high, blue led<br>$ gpioset 0 8=1<br>Off, PA8 output low<br>$ gpioset 0 8=0 |
| PE6 | Network signal LED | On, PE6 output high, green led<br>$ gpioset 4 6=1<br>Off, PE6 output low<br>$ gpioset 4 6=0 |

## 4.2. LTE

Depending on the model, the module model is EG915U-EC or EG915Q-NA. Regardless of the module model, it supports two usage modes: network card mode and UART module mode. The hardware block diagram is as follows:




Hardware resource list:

| Pin Name | Description | Remarks |
|---|---|---|

| | | |
|---|---|---|
| PA15 | LTE module power control | 0:Power off<br>1:Power on |
| PF5 | LTE module power on/off control pin | 3s low pulse power-on state reverses |
| PB7 | LTE module wake-up CPU pin | 0:Wake up CPU level<br>1: Normal operating level |
| PE14 | CPU wakes up LTE module pin | 0: wake up the LTE module level<br>1: Normal operating level |
| USART3 | LTE module communicates with CPU UART | Device node: /dev/ttySTM3<br>Baud rate: 115200<br>Start bit: 1 bit<br>Data bit: 8 bits<br>Stop bit: 1 bit<br>No checksum |
| USB1 | LTE module communicates with CPU via USB port | Device node: /dev/ttyUSBx |
| PA4 | LTE module enables USB function on VBUS pin | 0: Disable 5V boost, module VBUS voltage 0V<br>1: Turn on 5V boost, module VBUS voltage 5V |
| PE2 | LTE module internal and external antenna switching pin | 0: Connect to an external antenna<br>1: Connect to the built-in antenna |
| PE6 | LTE module indicator, green | 0: Turn off the light<br>1: Turn on the light |

If you are running the Queclink pre-installed software, close the selftask program before manual testing to avoid serial port resource conflicts.

```
$ /etc/init.d/S99selftask stop
$
```

#### 4.2.1 UART Modem

When work as a UART module, use USART3, corresponding to the device node /dev/ttySTM3.

Reference testing commands as follows:

Set the baud rate to 115200bps and remove the incrnl attribute to avoid automatically converting input characters \r to \n. Remove the isig icanon echo echoe attribute to avoid output causing incorrect module command format +CME ERROR: 58 error.

```
$ stty -F /dev/ttySTM3 ispeed 115200 ospeed 115200 cs8 -icrnl -isig -icanon -echo -echoe
```

Receive module uart output,

```
$ cat /dev/ttySTM3 &
```

PA15 module power supply enable output high,

```
$ gpioset 0 15=1
```

PF5 module startup signal,

```
$ gpioset 5 5=1
$ sleep 3
```

```
$ gpioset 5 5=0
```

Receive the module startup URC message,

```
RDY
```

Turn off echo，

```
$ echo "ATE0" > /dev/ttySTM3
```

Internal antenna or external antenna can be selected, through GPIO PE2 pin. The example is as follows:

Select internal antenna：

```
$ gpioset 4 2=1
```

Select external antenna：

```
$ gpioset 4 2=0
```

The EG915U-EC and EG915Q-NA modules have Pin to Pin compatibility for wake-up and sleep pins, but the test commands are different.

WAKE_ LTE (DTR pin PE14) controls the sleep of the module, high level allows sleep, and low level wakes up the module,

```
$ gpioset 4 14=0
```

EG915U-EC modem, Query the DTR pin status via command, if it is 0, sleep is not allowed,

```
$ echo "AT+QGPIOR=25" > /dev/ttySTM3
+QGPIOR: 0

OK
```

The EG915Q-NA module does not respond to AT commands when in sleep mode, but it can respond to AT commands when not in sleep mode. The CPU determines whether the module should sleep by controlling the wake-up pin of the LTE module, which allows for testing the functionality of this pin. The wake-up test command is as follows.

```
$ gpioset 0 4=0
$ echo "AT+QSCLK=1" > /dev/ttySTM3
$ sleep 4
$ echo "AT" > /dev/ttySTM3

OK
```

If the module can reply, it means the module has not entered sleep mode.

DTR pin output high level,

```
$ gpioset 4 14=1
```

EG915U-EC modem, Query the DTR pin status via command, if it is 1, sleep is allowed

```
$ echo "AT+QGPIOR=25" > /dev/ttySTM3
+QGPIOR: 1
```

```
OK
```

The EG915Q-NA module enters sleep mode and no longer responds to commands. The sleep test command is as follows.

```
$ gpioset 0 4=0
$ echo "AT+QSCLK=1" > /dev/ttySTM3
$ sleep 4
$ echo "AT" > /dev/ttySTM3
```

```
(No response with AT cmd.)
```

If the module can reply, it means the module has not entered sleep mode.

Send the AT+QSCLK=1 command to enable sleep function,

```
$ echo "AT+QSCLK=1" > /dev/ttySTM3
OK
```

Sending any AT command will wake up the module, but at the appropriate time, it will enter sleep again unless the DTR pin output is at low level or the sleep function is turned off using the AT+QSCLK=0 command.
The module can notify the CPU through the level change of the WAKE_MCU (RI pin PB7). Due to the rapid level change, it is not possible to accurately obtain it using gpioget. Therefore, the example_input_intr tool can be used for monitoring it.

For example, using command to turn off the module,

```
$ echo "AT+QPOWD" > /dev/ttySTM3
POWERED DOWN
```

Monitoring receives GPIO level change events,

```
$ example_input_intr &
type:1, code:261, value:0
type:1, code:261, value:1
type:1, code:261, value:0
```

In the EG915U-EC module, the Modem can be controlled to wake up the CPU GPIO pin using an AT command, which corresponds to the CPU's GPIO PB7, and on the Modem side, it is Pin 26. The AT command is AT+QGPIOW=26,0/1. The test example is as follows:

First, start the example_input_intr tool to monitor GPIO PB7 and run it in the background. When the status of GPIO PB7 changes, an event printout will be generated.

```
$ example_input_intr &
```

Control the module to input a low-level signal to GPIO PB7 through the command as follows.

```
$ echo "AT+QGPIOW=26,0" > /dev/ttySTM3
OK
type:1, code:261, value:1
```

Control the module to input a high-level signal to GPIO PB7 through the command as follows.

```
$ echo "AT+QGPIOW=26,1" > /dev/ttySTM3
OK
type:1, code:261, value:0
```

When the device uses the EG915Q-NA module, the test commands are different. Control the module to input a low-level signal to GPIO PB7 through the command.

```
$ AT+QGPIOCFG=1,28,1,3,0
$ AT+QGPIOCFG=AT+QGPIOCFG=3,28,0
OK
type:1, code:261, value:1
```

Control the module to input a high-level signal to GPIO PB7 through the command.

```
$ AT+QGPIOCFG=AT+QGPIOCFG=3,28,1
OK
type:1, code:261, value:0
```

Use the provided example_modem_at tool for command testing, as detailed in the "Example of Codes" section.
The LTE module can serve as a wake-up source for system sleep, as detailed in the "System Sleep" section.

The following demonstrates the process of how to connect to the network, send and receive TCP data.

Check for correct SIM card reading,

```
$ echo "AT+CPIN?" > /dev/ttySTM3
+CPIN: READY

OK
```

Check CS status,

```
$ echo "AT+CREG?" > /dev/ttySTM3
+CREG: 0,1

OK
```

Attach PS domain,

```
$ echo "AT+CGATT=1" > /dev/ttySTM3
OK
$ echo "AT+CGATT?" > /dev/ttySTM3
+CGATT: 1

OK
```

Activate PDP,

```
$ echo "AT+QIACT=1" > /dev/ttySTM3
OK
```

Check the PDP status and obtained IP address,

```
$ echo "AT+QIACT?" > /dev/ttySTM3
+QIACT: 1,1,3,"10.162.247.73","2408:8456:3040:AB7:1:1:A0D9:4891"

OK
```

Ping domain name to check network connectivity,

```
$ echo "AT+QPING=1,\"www.baidu.com\"" > /dev/ttySTM3
OK

+QPING: 0,"157.148.69.74",64,313,255
+QPING: 0,"157.148.69.74",64,61,255
+QPING: 0,"157.148.69.74",64,61,255
+QPING: 0,"157.148.69.74",64,50,255
+QPING: 0,4,4,0,50,313,87
```

Open socket, using 218.17.50.142:971 server/port as the example,

```
$ echo "AT+QIOPEN=1,0,\"TCP\",\"218.17.50.142\",971,0,0" > /dev/ttySTM3
OK

+QIOPEN: 0,0
```

Check the status of the socket and confirm that it is connected,

```
$ echo "AT+QISTATE?" > /dev/ttySTM3
+QISTATE: 0,"TCP","218.17.50.142",971,0,2,1,0,0,"uart1"

OK
```

Send the test string '12345' in HEX format,

```
$ echo "AT+QISENDEX=0,\"3132333435\"" > /dev/ttySTM3
SEND OK
```

The server responds with data '67890', and the module will notify the module with a URC message upon receiving the data,

```
+QIURC: "recv",0
```

At this point, the received data can be read from the cache and the actual length and data will be returned,

```
$ echo "AT+QIRD=0,1500" > /dev/ttySTM3
+QIRD: 5

67890
```

```
OK
```

Close socket,

```
$ echo "AT+QICLOSE=0" > /dev/ttySTM3
OK
```

Check the status of the socket and confirm that it is closed,

```
$ echo "AT+QISTATE?" > /dev/ttySTM3
OK
```

#### 4.2.2 Ethernet Adapter

The Modem's USB port is connected to the CPU's USB Host Controller. Setting the Modem USB VBUS to high will enable the Model's USB functionality. The GV850 device comes preloaded with the necessary drivers, allowing the module to be used as a network device in the Linux system.

It is important to note that the module can only establish one PDP connection at a time. If a PDP connection has been established in the module's UART modem mode, it needs to be closed first. Then, the modem can be used as an Ethernet Adapter.

If you are running the pre-installed software from Queclink, please shut down the selftask program before testing. This software will automatically start upon boot and create a PDP for the module. To avoid PDP conflicts, it is necessary to stop the program first.

```
$ /etc/init.d/S99selftask stop
```

Restart the module.

```
$ gpioset 0 15=1;sleep 1;gpioset 5 5=1;sleep 3;gpioset 5 5=0
```

Then, turn on the Modem USB VBUS 5V, and the module starts working in USB device mode. Taking the EG915U-EC model as an example, you can then see the system begin to enumerate and recognize the device.

```
$ gpioset 0 4=1
[   642.617144] usb 1-1: new high-speed USB device number 2 using ehci-platform
[   642.817845] usb 1-1: config 1 interface 0 altsetting 0 endpoint 0x81 has an invalid bInterval 32, changing to 9
[   642.836277] cdc_ether 1-1:1.0 usb1: register 'cdc_ether' at usb-5800d000.usbh-ehci-1, CDC Ethernet Device, 02:4b:b3:b9:eb:e5
[   643.020467] usbcore: registered new interface driver option
[   643.026565] usbserial: USB Serial support registered for GSM modem (1-port)
[   643.033358] option 1-1:1.2: GSM modem (1-port) converter detected
[   643.040808] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB0
[   643.048384] option 1-1:1.3: GSM modem (1-port) converter detected
[   643.054752] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB1
[   643.062429] option 1-1:1.4: GSM modem (1-port) converter detected
[   643.069557] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB2
[   643.076386] option 1-1:1.5: GSM modem (1-port) converter detected
[   643.083580] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB3
```

```
[   643.091082] option 1-1:1.6: GSM modem (1-port) converter detected
[   643.098129] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB4
[   643.105420] option 1-1:1.7: GSM modem (1-port) converter detected
[   643.112104] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB5
[   643.119807] option 1-1:1.8: GSM modem (1-port) converter detected
[   643.126915] usb 1-1: GSM modem (1-port) converter now attached to ttyUSB6
```

The lsmod command shows that the drivers option and usb_wwan have been automatically loaded.

```
$ lsmod
Module                  Size  Used by    Tainted: G
option                 49152  0
usb_wwan               20480  1 option
...
```

The VID/PID of the EG915U-EC and EG915Q-NA modules are different, which can be queried through the lsusb command. For the EG915U-EC module, the VID/PID is 2c7c:0901, and for the EG915Q-NA module, the VID/PID is 2c7c:6007.

```
$ lsusb
```

| model | PID/VID | Device Type | Device Node | Description |
|---|---|---|---|---|
| EG915U-EC | 0x2c7c<br>0x0901 | network | usb1 | ECM/RNDIS |
| | | TTY | /dev/ttyUSB0 | AT Command |
| | | | /dev/ttyUSB1 | DIAG |
| | | | /dev/ttyUSB2 | MOS |
| | | | /dev/ttyUSB3 | CP log |
| | | | /dev/ttyUSB4 | AP log |
| | | | /dev/ttyUSB5 | Modem |
| | | | /dev/ttyUSB6 | GNSS |
| EG915Q-NA | 0x2c7c<br>0x6007 | network | usb1 | ECM/RNDIS |
| | | TTY | /dev/ttyUSB0 | AT Command |
| | | | /dev/ttyUSB1 | Log |
| | | | /dev/ttyUSB2 | modem |
| | | | /dev/ttyUSB3 | |

The quectel-CM tool can be used to quickly establish a data connection. quectel-CM can be run in the background to prevent printing from affecting command line operations.

```
$ quectel-CM
[01-01_16:19:05:810] QConnectManager_Linux_V1.6.5.1
[01-01_16:19:05:820] Find /sys/bus/usb/devices/1-1 idVendor=0x2c7c idProduct=0x901, bus=0x001, dev=0x002
[01-01_16:19:05:823] Auto find qmichannel = /dev/ttyUSB0
[01-01_16:19:05:823] Auto find usbnet_adapter = usb1
[01-01_16:19:05:825] netcard driver = cdc_ether, driver version = 5.15.67
[01-01_16:19:05:828] Modem works in ECM_RNDIS_NCM mode
...
```

```
[01-01_16:47:38:086] ip link set dev usb1 up
[01-01_16:47:38:104] busybox udhcpc -f -n -q -t 5 -i usb1
udhcpc: started, v1.35.0
[01-01_16:47:38:143] AT< +QNETDEVSTATUS: 1
udhcpc: broadcasting discover
udhcpc: broadcasting select for 10.141.9.199, server 192.168.1.1
udhcpc: lease of 10.141.9.199 obtained from 192.168.1.1, lease time 30840
[01-01_16:47:38:388] deleting routers
[01-01_16:47:38:444] adding dns 120.80.80.80
[01-01_16:47:38:444] adding dns 221.5.88.88
...
```

After the data connection is successfully established, you can query that the local network device usb1 has obtained the assigned IP address.

```
$ ifconfig usb1
usb1        Link encap:Ethernet    HWaddr 02:4B:B3:B9:EB:E5
            inet addr:10.69.160.209    Bcast:10.69.160.255    Mask:255.255.255.0
            inet6 addr: 2408:8456:3010:9093:4b:b3ff:feb9:ebe5/64 Scope:Global
            inet6 addr: fe80::4b:b3ff:feb9:ebe5/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST    MTU:1500    Metric:1
            RX packets:11 errors:0 dropped:0 overruns:0 frame:0
            TX packets:23 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:1871 (1.8 KiB)    TX bytes:2977 (2.9 KiB)
```

Query the routing table and you can see that the gateway address has been obtained.

```
$ route –n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.69.160.1     0.0.0.0         UG    0      0        0 usb1
10.69.160.0     0.0.0.0         255.255.255.0   U     0      0        0 usb1
...
```

The DNS server address has also been obtained.

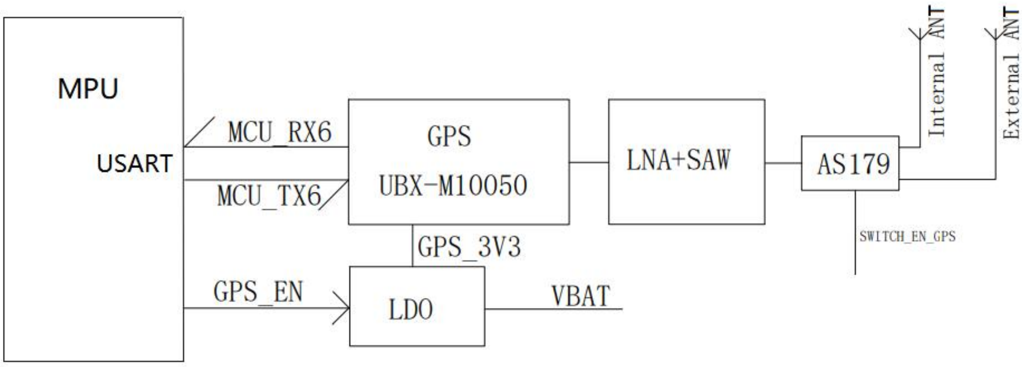
```
$ cat /etc/resolv.conf
nameserver 120.80.80.80 # usb1
nameserver 221.5.88.88 # usb1
```

You can test the connectivity of the network.

```
$ ping baidu.com
PING baidu.com (39.156.66.10): 56 data bytes
64 bytes from 39.156.66.10: seq=0 ttl=48 time=73.783 ms
64 bytes from 39.156.66.10: seq=1 ttl=48 time=57.459 ms
...
```

### 4.3. GNSS

Module model: UBX_ M10050, connected through UART. It uses USART6, which corresponds to /dev/ttySTM6. It supports u-blox and NMEA protocols.




硬件资源列表:

| Pin Name | Description | Remarks |
|---|---|---|
| PD13 | GNSS module power control pin | 0: Power off<br>1: Power on |
| PB2 | GNSS module built-in, external antenna detection pin. | 0: Detected that an external antenna is inserted<br>1: No external antenna detected<br>For this function test, the GPS chip must be powered on, otherwise the test will be invalid. |
| USART6 | GNSS module and CPU communication port | Device node: /dev/ttySTM6<br>Baud rate: 38400<br>Start bit: 1bit<br>Data bit: 8bits<br>Stop bit: 1bit<br>No checksum |

The reference testing commands are as follows:

Set the baudrate (38400 by default for M10050),

```
$ stty -F /dev/ttySTM6 ispeed 38400 ospeed 38400 cs8
```

PD13 power supply enable output high level

```
$ gpioset 3 13=1
```

Receive the NMEA data sent by the GPS module

```
$ cat /dev/ttySTM6
$GNRMC,041722.00,A,2234.41319,N,11356.88217,E,0.002,,050923,,,D,V*11
$GNVTG,,T,,M,0.002,N,0.005,K,D*3F
$GNGGA,041722.00,2234.41319,N,11356.88217,E,2,12,0.52,111.9,M,-2.7,M,,*5A
$GNGSA,A,3,11,15,24,20,23,29,05,13,18,,,,0.97,0.52,0.82,1*06
$GNGSA,A,3,09,36,10,34,05,11,,,,,,,0.97,0.52,0.82,3*0F
$GNGSA,A,3,07,13,28,02,06,59,16,40,27,09,30,20,0.97,0.52,0.82,4*0E
$GNGSA,A,3,,,,,,,,,,,,,0.97,0.52,0.82,5*06
```

(...)

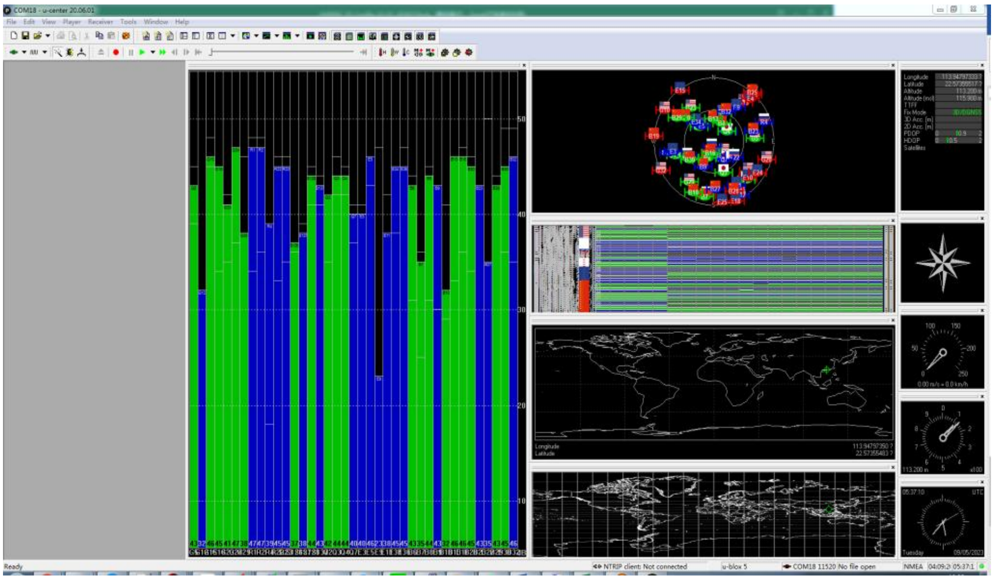
Additionally, NMEA data can be forwarded to RS232_<N> serial port, and then open RS232 through the u-center tool to more intuitively parse NMEA data. The following example is to forward NMEA data to RS232_ 2 serial ports.
Set RS232_2 baud rate to same 38400:

```
$ stty -F /dev/ttySTM2 ispeed 38400 ospeed 38400 cs8
```

Forward the data to RS232_2:

```
$ cat /dev/ttySTM6 > /dev/ttySTM2
```

Then select the correct COM port and baud rate in the u-center tool to start receiving and parsing data.

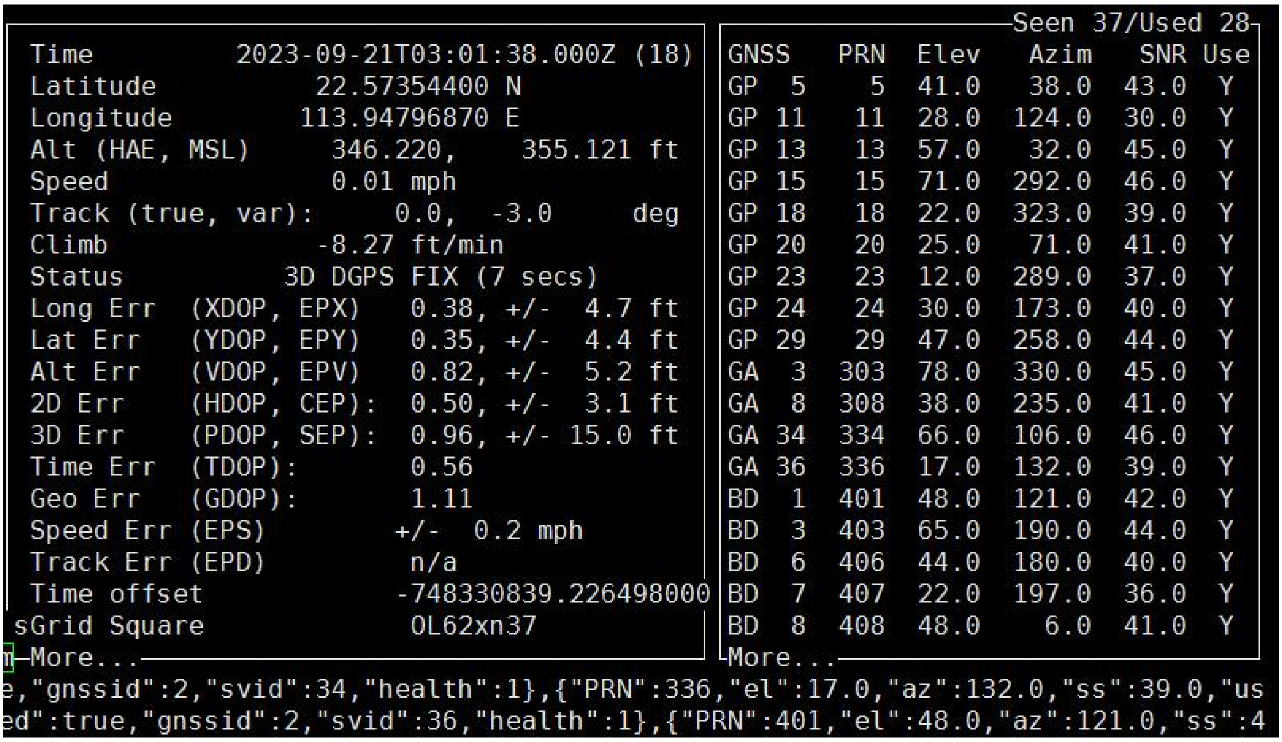


Linux system can also provide parsing and control tools that support multi-protocol such as NMEA and u-blox through the integration of gpsd software. For more information on how to use the tools, please refer to the following website:

- https://gpsd.io/

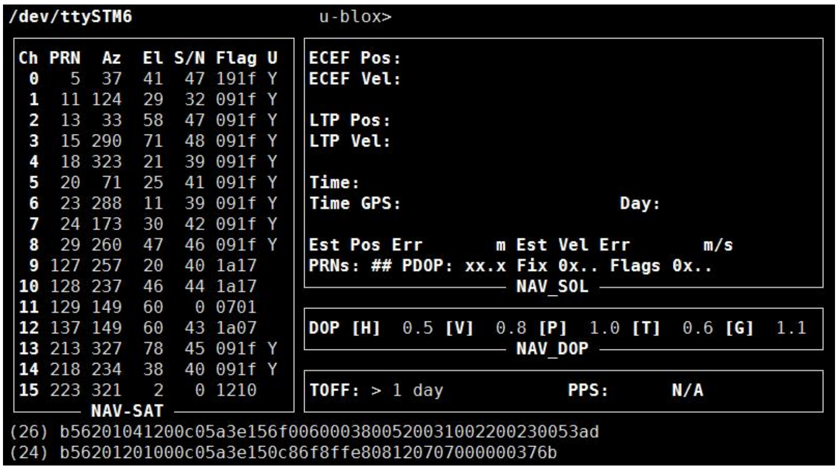
cgps and gpsmon tools can instantly parse and display positioning data. cgps receives and parses JSON data containing positioning data information provided by gpsd services. And gpsmon directly parses and displays the raw data of the GPS module. Depending on the protocol supported by the module, choose to use u-blox or NMEA protocol accordingly.

```
$ cgps
```




$ gpsmon

The results of parsing data using the u-blox protocol:




$ gpsmon –n

The results of parsing data using the NMEA protocol:

```
/dev/ttySTM6                    NMEA0183>

 Time: 2023-09-21T03:02:29.000Z   Lat: 22 34.413000' N   Lon: 113 56.877900' E
                                  Cooked TPV

 GPZDA GPGGA GPRMC GPGSA GPGBS GPGSV
                                  Sentences

 SVID  PRN  Az El SN HU | Time:       030229.00      | Time:       030229.00
 GP  5    5  38 41 44  Y | Latitude:     2234.4130 N | Latitude:   2234.4130
 GP 13   13  32 57 45  Y | Longitude:  11356.8779 E  | Longitude:  11356.8779
 GP 15   15 293 71 46  Y | Speed:      0.0078        | Altitude:   106.85
 GP 18   18 323 22 40  Y | Course:     0.000         | Quality:    2   Sats: 32
 GP 20   20  72 24 40  Y | Status:     A        FAA: | HDOP:       0.50
 GP 23   23 289 12 39  Y | MagVar:     -3.0 W        | Geoid:      -2.98
 GP 24   24 173 31 40  Y |            RMC            |            GGA
 GP 29   29 258 47 44  Y |
 GP  0    0 149 60 40  N | Mode: A3 Sats: 5 13 15 +  | UTC:            RMS:
 GP 11   11 125 28 22  N | DOP H=0.5   V=0.8  P=1.0  | MAJ:            MIN:
 GP 30   30  44  5  0  N | TOFF: > 1 day             | ORI:            LAT:
 SB127   40 257 20 39  N | PPS: N/A                  | LON:            ALT:
    v    GSV             |        GSA + PPS          |            GST
(76) $GPGSV,10,10,40,194,12,151,38,195,57,141,45,196,65,049,44,199,60,149,40*79
```

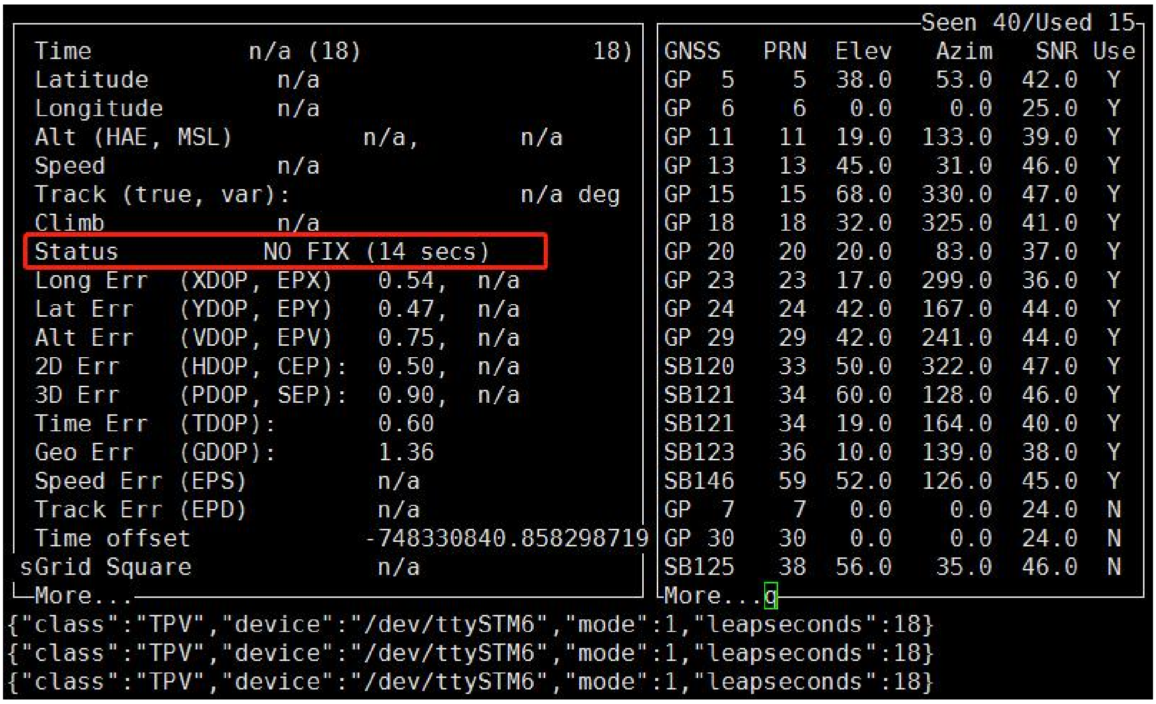
Check u-blox version:

