USR IOT
Be honest,Do best

USR-G809 OpenCPU User Manual

Build a Smarter IoT world, Your Trustworthy Partner

Content

1. Brief introduction of USR-G809

USR-G809 is an industrial 4G router gateway that integrates 4G LTE, DIDO, serial, Ethernet ports (4LAN and 1WAN), and other interfaces. It combines 4G router, serial server, and IO control, making it a fully featured LTE router designed specifically for mission-critical IIoT applications.

1.1. Hardware specification

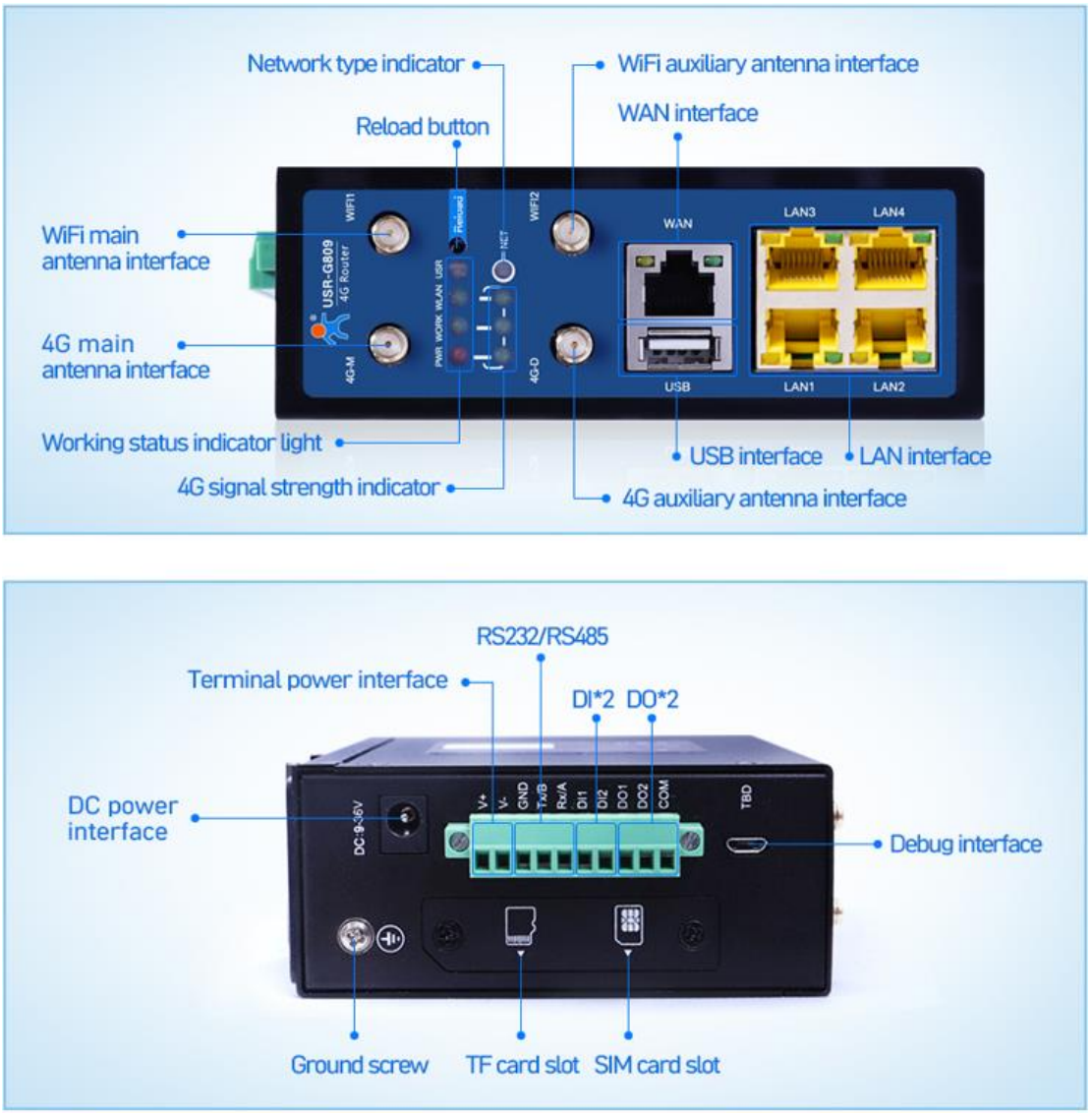
Hardware Specifications		
Cellular Interface	Antenna	2 x SMA-K Note: North America models: 2 x SMA 4G antenna connectors.
	SIM card	1 x (3 V & 1.8 V) Standard 2FF SIM, drawer-type sim card slots
Ethernet	1 x WAN port 10/100 Mbps, compliance with IEEE 802.3, supports auto MDI/MDIX,1.5KV network isolation transformer protection	
	4 x LAN ports, 10/100 Mbps, compliance with IEEE 802.3, IEEE 802.3u standards, supports auto MDI/MDIX,1.5KV network isolation transformer protection	
Indicators	LED	PWR: red, always on after powered on Work: green, blinking when the router is ready and working properly WLAN: green, always solid on when Wi-Fi is enabled and working properly USR: user defined. Net indicator: Red always on after connected to 2G network Green always on after connected to 3G network Orange always on after connected to 4G network Signal strength：3 solid bars, strongest signal
Terminal block	Pinout	V+,V-：2 cores terminal power supply socket, built-in power supply phase-reversal protection GND: ground terminal Tx/B:RS-232/RS485 pin（setup by software） Rx/A:RS-232/RS485 pin（setup by software） DI:2 x Digital input, passive switch DO:2 x Digital open collector output, max output 36 V, 300 mA COM: common terminal, use in conjunction with DOs
Wi-Fi Interface	Standards & Frequency	IEEE 802.11b/g/n,2.4GHz, AP mode
	Data speed	IEEE 802.11b/g, maximum 54Mbps.IEEE 802.11n, maximum 150Mbps
	Antenna	2 x RP-SMA-K
	Transmission distance	80 meters by line of sight. Actual transmission distance depends on environment of the site.
Power Supply	Adaptor	DC 12V/1A
	Connector	DC Power Jack Barrel Type Female 5.5*2.1mm Round socket or industrial terminal block, reverse polarity protection
	Input voltage	DC 9~36V

	Working power	Average 522mA/12V, Maximum 811mA/12V
Serial Interface	RS485/RS232(alternative)	industrial terminal block. Note:RS232 default.
	Baud rate(bps)	1200,2400,4800,9600,19200,38400,57600,115200,230400
	Data bits	8
	Stop bits	1,2
	Parity	NONE,ODD,EVEN
Physical Characteristics	Housing	Metal shell,IP30
	Dimensions	125.0*103.0*45.0mm (L*W*H, antenna pedestal, terminal block and DIN Rail are not included)
	Installation method	Ear mounting, DIN-Rail mounting
	EMC	Static IEC61000-4-2, level 3 Pulsed Electric Field IEC61000-4-4, level 3 Surge IEC61000-4-5, level 3
	Operating temperature	-20℃ ~ +70℃
	Storage temperature	-40℃ ~ +125℃
	Operating humidity	5% ~ 95%RH（non-condensing）
Others	Reload	Pinhole button, restore factory defaults/firmware resume/firmware upgrading with USB
	TBD	Debug interface (TTL Level)
	USB	Firmware upgrading
	TF	TF card slot
	Ground protection	Screw
	Embedded Watchdog	Device runs self-detection, auto recovers from malfunctions
Certificate	CE	

1.2. Download SDK

SDK download: **https://github.com/JinanUSR-IOT/openwrt**

1.3. Interface



1.4. Hardware resources

1>The main control chip is MT7620A, with 128M memory and 32M storage.

2>It has 1*100M WAN port, 4*100M LAN ports, and supports 5G WIFI (reserved) compatible with MT7612E and mainstream 4G modules.

1.4.1. The description of indicator lights

Item	Description
PWR	Power indicator, always on red after powered on.
WORK	Work indicator, 1 sec blink after booting.
WLAN	Wi-Fi indicator, always on green when Wi-Fi is enabled and working properly.
USR	User-defined indicator, can be set via the webpage(socket, VPN...).

NET	Always on after connecting to the network. Two colors indicate 4G network, green indicates 3G and red indicates 2G.
SIG(1-3)	Signal strength indicator, the more lights on, the stronger the signal.

