WPAN Gateway

Generated by Doxygen 1.9.1

Chapter 1

Main Page

Welcome to the WPAN gateway documentation! This software was built as part of a 2023 HES-SO MSc thesis at HEIG-VD's ReDS institute.

The goal of this software is to provide a way to access WPAN nodes remotely.

The original use case is regulation of a household's power consumption. It is supposed that the household has autonomous power production capacities like solar panels as well as access to the electrical power grid. This software provides a way to interface with various WPAN nodes to control their power production or consumption. A more specific use case is to minimize a household's power consumption from the grid. This can be achieved by filling power storage (electric car, water heater, backup battery) when the household has surplus self-generated power (solar, wind). For this to work, nodes must monitor household power production, consumption and net grid consumption. A server has to retrieve this data and control power consumers accordingly.

A second use case is to route some internet traffic to a remote low-speed network using the extended range WPAN provides, instead of routing an Ethernet cable.

1.1 Feature backlog

- WPAN manager initialization with path to config file
- Allow static node update during execution by parsing config file
- Add power and power in dBm to options in config file
- Change Node type management from several typed lists to one dynamically allocated pointer vector
- Add source port/address to DATA frames for reliable response addresses or check frame ID to determine this
- ISM3_Linux driver with hardware reset (needs new version of shield)
- Add universal Node subclass that handles all protocols. To be done once Node type management has been changed to a single dynamically allocated pointer vector.

Author	Marc Leemann	marc.leemann@master.hes-so.ch
Supervisor	Bertrand Hochet	bertrand.hochet@heig-vd.ch

v1: February 2023

Chapter 2

Hardware setup

This software was originally developped to be run on a Raspberry Pi CM4 supplemented by a custom ISM 868 MHz radio module (from here on referred to as "shield").

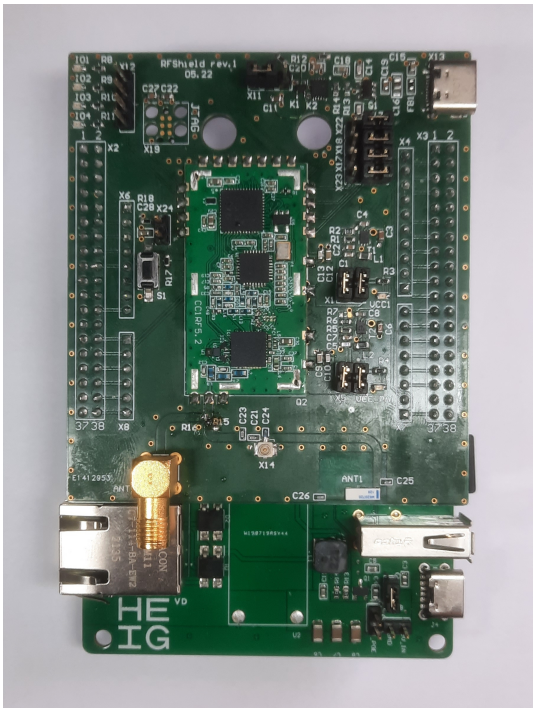

Figure 2.1 Shield and Pi CM4 assembly

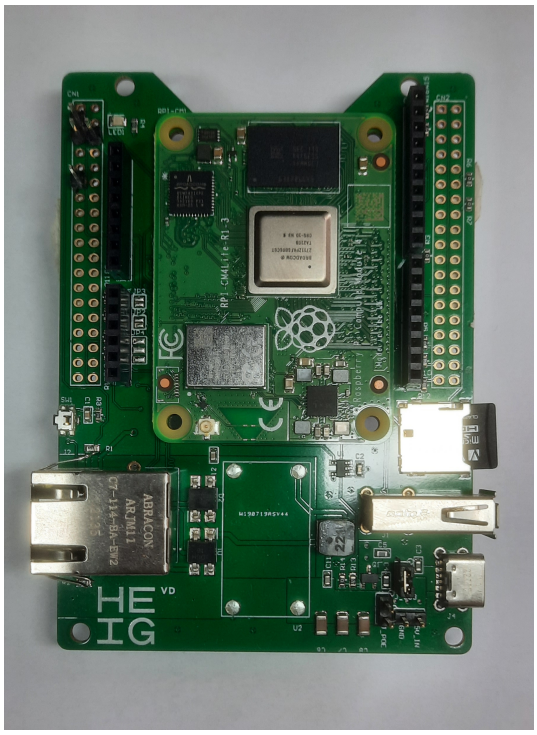

Figure 2.2 Pi CM4 board

The Pi CM4 lies on a custom breakout board developped for IoT applications. The shield is a breakout board for a custom radio module. This arrangement allows the CM4 to communicate with the shield via serial peripheral /dev/ttyAMA1, as defined in hardware.h.

Chapter 3

Linux configuration

The CM4 microcomputer uses a custom distribution using Buildroot. It uses the Pi CM4's default bootloader. See platform_config for more information about Buildroot.

3.1 SSH configuration

To have a more comfortable debugging experience, it is suggested to copy and run programs remotely using SSH. This requires `setting up SSH keys` for quick remote identification. It is advised to limit the key length to reduce connection times during a development phase.

When using the same SSH key for multiple remote hosts, SSH will display a warning and fail. Of course, using the same SSH key is not recommended for a final application. We can simply `add the remote host to SSH's known hosts list to connect`.

```
ssh-keyscan -H 192.168.1.10 » ~/.ssh/known_hosts # change IP accordingly
```

3.2 Ulimit

Ulimit is the Linux resource management utility. When the border router has to handle many concurrent connections, it may run out of resources. It is possible to allocate more resources to user processes by using `ulimit`.

The most likely crash source is going to be the max limit of open file descriptors. File descriptors allow access to sockets. Program may crash when reaching too high a number of connected Nodes. Use

```
ulimit -n 1024
```

to grant more open file descriptors to the program and avoid this problem.

3.3 Automatically starting the application

It is possible to start the gateway application with the system by creating a service for it. Service manager is BusyBox. Service files are located in /etc/init.d. They have the following structure:

```
#!/bin/sh
PROGRAM_DIR=~/
PROGRAM_NAME=gateway
start() {
  echo "Starting $PROGRAM_NAME: "
  $PROGRAM_DIR$PROGRAM_NAME
}
stop() {
  printf "Stopping $PROGRAM_NAME: "
  killall $PROGRAM_NAME
}
restart() {
  stop
  start
}
case "$1" in
  start)
  start
  ;;
  stop)
  stop
  ;;
  restart|reload)
  restart
  ;;
*)
  echo "Usage: $0 {start|stop|restart}"
  exit 1
esac
exit $?
```

Service file should be named S[XX]gateway, with XX a two-digit order of execution. Example:

```
S90gateway # Service will be started after service 89
```

BusyBox will execute all properly-named service scripts on startup.

3.4 Automatically restarting the application after a crash

It is desirable to restart automatically after a crash, as the application is not guaranteed to be stable. Fortunately, BusyBox can do this for us. The respawn action tells BusyBox to monitor program status and exit code, restarting it if it should fail.

Note: This should only be done in the final application, using a newer version of the shield that includes a hardware reset. Otherwise the service will probably hang in an incomplete initialization state, as it frequently does at this stage. This is because the current version of gateway software does not restart reliably without hardware-resetting the ISM3 module.

To have BusyBox automatically restart a service, modify /etc/inittab to add the following line:

```
[id]::respawn:[PROGRAM_PATH] [args]
```

ID is the tty to run the service on.

For example, if we want to have our service output messages to the console:

```
console::respawn:~/gateway
```

For more information about BusyBox configuration, refer to /etc/inittab header.

```
cat /etc/inittab
```

Final note: Since startup scripts in /etc/init.d are executed by /etc/init.d/rcS, you should remove any startup script for the application if you choose to have the service run with the BusyBox respawn option.

Chapter 4

Platform configuration

This project uses Buildroot. Buildroot is a toolchain that simplifies custom Linux target creation. It is available on the Buildroot project website. It is an archive to be downloaded and compiled on another host system. It also provides cross-compilers. These are useful to quickly compile the Linux kernel and whatever program to be developped using the host machine's considerably higher computing power.

The reason Buildroot is interesting is that it allows easy configuration of a custom Linux system. We can tinker with the kernel and configurations in order to only get the functionalities we need. This allows faster boot times and security upgrades.

4.1 Buildroot configuration and initial build

Initial setup is easy per the Buildroot manual:

1. Check that the host PC has all the required packages.
2. Download and extract Buildroot from the Buildroot downloads page.

Once again, most of the work has been done for us, in the form of default configurations to save us the tough work of configuring everything ourselves. Our target, the Raspberry Pi CM4, has a lot of community support

1. List available default configurations by running this command in the buildroot root folder:
   ```
   make list-defconfigs
   ```
   Bingo! There is a default configuration for the Pi CM4:
   ```
   raspberrypicm4io_defconfig
   ```
2. Configure Buildroot for the Pi CM4:
   ```
   make raspberrypicm4io_defconfig
   ```
3. Build the kernel and toolchain (can take over an hour the first time, be patient):
   ```
   make all
   ```
4. Configure Buildroot:
   ```
   make menuconfig
   ```
5. Configure Linux:
   ```
   make linux-menuconfig
   ```
6. Build the toolchain:
   ```
   make
   ```

4.2 Flashing the built Linux

Buildroot creates a complete SD card image for us. We can easily flash it with a dedicated script.

```
#!/bin/sh
SD_ROOT_FOLDER=/dev/sda # Modify according to your SD path (use lsblk to find it)
sudo umount $SD_ROOT_FOLDER
#initialize 960MiB to 0
sudo dd if=/dev/zero of=$SD_ROOT_FOLDER count=240000
sudo dd if=output/images/sdcard.img of=$SD_ROOT_FOLDER # flash SD image
sudo umount -r ${SD_ROOT_FOLDER}1
sudo umount -r ${SD_ROOT_FOLDER}2
sudo umount -r ${SD_ROOT_FOLDER}1 # Twice to get umount confirmation
sudo umount -r ${SD_ROOT_FOLDER}2
```

4.3 Platform configuration

4.3.1 Getting a serial terminal

```
make menuconfig
```

Go to "System configuration" -> "Run a getty after boot" and set the "TTY port" to ttyACM0, "Baudrate" to 115200 and "TERM environment variable" to vt100 per this article. Modify /boot/cmdline.txt on your SD card and add the following line:

```
console=ttyAMA0,115200 root=/dev/mmcblk0p2 rootwait
```

Getting a serial terminal means you can see boot messages from a console application like minicom. This is particularly useful in the early stages of development when network configurations may not be operational making SSH impossible. We can better troubleshoot kernel panics (they will happen some day).

Connect to the CM4 with an USB serial adapter connected to UART0. You may need to cross RX with TX if the default wiring does not work.

Use minicom or another serial program:

```
minicom -b 115200 -D /dev/ttyUSB0 # replace /dev/ttyUSB0 with your device
```

To find your device, compare

```
ls /dev
```

output before and after plugging the USB to serial adapter.

See below an example of wiring.

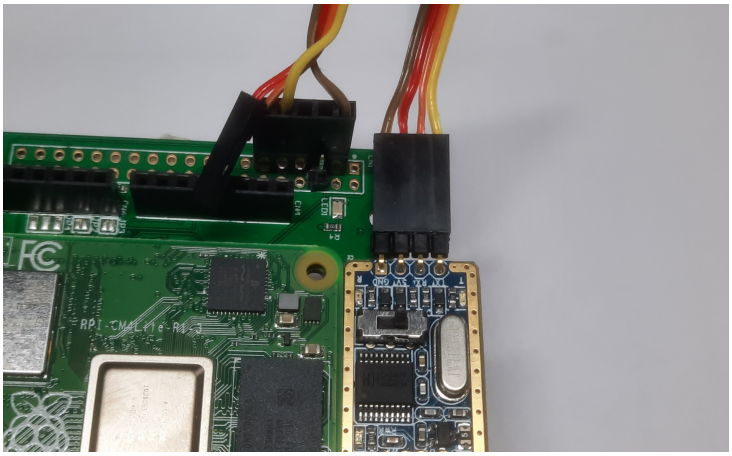

Figure 4.1 Wiring example

Signal	Color
GND	Brown
RX	Orange
TX	Yellow

Connecting +5V is not necessary and could theoretically lead to problems. It is safe in most applications. Using a USB-to-serial adapter to power the CM4 is not recommended. The kernel will display undervoltage alerts.