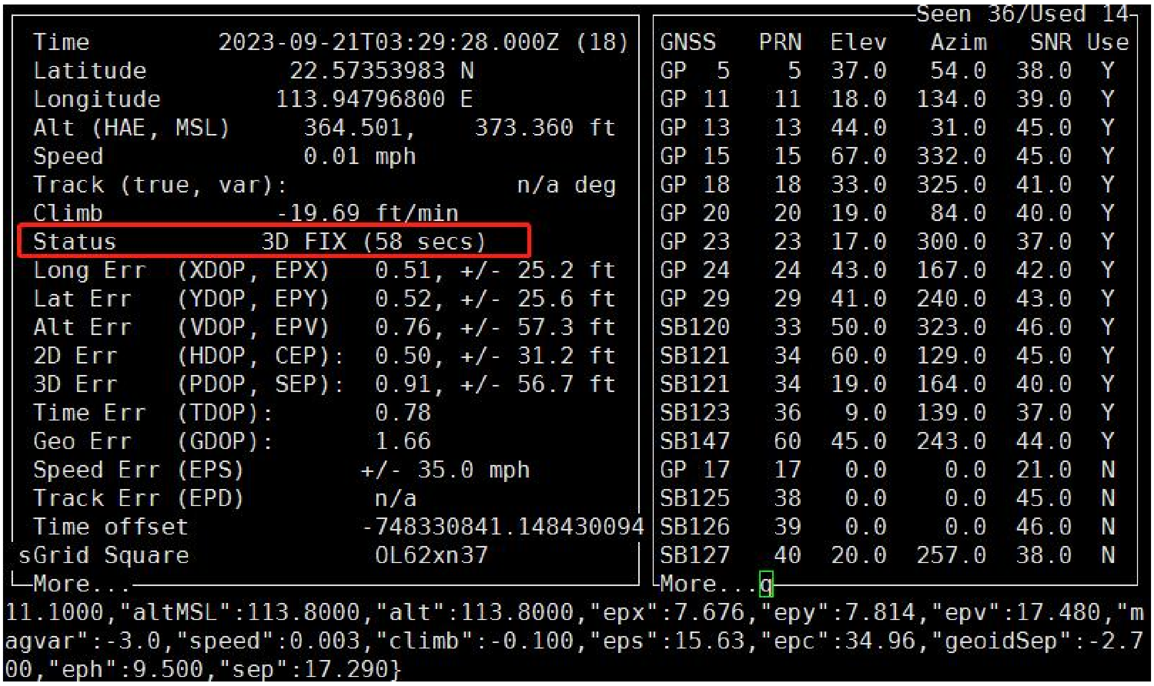
```
$ ubxtool -p MON-VER
```

```
UBX-MON-VER:
    swVersion ROM SPG 5.10 (7b202e)
    hwVersion 000A0000
    extension FWVER=SPG 5.10
    extension PROTVER=34.10
    extension GPS;GLO;GAL;BDS
    extension SBAS;QZSS
WARNING:    protVer is 10.00, should be 34.10.    Hint: use option "-P 34.10"

UBX-NAV-PVT:
    iTOW 357228000 time 2023/9/21 03:13:30 valid x37
    tAcc 24 nano -443396 fixType 3 flags x3 flags2 xea
    numSV 32 lon 1139479540 lat 225735412 height 110133
    hMSL 112846 hAcc 580 vAcc 1212
    velN -1 velE 2 velD 17 gSpeed 2 headMot 0
    sAcc 112 headAcc 17333086 pDOP 103 reserved1 0 16476 12118
    headVeh 3102272 magDec 0 magAcc 0
    (…)
```

You can use the following commands to perform cold start and calculate the time it takes from no positioning to positioning by the status change of the cgps monitoring tool:

```
$ ubxtool -p COLDBOOT -P 34.10
```



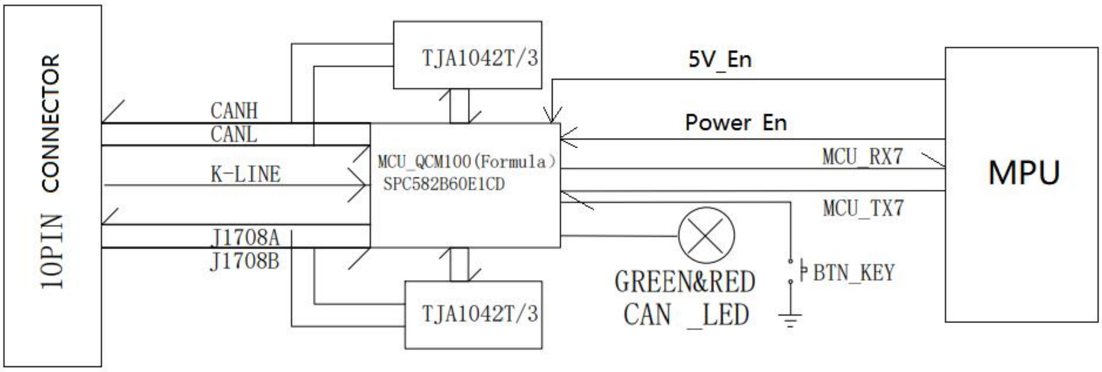

## 4.4. CAN Module

### 4.4.1 GV80 CAN OBD Module

GV850 uses the integrated CAN OBD module to process CAN messages and vehicle-related data.

The SPC582B60E1 module is connected to the CPU via USART7 UART, corresponding to /dev/ttySTM7. It supports vehicle-mounted CAN protocols such as J1939, J1708, FMS, and OBD. The module automatically parses vehicle parameters for host queries and supports a wide range of vehicle types, covering most mainstream models available on the market.

Additionally, it includes a Tachograph Reader function for reading Tachograph data and downloading driving record files remotely. The module also offers KLine functionality.




Hardware resource list:

| Pin Name | Description | Remarks |
|---|---|---|
| PG3 | CAN module power control pin | 0: Power off<br>1: Power on |
| PA4 | The device 5V boost is enabled, and this function needs to be enabled for the CAN module to work properly. | 1: Enable<br>0: Disable |
| PG1 | CAN module wakes up the CPU pin | 0: CAN module is in working state<br>1: CAN module is in sleeping state |
| USART7 | CAN module and CPU communication port | Device node: /dev/USART7<br>Baud rate: 115200<br>Start bit: 1bit<br>Data bit: 8bits<br>Stop bit: 1bit<br>No checksum |

If the pre-installed Queclink software is running, please shut down the canobd process before testing. This software will automatically start upon boot and open the serial port /dev/ttySTM7. To prevent conflicts, it is necessary to stop the program first.

```
$ /etc/init.d/S70canobd stop
```

Restart the CAN module.

```
$ gpioset 6 3=0;sleep 3;gpioset 6 3=1
```

Set the baud rate (default) to 115200, and because the module serial port data is binary, the parameter raw needs to be used when using the stty tool to set it. Otherwise, the default tty attribute may overwrite the read data, such as the enabled icrnl attribute by default, which will overwrite 0x0D with 0x0A.

```
$ stty -F /dev/ttySTM7 ispeed 115200 ospeed 115200 cs8 raw
```

PG3 CAN MCU Power supply output enable：

```
$ gpioset 6 3=1
```

PA4 5V voltage increase enable：

```
$ gpioset 0 4=1
```

On Linux system, the read and written binary data can be edited by using the hexedit tool, and then read and write by using the dd tool.

For example, write the binary command to be sent into the file out:

```
$ touch out
$ hexedit out
```

```
00000000 F5 B3 10 01 3B F6                                  ....;.
00000010
00000020
00000030
00000040
00000050
00000060
```

Start reading in advance (at the background) and write the read data to the in file,

```
$ dd if=/dev/ttySTM7 of=in &
```

Send the out file,

```
$ dd of=/dev/ttySTM7 if=out
```

Use the hexdump tool to display the read binary data.

```
$ hexdump -C in
```

```
root@Queclink-GV850:/tmp# hexdump -C in
00000000  f5 b4 14 01 49 30 08 0d  a8 f6                    |....I0....|
0000000a
root@Queclink-GV850:/tmp#
```

The module command/protocol description is detailed in the document "[24-01-03] CAN-Logistic v3 protocol XON-XOFF.pdf".

We have provided a CAN module testing tool to assist with testing. The name of the tool is example_external_can. This tool can send raw frames, receive module data, and query basic information. For detailed information, refer to the "Example of Codes" section.

We recommend using the CANOBD core interface to test the CAN module. For details, refer to the "Canobd" section in the "Queclink Software Module" chapter.

The module provides three configurable GPIO outputs, where OUT2 is connected to PG1 of the CPU and can notify the CPU of events. The testing method is as follows:

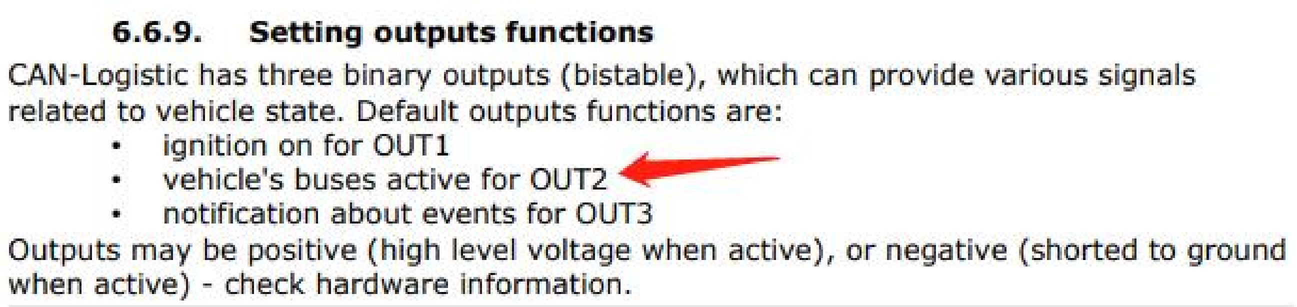
example_input_intr tool can be used to monitor PG1 event, event code is 257, event value 0 indicate CAN module enter sleep, 1 indicate CAN module is wroking.

```
$ example_input_intr
type:1, code:257, value:0
```

type:1, code:257, value:1

If the module has no serial port data and CAN bus data within 60 seconds, the CAN module will enter sleep mode and receive the event type:1, code:257, value:0. The CPU will wake up the CAN module by sending serial port data to it, and will receive the event type:1, code:257, value:1.

The CAN module can serve as a wake-up source for system sleep. On hardware, OUT2 is connected to PG1 of CPU as the wake-up source. The OUT2 function is configurable, with the default function being 'vehicle's buses active',



**6.6.9. Setting outputs functions**

CAN-Logistic has three binary outputs (bistable), which can provide various signals related to vehicle state. Default outputs functions are:

- ignition on for OUT1
- vehicle's buses active for OUT2
- notification about events for OUT3

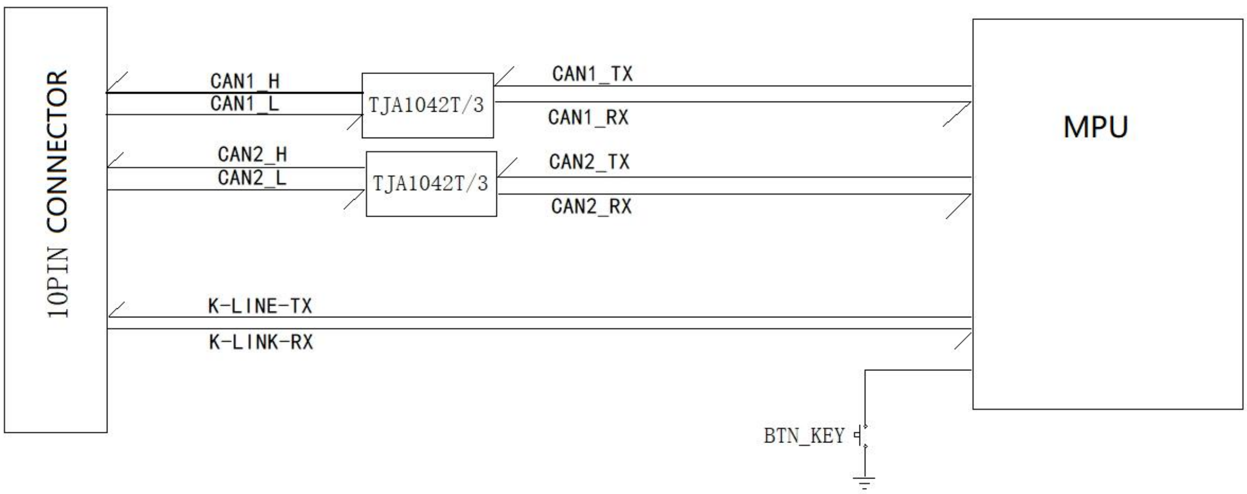
Outputs may be positive (high level voltage when active), or negative (shorted to ground when active) - check hardware information.

The module will enter sleep on its own and pull PG1 up. When the module is awakened, pressing the CAN sync button will pull PG1 down。

The module can be used as a system sleep wake-up source, supporting up to Stop mode. For test methods, refer to the "System Sleep" section.

#### 4.4.2 GV8551 Raw CAN FD

The GV851 device features two CAN FD transceiver channels, replacing the CAN OBD module in the GV850 device. CAN data services are handled by the CPU. The stm32mp133 main controller has two built-in CAN FD controllers. Both CAN modules (FDCAN1 and FDCAN2) comply with ISO 11898-1 (CAN protocol specification version 2.0 part A, B) and the CAN FD protocol specification version 1.0. For details on CAN FD, refer to the official manual of the stm32mp133 ST.

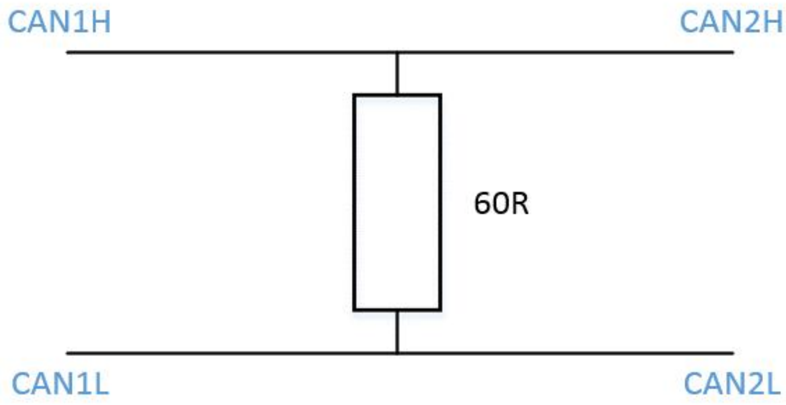



The TX/RX PIN of the CAN FD controller is connected to the CANTJA1042 transceiver, and the IO configuration is as follows:

| Pin Name | Description | Remarks |
|---|---|---|
| PA14 | CAN communication red LED | 0: Off<br>1: On |
| PD2 | CAN communication green LED | 0: Off<br>1: On |
| PE3 | CANFD1 RX | CANFD module pins inside the chip |
| PG10 | CANFD1 TX | CANFD module pins inside the chip |
| PB5 | CANFD2 RX | CANFD module pins inside the chip |
| PB13 | CANFD2 TX | CANFD module pins inside the chip |
| PA4 | Onboard 5V boost enable control pin, CANFD data transmission and reception needs to enable this function | 0: Disable<br>1: Enable |
| PG3 | Onboard 5V boost enable control pin, CANFD data transmission and reception needs to enable this function | 0: Disable power supply<br>1: Enable power supply |
| PG5 | CANFD1 transceiver working state selection pin | 0: CANFD transceiver is working<br>1: CANFD transceiver is sleeping |
| PH13 | CANFD2 transceiver working state selection pin | 0: CANFD transceiver is working<br>1: CANFD transceiver is sleeping |
| PH6 | Working mode switch button | 0: button pressed<br>1: button released |
| PF9(UART8) | K-LINE communication port RX pin | Device node: /dev/ttySTM4 |
| PE1(UART8) | K-LINE communication port TX pin | V1.02 and earlier versions only support RX. Devices with hardware versions greater than this support TX and RX |

| | | |
|---|---|---|
| | | Device node: /dev/ttySTM4 |

For CAN communication test, please connect the two CAN communication ports through a 60 ohm resistor to form a loopback test link as shown in the figure below. The physical wiring is as shown below.




Enable 5V boost (PA4),

```
$ gpioset 0 4=1
```

Enable the CAN transceiver (PG3) power supply,

```
$ gpioset 6 3=1
```

Set the CAN1 transceiver to working state (PG5),

```
$ gpioset 6 5=0
```

Set the CAN2 transceiver to working state (PH13),

```
$ gpioset 7 13=0
```

View the system CAN devices,

```
$ ifconfig -a | grep -C 7 can
can0      Link encap:UNSPEC   HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP   MTU:72   Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:0 (0.0 B)   TX bytes:64 (64.0 B)
          Interrupt:52
```

```
can1      Link encap:UNSPEC    HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          UP RUNNING NOARP    MTU:72    Metric:1
          RX packets:15 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:120 (120.0 B)    TX bytes:0 (0.0 B)
          Interrupt:54
```

Set the CAN1 (device can0) baud rate,

```
$ ip link set can0 type can bitrate 100000 dbitrate 200000 fd on
```

Set the CAN2 (device can1) baud rate,

```
$ ip link set can1 type can bitrate 100000 dbitrate 200000 fd on
```

Enable CAN1,

```
$ ip link set can0 up
```

Enable CAN2.

```
$ ip link set can1 up
```

After the settings are complete, use the can-utils tool installed on the system to test it.

CAN2 uses the candump tool to receive data and runs in the background.

```
$ candump can1 &
```

CAN1 uses the cansend tool to send test data.

```
$ cansend can0 123#1122334455667788
```

The candump tool running in the background will print the data received by CAN2.

```
  can1  123   [8]  11 22 33 44 55 66 77 88
```

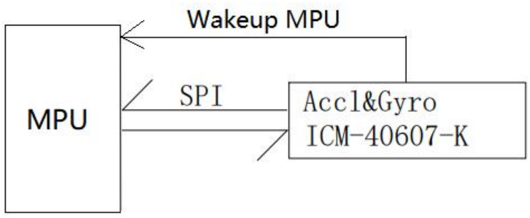
Check CAN1 status information,

```
$ ip -details link show can0
3: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UP mode DEFAULT group default qlen 10
    link/can  promiscuity 0 minmtu 0 maxmtu 0
    can <FD> state ERROR-ACTIVE (berr-counter tx 0 rx 25) restart-ms 0
        bitrate 100000 sample-point 0.875
        tq 41 prop-seg 104 phase-seg1 105 phase-seg2 30 sjw 1 brp 1
        m_can: tseg1 2..256 tseg2 2..128 sjw 1..128 brp 1..512 brp_inc 1
        dbitrate 200000 dsample-point 0.875
        dtq 208 dprop-seg 10 dphase-seg1 10 dphase-seg2 3 dsjw 1 dbrp 5
        m_can: dtseg1 1..32 dtseg2 1..16 dsjw 1..16 dbrp 1..32 dbrp_inc 1
        clock 24000000 numtxqueues 1 numrxqueues 1 gso_max_size 65536 gso_max_segs 65535 parentbus platform parentdev 4400e000.can
```

The GV851 device can use the SocketCAN interface provided by the Linux system for communication. For more information about SocketCAN, see the official website: https://www.kernel.org/doc/html/latest/networking/can.html.

### 4.5. G-sensor

Sensor model: ICM-40607-K, connected through SPI bus. The ICM-40607-K is a 6-axis MEMS MotionTracking device that combines a 3-axis gyroscope and a 3-axis accelerometer. The system provides IIO driver and device node /sys/bus/iio/devices/iio: device2.




硬件资源列表:

| Pin Name | Description | Remarks |
| --- | --- | --- |
| PC13 | G-sensor module wakes up the CPU pin | 0: BLE module message is ready or a Bluetooth event occurs<br>1: BLE module has no message or event |
| SPI4 | G-sensor module and CPU communication port | SPI communication is managed by the II0 system<br>Device node: /sys/bus/iio/devices/iio:device2 |

For current GV850, after power on, the G-sensor is turned on by default.
Use the provided example_gsensor tool for testing, as detailed in the "Example of Codes" section.
Check the IIO driver corresponding to G-sensor.

```
$ lsmod
Module                  Size    Used by       Tainted: G
inv_CPU_iio_spi        16384    0
inv_CPU_iio             73728    2 inv_CPU_iio_spi
```

When the system enters the Standby mode, the SPI state cannot be maintained. After exiting the Standby mode, the SPI needs to be reinitialized. You can reinitialize the SPI by unloading and loading the driver. The method is as follows:

Remove the driver:

```
$ rmmod inv_CPU_iio_spi
[ 1296.267959] inv-CPU-iio-spi spi0.0: inv-CPU-iio module removed.
$ rmmod inv_CPU_iio
```

Reload the driver:

```
$ modprobe inv_CPU_iio
$ modprobe inv_CPU_iio_spi
```

```
[ 1348.975145] inv_CPU: inv_CPU_probe: power on here.
[ 1348.978562] inv_CPU: inv_CPU_probe: power on.
[ 1349.093593] inv_CPU: id name = icm42600
[ 1349.096600] inv_CPU: whoami= dd
[ 1349.219978] inv_CPU: inv_CPU_initialize: initialize result is 0....
[ 1349.230144] inv_CPU: wakeup_source is created successfully
[ 1349.243890] inv-CPU-iio-spi spi0.0: icm42600 ma-kernel-9.3.3-test2 is ready to go!
[ 1349.250175] inv_CPU: Data read from FIFO
```

Enter the driver sysfs directory, and you can see the interface provided by the driver:

```
$ cd /sys/bus/iio/devices/iio:device2
```

Read the value of the IMU on-chip register. Because the on-chip registers are mainly divided into Bank0 and Bank4, they will be printed separately. For detailed meaning of the registers, see "DS-000407 ICM-40607-K v1.0 for Queclink.pdf".

```
$ cat debug_reg_dump
bank 0
0x0: 0x0
0x1: 0x0
0x2: 0x0
...
bank 4
0x40: 0xa2
0x41: 0x85
...
0x46: 0x45
0x47: 0x5b
```

The G-sensor driver provides two interfaces, debug_reg_write_addr and debug_reg_write (the interface parameters are in decimal), which can modify the register values. Because the registers mentioned above are divided into Bank0 and Bank4, you need to switch to the corresponding Bank before modifying the addr register value.

Writing 4 to register 0x76 switches to Bank4.

```
$ echo 118 > debug_reg_write_addr; echo 4 > debug_reg_write
```

Then you can modify the 4Ah (74) register of Bank4 and write 0xC8 (200).

```
$ echo 74 > debug_reg_write_addr; echo 200 > debug_reg_write
```

Similarly, before modifying the 0x57 (87) register of Bank0, you need to switch to Bank0 first.

```
$ echo 118 > debug_reg_write_addr; echo 0 > debug_reg_write
$ echo 87 > debug_reg_write_addr; echo 5 > debug_reg_write
```

The module supports the vibration wake-up (WAKE ON MOTION) function. INT1 is connected to the CPU via GPIO PC13 and can be used as a system sleep wake-up source.

Enable the vibration wake-up function.

```
$ echo 1 > event_motion_detect_enable
[    74.262869] inv_CPU: Motion Detect Enabled
```

However, because the original driver enables UI_DRDY_INT1_EN in bit3 of the INT_SOURCE0 register after enabling vibration wake-up, the interrupt will be triggered continuously and needs to be turned off.

**14.51 INT_SOURCE0**

Name: INT_SOURCE0
Address: 101 (65h)
Serial IF: R/W
Reset value: 0x10
Clock Domain: SCLK_UI

| BIT | NAME | FUNCTION |
|---|---|---|
| 7 | - | Reserved |
| 6 | UI_FSYNC_INT1_EN | 0: UI FSYNC interrupt not routed to INT1<br>1: UI FSYNC interrupt routed to INT1 |
| 5 | PLL_RDY_INT1_EN | 0: PLL ready interrupt not routed to INT1<br>1: PLL ready interrupt routed to INT1 |
| 4 | RESET_DONE_INT1_EN | 0: Reset done interrupt not routed to INT1<br>1: Reset done interrupt routed to INT1 |
| 3 | UI_DRDY_INT1_EN | 0: UI data ready interrupt not routed to INT1<br>1: UI data ready interrupt routed to INT1 |
| 2 | FIFO_THS_INT1_EN | 0: FIFO threshold interrupt not routed to INT1<br>1: FIFO threshold interrupt routed to INT1 |
| 1 | FIFO_FULL_INT1_EN | 0: FIFO full interrupt not routed to INT1<br>1: FIFO full interrupt routed to INT1 |
| 0 | UI_AGC_RDY_INT1_EN | 0: UI AGC ready interrupt not routed to INT1<br>1: UI AGC ready interrupt routed to INT1 |

Only the WOM_*** part of INT_SOURCE1 is kept as the interrupt source.

**14.52 INT_SOURCE1**

Name: INT_SOURCE1
Address: 102 (66h)
Serial IF: R/W
Reset value: 0x00
Clock Domain: SCLK_UI

| BIT | NAME | FUNCTION |
|---|---|---|
| 7:4 | - | Reserved |
| 3 | SMD_INT1_EN | 0: SMD interrupt not routed to INT1<br>1: SMD interrupt routed to INT1 |
| 2 | WOM_Z_INT1_EN | 0: Z-axis WOM interrupt not routed to INT1<br>1: Z-axis WOM interrupt routed to INT1 |
| 1 | WOM_Y_INT1_EN | 0: Y-axis WOM interrupt not routed to INT1<br>1: Y-axis WOM interrupt routed to INT1 |
| 0 | WOM_X_INT1_EN | 0: X-axis WOM interrupt not routed to INT1<br>1: X-axis WOM interrupt routed to INT1 |

Modify the INT_SOURCE0 register to disable unnecessary interrupt sources.

```
$ echo 118 > debug_reg_write_addr; echo 0 > debug_reg_write
$ echo 101 > debug_reg_write_addr; echo 0 > debug_reg_write
```

Use the following method to force the system into Stop mode. For details, see the "System Sleep" section.

```
$ echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
$ echo mem > /sys/power/state
[   145.630453] PM: suspend entry (deep)
[   145.632931] Filesystems sync: 0.000 seconds
[   145.638156] Freezing user space processes ... (elapsed 0.001 seconds) done.
[   145.645593] OOM killer disabled.
[   145.648595] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[   145.656064] printk: Suspending console(s) (use no_console_suspend to debug)
```

After the system enters sleep mode, as long as the vibration device reaches the detection threshold, it will wake up and exit sleep mode to return to the system command line.

Modify the sensitivity of vibration wake-up. The smaller the threshold, the more sensitive it is. You can modify the thresholds of the three axes (4Ah, 4Bh, 4Ch registers) X, Y, and Z separately.

**17.11 ACCEL_WOM_X_THR**

Name: ACCEL_WOM_X_THR
Address: 74 (4Ah)
Serial IF: R/W
Reset value: 0x00
Clock Domain: SCLK_UI

| BIT | NAME | FUNCTION |
|---|---|---|
| 7:0 | WOM_X_TH | Threshold value for the Wake on Motion Interrupt for X-axis accelerometer<br>WoM thresholds are expressed in fixed "mg" independent of the selected Range [0g : 1g]; Resolution 1g/256=~3.9mg |

To modify, first switch to Bank4, and then modify the three-axis registers in sequence.

```
$ echo 118 > debug_reg_write_addr;echo 4 > debug_reg_write
$ echo 74 > debug_reg_write_addr;echo 200 > debug_reg_write
$ echo 75 > debug_reg_write_addr;echo 200 > debug_reg_write
$ echo 76 > debug_reg_write_addr;echo 200 > debug_reg_write
```

Turn off the vibration wake-up feature.

```
$ echo 0 > event_motion_detect_enable
[   518.980117] inv_CPU: Motion Detect Disabled
```

Attached: Complete test instructions for turning on vibration wake-up, using a high sensitivity threshold, which can be triggered by just tapping the device.

```
$ cd /sys/bus/iio/devices/iio:device2
$ echo 1 > event_motion_detect_enable
$ echo 118 > debug_reg_write_addr
$ echo 0 > debug_reg_write
$
$ echo 101 > debug_reg_write_addr
```

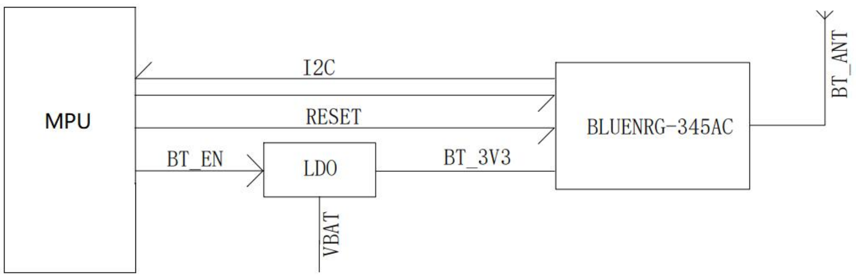
```
$ echo 0 > debug_reg_write
$
$ echo 118 > debug_reg_write_addr
$ echo 4 > debug_reg_write
$
$ echo 74 > debug_reg_write_addr
$ echo 1 > debug_reg_write
$
$ echo 75 > debug_reg_write_addr
$ echo 1 > debug_reg_write
$
$ echo 76 > debug_reg_write_addr
$ echo 1 > debug_reg_write
$ cd -
```

Turn off vibration to wake up.

```
$ cd /sys/bus/iio/devices/iio:device2
$ echo 0 > event_motion_detect_enable
$ cd -
[    669.702742] inv_CPU: Motion Detect Disabled
```

### 4.6. BLE

Module model: BlueNRG-345AC, connected through I2C bus. STM32MP133 platform reads and writes from I2C bus 0 through/dev/i2c-0 device.




Hardware resource list:

| Pin Name | Description | Remarks |
|---|---|---|
| PE15 | BLE module power control pin | 0: Power off<br>1: Power on |
| PG7 | BLE module reset control pin | 0: Normal operation<br>1: Trigger module reset |
| PG4 | BLE module wakes up the CPU pin | 0: BLE module message is ready or event occurs<br>1: BLE module has no message or event occurs |
| PH12 | CPU wakes up the BLE module pin | 0: wake up the BLE module |

| | | |
|---|---|---|
| | | 1: allow the BLE module to enter sleep mode |
| I2C2 | CPU communicates with the BLE module port | Device node: /dev/i2c-0 |

The reference testing commands are as follows:

PE15 power supply enable output high,

```
$ gpioset 4 15=1
```

PG7 is used to reset BLE, set 1 to reset BLE module, set 0 to make BLE module work normal.

```
$ gpioset 6 7=0
```

Scan I2C bus 0,

```
$ i2cdetect -y 0
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- 3e --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

After scanning the slave device on the I2C bus, the device boot message can be read. The BLE module serves as the I2C slave device with address 0xBE and register address 0x01, and reads 220 bytes each time. The command/protocol description is detailed in the document "BLE100 @ Bluetooth Internal Protocol".

