IQRF OS

Operating System

Version 3.00
for TR-52B and TR-53B

User's Guide

Smarter wireless. Simply.

Content

IQRF platform

IQRF is a wireless license free platform for ISM bands (868 and 916 MHz). Compact transceiver module (TR) has built-in operating system (OS) and is fully user programmable in C language using powerful OS functions including RF (wireless) as well as SPI (4-wire serial) communication and complex IQMESH networking support. No link layer is provided, the entire functionality is fully up to user application.

Compatibility

TR module	current OS	modulation
TR–11A	v2.08	ASK
TR–21A v1.02 and v1.03		ASK
TR–31B	v2.10 for 3xB	ASK
TR–32B		ASK
TR–52B	v3.00	**FSK**
TR–53B		**FSK**

Communication is possible among TR modules with the same type of modulation only. OS v3.00 is compatible with 2.11 in STD mode only.

The modules are delivered with IQRF OS allowing realization of common networking device (Node) as well as network Coordinator (software selectable), both able to work additionally also as a router on background (see RF networking).

IQRF transceiver modules allow **upgrades** to current OS version. This service must be done by the manufacturer.

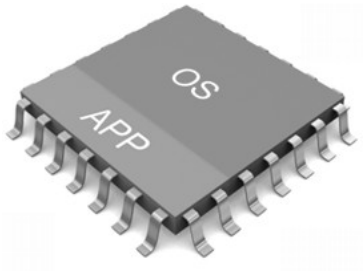
IQRF OS versions and history

Version	Main differences	Release	Status
v3.00 for 5xB	• Up to 65 000 devices and up to 240 hops in IQMESH network • New power management, 35 uA in XLP RX • Discovery, real time transparent routing, low power routers • Many other outstanding features	Jan 2011	**current** for **TR-52B** and **TR-53B**
v2.11 for 5xB	• RF power management supported (`setRFmode`, `checkRF`, …) • RF channels available • selectable RF bit rate (provisionally for experimental purpose)	Mar 2010	not for new designs for **TR-52B** and **TR-53B**
v2.10 for 5xB	In addition to that for 3xB: • 868 MHz or 916 MHz **band** software selectable • Enhanced battery check • RF IC sleep mode supported	Jan 2010	for TR-52B and TR-53B not for new designs
v2.10 for 3xB	• concept of OS **plug-ins** • RF power not limited during bonding • green LED support, LED functions renamed • User **RAM limited** to `0x1CF`	Dec 2009	**current** for **TR-31B** and **TR-32B**
v2.09	• minor change in first falling to Sleep mode • bonding robustness increased	Jul 2009	not for new designs
v2.08	• **broadcast** message support added	Oct 2008	**current** for **TR-11A** and **TR-21A**
	• implemented in **TR-31B** modules	Nov 2008	
v2.07	• bug in the **setLoggingOff()** function fixed • **Wake-up on pin change** improved. To utilize it, the sequence GIE = 0; RBIE = 1; is required just before iqrfSleep().	Sep 2008	not for new designs
v2.06	• minor change in routing	Aug 2008	not for new designs
v2.05	• higher **RF noise immunity** • corrected transfer of MPRWx while not routing • several minor bugs not affecting module functionality corrected	Aug 2008	not for new designs
v2.04	• **setNetworkFilteringOn**() switches just packet from active network (1 or 2), non-networking communication ignored • **Wake-up on pin change** under user's control. Default disabled. To enable, set RBIE = 1 before iqrfSleep(). Not compatible with previous versions (permanently enabled in Sleep up to v2.03).	Jul 2008	internal release only
v2.03	• BufferCOM size increased from 35B to 41B • Number of nodes in one network increased from 128 to 239 • Minor bug in routing fixed	Jul 2008	not for new designs
v2.02	• minor SPI bug fixed	May 2008	not for new designs
v2.01	• function **wipeBondNR**() added • function **batteryValueOK**() added	Mar 2008	not for new designs
v2.00	• Much more **effective**, **easier** to use, higher **performance** • **Networking** totally reworked. Extended capability. **Complete IQMESH**. • **SPI on background** • **Encoded** network communication • Indirect RAM access • Temperature measurement supported by OS • Supports user application debugging directly by IQRF OS • Many other improvements • **IDE** – complete **development environment** with all SW tools integrated including effective debug tools	Jan 2008	not for new designs
v1.14	previous generation	Jul 2007	not for new designs

OS Principles

The IQRF system is designed to allow using of RF wireless connection according to user's needs. Transceiver modules contain microcontrollers for controlling the transceiver operations and for executing of user defined functioning.

Patented IQRF transceiver module architecture has two software layers:



- Basic routines programmed in advance by the manufacturer. The set of such functions is called **operating system** (OS).
- **Application layer** utilizes routines from the basic layer to customize the module for user specific operation.

In opposite to Solution stack, there is no need to compile protocol related routines, just the application is compiled. This approach reduces time and development costs significantly when creating connectivity applications.

OS offers software functions prepared in advance for all common user requirements. Thus, it is not necessary to create the whole user program by oneself (using microcontroller instructions and C commands only) but the user adds a user part of software to the OS only.

The user application so called „runs under the operating system" which means that this is invoked from OS, uses OS functions and is (should be) under OS control.

OS functions need not run sequentially (next function invoked not until the preceding one is finished) but some operations can run so called "in background" (the function arranges execution of requested operations which runs independently and immediately returns the control back to superior program). In this way more processes can run "simultaneously". Then the program structure is that besides of execution running sequentially "in foreground" several tasks in background can be running. IQRF OS allows to run even very complex operation including complete SPI communication protocol in background. This makes real-time programming really easy.

IQRF OS supports communications:

- **RF** (radio), including networks – in **peer-to-peer** and **IQMESH** topologies.
- Standard serial **SPI** (slave mode) interface for connection to peripherals or to PC (e.g. via CK-USB-04).

Other communications can be realized with a user program (I^2C, UART, ...).

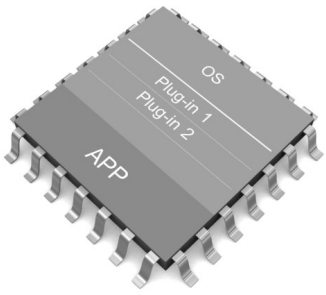
Complex standard communication interfaces (**USB**, **Ethernet**, **GSM**, …) can be realized using IQRF gateways.

OS supports low power consumption of IQRF transceivers with the **Sleep** functions when operation of TR module or RF IC is reduced/stopped.

To increase the reliability the **watchdog** function is used. This is implemented in microcontroller hardware and controlled via user program.

Concept of OS plug-ins

IQRF operating system can be extended via optional plug-ins.