4.3.2 Network configuration

We need to configure the network in order to use SSH. The CM4 board has Ethernet and Wi-Fi capabilities. Ethernet does not require installing packages. Wi-Fi requires the following configuration (per this article):

```
make menuconfig
```

Allow automatic loading of wireless driver:

- System configuration -> /dev management -> Dynamic using devtmpfs + mdev

Install onboard Wi-Fi firmware:

- Target packages -> Hardware handling -> Firmware -> rpi 4 (extended)
- Target packages -> Hardware handling -> Firmware -> bcrmfmac-sdio-firmware-rpi

Install Wi-Fi configuration packages:

- Target packages -> Networking applications -> iw
- Target packages -> Networking applications -> wpa-supplicant -> Enable nl80211 support
- Target packages -> Networking applications -> wpa_supplicant -> Install wpa_passphrase binary (optional)

Install RF switch into kernel and not as a module:

```
make linux-menuconfig
```

- Networking support -> RF switch subsystem support (Y)

Configuration of network interfaces is done by modifying /etc/network/interfaces. Find below a sample configuration for a static Ethernet address and a DHCP Wi-Fi address.

```
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet static
    address 192.168.0.10
    gateway 192.168.0.1
    netmask 255.255.255.0
auto wlan0
iface wlan0 inet dhcp
    pre-up wpa_supplicant -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf -B
    post-down killall -q wpa_supplicant
    wait-delay 15
iface default inet dhcp
```

WPA configuration is done by editing /etc/wpa_supplicant.conf or using the wpa_passphrase utility:

```
country=CH
update_config=1
network={
    ssid="WIFINAME"
    psk="PASSWORD"
}
```

4.3.3 SSH configuration

Buildroot ships Dropbear as the default SSH client. It is a lightweight client. We chose to use OpenSSH to get an easier time setting up SSH keys. To remove Dropbear and install OpenSSH:

```
make menuconfig
```

Uncheck Dropbear:

- Target packages->Networking applications->dropbear

Check OpenSSH client, server and key utilities packages:

- Target packages->Networking applications->openssh

SSH keys should be stored in ~/.ssh and accessible only by root user. We need to create the .ssh folder in rootfs_overlay, populate it with the generated authorized_keys file and set its permissions.

Set the permissions by editing buildroot/system/device_table.txt:

```
#path                      type  perms
/root/.ssh/                  d   700 0   0   -   -   -   -   -
/root/.ssh/authorized_keys f   600 0   0   -   -   -   -   -
```

4.3.4 Debugging programs

```
make menuconfig
```

Per this book snippet:

- Toolchain -> Build cross gdb for the host
- Toolchain -> Enable WCHAR support
- Toolchain -> Thread library debugging
- Target packages -> Debugging, profiling and benchmark -> gdb
- Target packages -> Debugging, profiling and benchmark -> gdbserver
- Target packages -> Debugging, profiling and benchmark -> full debugger

Using the debugger (per buildroot manual):

On the CM4:

```
gdbserver :2345 [PROGRAM] # GDB server listens on TCP port 2345
```

On the host PC, from cross-compilation directory:

```
<buildroot>/output/host/bin/<tuple>-gdb -ix <buildroot>/output/staging/usr/share/buildroot/gdbinit
       [CROSSCOMPILED_PROGRAM]
(gdb) target remote <target ip address>:2345 # Connect to gdb server
```

4.4 Device tree overlay

The device tree overlay is a file that describes the hardware setup so that Linux can know where its peripherals are. It doesn't describe all hardware by default, to keep a lightweight system.

Since the gateway application uses the CM4's serial device 4, we have to tell Linux it exists and how. Fortunately, the tedious description work has already been done for us. All we need to do is add the following line at the end of /boot/ dtoverlay.

```
# enable UART4
dtoverlay=uart4
```

4.5 Saving linux configuration in Buildroot

4.5.1 Rootfs overlay

To have your Linux configuration kept by Buildroot instead of resetting it on build, configure and populate rootfs_overlay.

By default, buildroot/board/raspberrypicm4io/rootfs_overlay is used. Putting any file there will have it replace the originally generated file.

It is possible to modify this path:
```
make menuconfig
```

- System configuration -> Root filesystem overlay directories

4.5.2 /boot/config.txt

It is possible to modify default /boot/config.txt by editing buildroot/board/raspberrycm4io/config_cm4io.txt.

It is thus possible to add the UART4 device tree overlay as described before:
```
# enable UART4
dtoverlay=uart4
```

4.5.3 Network configuration

Persistent network configuration changes can be kept by editing buildroot/board/raspberrycm4io/interfaces. This file matches /etc/network/interfaces on the Linux image. It is copied after building (see post-build script) The current configuration uses DHCP for the Wi-Fi interface and static addressing for Ethernet.
```
auto lo
iface lo inet loopback
auto eth0
#iface eth0 inet dhcp
iface eth0 inet static
    address 192.168.0.10
    gateway 192.168.0.1
    netmask 255.255.255.0
    #pre-up /etc/network/nfs_check
    #wait-delay 15
    #metric 100
auto wlan0
iface wlan0 inet dhcp
    pre-up wpa_supplicant -D nl80211 -i wlan0 -c /etc/wpa_supplicant.conf -B
    post-down killall -q wpa_supplicant
    wait-delay 15
#    metric 100
iface default inet dhcp
```

WPA configuration is in buildroot/board/raspberrycm4io/wpa_supplicant.conf. This file matches /etc/wpa_↩ supplicant.conf. It is called on configuration of network interfaces. Find below an example configuration:
```
country=CH
update_config=1
network={
    ssid="WIFINAME"
    psk="PASSWORD"
}
```

4.5.4 Post-build script

Sometimes it is not possible to put configs in rootfs_overlay. We can use the post_build.sh script in board/raspberrypicm4io/post-build.sh to apply any change we want to the built image before it is created.

The script is executed before imaging, but after building.

```
#!/bin/sh
 set -u
 set -e
 # Add a console on tty1
 if [ -e ${TARGET_DIR}/etc/inittab ]; then
     grep -qE '^tty1::' ${TARGET_DIR}/etc/inittab || \
    sed -i '/GENERIC_SERIAL/a\
 tty1::respawn:/sbin/getty -L  tty1 0 vt100 # HDMI console' ${TARGET_DIR}/etc/inittab
 fi
 # enable wi-fi per
 #https://blog.crysys.hu/2018/06/enabling-wifi-and-converting-the-raspberry-pi-
 #in t-o-a-wifi-ap/
 cp package/busybox/S10mdev ${TARGET_DIR}/etc/init.d/S10mdev
 chmod 755 ${TARGET_DIR}/etc/init.d/S10mdev
 cp package/busybox/mdev.conf ${TARGET_DIR}/etc/mdev.conf
 cp board/raspberrypicm4io/interfaces ${TARGET_DIR}/etc/network/interfaces
 cp board/raspberrypicm4io/wpa_supplicant.conf ${TARGET_DIR}/etc/wpa_supplicant.conf
```

The script was modified to copy the network configurations into the relevant folders. It is probably not necessary and could be done using rootfs_overlay, but it is working.

Chapter 5

Build configuration

5.1 Build environment setup

To cross compile, we need to add the Buildroot cross compiling toolchain to our PATH. In any console:

```
vim ~/.bashrc
export PATH="$PATH:/[PATH_TO_BUILDROOT]/buildroot-2022.08/output/host/usr/bin" # add this line to .bashrc
```

5.2 Cross compilation

Use arm-linux-gcc or arm-linux-g++ to compile programs. Use normal compiler flags (for example -g for a GDB debug build).

5.3 Project makefile

It is easier to use a makefile for compiling. The current makefile is located at the root of the project. It handles cross compilation.

5.4 Launching on a remote target

Once SSH key setup is done, it is easy and quick to compile and run the program using a small script.

```
#!/bin/sh
PROGRAM_NAME="gatewayTest"
HOSTNAME="cm4"
if make
then
   echo "Copying program..."
   scp $PROGRAM_NAME root@$HOSTNAME:~/$PROGRAM_NAME # copy program on remote host root
   scp nodes.txt root@$HOSTNAME:~/nodes.txt # copy config file
   echo "Program output:"
   ssh -tt root@$HOSTNAME "killall $PROGRAM_NAME ; ./$PROGRAM_NAME" # kill running version and launch in an
       SSH terminal
fi
```

Chapter 6

Test utilities

Some test utilities are included in the project. Find them in the Utilities folder.

6.1 Python gateway script

Location: Utilities/test_choice.py

This script is a basic gateway for test purposes. It allows group wake/sleep, unicast TX, RX and changing beacon data. It is run from any host PC with a RM1S3 module connected in standalone mode.

Reference configuration for RM1S3 module use from USB:

Name	Side
X1	On (vertical)
X5	On (vertical)
X11	Right
X17	Left
X18	Left
X22	Left
X23	Left
X24	No jumper

Use this script to test node connection and handlers.

6.2 Send UDP packet script

Location: Utilities/sendPacket.sh

This script will send a text messages to a remote UDP socket. Use it to send text to the CM4's UDP socket. Run either gateway application in border router mode or UDP server test on the CM4 to test these applications.

6.3 UDP client

Location: Utilities/udpClient

This program is a simple UDP client to be run on another machine. Use it to test CM4 UDP connectivity and border router mode of gateway test application.

Program will ask for an input, send it to the target and print any received frames.

6.4 Radio driver test

Location: Utilities/radioDriver

This program was originally used to port the radio driver to Linux. It is a basic client test. It can be used to test frame reception, transmission and handler configuration. Use it in conjunction with any gateway program. The simplest, most reliable way to test a configuration is to use it with the Python gateway.

6.5 Serial communication test

Location: Utilities/serialTest

This program is used to test the CM4 board's serial connection. Change serial device in main to match yours. Default is /dev/ttyAMA1. This matches the RM1S3's serial connection.

Usage:

- Connect a serial-to-USB adapter between a PC and the CM4's serial device under test.
- Open a serial terminal on the PC and connect to the adapter.
- Run this program on the CM4 without the radio shield. "Serial test utility" should print on the terminal.
- Write any text in the serial terminal. The text should print case-inverted.

6.6 UDP server test

Location: Utilities/UdpServTest

This program is a simple UDP server. It is used to test remote PC to host PC connectivity. Run it on the CM4 and send UDP frames to it. Source can be modified to not only print the RX frames but also send them back.

Chapter 7

Codebase overview

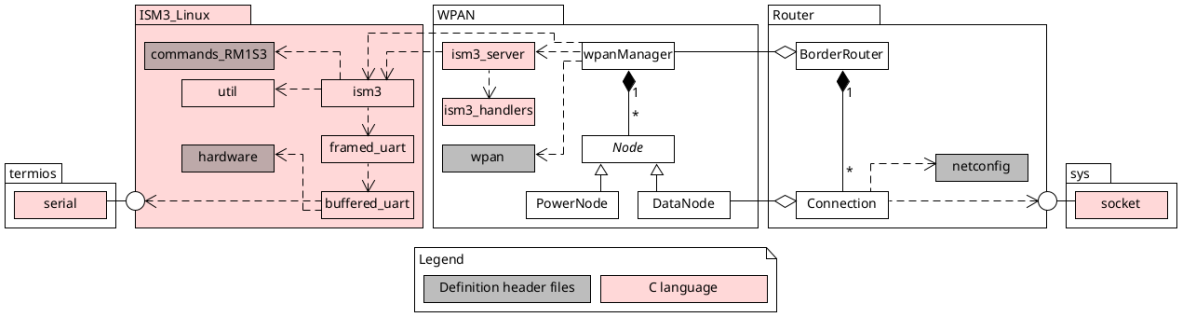

Figure 7.1 Project architecture

7.1 ISM3_Linux

This package contains the ISM3 driver stack. It is all written in C, ported from an STM32 platform. It communicates with the radio shield via a serial interface.

7.2 WPAN

This package contains the Wireless Personal Area Network Manager. The wpanManager_desc uses the ism3_↩ server.h adaptation layer to use server functions (group wake control and config). It uses the ism3.h layer directly to send frames to nodes.

It manages connected Nodes and exports a list of all nodes.

7.3 Router

This package contains the Border router. The latter gets a Node list from wpanManager_desc and opens UDP sockets for each DataNode.

7.4 Border router

7.4.1 Description

This class maintains a Connections list so that datagrams can be forwarded to and from remote datanodes. It polls the wpanManager_desc for new connected nodes and creates a new Connection to them if applicable.

7.4.2 Usage

7.4.2.1 Calling methods

This class is a Singleton. It builds and maintains its own instance. Access the instance through BorderRouter::getInstance. Instance cannot be initialized or accessed outside of this method.

7.4.2.2 Initialization

Initialize linked wpanManager first, then initialize border router with BorderRouter::init.

7.4.2.3 Ticking

Tick regularly enough using BorderRouter::tick(). The latter function also ticks wpanManager. It is possible to only tick wpanManager with its wpanManager::tick, so border router can be ticked only periodically. Ticking also ticks open Connections, polling their sockets and dataNodes for new data.

7.4.3 Connections

7.4.3.1 Description

A Connection is a datagram forwarder from a UDP socket to a DataNode. This is the core of the Border router functionality.

7.4.3.2 Usage

7.4.3.2.1 Instantiation The class is instantiated with a pointer to its DataNode. It opens a socket to UDP port = BASE_GW_IN_PORT + DataNode::address. Any device can then connect to a DataNode provided they can reach the UDP port.

7.4.3.2.2 Ticking Connections must be manually polled since DataNode data reception does not throw interrupts yet.

7.4.3.2.3 Packet forwarding As of now, Connection forwards any incoming UDP datagrams to its remote node. It forwards all datagrams coming from its node to the **last UDP sender**. This is basic, can be unpractical and unsafe, but functional for now. Security is left to the user via access control on the gateway's sockets.

A way to eliminate this problem would be to include source address and port in datagrams transmitted via Data transfer protocol. This would however reduce already limited payload, since an IPv6 address is 16-byte long and a port number is 2-byte long. This would thus take away 18 bytes out of at least one Data transfer protocol packet or out of every packet depending on implementation.