```
$ i2ctransfer -y 0 w1@0x3e 0x01 r220
```

The provided example_ble tool can also be used for command testing, as detailed in the "Example of Codes" section. The BLE module is developed by Queclink itself. The command/protocol description is detailed in the document "BLE100 @ Bluetooth Internal Protocol".

The sleep of the BLE module can be controlled, PH12 output high level allows sleep and low level wakes up the module,

```
$ gpioset 7 12=1
```

BLE Module events can be notified to the CPU through the PG4 pin, such as sending the command AT+F=12 to the BLE module, which will wake up the CPU,

```
$ example_ble AT+F=12
recv from BLE:
+ACK:F,12,1,OK
```

When the BLE module has a reply message or reports a message, it will pull down the BLE wake-up CPU pin.Level change events will be monitored on the PG4 pin.We can monitor and view it through the interrupt number of PG4 or /dev/input/event0.

Check the interrupt count of PG4.

```
$ cat /proc/interrupts | grep PG4
74:            6    stm32gpio      4 Edge          Wakeup-PG4
```

Monitor PG4 input events,

```
$ example_input_intr
type:1, code:259, value:0
type:1, code:259, value:1
```

When a Bluetooth module request command occurs, check the above events and counts.

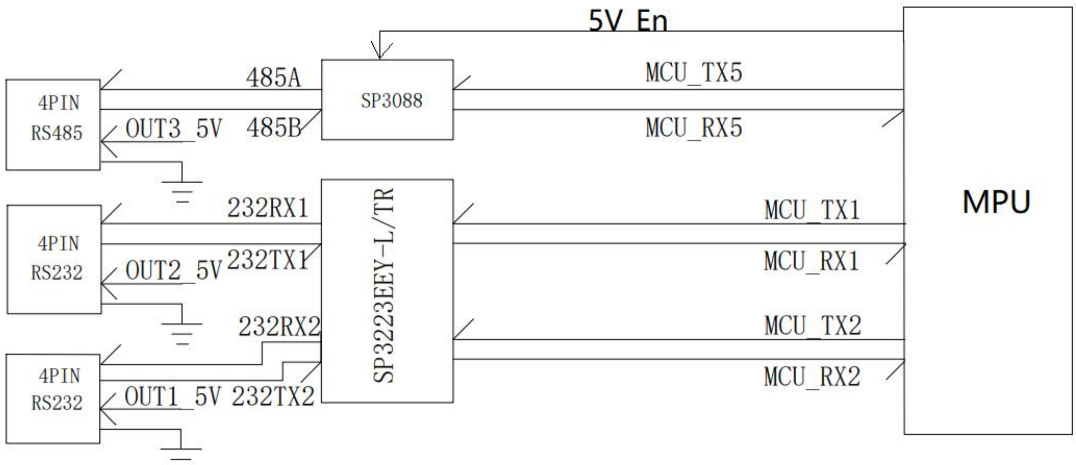
```
$example_ble AT+F=1
```

The BLE module can serve as a wake-up source for system sleep, as detailed in the "System Sleep" section.

## 4.7. RS232/RS485

There are 2 RS232 and 1 RS485.

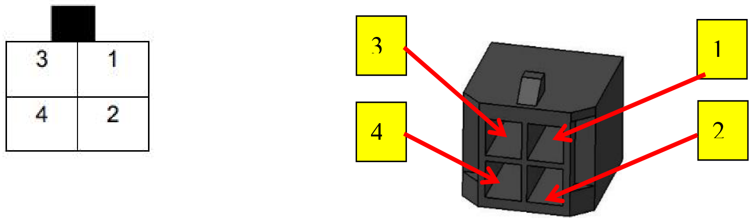
| 4-pin | Hardware | Device | Description |
|---|---|---|---|
| RS485 | USART5 | /dev/ttySTM5 | / |
| RS232_1 | USART1 | /dev/ttySTM1 | / |
| RS232_2 | USART2 | /dev/ttySTM2 | / |




The RS485 port's transceiver switching is automatically controlled by hardware, and no software management is required. When using the RS385 function and DV5_X, you need to turn on the 5V boost enable, and the control pin is GPIO PA4.

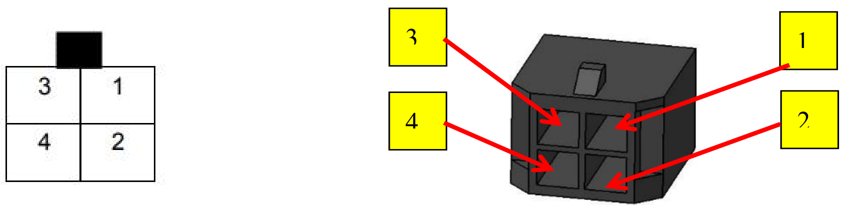
```
$ gpioset 0 4=1
```

The front view of the 4-pin RS485 connector is as follows:



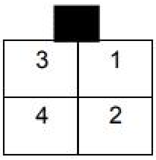

| Pin | Pin Name | Cable Color | Description | Device Nodes | Remarks |
|---|---|---|---|---|---|
| 1 | GND | Black | External Accessory Ground | / | / |
| 2 | DC5V_3 | Red | External Accessory Power 250mA Max | gpiochip6 8(PG8) | 0: Disable output<br>1: Enable output |
| 3 | 485B | Orange white | RS485B | / | / |
| 4 | 485A | Orange black | RS485A | / | / |

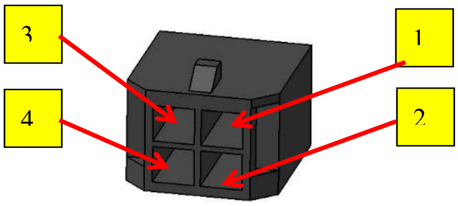
The front view of the 4-pin RS232-1 connectors is as follows:




| Pin | Pin Name | Cable Color | Description | Device Nodes | Remarks |
|---|---|---|---|---|---|
| 1 | GND | Black | External Accessory Ground | / | / |
| 2 | DC5V_1 | Red | External Accessory Power 250mA Max | gpiochip4 12(PE12) | 0: Disable output<br>1: Enable output |
| 3 | TX232_1 | Gray black | UART TXD1 RS232 | / | / |
| 4 | RX232_1 | Gray white | UART RXD1 RS232 | / | / |

The front view of the 4-pin RS232-2 connectors is as follows:




| Pin | Pin Name | Cable Color | Description | Device Nodes | Remarks |
|---|---|---|---|---|---|
| 1 | GND | Black | External Accessory Ground | / | / |
| 2 | DC5V_2 | Red | External Accessory Power 250mA Max | gpiochip4 13(PE13) | 0: Disable output<br>1: Enable output |
| 3 | TX232_2 | Gray black | UART TXD2 RS232 | / | / |
| 4 | RX232_2 | Gray white | UART RXD2 RS232 | / | / |

The test method is as follows, taking RS232_1 as an example,

Set the baud rate,

```
$ stty -F /dev/ttySTM1 ispeed 115200 ospeed 115200 cs8 -icrnl -isig -icanon -echo -echoe
```

Send data,

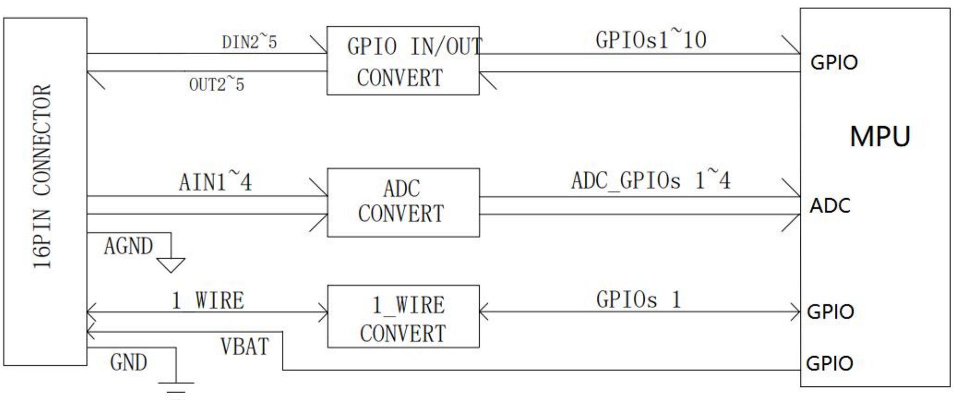
```
$ echo "12345" > /dev/ttySTM1
```

Receive data,

```
$ cat /dev/ttySTM1
```

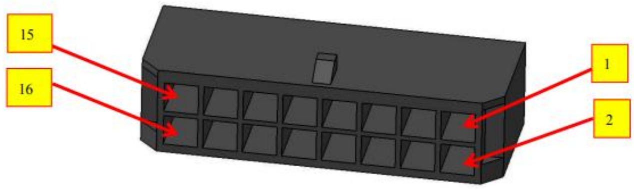
### 4.8. GPIO&ADC&1-WIRE

There are 5 DIN ports and 5 OUT ports ,4 AIN ports 1 and 1-wire bus .DIN is the abbreviation of Negative trigger input.OUT is the abbreviation of Open drain output. AIN is the abbreviation of Analog Input.




The front view of the 16-pin connector is as follows:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 15 | 13 | 11 | 9 | 7 | 5 | 3 | 1 |
| 16 | 14 | 12 | 10 | 8 | 6 | 4 | 2 |



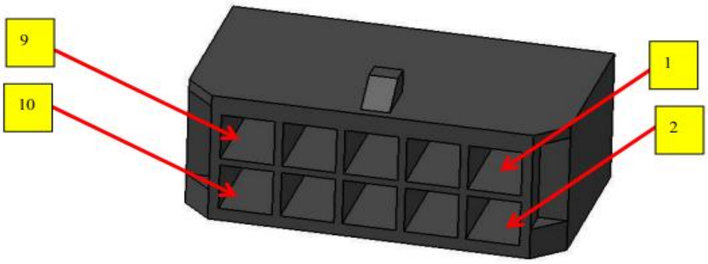

Descriptions of IOs and ADCs are as follows:

| Pin | Pin Name | Cable Color | Description | Device Nodes | Remarks |
|---|---|---|---|---|---|
| 1 | AIN1 | Brown/white | Analog Input1 0~32V | /sys/bus/iio/devices/iio:device1/in_voltage_scale /sys/bus/iio/devices/iio:device1/in_voltage10_raw | Volt=scale*raw*(18+200)/18 Unit: mV |
| 2 | DIN2 | Orange/black | Negative trigger input2 | gpiochip0 3 | $ gpioget gpiochip0 3 |
| 3 | AIN2 | Red/brown | Analog Input2 0~32V | /sys/bus/iio/devices/iio:device1/in_voltage_scale /sys/bus/iio/devices/iio:device1/in_voltage4_raw | Volt=scale*raw*(18+200)/18 Unit: mV |
| 4 | DIN3 | Blue | Negative trigger input3 | gpiochip2 10 | $ gpioget gpiochip2 10 |
| 5 | AIN3 | White/black | Analog Input3 0~32V | /sys/bus/iio/devices/iio:device0/in_voltage_scale /sys/bus/iio/devices/iio:device0/in_voltage2_raw | Volt=scale*raw*(18+200)/18 Unit: mV |
| 6 | DIN4 | Black/brown | Negative trigger input4 | gpiochip2 11 | $ gpioget gpiochip2 11 |
| 7 | AIN4 | Gray/black | Analog Input4 0~32V | /sys/bus/iio/devices/iio:device1/in_voltage_scale /sys/bus/iio/devices/iio:device1/in_voltage0_raw | Volt=scale*raw*(18+200)/18 Unit: mV |
| 8 | DIN5 | Pink | Negative trigger input5 | gpiochip2 12 | $ gpioget gpiochip2 12 |
| 9 | OUT3 | Brown | Open drain output3 | gpiochip8 0 | $ gpioset gpiochip8 0=value |
| 10 | OUT5 | Orange | Open drain output5 | gpiochip0 6 | $ gpioset gpiochip0 6=value |
| 11 | OUT2 | Yellow | Open drain output2 | gpiochip6 15 | $ gpioset gpiochip6 15=value |
| 12 | OUT4 | White | Open drain output4 | gpiochip3 7 | $ gpioset gpiochip3 7=value |
| 13 | 1W_DATA | Green | 1-WIRE data | gpiochip7 2 | / |

| | | | | | |
|---|---|---|---|---|---|
| 14 | GND | Black | Ground | / | / |
| 15 | VDD_1WIRE | Red white | Power for 1-wire devices 3.3V | gpiochip1 8 | / |
| 16 | AGND | Black gray | Analog Ground | / | / |

The front view of the 10-pin connector is as follows:

| | | | | |
|---|---|---|---|---|
| 9 | 7 | 5 | 3 | 1 |
| 10 | 8 | 6 | 4 | 2 |




Descriptions of IOs and ADCs are as follows:

| Pin | Pin Name | Cable Color | Description | Device Nodes | Remarks |
|---|---|---|---|---|---|
| 1 | DCIN | Red | DC Power 8-32V | / | / |
| 2 | GND | Black | Ground | / | / |
| 3 | IGN | White | Positive trigger input | gpiochip8 3 | $ gpioget gpiochip8 3 |
| 4 | DIN1 | Orange | Negative trigger input1 | gpiochip2 8 | $ gpioget gpiochip2 8 |
| 5 | K-LINE | Pink | ISO K Line | / | / |
| 6 | OUT1 | Yellow | Open drain output1 with latch | gpiochip2 9<br>gpiochip1 1 | $ gpioset gpiochip2 9=value<br>$ gpioset gpiochip1 1=0;sleep 0.02; gpioset gpiochip1 1=1;sleep 0.02; gpioset gpiochip1 1=0 |
| 7 | CAN1L | Brown black | CAN Bus CAN1L | / | / |
| 8 | CAN1H | Brown white | CAN Bus CAN1H | / | / |

| | | | | | |
|---|---|---|---|---|---|
| 9 | CAN2L | Blue | CAN Bus CAN2L | / | / |
| 10 | CAN2H | Brown | CAN Bus CAN2H | / | / |

The STM32MP133 platform can use the gpio tools tool to print GPIO group information.

```
$ gpiodetect
gpiochip0 [GPIOA] (16 lines)
gpiochip1 [GPIOB] (16 lines)
gpiochip2 [GPIOC] (16 lines)
gpiochip3 [GPIOD] (16 lines)
gpiochip4 [GPIOE] (16 lines)
gpiochip5 [GPIOF] (16 lines)
gpiochip6 [GPIOG] (16 lines)
gpiochip7 [GPIOH] (15 lines)
gpiochip8 [GPIOI] (8 lines)
```

Check the occupancy of the system GPIO port.

```
$ gpioinfo
gpiochip0 - 16 lines:
        line   0:      "PA0"      kernel   input   active-high [used]
        line   1:      "PA1"      unused   input   active-high
        line   2:      "PA2"      kernel   input   active-high [used]
        line   3:      "PA3"      unused   input   active-high
        line   4:      "PA4"      unused   input   active-high
        line   5:      "PA5"      kernel   input   active-high [used]
        (...)
```

Monitor GPIO level changes. This command will change the GPIO port mode to input mode.

```
$ gpiomon   gpiochipX   line_numbe
Or
$ gpiomon   X   N
```

Set the GPIO level. Setting the GPIO port level will change the GPIO port mode to output mode. You can set the BIAS of the GPIO port through the -B option. Set the drive mode of the GPIO port through the -D option.

```
$ gpioset   gpiochipX   line_numbe=value
Or
$ gpioset   X   N=value
```

```
$ gpioset --help
Usage: gpioset [OPTIONS] <chip name/number> <offset1>=<value1> <offset2>=<value2> ...

Set GPIO line values of a GPIO chip and maintain the state until the process exits

Options:
  -h, --help:        display this message and exit
```

```
   -l, --active-low:    set the line active state to low
   -B, --bias=[as-is|disable|pull-down|pull-up] (defaults to 'as-is'):
            set the line bias
   -D, --drive=[push-pull|open-drain|open-source] (defaults to 'push-pull'):
            set the line drive mode

Biases:
   as-is:    leave bias unchanged
   disable:      disable bias
   pull-up:      enable pull-up
   pull-down:  enable pull-down

Drives:
   push-pull:    drive the line both high and low
   open-drain: drive the line low or go high impedance
   open-source:      drive the line high or go high impedance
```

Get the GPIO level. This command will change the GPIO port mode to input mode.

```
$ gpioget    gpiochipX    line_numbe
Or
$ gpioget    X    N
```

Get GPIO level, this command will not change GPIO port mode. This command is a tool developed by Queclink itself, and you need to install the Quecklink software suite to use it.

```
$ gpiosnoop    gpiochipX    line_numbe
Or
$ gpiosnoop    X    N
```

## 4.9. Watchdog

GV850 adopts an external independent hardware watchdog.

Hardware resource list:

| Pin Name | Description | Remarks |
|---|---|---|
| PI7 | Watchdog enable IO | output high, enable watchdog<br>output low, disable watchdog |
| PG14 | Feed watchdog IO | Flip the level within 1.7s, otherwise a reset will be triggered. |

The software watchdog feed is implemented through a qdog driver and a sysfs interface is provided to enable and disable the watchdog,

```
$ lsmod | grep qdog
```

```
qdog                         16384    0
```

Turn on watchdog and restart the watchdog automatically,

```
$ echo 1 > /proc/qlwatchdog_enabled
```

Turn off watchdog,

```
$ echo 0 > /proc/qlwatchdog_enabled
```

After enabling the watchdog, if you actively stop feeding the watchdog, the system will reset due to the cessation of feeding the watchdog. When testing this feature, you need to remove the universal USB Type-A port of the USB cable, leaving only the USB-to-serial connection. If the universal USB Type-A port is connected to the development computer, the device's watchdog will not reset.

```
$ echo 1 > /proc/qlwatchdog_feed_stop
```

### 4.10. RTC

STM32MP133 has built-in RTC, device/dev/rtc0, and can be set and obtained through the system's built-in hwclock tool. When the system starts, it will be loaded and set as the local time of the system. Reference command:

Query the current system time,

```
$ date
Wed Jan    5 03:19:16 UTC 2000
```

Set the system time to local time

```
$ date -s "2023-09-27 14:26:30"
Wed Sep 27 14:26:30 UTC 2023
```

Set the system time to RTC

```
$ hwclock –w
```

Read time from RTC

```
$ hwclock -r
Wed Sep 27 14:27:12 2023    0.000000 seconds
```

The RTC can serve as a wake-up source for system sleep, as detailed in the "System Sleep" section.

### 4.11. Power&Battery

Main power function and interface description are as follows:

| Function | Device Nodes | Remarks |
|---|---|---|
| Voltage detection | /sys/bus/iio/devices/iio:device1/in_voltage_scale<br>/sys/bus/iio/devices/iio:device1/in_voltage2_raw | Volt=scale*raw*(82+1000)/82 + 800<br>Unit: mV |

Backup battery power function and interface description are as follows:

| Function | Device Nodes | Remarks |
|---|---|---|
| Voltage detection | /sys/bus/iio/devices/iio:device1/in_voltage_scale | Volt=scale*raw*(200+200)/200 |

| | | |
|---|---|---|
| | /sys/bus/iio/devices/iio:device1/in_voltage1_raw | Unit: mV |
| Power supply On<br><br>Power supply Off | gpiochip5 12<br>D flip-flop data pin<br><br>gpiochip7 10<br>D flip-flop clock pin | On<br>$ gpioset gpiochip5 12=1<br>$ gpioset gpiochip7 10=0;sleep 0.02;gpioset gpiochip7 10=1;sleep 0.02;gpioset gpiochip7 10=0<br><br>Off<br>$ gpioset gpiochip5 12=0<br>$ gpioset gpiochip7 10=0;sleep 0.02;gpioset gpiochip7 10=1;sleep 0.02;gpioset gpiochip7 10=0 |
| Charging Start<br>Charging Stop | gpiochip0 11 | Start<br>$ gpioset gpiochip0 11=1<br>Stop<br>$ gpioset gpiochip0 11=0 |
| Charging Status | gpiochip6 12 | $ gpioget gpiochip6 12<br>0, Charging<br>1, Not Charging |
| Charging IC On<br><br>Charging IC Off | gpiochip0 13 | On<br>$ gpioset gpiochip0 13=1<br>Off<br>$ gpioset gpiochip0 13=0<br>The power supply of the ammeter IC is associated with the power supply input of the battery charging management IC. It is necessary to enable it first (backup battery on, charging ammeter IC on), then enable charging (backup battery charging starts), and then detect the charging current |
| Charging Current | /sys/bus/iio/devices/iio:device1/in_voltage_scale<br>/sys/bus/iio/devices/iio:device1/in_voltage16_raw | Current=scale*raw<br>Unit: mA |
| Battery Temperature Detection On<br><br>Battery Temperature Detection Off | gpiochip6 13 | On<br>$ gpioset gpiochip6 13=1<br>Off<br>$ gpioset gpiochip6 13=0 |
| Battery Temperature | /sys/bus/iio/devices/iio:device1/in_voltage_scale<br>/sys/bus/iio/devices/iio:device1/in_voltage15_raw | Volt=scale*raw<br>Unit: mV |

#### 4.11.1. Get Main power voltage

Read the main ADC sampling value,

```
$ cat /sys/bus/iio/devices/iio:device1/in_voltage_scale
0.805664062
$ cat /sys/bus/iio/devices/iio:device1/in_voltage2_raw
1057
```

According to the formula,
Volt = 0.805 * 1057 * (82 + 1000)/82 + 800 = 12027.53 mV

Other tests can be carried out according to the "Obtaining the main power voltage" test method.

#### 4.11.2. Get battery voltage

First enable the battery voltage ADC acquisition,

```
$ gpioset gpiochip6 2=1
```

Turn on the battery power supply switch.

```
$ gpioset gpiochip5 12=1
$ gpioset gpiochip7 10=0;sleep 0.02;gpioset gpiochip7 10=1;sleep 0.02;gpioset gpiochip7 10=0
```

Read the battery ADC sampling value,

```
$ cat /sys/bus/iio/devices/iio:device1/in_voltage_scale
0.805664062
$ cat /sys/bus/iio/devices/iio:device1/in_voltage1_raw
2373
```

calculate,
Volt = 0.806*2373*(200+200)/200 = 3825.276mV， read out value 3825mV。

#### 4.11.3. Battery Charging

Turn on the battery power supply switch.

```
$ gpioset gpiochip5 12=1
$ gpioset gpiochip7 10=0;sleep 0.02;gpioset gpiochip7 10=1;sleep 0.02;gpioset gpiochip7 10=0
```

Turn on the charging current meter IC power supply,

```
$ gpioset gpiochip0 13=1
```

Turn on backup battery charging.

```
$ gpioset gpiochip0 11=1
```

At this time, the battery is in charging state, and its charging state, charging current, and battery temperature can be read.

Check the battery charging status.

```
$ gpioget -B as-is gpiochip6 12
0
```

Get the charging current and read the current ADC sampling value.

```
$ cat /sys/bus/iio/devices/iio:device1/in_voltage_scale
0.805664062
$ cat /sys/bus/iio/devices/iio:device1/in_voltage16_raw
242
```

calculate,

Current = 0.806 * 242 = 195 mA。

Remove external power and check the charging status.

```
$ gpioget -B as-is gpiochip6 12
1
```

Turn off the backup battery switch, and the system will be completely powered off and shut down.

```
$ gpioset gpiochip5 12=0
$ gpioset gpiochip7 10=0;sleep 0.02;gpioset gpiochip7 10=1;sleep 0.02;gpioset gpiochip7 10=0
```

#### 4.11.4. Reading battery temperature

Enable battery temperature reading,

```
$ gpioset gpiochip6 13=1
```

Read the battery temperature ADC acquisition value,

```
$ cat /sys/bus/iio/devices/iio:device1/in_voltage_scale
0.805664062
$ cat /sys/bus/iio/devices/iio:device1/in_voltage15_raw
1963
```

calculate,

Volt = 0.806*1963 = 1582.178mV

Then according to the conversion table below, the temperature is about 25~30℃.

Temperature and voltage relationship conversion table:

| Temperature (℃) | Sampling voltage value (V) |
|---|---|
| -40 | 3.139534748 |
| -35 | 3.091365042 |

| | |
|---|---|
| -30 | 3.032462079 |
| -25 | 2.96174315 |
| -20 | 2.878204686 |
| -15 | 2.78153967 |
| -10 | 2.6715004 |
| -5 | 2.548154561 |
| 0 | 2.413356082 |
| 5 | 2.269426314 |
| 10 | 2.118305522 |
| 15 | 1.96255978 |
| 20 | 1.805502468 |
| 25 | 1.65 |
| 30 | 1.498198198 |
| 35 | 1.352867595 |
| 40 | 1.215877226 |
| 45 | 1.087758933 |
| 50 | 0.969656098 |
| 55 | 0.861876616 |
| 60 | 0.764269249 |
| 65 | 0.678039091 |
| 70 | 0.601275761 |
| 75 | 0.532704403 |
| 80 | 0.471994173 |

After the device enters Standby mode, the GPIO state cannot be maintained. The IO latch circuit can maintain the PF12 state to ensure that the battery power supply state can be maintained after entering Standby mode.

When GPIO PH10 rises, PF12 outputs 1/0 and the status will be latched.

For example, in the following command, PF12 outputs 1 and is latched.