1.4.2. LED pin definitions

Name	GPIO	Label	Flag
PWR	-	-	-
WORK	GPIO#0	green:work	GPIO_ACTIVE_LOW
WLAN	GPIO#72	green:wlan	GPIO_ACTIVE_LOW
USR0	GPIO#61	red:usr0	GPIO_ACTIVE_LOW
USR1	GPIO#62	green:usr1	GPIO_ACTIVE_LOW
NET0	GPIO#33	red:mode0	GPIO_ACTIVE_LOW
NET1	GPIO#32	green:mode1	GPIO_ACTIVE_LOW
SIG0	GPIO#31	green:sgn0	GPIO_ACTIVE_LOW
SIG1	GPIO#35	green:sgn1	GPIO_ACTIVE_LOW
SIG2	GPIO#34	green:sgn2	GPIO_ACTIVE_LOW

1.4.3. Terminal definitions

Terminal interface	Description
V+, V-	Power interface, built-in anti-reverse protect
GND	Ground terminal
Tx/B	R5232 or RS485, can be set via webpage
Rx/A	R5232 or RS485, can be set via webpage
D11, DI2, D01, D02	DI/DO terminal interface
COM	Do loop terminal

1.4.4. Digital_IO definitions

Name	GPIO	Label	Flag
DI1	GPIO#24	gpio-in0	GPIO_ACTIVE_HIGH
DI2	GPIO#28	gpio-in1	GPIO_ACTIVE_HIGH
DO1	GPIO#29	dout0-ctl	GPIO_ACTIVE_HIGH
DO2	GPIO#30	dout1-ctl	GPIO_ACTIVE_HIGH

1.4.5. LTE module pin definitions

Name	GPIO	Label	Flag
LTE_POWER_CONTROL	GPIO#64	modem-power	GPIO_ACTIVE_HIGH
LTE_RESET	GPIO#25	modem-reset	GPIO_ACTIVE_HIGH

1.4.6. UART

Level	3.3V
Band rate	57600
Data bits	8
Parity	None
Stop bit	1
Flow control	None

2. Environment preparation

1>Install a Linux environment, such as the Ubuntu system, switch to the root user, and install software package dependencies.

```
apt-get install g++
apt-get install libncurses5-dev
apt-get install zlib1g-dev
apt-get install bison
apt-get install flex
apt-get install unzip
apt-get install autoconf
apt-get install gawk
apt-get install make
apt-get install gettext
apt-get install gcc
apt-get install binutils
apt-get install patch
```

```
apt-get install bzip2
apt-get install libz-dev
apt-get install asciidoc
apt-get install subversion
```

Note: Make sure to confirm that the software packages are installed successfully. After all installations are completed, exit the root user and switch back to a regular user.

2>Create an "openwrt" folder and take out the G809 source code from Git. After downloading the code, enter the root directory of the source code.

```
git clone https://github.com/USR-IOT/openwrt.git
```

3>Execute the following command to update the installation package.

```
./scripts/feeds update -a
./scripts/feeds install -a
```

After the updating installation package, the next step is to configure the SDK.

3. SDK Configuration

3.1. Configuration file

3.1.1. Use the configuration file provided by USR

Copy `package/USR/configs/USR-G809.config` into the root directory of the SDK, and rename it as `.config`.

Alternatively, you can load the package `package/USR/configs/USR-G809.config` by selecting `Load` in `make menuconfig.`

This will automatically load the configurations of USR into `make menuconfig`. Save and exit.

3.1.2. Custom configurations

Open the graphical configuration interface by running “make menuconfig”.

1>Select the CPU model:

Target System -> MediaTek Ralink MIPS

2>Select the CPU sub-model:

Subtarget -> MT7620 based boards

3>Select the specific router model:

Target profile -> USR USR-G809 32M

4>Select the pre-installed software:

Utilities -> Choose software provided by OpenWrt

Utilities -> USR Applications, choose demo software provided by USR

Save and exit.

4. Firmware compilation

Run make V=s to compile.

Note: After the first successful compilation, you can add "-jxx" after "make" to specify xx threads for simultaneous compilation, such as "make -j32 V=s".

The compiled bin file is located in the directory: bin/targets/ramips/mt7620/.

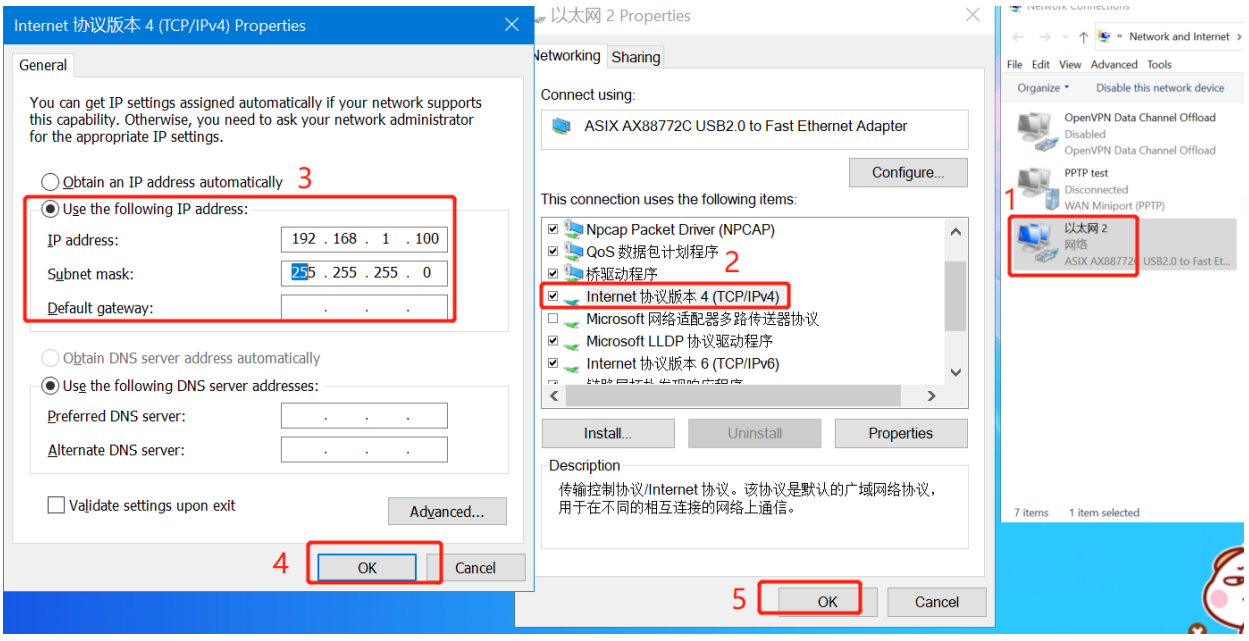
The complete firmware file is: `openwrt-ramips-mt7620-usr_usr-g809-squashfs-sysupgrade.bin`.

```
total 15M
-rw-r--r-- 1 1000 1000 2.0K Mar 15 13:36 config.buildinfo
-rw-r--r-- 1 1000 1000  288 Mar 15 13:36 feeds.buildinfo
-rw-r--r-- 1 1000 1000 7.0M Mar 15 13:37 openwrt-ramips-mt7620-usr_usr-g809-initramfs-kernel.bin
-rw-r--r-- 1 1000 1000 4.9K Mar 15 13:37 openwrt-ramips-mt7620-usr_usr-g809.manifest
-rw-r--r-- 1 1000 1000 7.3M Mar 15 13:37 openwrt-ramips-mt7620-usr_usr-g809-squashfs-sysupgrade.bin
drwxr-xr-x 1 1000 1000  12K Mar 15 13:37 packages
-rw-r--r-- 1 1000 1000 1.4K Mar 15 13:37 profiles.json
-rw-r--r-- 1 1000 1000  686 Mar 15 13:37 sha256sums
-rw-r--r-- 1 1000 1000   20 Mar 15 13:36 version.buildinfo
```

5. Firmware flashing

5.1. Firmware flashing via uboot

1> Configure the IP address of the computer, either statically or through DHCP. For static IP, use the following configuration as a reference:



2>To use tftpd32 to open the TFTP service and access the firmware directory.

3> Connect the USR-G809 to the computer via an Ethernet cable, either directly or through the same LAN.

4>Use a USB-TTL serial cable (supporting 3v3) to connect to the debug serial port of the device. Power on the device and quickly enter the number "2" on the keyboard, then enter "y". Next, modify the device IP to another IP in the same network segment as the computer, modify the server IP to the IP address of the computer, and enter the Linux Kernel filename as the name of the firmware. Finally, press enter and wait for the flashing process to complete.