Given the probable use case of one local server centralizing all WPAN information and making it accessible online, the security gains of reliable UDP (!) transfer are slim provided the user configures the border router's firewall correctly.

Chapter 8

Hardware documentation

ISM3 documentation

Pi CM4 board schematic

Shield schematics and layout

Chapter 9

Using the ISM3 module

9.1 Overview

The ISM3 radio module is a serial-accessible radio module for LPWAN applications. It operates in the ISM 868 or 915 MHz band.

Note: ISM stands for `Industrial-Scientific-Medical`, which refers to the common name given to its operating band. The ISM 868 band is an unlicensed (meaning anyone can use it as long as they follow some common rules) band with center frequency at 868 MHz. It is used for IoT applications in Europe. Its US counterpart is at 915 MHz. There are other ISM bands, such as the 2.4 and 5 GHz bands used by Wi-Fi.

9.2 Usage

Initialize handler functions (ism_init with ism3_handlers.h as arguments), configure module and set sync mode.

ism_tick regularly to update serial RX/TX buffers. Without ticking, data may neither be transmitted nor received.

Transmit frames with ism_tx.

Receive frames with configured handlers (see ism_init).

Note: At the time of writing, the module does not feature a GPIO hardware reset. This means that it cannot be reset by the driver. It does, however, need to be reset before being reconfigured. Until a new version of the shield is developed, the user has to manually reset the module on program launch to ensure proper configuration.

9.3 Configuration

See Configuring the ISM3 module

9.4 Power modes

See ISM3 sync modes

9.5 States

See ISM3 states

9.6 Troubleshooting

See ISM3 Troubleshooting

9.7 Configuring the ISM3 module

9.7.1 Where is the config?

Configuration settings are scattered across:

- ism_config
- ism_set_sync_mode
- the configuration_commands array
- the BAUDRATE macro

9.7.2 How do I change it?

Use ism_config to set address, group, power and beacon ID to sync to.

Use ism_set_sync_mode to set the desired sync mode. See ISM3 sync modes for help on which one does what.

Modify the configuration_commands array's contents to match your liking. Refer to document RM1S3 Host Commands for settings information. You can simply look up the command code and the command reference should tell you what the parameter sets.

Attention: Changing the baudrate in configuration_commands is not enough on its own. Baudrate must also be changed in BAUDRATE macro for ISM3 baudrate to match host baudrate.

9.7.3 Default gateway configuration

A default configuration is available with ism_server_init. This configuration is called from wpanManager constructor. See implementation for details.

9.7.3.1 ism_config

Call with following parameters:

Name	Value
address_	0
group_	0xFFFFFFFF
power_	0x10
power_dbm_	0x12
associated_beacon↩ _id	any (default: 0xD322FE7D02D3D117)

9.7.3.2 ism_set_sync_mode

Set as SM_TX. See ISM3 sync modes.

9.7.3.3 configuration_commands array

This array defines the commands that will be sent on initialization. Commands will be sent in order. The #↩ NUMBER_OF_RECONFIGURATION_CMD macro defines the number of commands that will be sent in case of reconfiguration. Reconfiguration commands are the last ones in the array.

```
static commands_t configuration_commands = {
    {0x04, CMD_SET_HOST_BAUDRATE,     0x00, 0x4B, 0x00}, // 19200 in hex notation
    {0x01, CMD_GET_FIRMWARE_VERSION},
    {0x01, CMD_GET_HARDWARE_VERSION},
    {0x01, CMD_GET_UNIQUE_DEVICE_ID},
    {0x05, CMD_SET_PATTERN,           0x19, 0x17, 0x10, 0x25},
    {0x01, CMD_GET_PHYS_CHANNELS_PLAN_SIZE},
    {0x02, CMD_SET_TX_RETRY_COUNT,    0x02}, // Tx retry count = 2
    {0x02, CMD_SET_GPIO0_SIGNAL,      0x01}, // GPIO0 = SIG_SYNC_OUT
    {0x02, CMD_SET_GPIO1_SIGNAL,      0x02}, // GPIO1 = SIG_TIMESLOT_OUT
    {0x02, CMD_SET_GPIO2_SIGNAL,      0x04}, // GPIO2 = SIG_TX_PENDING_OUT
#ifdef NDEBUG
    // interval = 1, missMax = 100, initialTrack = 402ms, trackOn = 202ms, trackOff = 400s
    {0x0A, CMD_SET_SYNC_RX,           1, 100, 0x01, 0x92, 0x00, 0xCA, 0x06, 0x1A, 0x80},
    // {0x0A, CMD_SET_SYNC_RX, 1, 100, 0x01, 0x92, 0x00, 0xCA, 0x00, 0xEA, 0x60}, // 60s
#else
    // interval = 1, missMax = 100, initialTrack = 402ms, trackOn = 202ms, trackOff = 5s
    {0x0A, CMD_SET_SYNC_RX,           1, 100, 0x01, 0x92, 0x00, 0xCA, 0x00, 0x13, 0x88},
#endif
    {0x02, CMD_SET_EVENT_INDICATION,  1}, // Enable IND_TDMA_STATE_CHANGED
    {0x02, CMD_SET_RADIO_MODE,        0x00}, // Stop TDMA
    {0x02, CMD_SET_RF_PHY,            0x00}, // use phy as parameter
    {0x02, CMD_SET_ACTIVE_CHANNELS,   0x00}, // use channels as parameter
    // SERVER CONFIG
    {0x02, CMD_SET_DATA_SLOT_COUNT, 0xF}, // 15 data slots
    // Following parameters can change during execution, the numbers of commands should be set in
       NUMBER_OF_RECONFIGURATION_CMD
    {0x02, CMD_SET_SYNC_MODE,         0x00}, // use sync_mode as parameter
    {0x06, CMD_SET_SYNC_RX_LOW_POWER, 0x00, 0x00, 0x00, 0x00, 25}, // use group as parameter, sync interval
       = 25
    {0x02, CMD_SET_RF_POWER,          0x00}, // use power as parameter
    {0x02, CMD_SET_ADDRESS,           0x00}, // use address as parameter
    {0x05, CMD_SET_GROUP,             0x00, 0x00, 0x00, 0x00}, // use group as parameter
    {0x09, CMD_SET_ASSOCIATED_BEACON_ID, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}, // use
       associated_beacon_id as parameter
    {0x04, CMD_SET_DATA_SLOT_RANGE_TYPE, 0, 254, 0x01}, // enable RX on all data slots
    {0x03, CMD_SET_DATA_SLOT_TYPE,    0x00, 0x00},
    {0x02, CMD_SET_RADIO_MODE,        0x01}, // Start TDMA
    {0x00}
}; ///< Configuration commands
```

Example: if #NUMBER_OF_RECONFIGURATION_CMD=2, CMD_SET_DATA_SLOT_TYPE and CMD_SET_↩ RADIO_MODE will be sent on reconfiguration.

9.7.3.4 BAUDRATE macro

This macro defines linux hardware baudrate. It should match baudrate set in configuration_commands. Default value is 19200. It should not be set as a random value since CM4 hardware only supports a few fixed baudrates. See linux_uart_init source in buffered_uart.c and this blog post for additional information about UNIX compliant baudrates.

9.7.4 What can I change?

You can change host baudrate, pattern, TX retry count, GPIO signals, Sync RX, RF Phy and channels, data slot count, sync mode, sync rx interval, power, address and group.

To keep gateway functionality, do not change sync mode, address and group.

9.8 ISM3 sync modes

The ISM module used in has several sync modes for the user to choose from. The following descriptions should help the user choose the relevant mode for their application.

9.8.1 ISM_TX

Module is in gateway mode. It will be the master of its WPAN (Wireless Personal Area Network).

9.8.2 SM_RX_ACTIVE

Module is in RX mode. It will remain in SYNCHRONIZED state all the time, and receive unicast and multicast frames. The module will not enter power saving mode.

9.8.3 SM_RX_LOW_POWER

Module is in low power mode. It will not react to group wakeups and remain asleep. The only way for the module to receive data is to get it from a beacon data change.

9.8.4 SM_RX_LOW_POWER_GROUP

Module is in low power mode and will react to group wakeups. When the gateway wakes one of the groups the module belongs to, it will enter state ISM_SYNCHRONIZED. When the node is synchronized, it will receive unicast and multicast frames.

9.9 ISM3 states

9.9.1 General information

It takes approximately 15 seconds for a node to lose sync with the current configuration in ism3.c. This is because the node tries several times to reach the gateway until it determines it has lost sync.

9.9.2 ISM_OFF

Module is uninitialized.

9.9.3 ISM_NOT_SYNCHRONIZED

Module is waiting for sync. It is not connected to a gateway.

9.9.4 ISM_SYNCHRONIZED

Module is synchronized to a gateway. Gets in this state on wakeup from gateway if it is in SM_RX_LOW_POWER↩ _GROUP power mode. Stays in this state for as long as it is connected in SM_RX_ACTIVE power mode.

9.9.5 ISM_LOW_POWER_SYNC

Module is synced to a gateway in SM_LOW_POWER or SM_LOW_POWER_GROUP power mode, but not woken up. Cannot be woken up in SM_LOW_POWER.

9.9.6 ISM_TX_SYNC

Module is the gateway of its WPAN.

9.9.7 ISM_VERSION_READY

Module is getting initialized and has not received a power mode.

9.10 ISM3 Troubleshooting

Things don't always happen as planned and that is a normal part of development. There are a few standard steps one can take to identify problems.

9.10.1 Documents

ISM3 documentation

Pi CM4 board schematic

Shield schematics and layout

9.10.2 Jumper setup

There are a few jumpers to choose different configurations on the radio shield:

- X11: power source. Left: power from NUCLEO connector. Right: power from USB-C connector.
- X17,X18,X22,X23: UART source. Left: USB-C UART. Right: UART from NUCLEO connector.
- X1,X5: Power supply toggles. Left jumper closes power supply circuit. Right jumper closes power supply indicator LED circuit.
- X24: Integrated STM32 chip BOOT0 pin. Boot partition selection. Do not bridge unless you know what you are doing. Contact nicolas.brunner@heig-vd for information.

Reference configuration for use with CM4 is:

Name	Side
X1	On (vertical)
X5	On (vertical)
X11	Left
X17	Right
X18	Right
X22	Right
X23	Right
X24	No jumper

This configuration is shown on radio module picture (Hardware setup).

9.10.3 Module power supply

On the radio shield, check LEDs VCC1 and VCC_PA are lit up when jumpers X1, X5 are set. If not lit up, check jumper X11. Default setting is left for power coming from CM4 board.

9.10.4 UART troubleshooting

9.10.4.1 Check hardware connections

Check jumpers X17, X18, X22, X23 are correctly set.

UART Jumper definitions:

Name	Signal	Description
X17	CTS	Clear to send
X18	TX	Transmit
X22	RTS	Request to send
X23	RX	Receive

AN0059.0 from Silicon Labs provides a good overview of these signals and their waveforms.

9.10.4.2 Check Linux serial

The CM4 ISM3 driver uses serial port /dev/ttyAMA1 defined in UART_DEV. Check that this device links to UART4 on CM4. Check that correct device tree overlay is used: on CM4 file system, check that line dtoverlay=uart4 is present in /boot/config.txt. This binds UART4 to a /dev serial link in the Linux operating system. If your application uses other UARTs, check which /dev/ttyAMAX maps to which serial hardware device.

A serial test loopback program allows easy testing from a host PC with a USB to serial adapter connected on the relevant port. The program is located in Utilities/serialTest (see Test utilities).

9.10.4.3 Check UART signals

Check that UART frames are sent on the UART. Use a serial test program or write directly through command line with:

```
echo -ne '\033[2J' > /dev/ttyAMA1
```

Read with

```
cat -v < /dev/ttyAMA1
```

Or see Test utilities.

If no results are obtained, using an oscilloscope or logic analyzer will allow you to check for UART frames manually. If shield sends an answer, you are probably on the right track! See error code definitions in ISM3 doc and try to solve the problem.

If shield does not send an answer, baudrate configuration may be wrong. Check config using Configuring the ISM3 module as reference.

9.10.5 Shield configuration

In gateway mode, check that LEDs IO1 and IO2 light up (default configuration). If they don't, try to restart the program after resetting ISM3 module by pressing the reset button on the left side.

Default configuration LED signals:

LED	Signal	Description
1	SIG_SYNC_OUT	Beacon sent
2	SIG_TIMESLOT_OUT	TX/RX timeslot
3	SIG_TX_PENDING_OUT	Pending transmission

9.10.6 Radio communication

If configuration LEDs light up, radio link may be defective. Check R15-R16 position matches desired antenna setup. Check impedance matching on C21, C23, C24. If using on-shield antenna, check C25 and C26.

Chapter 10

Node protocols

Protocols are consistently and simply framed like so:

Offset	Length	Data
0	1	Protocol ID
1	1	Command
2	X	Command data

This payload is inserted in the standard radio module frame, which contains the message destination and length.

Command data size is fixed in advance on a command-per-command basis. Nodes are free to check for correct size or assume gateway and transmission are perfect. Gateway checks for command data to be big enough, but typically does not perform content checks.

The following protocols have been implemented:

- Network protocol
- Application protocol
- Application error protocol
- Data transfer protocol

10.1 Network protocol

This protocol contains all relevant commands to handle static and dynamic addressing.