```
$ gpioset 5 12=1;gpioset 7 10=0;sleep 0.0001;gpioset 7 10=1;sleep 0.0001;gpioset 7 10=0
```

For example, in the following command, PF12 outputs 1 and is latched.

### 4.12. Hardware version

The device provides a queryable hardware version. The hardware version is obtained by reading the resistance value of the hardware configuration through ADC.

Hardware resource list:

| Pin Name | Description | Remarks |
|---|---|---|
| PA1 | Hardware version ADC acquisition pin | Device Node:<br>/sys/bus/iio/devices/iio:device1/in_voltage3_raw<br>/sys/bus/iio/devices/iio:device1/in_voltage_scale |

| | | |
|---|---|---|
| | | Voltage calculation method:<br>Volt=scale*raw<br>Unit: mV<br><br>Voltage value range:<br>[1300,1475] corresponds to R1.03<br>[1125,1300] corresponds to R1.04<br>[950,1125] corresponds to R1.05<br>[775, 950] corresponds to R1.06 |

Example:

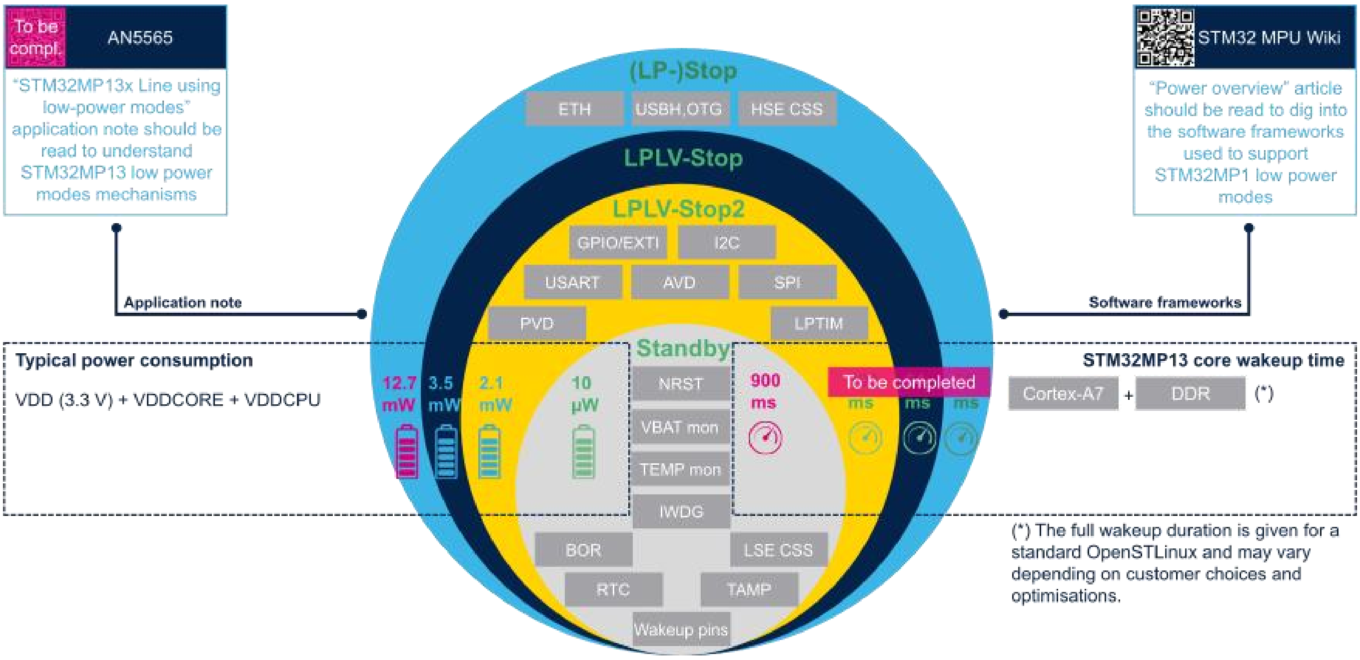
```
$ cat /sys/bus/iio/devices/iio:device1/in_voltage_scale
0.805664062
$ cat /sys/bus/iio/devices/iio:device1/in_voltage3_raw
1500
```

0.805664062 * 1500 = 1208 is in the range of [1125,1300], indicating that the current hardware version is R1.04.

# 5. System Sleep

## 5.1. ST official description

This section introduces the low-power design of the stm32mp133 platform and the control methods for entering low-power. CPU provides multiple energy consumption operation modes,




The wake-up sources supported in each mode are different, as shown in the following table,

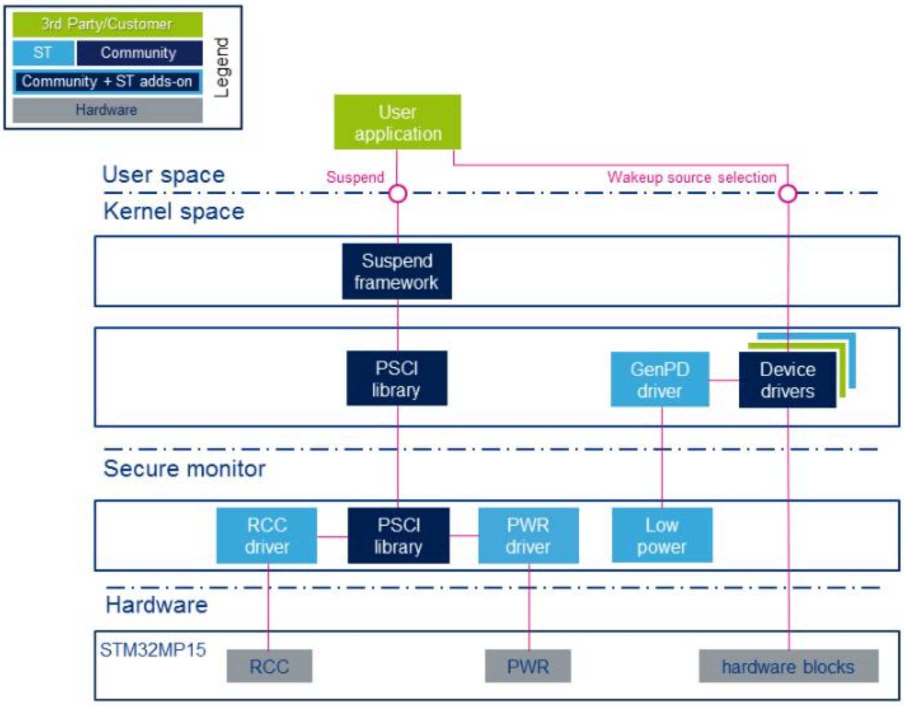
| Platform mode | Available wakeup sources |
|---|---|
| Stop | BOR, PVD, AVD, Vbat mon, Temp mon, HSE CSS, LSE CSS, RTC, TAMP, USBH, OTG, ETH, USART, I2C, SPI, DTS, LPTIM, IWDG, GPIO, Wakeup pins (from PWR) |
| LPLV-Stop<br>LPLV-Stop2 | BOR, PVD, AVD, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, USART, I2C, SPI, DTS, LPTIM, IWDG, GPIO, Wakeup pins (from PWR) |
| Standby | BOR, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, IWDG, Wakeup pins (from PWR) |

That is to say, the low-power mode CPU can enter depends on the wake-up source required by the application scenario,

Table 9. Deepest power mode per wake-up source group and equivalence between Linux and STM32MP13x device system power modes

| Wake-up source | Linux command | STM32MP13x device system deepest power mode | System DDR | Linux kernel state | Power consuming | Wake-up time | Comment/Application guideline |
|---|---|---|---|---|---|---|---|
| **Group 1:** USB, CEC, ETH | "mem" | Stop or LP-Stop | SR (VTT off) | "Suspend-to-ram" | Medium | Medium | LP-Stop: driving external PWR_LP/ PWR_ON permits designing the custom strategy for the external regulator. Typical application is to switch off DDR3 termination supply (VTT) (most likely not needed in 16-bit DDR design) |
| **Group 2:** PVD, AVD, DTS, USART, I2C, SPI, LPTIM, GPIOs | "mem" | LPLV-Stop or LPLV-Stop2 | SR (VTT off) | "Suspend-to-ram" | Low | Medium | LPLV-Stop(2): save power thanks to the power retention. Suitable for applications with aggressive power constraints and tolerant with limitations of wake-up source (refer to Table 4. Low-power mode wake-up capabilities of the system) |
| **Group 3:** BOR, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, wake-up pins | "mem" | Standby | SR | "Suspend-to-ram" | Low | Medium | Standby saves more power at the expense of wake-up time |
| | "shutdown" | Off/VBAT | Off | Shutdown | Very low | High | - |

According to the GV850 specifications and application scenarios, it is required to realize modes LPLV-STOP/LPLV-STOP2 and Off/VBAT. OpenSTLinux implements a power management mechanism, as shown in the following figure,




Only by using the provided Linux sysfs interface, the configuration and enabling/disabling of wake-up sources and initiating of state/mode switchover request can be done. By calling the PWR driver to control the hardware PWR, adjust the VDDCORE and VDDCPU voltages according to the following table. After both voltages meet the conditions, the CPU as a whole can enter the corresponding energy consumption state.

Due to differences in power management hardware between GV850 and the official demo board, GV850 uses separate components instead of power management IC (PMIC), and GPIO is used for PWR control instead of I2C interface. Therefore, GPIO needs to be adapted and adopted in the PWR driver.

Table 8. **STPMIC1x (LP mode) programming: LP-Stop LPLV-Stop and Standby mode**

| Supply name | Control register (LP mode) /@ | LP-Stop | LPLV-Stop | LPLV-Stop2 | Standby with DDR SR | Standby w/o DDR SR |
|---|---|---|---|---|---|---|
| $V_{DDCORE}$ | BUCK4/0x33 | 0x69 (1.25 V) | 0x33 (0.9 V) | 0x33 (0.9 V) | 0x30 (off) | |
| $V_{DDCPU}$ | BUCK1/0x30 | 0x69 (1.25 V) | 0x33 (0.9 V) | 0x30 (off) | | |
| $V_{DD\_DDR}$ | BUCK2/0x31 | 0x79 (1.35 V) | | | | 0x7A (off) |
| $V_{DD}$ | BUCK3/0x32 | 0xD9 (3.3 V) | | | | |
| $V_{REF\_DDR}$ | VREFDDR/0x34 | 0x1 | | | | 0x0 (off) |
| $V_{DDA}$ | LDO1/0x35 | 0x51 (2.9 V) | | | 0x50 (off) | |
| $V_{DD\_USB}$ | LDO4/0x38 | 0x1 (3.3 V) | | | 0x0 (off) | |
| $V_{DD\_SD}$ | LDO5/0x39 | 0x51 (2.9 V) | | | 0x50 (off) | |

### 5.2. Device wakeup source

The mode that the device enters when it goes into sleep mode depends on the instruction to enter sleep mode and the currently enabled wakeup source.
Based on common usage scenarios, GV850 supports multiple wakeup sources in three low-power modes, as listed below:

| Name | GPIO name | Type | Supported sleep modes | Support interrupt detection | Remark |
|---|---|---|---|---|---|
| RTC | - | Inside the CPU | Stop,Standby | - | The default value is enabled. |
| UART | - | interface | Stop | - | The default setting is disabled. You need to enable it first. See below for the method. |
| LTE | GPIO PB7 | External modules | Stop<br>V1.04 hardware version supported. | Yes<br>Supported by V1.04 hardware version. | LTE module RI event. |
| USB | GPIO PI2 | interface | Stop,Standby | - | USB plug/unplug detection, Edge trigger. |
| CAN | GPIO PG1 | External modules | Stop | Yes | Connect CAN OBD OUT2 PIN. |
| BLE | GPIO PG4 | External modules | Stop | Yes | Bluetooth module events. |
| IMU | GPIO PC13 | External modules | Stop,Standby | Yes | SPI INT1 interrupt. |
| IGN | GPIO PI3 | interface | Stop,Standby | Yes | IGN signal input, Edge trigger wakeup. |
| POWER | GPIO PI1 | interface | Stop,Standby | No | External power connection/disconnection detection, Edge trigger wake-up. |
| IO Input | GPIO PA3 | interface | Stop,Standby | Yes<br>Supported by V1.04 hardware version. | DIN2 of 16PIN interface. Interrupt uses GPIO PD9. |
| Button | GPIO PH6 | interface | Stop | Yes | button. |

The current consumption results of the GV850 device in LPLV-Stop2 or Standby mode are shown in the following table, with no other interfaces or peripherals enabled by default:

| Memory type | Memory model | Power consumption: High -> Low | | |
|---|---|---|---|---|
| | | Wake up quickly, the system continues to run | Wake up quickly, the system continues to run | Slow wake-up, system restart |
| | | LPLV-Stop2 | Standby with DDR SR | Standby w/o DDR SR |
| DDR3L | IMD128M16R322J8LY | 12V-2.1mA | 12V-1.4mA | 12V-620uA |
| | | 3.8V-5.6mA | 3.8V-3.6mA | 3.8V-800uA |

The internal battery (3.8V) leakage current is 4uA when the device is turned off.

The STM32MP133 platform function and mode dependency table is as follows:

**Table 35. Functionalities depending on system operating mode(1) (continued)**

| Peripheral | Run | Stop and LP-Stop | | LPLV-Stop LPLV-Stop2 | | Standby | | VBAT |
|---|---|---|---|---|---|---|---|---|
| | | - | Wakeup capability | - | Wakeup capability | - | Wakeup capability | |
| SPIx (x=1:5) | O | O(8) | O(8) | R | O(8) | - | - | - |
| GPIOs | O | R | O(11) | R | O(11) | - | 6 pins (12) | - |

Legend: Y = Yes (Enable). O = Optional (Disable by default. Can be enabled by software). R = data/state retained. - = Not available; highlighted in gray for wakeup mode.

As can be seen from the above table,

**LPLV-Stop2 mode features:**

GPIO and SPI data/status will be maintained during sleep. This means that the GPIO status remains unchanged after entering sleep mode, and the G-sensor of the SPI bus continues to work after waking up;

All GPIOs can be used as interrupt wakeup sources;

**Standby mode features:**

GPIO and SPI data/status cannot be maintained during sleep. This means that after entering sleep, the GPIO status will be restored to the default status.

Only 6 special GPIOs (PA3, PC13, PI1, PI2, PI3, PF8) can be used as interrupt wakeup sources;

Under the current default wakeup source of the device, using mem and poweroff will enter the Standby mode. The DDR SR control effect is as follows:

| command | DDR Power Supply | Awakening state | feature |
|---|---|---|---|
| mem | 1: Keep the power on | The system wakes up and | Fast startup, high power consumption |

| | | | |
|---|---|---|---|
| | | runs directly. | |
| | 0: Do not keep power on | The system wakes up but does not function properly | Abnormal use |
| poweroff | 1: Keep the power on | System wake-up and reboot | Slow startup, system is normal, power consumption is not the lowest |
| | 0: Do not keep power on | System wake-up and reboot | Slow startup, normal system, lowest power consumption |

The normal command combination is the mem command with DDR power supply, and the poweroff command with DDR power off.

As can be seen from the above, Standby has two modes: Standby with DDR SR and Standby w/o DDR SR. The difference between the two is whether the memory DDR keeps powered on and self-refreshed after entering the Standby mode, which will determine the current consumption and the wake-up speed.

GV850 achieves compatibility between Standby and two modes through DDR latch control circuit.

When PE5 rises, PG0 outputs 1, and the power supply to DDR is controlled through NRST.

```
$ gpioset 6 0=1;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
```

After entering Sandby, DDR will maintain power supply and self-refresh, and the system will resume operation after waking up.

```
$ echo mem > /sys/power/state
[    875.043491] PM: suspend entry (deep)
[    875.055564] Filesystems sync: 0.009 seconds
[    875.066293] Freezing user space processes ... (elapsed 0.001 seconds) done.
[    875.073619] OOM killer disabled.
[    875.076780] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[    875.084108] printk: Suspending console(s) (use no_console_suspend to debug)
```

When PE5 rises, PG0 outputs 0 and the power supply to DDR is controlled by PWR_NRST.

```
$ gpioset 6 0=0;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
```

After entering Standby, DDR will stop supplying power and can enter a lower power consumption state. However, the system will be reset after waking up.

```
$ poweroff
Stopping factorytest: start-stop-daemon: warning: killing process 189: No such process
killall: factorytest: no process killed
Failed
Stopping batterymgr: OK
stop
Stopping gpsd: OK
Stopping dropbear sshd: OK
Stopping network: OK
```

```
Stopping bluetoothd: OK
Stopping tee-supplicant: FAIL
Stopping system message bus: done
Saving random seed: OK
stop
Stopping klogd: OK
Stopping syslogd: OK
umount: tmpfs busy - remounted read-only
umount: devtmpfs busy - remounted read-only
[    935.611223] UBIFS (ubi0:3): background thread "ubifs_bgt0_3" stops
The system is going down[    935.621113] watchdog: watchdog0: nowayout prevents watchdog being stopped!
 NOW!
Sent SIGTE[    935.629251] watchdog: watchdog0: watchdog did not stop!
RM to all processes
Sent SIGKILL to all proce[    937.658604] reboot
```

Check the default wakeup sources enabled by the system, which are wakeup0, wakeup1 and wakeup2.

```
$ ls -la /sys/class/wakeup/
lrwxrwxrwx    wakeup0 -> ../../devices/platform/soc/5c004000.rtc/wakeup/wakeup0
Lrwxrwxrwx    wakeup1 -> ../../devices/platform/soc/5c004000.rtc/rtc/rtc0/alarmtimer.0.auto/wakeup/wakeup1
lrwxrwxrwx    wakeup2 -> ../../devices/platform/wakeup/wakeup/wakeup2
```

You can further view the name of each wake-up source.

```
$ cat /sys/class/wakeup/wakeup0/name
5c004000.rtc
```

```
$ cat /sys/class/wakeup/wakeup1/name
alarmtimer.0.auto
```

```
$ cat /sys/class/wakeup/wakeup2/name
wakeup
```

After enabling other wakeup sources, such as UART, you can see the newly added wakeup sources wakeup4 and wakeup5.

```
$ ls -la /sys/class/wakeup/
lrwxrwxrwx    wakeup0 -> ../../devices/platform/soc/5c004000.rtc/wakeup/wakeup0
lrwxrwxrwx    wakeup1 -> ../../devices/platform/soc/5c004000.rtc/rtc/rtc0/alarmtimer.0.auto/wakeup/wakeup1
lrwxrwxrwx    wakeup2 -> ../../devices/platform/wakeup/wakeup/wakeup2
lrwxrwxrwx    wakeup4 -> ../../devices/platform/soc/40010000.serial/tty/ttySTM0/wakeup4
lrwxrwxrwx    wakeup5 -> ../../devices/platform/soc/40010000.serial/wakeup/wakeup5
```

### 5.3. RTC Wake-up

It is enabled by default. By using the rtcwake tool, scheduled wake-up can be completed. The usage method is as follows:

Check the current system time,

```
$ date
Wed Sep 27 14:36:57 UTC 2023
```

Initiate sleep and wake up at 14:39,

```
$ rtcwake -t `date -d 14:39 +%s` -m mem -d /dev/rtc0
wakeup from "mem" at Wed Sep 27 14:38:58 2023
[  825.648590] PM: suspend entry (deep)
[  825.651151] Filesystems sync: 0.000 seconds
[  825.662977] Freezing user space processes ... (elapsed 0.001 seconds) done.
[  825.670398] OOM killer disabled.
[  825.673390] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[  825.680988] printk: Suspending console(s) (use no_console_suspend to debug)
```

Or directly specify the sleep interval,

```
$ rtcwake -s 60 -m mem -d /dev/rtc0
wakeup from "mem" at Sat Jan  1 22:54:38 2000
[   27.544938] PM: suspend entry (deep)
[   27.547565] Filesystems sync: 0.000 seconds
[   27.559313] Freezing user space processes ... (elapsed 0.001 seconds) done.
[   27.566787] OOM killer disabled.
[   27.569771] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[   27.577385] printk: Suspending console(s) (use no_console_suspend to debug)
```

Will wake up after 60 seconds of sleep by itself and return to the system command prompt.

```
[   27.584948] inv-CPU-iio-spi spi0.0: icm42600 suspend
[   27.588715] dwc2 49000000.usb-otg: suspending usb gadget g_ether
[   27.593181] Disabling non-boot CPUs ...
[   27.597019] dwc2 49000000.usb-otg: resuming usb gadget g_ether
[   27.602441] nand: SDR timing mode 4 not acknowledged by the NAND chip
[   27.604035] inv-CPU-iio-spi spi0.0: icm42600 resume
[   27.635996] OOM killer enabled.
[   27.639113] Restarting tasks ... done.
[   27.657140] PM: suspend exit
root@Queclink-GV850:~#
```

Enter Standby w/o DDR SR mode and test RTC wakeup as follows:

Use the rtcwake tool with the parameter –m on to start a 20 second timer. Do not exit the program, otherwise the timer will be turned off, so run it in the background.

```
$ rtcwake -s 20 -m on -d /dev/rtc0 &
```

Use the power off command to initiate low power consumption.

```
$ gpioset 6 0=0;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
$ poweroff
Stopping factorytest: start-stop-daemon: warning: killing process 189: No such process
killall: factorytest: no process killed
Failed
Stopping batterymgr: killall: batterymgr: no process killed
Failed
stop
Stopping gpsd: OK
Stopping dropbear sshd: OK
Stopping network: OK
Stopping bluetoothd: OK
Stopping tee-supplicant: FAIL
Stopping system message bus: done
Saving random seed: OK
stop
Stopping klogd: OK
Stopping syslogd: OK
umount: tmpfs busy - remounted read-only
umount: devtmpfs busy - remounted read-only
[   576.839871] UBIFS (ubi0:3): background thread "ubifs_bgt0_3" stops
The system is going down[   576.849778] watchdog: watchdog0: nowayout prevents watchdog being stopped!
 NOW!
Sent SIGTER[   576.857991] watchdog: watchdog0: watchdog did not stop!
M to all processes
Sent SIGKILL to all proce[   578.886453] reboot
```

After the timeout wake-up, the system restarts and prints the boot log.

```
NOTICE:   CPU: STM32MP133A Rev.Y
NOTICE:   Model: STMicroelectronics custom STM32CubeMX board -
openstlinux-5.15-yocto-kirkstone-mp1-v22.11.23
NOTICE:   BL2: v2.6-stm32mp1-r2.0(release):()
NOTICE:   BL2: Built : 03:57:39, Mar  8 2024
NOTICE:   BL2: Booting BL32
...
```

### 5.4. UART Wake-up

It is disabled by default. Taking the system console UART device ttySTM0 as an example to show the enabling method,

Check the default value,

```
$ cat /sys/devices/platform/soc/40010000.serial/power/wakeup
disabled
```

```
$ cat /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
disabled
```

Modify the wake-up source to enable state,

```
$ echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

Initiate a sleep request,

```
$ echo mem > /sys/power/state
[    192.680917] PM: suspend entry (deep)
[    192.695818] Filesystems sync: 0.012 seconds
[    192.699298] Freezing user space processes ... (elapsed 0.001 seconds) done.
[    192.706747] OOM killer disabled.
[    192.709813] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[    192.717339] printk: Suspending console(s) (use no_console_suspend to debug)
```

During sleep, if there are no other wake-up sources, it will wake up when UART receives data and return to the system command line login.

```
[    192.725008] inv-CPU-iio-spi spi0.0: icm42600 suspend
[    192.728334] dwc2 49000000.usb-otg: suspending usb gadget g_ether
[    192.732793] Disabling non-boot CPUs ...
[    192.736417] dwc2 49000000.usb-otg: resuming usb gadget g_ether
[    192.741961] nand: SDR timing mode 4 not acknowledged by the NAND chip
[    192.743202] inv-CPU-iio-spi spi0.0: icm42600 resume
[    192.775183] OOM killer enabled.
[    192.778435] Restarting tasks ... done.
[    192.783454] PM: suspend exit
^Z
Welcome to Buildroot
Queclink-GV850 login:
```

After the test is completed, restore the wakeup source to the disabled state. The device cannot enter a higher sleep level when the UART wakeup source is enabled.

```
$ echo disabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo disabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

### 6.3 USB Wake-up

Supports Stop and Standby modes. When in low power mode, it will wake up when a USB Type-A port is plugged in and 5V voltage is detected on VBUS.

In Standby with DDR SR mode, the test process is as follows:
Unplug the general USB Type-A port of the cable and keep the USB to serial port connection.

```
$ gpioset 6 0=1;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
```

```
$ echo mem > /sys/power/state
```

At this time, the system enters sleep mode. After plugging the general USB Type-A port into the R&D coCPUter, the device is awakened.

### 6.4 G-Sensor Wake-up

Support Stop and Standby modes. G-sensor devices support multiple detection modes, taking vibration wake-up WOM (Wake On Motion) as an example.

Enable WOM. For detailed meaning of the command, see the "Interface" -> "IMU" section.

```
$ cd /sys/bus/iio/devices/iio:device2;echo 1 > event_motion_detect_enable;echo 118 > debug_reg_write_addr; echo 0 > debug_reg_write;echo 101 > debug_reg_write_addr; echo 0 > debug_reg_write;echo 118 > debug_reg_write_addr;echo 4 > debug_reg_write;echo 74 > debug_reg_write_addr;echo 20 > debug_reg_write;echo 75 > debug_reg_write_addr;echo 20 > debug_reg_write;echo 76 > debug_reg_write_addr;echo 20 > debug_reg_write;cd -
[    731.502424] inv_CPU: Motion Detect Enabled
```

Entering Standby w/o DDR SR sleep

```
$ poweroff
```

Then pick up the device and shake it to wake it up. After the test is completed, you can turn off the WOM function of the G-sensor.

```
$ cd /sys/bus/iio/devices/iio:device2;echo 0 > event_motion_detect_enable;cd -
[    900.148720] inv_CPU: Motion Detect Disabled
```

### 6.5 IGN Wake-up

Support Stop and Standby modes. When entering low power mode, the IGN signal connected to 12V high level will wake up the device. If you want to enter Stop mode, you need to enable some corresponding wake-up sources. The chip decides to enter the corresponding sleep mode according to the level of the wake-up source.

Enable serial terminal wake-up,

```
$ echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

Entering the Stop mode of sleep,

```
$ echo mem > /sys/power/state
```

Manually short the 10-pin connector IGN line to the 12V power supply. The system will be awakened and the operating system will be restored to the state before hibernation.

NRST controls the power supply to DDR and can maintain memory power supply when power is off.

```
$ gpioset 6 0=1;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
```

Enter Standby w/o DDR SR sleep mode,

```
$ echo mem > /sys/power/state
```

Manually short the 10-pin connector IGN line to the 12V power supply. The system will wake up and the device system will restart.

### 6.6 POWER Wake-up

Supports Stop and Standby modes. Use the internal battery as the power source and turn off the external power source. After entering low-power mode, connecting an external power source will wake up the device. When testing this item, you need to unplug the USB Type-A port connecting the device to the development coCPUter and keep the USB to serial port connection. Enter commands through the serial port.

For the Stop sleep mode test command, refer to IGN wake-up.

First, connect the battery to the device. Turn on the battery power enable. Then enter the Standby with DDR SR sleep mode. The test process is as follows.

Open the battery,

```
$ gpioset 5 12=1;gpioset 7 10=0;sleep 0.02;gpioset 7 10=1;sleep 0.02;gpioset 7 10=0
```

NRST controls the power supply to DDR and can maintain memory power supply when power is off

```
$ gpioset 6 0=1;gpioset 4 5=0;sleep 0.0001;gpioset 4 5=1;sleep 0.0001;gpioset 4 5=0
```

Enter Standby with DDR SR sleep mode,

```
$ echo mem > /sys/power/state
```

Manually short the external power supply positive terminal (10-pin connector DCIN line) with the 12V power supply. The system will be awakened and the operating system will be restored to the state before hibernation.

### 6.7 IO Input Wake-up

Supports Stop and Standby modes. The wake-up pin is GPIO PA3. The corresponding terminal is DIN2. For test commands, refer to IGN wake-up. Manually short-circuit the DIN2 terminal line (16-pin connector DIN2 line) with the ground line. At this time, the system will be awakened and the operating system will be restored to the state before hibernation.

### 6.8 Button Wake-up

Support Stop mode. When in low power mode, press the button to wake up.
Enable serial port wakeup, which allows the mem command to put the system into stop mode. The test process is as follows.
Enable the serial port wakeup source, so that the mem command enters the Stop sleep mode.

```
$ echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
```

Go to sleep,

```
$ echo mem > /sys/power/state
```

After entering sleep mode, you can wake up the system by manually pressing a button.

### 6.9 CAN OBD module Wake-up

Supports Stop mode. When entering low power mode, sending J1939 data packets in the CAN BUS1 bus of the CAN module can wake up the CAN module, and the CAN module wakes up the CPU.

The test method is as follows,

First put the CAN OBD module into sleep mode, and then put the CPU system into Stop mode using the following command:

```
$ echo enabled > /sys/devices/platform/soc/40010000.serial/tty/ttySTM0/power/wakeup
$ echo enabled > /sys/devices/platform/soc/40010000.serial/power/wakeup
$ echo mem > /sys/power/state
```

```
[ 1837.571084] PM: suspend entry (deep)
[ 1837.583261] Filesystems sync: 0.009 seconds
[ 1837.586752] Freezing user space processes ... (elapsed 0.001 seconds) done.
[ 1837.594050] OOM killer disabled.
[ 1837.597421] Freezing remaining freezable tasks ... (elapsed 0.001 seconds) done.
[ 1837.604814] printk: Suspending console(s) (use no_console_suspend to debug)
```

When a J1939 data packet is sent in the CAN BUS1 bus of the CAN module and the CAM module wakes up, the GPIO PG1 input level change will wake up the CPU system from Stop mode.

```
[   180.861061] inv-CPU-iio-spi spi0.0: icm42600 suspend
[   180.865177] dwc2 49000000.usb-otg: suspending usb gadget g_ether
[   180.869810] Disabling non-boot CPUs ...
[   180.873631] dwc2 49000000.usb-otg: resuming usb gadget g_ether
[   180.879675] nand: SDR timing mode 4 not acknowledged by the NAND chip
[   180.881261] inv-CPU-iio-spi spi0.0: icm42600 resume
[   180.913179] OOM killer enabled.
[   180.916295] Restarting tasks ... done.
[   180.936626] PM: suspend exit
```

# 6. Example of Codes

In order to facilitate developers to familiarize themselves with and use the modules on the device, example source code for some module interfaces is provided for reference.

## 6.1. utils_info

Query and print product information, such as SN, MCUID and hardware version information.

```
$ utils_info
SN:***
MCUID:313538323532511100270024
HW:HWR103
```

## 6.2. example_modem_at

It demonstrates how to send commands to the LTE module and receive response data. For more information on the module, please refer to the "LTE" section.

The method is as follows, with the main steps being to set baud rate, enable power supply, power on the module, turn off command echo and test command:

```
$ stty -F /dev/ttySTM3 ispeed 115200 ospeed 115200 cs8 -icrnl -isig -icanon -echo -echoe
$ gpioset 0 15=1
$ gpioset 5 5=1
$ sleep 3
$ gpioset 5 5=0
```

Use the tool to send the ATE0 command to turn off echo,

```
$ example_modem_at ATE0
ATE0


OK
```

Send the AT+GMR command to query the firmware version of the LTE module,

```
$ example_modem_at AT+GMR
EG915UEUABR02A05M08


OK
```

### 6.3. example_formula_can

It demonstrates how to send commands to the CAN module and receive response data. For more information on the module, please refer to the "CAN" section.

Since the CANOBD software module is newly developed, it is recommended to use the CANOBD UBUS interface for testing first. Refer to the "Canobd" section in the "Queclink Software Module" chapter.

The method is as follows, with the main steps being to set baud rate, enable power supply and test command:

```
$ stty -F /dev/ttySTM7 ispeed 115200 ospeed 115200 cs8 raw
$ gpioset 6 3=1
$ gpioset 0 4=1
```

Embedded commands inside the tool, parameters (OR values) can be used to control the sequence of the commands to be executed,

```
$ example_external_can
```

```
Usage:
        example_external_can <testing mask>

Testing mask:
       -Raw frame send, [addr cmd D1 .. Dn] no need prefix, suffix and SK,    --0x00
       -Read SN,      --0x01
       -Read version,      --0x02
       -Read boot version,     --0x04
       -Read INPUT_3 voltage,      --0x08
       -Read V_IN voltage,     --0x10
       -Enter develop mode,  --0x20
       -CAN loop test,     --0x40
       -K-Line test,  --0x80
       -GPIO output functions settings,       --0x100
       -IO test O1,O2,O3 activated,      --0x200
       -IO test O2 activated,   --0x400
       -IO test LED green on,        --0x800
       -IO test LED red on,     --0x1000
       -IO test all disactivated,      --0x2000
       -Read THR support flag,      --0x4000
       -Wakeup CAN module ,      --0x8000
```

Execute Read version command,

```
$ example_external_can 0x02
```

```
STEP
Read version, write len=6:
F5 B3 10 01 3B F6
read len=10:
F5 B4 14 01 49 30 11 00 AC F6
```

```
FW Revision:3.017
```

Execute Read version and CAN loop test commands,

```
$ example_external_can 0x42
STEP
Read version, write len=6:
F5 B3 10 01 3B F6
read len=10:
F5 B4 14 01 49 30 11 00 AC F6
FW Revision:3.017

STEP
CAN loop test, write len=9:
F5 B3 43 02 00 80 10 77 F6
read len=8:
F5 B4 22 3A CC F2 31 F6
```

Loop to read the serial port data of the CAN module,

```
$example_external_can 0x0
Listen can communication interface...
select timeout 10.0 sec
```

Send the original data frame of CAN module,

```
$ example_external_can 0x0 0x3b 0x10 0x01
STEP
Raw frame send, [addr cmd D1 .. Dn] no need prefix, suffix and SK, write len=6:
F5 3B 10 01 B3 F6
read len=8:
B4 14 01 49 30 11 00 AC
```

### 6.4. example_gsensor

Demonstrates how to provide a sysfs interface through the driver to complete the initialization, data collection, and command testing of the IMU device:

```
$ example_gsensor
Usage:
        test-sensors-sysfs [-d <device_no>] [-a <rate>] [-g <rate>] [-c]

Options:
      -h, --help
                  Show this help and quit.
      -d, --device
                  Choose device by numero.
      -a, --accel
```

```
                Turn accelerometer on with ODR (Hz).
     -g, --gyro
                Turn gyroscope on with ODR (Hz).
     -c, --convert
                Show data after unit conversion (m/s^2, rad/s)
     -b, --batch
                Set batch timeout in ms.
Version:
     1.1.0
```

For example, the sampling frequency is 100Hz,

```
$ example_gsensor -d 2 -a 100 -g 100
…
Accel body (LSB)  ,  +113,  +13,  +4077,   17478588377202,  176.818,  1.128
Gyro  body (LSB)  ,    -4,   +5,     +1,   17478588342172,   16.954,  1.163
Accel body (LSB)  ,  +114,  +13,  +4082,   17478598377202,   10.000,  1.338
Gyro  body (LSB)  ,    -3,   +5,     +1,   17478598307142,    9.965,  1.408
Accel body (LSB)  ,  +111,   +8,  +4084,   17478608377202,   10.000,  1.176
Gyro  body (LSB)  ,    -5,   +6,     +1,   17478608272112,    9.965,  1.281
Accel body (LSB)  ,  +111,  +12,  +4094,   17478618377202,   10.000,  1.226
Gyro  body (LSB)  ,    -4,   +4,     +1,   17478618237082,    9.965,  1.366
Accel body (LSB)  ,  +114,  +14,  +4092,   17478628377202,   10.000,  1.109
Gyro  body (LSB)  ,    -4,   +5,     +0,   17478628202052,    9.965,  1.284
```

Sampling results after unit conversion,

```
$ example_gsensor -d 2 -a 100 -g 100 -c
…
Accel body (m/s^2),  +0.270545,  +0.021548,  +9.761160,   17568890862843,  169.503,  1.343
Gyro  body (rad/s),  -0.005326,  +0.005326,  +0.001065,   17568890827813,    9.549,  1.378
Accel body (m/s^2),  +0.270545,  +0.028730,  +9.806650,   17568900862843,   10.000,  1.376
Gyro  body (rad/s),  -0.004261,  +0.005326,  +0.001065,   17568900792783,    9.965,  1.446
Accel body (m/s^2),  +0.268151,  +0.021548,  +9.782708,   17568910862843,   10.000,  1.426
Gyro  body (rad/s),  -0.005326,  +0.005326,  +0.000000,   17568910757753,    9.965,  1.531
Accel body (m/s^2),  +0.265756,  +0.023942,  +9.787497,   17568920862843,   10.000,  1.423
Gyro  body (rad/s),  -0.005326,  +0.006392,  +0.000000,   17568920722723,    9.965,  1.563
```

## 6.5. example_ble

It demonstrates how to send commands to the BLE module and receive response data. For more information on the module, please refer to the "BLE" section.