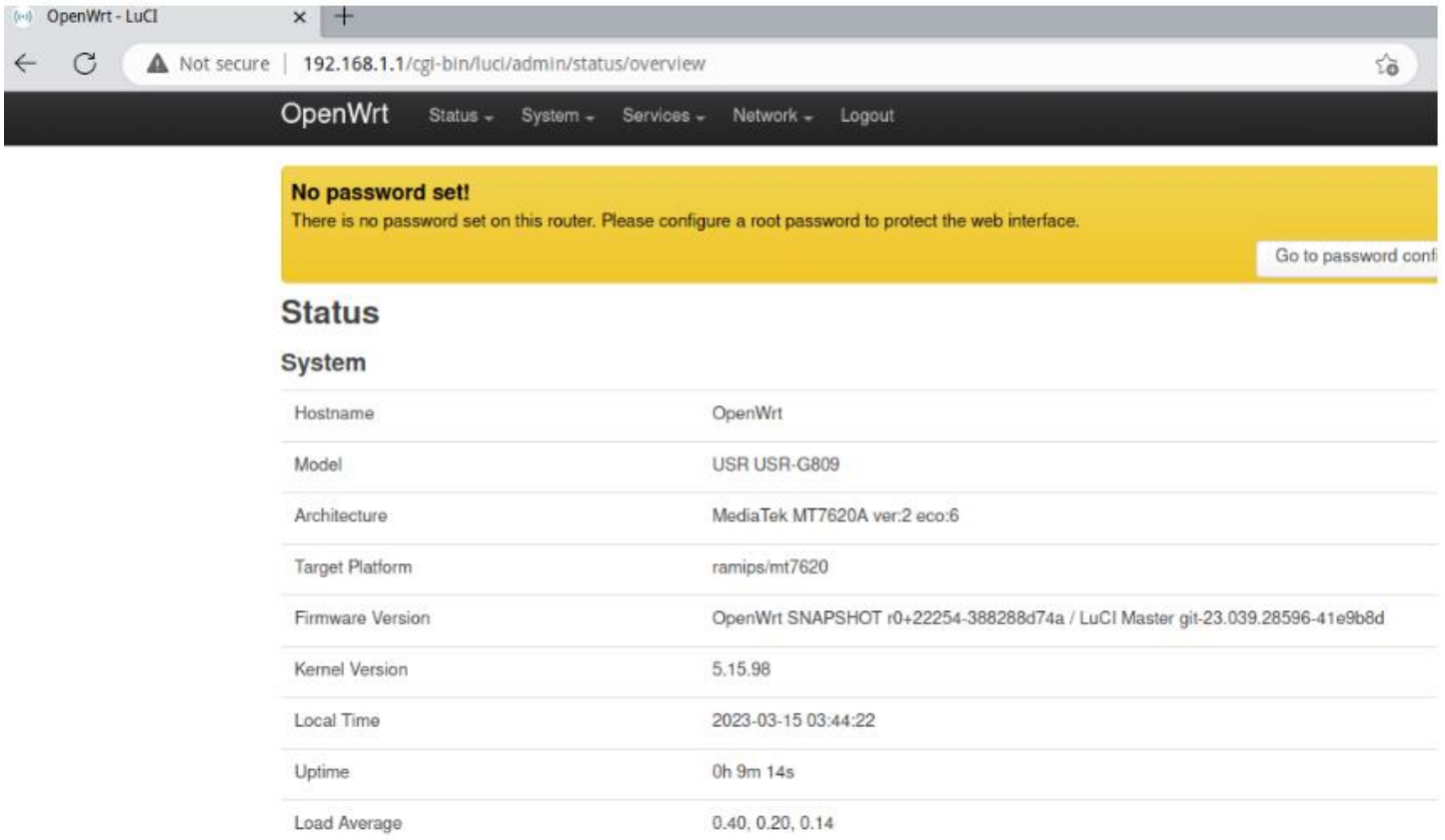
5>Wait for the system to start up and then access the built-in web page through the LAN port of the G809. The default username and password are both "root". See the image below for reference:

```
2: System Load Linux Kernel then write to Flash via TFTP.
 Warning!! Erase Linux in Flash then burn new one. Are you sure?(Y/N)
 Please Input new ones /or Ctrl-C to discard
       Input device IP (192.168.1.111) ==:192.168.1.111
       Input server IP (192.168.1.206) ==:192.168.1.206
       Input Linux Kernel filename () ==:openwrt-ramips-mt7620-usr_usr-g809-squashfs-sysupgrade.bin
```

```
Trying Eth0 (10/100-M)

 Waitting for RX_DMA_BUSY status Start... done


 ETH_STATE_ACTIVE!!
TFTP from server 192.168.1.206; our IP address is 192.168.1.111
Filename 'openwrt-ramips-mt7620-usr_usr-g809-squashfs-sysupgrade.bin'.

 TIMEOUT_COUNT=10,Load address: 0x80100000
Loading: Got ARP REPLY, set server/gtwy eth addr (04:d9:f5:84:fa:96)
Got it
#################################################################
         #################################################################
         #################################################################
         #################################################################
```

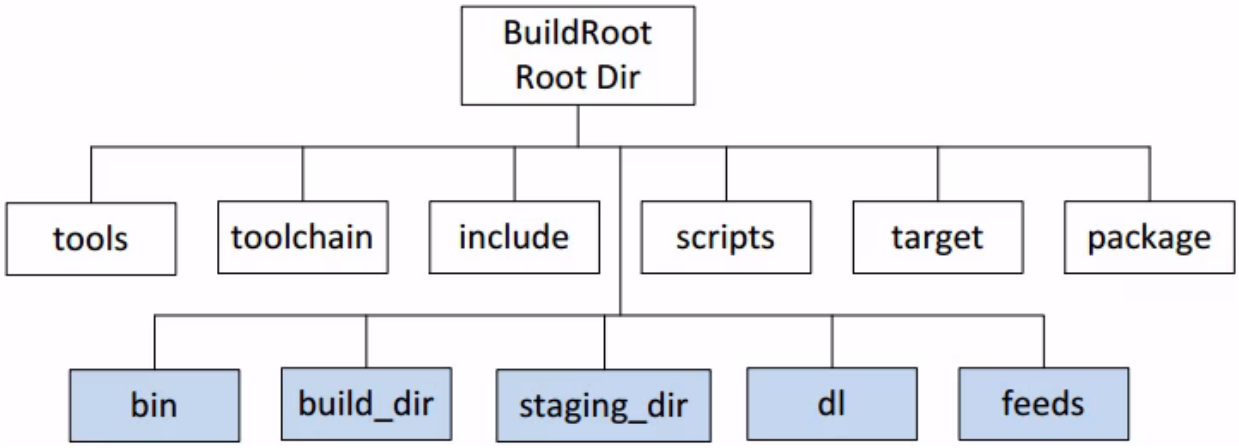
```
         #################################################################
         #################################################################
done
Bytes transferred = 8651563 (84032b hex)
NetBootFileXferSize= 0084032b
raspi_erase_write: offs:50000, count:84032b
raspi_erase: offs:50000 len:840000
................................................................................
....................
raspi_write: to:50000 len:840000
................................................................................
....................
raspi_read: from:50000 len:10000
raspi_read: from:60000 len:10000
```

```
raspi_read: from:890000 len:10000
Done!
raspi_write: to:401b0 len:3

## Booting image at bc050000 ...
raspi_read: from:50000 len:40
   Image Name:   MIPS OpenWrt Linux-5.15.98
   Image Type:   MIPS Linux Kernel Image (lzma compressed)
   Data Size:    2340245 Bytes =  2.2 MB
   Load Address: 80000000
   Entry Point:  80000000
raspi_read: from:50040 len:23b595
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
No initrd
## Transferring control to Linux (at address 80000000) ...
## Giving linux memsize in MB, 128

Starting kernel ...

[    0.000000] Linux version 5.15.98 (yanlufei@ubuntu_server) (mipsel-ope
 r22251-3e9005546a) 12.2.0, GNU ld (GNU Binutils) 2.40.0) #0 Tue Mar 14 0
[    0.000000] Board has DDR2
[    0.000000] Analog PMU set to hw control
[    0.000000] Digital PMU set to hw control
[    0.000000] SoC Type: MediaTek MT7620A ver:2 eco:6
```



6. SDK Introduction

6.1. OpenWrt Source Code Directory Introduction:



The above image shows the directory structure of OpenWrt. The top row shows the original directories, whi le the second row shows the directories generated during the compilation process.