Following commands exist:

- NETWORK_PING
- NETWORK_GETUID
- NETWORK_SETADDR
- NETWORK_SETGROUP
- NETWORK_DISCONNECT
- NETWORK_RENEW_LEASE
- NETWORK_GET_PROTOCOLS

Dynamic address management commands are a little bit special and follow their own format since they are broadcast to the network.

10.1.1 Dynamic address management commands

These commands are used as a sequence initiated by a Node configured in dynamic addressing mode. This mode provides automatic configuration of Nodes and does not require flashing different flavors of the same program on Nodes.

The sequence follows the classic DHCP pattern of DORA, described in this Wikipedia excerpt.

Note: Server and gateway are conceptually different but refer to the same entity. Server refers to the DORA server, which attributes addresses, and gateway is the broader WPAN manager and IP forwarder application. Please do not be surprised to see these two terms get used interchangeably.

Since Nodes start out technically address-less and default server address is not specified, addressing conflicts could occur if a dynamically-addressed Node's initial address were the same as a statically-addressed Node's. Broadcasting DORA commands allows both server and client to pick their frames up reliably. This however requires a second layer of addressing to distinguish between concurrent DORA requests from different clients.

As with MAC addressing in DHCP, this DORA implementation uses hardware addresses to direct messages. The hardware address is the ISM3 unique ID.

Consequently, DORA frames look a little bit different from other Network protocol frames:

Offset	Length	Data
0	1	Protocol ID
1	1	Command
2	12	Source UID
14	12	Destination UID
26	X	Command data

See the relevant commands for command data information:

- NETWORK_DISCOVER
- NETWORK_OFFER
- NETWORK_REQUEST
- NETWORK_ACK
- NETWORK_DORA_ERR

To summarize:

1. Node broadcasts a Discover command
2. wpanManager responds with an Offer containing an address for the node
3. Node Requests the address and the group it wants to be in.
4. wpanManager Acknowledges the node's new address and group and gives it a lease duration

Lease duration is specified in minutes, depending on NETWORK_LEASE_UNIT_MINUTES. A lease of 0 minutes means the address was not granted. Nodes are responsible for lease renewal, as WPAN manager will consider them timed out once their lease expires. A timed-out Node is not accessible anymore, it is considered disconnected. A NETWORK_DISCONNECT command is sent to timed-out nodes so that they may know that they can no longer use their address.

10.2 Application protocol

This protocol contains relevant commands to handle a network of power producers and consumers.

Following commands exist:

- APP_GETMANIFEST
- APP_GETPOWER
- APP_SETPOWER
- APP_GETPOWERSETTING
- APP_SETPOWERSETTING
- APP_GETPOWERSETTINGS

The idea behind this protocol is to provide simple means of getting and setting power consumptions for different nodes. For easier identification, a manifest can be obtained with the node's description.

There are two ways to set a node's target power:

1. Setting an absolute target
2. Using a power setting

Setting an absolute target is not always possible. Many devices control power usage in steps. For example, an electric car charging station may have several power modes corresponding to certain charge rates. If the target use case is, for example, to regulate household total power consumption, setting an unreachable target can cause some regulators to hang.

To accomodate the impossibility to set arbitrary power targets, the present protocol can transmit power settings. Power settings can represent anything depending on target application, but they typically represent power levels in kW. The application server first gets the power settings then can decide which one to use depending on its needs.

The following types can be changed to improve precision or reduce channel usage:

- powerkW_t (default: float)
- powerTarget_t (default: float)
- powerSetting_t (default: uint8_t)

General rules for node implementation:

- Nodes should allow a server to get the actual measured power consumption with APP_GETPOWER if it is available, even if target power is set with a power setting.
- Nodes should not allow a server to set an arbitrary power if they cannot set this power.
- Nodes can change the values behind their power settings during execution, provided they communicate the change (for example with an unsollicited APP_GETPOWERSETTINGS).
- Node implementation can exclude some commands. If a node receives a command it has no routine for, it should reply with APP_ERR_NOCOMMAND. This can be the case if a node cannot measure its power consumption.

10.3 Application error protocol

This protocol allows exchange of error messages regarding application commands. There is no command data.

Following messages exist:

- APP_ERR_NOCOMMAND
- APP_ERR_INVALIDINDATA
- APP_ERR_OUTDATATOOBIG
- APP_ERR_NODEMEM

Whether a node or a gateway actually implement these error messages is up to the user. Current PowerNode implementation only prints node error messages.

10.4 Data transfer protocol

This protocol allows exchange of UDP datagrams between node and gateway. It does not have commands.

The communicators split the *datagram* in *packets* small enough for transmission using ISM3 module. This maximal size is defined by DATA_MAX_PACKET_LENGTH.

The packet header allows reconstruction of the original datagram if all packets were received. No error detection is performed. Datagrams with missing packets are marked as valid unless the last packet is missing. In the latter case, they are open to completion and should be cleared periodically.

Frame definition :

Offset	Length	Data
0	1	Protocol ID: DATA_PROTOCOL_ID
1	m	Datagram ID
1+m	n	Current packet index
1+m+n	n	Last packet index
1+m+2∗n	X	raw data

n = size of packet_index_t

m = size of datagram_id_t

Max value of X is defined in DATA_MAX_PACKET_LENGTH. It depends on packet_index_t and datagram_id_t size. Default implementation has m=n=1. In practice, it is impossible at the moment to reliably transmit more than 3 consecutive packets, limiting datagram size to 705. This limit is probably sufficient for our low-power, low-data rate application. It could probably pointlessly be surpassed with a different PHY layer.

Chapter 11

Todo List

Member ism_init (ism_unicast_function_t rx_unicast_function, ism_multicast_function_t rx_multicast_function, ism_beacon_data_function_t beacon_data_function, ism_state_function_t state_function, ism_stat_function_t stat_function)

configure RESET pin and use it as in this example snippet

Chapter 12

Class Index

12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 13

File Index

13.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 14

Class Documentation

14.1 BorderRouter Class Reference

Forwards external packets to remote nodes. See Border router.

`#include <BorderRouter.h>`

Public Member Functions

- BorderRouter (const BorderRouter &)
  Copy constructor override.
- void operator= (const BorderRouter &)
  Assignment operator override.
- eBorderRouterStatus init (wpanManager *_wpan)
  Initialization function.
- void deInit ()
  DeInit.
- eBorderRouterStatus reInit ()
  Deinitialize then initialize.
- eBorderRouterStatus getStatus ()
  Status getter.
- void tick (uint32_t delayMs)
  Master ticker with delay.

Static Public Member Functions

- static BorderRouter & getInstance ()
  Singleton get instance.

Private Member Functions

- BorderRouter ()

  Unusable constructor.
- virtual ~BorderRouter ()

  Call deinit.
- void updateNodeList ()

  Update node lists from WPAN instance.
- void updateRoutes ()

  Update Connection vector to reflect WPAN connections.

Private Attributes

- vector< Connection > routes

  Opened Connection instance vector.
- vector< Node * > nodes

  WPAN manager nodes.
- eBorderRouterStatus status

  Own status.
- wpanManager * wpan

  Pointer to WPAN manager instance.

14.1.1 Detailed Description

Forwards external packets to remote nodes. See Border router.

14.1.2 Member Function Documentation

14.1.2.1 getInstance()

```
BorderRouter & BorderRouter::getInstance ( )  [static]
```

Singleton get instance.

Returns

reference to instance

14.1.2.2 getStatus()

```
eBorderRouterStatus BorderRouter::getStatus ( )
```

Status getter.

Returns

border router status

14.1.2.3 init()

```
eBorderRouterStatus BorderRouter::init (
            wpanManager * _wpan )
```

Initialization function.

Returns

own status

WPAN manager must be initialized elsewhere

14.1.2.4 reInit()

```
eBorderRouterStatus BorderRouter::reInit ( )
```

Deinitialize then initialize.

Returns

own status

14.1.2.5 tick()

```
void BorderRouter::tick (
            uint32_t delayMs )
```

Master ticker with delay.

Parameters

delayMs	delay in ms

Ticks WPAN manager and all Connections. Updates node lists and Connections if a new DataNode was connected

The documentation for this class was generated from the following files:

- Router/BorderRouter.h
- Router/BorderRouter.cpp

14.2 Connection Class Reference

Connection class.

`#include <Connection.h>`

Public Member Functions

- Connection (DataNode *_pNode)
  Constructor.
- ~Connection ()
  Destructor.
- void tick ()
  check Connection IO
- int getNodeAddr ()
  Getter function for node address.
- eConnectionState getStatus ()
  Getter function for Connection status.

Private Member Functions

- Connection ()
  Unused constructor.
- void deInit ()
  deInit Connection
- void initSocket ()
  initialize UDP socket depending on node address
- void deInitSocket ()
  deinit UDP socket
- int extToLocalHandler ()
  RX data from UDP socket and TX to WPAN Node.
- int localToExtHandler ()
  RX data from WPAN Node and TX to UDP socket TX to last RX address.

Private Attributes

- int sock_ext
  UDP socket file descriptor.
- DataNode * pNode
  Linked DataNode.
- uint8_t nodeAddr
  Node address.
- struct sockaddr srcAddr
  UDP RX source address.
- socklen_t srcAddrLen
  IP source address length.
- eConnectionState status
  Status of Connection.

14.2.1 Detailed Description

Connection class.

14.2.2 Constructor & Destructor Documentation

14.2.2.1 Connection()

```
Connection::Connection (
            DataNode * _pNode )
```

Constructor.

Parameters

_pNode	pointer to target dataNode

14.2.2.2 ~Connection()

```
Connection::~Connection ( )
```

Destructor.

De-initialize used sockets

14.2.3 Member Function Documentation

14.2.3.1 getNodeAddr()

```
int Connection::getNodeAddr ( )
```

Getter function for node address.

Returns

node address

14.2.3.2 getStatus()

```
eConnectionState Connection::getStatus ( )
```

Getter function for Connection status.

Returns

eConnectionState status enum value

14.2.3.3 tick()

```
void Connection::tick ( )
```

check Connection IO

Check for new messages from socket and DataNode. Call communication handlers if new data is available for transfer.

The documentation for this class was generated from the following files:

- Router/Connection.h
- Router/Connection.cpp

14.3 ism_stat_t Struct Reference

Easy access to ISM statistics. Request with ism_request_stat.

```
#include <ism3.h>
```

Public Attributes

- uint32_t rxOk
  number of data frame correctly received
- uint32_t rxCrcError
  number of data frame received with CRC error
- uint32_t tx
  number of data frame transmission
- uint32_t txLbtFail
  number of data frame LBT failure
- uint32_t txAck
  number of data frame acknowledged
- uint32_t txNack
  number of data frame not acknowledged
- uint32_t syncRxOk
  number of successful sync frame reception
- uint32_t syncRxCrcError
  number of CRC error in sync frame reception
- uint32_t syncRxBadFrame
  number of bad sync frame received
- uint32_t syncRxTimeout
  number of sync frame reception timeout
- uint32_t syncRxLost
  number of synchronization lost
- int16_t syncMinDelta
  Minimum value for the delta of synchronization in tick.
- int16_t syncMaxDelta
  Maximum value for the delta of synchronization in tick.
- int32_t syncSumDelta
  Sum of all the delta of synchronization in tick.
- uint32_t lpsyncRxOk
  number of successful sync frame reception
- uint32_t lpsyncRxCrcError
  number of CRC error in sync frame reception
- uint32_t lpsyncRxBadFrame
  number of bad sync frame received
- uint32_t lpsyncRxTimeout
  number of sync frame reception timeout
- uint32_t lpsyncRxLost
  number of synchronization lost
- int16_t lpsyncMinDelta
  Minimum value for the delta of synchronization in tick.
- int16_t lpsyncMaxDelta
  Maximum value for the delta of synchronization in tick.
- int32_t lpsyncSumDelta
  Sum of all the delta of synchronization in tick.
- uint32_t syncTxOk
  number of sync frame transmission
- uint32_t syncTxLbtFail
  number of sync frame LBT failure
- uint32_t rxScanTime

Time spend in reception during scan in second.

- uint32_t rxTime

  total time spend in reception (excepted scan) in second
- uint32_t txTime

  total time spend in transmission in second
- uint32_t txUnicast

  Number of unicast demand (return no error)
- uint32_t txUnicastAck

  Number of acknowledged unicast.
- uint32_t txUnicastNack

  Number of not acknowledged unicast.
- uint32_t txMulticast

  Number of multicast demand (return no error)
- uint32_t txMulticastAttempt

  Number of multicast transmission attempt (txMulticast ∗ (countdown + 1))
- uint32_t rxUnicastOk

  Number of correctly received unicast frame.
- uint32_t rxMulticastOk

  Number of correctly received multicast frame.
- uint32_t rxBadFrame

  Number of wrong frame type or size in a data slot.
- uint32_t rxWrongBeaconId

  Number of received frame with the wrong beacon id.
- uint32_t rxUnicastDuplicate

  Number of duplicate unicast frame received.
- uint32_t rxUnicastWrongAddress

  Number of unicast frame with wrong destination address received.
- uint32_t rxMulticastWrongGroup

  Number of multicast frame with wrong destination group received.
- uint32_t scanOnPhase

  Number of phase with active scanning.
- uint32_t scanLock

  Number of time a channel is locked because of RSSI over threshold.
- uint32_t scanLockTimeout

  Number of time a locked channel timeout because of lack of pattern received.
- uint32_t scanLockRssiTooLow

  Number of time a locked channel end because of a RSSI under threshold.
- uint32_t scanRxCrcError

  Number of CRC error in scan frame reception.
- uint32_t scanRxBadFrame

  Number of bad scan frame received.
- uint32_t scanRefuseUnassociated

  Number of synchronization fail because of beacon refuse unassociated.
- uint32_t scanSuccess

  Number of scan success.