The method is as follows, with the main steps being to enable power supply and test command:

```
$ gpioset 4 15=1
```

The example_ble tool help information is as follows.

```
$ example_ble    -h
Usage:
        ./example_ble <BLE command string>
        -r Loop read BLE message.
        -t BLE cmd terminal.
Example:
        ./example_ble AT+X=10,0
        ./example_ble -r
        ./example_ble -t
```

Use the tool to send the AT+F=1 command to read the BLE firmware version.

```
$ example_ble AT+F=1
Send    Commond: AT+F=1
Recv Response: +ACK:F,1,01.01,OK
```

Query the BOOT APP version of the BLE module,

```
$ example_ble AT+F=17,0
Send    Commond: AT+F=17,0
Recv Response: +ACK:F,17,0,GV850_BT_BOOTR00A01V01,OK
```

```
$ example_ble AT+F=17,1
Send    Commond: AT+F=17,1
Recv Response: +ACK:F,17,1,GV850_BT_R00A01V01,OK
```

For efficient testing, example_ble can support AT command line mode. In this mode, you can input continuously and get the returned results. Ctrl+Backspace can delete the input content, and Ctrl+C can exit the program.

This mode can be used to monitor Bluetooth active reporting events or transparent transmission messages.When using a serial terminal connection, if the Enter key is displayed as ^M and the carriage return is not reached, press the Ctrl+Enter key combination.

```
example_ble -t
at+f=17,1
+ACK:F,17,1,GV850_BT_R00A01V01,OK
at+f=17,0
+ACK:F,17,0,GV850_BT_BOOTR00A01V01,OK
at+f+18      Error input, After entering a newline, enter again.

at+f=18
+ACK:F,18,1,49A7101029DF,7805413D60E0,OK

at+x=10,0
+ACK:X,10,0,1,OK    Query the connect status.
```

### 6.6. example_input_intr

Demonstrates how to perform interrupt detection. GPIO PH6, PI3, PD9, PG4, PG1 are registered as Input devices through the gpio-keys driver, and the level change events of these pins can be received through /dev/input/event0.

```
$ cat /proc/interrupts |grep stm32gpio
 68:          0   stm32gpio    1 Edge      Wakeup-PG1
 69:          0   stm32gpio    3 Edge      Wakeup-PI3
 70:          0   stm32gpio    4 Edge      Wakeup-PG4
 71:          0   stm32gpio    6 Edge      Wakeup-PH6
 72:          5   stm32gpio    7 Edge      Wakeup-PB7
 73:          0   stm32gpio    9 Edge      Wakeup-PD9
 ...
```

Start the detection program and read the /dev/input/event0 node. When the above interrupt occurs, a key input event will be generated.

```
$ example_input_intr
```

When the button is pressed/released, the following will be printed, and the corresponding pin is PH6.

```
type:1, code:260, value:1
type:1, code:260, value:0
```

When the IGN signal is ON/OFF, the following will be printed, and the corresponding pin is PI3.

```
type:1, code:258, value:1
type:1, code:258, value:0
```

When DIN2 is triggered/contacted at a low level, the following will be printed, and the corresponding pin is PD9.

```
type:1, code:262, value:1
type:1, code:262, value:0
```

When the Bluetooth module performs interaction, the following printout will be displayed, and the corresponding pin is PG4.

```
type:1, code:259, value:0
type:1, code:259, value:1
```

When the CAN module enters sleep mode and exits sleep mode, the following information will be printed, and the corresponding pin is PG1.

```
type:1, code:257, value:0
type:1, code:257, value:1
```

### 6.7. gpiosnoop

This tool is used to get the output value of the GPIO port in the output state. For example, after setting the value of

GPIO PA4, read the value set by PA4 without changing the output mode of PA4. Unlike gpioget, it does not change the input and output direction of the GPIO port.

```
$ gpiosnoop 0 4
0
```

or

```
$ gpiosnoop gpiocchip0 4
0
```

# 7. Queclink Software Modules Queclink Software Modules

In order to speed up the development progress of developers on this hardware platform and reduce the development difficulty, Queclink will gradually complete the basic, public code development work. This part of the work mainly includes the maintenance of the compilation environment, C language coding standards, C language basic library usage standards, device storage and directory planning, basic C tool library, complete log library, device system, basic service module, hardware module business encapsulation library, etc.

Our goal is to build a standardized, stable, reasonable, and sustainable Liunx embedded development platform. This goal is being continuously promoted, and service modules for some modules are currently provided.
In order to accelerate the development progress of developers on this hardware platform and reduce the development difficulty. Queclink will gradually complete the basic, public code development work. This part of the work mainly includes the maintenance of the compilation environment, C language coding standards, C language basic library usage standards, device storage and directory planning, basic C tool library, complete log library, device system, basic service module, hardware module business encapsulation library, etc.

Our goal is to build a standardized, stable, reasonable, and sustainable Liunx embedded development platform. This goal is being continuously promoted, and service modules for some modules are currently provided.

## 7.1. Canobd

### 7.1.1. Introduction to CAN Module

The GV850 series products have a powerful CAN OBD module to support vehicle-mounted J1939/J1708/FMS/OBD CAN services. Installing the device in the vehicle and connecting it to the vehicle CAN bus can obtain real-time vehicle operation data, actuator status, fault information, driver information, vehicle behavior statistics, and driver behavior statistics. This information can be further processed to obtain more use value.

The CAN module supports obtaining data from Tachograph and downloading driving record files. The CAN module supports the KLine hardware interface and KLine protocol.

The biggest advantage of this CAN module is that it supports a wide range of vehicle models, almost covering the mainstream models on the market. It not only supports passenger cars but also heavy vehicles such as buses and trucks. Secondly, the protocol design of this module is more suitable for expansion and updating. It is a very excellent vehicle-mounted CAN bus module.

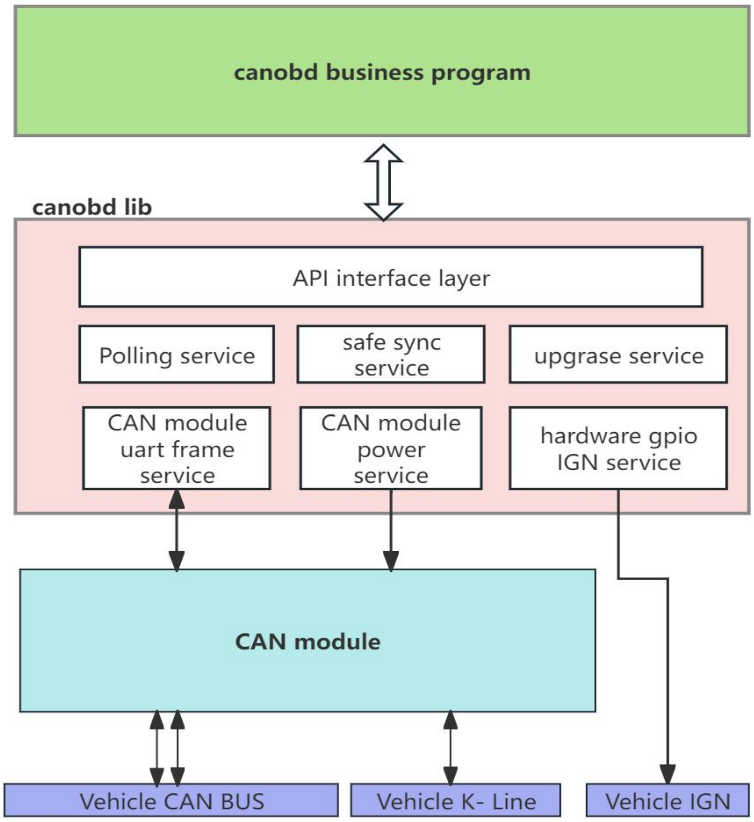
In order to facilitate developers to use the module more deeply, Queclink provides the module's operation library and basic service module. They are canobd support library and canobd business software. The canobd support library and business software are written in C code. The canobd support library is provided in the form of a dynamic library. The canobd business code provides a UBUS bus call interface to the outside world. The overall structure diagram of canobd is shown in the figure below.

The GV850 series products have a powerful CAN OBD module to support vehicle-mounted J1939/J1708/FMS/OBD CAN services. Installing the device in the car and connecting it to the vehicle CAN bus can obtain real-time vehicle operation data, actuator status, fault information, driver information, vehicle behavior statistics and driver behavior statistics. This information can be further processed to obtain more use value.

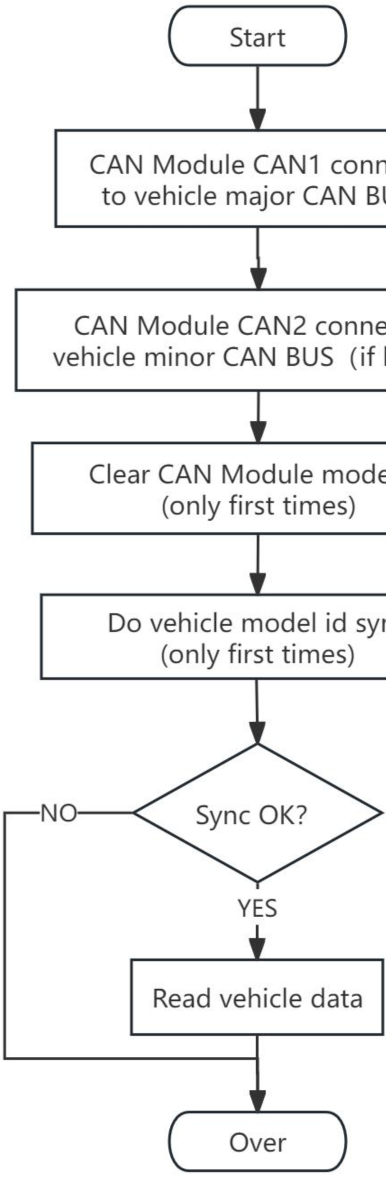
The CAN module supports obtaining data from Tachograph and downloading driving record files. The CAN module supports KLine hardware interface and KLine protocol.

The biggest advantage of this CAN module is that it supports a large number of models, almost covering the mainstream models on the market. It not only supports passenger cars but also heavy vehicles such as buses and trucks. Secondly, the protocol design of this module is more suitable for expansion and updating. It is a very excellent vehicle-mounted CAN bus module.

In order to facilitate developers to use the module more deeply, Queclink provides the module's operation library and basic service module. They are canobd support library and canobd business software respectively. The canobd support library and business software are written in C code. The canobd support library is provided in the form of a dynamic library. The canobd business code provides a UBUS bus call interface to the outside world. The overall structure diagram of canobd is shown in the figure below.




The basic usage process of the CAN module is shown in the figure below.
The basic usage process of the CAN module is shown in the figure below.




#### 7.1.2. CAN module automotive parameter table

The CAN module supports a lot of vehicle parameters. For details, refer to the "[24-01-03] CAN-Logistic v3 protocol XON-XOFF.pdf" manual "Inquiries about car's parameters from the CAN-bus" section. The following lists canobd Supports vehicle parameters that are already supported by the library.

The CAN module supports a lot of vehicle parameters. For details, refer to the "[24-01-03] CAN-Logistic v3 protocol XON-XOFF.pdf" manual "Inquiries about car's parameters from the CAN-bus" section. The The following lists the vehicle parameters that the canobd support library already supports.

**Car parameter table 1:**

**Car parameter table 1:**

| Parameter name | describe |
|---|---|

| | |
|---|---|
| ignition_key | Ignition status |
| total_distance_unit | Total mileage unit |
| total_distance | Total mileage |
| total_fuel_used | Total fuel consumption |
| fuel_level_in_liters | Fuel level (liters) |
| fuel_level_in_percents | Fuel remaining (percentage) |
| range | Remaining mileage |
| vehicle_speed | Car speed |
| engine_speed | Engine speed |
| accelerator_pedal_pressure | Accelerator pedal pressure |
| brake_pedal_pressure | Brake pedal pressure |
| engine_coolant_temperature | Engine coolant temperature |
| total_engine_hours | Total engine hours |
| total_driving_time | Total driving time |
| total_engine_idle_time | Total engine idling time |
| total_idle_fuel_used | Total idle fuel usage |
| axle_weight | Axle load |
| tachograph_information | Speed recorder information |
| detailed_information | Vehicle details |
| lights | Light status |
| doors | Door status |
| rapid_brakings | Emergency braking times |
| rapid_accelerations | Rapid acceleration times |
| total_vehicle_overspeed_time | Total time of vehicle speeding |
| total_vehicle_engine_overspeed_time | Total time the vehicle's engine is overspeeding |
| sw_version | reserve |
| battery_level_in_percents | Electric vehicle battery charge percentage |
| gaseous_fuel | Gas remaining |
| battery_charging_status | Electric vehicle battery charge status |
| tacho_all_infor | Tachometer information |
| drivetrain_related_info | Transmission system related information |
| battery_voltage | Electric vehicle battery voltage |
| battery_charging_cycles | Electric vehicle battery charging time |
| total_energry_recuperated | Tram Total Power Recovery |
| battery_temperature | Electric vehicle battery temperature |
| battery_charging_current | Electric vehicle battery charging current |
| battery_power | Electric vehicle battery charge |
| battery_soh | Electric vehicle battery health status |
| total_energy_used | Total energy consumption of trams |
| total_energy_used_when_idling | Total energy consumption of electric vehicles at idling |
| total_energy_charged | Total battery charge |
| tacho_rtc | RTC time of the tachometer |

**Car parameter table 2:**

| Parameter name | describe |
|---|---|
| adblue_level | Catalyst fluid volume |
| axle_weight_1st | Axis 1 load |
| axle_weight_3rd | Axis 3 load |
| axle_weight_4th | Axis 4 load |
| current_fuel_consumption | Current fuel consumption |
| current_fuel_consumption_unit_is_l_per_h | Current fuel consumption unit L/h |
| tachograph_overspeed_indicator | Driving Recorder Speeding Indicator |
| tachograph_vehicle_moving_indicator | Driving Recorder Vehicle Driving Indicator |
| drive_direction_from_tachograph | Driving recorder displays driving direction |
| input3 | Input3 analog input signal |
| engine_braking_factor | Engine braking deceleration times |
| pedal_braking_factor | Pedal brake system deceleration times |
| total_accelerator_kick_downs | Total number of downshifts during acceleration |
| total_effective_engine_speed_time | Total effective engine speed time |
| total_cruise_control_time | Total cruise control time |
| total_accelerator_kick_down_time | Total time of acceleration and resistance |
| total_brake_applies | Total brake application |
| engine_torque | Engine torque |
| outair_tmeperature | Outdoor air temperature |
| diagnastic_trouble_codes | Diagnostic Trouble Codes |
| diagnatic_trouble_codes_format | Diagnostic trouble code format |
| retarder_selection | Reducer gear selection |

**Car parameter table 3:**

| Parameter name | describe |
|---|---|
| th_driver1_card_number | Driver Card 1 Number |
| th_driver2_card_number | Driver Card 2 Number |
| th_driver1_name | Driver 1 Name |
| th_driver2_name | Driver 2 Name |
| vin | Vehicle VIN number |
| registration_number | License plate number |
| service_distance | Distance of service |
| cold_engine_start_counts | Engine Cold Start Count |
| engine_all_start_counts | Engine all start counts |
| engine_start_by_ign_counts | by IGN Launch Engine |
| driving_time_with_cold_engine | Driving time with a cold engine |
| handbrake_applies_on_ride | the handbrake was used |

**Driver Information Card Form:**

| Parameter name | describe |
|---|---|
| End_Of_Last_Daily_Rest_Period | End of the last break of the day |
| End_Of_Last_Weekly_Rest_Period | The last week of break is over |
| End_Of_Second_Last_Weekly_Rest_Period | The second break of the last week is over |
| Maximum_Daily_Period | Maximum daily cycle |
| Number_Of_Times_9h_Daily_Driving_Times_Exceed | Number of times you drive more than 9 hours per day |
| Number_Of_Used_Reduced_Daily_Rest_Period | Use reduces the number of breaks per day |
| Reaining_Current_Drive_Time | Remaining driving time |
| Reasoning_Time_Until_Next_Break_Or_Rest | Time remaining before next break or rest |
| Duration_Of_Next_Break_Rest | Next break time |
| Reasoning_Time_Of_Current_Break_Rest | Next break time |
| Time_Left_Until_Next_Driving_Period | Time remaining until next driving period |
| Duration_Of_Next_Driving_Period | Duration of next driving session |
| Reasoning_Driving_Time_On_Current_Shift | Remaining driving time for the current shift |
| Time_Left_Until_New_Daily_Rest_Period | The rest of the day until the new daily break time |
| Minimum_Daily_Rest | Minimum daily rest time |
| Remaining_Driving_Time_Of_Current_Week | Remaining driving time this week |
| Time_Left_Until_New_Weekly_Rest_Period | Time remaining until the new weekly break |
| Minmum_Weekly_Rest | Minimum weekly rest time |
| Open_Compensation_In_The_Last_Week | Public deductions in the last week |
| Open_Compensation_In_Weekly_Before_Last | Public deductions for the previous week |
| Open_Compensation_In_2nd_Week_Before_Last | Deduction for the first two weeks |
| Continuous_Driving_Time | Additional information (coming soon) |
| Cumulative_Break_Time | Continuous driving time |
| Current_Duration_Of_Select_Activity | Cumulative break time |
| Accumulated_Driving_Time_Privious_And_Current_Week | The current duration of the selected activity |
| Current_Daily_Driving_Time | Total driving time in the previous week and the current week |
| Current_Weekly_Driving_Time | Current daily driving time |
| Cumulative_Uninterrupted_Rest_Time | Current weekly driving time |
| Maximum_Daily_Driving_Time | Cumulative uninterrupted rest time |

### 7.1.3. CANOBD Core Interface

CANOBD business service calls the C API function of canobd support library and provides UBUS bus calling method. The core interface is listed here for quick understanding. If you need more abundant interfaces, please contact Quecklink.

CANOBD business service calls the C API function of canobd support library and provides UBUS bus calling method. The core interface is listed here for quick understanding. If you need more abundant interfaces, please contact Quecklink.

#### 7.1.3.1. Get API version

This interface is used to obtain version library information, the interface is get_canobd_api_version , and the payload is empty.

This interface is used to obtain version library information, the interface is get_canobd_api_version, and the payload is empty.

Example:

Example:

```
$ ubus call canobd get_canobd_api_version
```

#### 7.1.3.2. Raw frame channel

This interface encapsulates the raw frame into a protocol and then transparently transmits it to the CAN module, and then returns the frame returned by the CAN module. This interface is used for testing and special application scenarios. Interface action_canobd_raw_frame_send , payload {"frame":[ String array ]} . The parameters are as shown in the following table.

This interface encapsulates the raw frame into a protocol and then transparently transmits it to the CAN module, and then returns the frame returned by the CAN module. This interface is used for testing and special application scenarios. Interface action_canobd_raw_frame_send, payload {"frame": [String array]}. Parameters are as follows.

| name | type | Remark |
|---|---|---|
| frame | string array | The request frame is organized according to the CAN module frame protocol , starting from the address bit, excluding the check bit, and no escape is required.<br>The request frame is organized according to the CAN module frame protocol, starting from the address bit, excluding the check bit, and no escape is required. |

Example:

Example:

```
$ ubus call canobd action_canobd_raw_frame_send '{"frame":["0xb3", "0x20","0xd7"]}'
```

#### 7.1.3.3. Query module status

This interface is used to obtain all status information of the CAN module and canobd support library. Interface get_state , the payload is empty. When carrying {"scope": "String"} payload, the corresponding status can be obtained separately. The request payload field list is as follows.

This interface is used to obtain all status information of the CAN module and canobd support library. Interface get_state, the payload is empty. When carrying {"scope": "String"} payload, the corresponding status can be obtained separately. The request payload field list is as follows.

| name | type | Remark |
|---|---|---|
| scope | string | sync_state: Get real-time model synchronization status<br>sync_history_state: Get the established model synchronization state<br>base_init_info_state: Get the basic configuration and status information of the module<br>upgrade_state: Get the upgrade status |

| | | |
|---|---|---|
| | | fireware_state: Get firmware status<br>module_conf_state: Get module configuration state<br>car_params_poll_state: Get vehicle parameter polling state<br>car_base_state: Get vehicle basic information and status<br><br>sync_state: Get real-time model synchronization status<br>sync_history_state: Get established model synchronization status<br>base_init_info_state: Get basic configuration and status information of the module<br>upgrade_state: Get upgrade status<br>fireware_state: Get firmware status<br>module_conf_state: Get module configuration status<br>car_params_poll_state: Get vehicle parameter polling status<br>car_base_state: Get vehicle basic information and status |

Example:

```
$ ubus call canobd get_state all                                    states
$ ubus call canobd get_state '{"scope": "sync_state"}'               synchronization state
$ ubus call canobd get_state '{"scope": "sync_history_state"}'    history synchronization state
$ ubus call canobd get_state '{"scope": "base_init_info_state"}' basic information state
$ ubus call canobd get_state '{"scope": "upgrade_state"}'         upgrade state
$ ubus call canobd get_state '{"scope": "fireware_state"}'          firmware version state
$ ubus call canobd get_state '{"scope": "module_conf_state"}'    configuration version state
$ ubus call canobd get_state '{"scope": "car_params_poll_state"}' vehicle parameter polling state
$ ubus call canobd get_state '{"scope": "car_base_state"}'          Vehicle and bus state
```


```
{
	"device_mode": "CANHEX_MODE_OPERATING",
	"mode_reason0": "CANHEX_MODE_REASON_NONE",
	"mode_reason1": "CANHEX_MODE_REASON_NONE",
	"serial_mode": "CANHEX_MODE_SERIAL_XONXOFF",
	"protocol_work_mode": "CANHEX_MODE_WORK_WORKING",
	"new_firmware_valid": 0,
	"new_conf_valid": 0,
	"current_conf_valid": 1,
	"serial_conf_valid": 1,
	"model_id": "004F",
	"model_id_state": "SET_BY_SYNC",
	"car_name": "Seat Altea (04-)",
	"thr_support": 0,
	"sn": "3863285",
	"event_ack_enable": 0,
	"canbus_active_mode": 0,
	"fuel_measure_delay": 0,
```

```
    "nonautomatic_sync": 0,
    "can1_send_disable": 0,
    "uart_tx_wakeup": 0,
    "short_wakeup": 0,
    "low_power_mode": 1,
    "ignore_diagnostic_tools": 0,
    "deep_sleep_mode_disable": 0,
    "thr_compatibility": 0,
    "can2_send_disable": 0 ,
    "query_tachograph_mode": 0,
    "scope": "base_init_info_state",
    "from": "cache",
    "rtn": "success"
}
```


Reply load parameter table:

| name<br>name | type<br>type | illustrate<br>Remark |
|---|---|---|
| device_mode | string | CANHEX_MODE_OPERATING: Operation mode<br>CANHEX_MODE_TEST: Test mode<br>CANHEX_MODE_CAN_BUS_SYNC_PROCEED: CAN BUS synchronization processing status<br>CANHEX_MODE_DIAG_START: Diagnosis starts<br>CANHEX_MODE_DIAG_END: Diagnosis ends<br>CANHEX_MODE_CAN_BUS_SYNC_FAILED: CAN BUS synchronization failed<br>CANHEX_MODE_FAIL_SAFE: Fail-safe mode<br><br>CANHEX_MODE_OPERATING: Operation mode<br>CANHEX_MODE_TEST: Test mode<br>CANHEX_MODE_CAN_BUS_SYNC_PROCEED: CAN BUS synchronization processing status<br>CANHEX_MODE_DIAG_START: Diagnosis start<br>CANHEX_MODE_DIAG_END: Diagnosis end<br>CANHEX_MODE_CAN_BUS_SYNC_FAILED: CAN BUS synchronization failed<br>CANHEX_MODE_FAIL_SAFE: Fail-safe mode |
| mode_reason0 | string | CANHEX_MODE_REASON_NONE: Module reason register value 0, clear status<br><br>CANHEX_MODE_REASON_ONGOING: ongoing state<br><br>CANHEX_MODE_REASON_SUCCESS: Success status<br><br>CANHEX_MODE_REASON_CAN_BUS_UNKNOWN: CAN BUS is not recognized<br><br>CANHEX_MODE_REASON_CAN_BUS_UNCONNECT: CAN BUS is not |

| | | |
|---|---|---|
| | | recognized<br><br>CANHEX_MODE_REASON_ANALYSIS: CAN BUS not connected analysis<br>CANHEX_MODE_REASON_NONE: Module reason register 0 value, clear status<br><br>CANHEX_MODE_REASON_ONGOING: In progress status<br><br>CANHEX_MODE_REASON_SUCCESS: Success status<br><br>CANHEX_MODE_REASON_CAN_BUS_UNKNOWN: CAN BUS is not recognized<br><br>CANHEX_MODE_REASON_CAN_BUS_UNCONNECT: CAN BUS is not recognized<br><br>CANHEX_MODE_REASON_ANALYSIS: CAN BUS is not connected to analysis |
| mode_reason1 | string | Same as above |
| serial_mode | string | CANHEX_MODE_SERIAL_NONE: Serial port mode, clear status<br>CANHEX_MODE_SERIAL_XONXOFF: XONXOFF mode<br>CANHEX_MODE_SERIAL_ASCII: ASCII mode<br>CANHEX_MODE_SERIAL_CFG_MODE:<br>CANHEX_MODE_SERIAL_WORK_MODE:<br><br><br>CANHEX_MODE_SERIAL_NONE: Serial port mode, clear state<br>CANHEX_MODE_SERIAL_XONXOFF: XONXOFF mode<br>CANHEX_MODE_SERIAL_ASCII: ASCII mode<br>CANHEX_MODE_SERIAL_CFG_MODE:<br>CANHEX_MODE_SERIAL_WORK_MODE: |
| protocol_work_mode | string | CANHEX_MODE_WORK_NONE: Clear status<br>CANHEX_MODE_WORK_WORKING: Working mode<br>CANHEX_MODE_WORK_CFG_MODE: Configuration mode<br><br>CANHEX_MODE_WORK_NONE: Clear status<br>CANHEX_MODE_WORK_WORKING: Working mode<br>CANHEX_MODE_WORK_CFG_MODE: Configuration mode |
| new_firmware_valid | int | 0: No new configuration can take effect at present<br>1: There is a new configuration that can take effect.<br>0: No new configuration can be effective at the moment<br>1: New configuration can be effective at the moment |
| new_conf_valid | int | 0: No new configuration can take effect at present<br>1: There is a new configuration that can take effect.<br>0: No new configuration can be effective at the moment<br>1: New configuration can be effective at the moment |
| current_conf_valid | int | 0: The current configuration is invalid |

| | | |
|---|---|---|
| | | 1: The current configuration is valid<br>0: The current configuration is invalid<br>1: The current configuration is valid |
| serial_conf_valid | int | 0: Serial port not configured (cannot be installed)<br>1: The serial port has been configured<br>0: Serial port not configured (cannot be installed)<br>1: Serial port configured |
| model_id | string | Vehicle model id<br>Vehicle model id |
| model_id_state | string | MISS: Clear status<br>EMPTY: model id is not set<br>SET_BY_HAND: Manually set the model id<br>SET_BY_HAND2: Manually set the machine model<br>SET_BY_SYNC: Automatic synchronization |
| car_name | string | Vehicle Name |
| thr_support | int | 0: Does not support the reading function of the driving recorder<br>1: Support |
| s | string | CAN module SN |
| event_ack_enable | int | 0: Received an invalid response to the event<br>1: Received event must be responded |
| canbus_active_mode | int | 0: CAN node works in silent mode<br>1: The CAN node operates in active mode. |
| fuel_measure_delay | int | 0: No delay in fuel measurement<br>1: Fuel measurement delay |
| nonautomatic_sync | int | 0: Allow automatic synchronization when the model id is empty.<br>1: Disable automatic synchronization when the model id is empty. |
| can1_send_disable | int | 0: can2 can send information<br>1: can2 prohibits sending information |
| uart_tx_wakeup | int | 0: The TX pin status is idle when the CAN module is in sleep mode.<br>1: The TX pin status is working when the CAN module is in sleep mode. |
| short_wakeup | int | 0: Enter deep sleep after 60 seconds after waking up<br>1: Enter deep sleep 2 seconds after waking up |
| low_power_mode | int | 0: Disable low power mode<br>1: Enable low power mode |
| ignore_diagnostic_tools | int | 0: Stop sending messages when a diagnostic tool is detected<br>1: Ignore diagnostic tools |
| deep_sleep_mode_disable | int | 0: Enable deep sleep function<br>1: Disable the deep sleep function |
| thr_compatibility | int | 0: Dashcam compatibility is disabled<br>1: Enable the driving recorder function compatibility |
| can2_send_disable | int | 0: can2 can send information<br>1: can2 prohibits sending information |
| query_tachograph_mode | int | 0: Disable request for driving recorder<br>1: Request the dashcam to be in IGN ON state only. |

| | | |
|---|---|---|
| | | 2: Smart mode, requested when CABBUS is active.<br>3: You can always request. |

#### 7.1.3.4. Query car Model ID, VIN information

This interface is used to obtain the vehicle name, model id and model id source in the CAN module. Interface get_car_params_single, payload {"params_name":"String", "from": "String"}. See the table below for parameter payload description.

| name | type | Remark |
|---|---|---|
| params_name | string | car_name: vehicle name |
| from | string | cache: Get from the cache of the canobd support library |
| | string | module: Get directly from the CAN module and update the cache |

```
$ ubus call canobd get_car_params_single '{"params_name":"model_id", "from": "cache"}'
$ ubus call canobd get_car_params_single '{"params_name":"model_id", "from": "module"}'
$ ubus call canobd get_car_params_single '{"params_name":"car_name", "from": "module"}'
$ ubus call canobd get_car_params_single '{"params_name":"vin", "from": "module"}'
```


```
{
	"model_id": "004F",
	"model_id_state": "SET_BY_SYNC",
	"car_name": "Seat Altea (04-)",
	"from": "module",
	"rtn": "success"
}
```


Reply payload meaning table:

| name | type | Remark |
|---|---|---|
| model_id | string | Hex value, vehicle model id, CAN module definition. |
| model_id_state | string | MISS: Clear status<br>EMPTY: model id is not set<br>SET_BY_HAND: Manually set the model id<br>SET_BY_HAND2: Manually set the machine model<br>SET_BY_SYNC: Automatic synchronization |
| car_name | string | Vehicle Name |

#### 7.1.3.5. Clear car model id

This interface is used to clear the vehicle model id in the CAN module and delete the synchronization record kept in the canobd support library. Interface clear_car_model_id, payload empty.