Tools--During the compilation process, certain tools are required. The "tools" directory contains commands for obtaining and compiling these tools. It contains various Makefiles, some of which may include patches. Each Makefile includes the line "$(eval $(call HostBuild))", indicating that the tool is being compiled for use on the host.

Toolchain--This directory contains commands for obtaining kernel headers，C library, bin-utils, compiler, debugger.

Target--This directory contains the kernel configuration files and other platform-specific files.

Package--This directory contains the “Makefiles” for each software package. OpenWrt defines a set of “Makefile” templates, and each software package refers to this template to define its own information, such as the version number, download address, compilation method, installation address, and so on.

Include--The OpenWrt Makefiles are stored in this directory.

Scripts--There are some Perl scripts in this directory that are used for software package management.

Dl--This directory contains the downloaded source code for each package.

Build_dir--This directory contains the build output for each package.

Staging_dir--This is the final installation directory. The tools and toolchain will be installed here, and the rootfs will also be placed here.

Feeds--This directory contains the package feeds for OpenWrt.

Bin--After compilation, the firmware and “ipk” files will be placed in this directory.

6.2. SDK Introduction of USR-G809

This SDK is based on the official OpenWrt v22.03.03 and adds the USR's programs in the package/USR

directory. The specific functions of each feature can be found in 有人 demo 程序说明. `GobiNet`, `qmi_wwan_q` and `quectel-CM` are open source drivers and dial-up tools from Quectel, please refer to the Quectel official documentation for more details. The following are the source code directories for reference:

```
package/USR/
├── configs
│   ├── USR-G809.config
│   └── USR-G809.config.old
├── kernel
│   ├── GobiNet
│   │   ├── Makefile
│   │   └── src
│   │       └── ...
│   └── qmi_wwan_q
│       ├── Makefile
│       └── src
│           └── ...
├── libs
│   └── libusrdtu
│       ├── files
│       │   ├── usr_dtu
│       │   └── usr_dtu_service
│       ├── Makefile
│       └── src
│           ├── example
│           │   └── dtu_demo.c
│           ├── libusrdtu.so
│           ├── Makefile
│           └── usrdtu.h
├── README.md
└── utils
    ├── quectel-CM
    │   ├── Makefile
    │   └── src
    │       └── ...
    ├── usr_dialnet
    │   ├── files
    │   │   ├── cellular_config
    │   │   └── usr_dialnet_service
    │   ├── Makefile
    │   └── src
    │       ├── gpio_opt.c
    │       ├── gpio_opt.h
    │       ├── Makefile
    │       └── usr_dialnet.c
    └── usr_digital_io
```

```
        ├── files
        │   └── usr_digital_io.sh
        └── Makefile
```

6.3. Introduction of USR demo program:

6.3.1. DTU demo program

6.3.1.1. Function

The DTU connects an external MCU through an internal extended serial port (ttyS0), and the MCU provides a external serial port for serial data exchange. The system hardware watchdog function is also implemented.

6.3.1.2. Hardware source

Name	Options	Default
Baud rate	1200/2400/4800/9600/19200/38400/57600/115200/230400	115200
Data bits	8	8
Stop bit(s)	1/2	1
Parity	0: NONE 1: ODD 3: EVEN	NONE
Packaging time	10~60000ms	10ms
Packaging length	5~1500 bytes	1000
Serial mode	0: RS232 1: RS485	RS232
Flow control	0: NFC	NFC

6.3.1.3. Demo code introduction

1>The demo code is located in `package/USR/libs/libusrdtu`, which provides a binary library for serial port operations and a TCP client interaction example. It can be enabled by configuring `CONFIG_PACKAGE_libusrdtu=y` and `CONFIG_PACKAGE_dtu_demo=y`.

2>The configuration file is: `usr_dtu`

Network parameters:

Name	Item	Description
Server address	sa_server	Support IP address
Server port	sa_port	

Enable or forbidden	sa_enable	ON: enable OFF: forbidden

Source code directory

```
libusrdtu/
├── files                       #including configuration file and startup scripts
│   ├── usr_dtu                         # configuration file
│   └── usr_dtu_service                 # startup scripts
├── Makefile                            #for OpenWrt compilation
└── src                                         #source file and Makefile
    ├── example                         #
    │   └── dtu_demo.c          #dtu main program
    ├── libusrdtu.so            #dtu library file
    ├── Makefile                        #Makefile
    └── usrdtu.h                        #header file
```

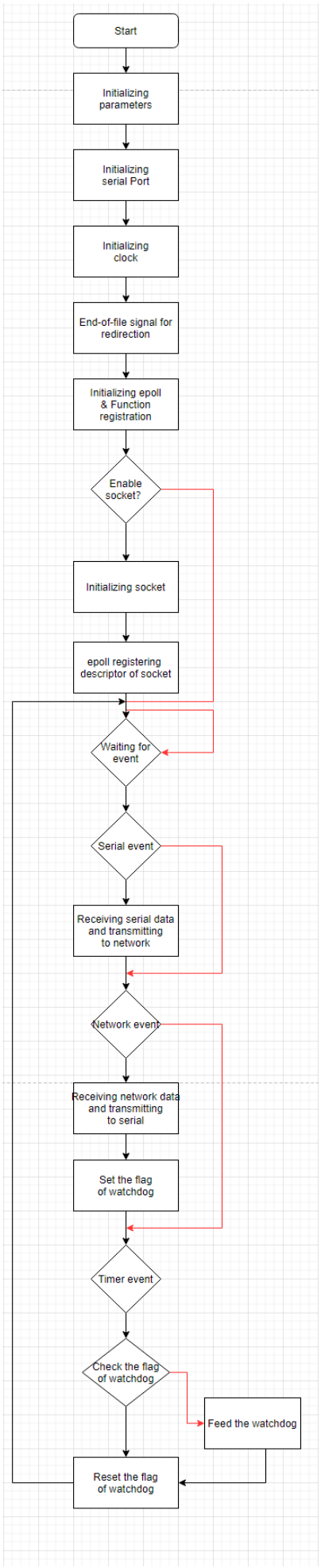
```
/* serial port parameter structure */
typedef struct _uci_param
{
    unsigned int baud; /*baud rate*/
    unsigned char parity_type; /*parity*/
    unsigned char data_bits; /*data bits*/
    unsigned char stop_bits; /*stop bit*/
    unsigned char flow_type; /*flow control*/
    unsigned char mode_type; /*serial port mode*/
    unsigned int pack_period; /*packaging time*/
    unsigned int pack_length; /* packaging length*/
    const char *devname; /*internal extended serial port(/dev/ttyS0)*/
    const char *hello_msg; /*initialization,string*/
} DTU_PARAM;
```

The library functions can be referred to the function comment description in the usrdtu.h file. The following a re brief descriptions of the relevant APIs:

```
/*********
 * @brief: serial port handle initialization, non-blocking
 * @param: dtu_config: DTU configuration
 * @return: file descriptor, if failed, return -1;
 * @modification: none
 *********/
int usrdtu_create(DTU_PARAM dtu_config);

/*********
 * @brief:destroy serial
 * @param: fd_serial: descriptor of serial file, created by usrdtu_create()
 * @return: If failed,return -1;
```

```
 * @modification: none
 *********/
int usrdtu_destroy(int fd_serial);

/*********
 * @brief: receive data, non-blocking
 * @param: fd: descriptor of serial file, created by usrdtu_create()
 * @param: data: buffer for storing data
 * @param: len: length of buffer for storing data
 * @return: reading data length, if failed, return -1
 * @modification: none
**********/
int usrdtu_rceive_data(int fd, char *data, int len);

/*********
 * @brief: send data
 * @param: fd: descriptor of serial file, created by usrdtu_create()
 * @param: data: buffer for storing data
 * @param: len: length of buffer for storing data
 * @return: sending data length, if failed, return -1
 * @modification: none
*********/
int usrdtu_send_data(int fd, char *buf, short len);

/***********
 * @brief: Serial port watchdog feeding
 * @param: fd: descriptor of serial file, created by usrdtu_create()
 * @return: None
 * @modification: none
***********/
   void usrdtu_dog(int fd);
```

The demo flow is roughly as follows:


Start
Initializing parameters
Initializing serial Port
Initializing clock
End-of-file signal for redirection
Initializing epoll & Function registration
Enable socket?
Initializing socket
epoll registering descriptor of socket
Waiting for event
Serial event
Receiving serial data and transmitting to network
Network event
Receiving network data and transmitting to serial
Set the flag of watchdog
Timer event
Check the flag of watchdog
Feed the watchdog
Reset the flag of watchdog

6.3.1.4. Function verification

1>Open the “NetAssist” on the computer and configure it as a TCP server.