Plug-in is a SW module delivered (typically by the IQRF manufacturer) as a file with the `.IQRF` extension. It should be uploaded to the TR module by the IQRF IDE and an IQRF programmer (e.g. CK-USB-04). The procedure is similar to uploading a user program.

More plug-ins can be used at the same time.

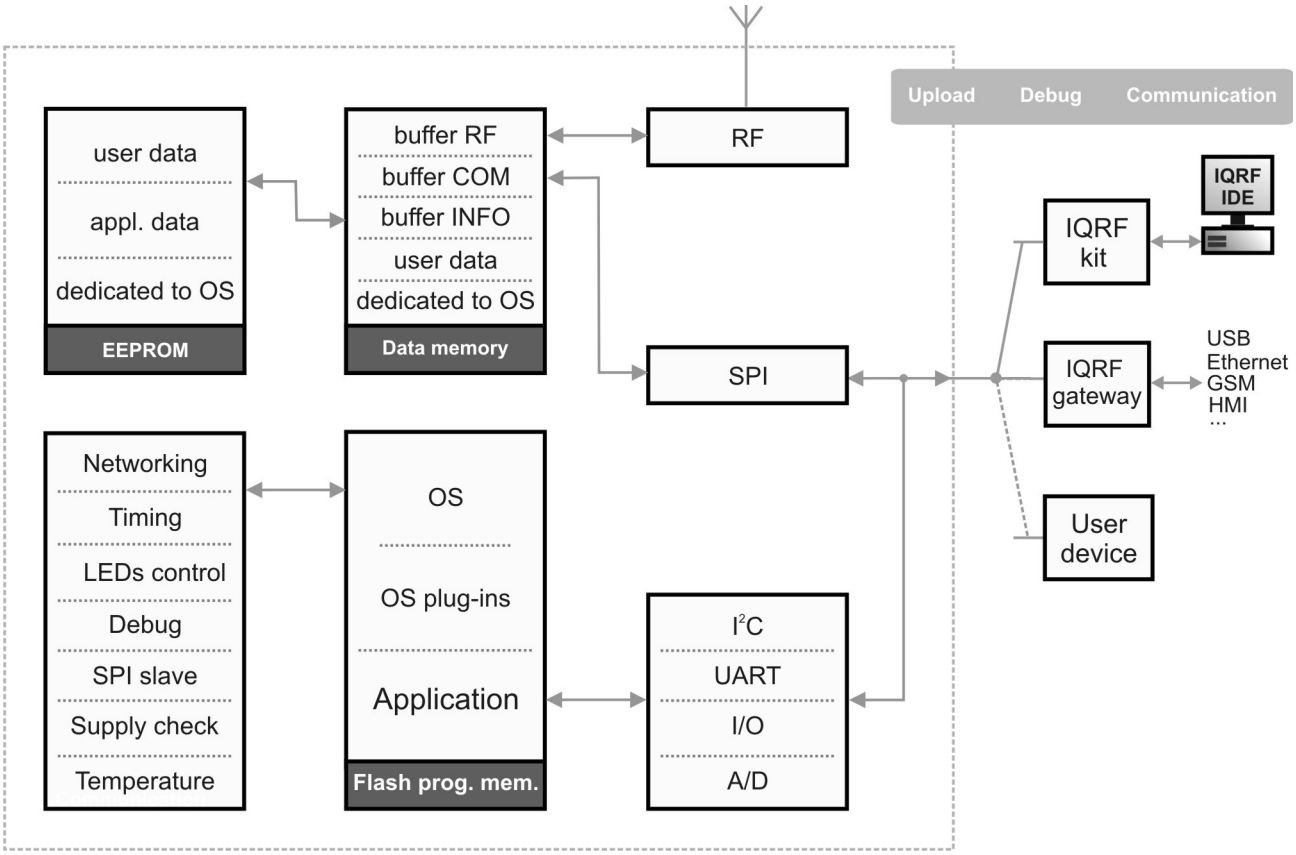
To utilize a plug-in, corresponding header files (with the .H extension, also delivered with the plug-in) should be included to source program similarly to other system header files.

Example: `#include "plug-ins/PlugInXY.h"`

Then all plugged-in functions are available like standard system ones.

IQRF OS Architecture

Hardware of the transceiver module with a microcontroller including the IQRF OS results in architectural model:



Individual blocks:

- Memories:
  - program memory (Flash)
  - data memory (RAM)
  - data memory (EEPROM)
- Communication interface:
  - RF (wireless)
  - SPI (standard serial, 4-wire)
- Temperature sensor (TR-52B only)
- Power supply check
- Digital I/O (input/output).
- A/D converter
- Time base support: calibrated 10 ms interval (tick) generator in background and supporting functions
- 2 LEDs control in OS background
- IQMESH networking
- Debug: OS support for testing and debugging

Resources partially depend on transceiver module type.

RF circuitry

Main features:

- **Bands:** 868 MHz / 916 MHz, SW selectable (default 868 MHz). Default 916 MHz on request.
- **Channels:** Frequency channels SW selectable in accordance with the CEPT ERC/REC 70-03 General License. See Appendix 2.
- **Bit rate:** SW selectable (1.2 kb/s, 19.2 kb/s, 57.6 kb/s and 86.2 kb/s). Default is 19.2 kb/s, other bit rates are provisionally intended for experimental purposes only. Routing is tested for 19.2 kb/s only.
- RF output power: up to 3.5 mW, SW selectable in 8 steps.
- Various **sleep** modes to reduce overall current consumption and optimize response times.
- RF RX and TX **power management** modes.

Microcontroller

IQRF OS for TR-52B, TR-53B and compatibles is implemented in the **PIC16F886** MCUs (8-bit microcontrollers by Microchip) – datasheet see [8].

PIC hardware resources and their utilization in TR modules with OS:

PIC HW resources		Utilization
Program memory	Flash	1024 instructions
Data memory	RAM	40 B – user data 172 B – communication/system buffers
Data memory	EEPROM	Node: 160 B user data + 32 B application data
		Coordinator: 0 B user data + 32 B application data
I/O pins	TR-52B	6 × I/O
	TR-53B	7 × I/O
A/D converter (10b)		2 (TR-52B) or 3 (TR-53B) external analog inputs
Serial communication	SPI (slave)	Supported by OS in background
	I^2C	Realized by PIC HW module and user function – see Application examples [10]
	UART	Realized by PIC HW module and user function – see Application examples [10]
Interrupt		Not user available. It can be disabled just for a short period if necessary.
Stack (for subroutines, shared with interrupt)		Max. 2 levels of subroutine calling is allowed except of `RFTXpacket()` and `RFRXpacket()` (1 level allowed) and `bondNewnode()`, `bondRequest()` and `answerSystemPacket()` (must not be called in subroutines at all).
Power-on reset		Utilizes HW filter to eliminate improper power-up rising and spikes to some extent.
Brown-out reset		Enabled during operation and disabled in Sleep. Reset voltage level = 2.1 V. Typical power-on reset and OS boot takes ~720 ms and depends on supply rise shape.
Power-up timer		Disabled
Watchdog		Default disabled in Configuration word and enabled by SW (can be disabled by SWDTEN bit). Time-out can be set from 1 ms to 268 s or disabled at all by SW, default ~ 4 s. WDT time-out varies with temperature, supply voltage and other conditions from part to part. See PIC datasheet [8].
Oscillator		Internal RC, 8 MHz (500 ns instruction). Do not switch to another clock. IQMESH timing precision is not limited by precision of this oscillator when calibrated by the `calibrateTimer()` OS function.
Configuration words ("Fuses")		`CONFIG1 = 0x2214, CONFIG2 = 0xFCFF`