Example:

```
$ ubus call canobd clear_car_model_id
```


```
{
	"rtn": "success"
}
```

#### 7.1.3.6. Set the car model id

This interface is used to manually set the vehicle model id in the CAN module, which will delete the synchronization record kept in the canobd support library. It is still necessary to call the "Car model id security synchronization" interface to verify the model id process. The canobd support library can be compatible with the scenarios of manually setting the model id and automatically synchronizing the model id.

| name | type | Remark |
|---|---|---|
| model_id | string | 0-ffff, HEX value. Vehicle model id, the specific value can be found in the information provided by the CAN module manufacturer. |

Example:

```
$ ubus call canobd set_car_model_id_hand '{"model_id": "0x0173"}'
{
	"rtn": "success"
}
```

#### 7.1.3.7. Perform secure synchronization of car model ID

This interface performs the vehicle model id security synchronization process according to the internal state of the CAN module. When the vehicle VIN code can be obtained, the synchronization is considered successful. Therefore, after manually setting the model id, you also need to use this interface to verify whether the set model id is normal.
Interface sync_car_model_id_safe, payload empty.
After the interface is called successfully, use the ubus call canobd get_state command to query the result.

Example:

```
$ ubus call canobd sync_car_model_id_safe
{
	"rtn": "success"
}
```

#### 7.1.3.8. Get synchronization real-time status

This interface is used to obtain the real-time status of the CAN module vehicle model id synchronization. The synchronization confirmation status is obtained through the "14. Get synchronization confirmation status" interface.
Interface get_state, payload {"scope": "String"}.

Example:

```
ubus call selftask canobd.get_state '{"scope": "sync_state"}'
{
	"safe_sync_type": "MISS",
	"sync_frame_state": "MISS",
	"dev_mode_sync_result": "MISS",
	"dev_mode_sync_reason": "MISS",
```

```
    "safe_sync_result": "MISS",
    "ignition_on": "UNKNOWN",
    "engine_on": "UNKNOWN",
    "model_id_state": "MISS",
    "model_id": "0000",
    "vehicle_info": "0",
    "vehicle_info_valid": 0,
    "vin": "",
    "vin_len": 0,
    "scope": "sync_state",
    "from": "cache",
    "rtn": "success"
}
```


The reply load parameters are shown in the following table.

| name<br>name | type<br>type | illustrate<br>Remark |
|---|---|---|
| safe_sync_type | string | MISS: Clear status<br>AUTO_SYNC: Automatically synchronize and obtain model id<br>SET_MODEL: Manually set the model id |
| sync_frame_state | string | MISS : Clear status<br>ONGOING : The synchronization process is in progress<br>OK : Synchronization successful<br>FAILED_UNRECOGNIZED : Unrecognized failure<br>FAILED_COMMU : Communication failed<br>FAILED_START : Synchronization start failed |
| dev_mode_sync_result | string | MISS: Clear status<br>ONGOING: Ongoing<br>FAILED: Failed |
| dev_mode_sync_reason | string | MISS: Clear status<br>ONGOING: Ongoing<br>OK: Completed<br>ANALYSIS: Analysis<br>UNRECOGNIZED: Unrecognized<br>UNCONNECTED: Not connected to the CAN bus |
| safe_sync_result | string | MISS: Clear status<br>OK: Security synchronization successful<br>FAILED: Security synchronization failed |
| ignition_on | string | UNKNOWN : IGN status, unknown<br>OFF ： IGN OFF<br>ON ： IGN ON |
| engine_on | string | UNKNOWN: Engine status, unknown<br><br>OFF: Engine OFF |

| | | |
|---|---|---|
| | | ON: Engine ON |
| model_id_state | string | MISS: Clear status<br><br>EMPTY: model id is not set<br><br>SET_BY_HAND: Manually set the model id<br><br>SET_BY_HAND2: Manually set the machine model<br><br>SET_BY_SYNC: Automatic synchronization |
| model_id | string | Hex value, 0~0xffff. |
| vehicle_info | string | Hex value. |
| vehicle_info_valid | int | 0: invalid; 1: valid. |
| vin | string | VIN code: VIN code |
| vin_len | int | Value, VIN code length |

#### 7.1.3.9. Search model id by car parameter name

This interface is used to query the model id in the CAN module using the vehicle name. The -t parameter needs to be used to set the ubus call timeout. The request timeout should be at least 80 seconds. Interface action_canobd_model_id_search, payload {"car_name": "String"}, timeout 80. Because the query is a keyword query, there will be multiple results. The user needs to match the model id of the most accurate result as the target model id.

| name | type | Remark |
|---|---|---|
| car_name | string | Vehicle name keywords |

Example:

```
$ ubus call canobd action_canobd_model_id_search '{"car_name": "GL8"}' -t 300
{
        "search_name": "GL8",
        "list_num": 1,
        "list": [
                {
                        "car_name": "BUICK GL8 (10-)",
                        "model_id": "0x0173"
                }
        ],
        "rtn": "success"
}
```

#### 7.1.3.10. Get vehicle parameter table values

This interface is used to obtain the car parameter table. These data are maintained by the car parameter polling service. For the meaning of the reply payload, refer to "Car Parameter Table 1", "Car Parameter Table 2" and "Car

Parameter Table 3". Interface get_car_params_content , payload {"scope": " String "} .

| name | type | Remark |
|---|---|---|
| scope | string | All: Get all driver card information<br>car_param1: Get the information of car parameter table 1<br>car_param2: Get the information of car parameter table 2<br>car_param3: Get the information of car parameter table 3 |

Example:

```
$ ubus call canobd get_car_params_content
$ ubus call canobd get_car_params_content '{"scope": "car_param1"}'
$ ubus call canobd get_car_params_content '{"scope": "car_param2"}'
$ ubus call canobd get_car_params_content '{"scope": "car_param3"}'
```

#### 7.1.3.11. Get car driver record information

This interface reads the driving time and driving behavior information of the driver card from the cache of the canobd support library. Interface get_car_driver_card_record , payload empty or {"scope": String}, payload parameters as shown in the following table, and the response result refers to "Driver Information Card Table" .

| name | type | Remark |
|---|---|---|
| scope | string | All: Get all driver card information |
| | string | card1: Get the information of driving card 1 |
| | string | card2: Get the information of driving card 2 |

Example:

```
$ ubus call canobd get_car_driver_card_record '{"scope": "card1"}'
$ ubus call canobd get_car_driver_card_record '{"scope": "card2"}'
$ ubus call canobd get_car_driver_card_record '{"scope": "all"}'
```

#### 7.1.3.12. Firmware upgrade interface

This interface is used to upgrade the firmware of the CAN module. The module firmware is provided by the CAN module manufacturer. The firmware of each module is bound to the module's SN, and different devices cannot be universal. Interface action_canobd_module_upgrade , payload {"scope": " String ", "file_path":" String "} . The meaning of the parameters is as follows.

| name | type | Remark |
|---|---|---|
| scope | string | firmware : The upgrade type is firmware |
| file_path | string | The path of the firmware file in the device. You need to place the firmware file in the specified location first, and then fill in this field |

Example:

```
$ ubus call canobd action_canobd_module_upgrade '{"scope": "firmware", "file_path":"/root/CL_v3.0.14_sn3863230.frm"}'
{
```

```
    "rtn": "success"
}
```

Query progress

Example:

```
$ ubus call canobd get_state '{"scope": "upgrade_state"}'
```

#### 7.1.3.13.Configure the upgrade interface

This interface is used to upgrade the configuration of the CAN module. The configuration file is provided by the CAN module manufacturer or generated by the tool provided by the manufacturer. Interface action_canobd_module_upgrade , payload {"scope": " String ", "file_path":" String "} . The meaning of the parameters is as follows.

| name | type | Remark |
|---|---|---|
| scope | string | conf: The upgrade type is configuration |
| file_path | string | The path of the configuration file in the device. You need to place the configuration file in the specified location first, and then fill in this field. |

Example:

```
$ ubus call canobd action_canobd_module_upgrade '{"scope": "conf", "file_path":"/root/queclink_xonxoff_115200.frm"}'
```

#### 7.1.3.14 Query upgrade progress

This interface is used to query the firmware or configuration upgrade status and progress. After calling "27. Firmware Upgrade Interface " or "28. Configuration Upgrade Interface ", you can query the upgrade status through this interface. Interface get_state , payload {"scope": " String "} .

Example:

```
$ ubus call canobd get_state '{"scope": "upgrade_state"}'
{
    "upgrade_state": "Upd Stete Ok",
    "content_type": "Upgrade Content Cfg",
    "sn": "",
    "upgrade_version": "B18E",
    "rate_progress": 100,
    "time_cost": 7,
    "desc": "Upd Fsm Check Ok",
    "scope": "upgrade_state",
    "from": "cache",
    "rtn": "success"
}
```

Reply payload meaning table:

| name | type | Remark |
|---|---|---|
| upgrade_state | string | Upd Stete None : No upgrade<br>Upd Stete Start : Upgrade starts |

| | | |
|---|---|---|
| | | Upd Stete Doing : Updating in progress<br>Upd Stete Ok : Upgrade completed<br>Upd Stete Terminated : Updating canceled<br>Upd Stete Failed : Updating failed |
| content_type | string | Upgrade Content None : Upgrade content has no type<br>Upgrade Content Fw : The upgrade type is firmware<br>Upgrade Content Cfg : The upgrade type is configuration<br>Upgrade Content Cfg Keep Model : The upgrade type is configuration and the model id is retained |
| s | string | CAN module SN: CAN module SN, returned when the firmware is upgraded |
| upgrade_version | string | Version information: firmware version information or configuration check number |
| rate_progress | int | Range -1~100. -1 means failure, 0~100 means the progress of the processing |
| time_cost | int | Duration, the time it takes to upgrade, in seconds |
| desc | string | Description, description of the upgrade process. |

#### 7.1.3.15. Enter /Exit Test Mode

Enter or exit the test mode. This interface is used to simulate the parameters and status of the car to help CAN module developers to verify the developed software. Interface set_canobd_test_mode , payload {"enable": " String "} , parameters are as follows.

| name | type | Remark |
|---|---|---|
| enable | string | 1: Enter test mode<br>0: Exit test mode |

Example:

```
$ ubus call canobd set_canobd_test_mode '{"enable": "1"}'
$ ubus call canobd set_canobd_test_mode '{"enable": "0"}'
```

#### 7.1.3.16. Module power control

This interface is used to control the power on and off of the power pin of the CAN module. Interface set_canobd_power , payload {"enable": " String "} . Parameters are as shown in the following table.

| name | type | Remark |
|---|---|---|
| enable | string | 0: Turn on the power<br>1: Turn off the power |

Example:

```
Turn on the module power
$ ubus call canobd set_canobd_power '{"enable": "1"}'
Turn off the module power
```

```
$ ubus call canobd set_canobd_power '{"enable": "0"}'
```

## 7.2. Batterymgr

### 7.2.1. Service Introduction

This software module manages the safe and efficient charging and discharging of the internal battery . With it running in the background, users do not have to worry about when to charge the battery, whether the charging temperature is too high , etc. Battery management is essential for the GV850 hardware to meet CE and other certifications.

The battery capacity of this product is 1100mAh 4.07Wh. The nominal voltage is 3.7V. This software module realizes highly configurable battery charging and discharging process management. It mainly realizes five basic functions.

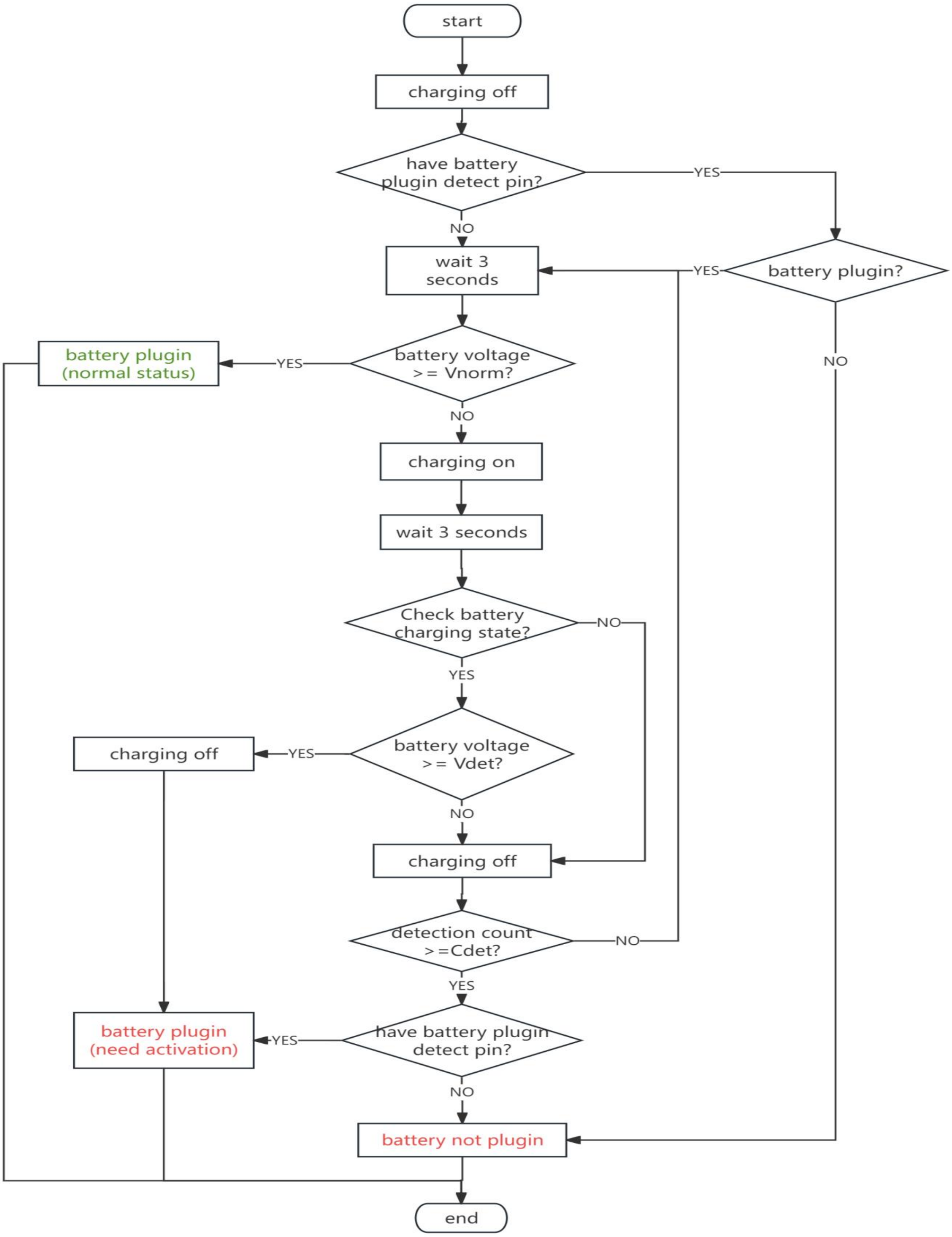
1) Battery in-place inspection and activation management.
2) Battery overcharge, over discharge and recharge management.
3) Battery high and low temperature charging management.
4) Calculate battery capacity (based on the charge and discharge curve).
5) Battery charge and discharge control, attribute and status acquisition interface.

### 7.2.2. Batterymgr management logic

#### 7.2.2.1. Battery in-place check

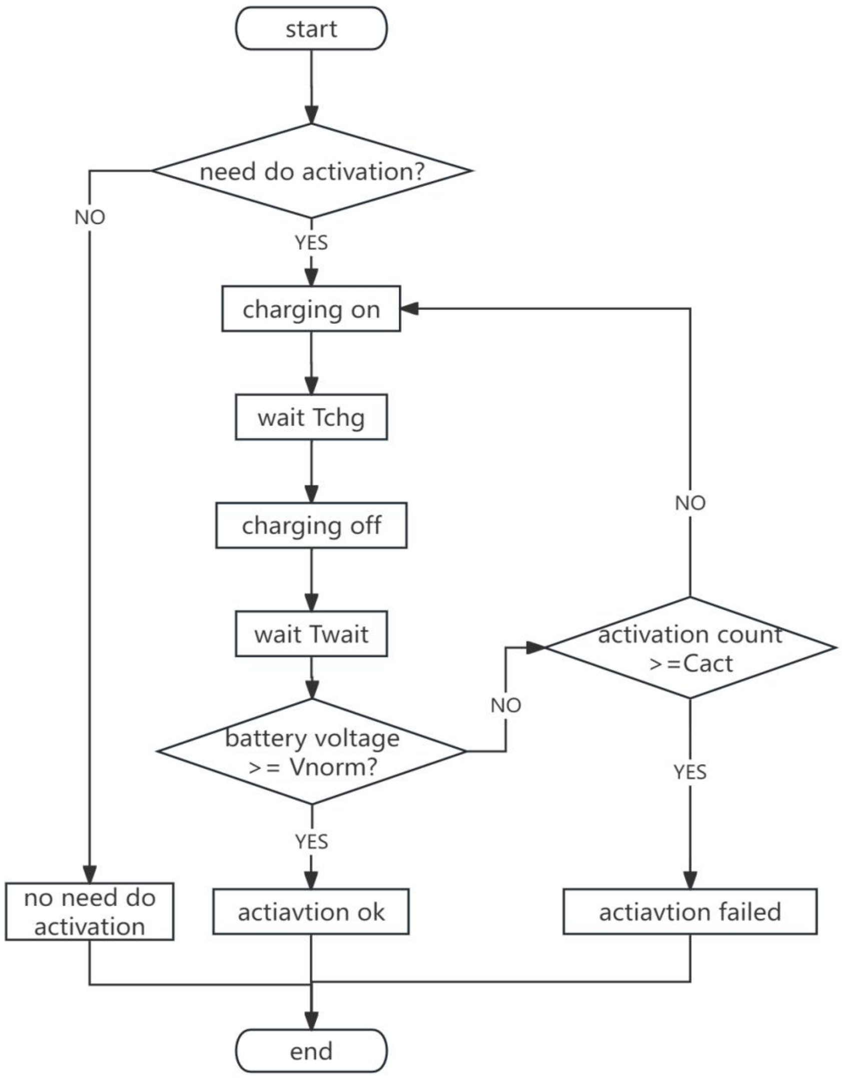
When the battery management service is started, the battery presence check process will be performed first, and the battery presence status will be determined by judging the battery voltage or the battery presence detection pin. After the presence check process, the battery presence status can be obtained. Currently, the software has set four states, namely "Initialization not completed", "Battery not in place", "Battery in place and normal", and "Battery in place, but needs to be activated".

There is no battery presence detection pin in the hardware at present, but we have taken it into account in the implementation of the service program. The in-place check process is shown in the figure below. In the figure below, Vnorm is the lowest check voltage in the normal state of the battery. Vdet is the lowest voltage in the battery self-protection state (activation action is required). Cdet is the maximum number of battery in-place checks.


start
charging off
have battery plugin detect pin?
YES
NO
battery plugin?
YES
wait 3 seconds
NO
battery plugin (normal status)
YES
battery voltage >= Vnorm?
NO
charging on
wait 3 seconds
Check battery charging state?
NO
YES
charging off
YES
battery voltage >= Vdet?
NO
charging off
detection count >=Cdet?
NO
YES
battery plugin (need activation)
YES
have battery plugin detect pin?
NO
battery not plugin
end

#### 7.2.2.2. Battery activation

After the battery in-place check process is completed, if the battery in-place status is "battery in place, but needs to be activated", the battery activation process is started. The activation process is to charge the battery intermittently until the battery voltage reaches the Vnorm state or fails to time out. There are five states generated in the activation process, namely "activation process not started", "no activation required", "activating", "activation failed", and "activation successful". The process is shown in the figure below.



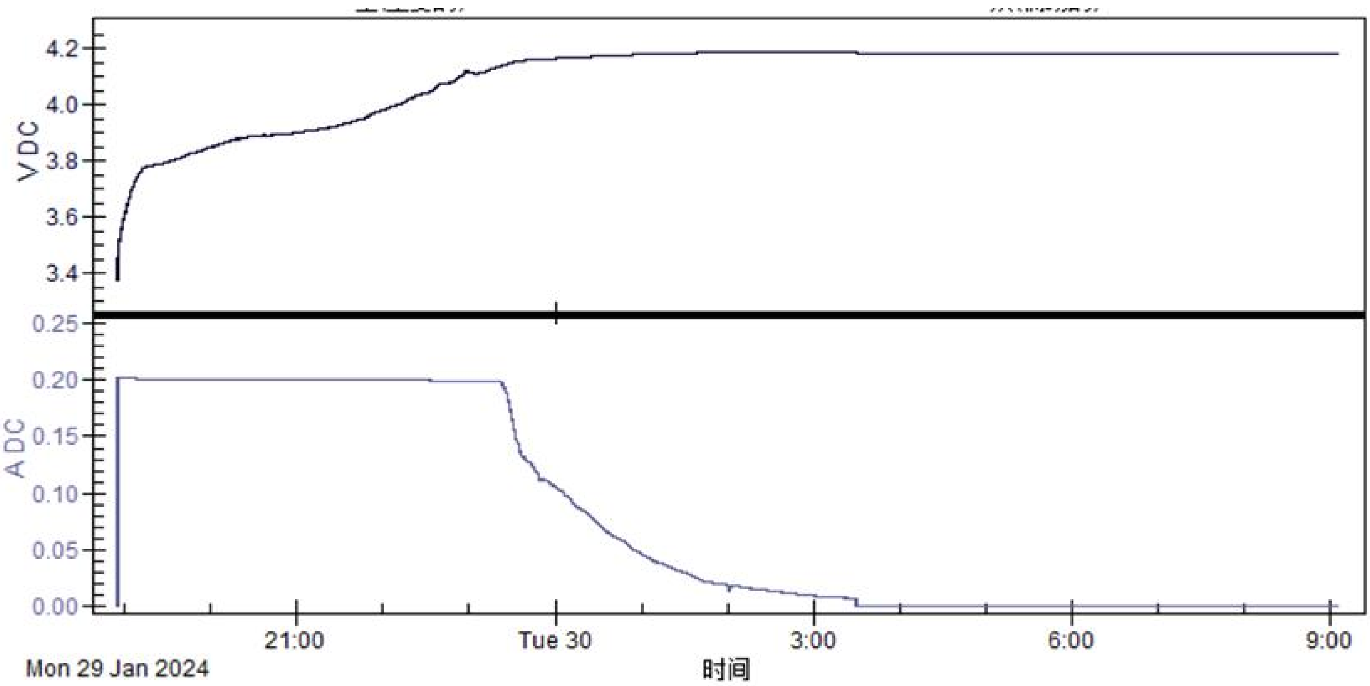

Where Vnorm is the lowest voltage value of the battery under normal conditions. Tchg is the activation continuous charging duration, usually 600 seconds. Twait is the activation stop charging duration, usually 3 seconds. Cact is the number of activation charges. When the number of activation charges exceeds this threshold, and the battery voltage still does not meet the minimum threshold Vnorm of the normal voltage, it is determined to be an activation failure.

After the activation process is completed, the battery initialization is complete. The next step is to manage the battery usage.

#### 7.2.2.3. Overcharge protection mechanism:

There are many reasons for overcharging, one of which is charging the battery for a long time after it is fully charged. This will reduce the battery life or bring the risk of battery damage. The battery management module determines whether the battery is fully charged under the guidance of the configuration. If it is fully charged, it will directly turn off the charging enable. This achieves the purpose of overcharging protection. The following is a brief description of the basic process of battery charging.

The battery charging process is managed by the charging IC and is divided into two stages: constant current charging and constant voltage charging. When the battery charging voltage is less than a threshold, the charging IC works in the constant current charging stage. When the battery voltage is greater than or equal to a threshold, it switches to constant voltage charging. Overcharging occurs in the constant voltage charging stage. Constant voltage charging will continue to charge the battery, but the current will gradually decrease until it is trickle charged. The battery management service uses software to control the charging enable. When it is determined that the battery is full, the charging enable can be directly turned off to stop charging. The charging process of this product is shown in the figure below.




#### 7.2.2.4. Recharge mechanism

When the battery is fully charged and drops to the recharge voltage, the charging enable is turned on to charge the battery. This can reduce the number of battery charges and increase the battery life. When the external power is plugged in or out, the recharge flag will be cleared.

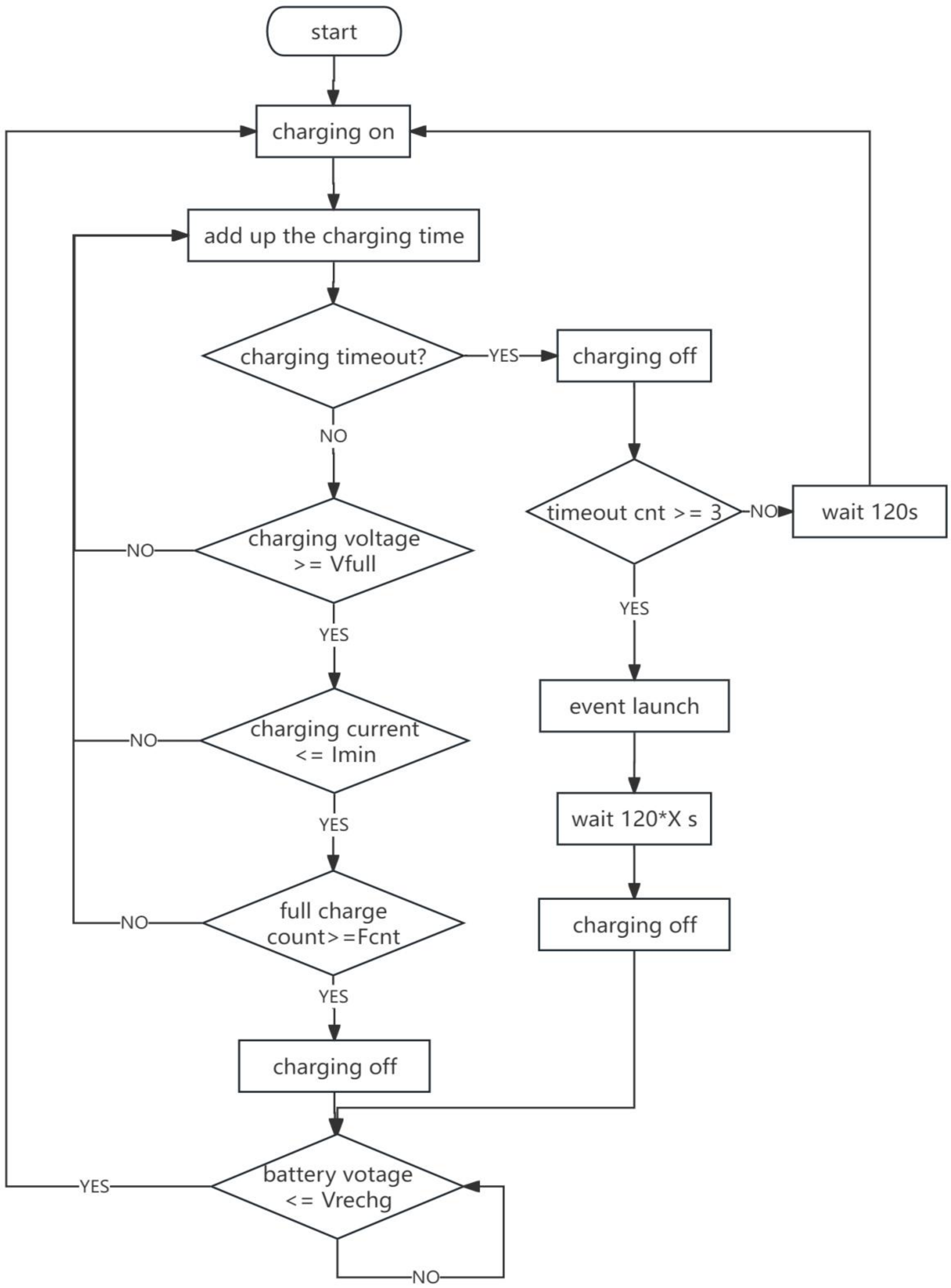
#### 7.2.2.5. Over-discharge protection mechanism

When the battery voltage is lower than the shutdown voltage, the battery is disconnected and the power is turned off to avoid the risk of over-discharge of the battery. This protects the battery voltage from being too low.

#### 7.2.2.6. Charging timeout mechanism

Through theoretical calculation and actual charging time, the baseband gives the charging timeout time Tc. When the charging time exceeds Tc, the software controls the charging enable function to be turned off for 2 minutes, and then the charging enable is turned on again. In this case, retry twice. If the charging is still not completed after two retries, and the battery voltage is not 100%, the "battery damage or charging function damage fault" is reported. The actual charging time*1.5 is used to get it.