2>Modify the DTU’s parameter configuration file, turn on TCP, and configure the address and port to the server 's address and port.

3>Computer and USR-G809 are connected via USB to 232 serial cable，and open the “UartAssist”.

4>Restart the DTU_demo program: `/etc/init.d/usr_dtu_service restart`

5>Check whether the device is connected. users can use the "NetAssist" and "UartAssist" to send data to each other.

6.3.2. Dialnet dialing demo program

6.3.2.1. Function Introduction

This is a cellular module dial-up program that identifies the module type and executes different dial-up procedures. The LED light displays the current network standard and signal strength after connecting to the in ternet. It can automatically restore the connection when disconnected.

6.3.2.2. Hardware Resource Introduction

The G809 uses the Mini PCIe interface to connect to the cellular module. The code has been adapted to EC25 and G405tf modules, and can support 2G, 3G, and 4G networks.

6.3.2.3. Code instruction

The code is located in `package/USR/utils/usr_dialnet`. Directory structure is as following:

```
usr_dialnet
├── files
│   ├── cellular_config            //configuration of dial-up software
│   └── usr_dialnet_service     //Start script
```

```
├── Makefile
└── src
    ├── gpio_opt.c                  //gpio control
    ├── gpio_opt.h
    ├── Makefile
        └── usr_dialnet.c             //dial-up program
```

`cellular_config`

This includes the configuration of the APN parameters. The built-in webpage can write the configuration to the file, and the `usr_dialnet.c` program reads the configuration file parameters when it starts.

```
config apn 'APN'
        option name ''
        option usr ''
        option paw ''
```

`usr_dialnet_service:`

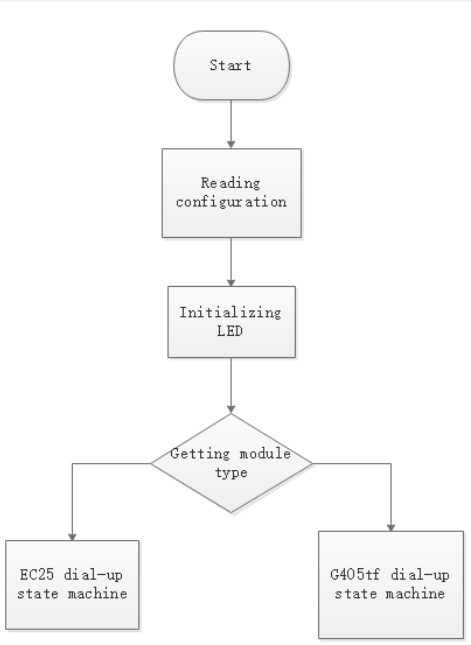
Auto-start script. In the OpenWrt system, the "init" process is replaced by "procd", which acts as the parent process and can monitor the status of the child process. Once the child process exits, it can attempt to restart the process at some point in time.

`gpio_opt.c`

This program is used to control the input/output mode and output high/low level of GPIO.

`usr_dialnet.c`

Dial-up program which includes the main function. The flow chat is as the following picture:



Configure the reading function

Upon entering the main function, the program first reads the APN parameters from the configuration file, which is stored in the `/etc/config` directory. The program uses uci (Unified Configuration Interface) for reading and writing.

```
    {
    char tmp_buf[128];
    memset(tmp_buf, 0, strlen(tmp_buf));
    shell_get_for_single("uci get cbi_file.APN.name", tmp_buf, sizeof(tmp_buf));
    if (0 == strlen(tmp_buf))
    {
        return 0;
    }
    else
    {
        memcpy(modem.apn.name, tmp_buf, strlen(tmp_buf));
        memset(tmp_buf, 0, strlen(tmp_buf));
        shell_get_for_single("uci get cbi_file.APN.usr", tmp_buf, sizeof(tmp_buf));
        if (0 == strlen(tmp_buf))
        {
            return 0;
        }
        memcpy(modem.apn.user, tmp_buf, strlen(tmp_buf));
        memset(tmp_buf, 0, strlen(tmp_buf));
        shell_get_for_single("uci get cbi_file.APN.paw", tmp_buf, sizeof(tmp_buf));
        if (0 == strlen(tmp_buf))
        {
            return 0;
        }
        memcpy(modem.apn.psw, tmp_buf, strlen(tmp_buf));
        memset(tmp_buf, 0, strlen(tmp_buf));
        // shell_get_for_single("uci get cbi_file.APN.auth", tmp_buf, sizeof(tmp_buf));
        // if (0 == strlen(tmp_buf))
        // {
        //     return 0;
        // }
        memcpy(modem.apn.auth, "1", strlen("1"));
    }
    }
```

Next, the program initializes the network standard and signal strength LED by setting the corresponding GPIO pins to output mode and setting them to a high level (turning off the LED).

```
    /*
LED of cellular network standards
*/
void module_net_led_control(int net_value)
{

    switch (net_value)
    {
    case LTE_MODE_2G: // 2G
        dialnet_setval(LED_MODE0, LED_ON);
```

```
        dialnet_setval(LED_MODE1, LED_OFF);
        break;

    case LTE_MODE_3G: // 3G
        dialnet_setval(LED_MODE0, LED_OFF);
        dialnet_setval(LED_MODE1, LED_ON);
        break;
    case LTE_MODE_4G: // 4G
        dialnet_setval(LED_MODE0, LED_ON);
        dialnet_setval(LED_MODE1, LED_ON);
        break;

    default: // others
        dialnet_setval(LED_MODE0, LED_OFF);
        dialnet_setval(LED_MODE1, LED_OFF);
        break;
    }
}

/*
LED of gpio
*/
void module_signal_led_control(int signal)
{
    ......
   }
```

The G809 currently supports two cellular modules, and the dialing process for these two modules differs slig htly. Therefore, it is necessary to distinguish the module type by using the module's PID and VID in the dialing application.

```
 /*
 * Getting module type.
 * return:  1, EC25,  2, G405TF -1, fail
 */
int lte_get_module_type()
{
    char s_usb_list_buf[MAX_BUF_SIZE] = {0};

    shell_get_for_single("lsusb | grep 2c7c:0125", s_usb_list_buf,
sizeof(s_usb_list_buf));
    if (strlen(s_usb_list_buf) > 0)
    {

        return EC25;
    }
```

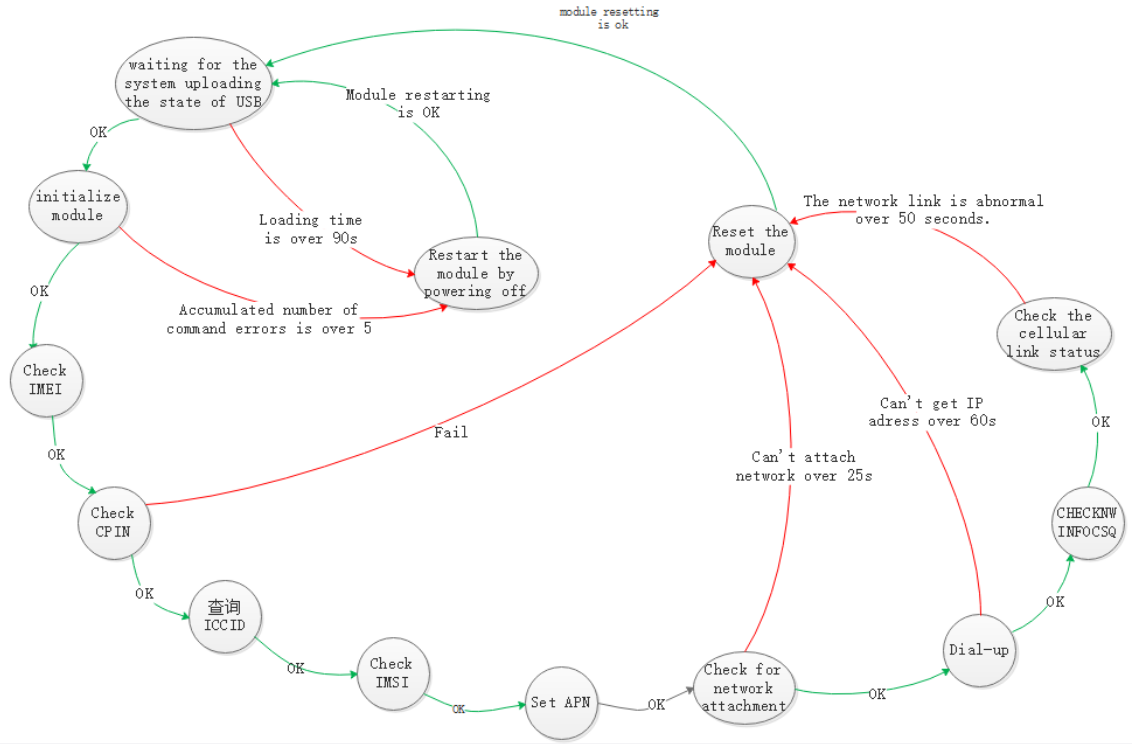
```
    shell_get_for_single("lsusb | grep 19d2:0579", s_usb_list_buf,
sizeof(s_usb_list_buf));
    if (strlen(s_usb_list_buf) > 0)
    {
        return G405TF;
    }

    return -1;
   }
```

After getting the module type, execute the corresponding resident network code according to the type.