14.3.1 Detailed Description

Easy access to ISM statistics. Request with ism_request_stat.

The documentation for this struct was generated from the following file:

- ISM3_Linux/ism3.h

14.4 sDatagram Struct Reference

local datagram

`#include <wpan.h>`

Public Attributes

- bool ready
  Whether the datagram has received its last packet.
- datagram_id_t id
  Datagram ID.
- uint16_t size
  Data size.
- uint8_t * data
  Pointer to data.

14.4.1 Detailed Description

local datagram

This data type is ready to be filled with Data packets. Enough memory should be allocated to contain all expected packets for the datagram. Size should be updated on last packet reception to reflect actual data size.

The documentation for this struct was generated from the following file:

- Protocol/wpan.h

14.5 sManifest Struct Reference

power node description

`#include <wpan.h>`

Public Attributes

- uint8_t priority
  Lower is more priority, 0=do not adjust.
- uint8_t descriptionLength
  Node description length.
- char * description
  Node description.

14.5.1 Detailed Description

power node description

Valuable information for a household power regulation application

The documentation for this struct was generated from the following file:

- Protocol/wpan.h

14.6 sPowerSettings Struct Reference

power settings

`#include <wpan.h>`

Public Attributes

- uint8_t nPowerSettings
  Number of power settings.
- uint8_t powerSettingWidth
  Length of one power setting (float=4)
- powerTarget_t * powerSettingskW
  *Power setting array, could be switched to void * type.*

14.6.1 Detailed Description

power settings

Wrapper structure to conveniently send and receive power settings.

The documentation for this struct was generated from the following file:

- Protocol/wpan.h

14.7 uartParam_s Struct Reference

UART parameter holder.

Public Attributes

- parity_t parity
  parity
- uint32_t word_length
  word length in bits
- flowControl_t flow_control
  flow control setting
- stopBits_t stopBits
  stop bits setting
- uint32_t baudrate
  desired baudrate

14.7.1 Detailed Description

UART parameter holder.

The documentation for this struct was generated from the following file:

- ISM3_Linux/buffered_uart.c

Chapter 15

File Documentation

15.1 ISM3_Linux/buffered_uart.c File Reference

Driver for UART using circular buffer.

```
#include <assert.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <sys/signal.h>
#include <sys/types.h>
#include "buffered_uart.h"
#include "hardware.h"
```

Classes

- struct uartParam_s

  UART parameter holder.

Macros

- #define **RX_BUFFER_SIZE** 512
- #define **TX_BUFFER_SIZE** 1024
- #define **N_BAUD_SETTINGS** 22

Functions

- void **buffered_uart_init** (uint32_t baudrate, uint32_t word_length, parity_t parity, flowControl_t flow_control)
- void **buffered_uart_deinit** (void)
- void **buffered_uart_start** (void)
- void **buffered_uart_set_baudrate** (uint32_t baudrate)
- uint16_t **buffered_uart_read** (uint8_t ∗data, uint16_t size)
- uint16_t **buffered_uart_write** (const uint8_t ∗data, uint16_t size)
- uint16_t **buffered_uart_get_read_available** (void)
- uint16_t **buffered_uart_get_write_available** (void)
- bool **buffered_uart_is_busy** (void)

Variables

- uartParam_s **uConfig**

15.1.1 Detailed Description

Driver for UART using circular buffer.

Author

nicolas.brunner@heig-vd.ch

Date

07-August-2018

Copyright

HEIG-VD

License information

15.2 ISM3_Linux/buffered_uart.h File Reference

Driver for UART using circular buffer.

```
#include <stdbool.h>
#include <stdint.h>
#include "hardware.h"
```

Enumerations

- enum parity_t { **PARITY_NONE** , **PARITY_EVEN** , **PARITY_ODD** }
  UART parity setting.
- enum stopBits_t { **STOP_BITS_1** , **STOP_BITS_2** }
  UART stop bits setting.
- enum flowControl_t { **FLOW_CONTROL_NONE** , **FLOW_CONTROL_RTS_CTS** }
  UART flow control setting.

Functions

- void **buffered_uart_init** (uint32_t baudrate, uint32_t word_length, parity_t parity, flowControl_t flow_control)
- void **buffered_uart_deinit** (void)
- void **buffered_uart_start** (void)
- void **buffered_uart_set_baudrate** (uint32_t baudrate)
- uint16_t **buffered_uart_read** (uint8_t *data, uint16_t size)
- uint16_t **buffered_uart_write** (const uint8_t *data, uint16_t size)
- uint16_t **buffered_uart_get_read_available** (void)
- uint16_t **buffered_uart_get_write_available** (void)
- bool **buffered_uart_is_busy** (void)

15.2.1 Detailed Description

Driver for UART using circular buffer.

Author

nicolas.brunner@heig-vd.ch

Date

07-August-2018

Copyright

License information

15.3 ISM3_Linux/commands_RM1S3.h File Reference

List of the host commands.

Macros

- #define **CMD_GET_FIRMWARE_VERSION** 0x11
- #define **CMD_GET_HARDWARE_VERSION** 0x13
- #define **CMD_GET_UNIQUE_DEVICE_ID** 0x15
- #define **CMD_SET_DATA_SLOT_COUNT** 0x20
- #define **CMD_GET_DATA_SLOT_COUNT** 0x21
- #define **CMD_GET_PROTOCOL_STATE** 0x27
- #define **CMD_CLEAR_STAT** 0x2A
- #define **CMD_GET_STAT** 0x2B
- #define **CMD_SET_ACTIVE_CHANNELS** 0x2C
- #define **CMD_GET_ACTIVE_CHANNELS** 0x2D
- #define **CMD_SET_DATA_SLOT_RANGE_TYPE** 0x3A
- #define **CMD_SET_DATA_SLOT_TYPE** 0x3C
- #define **CMD_GET_DATA_SLOT_TYPE** 0x3D
- #define **CMD_GET_LAST_RX_SYNC_INFO** 0x31
- #define **CMD_SET_PATTERN** 0x40
- #define **CMD_GET_PATTERN** 0x41
- #define **CMD_SET_ENCRYPTION_KEY** 0x42
- #define **CMD_SET_ADDRESS** 0x44
- #define **CMD_GET_ADDRESS** 0x45
- #define **CMD_SET_GROUP** 0x4C
- #define **CMD_GET_GROUP** 0x4D
- #define **CMD_SET_TX_RETRY_COUNT** 0x5C
- #define **CMD_GET_TX_RETRY_COUNT** 0x5D
- #define **CMD_SET_RF_POWER** 0x60
- #define **CMD_GET_RF_POWER** 0x61
- #define **CMD_SET_RF_PHY** 0x62

- #define **CMD_GET_RF_PHY** 0x63
- #define **CMD_GET_PHYS_CHANNELS_PLAN_SIZE** 0x65
- #define **CMD_SET_RF_POWER_DBM** 0x66
- #define **CMD_SET_RADIO_MODE** 0x70
- #define **CMD_GET_RADIO_MODE** 0x71
- #define **IND_TDMA_STATE_CHANGED** 0x73
- #define **CMD_SET_EVENT_INDICATION** 0x74
- #define **CMD_GET_EVENT_INDICATION** 0x75
- #define **CMD_UNLOCK_FACTORY_CMD** 0x90
- #define **CMD_GET_RF_TEST_BER** 0x93
- #define **CMD_SET_RF_TEST_CHANNEL** 0x94
- #define **CMD_GET_RF_TEST_CHANNEL** 0x95
- #define **CMD_SET_RF_TEST_MODE** 0x96
- #define **CMD_GET_RF_TEST_MODE** 0x97
- #define **CMD_GET_RSSI** 0x99
- #define **CMD_SET_SYNC_MODE** 0xA0
- #define **CMD_GET_SYNC_MODE** 0xA1
- #define **CMD_SET_SYNC_RX** 0xA2
- #define **CMD_GET_SYNC_RX** 0xA3
- #define **CMD_SET_SYNC_RX_LOW_POWER** 0xA4
- #define **CMD_GET_SYNC_RX_LOW_POWER** 0xA5
- #define **CMD_SET_SYNC_TX_LOW_POWER** 0xA6
- #define **CMD_GET_SYNC_TX_LOW_POWER** 0xA7
- #define **CMD_SET_SYNC_BEACON_ID** 0xA8
- #define **CMD_GET_SYNC_BEACON_ID** 0xA9
- #define **CMD_SET_SYNC_USER_DATA** 0xAA
- #define **CMD_GET_SYNC_USER_DATA** 0xAB
- #define **CMD_GET_SYNC_TX_INTERVAL** 0xAD
- #define **CMD_SET_ASSOCIATED_BEACON_ID** 0xB0
- #define **CMD_GET_ASSOCIATED_BEACON_ID** 0xB1
- #define **CMD_SET_ACCEPT_UNASSOCIATED** 0xB2
- #define **CMD_GET_ACCEPT_UNASSOCIATED** 0xB3
- #define **CMD_SET_HOST_BAUDRATE** 0xC4
- #define **CMD_GET_HOST_BAUDRATE** 0xC5
- #define **CMD_SET_TX_RETRY_RESTRICTION** 0xC8
- #define **CMD_GET_TX_RETRY_RESTRICTION** 0xC9
- #define **CMD_GET_RF_FRAME_MAX_SIZE** 0xCB
- #define **CMD_SET_HOST_UART_SETTINGS** 0xCC
- #define **CMD_GET_HOST_UART_SETTINGS** 0xCD
- #define **CMD_SET_GPIO0_SIGNAL** 0xE0
- #define **CMD_GET_GPIO0_SIGNAL** 0xE1
- #define **CMD_SET_GPIO1_SIGNAL** 0xE2
- #define **CMD_GET_GPIO1_SIGNAL** 0xE3
- #define **CMD_SET_GPIO2_SIGNAL** 0xE4
- #define **CMD_GET_GPIO2_SIGNAL** 0xE5
- #define **CMD_SET_GPIO3_SIGNAL** 0xE6
- #define **CMD_GET_GPIO3_SIGNAL** 0xE7
- #define **CMD_SET_GPIO_OUT** 0xE8
- #define **CMD_GET_GPIO_IN** 0xE9
- #define **CMD_SET_CTS** 0xEA
- #define **CMD_GET_RTS** 0xEB
- #define **CMD_CLEAR_CLOCK_CHECK** 0xEC
- #define **CMD_GET_CLOCK_CHECK** 0xED
- #define **CMD_SEND_UNICAST** 0xD2
- #define **IND_RECEIVED_UNICAST** 0xD3

- #define **CMD_SEND_MULTICAST** 0xD4
- #define **IND_RECEIVED_MULTICAST** 0xD5
- #define **IND_UPDATED_SYNC_DATA** 0xD9
- #define **CMD_LOAD_DEFAULT_PARAMETERS** 0xF6
- #define **CMD_CLEAR_TIMESLOT_MAX_VALUES** 0x3E
- #define **CMD_GET_TIMESLOT_MAX_VALUES** 0x3F
- #define **CMD_GET_DEBUG_DATA** 0xFF
- #define **IND_ERROR** 0x03

15.3.1 Detailed Description

List of the host commands.

Author

nicolas.brunner@heig-vd.ch

Date

25-March-2013

Copyright

HEIG-VD

License information

15.4 ISM3_Linux/framed_uart.c File Reference

Driver for UART using frame.

```
#include <assert.h>
#include <time.h>
#include <stdio.h>
#include "buffered_uart.h"
#include "framed_uart.h"
```

Macros

- #define **RX_MAX_FRAME_SIZE** 256

Functions

- void **framed_uart_init** (framed_uart_function_t rx_function_, uint32_t frame_timeout_)
- void **framed_uart_deinit** (void)
- void **framed_uart_start** (void)
- bool **framed_uart_tx** (const uint8_t *data, uint16_t size)
- void **framed_uart_set_baudrate** (uint32_t baudrate)
- void **framed_uart_flush** (void)
- void **framed_uart_tick** (void)
- bool **framed_uart_is_busy** (void)

15.4.1 Detailed Description

Driver for UART using frame.

Author

nicolas.brunner@heig-vd.ch

Date

06-August-2018

Copyright

License information

15.5 ISM3_Linux/framed_uart.h File Reference

Driver for UART using frame.

```
#include <stdbool.h>
#include <stdint.h>
```

Typedefs

- typedef void(∗ **framed_uart_function_t**) (const uint8_t ∗data, uint16_t size)

Functions

- void **framed_uart_init** (framed_uart_function_t rx_function, uint32_t frame_timeout)
- void **framed_uart_deinit** (void)
- void **framed_uart_start** (void)
- void **framed_uart_set_baudrate** (uint32_t baudrate)
- void **framed_uart_flush** (void)
- void **framed_uart_tick** (void)
- bool **framed_uart_tx** (const uint8_t ∗data, uint16_t size)
- bool **framed_uart_is_busy** (void)

15.5.1 Detailed Description

Driver for UART using frame.