These resources can be under OS supervision and the user should access them in accordance with this manual and possible requirements resulting from hardware construction of the module and OS implementation.

Configuration changes and direct access to some resources by the user can be limited or not allowed at all. Serviceability of some resources depends on using of some other ones at the same time (some hardware communication modules, pins and memory areas are shared for more functions, …).

Parts of memories are dedicated to PIC core, peripherals and operating system. Direct access (via the EEDATA register) to the EEPROM is not allowed at all, extra OS functions are intended for this. Flash memory is user accessible for uploading the program and OS plug-ins to the microcontroller using the IQRF development kits only. Indirect RAM access using the FSR register is not allowed due to security reasons. Instead of this IQRF OS provides complete support for indirect addressing using extra system functions. Not dedicated user inputs/outputs, peripherals (e.g. I^2C and UART) and RAM locations can be accessed directly according to user's needs.

Details see datasheets of the transceiver modules [7], PIC datasheets [8] and Appendix 1 – RAM and EEPROM maps. In doubt, refer to IQRF support by the manufacturer [6].

Memories

For memory purposes the IQRF OS uses internal memories of the microcontroller only. (TR-5xB also uses external serial EEPROM dedicated to OS, not user accessible.)

Individual parts of memories are:
- Dedicated to the microcontroller
- Dedicated to the OS
- Other areas are available for the user

Memories can be under OS supervision and the user should access them in accordance with this manual and possible requirements resulting from hardware construction of the module and OS implementation.

Illegal modification of dedicated memory locations can cause system crash.

There are several header files (with the .H extension) delivered with IQRF examples and tutorials. They are intended for C compiler to provide easy and seamless linking the OS with the user program. Of course, these text files could serve to user's survey concerning memories – but the user should nowise modify them. (The `16F886.h` is based on standard file made by the C compiler manufacturer that is why they contain some IQRF irrelevant information to spare.)

User's own definitions should be placed to extra user header files. Names of user variables must not collide with names predefined in delivered header files.

Refer to Appendix 1 - RAM and EEPROM maps.

Program memory (Flash)

The user can use this as a program memory only. The program remains stored there even after power off. Overwriting is not unlimited, number of erase/write cycles is about 100 000 typically.

User program can be uploaded into the TR module using appropriate IQRF development kits, e.g. CK-USB-04 and IQRF IDE servicing program [9]. Codes in standard .HEX format or encrypted codes in the .IQRF format can be uploaded.

- OS and plug-ins occupy the memory from `0x0000` to `0x1BFF`
- Remaining area `0x1C00` – `0x1FFF` (1024 machine instructions) is available for user program .

User program should begin from address `0x1C00`. It is automatically arranged by the IQRF header files.

Data memory (RAM)

RAM data is fully under supervision of running program and is lost after power off.

Individual RAM parts:

- dedicated to the **microcontroller** and its **peripherals** (PIC special function registers – SFRs). Direct using is mostly restricted, e.g. the application need not use registers `INDF`, `EECON1`, `EECON2`, `EEADR`, `PIR2`, `PIE1` and `PIE2`. In doubt, refer to IQRF support by the manufacturer [6].
- dedicated to **OS**:
  - IQRF **communication** (RF and SPI) is packet oriented therefore buffer servicing is supported. There are four basic buffers primarily dedicated to communication and block operation:
    - **`bufferRF`** (`0x110` – `0x14F`), 64 B – for RF communication
    - **`bufferCOM`** (`0xA0` – `0xC8`), 41 B – for serial communication (especially SPI). Use 35 B (`0xA0` – `0xC2`) only, the others are reserved for OS and future compatibility.
    - **`bufferINFO`** (`0x20` – `0x42`), 35 B – for OS and user block operations

    These communication buffers are especially intended for transferred data but can be used according user's need in specific cases as well. There are specialized OS functions for comfortable buffer to buffer data copying.
    - **`bufferAUX`** (`0x1C0 – 0x1DF`), 32 B – auxiliary buffer, to store bufferINFO temporarily to make it free for other operations.
  - **buffer `networkInfo`** (22 B) is an area dedicated to network system information.
  - system variables. Some of them (`toutRF`, `userStatus` etc.) can be directly accessed by the user.
  - OS work variables (not documented, the user need not modify them).
- Area (`0x190` – `0x1B7`) is **available for the user** (40 B). It is located in the RAM bank 3. Selection of bank 3 is automatically arranged by the IQRF header files. Additionally, two `userRegx` registers (`0x1F0` and `0x1F1`) are available in area shared for all banks. If needed, refer to the PIC datasheets [8] for information about RAM banking.

See Appendix (RAM map).

For block access special OS functions are intended instead of access via FSR and INDF registers which is restricted due to security reasons. Indexes of arrays are not allowed to be variables. (A[1]=0 is allowed, A[i]=0 is restricted due to using FSR by the C compiler). See IQRF OS Reference Guide [1].

Data memory (EEPROM)

EEPROM data remains stored even after power off. Overwriting is not unlimited, number of erase/write cycles is 100 000 min., (typically 1 000 000). EEPROM is especially intended for configuration parameters and data.

Individual EEPROM parts:

- **User data:** 160B from `0x00` to `0x9F` (Nodes only). This area is not user available for Coordinators.
- **Application data:** 32 B from `0xA0` to `0xBF` (Nodes as well as Coordinators). The user can use this area for his particular needs (especially intended for configuration and similar purposes). It is accessible for reading via the `appINFO()` function in a comfortable way. The factory settings string is: `"Hello everybody. IQRF is here! "`
- **Dedicated to OS:** (Node as well as Coordinator)
  Remaining EEPROM area (`0xC0` – `0xFF`) is dedicated to OS. It is not accessible by the user.