Dial-up and network link maintenance are implemented using state machines, and the state definitions are as follows:

```
    typedef enum
{
    EFINDUSBSTA = 0, // waiting for the system uploading the state of USB
    ERESET,                      // reset the module
    POWEROFF,            // Restart the module by powering off
    EINITMODEL,          // initialize the module
    CHECKIMEI,           // check IMEI
    CHECKCPIN,           // check CPIN
    CHECKICCID,          // check ICCID
    CHECKIMSI,           // check IMSI
    CHECKAPN,            // set APN
    ESTICKNET,           // check for network attachment
    GOBINET,             // dial-up
    CHECKNWINFOCSQ,      // check the cellular network standard and signal strenth
    EGETNETSTA,          // check the cellular link status
   } e_model_sta;
```

The flow chat of state machine



Introduction to the Successful Network Residency Process

1. Enter the EFINDUSBSTA state, waiting for the recognition of the AT port ttyUSB.

2. Enter the EINITMODEL state, mainly initializing the serial port read-write file descriptor and turning on the RF switch.

3. Enter the CHECKIMEI state, read the module IMEI to determine if the module is normal.

4. Enter the CHECKCPIN, CHECKICCID, CHECKIMSI, states in turn, all of which are SIM status queries. If all of them pass, it indicates that the SIM card status is normal. The operator type can be obtained by parsing the IMSI f or subsequent setting of APN parameters.

5. Enter the CHECKAPN state, where it is judged whether the previously obtained APN parameter is empty (i.e. whether the user has manually set the APN parameter). If it is empty, use the default APN parameter, otherwise use the user's APN parameter.

6. Enter the ESTICKNET state, query the network attachment status by sending AT+CGATT?, and wait for successful network attachment.

7. Enter the GOBINET state, the EC25 module uses the quectel-CM tool for dial-up, and the dial-up Internet access can be completed by running quectel-CM in the background. The G405tf module uses AT commands for dial-up Internet access, sending AT+ZGACT=1,1 to connect to the RNDIS link. If +ZCONSTAT:1,1 is returned, it means that the network has been established, and then use udhcpc to obtain an IP.

8. Enter the CHECKNWINFOCSQ state. This step is to obtain the current network mode and signal value to update the LED status.

9. Enter the EGETNETSTA state. This step will cyclically detect network attachment status, IP status, DNS status, network mode, and signal value. When the network link status is abnormal, the module will be reset, and then dial-up will be performed again.

10. In addition, there are ERESET and POWEROFF two abnormal processing states. When the normal network residency process fails, it will jump to the abnormal processing state, and after the module is reset or restarted, it will jump to the EFINDUSBSTA state for re-residency.

6.3.2.4. Verification

Dial-up successfully of EC25 module

```
[03-14_04:13:37:572] QConnectManager_Linux_V1.6.4
[03-14_04:13:37:581] Find /sys/bus/usb/devices/1-1.1 idVendor=0x2c7c idProduct=0x125, bus=0x001, dev=0x003
[03-14_04:13:37:584] Auto find qmichannel = /dev/qcqmi0
[03-14_04:13:37:585] Auto find usbnet_adapter = usb0
[03-14_04:13:37:586] netcard driver = GobiNet, driver version = V1.6.3
[usr_dialnet.c : 674]: IP = [10.116.105.74]
[03-14_04:13:37:587] Modem works in QMI mode
[03-14_04:13:37:669] Get clientWDS = 9
[03-14_04:13:37:733] Get clientDMS = 10
[03-14_04:13:37:861] Get clientNAS = 11
[03-14_04:13:37:893] Get clientUIM = 12
[03-14_04:13:37:925] Get clientWDA = 13
[03-14_04:13:37:957] requestBaseBandVersion EC20CEFAGR06A05M4G  1  [Jul 13 2017 22:00:00]
[03-14_04:13:38:085] requestGetSIMStatus SIMStatus: SIM_READY
[03-14_04:13:38:118] requestGetProfile[1] 3GNET/1111/1111/1/IPV4V6
[03-14_04:13:38:150] requestRegistrationState2 MCC: 460, MNC: 1, PS: Attached, DataCap: LTE
[03-14_04:13:38:182] requestQueryDataCall IPv4ConnectionStatus: DISCONNECTED
[03-14_04:13:38:182] ifconfig usb0 0.0.0.0
[03-14_04:13:38:197] ifconfig usb0 down
[03-14_04:13:38:278] requestSetupDataCall WdsConnectionIPv4Handle: 0x871eba50
[03-14_04:13:38:438] ifconfig usb0 up
[03-14_04:13:38:454] busybox udhcpc -f -n -q -t 5 -i usb0
udhcpc: started, v1.36.0
udhcpc: broadcasting discover
udhcpc: broadcasting select for 10.116.105.74, server 10.116.105.73
udhcpc: lease of 10.116.105.74 obtained from 10.116.105.73, lease time 7200
[03-14_04:13:38:492] udhcpc: ip addr add 10.116.105.74/255.255.255.252 broadcast + dev usb0
[03-14_04:13:38:505] udhcpc: setting default routers: 10.116.105.73
[usr_dialnet.c : 1676]: cellular state:11
```

```
root@OpenWrt:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=111 time=220.966 ms
64 bytes from 8.8.8.8: seq=1 ttl=111 time=161.018 ms
64 bytes from 8.8.8.8: seq=2 ttl=111 time=168.459 ms
64 bytes from 8.8.8.8: seq=3 ttl=111 time=434.730 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 161.018/246.293/434.730 ms
```

Dial-up successfully of G405tf module

```
udhcpc: started, v1.33.2
udhcpc: sending discover
udhcpc: sending discover
udhcpc: sending select for 10.132.73.191
udhcpc: lease of 10.132.73.191 obtained, lease time 86400
```

```
root@OpenWrt:/# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=111 time=345.479 ms
64 bytes from 8.8.8.8: seq=1 ttl=111 time=446.913 ms
64 bytes from 8.8.8.8: seq=2 ttl=111 time=135.992 ms
64 bytes from 8.8.8.8: seq=3 ttl=111 time=165.869 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 135.992/273.563/446.913 ms
```

6.3.3. USB mounting

6.3.3.1. Configuration of make menuconfig

Add USB support

Kernel modules –>

USB Support –>

<*> kmod-usb-core. ## the default

<*> kmod-usb-ohci. ## the default is old usb1.0

<*> kmod-usb-uhci. ## usb1.1

<*> kmod-usb-storage.

<*> kmod-usb-storage-extras.

<*> kmod-usb2. ## usb2.0

Add SCSI support.

Kernel modules –>

Block Devices –>

<*>kmod-scsi-core ## usb3.0

Add USB mounting.

Base system –>

<*>block-mount

Add file system support.