The charging process, recharge mechanism, and charging timeout process are shown in the figure below. Tc is used as the charging timeout threshold. Vfull is the full-charge voltage value, which is mainly used to determine whether it is in a constant-voltage charging state at the time. Whether it is fully charged needs to be determined based on the charging current.




#### 7.2.2.7. High and low temperature charging and discharging management

The real-time temperature of the battery is collected through the NTC sensor carried by the battery. The charging and discharging are controlled according to the battery temperature. Currently, the default range is 0-50 degrees Celsius for charging, otherwise charging is stopped. When a high or low temperature event occurs, the charging function will be turned on again when the temperature needs to be restored to 5-45 degrees Celsius. Discharges are currently not treated at high and low temperatures.

#### 7.2.2.8. Battery Charge Calculation

At present, the management program uses three reference tables for battery charge calculation: charging battery voltage, discharging battery voltage, and charging current. At present, the battery charge needs to be actually tested and calibrated in the above three tables to achieve an accurate and reasonable charge value.

When the battery is in the discharge state, only the discharge curve is used to calculate the power value. In the charging state, the first half of the power depends on the charging voltage curve, and the second half of the power depends on the charging current curve. The charging power dividing line depends on the configuration file.

#### 7.2.3. Batterymgr Core Interface

##### 7.2.3.1. Get battery management configuration

Get the battery-related configuration of the Batterymgr service background.

```
$ ubus call selftask batt.get_config
{
	"detection": {
		"det_vol": 1000,
		"det_cnt": 3
	},
	"activate": {
		"act_vol": 3000,
		"act_cnt": 2,
		"act_charge_time": 600,
		"act_charge_wait_time": 3
	},
	"common": {
		"vol_extern_power": 10500,
		"charge_vol_max": 4400,
		"charge_cur_max": 400,
		"vol_power_off": 3460,
		"vol_fall_charge": 3950,
		"vol_fall_charge_adapt": 0,
		"vol_full_charge": 4120,
		"vol_full_charge_adapt": 0,
		"per_full_charge": 98,
		"cur_full_charge": 160,
		"cur_full_charge_adapt": 0,
		"timeout_charge": 28800,
		"timeout_charge_silent": 120,
		"timeout_charge_long_silent": 1200,
		"timeout_charge_try_cnt": 3
	},
	"temp_threshold": [
		{
			"temp_lower": 0,
			"temp_upper": 50,
			"temp_lower_recover_delta": 5,
			"temp_upper_recover_delta": 5
```

```
            }
        ],
        "rtn": "success"
}
```

The configuration parameters are described in the following table.

| name | type | Remark |
|---|---|---|
| detection.det_vol | int | The minimum voltage threshold for battery voltage presence check, in mV. |
| detection.det_cnt | int | Maximum number of battery presence checks |
| activate.act_vol | int | The minimum voltage threshold for successful battery activation, in mV. |
| activate.act_cnt | int | Battery activation times |
| activate.act_charge_time | int | Battery activation continuous charging time, in seconds |
| common.vol_extern_power | int | Minimum voltage value for external power supply in place check, in mV. |
| common.charge_vol_max | int | The maximum charging voltage of the battery, in mv. |
| common.charge_cur_max | int | The maximum battery charging current threshold, in mA. |
| common.vol_power_off | int | Battery shutdown voltage, in mV. |
| common.vol_fall_charge | int | Recharge voltage threshold. When the voltage is lower than the threshold, the recharge mark is cleared and charging is performed. |
| common.vol_fall_charge_adapt | int | Recharge voltage threshold adjustment value, reserved. |
| common.vol_full_charge | int | Full charge voltage threshold, in mV. |
| common.vol_full_charge_adapt | int | Full charge voltage adjustment value, reserved. |
| common.per_full_charge | int | Full power percentage, reserved. |
| common.cur_full_charge | int | Minimum charging current threshold for full charging, in mA. |
| common.cur_full_charge_adapt | int | Corrected value of cur_full_charge, reserved. |
| common.timeout_charge | int | Charging timeout, in seconds. |
| common.timeout_charge_silent | int | After charging times out, stop charging for a certain period of time. Unit: seconds. |
| common.timeout_charge_long_silent | int | When multiple charging times have expired, the charging will stop for a certain period of time, in seconds. |
| common.timeout_charge_try_cnt | int | Charging timeout threshold. If the number of times exceeds this threshold, it is considered a timeout event. |
| temp_threshold.temp_lower | int | Low temperature threshold, controls whether charging is allowed. |
| temp_threshold.temp_upper | int | High temperature threshold, controls whether charging is allowed. |
| temp_threshold.temp_lower_recover_delta | int | The difference between the recovery value and the threshold after a low temperature event. Always a positive number. |
| temp_threshold.temp_upper_recover_delta | int | The difference between the recovery value and the threshold after a high temperature event. Always a positive number. |

#### 7.2.3.2. Get battery status

Get the status of the battery.

```
$ ubus call selftask batt.get_state
{
	"mgr_enable": "1",
	"calibration_state": "BATT_CAL_NONE",
	"detection_state": "BATT_DET_EXIST_VOL",
	"activate_state": "BATT_ACT_NO_NEED",
	"discharge_enable": "1",
	"discharge_voltage": "4.099",
	"charge_enable": "1",
	"charge_state": "CHARHING",
	"charge_voltage": "4.099",
	"charge_current": "194",
	"recharge_state": "CLEAN_RECHAGE",
	"extern_power_state": "INSERT",
	"extern_power_voltage": "12.026",
	"temp": "29",
	"percent": "15",
	"rtn": "success"
}
```

The battery management module provides an external interface for querying the battery status and controlling the battery charging and discharging hardware. The queryable status is shown in the following table:

| name | type | Remark |
|---|---|---|
| mgr_enable | string | Whether battery management is enabled. 0: Disable; 1: Enable. |
| calibration_state | string | Battery voltage calibration status. BATT_CAL_NONE: not calibrated; BATT_CAL_DONE: calibrated. |
| detection_state | string | Battery initialization check status .<br>BATT_DET_NONE: initialization is not completed;<br>BATT_DET_NOT_EXIST: The battery is not in place;<br>BATT_DET_EXIST_VOL: The battery is in place and normal;<br>BATT_DET_EXIST_VOL_ACT: The battery is present but needs to be activated. |
| activate_state | string | Battery activation status.<br>BATT_ACT_NONE: the activation process is not started;<br>BATT_ACT_NO_NEED: No activation required;<br>BATT_ACT_DOING_NOW: activating;<br>BATT_ACT_FAILED: activation failed;<br>BATT_ACT_SUCCESS: Activation successful. |
| discharge_enable | string | Battery discharge enable status. 0: Disable; 1: Enable. |
| discharge_voltage | string | Battery voltage when battery discharge is enabled. Unit: V. |
| charge_enable | string | Battery charging enable status. 0: Disable; 1: Enable. |
| charge_state | string | Battery charging status. CHARHING: Charging; NOT_CHARGE: Not charging. |

| | | |
|---|---|---|
| charge_voltage | string | Battery voltage when battery charging is enabled. Unit: V. |
| charge_current | string | Battery charging current. Unit: ma. |
| recharge_state | string | Battery recharge status, that is, the mark is set after full charge and cleared when the battery drops to the recharge voltage.<br>WAIT_RECHAGE : After fully charged, set the flag and wait for recharging;<br>CLEAN_RECHAGE : Clear the mark and perform recharge. |
| extern_power_state | string | External power plugged in.<br>INSERT : external power access;<br>NOT_INSERT : external power is not connected; |
| extern_power_voltage | string | External voltage value, unit is V. |
| percent | string | Battery level, range 0-100. |
| temp | string | Battery NTC temperature. Unit: degrees. |

ubus call selftask batt.get_config

#### 7.2.3.3. Battery discharge/charge enable

batt.set_control '{"scope": "String", "enable":"String", "attr":"String"}'

Example:

```
$ ubus call selftask batt.set_control '{"scope": "discharge", "enable":"1", "attr":"ONLY_SET_HW"}'
{
	"rtn": "success"
}
```

| name | type | Remark |
|---|---|---|
| scope | String | The object to be set. The possible range is "discharge " : means to control discharge enable; "charge": means to control charge enable. |
| enable | string | Enable flag. 0: Disable; 1: Enable. |
| attr | string | Control attribute flags.<br>ONLY_SET_HW : Only set the hardware pins, without changing the program's internal flags.<br>BOTH_SET_HW_SW : Set the hardware pin and change the program internal flag.<br>ALL_SET_AND_FORCE : Set the hardware pins and change the program's internal flags. Disable the program's internal control mechanism and force the corresponding control to be set.<br>RECOVERY_SW_CONTROL : Set the hardware pins and change the program internal flags. Recover the program's internal control mechanism. |

### 7.3. Selftask

In order to meet the testing needs, Queclink developed the selftask module to simulate actual business scenarios. Its main function is to collect system and module information and send messages to the server at set periodic intervals .

selftask device runs automatically after booting, and can be stopped/started by the following commands .

Stop the program :

```
$ /etc/init.d/S99selftask stop
OK
```

Start the program:

```
$ /etc/init.d/S99selftask start
OK
```

#### 7.3.1. Reporting messages

The message uses JSON format , as shown below:


```
{
	"SYSTEM":{
		"version":"GV850_R00A01V01",
		"model":"GV850CEU",
		"hardware_version":"V1.01",
		"kernel_version":"5.15.67",
		"date":"Sat Jan 1 00:35:11 UTC 2000",
		"uptime":"00:35:02",
		"rootfs_size":"19.2 MB",
		"ram_size":"78032/106632 KB"
	},
	"LTE":{
		"version":"EG915UECABR03A03M08_01.200.01.200",
		"imei":"866344050767040",
		"csq":"9,99",
		"qcsq":"\"LTE\",94,-128,45,-16",
		"sim":"READY",
		"iccid":"89860119801697983674",
		"cs":"0,1",
		"qspn":"\"CHN-UNICOM\",\"UNICOM\",\"\",0,\"46001\"",
		"qnwinfo":"\"FDD LTE\",\"46001\",\"LTE BAND 3\",1650",
		"ps":"1",
		"pdp":"1,1,1,\"10.69.230.192\"",
		"sock":"0,\"TCP\",\"218.17.50.142\",971,0,2,1,0,0,\"uart1\""
	},
	" CANOBD " :{
		"vehicle_sleep": "Active",
		"can1_state": "Car Can Bus Not Used",
		"can2_state": "Car Can Bus Error",
		"can_bus_ign": "Can Ign Not Available",
		"pin_ign": "Car Ign On",
		"engine_state": "Car Engine Off",
		"ddd_dstate": "Car Remote Ddd Download Not Support",
```

```
        "th_comm_state": "Car Tachograph Comm State No",
        "kline_state": "Car Kline State Bus Not Use"
    },
    "BATT": {
        "mgr_enable": "1",
        "calibration_state": "BATT_CAL_NONE",
        "detection_state": "BATT_DET_EXIST_VOL",
        "activate_state": "BATT_ACT_NO_NEED",
        "discharge_enable": "1",
        "discharge_voltage": "4.099",
        "charge_enable": "1",
        "charge_state": "CHARHING",
        "charge_voltage": "4.099",
        "charge_current": "194",
        "recharge_state": "CLEAN_RECHAGE",
        "extern_power_state": "INSERT",
        "extern_power_voltage": "12.026",
        "percent": "15",
        "temp": "29"
    } ,
    "WDG":{

    },
    "RTC":{

    },
    "BLE":{
        "version":"NABE5_BT_R00A02V03",
        "boot_version":"NABE5_BT_BOOTR00A01V01",
    },
    "GSENSOR":{

    },
    "GPS":{
        "firmware version": "ROM SPG 5.10 (7b202e)",
        "state": "3D fixed",
        "utc time": "2024-01-18 14:14:24",
        "longitude": "113.947969",
        "latitude": "22.573546",
        "altitude": "116.400002",
        "speed": "0.009260km\/h"
    },
    "CAN":{
        "type":"external",
```

```
        "version":"3.0.8m",
    },
    "RS232":[{

    },
    {

    }],
    "RS485":{

    },
    "16PIN":{

    },
    "10PIN":{

    },
}
```


The message data parameters are described as follows:

SYSTEM section:

| name | type | Remark |
| --- | --- | --- |
| version | string | Software version , such as :<br>GV850_R00A01V01 |
| model | string | Device model , such as :<br>GV850CEU |
| har dware_version | string | Hardware version , such as:<br>V1.01 |
| kernel_version | string | Kernel version, such as:<br>5.15.67 |
| date | string | System time, such as:<br>Sat Jan 1 00:20:59 UTC 2000 |
| uptime | string | Run time, such as :<br>00:21:02 |
| rootfs_size | st ring | |
| r am_size | st ring | Memory status , such as: |

For the LTE part, the message parameters are described in the following table :

| name | type | Remark |
| --- | --- | --- |
| version | string | Firmware version, such as:<br>EG915UEUABR02A05M08_01.001.01.001 |
| im ei | string | Module IMEI. |

| | | |
|---|---|---|
| csq | string | Signal quality . |
| Qq | string | Signal quality . |
| sim | string | SIM card status , |
| iccid | string | SIM card number , |
| cs | string | CS domain registration status, |
| q spn | string | Operator information, |
| qnwinfo | string | Operator network information, |
| ps | string | PS domain registration status, |
| pd p | string | PDP data service information, |
| sock | string | Data connection information, |

The CANOBD part , the message parameters are described in the following table :

| name | type | illustrate |
|---|---|---|
| vehicle_sleep | string | Vehicle sleep mode<br>Active: The vehicle CAN bus is active and the engine is started<br>Sleep: Car CAN sleep and engine shutdown |
| can1_state | string | CAN1 bus status<br>Car Can Bus In Sleep: Car CAN bus sleep state<br>Car Can Bus Active: Car CAN bus active status<br>Car Can Bus Error: Car CAN bus error<br>Car Can Bus Not Used: Car CAN bus is not enabled |
| can2_state | string | CAN2 bus status<br>Same as CAN1 bus status |
| can_bus_ign | string | IGN signal obtained by the CAN module<br>Car Ign Off: IGN off state<br>Car Ign On: IGN on<br>Car Ign Bus Error: CAN BUS error<br>Can Ign Not Available: The parameter is invalid. |
| pin_ign | string | Hardware PIN pin IGN status :<br>Car Ign Off: IGN off state<br>Car Ign On: IGN on |
| engine_state | string | Engine status<br>Car Engine Off: Engine off<br>Car Engine On: Engine on<br>Car Engine Bus Error: CAN BUS error<br>Can Engine Not Available: The parameter is invalid. |
| ddd_dstate | st ring | DDD Download Status<br>Car Remote Ddd Download Disable : Remote DDD download function is disabled<br>Car Remote Ddd Download Enable : Remote DDD download function is enabled<br>Car Remote Ddd Download Comm Error : Remote DDD download communication error<br>Car Remote Ddd Download Not Support : Remote DDD download is not supported |

| | | |
|---|---|---|
| th_comm_state | st ring | Communication status of driving recorder<br>Car Tachograph Comm State No: No communication with the dashcam<br>Car Tachograph Comm State Online: Driving records are available online<br>Car Tachograph Comm State Comm Err: Car Tachograph communication error<br>Car Tachograph Comm State Not Support: Communication with the driving recorder is not supported |
| kline_state | st ring | KLine communication status<br>Car Kline State In Sleep: Kline is in sleep state<br>Car Kline State Active: Kline is in active state<br>Car Kline State Bus Error: Kline bus error<br>Car Kline State Bus Not Use: kline is not enabled |

In the BATT part , the message parameters are described in the following table :

| name | type | Remark |
|---|---|---|
| mgr_enable | string | Whether battery management is enabled. 0: Disable; 1: Enable. |
| calibration_state | string | Battery voltage calibration status. BATT_CAL_NONE: not calibrated; BATT_CAL_DONE: calibrated. |
| detection_state | string | Battery initialization check status .<br>BATT_DET_NONE: initialization is not completed;<br>BATT_DET_NOT_EXIST: The battery is not in place;<br>BATT_DET_EXIST_VOL: The battery is in place and normal;<br>BATT_DET_EXIST_VOL_ACT: The battery is present but needs to be activated. |
| activate_state | string | Battery activation status.<br>BATT_ACT_NONE: the activation process is not started;<br>BATT_ACT_NO_NEED: No activation required;<br>BATT_ACT_DOING_NOW: activating;<br>BATT_ACT_FAILED: activation failed;<br>BATT_ACT_SUCCESS: Activation successful. |
| discharge_enable | string | Battery discharge enable status. 0: Disable; 1: Enable. |
| discharge_voltage | string | Battery voltage when battery discharge is enabled. Unit: V. |
| charge_enable | string | Battery charging enable status. 0: Disable; 1: Enable. |
| charge_state | string | Battery charging status. CHARHING: Charging; NOT_CHARGE: Not charging. |
| charge_voltage | string | Battery voltage when battery charging is enabled. Unit: V. |
| charge_current | string | Battery charging current. Unit: ma. |
| recharge_state | string | Battery recharge status, that is, the mark is set after full charge and cleared when the battery drops to the recharge voltage.<br>WAIT_RECHAGE : After fully charged, set the flag and wait for recharging;<br>CLEAN_RECHAGE : Clear the mark and perform recharge. |
| extern_power_state | string | External power plugged in.<br>INSERT : external power access;<br>NOT_INSERT : external power is not connected; |
| extern_power_voltage | string | External voltage value, unit is V. |

| | | |
|---|---|---|
| percent | string | Battery level, range 0-100. |
| temp | string | Battery NTC temperature. Unit: degrees. |

For GPS , the message parameters are described in the following table :

| name | type | Remark |
|---|---|---|
| Fireware version | string | Firmware version, such as:<br>ROM SPG 5.10 (7b202e) |
| state | string | Working status. For example: 3D fixed |
| utc time | string | Utc time . The format is: YYYY-MM-DD HH:MM:SS |
| longitude | string | Longitude. For example: 113.947969 |
| latitude | string | Latitude. For example: 22.573546 |
| altitude | string | Altitude |
| speed | string | Speed. For example: 0.009260km/h |

BLE part ,

| name | type | Remark |
|---|---|---|
| Version | String | Firmware version , such as:<br>NABE5_BT_R00A02V03 |
| boot_version | string | BOOT version , such as:<br>NABE5_BT_BOOTR00A01V01 |

#### 7.3.2. Core Interface

##### 7.3.2.1. Reporting Configuration

Set the remote TCP server IP address and port, and message interval .

"modem.config":{"tcp_remote_addr":"String","tcp_remote_port":Integer,"report_interval":Integer}

Example:

```
$ ubus call selftask modem.config '{"tcp_remote_addr":"218.17.50.142","tcp_remote_port":971,"report_interval":3}'
{
    "rtn": "success"
}
```

##### 7.3.2. 2. AT command transparent transmission

Send AT commands and receive response messages .

"modem.raw":{"cmd":"String","timeout":"Integer"}

Example:

```
$ ubus call selftask modem.raw '{"cmd":"AT+CPIN?"}'
{
    "rtn": "success",
    "data": "\r\n+CPIN: READY\r\n\r\nOK\r\n"
}
```

#### 7.3.2. 3. Query status information

Query the basic status of the LTE module.
"modem.get_state":{}
Example :

```
$ ubus call selftask modem.get_state
{
    "version": "EG915UEUABR02A05M08_01.001.01.001",
    "imei": "866344050762298",
    "sim": "READY",
    "iccid": "89860119801697983674",
    "cs": "0,1",
    "ps": "1",
    "pdp": "1,1,1,\"10.32.148.5\"",
    "sock": "0,\"TCP\",\"218.17.50.142\",971,0,2,1,0,0,\"uart1\""
}
```

#### 7.3.2.4. Query positioning status

Query the GPS version and positioning status.
" gps.get_state ":{" firmware version ":"String"," state ":"String"}
Example:

```
$ ubus call selftask gps.get_state
{
    "firmware version": "ROM SPG 5.10 (7b202e)",
    "state": "2D fixed"
}
```

#### 7.3.2.5. Query GPS location information

" gps . get_location "
Example:

```
$ ubus call selftask gps .get_location
{
    "state": "2D fixed",
    "utc time": "2024-01-19 07:19:08",
    "longitude": "113.947976",
    "latitude": "22.573527",
    "altitude": "116.000000",
    "speed": "0.014816km/h"
}
```

## 7.4. Testcase

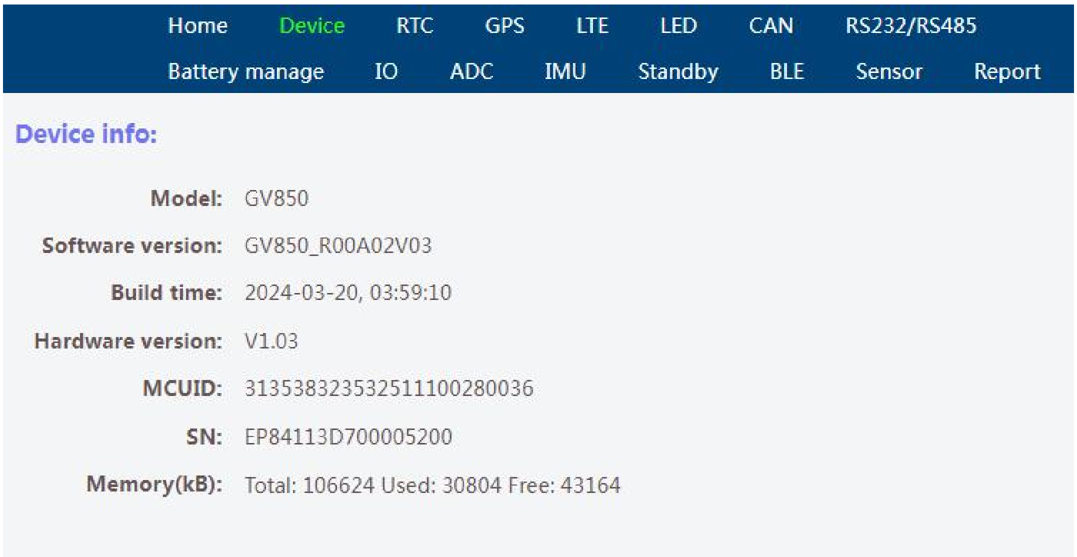
### 7.4.1 Module Introduction

We provide a web testing service for the device, which allows you to directly open the device's test web page through the browser of the development coCPUter to preview and test the functions. This service is convenient for customers to conduct functional testing and reference code. The web service uses the python+flask solution.
Specific features include:

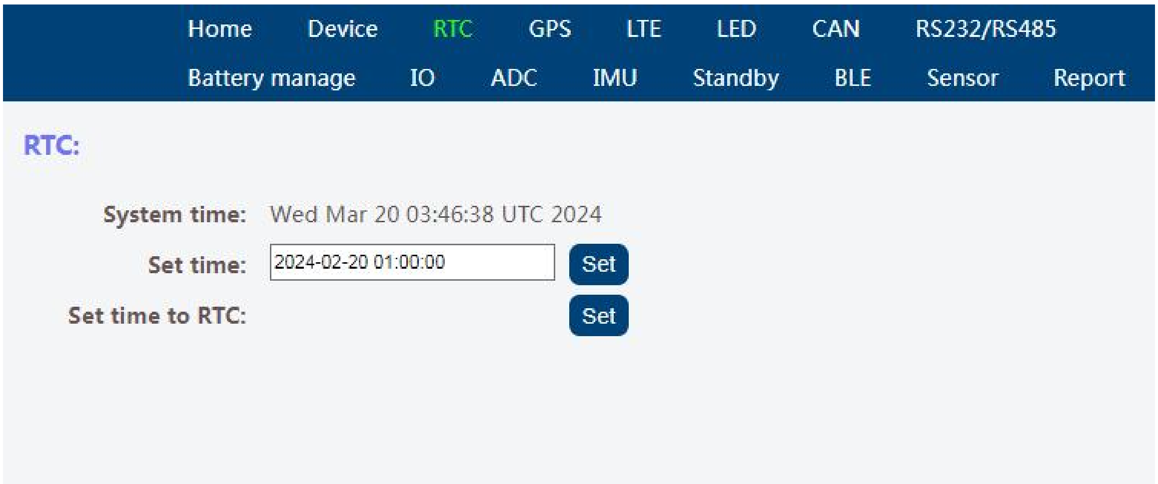
| Hardware Module | Test content |
|---|---|
| Device | View device version information and memory status |
| RTC | Check and set the device system time |
| G NSS | Check the device positioning status every three seconds |
| LTE | View LTE module information and transparent transmission AT command test |
| led | Control three LED lights to test |
| CAN | Use CAN module to send and receive data test |
| RS232/RS485 | Set RS485/RS232 port baud rate and send and receive data test |
| Batte r y manage | Read the device's external power and backup power voltage and control the device's battery charging and discharging . Query the battery's charging status and current. |
| IO | Read device DIN and set device OUT status |
| ADC | Read the value of the device AIN port |
| IMU | Read and set the G-Sensor register value |
| Standby | Test system sleep and wakeup through RTC |
| BLE | View the basic information of the device's BLE module |
| Sensor | Read the device's real-time G-Sensor data |
| Report | Set TCP server parameters. This server is used to receive selftask report messages. |

### 7.4.2. Page Display

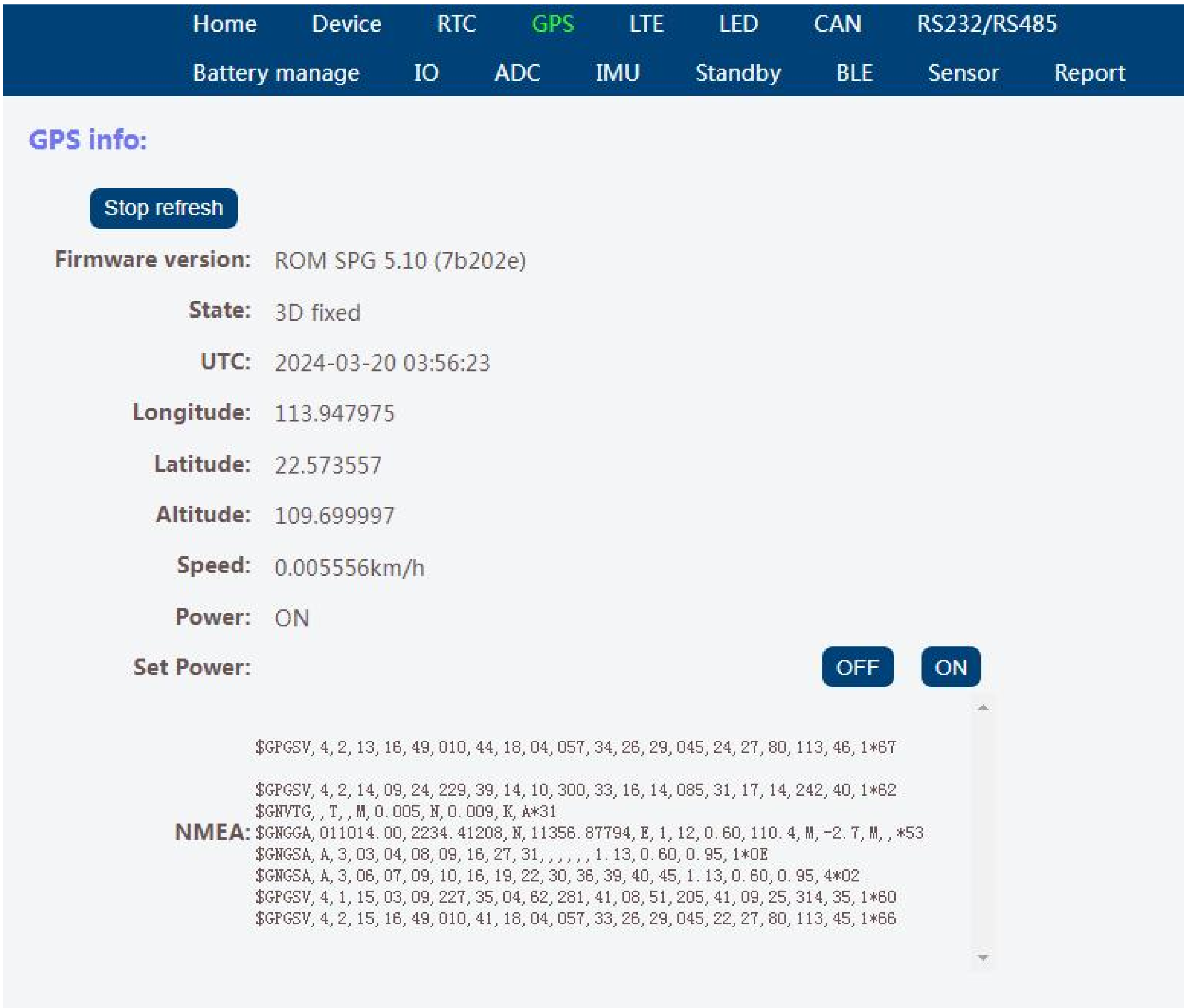
The Device subpage displays some basic information about the device and dynamically refreshes the memory usage.



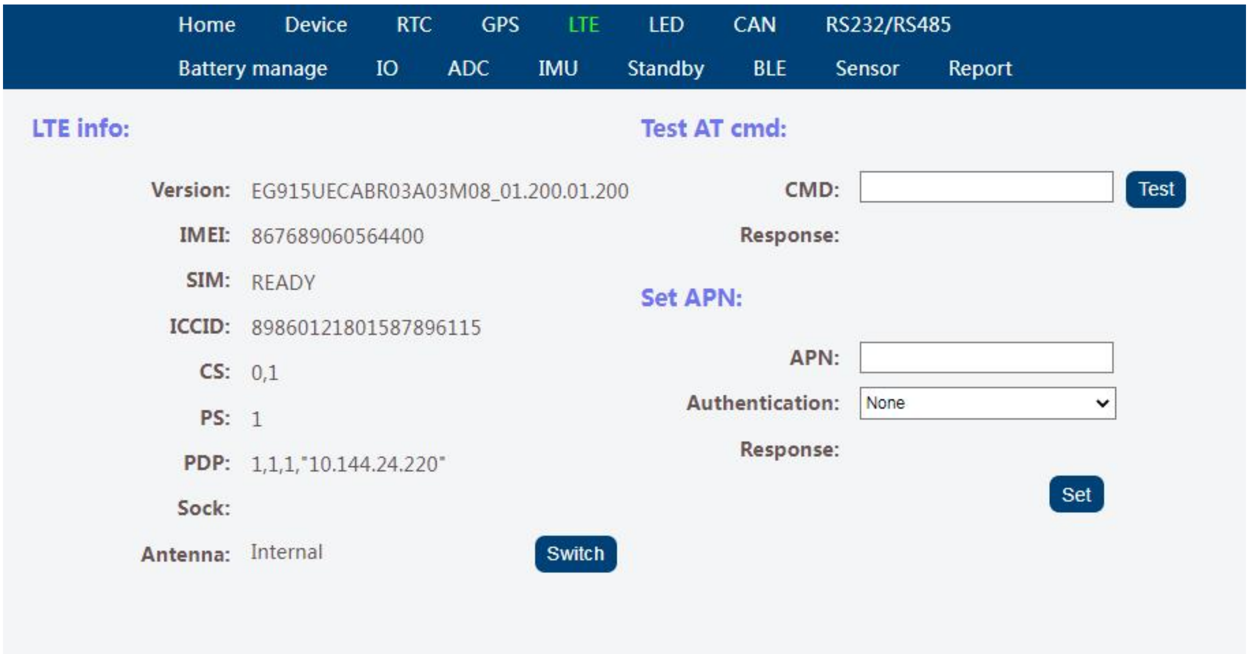

The RTC subpage supports setting the system time and setting the system time to the RTC.