Author

nicolas.brunner@heig-vd.ch

Date

06-August-2018

Copyright

HEIG-VD

License information

15.6 ISM3_Linux/hardware.h File Reference

Hardware definitions for RM1S3 driver.

Macros

- #define UART_DEV "/dev/ttyAMA1"
  Linux hardware serial port.
- #define **DEBUG**

15.6.1 Detailed Description

Hardware definitions for RM1S3 driver.

15.7 ISM3_Linux/ism3.c File Reference

Driver for the RM1S3.

```
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "commands_RM1S3.h"
#include "framed_uart.h"
#include "ism3.h"
#include "hardware.h"
#include "util.h"
```

Macros

- #define **RESET_DURATION** 10
- #define **START_DURATION** 500
- #define **FRAME_TIMEOUT** 150
- #define **BAUDRATE_CHANGE_DURATION** 5
- #define BAUDRATE 19200

  Hardware serial link to ISM3 baudrate.
- #define **TX_HEADER_SIZE** 4
- #define **BROADCAST_HEADER_SIZE** TX_HEADER_SIZE+4
- #define **RX_HEADER_SIZE** 6
- #define **SOURCE_INDEX** 2
- #define **DATA_SLOT_INDEX** 3
- #define **RSSI_INDEX** 4
- #define **LQI_INDEX** 5
- #define **RX_MULTICAST_HEADER_SIZE** 11
- #define **RX_MULTICAST_SOURCE_INDEX** 2
- #define **RX_MULTICAST_GROUP_INDEX** 3
- #define **RX_MULTICAST_COUNTDOWN_INDEX** 7
- #define **RX_MULTICAST_RSSI_INDEX** 9
- #define **RX_MULTICAST_LQI_INDEX** 10
- #define **GROUP_SIZE** 4
- #define **MAX_COMMAND_SIZE** 18
- #define **NUMBER_OF_RECONFIGURATION_CMD** 9
- #define **STATE_UNINITIALIZED** 0xFF
- #define **TX_STATUS_NONE** 0
- #define **TX_STATUS_WAIT_ACK** 1
- #define **TX_STATUS_ACK** 2
- #define **TX_STATUS_TIMEOUT** 3
- #define **DEFAULT_PHY1** 0
- #define **DEFAULT_PHY2** 3
- #define **UNALLOWED_PHY1** 1
- #define **UNALLOWED_PHY2** 2
- #define **DEFAULT_POWER** 0x06
- #define **DEFAULT_POWER_DBM** 14
- #define **NUMBER_OF_GPIO** 9

Typedefs

- typedef const uint8_t **commands_t**[ ][MAX_COMMAND_SIZE]

Functions

- void ism_init (ism_unicast_function_t rx_unicast_function_, ism_multicast_function_t rx_multicast_↩ function_, ism_beacon_data_function_t beacon_data_function_, ism_state_function_t state_function_↩ , ism_stat_function_t stat_function_)

  Initialize ISM module.
- void ism_config (uint8_t address_, uint32_t group_, uint8_t power_, uint8_t power_dbm_, uint64_↩ t associated_beacon_id_)

  Configure the ISM.
- void ism_get_config (uint8_t *address_, uint32_t *group_, uint8_t *power_, uint8_t *power_dbm_, uint64_t *associated_beacon_id_)

Get current ISM config.

- void ism_get_uid (uint8_t *uid_, uint8_t uid_size_)

  Get module unique identifier.
- bool ism_set_phy (uint8_t phy_, const uint8_t *channels_)
- void ism_disconnect (void)

  Stop ISM module and restart.
- void ism_set_sync_mode (ism_sync_mode_t mode)

  Set ISM sync mode.
- void **ism_power_down** (void)
- void ism_tick (void)

  Tick. Poll for new RX data or need for module reconfiguration.
- bool **ism_is_tx_pending** (void)
- bool **ism_is_ready** (void)
- bool ism_tx (uint8_t destination, const uint8_t *data, uint8_t size)

  TX unicast frame.
- bool ism_broadcast (uint32_t group, uint8_t number, const uint8_t *data, uint8_t size)

  TX multicast frame.
- uint8_t **ism_get_max_data_size** (void)
- char * **ism_get_firmware_version** (void)
- uint32_t ism_get_firmware_version_value (void)

  Get the firmware version coded into an integer. Version a.b.c => a << 16 + b << 8 + c.
- char * **ism_get_hardware_version** (void)
- bool ism_request_stat (void)

  Request communication statistics from ISM module.
- bool ism_request_state (void)

  Request TDMA state from ISM module.
- bool **ism_update_firmware** (const uint8_t *firmware, uint32_t size)
- uint32_t **ism_get_uart_rx_counter** (void)
- uint8_t **ism_get_channels_size** (uint8_t phy)
- bool **send_command** (const uint8_t *data)
- void **EXTI15_10_IRQHandler** (void)

15.7.1 Detailed Description

Driver for the RM1S3.

Author

nicolas.brunner@heig-vd.ch, marc.leemann@master.hes-so.ch

Date

06-August-2018

Copyright

HEIG-VD

15.7.2 Macro Definition Documentation

15.7.2.1 BAUDRATE

```
#define BAUDRATE 19200
```

Hardware serial link to ISM3 baudrate.

Must be lower than 30000 for using stop2 mode of RM1

15.7.3 Function Documentation

15.7.3.1 ism_broadcast()

```
bool ism_broadcast (
            uint32_t group,
            uint8_t number,
            const uint8_t * data,
            uint8_t size )
```

TX multicast frame.

Parameters

group	groups to broadcast to
number	number of frames to send
data	pointer to data to transmit
size	size of data to transmit (<ISM_MAX_DATA_SIZE)

Returns

true if frame was sent to module

15.7.3.2 ism_config()

```
void ism_config (
            uint8_t address,
            uint32_t group,
            uint8_t power,
            uint8_t power_dbm,
            uint64_t associated_beacon_id )
```

Configure the ISM.

Parameters

address	module address

Parameters

group	module group
power	desired power setting
power_dbm	matching power in dBm
associated_beacon↩ _id	beacon ID to synchronize to

15.7.3.3 ism_get_config()

```
void ism_get_config (
            uint8_t * address,
            uint32_t * group,
            uint8_t * power,
            uint8_t * power_dbm,
            uint64_t * associated_beacon_id )
```

Get current ISM config.

Parameters

address	fill this memory zone with module address
group	fill this memory zone with module group
power	fill this memory zone with desired power setting
power_dbm	fill this memory zone with matching power in dBm
associated_beacon↩ _id	fill this memory zone with beacon ID to synchronize to

15.7.3.4 ism_get_firmware_version_value()

```
uint32_t ism_get_firmware_version_value (
            void  )
```

Get the firmware version coded into an integer. Version a.b.c => a << 16 + b << 8 + c.

Returns

the firmware version coded into an integer

15.7.3.5 ism_get_uid()

```
void ism_get_uid (
            uint8_t * uid_,
            uint8_t uid_size_ )
```

Get module unique identifier.

Parameters

*uid_*	pointer to array to store uid into
*uid_↩ size_*	size of array. Must be > UID_SIZE

15.7.3.6 ism_init()

```
void ism_init (
            ism_unicast_function_t rx_unicast_function,
            ism_multicast_function_t rx_multicast_function,
            ism_beacon_data_function_t beacon_data_function,
            ism_state_function_t state_function,
            ism_stat_function_t stat_function )
```

Initialize ISM module.

Parameters

rx_unicast_function	the function called when unicast are received
rx_multicast_function	the function called when multicast data are received
beacon_data_function	the function called when beacon data are received
state_function	the function called when the state change
stat_function	the function called when stat are read

Initialize serial communication hardware and handler functions

Todo configure RESET pin and use it as in this example snippet

HAL_GPIO_WritePin(((pin_t)ISM_NRESET).GPIOx, ((pin_t)ISM_NRESET).GPIO_Pin, GPIO_PIN_RESET); HAL_GPIO_WritePin(((pin_t)ISM_BOOT0).GPIOx, ((pin_t)ISM_BOOT0).GPIO_Pin, GPIO_PIN_RESET); HAL_↩ GPIO_WritePin(((pin_t)ISM_NPEN).GPIOx, ((pin_t)ISM_NPEN).GPIO_Pin, GPIO_PIN_RESET);

HAL_Delay(RESET_DURATION); HAL_GPIO_WritePin(((pin_t)ISM_NRESET).GPIOx, ((pin_t)ISM_NRESET).GPIO↩ _Pin, GPIO_PIN_SET); HAL_Delay(START_DURATION);

15.7.3.7 ism_set_phy()

```
bool ism_set_phy (
            uint8_t phy,
            const uint8_t * channels )
```

Set the physical layer parameters, they will be use only after a ism_disconnect()

15.7.3.8 ism_set_sync_mode()

```
void ism_set_sync_mode (
            ism_sync_mode_t mode )
```

Set ISM sync mode.

Parameters

mode	new sync mode

15.7.3.9 ism_tx()

```
bool ism_tx (
            uint8_t destination,
            const uint8_t * data,
            uint8_t size )
```

TX unicast frame.

Parameters

destination	destination address
data	pointer to data to transmit
size	size of data to transmit (<ISM_MAX_DATA_SIZE)

Returns

true if frame was sent to module

15.8 ISM3_Linux/ism3.h File Reference

Driver for the RM1S3.

```
#include <stdbool.h>
#include <stdint.h>
```

Classes

- struct ism_stat_t

  Easy access to ISM statistics. Request with ism_request_stat.

Macros

- #define ISM_TIMESLOT_DURATION 20

  Timeslot duration.
- #define ISM_MAX_DATA_SIZE 239

  Max frame size for TX.
- #define ISM_INVALID_POWER 0xFF

  Power error code.
- #define ISM_MAX_POWER 52

  Max power.
- #define ISM_MAX_POWER_DBM 30

  Max power in dBm.
- #define UID_SIZE 12

  UID size in bytes.

Typedefs

- typedef void(∗ ism_unicast_function_t) (const uint8_t ∗data, uint8_t size, uint8_t source, int8_t rssi, uint8_t lqi)

  type of unicast RX handler
- typedef void(∗ ism_multicast_function_t) (const uint8_t ∗data, uint8_t size, uint8_t source, uint8_t countdown, int8_t rssi, uint8_t lqi)

  type of multicast RX handler
- typedef void(∗ ism_beacon_data_function_t) (const uint8_t ∗data, uint8_t size)

  type of beacon data change handler
- typedef void(∗ ism_state_function_t) (ism_state_t state, const uint8_t ∗gateway_id)

  type of TDMA state change handler
- typedef void(∗ ism_stat_function_t) (ism_stat_t stat)

  type of statistics RX handler

Enumerations

- enum ism_state_t { **ISM_OFF** , **ISM_NOT_SYNCHRONIZED** , **ISM_SYNCHRONIZED** , **ISM_LOW_POWER_SYNC** , **ISM_TX_SYNC** , **ISM_VERSION_READY** }

  ISM TDMA state. See ISM3 states.
- enum ism_sync_mode_t { SM_TX , SM_RX_ACTIVE , SM_RX_LOW_POWER , SM_RX_LOW_POWER_GROUP }

  ISM sync mode. See ISM3 sync modes.

Functions

- void ism_init (ism_unicast_function_t rx_unicast_function, ism_multicast_function_t rx_multicast↩ _function, ism_beacon_data_function_t beacon_data_function, ism_state_function_t state_function, ism_stat_function_t stat_function)

  Initialize ISM module.
- void ism_config (uint8_t address, uint32_t group, uint8_t power, uint8_t power_dbm, uint64_t associated_↩ beacon_id)

  Configure the ISM.
- void ism_get_config (uint8_t ∗address, uint32_t ∗group, uint8_t ∗power, uint8_t ∗power_dbm, uint64_↩ t ∗associated_beacon_id)

  Get current ISM config.
- void ism_get_uid (uint8_t ∗uid_, uint8_t uid_size_)

  Get module unique identifier.
- bool ism_set_phy (uint8_t phy, const uint8_t ∗channels)
- void ism_disconnect (void)

  Stop ISM module and restart.
- void ism_set_sync_mode (ism_sync_mode_t mode)

  Set ISM sync mode.
- void ism_tick (void)

  Tick. Poll for new RX data or need for module reconfiguration.
- bool ism_tx (uint8_t destination, const uint8_t ∗data, uint8_t size)

  TX unicast frame.
- bool ism_broadcast (uint32_t group, uint8_t number, const uint8_t ∗data, uint8_t size)

  TX multicast frame.
- bool **ism_is_tx_pending** (void)

- bool **ism_is_ready** (void)
- uint8_t **ism_get_max_data_size** (void)
- char * **ism_get_firmware_version** (void)
- uint32_t ism_get_firmware_version_value (void)

  Get the firmware version coded into an integer. Version a.b.c => a << 16 + b << 8 + c.
- char * **ism_get_hardware_version** (void)
- bool **ism_update_firmware** (const uint8_t *firmware, uint32_t size)
- bool ism_request_stat (void)

  Request communication statistics from ISM module.
- bool ism_request_state (void)

  Request TDMA state from ISM module.
- uint32_t **ism_get_uart_rx_counter** (void)
- uint8_t **ism_get_channels_size** (uint8_t phy)
- bool **send_command** (const uint8_t *data)

15.8.1 Detailed Description

Driver for the RM1S3.