EEPROM access:

- Values can be specified in application (source) program to be written to EEPROM while the program is uploaded into the microcontroller. EEPROM address range is `0x2100-0x21FF` instead of `0x00-0xFF` when using `cdata` and similar C statements (e.g. `__EEAPPINFO = 0x21A0`).
- The microcontroller can read/write data from/to EEPROM under user program control while the application is running using general OS functions for accessing EEPROM (eeWriteByte, ...). Short addresses (`0x00`–`0xFF`) are used in this case. Access via `EEADR` and `EEADTA` registers is restricted due to security. See IQRF OS Reference Guide [1].

The user should avoid exceeding the number of erase/write cycles allowed. Note that also some other OS functions (bond, bondRequest, ...) write to EEPROM as well.

Identification

Module data

Every IQRF module contains information about itself. This is accessible via the `moduleInfo()` function storing data to the `bufferINFO` in the following format:

address in `bufferInfo`	7	6	5	4	3	2	1	0
meaning	OS build		PIC type	OS version	Coordinator / Node	serial number		
					Module ID			

Coordinator / Node: reserved for future OS versions (set to 1 in this OS version). Coordinator / Node is SW selectable in IQRF this OS version.

0: Node
1: Coordinator

OS version:

upper nibble (4 b): major version
lower nibble (4 b): minor version

PIC type:

3: PIC16F886

OS build: for the manufacturer only. Differences among various builds has no influence to functionality from the user's point of view.

Example *(*all in hexadecimal*):*

```
                  [0] [1] [2] [3] [4] [5] [6] [7]
bufferINFO[0-7] =  1C  10  00  01  30  03  39  11
```

Meaning: Coordinator, Module ID = `0100101C`, IQRF OS version 3.00, PIC16F886, build # `0x1139`.

Module ID is displayed by the IQRF IDE development environment.

Application data

It is a 32 B block in EEPROM (area `0xA0` – `0xBF`) dedicated to the user application. It is possible to read data from it directly to the `bufferINFO` very effectively by a single instruction (`appINFO()`) only. This area is intended for arbitrary information concerning user application but is especially useful for repeatedly employed (often permanent) data such an identification information to be compared after receiving (with the `compareBufferINFO2RF()` function).

Refer to memory maps in Appendix as well.

Control

Operation modes

The TR modules can work in three modes:

- **Programming:** The user program or OS plug-in can be uploaded to the TR (including EEPROM content). This mode is available using the appropriate IQRF development kit and IQRF IDE development environment. See application note AN003 [11].
- **Run:** The TR module executes operation programmed by the user.
- **Debug:** Execution is stopped and data can be downloaded from the microcontroller and displayed by the IQRF IDE. This mode is fully under control of user program and interactive handling in IQRF IDE.

Real time

OS provides an efficient support for real time applications. It has a generator of time intervals running on background and appropriate functions. Basic interval (elementary OS time interval for timing on background – a "tick") is 10 ms. Specified number of ticks serve for timing of appropriate processes (delays, LED blinking, communication timeout checks, …) and also enables to create a user timebase. Tick is derived from internal RC oscillator.

- Capture is another efficient timing tool. It is an independent resettable timer (16-bit counter of ticks) freely running on background. It is suitable especially for working with long periods (up to 655 s).
- OS provides functions also for waiting on foreground.
- Short time intervals for timing on foreground can be derived also from instruction timing. The PIC16F886 is clocked with internal 8MHz RC oscillator. Thus, instruction cycle is 500 ns (1 µs for some instructions) – see PIC datasheets [8].

Some OS functions (especially `RFRXpacket` and several delays) share the same internal timers that is why these functions should not be used at the same time. Refer to the IQRF OS Reference guide [1], side effects.

Note that time precision of TR modules depends on precision of internal RC oscillator – see PIC datasheets [8]. Microcontrollers are individually calibrated by the manufacturer but despite of this fact the precision and stability are less than for a crystal oscillators. The precision is sufficient for asynchronous communication (UART) with reasonable speed, for clock and calendar functions another suitable method should be used. To increase precision for operations based on ticks, tick duration is calibrated to crystal precision automatically after reset and should be also done by the user (from time to time, in case of temperature or supply voltage change etc.) This is useful especially for IQMESH timing. See the `calibrateTimer()` function in the IQRF OS Reference guide [1].

Watchdog

To increase the reliability, the OS uses hardware watchdog of the microcontroller. It is a continuously running independent timer with a programmable overflow period. It should be used that never overflow during correct operation. It is accomplished via the `clrwdt()` instruction always executed in time, i.e. before the watchdog overflows. (This function is implemented not in OS but it is the PIC machine instruction supported with the C compiler). If an overflow occurs it is regarded as a program execution failure (especially due to an error in algorithm in application program) and the microcontroller responds with reset. If the failure is not a permanent one, it can lead to system recovering. The watchdog can run even in the Sleep mode (see below). Overflow in Sleep results in wake-up and continuing execution but not in the PIC reset.

The watchdog can be enabled/disabled in SW. Overflow period is user selectable (even while the program is running) from 1 ms to 268 ms. Setup registers are WDTCON (WDTPSx and SWDTEN bits) and OPTION (PS0, PS1 and PS2 bits, remaining bits including PSA must be left unchanged by the user) – see PIC datasheets [8]. Default timeout period is about 4 s. Watchdog can be disabled by `SWDTEN=0` and reenabled by `SWDTEN=1`.

TR module Sleep

Complete TR module (including the RF circuitry, microcontroller and temperature sensor) can be set in the standby (Sleep) mode. In this case almost no operation is executed but the power consumption is minimized.

Transition to the Sleep mode:

The Sleep mode is initiated in software using the `iqrfSleep()` function in appropriate location in the source program. Then all TR hardware resources controlled by the OS are automatically suspended: activity of the TR module including the RF circuitry, temperature sensor, microcontroller as well as its peripherals (stopping of timers, disconnecting of internal pull-ups, ...).

Before switching to the Sleep mode:

- Power consumption should be minimized even for hardware resources of TR controlled by the user (PIC pins, possible PIC internal peripherals) and possible external peripherals connected. It must be done in user program. See the TR [7]

and PIC [8] datasheets.

- The microcontroller should be configured for subsequent wake-up on pin change (if required):
  - Wake-up on pin change is under user's control, default disabled.
  - To enable, the sequence `GIE = 0; RBIE = 1; iqrfSleep(); RBIF = 0;` is required.

Returning to the operating mode (wake-up)**:**

- after watchdog overflow (if enabled)
- after pin change on some pins (depending on the TR type, typically the C5 pin), when configured as inputs (if enabled)
- after power-off/on

Tip: wake-up types can be identified via the `-TO` and `-PD` status flags – see Reset below.

After the wake-up the microcontroller continues with execution the command following the Sleep function.