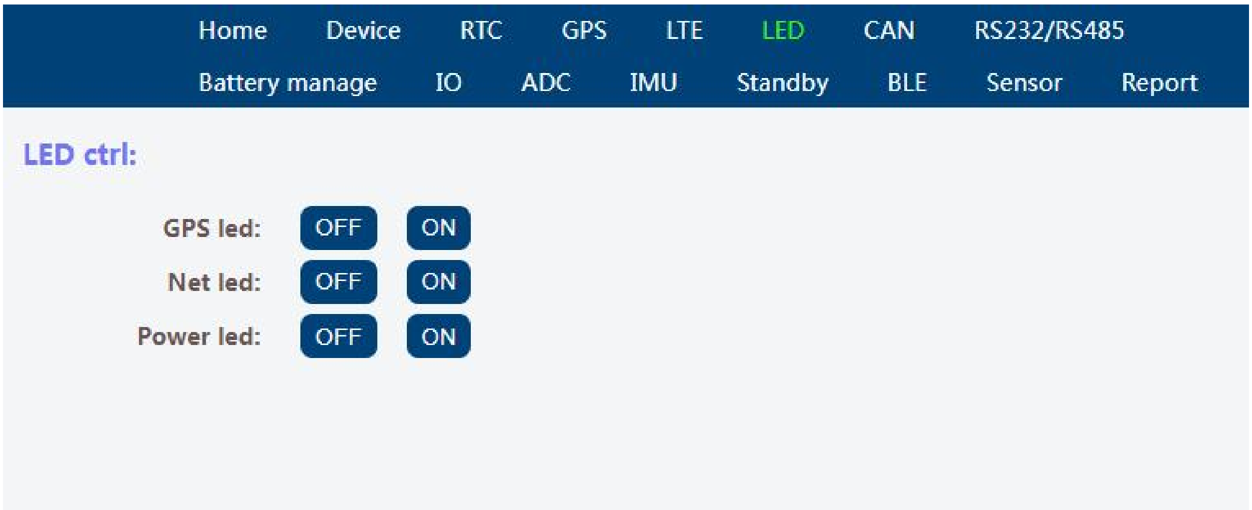

In the GPS subpage, click the OFF/ON button to turn the GPS power off/on. Click the " Stop refresh " button to turn the positioning information refresh on/off.



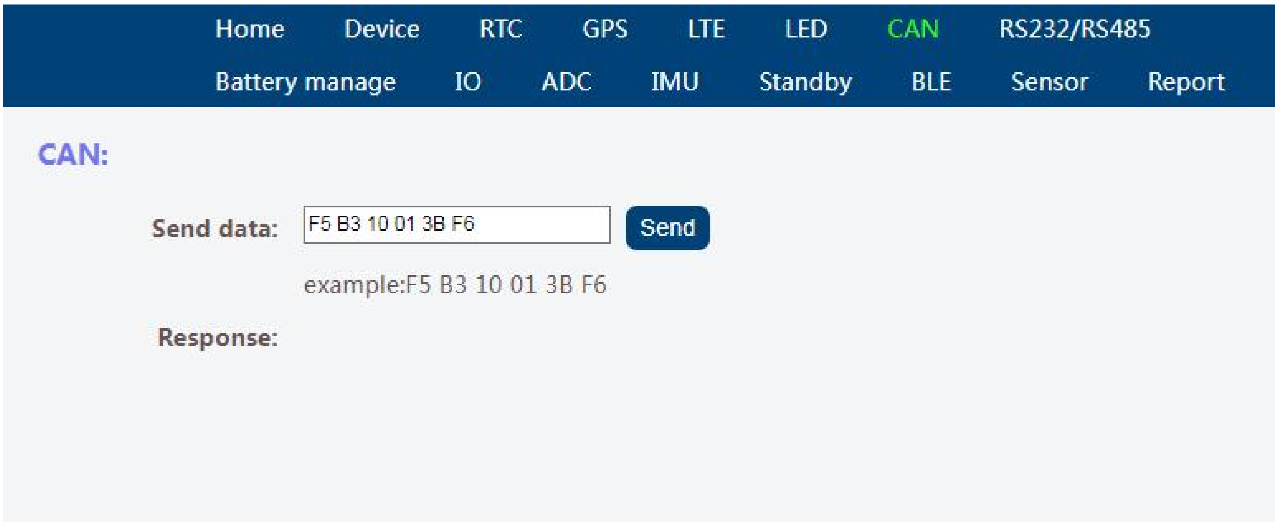

The LTE subpage is used for LTE module testing. Fill in the AT command in the CMD input box and click the " Test " button to test the command return value. It is often used to set the APN parameters of the module.



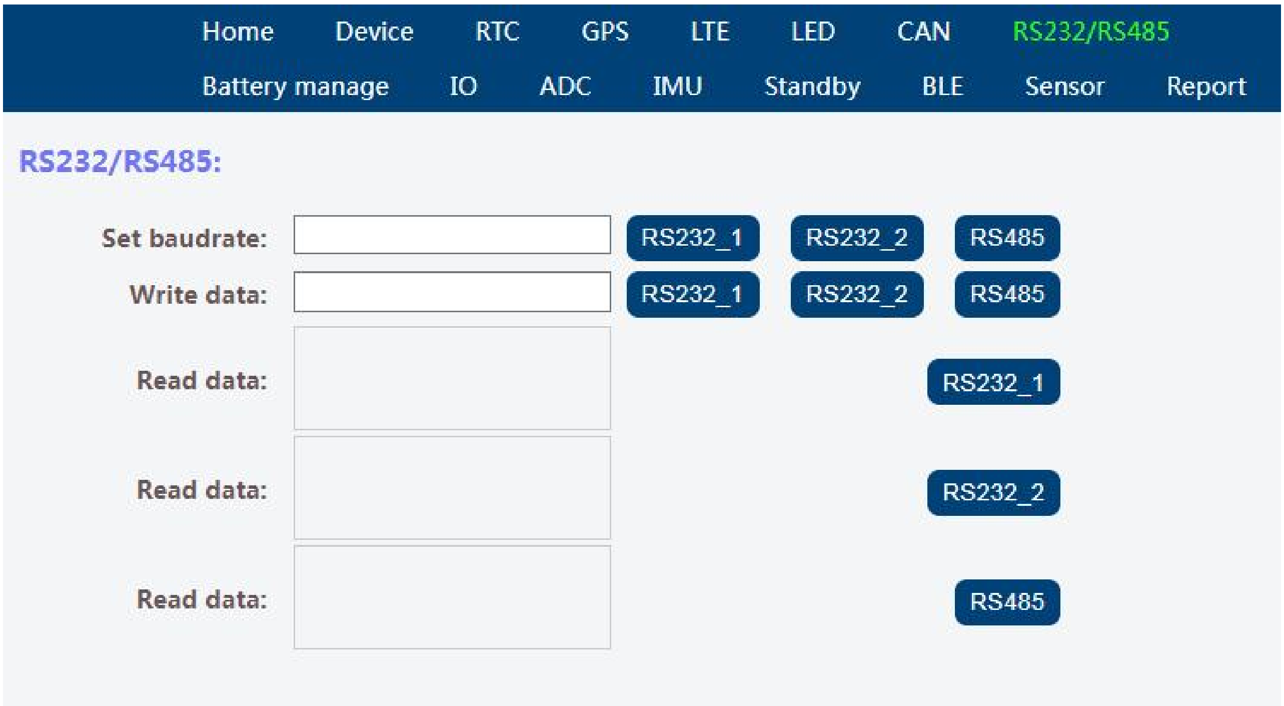

The LED subpage is used for LED testing. Clicking a switch can control the corresponding LED light.




The CAN subpage is used to test the data transmission and reception of the device CAN module.



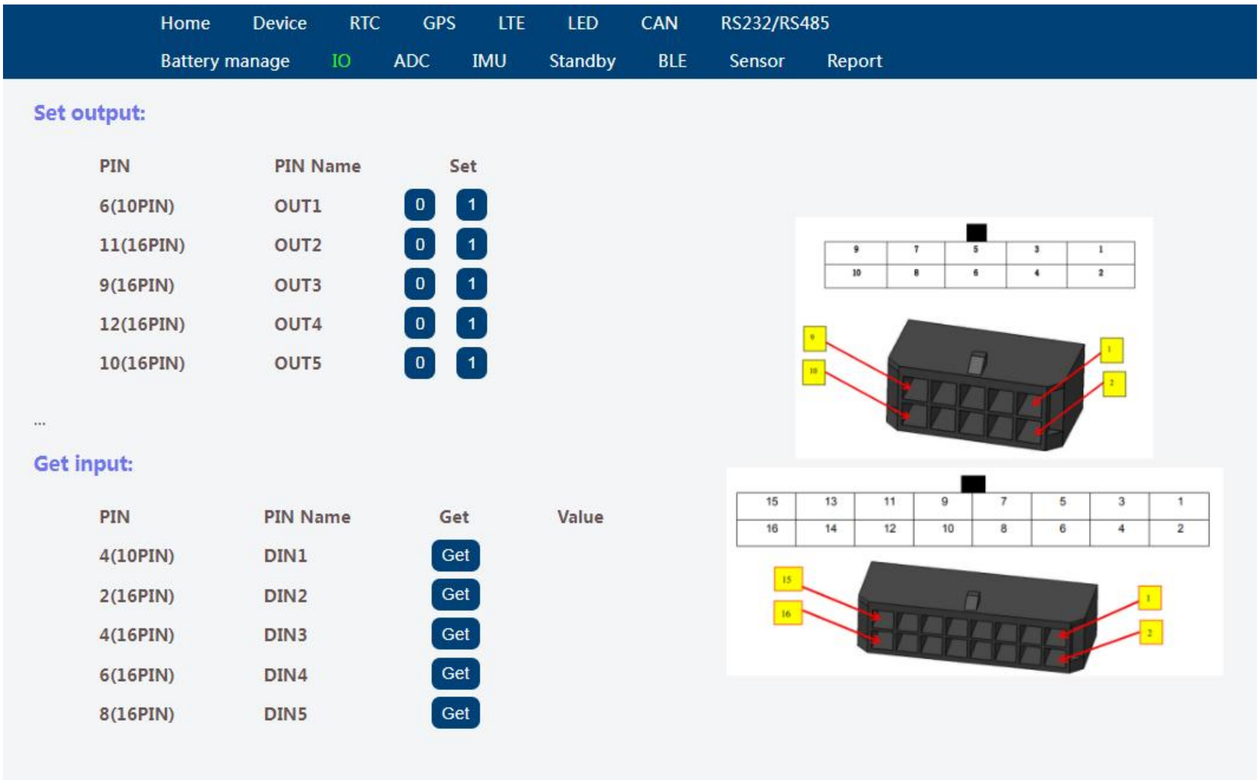

The RS232/RS485 subpage is used for RS485/232 port configuration and data transmission and reception test.



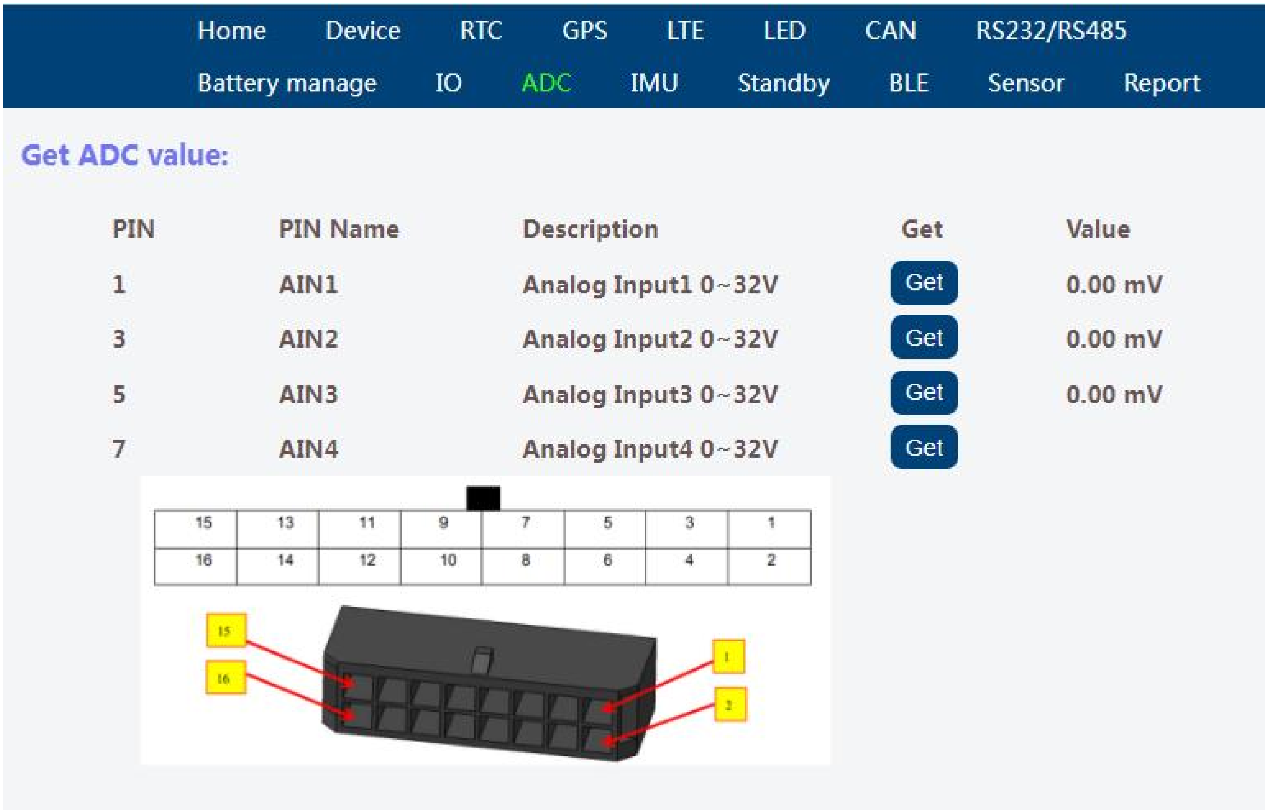

The Battery Manager sub-page is as follows. Since reading the device battery voltage will cause the battery to discharge, no data will be read after entering the page. You can click the corresponding button to test the corresponding function.


After clicking the "Start refresh" button on the Battery manager subpage, the page data will be automatically refreshed as follows:

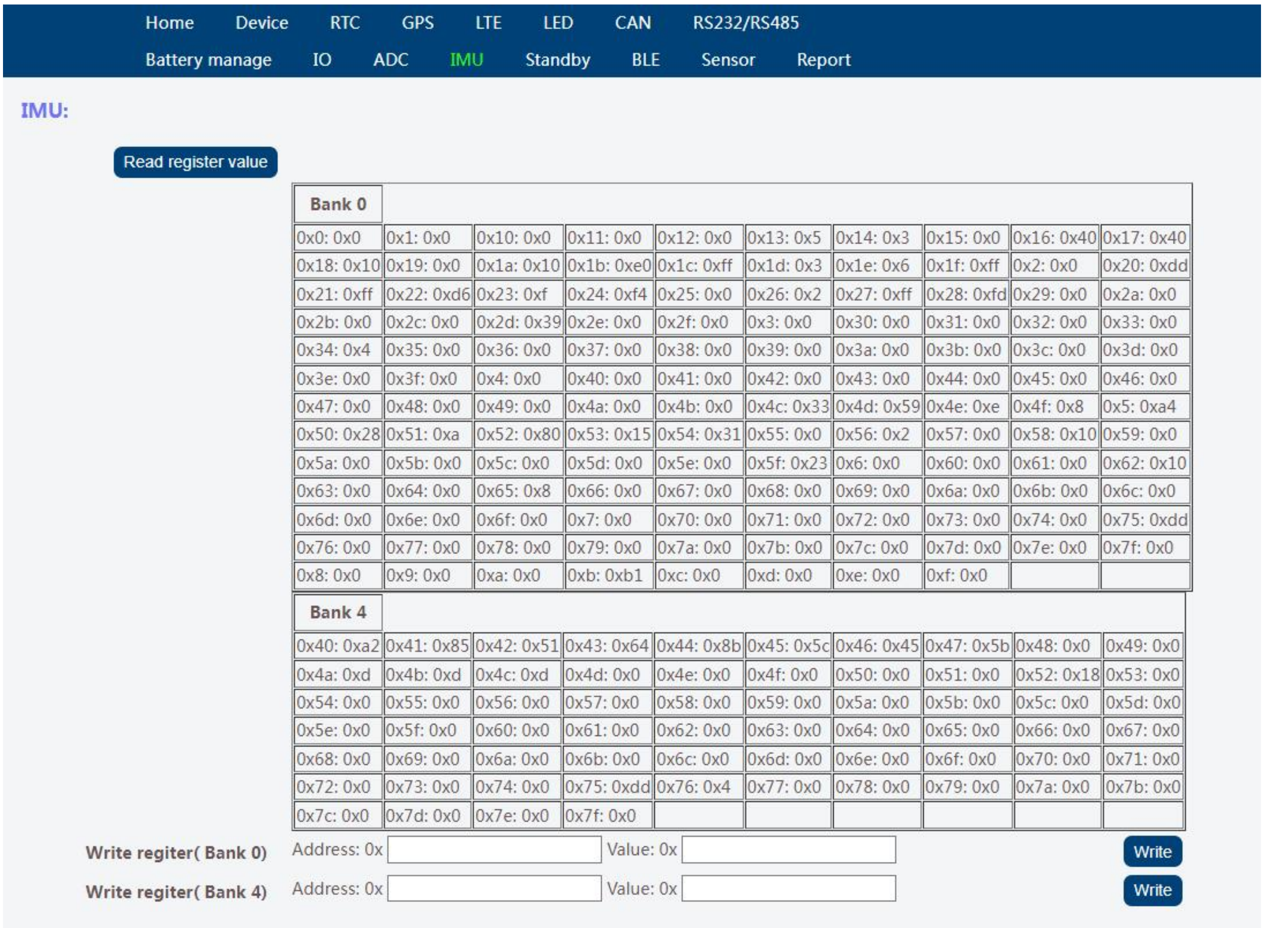

The IO sub-page can be used to set the device OUT terminal and read the DIN terminal.




The ADC subpage can be used to read the voltage value of the device's AIN terminal.



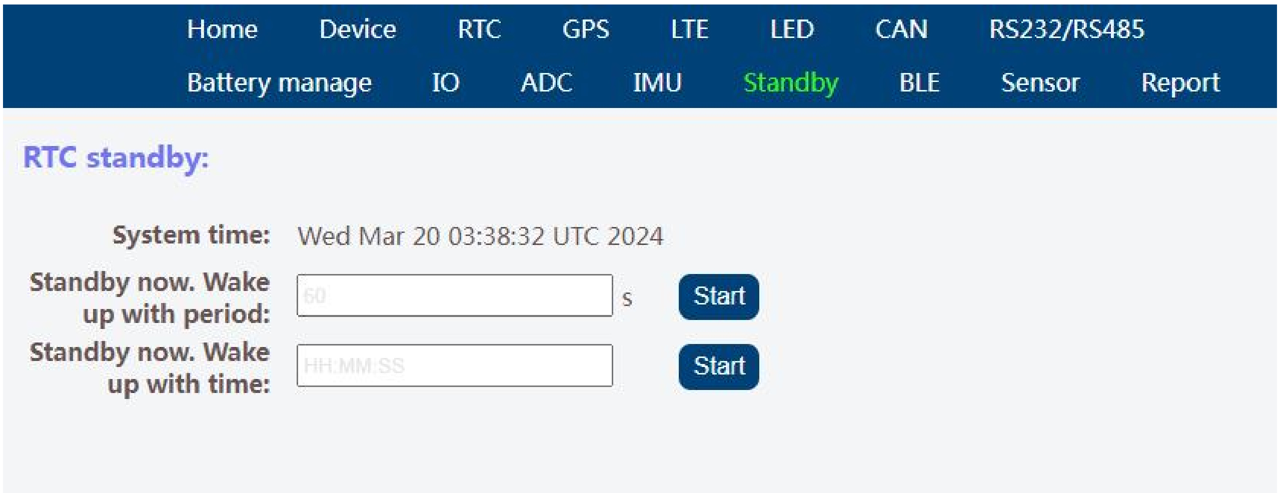

The IMU subpage can be used to read and set the values of the device's G-Sensor module registers.




| Bank 0 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0x0: 0x0 | 0x1: 0x0 | 0x10: 0x0 | 0x11: 0x0 | 0x12: 0x0 | 0x13: 0x5 | 0x14: 0x3 | 0x15: 0x0 | 0x16: 0x40 | 0x17: 0x40 |
| 0x18: 0x10 | 0x19: 0x0 | 0x1a: 0x10 | 0x1b: 0xe0 | 0x1c: 0xff | 0x1d: 0x3 | 0x1e: 0x6 | 0x1f: 0xff | 0x2: 0x0 | 0x20: 0xdd |
| 0x21: 0xff | 0x22: 0xd6 | 0x23: 0xf | 0x24: 0xf4 | 0x25: 0x0 | 0x26: 0x2 | 0x27: 0xff | 0x28: 0xfd | 0x29: 0x0 | 0x2a: 0x0 |
| 0x2b: 0x0 | 0x2c: 0x0 | 0x2d: 0x39 | 0x2e: 0x0 | 0x2f: 0x0 | 0x3: 0x0 | 0x30: 0x0 | 0x31: 0x0 | 0x32: 0x0 | 0x33: 0x0 |
| 0x34: 0x4 | 0x35: 0x0 | 0x36: 0x0 | 0x37: 0x0 | 0x38: 0x0 | 0x39: 0x0 | 0x3a: 0x0 | 0x3b: 0x0 | 0x3c: 0x0 | 0x3d: 0x0 |
| 0x3e: 0x0 | 0x3f: 0x0 | 0x4: 0x0 | 0x40: 0x0 | 0x41: 0x0 | 0x42: 0x0 | 0x43: 0x0 | 0x44: 0x0 | 0x45: 0x0 | 0x46: 0x0 |
| 0x47: 0x0 | 0x48: 0x0 | 0x49: 0x0 | 0x4a: 0x0 | 0x4b: 0x0 | 0x4c: 0x33 | 0x4d: 0x59 | 0x4e: 0xe | 0x4f: 0x8 | 0x5: 0xa4 |
| 0x50: 0x28 | 0x51: 0xa | 0x52: 0x80 | 0x53: 0x15 | 0x54: 0x31 | 0x55: 0x0 | 0x56: 0x2 | 0x57: 0x0 | 0x58: 0x10 | 0x59: 0x0 |
| 0x5a: 0x0 | 0x5b: 0x0 | 0x5c: 0x0 | 0x5d: 0x0 | 0x5e: 0x0 | 0x5f: 0x23 | 0x6: 0x0 | 0x60: 0x0 | 0x61: 0x0 | 0x62: 0x10 |
| 0x63: 0x0 | 0x64: 0x0 | 0x65: 0x8 | 0x66: 0x0 | 0x67: 0x0 | 0x68: 0x0 | 0x69: 0x0 | 0x6a: 0x0 | 0x6b: 0x0 | 0x6c: 0x0 |
| 0x6d: 0x0 | 0x6e: 0x0 | 0x6f: 0x0 | 0x7: 0x0 | 0x70: 0x0 | 0x71: 0x0 | 0x72: 0x0 | 0x73: 0x0 | 0x74: 0x0 | 0x75: 0xdd |
| 0x76: 0x0 | 0x77: 0x0 | 0x78: 0x0 | 0x79: 0x0 | 0x7a: 0x0 | 0x7b: 0x0 | 0x7c: 0x0 | 0x7d: 0x0 | 0x7e: 0x0 | 0x7f: 0x0 |
| 0x8: 0x0 | 0x9: 0x0 | 0xa: 0x0 | 0xb: 0xb1 | 0xc: 0x0 | 0xd: 0x0 | 0xe: 0x0 | 0xf: 0x0 | | |

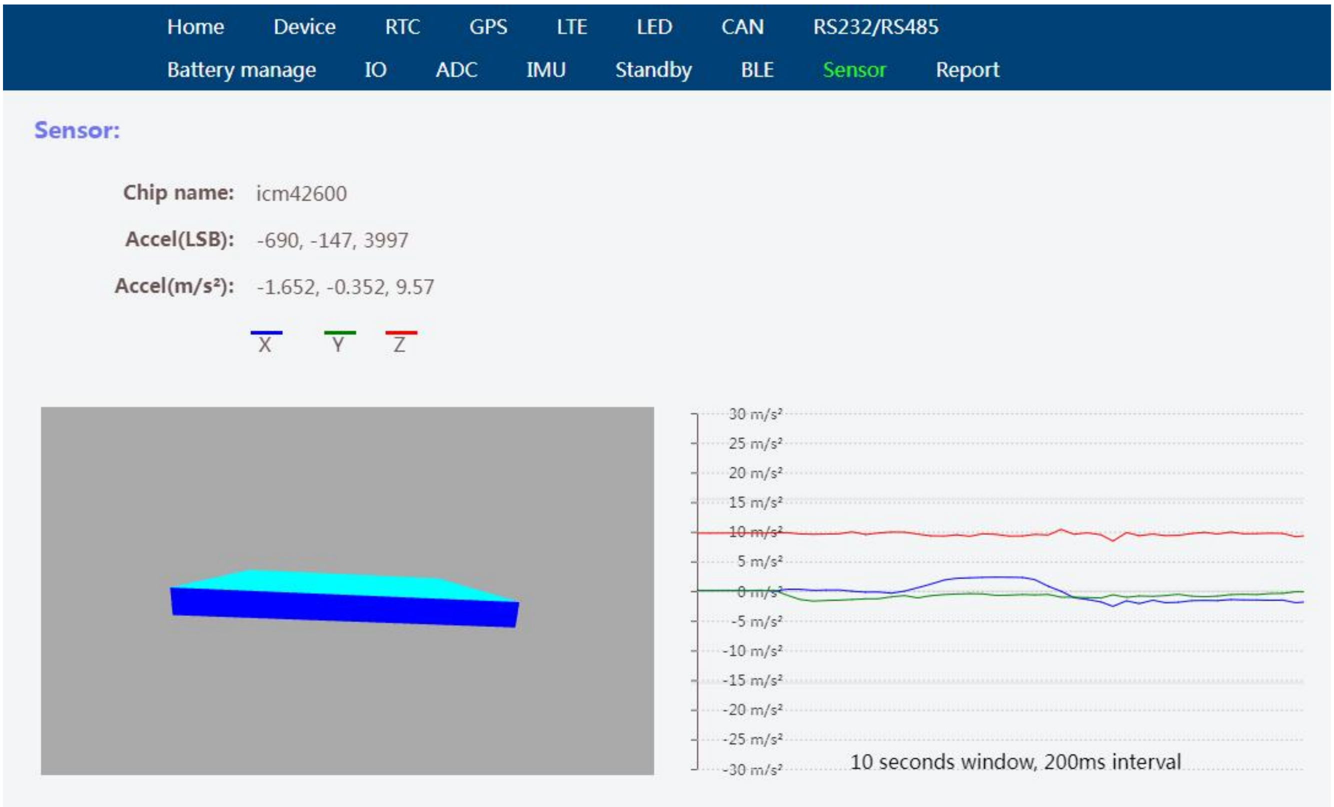
| Bank 4 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0x40: 0xa2 | 0x41: 0x85 | 0x42: 0x51 | 0x43: 0x64 | 0x44: 0x8b | 0x45: 0x5c | 0x46: 0x45 | 0x47: 0x5b | 0x48: 0x0 | 0x49: 0x0 |
| 0x4a: 0xd | 0x4b: 0xd | 0x4c: 0xd | 0x4d: 0x0 | 0x4e: 0x0 | 0x4f: 0x0 | 0x50: 0x0 | 0x51: 0x0 | 0x52: 0x18 | 0x53: 0x0 |
| 0x54: 0x0 | 0x55: 0x0 | 0x56: 0x0 | 0x57: 0x0 | 0x58: 0x0 | 0x59: 0x0 | 0x5a: 0x0 | 0x5b: 0x0 | 0x5c: 0x0 | 0x5d: 0x0 |
| 0x5e: 0x0 | 0x5f: 0x0 | 0x60: 0x0 | 0x61: 0x0 | 0x62: 0x0 | 0x63: 0x0 | 0x64: 0x0 | 0x65: 0x0 | 0x66: 0x0 | 0x67: 0x0 |
| 0x68: 0x0 | 0x69: 0x0 | 0x6a: 0x0 | 0x6b: 0x0 | 0x6c: 0x0 | 0x6d: 0x0 | 0x6e: 0x0 | 0x6f: 0x0 | 0x70: 0x0 | 0x71: 0x0 |
| 0x72: 0x0 | 0x73: 0x0 | 0x74: 0x0 | 0x75: 0xdd | 0x76: 0x4 | 0x77: 0x0 | 0x78: 0x0 | 0x79: 0x0 | 0x7a: 0x0 | 0x7b: 0x0 |
| 0x7c: 0x0 | 0x7d: 0x0 | 0x7e: 0x0 | 0x7f: 0x0 | | | | | | |

The Standby subpage is used to test the device's low power mode wake-up function. The page can be configured with a specified duration and a specified time point to wake the device from low power mode.



The BLE subpage is used to read the basic information of the BLE module.


The Sensor subpage is used to read the value of the G-Sensor gravity acceleration XYZ and demonstrate the real-time angle of the device.




The Report subpage is used to set the TCP server parameters. The server is used for receiving the device's active report messages.