Kernel modules –>

Filesystems –>

<*> kmod-fs-ext4 (Select EXT4)

<*> kmod-fs-vfat(Select FAT16 / FAT32)

```
        <*> kmod-fs-ntfs (Select NTFS)
```

Save and exit. Make V=99

6.3.3.2. Add automatically mount scripts of U drive.

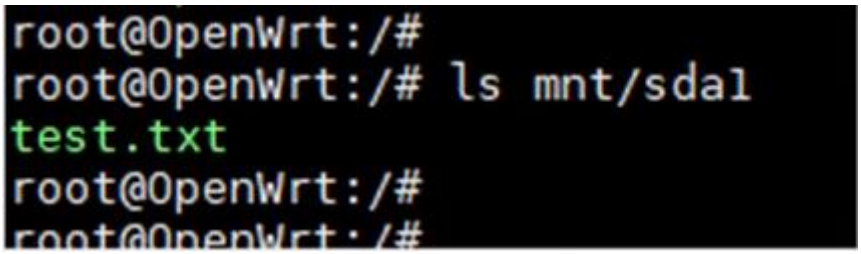
Create a file named `11-external_storage_mount` under `target/linux/ramips/mt7620/base-files/etc/hotplug.d/block/` and write the following content:

```
   #!/bin/ash

board=$(board_name)

case "$board" in
    usr,usr-g809)
    case "$ACTION" in
    add)
        for i in $(ls /dev/ | grep 'sd[a-z][1-9]')
        do
            mkdir -p /mnt/$i
            mount -o iocharset=utf8,rw /dev/$i /mnt/$i
            if [ $? -ne 0 ]
            then
                mount -o rw /dev/$i /mnt/$i
            fi
        done
        ;;
    remove)
        MOUNT=`mount | grep -o '/mnt/sd[a-z][1-9]'`
        for i in $MOUNT
        do
            umount $i
            if [ $? -eq 0 ]
            then
                rm -r $i
            fi
        done
        ;;
    esac
    ;;
   esac
```

6.3.3.3. Test the script:

Connect USB drive to G809, and USB drive is mounted under `/mnt/sdx` dictionary.

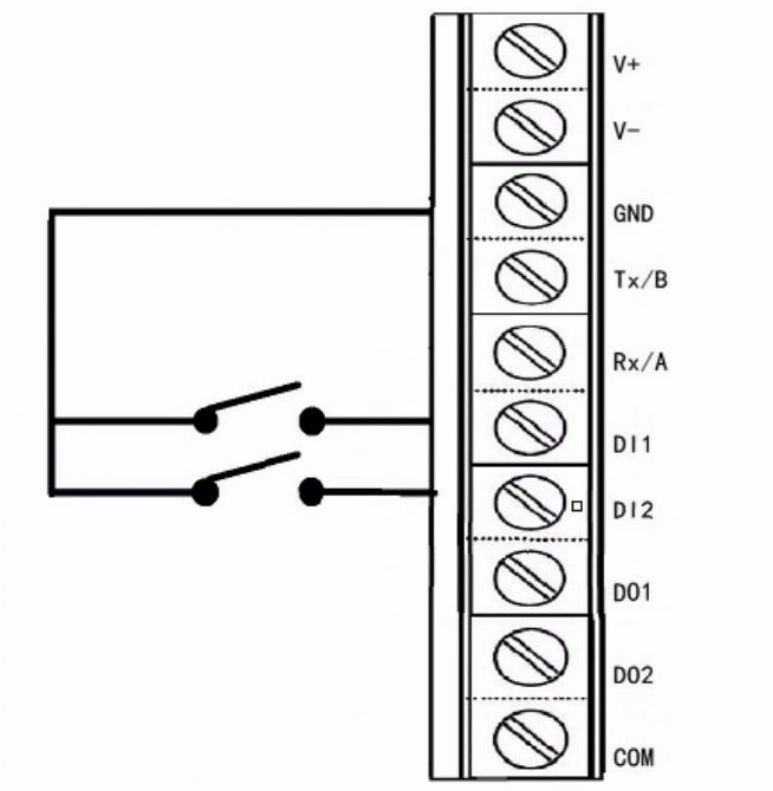
List the content of `/mnt/sda1` dictionary.



6.3.4. Instruction of digital_io program(package/USR/utils/usr_digital_io)

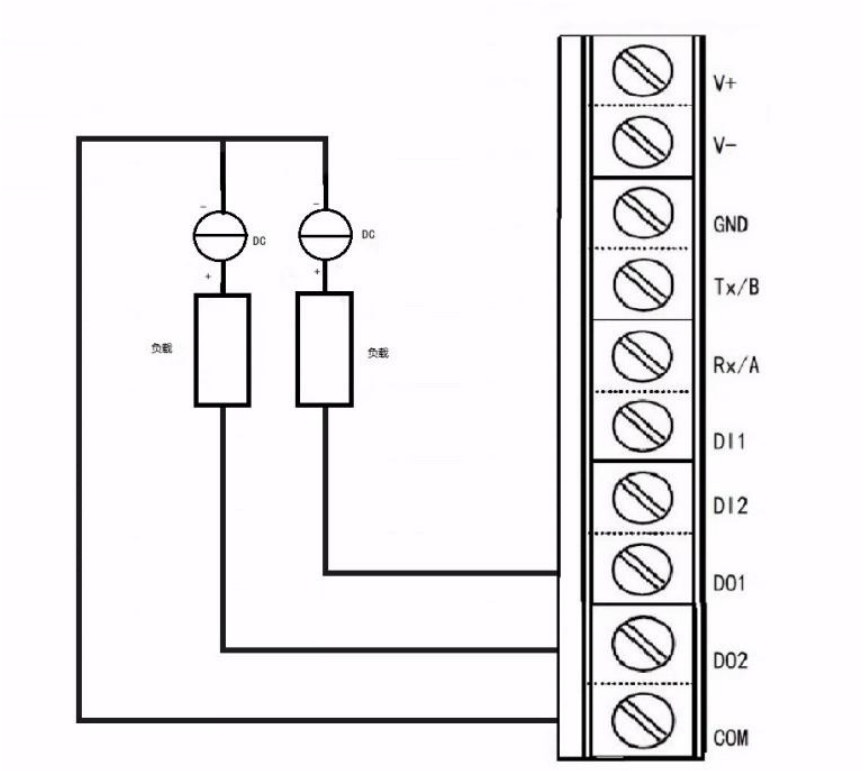
1.The digital_io demo program is used to demonstrate the usage of the DI/DO interface. The DI interface is used to receive external high/low level signals, while the DO interface is used to connect/disconnect the COM interface.

2.The pins used for DI have been defined in the dts file. Specifically, `gpio-in0` corresponds to DI1 and `gpio-in1` corresponds to DI2. The configuration of the active level can be viewed by reading/writing the files `/sys/class/gpio/gpio-in0/active_low` and `/sys/class/gpio/gpio-in1/active_low`. `1` indicates that the active level is high, while `0` indicates that the active level is low. The current level of DI can be obtained by reading the files `/sys/class/gpio/gpio-in0/value` and `/sys/class/gpio/gpio-in1/value` which allows for the detection of corresponding signals. The wiring diagram for DI is shown below:



3.The pins used for DO have been defined in the dts file. Specifically, `dout0-ctl` corresponds to DO1 and `dout1-ctl` corresponds to DO2. The connection to COM for DI1 and DI2 can be configured by writing to the files `/sys/class/gpio/dout0-ctl/value` and `/sys/class/gpio/dout1-ctl/value`. `1` indicates that the connection is closed, while `0` indicates that the connection is open.

The wiring diagram for DO is shown below:



4.LED_USR is a user-defined LED indicator that consists of a bi-color (red and green) LED.

It is controlled by two GPIO pins which have been configured as an LED indicator in the dts file.

Specifically, `red:usr0` corresponds to LED_USR1, `green:usr1` corresponds to LED_USR2.

The status of the LED indicator can be obtained and controlled by reading/writing to the files `/sys/class/leds/red:usr0/brightness` and `/sys/class/leds/green:usr1/brightness.`

5.The following is the demo code of digital_io.

```
#!/bin/sh

# High level
DIN1_TIGGER=1

# Low level
DIN2_TIGGER=0

DIN1_LABEL="/sys/class/gpio/gpio-in0"
DIN2_LABEL="/sys/class/gpio/gpio-in1"

DOUT1_LABEL="/sys/class/gpio/dout0-ctl"
DOUT2_LABEL="/sys/class/gpio/dout1-ctl"

LED_USR1="/sys/class/leds/red:usr0/"
LED_USR2="/sys/class/leds/green:usr1/"

# Active high
if [ -e "${DIN1_LABEL}" ]; then
    echo ${DIN1_TIGGER} > "${DIN1_LABEL}/active_low"
fi

# Active low
if [ -e "${DIN2_LABEL}" ]; then
    echo ${DIN2_TIGGER} > "${DIN2_LABEL}/active_low"
fi

echo 0 > "${LED_USR1}/brightness"
echo 0 > "${LED_USR2}/brightness"

while true
do
    DIN1_VALUE=$(cat ${DIN1_LABEL}/value)
    DIN2_VALUE=$(cat ${DIN2_LABEL}/value)

    if [ "${DIN1_VALUE}" == "${DIN1_TIGGER}" ]; then
        echo 1 > "${DOUT1_LABEL}/value"
        echo 1 > "${LED_USR1}/brightness"
    else
```

```
        echo 0 > "${DOUT1_LABEL}/value"
        echo 0 > "${LED_USR1}/brightness"
    fi

    if [ "${DIN2_VALUE}" == "${DIN2_TIGGER}" ]; then
        echo 1 > "${DOUT2_LABEL}/value"
        echo 1 > "${LED_USR2}/brightness"
    else
        echo 0 > "${DOUT2_LABEL}/value"
        echo 0 > "${LED_USR2}/brightness"
    fi

    sleep 1
    # can also use `usleep` cmd with BUSYBOX_CONFIG_USLEEP=Y
    # usleep 100

   done
```