Author

nicolas.brunner@heig-vd.ch, marc.leemann@master.hes-so.ch

Date

06-August-2018

Copyright

HEIG-VD

Original STM32 driver by Nicolas Brunner. Linux port and documentation by Marc Leemann.

15.8.2 Enumeration Type Documentation

15.8.2.1 ism_sync_mode_t

enum ism_sync_mode_t

ISM sync mode. See ISM3 sync modes.

Enumerator

SM_TX	TX mode.
SM_RX_ACTIVE	RX async mode (awake)
SM_RX_LOW_POWER	RX low power mode (only beacon data)
SM_RX_LOW_POWER_GROUP	RX low power group wake mode.

15.8.3 Function Documentation

15.8.3.1 ism_broadcast()

```
bool ism_broadcast (
            uint32_t group,
            uint8_t number,
            const uint8_t * data,
            uint8_t size )
```

TX multicast frame.

Parameters

group	groups to broadcast to
number	number of frames to send
data	pointer to data to transmit
size	size of data to transmit (<ISM_MAX_DATA_SIZE)

Returns

true if frame was sent to module

15.8.3.2 ism_config()

```
void ism_config (
            uint8_t address,
            uint32_t group,
            uint8_t power,
            uint8_t power_dbm,
            uint64_t associated_beacon_id )
```

Configure the ISM.

Parameters

address	module address
group	module group
power	desired power setting
power_dbm	matching power in dBm
associated_beacon↩ _id	beacon ID to synchronize to

15.8.3.3 ism_get_config()

```
void ism_get_config (
            uint8_t * address,
            uint32_t * group,
            uint8_t * power,
            uint8_t * power_dbm,
            uint64_t * associated_beacon_id )
```

Get current ISM config.

Parameters

address	fill this memory zone with module address
group	fill this memory zone with module group
power	fill this memory zone with desired power setting
power_dbm	fill this memory zone with matching power in dBm
associated_beacon_id	fill this memory zone with beacon ID to synchronize to

15.8.3.4 ism_get_firmware_version_value()

```
uint32_t ism_get_firmware_version_value (
            void  )
```

Get the firmware version coded into an integer. Version a.b.c => a << 16 + b << 8 + c.

Returns

the firmware version coded into an integer

15.8.3.5 ism_get_uid()

```
void ism_get_uid (
            uint8_t * uid_,
            uint8_t uid_size_ )
```

Get module unique identifier.

Parameters

*uid_*	pointer to array to store uid into
*uid_size_*	size of array. Must be > UID_SIZE

15.8.3.6 ism_init()

```
void ism_init (
            ism_unicast_function_t rx_unicast_function,
            ism_multicast_function_t rx_multicast_function,
            ism_beacon_data_function_t beacon_data_function,
            ism_state_function_t state_function,
            ism_stat_function_t stat_function )
```

Initialize ISM module.

Parameters

rx_unicast_function	the function called when unicast are received
rx_multicast_function	the function called when multicast data are received
beacon_data_function	the function called when beacon data are received
state_function	the function called when the state change
stat_function	the function called when stat are read

Initialize serial communication hardware and handler functions

Todo configure RESET pin and use it as in this example snippet

HAL_GPIO_WritePin(((pin_t)ISM_NRESET).GPIOx, ((pin_t)ISM_NRESET).GPIO_Pin, GPIO_PIN_RESET); HAL_GPIO_WritePin(((pin_t)ISM_BOOT0).GPIOx, ((pin_t)ISM_BOOT0).GPIO_Pin, GPIO_PIN_RESET); HAL_GPIO_WritePin(((pin_t)ISM_NPEN).GPIOx, ((pin_t)ISM_NPEN).GPIO_Pin, GPIO_PIN_RESET);

HAL_Delay(RESET_DURATION); HAL_GPIO_WritePin(((pin_t)ISM_NRESET).GPIOx, ((pin_t)ISM_NRESET).GPIO_Pin, GPIO_PIN_SET); HAL_Delay(START_DURATION);

15.8.3.7 ism_set_phy()

```
bool ism_set_phy (
            uint8_t phy,
            const uint8_t * channels )
```

Set the physical layer parameters, they will be use only after a ism_disconnect()

15.8.3.8 ism_set_sync_mode()

```
void ism_set_sync_mode (
            ism_sync_mode_t mode )
```

Set ISM sync mode.

Parameters

mode	new sync mode

15.8.3.9 ism_tx()

```
bool ism_tx (
            uint8_t destination,
            const uint8_t * data,
            uint8_t size )
```

TX unicast frame.

Parameters

destination	destination address
data	pointer to data to transmit
size	size of data to transmit (<ISM_MAX_DATA_SIZE)

Returns

true if frame was sent to module

15.9 ISM3_Linux/util.c File Reference

Utility library, use big endian.

```
#include <stdbool.h>
#include "util.h"
```

Functions

- uint8_t **util_get_number_of_bit_set** (uint8_t value)

15.9.1 Detailed Description

Utility library, use big endian.

Author

nicolas.brunner@heig-vd.ch & laurent.folladore@heig-vd.ch

Date

06-September-2012

Copyright

HEIG-VD

License information

15.10 ISM3_Linux/util.h File Reference

Utility library, use big endian.

```
#include <stdint.h>
```

Macros

- #define **UTIL_MIN**(x, y) ((x) < (y) ? (x) : (y))
- #define **UTIL_MAX**(x, y) ((x) > (y) ? (x) : (y))
- #define **UTIL_CEILING**(x, y) (((x) + (y) - 1) / (y))

Functions

- uint8_t **util_get_number_of_bit_set** (uint8_t value)

15.10.1 Detailed Description

Utility library, use big endian.

Author

nicolas.brunner@heig-vd.ch & laurent.folladore@heig-vd.ch

Date

06-September-2012

Copyright

HEIG-VD

License information

15.11 main.cpp File Reference

Testing program for WPAN Gateway application.

```
#include "main.h"
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <csignal>
#include "powernode.h"
#include "menu.h"
```

Functions

- void dispatchRxFrames (const uint8_t *data, uint8_t size, uint8_t source)

  Dispatch RX frames to clients based on source address.
- vector< Node > getStaticAddressList (string file)

  Get statically-addressed Nodes from local file.
- void signalHandler (int signum)

  Interrupt signal handler.
- int main (void)

  The application entry point.

Variables

- wpanManager * **instance**

15.11.1 Detailed Description

Testing program for WPAN Gateway application.

15.11.2 Function Documentation

15.11.2.1 dispatchRxFrames()

```
void dispatchRxFrames (
            const uint8_t * data,
            uint8_t size,
            uint8_t source )
```

Dispatch RX frames to clients based on source address.

Parameters

data	RX frame data
size	RX frame size
source	source address

Called from rx_unicast_handler in ism3_handlers

15.11.2.2 getStaticAddressList()

```
vector< Node > getStaticAddressList (
            string file )
```

Get statically-addressed Nodes from local file.

Parameters

file	configuration file path

Returns

vector of Node objects

Nodes in file must be defined like this (decimal) :

address group

Text after # is ignored

15.11.2.3 main()

```
int main (
            void  )
```

The application entry point.

Return values

int	

15.11.2.4 signalHandler()

```
void signalHandler (
            int signum )
```

Interrupt signal handler.

Parameters

signum	signal ID from OS

Exits program gratiously on interrupt signal

15.12 main.h File Reference

Header for main.c file.

```
#include <cstdint>
```

Functions

- void dispatchRxFrames (const uint8_t *data, uint8_t size, uint8_t source)

  Dispatch RX frames to clients based on source address.

15.12.1 Detailed Description

Header for main.c file.

15.12.2 Function Documentation

15.12.2.1 dispatchRxFrames()

```
void dispatchRxFrames (
            const uint8_t * data,
            uint8_t size,
            uint8_t source )
```

Dispatch RX frames to clients based on source address.

Parameters

data	RX frame data
size	RX frame size
source	source address

Called from rx_unicast_handler in ism3_handlers

15.13 Protocol/wpan.h File Reference

Definitions for WPAN protocols.

```
#include <stdbool.h>
```

Classes

- struct sPowerSettings

  power settings
- struct sManifest

  power node description
- struct sDatagram

  local datagram

Macros

- #define DEBUG_DATA

  Print DATA_PROTOCOL RX/TX frames.
- #define DEBUG_DORA

  Print debug messages for DORA protocol.
- #define DEBUG_CONNECTION

  Print connection debug messages.
- #define SHOW_TASKS

  Print general console messages.
- #define **DEBUG_BORDERROUTER**
- #define NETWORK_PROTOCOL_ID 0x01

  Network protocol ID.
- #define NETWORK_PING 0x01

  Ping.
- #define NETWORK_GETUID 0x02

  Get UID.
- #define NETWORK_SETADDR 0x04

  Set node address.
- #define NETWORK_SETGROUP 0x05

  Set node group.
- #define NETWORK_DISCONNECT 0x08

  Disconnect node.
- #define NETWORK_RENEW_LEASE 0x14

  Renew dynamic address lease.
- #define NETWORK_GET_PROTOCOLS 0x20

  Get Node supported protocols.
- #define NETWORK_DISCOVER 0x10

  Discover command.
- #define NETWORK_OFFER 0x11

  Offer address.
- #define NETWORK_REQUEST 0x12

  Request address and group.
- #define NETWORK_ACK 0x13

  NETWORK_ACK.
- #define NETWORK_DORA_ERR 0x18

  Error command.
- #define NETWORK_DORA_GROUP 0x80000000

  Default group for DORA command transmission.
- #define NETWORK_NACK_BASE_ADDR 0x04

  Default address for non-configured nodes.
- #define NETWORK_LEASE_UNIT_MINUTES 1

  Lease to minute conversion.
- #define NETWORK_LEASE_WIDTH 1

  Width of network lease in DORA commands.
- #define NETWORK_UID8_WIDTH 12

  Width of ISM3 UID.
- #define NETWORK_ADDR_WIDTH 1

  Width of ISM3 addresses.
- #define NETWORK_GROUP_WIDTH 4

  Width of ISM3 group bitfield.

- #define NETWORK_DORA_MAX_FRAME_SIZE (2∗NETWORK_UID8_WIDTH+NETWORK_ADDR_WIDTH+NETWORK_GR

  Max DORA frame size excluding protocol ID and command. Based on ACK frame length.
- #define NETWORK_MAX_FRAME_SIZE (sizeof(NETWORK_PROTOCOL_ID)+sizeof(NETWORK_PING)+NETWORK_DORA

  Max network frame size. Based on DORA ACK command length.
- #define APP_PROTOCOL_ID 0x10

  Application protocol ID.
- #define APP_CMD_HEADER_SIZE 2

  (App protocol ID + App command) size
- #define APP_GETMANIFEST 0x01

  Get node manifest.
- #define APP_GETPOWER 0x02

  Get current node power.
- #define APP_SETPOWER 0x03

  Set node target power.
- #define APP_GETPOWERSETTING 0x04

  Get node current power setting.
- #define APP_SETPOWERSETTING 0x05

  Set node power setting.
- #define APP_GETPOWERSETTINGS 0x08

  Get node power settings list.
- #define APP_MANIFEST_MINWIDTH 2

  Minimal size of a sManifest type.
- #define APP_MAX_POWER_SETTINGS ((ISM_MAX_DATA_SIZE-APP_CMD_HEADER_SIZE)/sizeof(powerSetting_t))

  Max number of power settings to fit in one frame.
- #define APP_MAX_DESC_LENGTH ((ISM_MAX_DATA_SIZE-APP_CMD_HEADER_SIZE-APP_MANIFEST_MINWIDTH))

  Max description length to fit in one frame.
- #define APP_ERR_PROTOCOL_ID 0x18

  Application error protocol ID.
- #define APP_ERR_CMD_HEADER_SIZE APP_CMD_HEADER_SIZE

  See APP_CMD_HEADER_SIZE.
- #define APP_ERR_NOCOMMAND 0x01

  App command is not supported.
- #define APP_ERR_INVALIDINDATA 0x02

  Invalid received app command data.
- #define APP_ERR_OUTDATATOOBIG 0x03

  Invalid transmitted app command data.
- #define APP_ERR_NODEMEM 0x04

  Node could not access requested data.
- #define DATA_PROTOCOL_ID 0x20

  Data protocol ID.
- #define DATA_HEADER_SIZE (sizeof(datagram_id_t)+2∗sizeof(packet_index_t))

  Data frame header size.
- #define DATA_MAX_PACKET_LENGTH (ISM_MAX_DATA_SIZE-DATA_HEADER_SIZE-1)

  Maximal number of bytes to fit in a data packet.
- #define DATA_PACKET_MAX_NUMBER (0x1 << (sizeof(packet_index_t)∗8))

  Maximal number of packets depending on packet_index_t max value.
- #define DATA_MAX_DATAGRAM_LENGTH (DATA_PACKET_MAX_NUMBER∗DATA_MAX_PACKET_LENGTH-1)

  Maximal datagram length depending on packet capacity and max number.