The user can use Sleep and wake-up without any restriction due to OS, all related microcontroller possibilities can be employed – see PIC datasheets [8].

Tip: sleep period can be setup via the watchdog timeout period.

Typical sleep power consumption ~2 μA can be reached with all peripherals off – see the TR datasheets [7].

RF Sleep

RF circuitry can be set in power saving standby mode while the microcontroller continues running using the `setRFsleep()` function. 0.6 mA typ. is saved. RF response is prolonged for 2 ms typ., 7 ms max. due to wake-up. Wake-up can be caused by `RFRXpacket()`, `RFTXpacket()`, `checkRF(x)` or `getSupplyVoltage()`.

RX chain

After RF transmission is finished RF chains are automatically disabled. In the `Stay in RX` mode the RX chain remains enabled for faster response to following `checkRF`, `RFRXpacket` or `RFTXpacket`.

Other PIC peripherals

There are PIC HW resources (I^2C, UART, …) not supported with the OS but accessible directly via PIC special function registers. Refer to datasheet of the microcontroller [8].

Reset

The following reset types available:

- **Power-on reset**: Utilizes HW filter to eliminate improper power-up rising and spikes to some extent. A lot of applications need no external reset circuitry.
- **Watchdog reset:** After WDT time-out.
- **Brown-out reset:** If power supply falls below given level for given time the reset is invoked. This option is disabled but can be enabled by the manufacturer on request. Then it is possible to have it enabled during operation and disabled in Sleep.

Reset can also be invoked by SW via the `reset()` function. It is the watchdog reset type.

To identify reset type four status flags are available in the userReg0 just after boot:

bit	7	6	5	**4**	**3**	2	**1**	**0**
Status flag				-TO	-PD		-POR	-BOR

`-TO` Watchdog time-out flag
- `-TO` = 0 after `reset` execution, WDT overflow or `reset()` function.
- `-TO` = 1 after power-up, `clrwdt` or `iqrfSleep`

`-PD` Power-down flag
- `-PD` = 0 after `iqrfSleep`
- `-PD` = 1 after power-up or `clrwdt`

`-POR` Power-on reset flag
- `-POR` = 0 after power-on reset (must be set in software then)
- `-POR` = 1 no power-on reset occurred

`-BOR` Brown-out reset flag
- `-BOR` = 0 after Brown-out reset (must be set in software then)
- `-BOR` = 1 no Brown-out reset occurred

Refer to the PIC datasheet [8], (`STATUS` and `PCON` registers) for details.

Content of RAM registers after resets is strictly defined (set / cleared / not affected / unknown). It partly depends on reset type. Refer to datasheet of the microcontroller [8]. Additionally, OS clears the user memory area except of the `userStatus` register which remains unchanged after `reset()`.

Tip: `userStatus` can be used to help to debug unexpected resets in user programs:
- Fill this register with different values at suspicious locations to identify where unexpected reset occurs.
- Use this register as a counter or timer to reveal the cause of the reset.

Temperature measurement

Temperature is measured by the on-board sensor using internal A/D converter of the microcontroller (TR-52B only). See `E08-TEMPERATURE` example [10] and IQRF OS Reference guide [1].

Battery check

See `getsupplyVoltage()` in IQRF OS Reference guide [1].

LED indication

Two on-board LEDs (red and green) can be served by the set of specialized functions running on OS background.

Debug

The IQRF platform provides user with an efficient debugging tool. To enjoy its powerful capabilities, the following configuration should be used: The transceiver module plugged e.g. into the CK-USB-04 development kit connected to PC via USB with the IQRF IDE development environment [9].

Debug is directly supported by the OS with the `debug()` function. This can be included in user program wherever you need to stop program executing and evaluate variables, EEPROM content or RAM registers. After uploading user program into the transceiver module the application is running until the `debug()` function is encountered. Then the program stops, the module is switched to the debug mode and data is downloaded and displayed on the screen.

The module stays in debug mode till the user wishes. Then the application program can continue execution until another `debug()` function is encountered and so on. To identify individual debug breakpoints the W register can be used. See IQRF IDE Help and `E06-RAM` example [10] for details.

SPI

Standard serial 4-wire bus running in OS background. See separate User's guide, Implementation in IQRF TR modules [5].

RF

RF overview

OS functions allow powerful and user-friendly control of RF communication. From the user's point of view it means working primarily with memory and buffers (R/W operations with RF communication buffer). IQRF OS automatically provides all needed services including full protocol impementation:

- at transmission level: HW setup, coding for transmission, timeouts, ...
- at packet level: preamble, consistency checking, coding, ...
- at network level: routing, including information about the network and device, filtering, discovery, ...

Supported modes:

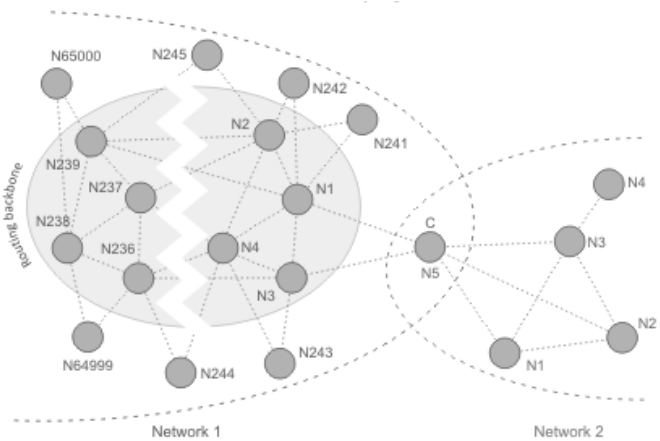
- **Peer-to-peer:** Two or more peer-to-peer devices, without a network Coordinator. Packets are available for all devices in range and completely managed by the user program. Number of devices is unlimited. Keep `PIN=0` (see below) in this mode. This is the default mode.
- **IQMESH:** Topology with one Coordinator mastering the network and up to 65 000 end devices (Nodes) and up to 239 devices in routing structure (backbone) with full network support. This mode is defined by setting the most significant bit (`_NTWF`) of the PIN register to 1. Nodes must be assigned (bonded) to the Coordinator's network. Peer-to-peer ("non-networking") packets are also allowed in IQMESH.

Memory locations and registers related either to Peer-to-peer or IQMESH:

`bufferRF[64]`	Buffer for RF routines (data to be sent by `RFTXpacket` or received by `RFRXpacket`), 64B
`PIN`	Packet information. See below.
`RX`	Packet addresse (specify before transmitting)
`TX`	Original packet sender (set by OS during receiving)
`DLEN`	Packet length (number of relevant bytes in `bufferRF`), 0-64 (specify before transmitting, set by OS after receiving)
`toutRF`	Timeout for packet receiving (1-255) in number of 10 ms ticks or for LP and XLP modes in cycles (RFIC On-Off periods). Default value is 50 (500 ms in STD mode).