6.4. Explanation of modifications based on OpenWrt

6.4.1. target/linux/ramips/image/mt7620.mk

This file contains information such as firmware size, default software packages, and other details.

```
  define Device/usr_usr-g809
  SOC := mt7620a #Name of DTS file (user-defined, it is consistent with the device model)
  IMAGE_SIZE := 32448k #Size of firmware
  DEVICE_VENDOR := USR #Provider
  DEVICE_MODEL := USR-G809  #Device's model
  DEVICE_VARIANT := 32M          #Size of flash
  DEVICE_PACKAGES := kmod-mt76x2 kmod-usb2 kmod-usb-ohci #needed packages of the device
  SUPPORTED_DEVICES += usr-g809
endef
  TARGET_DEVICES += usr_usr-g809
```

6.4.2. Explanation of modifications of dts file

The dts file is located at `target/linux/ramips/dts/mt7620a_usr_usr-g809.dts` and can be modified as needed. For example, it can be edited to add PCIe interface.

```
    &pcie {
    status = "okay";
};

&pcie0 {
    mt76@0,0 {
        reg = <0x0000 0 0 0 0>;
        mediatek,mtd-eeprom = <&factory 0x8000>;
        ieee80211-freq-limit = <5000000 6000000>;
```

```
    };
   };
```

Add rtc clock.

```
  &i2c {
    gpios = <&gpio0 1 GPIO_ACTIVE_LOW
             &gpio0 2 GPIO_ACTIVE_LOW>;
    status = "okay";
    pcf8563@51 {
        compatible = "nxp,pcf8563";
        reg = <0x51>;
    };
  };
```

Enable extended serial port.

```
   /* Configure the control serial port as ttyS1 */
chosen {
        bootargs = "console=ttyS1,57600";
       };
   /* Enable serial port */
&uart {
    status = "okay";
   };
   /* Configure the GPIO pins as UART*/
&state_default {
    uartf_gpio {
        ralink,group = "uartf";
        ralink,function = "gpio uartf";
    };
   };
```

7. How to add a package (using libusrdtu as an example)

```
usr@ubuntu:~/work/openwrt/openwrt/package/USR/libs$ tree libusrdtu/
libusrdtu/
├── files               #contains the configuration file and start script
│   ├── usr_dtu                         #configuration file
│   └── usr_dtu_service                 #startup script
├── Makefile                            #Makefile used for openwrt compilation
└── src                 # contains the source file of application and Makefile
    ├── example                         #
    │   └── dtu_demo.c          #Main program
    ├── libusrdtu.so            # dtu library file
    ├── Makefile                        # Makefile
    └── usrdtu.h                        # Header file

3 directories, 7 files
```

7.1. Analysis of file structure

The configuration file is copied to the /etc/config/ directory on the device from the top-level Makefile, and is available for application programs to call.

```
config uart2 'uart2'
    option parity 'NONE'
    option data '8'
    option stop '1'
    option flow 'NFC'
    option fl '1000'
    option baud '115200'
    option period_auto 'OFF'
    option ft '10'
    option mode '0'

config socket 'socket'
    option sa_server 'test.cn'
    option sa_port '2317'
       option sa_enable 'OFF'
```

OpenWrt uses UCI (Unified Configuration Interface) as a system for managing configuration parameters. The specific usage method can be found in the UCI instruction manual.

Startup script: It is a shell script used to manage various services of an application.

```
#!/bin/sh /etc/rc.common       #The specified way to execute the script(important)
START=10        #the start level of the script (Tips: The smaller the number, the earlier the
#                script will be started
STOP=10         # The stop level of the script is 10
USE_PROCD=1     #Enable procd
start_service() {
#procd start service
procd_open_instance
procd_set_param command /usr/bin/dtu_demo
procd_set_param respawn 3600 5 120960
procd_close_instance
}

stop_service() {
#procd stop service
ps | grep dtu_demo | grep -v grep | awk '{print $1}' | xargs kill
cnt=0
while [ 1 ]; do
    PID=`ps | grep dtu_demo | grep -v grep | awk '{print $1}'`
    if [ "$PID" == "" ]; then
        break;
    fi
```

```
    usleep 100000
    let cnt=cnt+1
    if [ $cnt -ge 15 ]; then
        ps | grep dtu_demo | grep -v grep | awk 'NR==1{print $1}' | xargs kill -s 9
        break
    fi
done
   }
```

Src directory: The files in this directory mainly consist of source code for the application program. The Makefile in this directory is responsible for compiling the source code into an executable file.

```
all: libusrdtu.so dtu_demo

dtu_demo: example/dtu_demo.c
    $(CC) $(CFLAGS) $(LDFLAGS) example/dtu_demo.c -o dtu_demo -I./ -L./ -lusrdtu -Werror
    $(STRIP) dtu_demo
clean:
    rm *.o dtu_demo -rf
install:
        @echo "none"
```

Top-level Makefile: This file is primarily used for OpenWrt compilation.

```
   #------ Makefile rules for integrating unofficial package in OpenWRT
include $(TOPDIR)/rules.mk

PKG_NAME:=[package name that should be the same as the folder name]
PKG_VERSION:=[package version]
PKG_RELEASE:=1

PKG_BUILD_DIR := $(BUILD_DIR)/$(PKG_NAME)

include $(INCLUDE_DIR)/package.mk

define Package/$(PKG_NAME)
        SECTION:=utils
        CATEGORY:=[The location of the package in menuconfig, such as Base system]
        DEPENDS:=[Dependency package which is separated by space. Adding a + sign in
front of a package name indicates that it will be displayed by default and selecting the
package will automatically select its dependencies. Without a + sign, the package will
not be displayed by default and its dependencies will only be displayed when selected.]
        TITLE:=[Title]
        PKGARCH:=[Processor, such as ar71xx. If contains all model, the parameter is all]
        MAINTAINER:=[Author name]
endef

define Package/$(PKG_NAME)/description
        [Brief introduction of the package]
```

```
endef

#Copy the source code files that are not in this directory to the corresponding directory.

# such as ../../xucommon/xucommon.c, copy xucommon.c to the source code ../../ in this
directory

define Build/Prepare

    mkdir -p $(PKG_BUILD_DIR)
    $(CP) ./src/* $(PKG_BUILD_DIR)/

endef

define Build/Configure
endef

define Build/Compile
endef

define Package/$(PKG_NAME)/conffiles
[When upgrading, keep the file/back up the files. Each file should be listed on a separate line]
endef

define Package/$(PKG_NAME)/install
        $(CP) ./files/* $(1)/
endef

define Package/$(PKG_NAME)/preinst
[Script to be executed before installation. Remember to include #!/bin/sh. If not, leave it blank.]
    #!/bin/sh
    uci -q batch <<-EOF >/dev/null
     delete ucitrack.@aria2[-1]
     add ucitrack aria2
     set ucitrack.@aria2[-1].init=aria2
     commit ucitrack
    EOF
    exit 0
endef

define Package/$(PKG_NAME)/postinst
[Script to be executed after installation. Remember to include #!/bin/sh. If not, leave it blank]
    #!/bin/sh
     rm -f /tmp/luci-indexcache
    exit 0
endef
```

```
Package/$(PKG_NAME)/prerm
[Script to be executed before deletion. Remember to include #!/bin/sh. If not, leave it blank]
endef

Package/$(PKG_NAME)/postrm
[Script to be executed after deletion. Remember to include #!/bin/sh. If not, leave it blank]
endef

$(eval $(call BuildPackage,$(PKG_NAME)))
```

After the package is prepared, you can execute "make menuconfig" in the SDK to check if the added package is updated in the menuconfig. Select the compilation method by pressing the space bar, where "[*]" means it will be compiled into the firmware, and "[M]" means it will only be compiled. After selecting the method, execute "make" to compile the firmware along with the new package.

Alternatively, you can compile only the package in the SDK by running `make package/USR/libs/libusrdtu/compile V=s`. After compilation is complete, the .ipk file will be generated in the bin directory.