Typedefs

- typedef float powerkW_t

  Type of raw power measurement.
- typedef float powerTarget_t

  Type of target setting power values.
- typedef uint8_t powerSetting_t

  Type of power setting index.
- typedef unsigned char packet_index_t

  Packet index type.
- typedef unsigned char datagram_id_t

  Datagram ID type.

15.13.1 Detailed Description

Definitions for WPAN protocols.

15.13.2 Macro Definition Documentation

15.13.2.1 APP_ERR_PROTOCOL_ID

```
#define APP_ERR_PROTOCOL_ID 0x18
```

Application error protocol ID.

See Application error protocol for details.

15.13.2.2 APP_GETMANIFEST

```
#define APP_GETMANIFEST 0x01
```

Get node manifest.

Sent data: none

Node reply:

Offset	Length	Data
0	1	priority
1	1	n = description length
2	n	description

Manifest contains a node's priority in power setting adjustments (see sManifest) and its string description. Limited by APP_MAX_DESC_LENGTH

15.13.2.3 APP_GETPOWER

```
#define APP_GETPOWER 0x02
```

Get current node power.

Sent data: none

Node reply:

Offset	Length	Data
0	size of powerkW_t	measured power

15.13.2.4 APP_GETPOWERSETTING

```
#define APP_GETPOWERSETTING 0x04
```

Get node current power setting.

Sent data: none

Node reply:

Offset	Length	Data
0	size of powerSetting_t	current power setting

15.13.2.5 APP_GETPOWERSETTINGS

```
#define APP_GETPOWERSETTINGS 0x08
```

Get node power settings list.

Sent data: none

Node reply:

Offset	Length	Data
0	1	n = number of power settings
1	1	w = length of one power setting (size of powerTarget_t)
2	n∗w	n available power settings of w size

Set n and w by changing powerTarget_t. Use sPowerSettings for easy transmission.

15.13.2.6 APP_PROTOCOL_ID

```
#define APP_PROTOCOL_ID 0x10
```

Application protocol ID.

See Application protocol for details.

15.13.2.7 APP_SETPOWER

```
#define APP_SETPOWER 0x03
```

Set node target power.

Sent data:

Offset	Length	Data
0	size of powerkW_t	target power

Node reply:

Offset	Length	Data
0	size of powerkW_t	measured power

15.13.2.8 APP_SETPOWERSETTING

```
#define APP_SETPOWERSETTING 0x05
```

Set node power setting.

Sent data:

Offset	Length	Data
0	size of powerSetting_t	target power setting

Node reply:

Offset	Length	Data
0	size of powerSetting_t	current power setting

15.13.2.9 NETWORK_ACK

```
#define NETWORK_ACK 0x13
```

NETWORK_ACK.

See Dynamic address management commands.

Command data:

Offset	Length	Data
0	1	Address
1	4	Group
5	1	Lease duration

Lease duration is 0 if NACK (gateway did not grant lease). On reception, node is free to use the leased address and group for the specified lease duration. Node can optionnaly memorize the gateway's address from this frame.

15.13.2.10 NETWORK_DISCONNECT

```
#define NETWORK_DISCONNECT 0x08
```

Disconnect node.

No command data. On reception, Node confirms disconnect with the same command, then resets its address and goes in dynamic addressing mode. This default behavior allows easy reconnection, but can be excluded from later Node remote implementations, so long as the disconnection is confirmed.

15.13.2.11 NETWORK_DISCOVER

```
#define NETWORK_DISCOVER 0x10
```

Discover command.

See Dynamic address management commands.

Command data: none

Node broadcasts this command to get an address offer from gateway.

15.13.2.12 NETWORK_DORA_ERR

```
#define NETWORK_DORA_ERR 0x18
```

Error command.

Command data: none

DORA server can broadcast this frame to signal an error.

15.13.2.13 NETWORK_DORA_GROUP

```
#define NETWORK_DORA_GROUP 0x80000000
```

Default group for DORA command transmission.

Gateway will wake up this group to allow nodes to get dynamic addresses. In cases of channel overload by nodes looking to get an address, gateway can put them to sleep with this group and prioritize traffic from already configured nodes.

15.13.2.14 NETWORK_GET_PROTOCOLS

```
#define NETWORK_GET_PROTOCOLS 0x20
```

Get Node supported protocols.

Sent data: none

Node reply:

Offset	Length	Data
0	1	Protocols

Supported protocols are in a bit field.

wpanManager asks node its supported protocols so it can pass this information on to the user application.

15.13.2.15 NETWORK_GETUID

```
#define NETWORK_GETUID 0x02
```

Get UID.

Sent data: none.

Node reply:

Offset	Length	Data
0	12	Node UID

On reception, Node replies with its UID. UID byte order is little endian.

15.13.2.16 NETWORK_LEASE_UNIT_MINUTES

```
#define NETWORK_LEASE_UNIT_MINUTES 1
```

Lease to minute conversion.

Lease duration in minutes = NETWORK_LEASE_UNIT_MINUTES $*$ lease duration

15.13.2.17 NETWORK_NACK_BASE_ADDR

```
#define NETWORK_NACK_BASE_ADDR 0x04
```

Default address for non-configured nodes.

Nodes can use this address as long as they did not get a lease from the DORA server.

15.13.2.18 NETWORK_OFFER

```
#define NETWORK_OFFER 0x11
```

Offer address.

See Dynamic address management commands.

Command data:

Offset	Length	Data
0	1	Address

Gateway broadcasts this command to offer a node an address. Gateway should choose a free address unless it wants a conflict.

15.13.2.19 NETWORK_PING

```
#define NETWORK_PING 0x01
```

Ping.

No command data. On reception, Node pings back with the same command.

15.13.2.20 NETWORK_PROTOCOL_ID

```
#define NETWORK_PROTOCOL_ID 0x01
```

Network protocol ID.

See Network protocol for details

15.13.2.21 NETWORK_RENEW_LEASE

```
#define NETWORK_RENEW_LEASE 0x14
```

Renew dynamic address lease.

Sent data:

Offset	Length	Data
0	1	Address

Gateway reply:

Offset	Length	Data
0	1	Lease duration

Node asks gateway to renew its address lease for the address specified in command data. Gateway confirms lease renewal with the new lease duration. Lease is reset on reception of new lease duration.

15.13.2.22 NETWORK_REQUEST

```
#define NETWORK_REQUEST 0x12
```

Request address and group.

See Dynamic address management commands.

Command data:

Offset	Length	Data
0	1	Address
1	4	Group

Node broadcasts this command to request previously offered address from gateway. Node also requests a group, typically to match its function. For example, if the node is a power producer, it may want to be in the same group as other power producers to facilitate communications to gateway. Group is ultimately not a decisive factor and is up to the user.

15.13.2.23 NETWORK_SETADDR

```
#define NETWORK_SETADDR 0x04
```

Set node address.

Sent data:

Offset	Length	Data
0	1	Node new address

Node reply: None, command is unconfirmed to account for address change time on Node.

15.13.2.24 NETWORK_SETGROUP

```
#define NETWORK_SETGROUP 0x05
```

Set node group.

Sent data:

Offset	Length	Data
0	4	Node new group

Node reply: None, command is unconfirmed to account for group change time on Node.

15.14 Router/BorderRouter.h File Reference

BorderRouter singleton.

```
#include <stdint.h>
#include <stdio.h>
#include <vector>
#include "netconfig.h"
#include "Connection.h"
#include "wpan.h"
#include "wpanManager.h"
```

Classes

- class BorderRouter

  Forwards external packets to remote nodes. See Border router.

Enumerations

- enum eBorderRouterStatus { **OK** , **NODELIST_OK** , **NOT_INIT** }

  Border router status enumerator.

15.14.1 Detailed Description

BorderRouter singleton.

Created on: Aug 29, 2022 Author: leemarc

15.14.2 Enumeration Type Documentation

15.14.2.1 eBorderRouterStatus

`enum eBorderRouterStatus`

Border router status enumerator.

OK if all Connections have been built. NODELIST_OK if node list was correctly fetched. NOT_INIT it not yet initialized.

15.15 Router/Connection.h File Reference

DataNode to UDP Connection class definition.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "datanode.h"
#include "wpan.h"
#include "netconfig.h"
```

Classes

- class Connection
  Connection class.

Macros

- #define **MAX_FRAME_LENGTH** 1000

Enumerations

- enum eConnectionState { CLOSED , OPEN , SOCKETS_READY , ERROR_SOCKET }
  State of Connection.

15.15.1 Detailed Description

DataNode to UDP Connection class definition.

Created on: Aug 29, 2022 Author: leemarc

15.15.2 Enumeration Type Documentation

15.15.2.1 eConnectionState

`enum eConnectionState`

State of Connection.

Enumerator

CLOSED	Connection is not initialized.
OPEN	Connection is properly initialized.
SOCKETS_READY	IP socket is initialized.
ERROR_SOCKET	IP socket was not properly initialized.

15.16 Router/netconfig.h File Reference

Config file for UDP sockets in Connection.h.

Macros

- #define MAX_CONNS 255
  Max number of concurrent connections for BorderRouter.
- #define BASE_GW_IN_PORT 6000
  Base port for Connection UDP sockets.
- #define FORCE_BIND
  Uncomment to force socket bind in Linux.

15.16.1 Detailed Description

Config file for UDP sockets in Connection.h.

15.17 testMenu/menu.cpp File Reference

Test menu functions.

```
#include "menu.h"
```

Functions

- int serverMenu ()
  Server test menu.
- int nodeMenu (wpanManager &gateway_)
  Node test menu.
- int powerNodeMenu (wpanManager &gateway_)
  PowerNode test menu.
- int dataNodeMenu (wpanManager &gateway_)
  DataNode test menu.
- int wpanManagerMenu (wpanManager &gateway_)
  wpanManager test menu
- int wpanManagerHeadless (wpanManager &gateway_)
  wpanManager headless test routine
- int borderRouterHeadless (wpanManager &gateway_)
  borderRouter headless test routine
- int printMenu (string description, const vector< string > &items)
  show a menu composed of argument items and title
- int mainMenu (wpanManager &gateway_)
  Main menu.

Variables

- const uint32_t timeoutMs = 5000

  default timeout for node commands

15.17.1 Detailed Description

Test menu functions.

15.17.2 Function Documentation

15.17.2.1 borderRouterHeadless()

```
int borderRouterHeadless (
            wpanManager & gateway_ )
```

borderRouter headless test routine

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Starts a borderRouter instance with dynamic addressing. Allows debugging depending on enabled debug messages in wpan.h

15.17.2.2 dataNodeMenu()

```
int dataNodeMenu (
            wpanManager & gateway_ )
```

DataNode test menu.

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Allows testing DataNode class implementation. TX small, variable-size or max size datagrams. Print RX datagrams.

15.17.2.3 mainMenu()

```
int mainMenu (
            wpanManager & gateway_ )
```

Main menu.

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if program should exit

Master menu that links to all sub menus. Takes a reference to the gateway instance and passes it down to the sub menus

15.17.2.4 nodeMenu()

```
int nodeMenu (
            wpanManager & gateway_ )
```

Node test menu.

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Allows testing Node class implementation. Allows waking up, putting to sleep, transmitting data, and network protocol commands testing.

15.17.2.5 powerNodeMenu()

```
int powerNodeMenu (
            wpanManager & gateway_ )
```

PowerNode test menu.

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Allows testing PowerNode class implementation. Allows testing all application protocol commands.

15.17.2.6 printMenu()

```
int printMenu (
            string description,
            const vector< string > & items )
```

show a menu composed of argument items and title

Parameters

description	menu title
items	vector of menu entries

Returns

user choice

15.17.2.7 serverMenu()

```
int serverMenu ( )
```

Server test menu.

Returns

0 if coming back to main menu

Allows testing ISM3 server-level functions. Allows waking up, sleeping, changing beacon data and reconfiguring the gateway.

15.17.2.8 wpanManagerHeadless()

```
int wpanManagerHeadless (
            wpanManager & gateway_ )
```

wpanManager headless test routine

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Starts wpanManager and ticks forever. Dynamic addressing is enabled. Allows debugging depending on enabled debug messages in wpan.h

15.17.2.9 wpanManagerMenu()

```
int wpanManagerMenu (
            wpanManager & gateway_ )
```

wpanManager test menu

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if coming back to main menu

Allows testing wpanManager. Tick, print connected node lists, start/stop dynamic address attribution, update node types, disconnect all clients.

15.18 testMenu/menu.h File Reference

Main menu for test application.

```
#include <iostream>
#include <string>
#include <vector>
#include "cm4_utils.h"
#include "ism3.h"
#include "ism3_handlers.h"
#include "ism3_server.h"
#include "node.h"
#include "powernode.h"
#include "datanode.h"
#include "wpanManager.h"
#include "BorderRouter.h"
```

Functions

- int mainMenu (wpanManager &gateway_)

  Main menu.

15.18.1 Detailed Description

Main menu for test application.

15.18.2 Function Documentation

15.18.2.1 mainMenu()

```
int mainMenu (
            wpanManager & gateway_ )
```

Main menu.

Parameters

*gateway_*	reference to the gateway instance

Returns

0 if program should exit

Master menu that links to all sub menus. Takes a reference to the gateway instance and passes it down to the sub menus

Index