IQMESH network and its individual devices can be configured very flexibly. IQMESH as well as peer-to-peer packets can be sent and received depending on setup of respective devices. Nodes can be assigned to groups. Individual and broadcast packets (for all network members) are supported and user addressing is allowed. IQMESH protocol has been defined as a light and portable to inexpensive microcontrollers with limited resources. One or two byte addressing is chosen.

Every IQRF device can simultaneously work in two independent networks. This OS version supports two networks for every device, working as a Coordinator in one network and as a Node in second network. It allows chaining networks up to unlimited number of devices and easy data sharing. Non-networking packets and packets coming from the other network can be filtered. Background routing is fully suppported. Each Node can provide background routing service for network packets or can be programed as a dedicated router. Both Coordinator and Node can be realized by a more complex device, a Gateway, providing an interface between IQMESH and other standards.



Although IQMESH is very flexible and supports high variability and dynamic changes in configuration (including changes in topology), it is primarily intended for more or less static systems. Devices are included in / excluded from the network by the bonding / unbonding procedure which should be considered to be an installation process by its nature. The Coordinator is not intended to be switched dynamically from device to device in a network. The Coordinator should manage RF communication in the whole network. Nodes are allowed to communicate anytime but it can be recommended just in special cases. In typical applications the Coordinator always initiates any communication. All IQMESH communication is coded. The coding differs from network to network beeing readable in given network only. In addition to "user" packets, IQMESH uses also system packets with auxiliary information (e.g. for bonding, routing etc.). Such system packets are completely transparent from the user's point of view.

Basic network information about current setup of given device (network identification, device number, current network, topology, ...) provides the `getNetworkParams()` function.

RF networking

IQMESH packet transmission is supported by a lot of additional sophisticated features. The communication is possible even between nodes out of RF range each other – using "hops" via other nodes in range (routing). In addition to the normal operation, every IQMESH device (TR module, gateway, ...) can work also as a router on background. IQMESH can additionally contain specialized plug-and-play routers.

Packets for Peer-to-peer communication consists of three block - PAH (packet header), DATA and CRC, while IQMESH packets consists of four blocks - PAH, NTWINFO (networking information), DATA and CRC. Every block has its own consistency check mechanism (CRCs) to achieve high reliability.

PIN	DLEN	CRCH	Networking info	CRCN	User data	CRCD	CRCS
PAH			NTWINFO		DATA		CRC

PAH

Packet header, 3 bytes long block, carries basic information about a packet, such as data length and flags (whether the packet is intended for peer-to-peer or IQMESH, indication of system communication, routing, direct peripheral addressing, encryption and acknowledgment request).

PIN

bit 7	6	5	4	3	2	1	0
_NTWF	_ACKF	_ROUTEF	_CRYPTF	_MPRWF	_SYSPF	_TIMEF	_AUXF

_NTWF:
- NTWF = 0 Peer-to-peer mode
- NTWF = 1 IQMESH mode

_ACKF: Acknowledge request (specify request to the packet before TX, accomplish request after RX). Handling with this flag and all acknowledge processing is fully up to the user (OS just transfers this flag without any consequences).

_ROUTEF (for IQMESH mode only):
- ROUTEF = 0 Routing not required for outgoing packets.
- ROUTEF = 1 Routing required for outgoing packets, routing registers RTDT0-3 must be defined.

_CRYPTF: Crypting requested. Reserved for future use.

_MPRWF (intended only for special applications supporting direct access to peripherals and services):
- MPRWF = 0 Module peripheral read/write not active
- MPRWF = 1 Module peripheral read/write active. MPRW0-2 is added to NTWINFO by OS.

_SYSPF: Dedicated to OS, not intended for users.

_TIMEF: Dedicated to OS, not intended for users.

_AUXF: Reserved for future use.

NTWINFO (applies for IQMESH mode only)

Networking information block with variable length based on PIN flags. Just five bytes (RX to PID) are mandatory (present in every IQMESH packet), the others depend on actual situation. For example, Star topology does not need routing information. Setting ROUTEF = 0 will make a packet without routing, while after setting ROUTEF = 1 six bytes describing the routing are expected to be added to the NTWINFO. This mechanism provides a way to fit various application needs. NTWINFO registers are set by the sender and are passed to all recipients via the packet. Thus, the recipient know which hop is actually in question and how long it is to wait before possible forwarding.

RX	TX	...	PID	RT0TX	RTDEF	RTDT0-3	MPRW0-2

RX Address of the device the packet is intended to in current network:
- 0 Coordinator
- 1 - 239 Nodes
- 240 - 253 Reserved
- 254 Universal address – see bondNewNode()
- 255 Broadcast

This must be specified by the user before sending an IQMESH packet.

TX Address of transmitting sender. It is automatically set by OS by RFTXpacket().
PID Packet identification (can be set by the user, e.g. not to respond twice to the same packet). See example E11-IQMESH-N [10].
RTOTX For OS only.
RTDEF Routing algorithm.
RTDT0-3 Routing data. See below.
MPRW0-2 Reserved for Direct peripheral access.

User data

Data from/to the bufferRF. The length can vary between 0 and 64 B.

RF transmitting

It is possible to combine sending peer-to-peer and IQMESH packets (depending on the NTWF flag).

- Peer-to-peer: Prepare data to the bufferRF, specify data length (DLEN = ...) and simply send the packet via the RFTXpacket(). All receivers obtain the data and DLEN only.
- IQMESH: To send IQMESH packets, an appropriate setup (Coordinator/Node selection etc.) should be done and the Node should be bonded to a network. Sending itself is similar to Peer-to-peer but the receiver address (and possible routing information) must be specified. Other networking information is added to the packet by OS automatically.

If bidirectional communication, PIN and DLEN should be updated before every transmitting followed after any reception.

Packets can be sent in several modes in dependency on receiver mode.

See the IQRF OS Reference guide [1] (RFTXpacket()) and examples E01-TX, E03-TR and E09-LINK [10].

RF receiving

The RFRXpacket() function attempts to receive a packet and returns control to application after successful reception or after the timeout. Timeout during packet receiving terminates the reception except of the Wait packet end option is enabled.. The user has full control on timing as the timeout can be set in ticks (~10ms in STD RX mode or in cycles ~40 ms in LP RX or ~600 ms in XLP RX) prior to the RFRXpacket() function call (toutRF = ...).

Result (the RFRXpacket() return value) depends on the conditions (filtering, current network, RX mode, packet type, addresse etc.).

To achieve desired noise immunity, programmable signal strength filtering is applied in LP, XLP and SSF RX modes (see below). Incoming signal with lower level than one of four predefined values is ignored. Relative RF range is shortened due to this filtration.

After successful reception respective values are valid: received data in bufferRF, data length (DLEN) and possibly all other networking information.

To achive ultra low power consumption, the following RX modes are available:

- STD: standard
- LP (low power): receiving combined with standby mode. Incomming packets should be detected by RFRXpacket() (utilizing signal strenght filtering) but not by checkRF(x).
- XLP (extra low power): receiving combined with deep standby mode. Incomming packets should be detected by RFRXpacket() (utilizing signal strenght filtering) but not by checkRF(x).
- RFIM (RF Immunity Mode): RFRXpacket() is prematurely terminated if RF signal falls below predefined level specified in the sefRFmode(x) function, bits FF. This should be used with checkRF() only: if checkRF() …

See the IQRF OS Reference guide [1] , RFRXpacket() and examples E02-RX, E03-TR and E09-LINK [10].

Filtering

In case of chaining networks it can be selected whether packets should be received from both networks (including peer-to-peer packets) or from the current network only. If filtering is off current network is automatically switched to the network the packet was received from.

Addressing

There are 3 types of addresses – see below (Routing, Addressing overview):

- Logical address: created by bonding.
- User address: created by setUserAddress(x).
- Virtual routing number, VRN: created by Discovery. For OS only. The user need not take care about VRNs at all.

Routing

Routing allows sending packets to addressees out of the sender's range using "hops" via devices which are in range each other. This IQRF OS supports up to 239 routing devices for a packet. Routing is separated from addressing and is transparent from th user's point of view. There are several routing algorithms specified in the `RTDEF` register according to network topology. Packet is routed via devices in specified order (routing vector) in defined time slots with specified period each. Retransmitting passes in reverse order. OS ensures that the packet is ignored by all devices except of the addresse and the devices specified in the routing vector.

For effective IQMESH the topology (placement of devices with respect to the range) should be designed in a redundant way - every device should have sufficient number of devices in range. Routing algorithm should be specified with respect to reliability and speed requirements. Due to time slots the efficiency should considerably depend on the order in the routing vector as well. The Addressee does not route the packet except of a broadcast one.

Thus, routing allows higher range, lower RF output power, more ways to deliver packets, higher noise immunity, resistance against failures and dropouts (self-healing) and flexibility with respect to dynamic changes in range among individual devices (moving of persons, obstacles or devices themselves) which results in better throughput and reliability.

Routing can be enabled or disabled for individual nodes. Routed packets can be received whenever the `RFRXpacket()` is active in routing device but they can be retransmitted in respective time slots only.

Discovery

Nodes can be placed according to their addresses (with respect to fixed routing vector 1, 2, 3, …) or in a random order. But the random order requires Discovery when internal routing backbone is created and routing paths are found automatically.

Discovery assorts Nodes to zones (groups of Nodes which can be reached by the same number of hops from the Coordinator). Number of zones can be limited by the user.

During Discovery the answeSystemPacket() function must run in a loop in every Node to be discovered. It is recommended to run Discovery with stronger filter then it is planned for common communication (lower RF power on the Coordinator side or stronger RSSI filter on the Node side – see example `E11-IQMESH-N` [10]). Nodes answer with the same output power (possible restoration is up to the user). Number of discovered nodes can be less than number of bonded nodes. Discovery results also depends on the `setRoutingOff()` function in Nodes.

After changes influencing the range (changes in topology, addressing, device placement, obstacles, permanent failure of a router etc.) the Discovery should be reinvoked.

Routing is possible under all following conditions:

- The routing device is bonded to respective network
- The packet was sent by the original sender with routing requirement (`ROUTEF = 1`)
- The `RFRXpacket()` function is active in the routing device when the packet to be routed is sent.
- The ROUTEF flag relates to outgoing packets and has no influence to routing incoming packets at all.

Dedicated router for STD packets doing nothing but background routing can be realized very simply by a neverending loop:

```
setNodeMode();
while (1)
{
  RFRXpacket();
  clrwdt();
}
```

It assumed this device has already been bonded. Due to power consumption it is recommended to supply such a router from mains adapter. Battery operated routers should use the LP or XLP receive modes.

Every application has usually very different requirements. For example, a typical Smart House application can be realized with 4 hops and there is a need for fast response, while collecting data from power meters usually needs a network supporting much more hops but the latency is allowed. Thus, IQMESH specification supports various routing algorithms.

IQMESH routing rules

- Every node routes a packet only once.
- Every node routes in the time slot corresponding to the address of the node, either logical (for SFM routing) or VRN (for DFM routing). See chapters Routing algorithms and IQMESH in practice.
- The Coordinator and the addresse does not route at all.
- Routers should not be moved (static routing backbone). After moving a router the discovery should be reinvoked.
- IQMESH supports communication between the Coordinator and a node only. Synchronous communication is recommended: requests initiated by the Coordinator and answers from nodes. Asynchronous packets are also allowed but they must be completely managed by the user. Attention must especially be paid to possible collisions. If both synchronous and asynchronous communications are combined each other, one of recommended approaches is to use different channels for both ones.

IQMESH allows up to 65 000 devices in single network but only 239 of them are allowed to route.
All algorithms works with the following parameters:

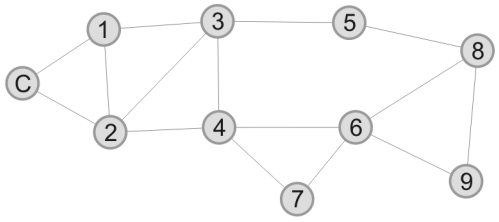
- `RTDT0`: number of hops per packet (0 – 239). 0 means direct delivery (without routing), e.g. 2 means 3 packets sent (sender + 2 routers). The user can set number of hops according to specific needs. But the maximal reasonable value is the number of routing nodes in the network. If an RF packet is sent with `RTDT0` = number of routers, it is called flooding. Every router resend the packet in dedicated time slot. This is the most robust but the most time consuming communication. It is necessary to use flooding for communication with a moving node (not knowing routers in range). If all nodes in the network operate also as backgroung routers it is possible to set `RTDT0` to the number of bonded nodes: `RTDT0 = eeReadByte(0)`. See example `E11-IQMESH`. `RTDT0` is decremented in each hop (arranged automatically by OS).
- `RTDT1`: time slot duration (in ticks). It should be longer than the transmit time for given packet. It depends on number of user data to be sent, `PIN`, `DLEN`, RF mode and RF speed. It can be approximately evaluated (in ticks) for 19.2 kb/s:
  - STD: (1 or 2) + 1 for every 24 B of user data
  - LP: (5 or 6) + 1 for every 24 B of user data
  - XLP: 120
- `RTDT2`: DID (Discovered ID) – identification of Discovery. Not set by the user.
- `RTDT3`: Upper byte of user address if user addressing is used.

If number of hops = number of bonded nodes the routing is very robust (flooding). But even flooding is no assurance that the packet is successfully routed (overdue if unsuitable order of Nodes – see chapter IQMESH in practice).

Routed packet is delivered in `frame = time slot length x number of hops` and **must not be answered until the frame is elapsed** otherwise a collision with routing devices occurs. Time starts from the packet sent by the Coordinator (the first slot is dedicated to the Coordinator). See example `E11-IQMESH-N` [10]).

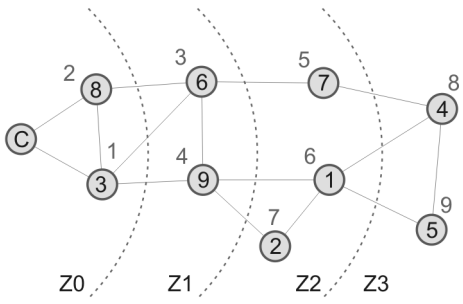
Routing algorithms

- **SFM (Static Full MESH)**
  Up to 240 devices in a network is allowed. Routing vector is fixed (1, 2, …, 239), logical addresses are used for addressing (`RX = logical address`). Broadcast address is 0xFF. Bonding can be done before placement, addresses must be known and devices must be placed with respect to topology (adresses should increase with the distance from the Coordinator). Discovery is no use.



  Example: Selecting `RTDT0=4` (4+1 hops) and `RTDT1=5` (50 ms time slot) is analogic to the only routing algorithm in OS v2.xx.

- **DFM (Discovered Full MESH)**
  Up to 240 devices in a network is allowed. Random placement is allowed and addresses need not be known for routing purpose but Discovery has to be performed after placement and bonding. Routing uses renumbered addresses (VRN, Virtual Routing Numbers) as a result of the Discovery process which creates a routing backbone with up to 240 devices devided to zones (based on minimal number of hops to individual Nodes). VRNs increase with the distance from the Coordinator (virtual routing backbone), are intended for OS only, the user need not take care about it. After a change in topology Discovery should be repeated. DFM is analogic to SFM but VRNs are used for routing instead of logical addresses. User addressing is completely the same (`RX = logical address`). Broadcast address is 0xFF.
  1 – **9**: logical addresses, 1 – 9: VRNs, Z0 – Z3: zones.



  - **DOM (Discovered Optimized MESH):** DOM is a special case of DFM. It is quite the same but number of hops is optimized (reduced) by the `optimizeHops()` function. It sets the `RTDT0` (number of hops) according to Discovery results to VRN of addressed Node to speed up the transmission at the cost of reduced redundancy.

- **DFM2B (Discovered Full MESH, 2 B)**
  The same as DFM but up to 65 000 devices and virtual routing backbone with up to 239 Nodes in a network is allowed. 2 B user addressing must be used based on `setUserAddress(x)`. Adressing: RTDT3 = high byte, RX = low byte. Broadcast address is 0xFFFF. Groups can be created by assigning the same addresses to more Nodes. `optimizeHops()` is not intended for DFM2B. See IQRF OS Reference guide [1], `bondNewNode()` and `setUserAddress()` for details. *Not fully implemented and tested yet.*

- **Tree:** Just one router in every zone is used. It is the fastest way to return packets back to the Coordinator but without a redundancy at all. It works for packets from Nodes to the Coordinator only. *Not fully implemented and tested yet.*

Tip: Routing algorithms can be mixed each other. Thus, the user can use faster algorithm first and then more redundant one(s) for not responding Nodes only.

Addressing overview

Routing algorithm	RTDEF	Address range	Addressing by	Addressing	Broadcast address
Not implemented	0x00	–	–	–	–
SFM	0x01	1 to 239	logical address	RX = …	0xFF
DFM	0x02	1 to 239	logical address	RX = …	0xFF
DFM2B	0x42	1 to 65 000	user address	RTDT3 = high byte RX = low byte	0xFFFF
Tree	0x08	1 to 239	logical address	RX = …	0xFF

Bonding

Devices are bonded to an IQMESH network when they are assigned to given Coordinator. Bonding is a mutual relationship between Coordinator and Node. Coordinator assigns a number (1 to 239) to the Node which can serve as device address. This short (1 B) address is used within the network. Individual network is identified via the unique four byte Module ID of the Coordinator - see Identification. This long ID is used outside the network.

Bonding is based on Node request (`bondRequest()`) confirmed by the Coordinator (`bondNewNode()`) via exchanging RF system packets. RF power is not limited by OS during bonding. To avoid possible influence on other modules, bonding can be performed on minimal distance with RF power lowered by the user.

The following bonding information is written in system EEPROMs (but they are not intended for direct user access):

- Coordinator:
  - Bit array. Individual flags = 1 if respective Node is bonded on Coordinator side
- Node:
  - Node number: short (1 B) device address
  - Network identification (4 B)
  - Flag if the Node is bonded on Node side

Address assigned by bonding can be specified by the user. If omitted, default is number of bonded nodes + 1. Thus, `bondNewNode(0)` is suitable for the initial bonding without discontinuities due to possible previous unbondings only.

The user can check results and make arbitrary changes in bonding at any time. There is a set of OS functions dedicated to bonding and related operations (access results, unbonding, rebonding etc). But once the Node is bonded and respective records are written to EEPROMs on both sides, Coordinator as well as Node starts keeping its own bonding information independently and no subsequent changes in bonding are carried over to opposite side via RF automatically arranged by OS.

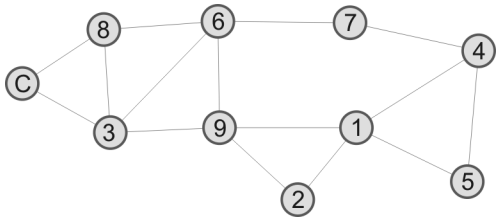
In short, only `bondRequest` and `bondNewNode` exchange RF system packets between Coordinator and Node. All subsequent changes in bonding by either Coordinator or Node are written to EEPROM just on one side. For example, `removeBondedNode()`, `rebondNode()` and `clearAllBonds()` operate with the Coordinator bit array only and `removeBond` operates with the Node flag only. If synchronization between Coordinator and Node after changes is needed it must be done by the application program. Static systems that suit IQRF best have moderate requirements for changes in bonding.

IQMESH in practice

Routing explanation examples

Bonding
Let us have 9 nodes bonded to a Coordinator. Every node has a label with its logical address (1 to 9, assigned during bonding). These addresses are used for actual addressing.

Installation
Place the nodes to desired positions. Nodes in range each other are marked with interconnection lines on the figure. These links exist but are not known at the moment.

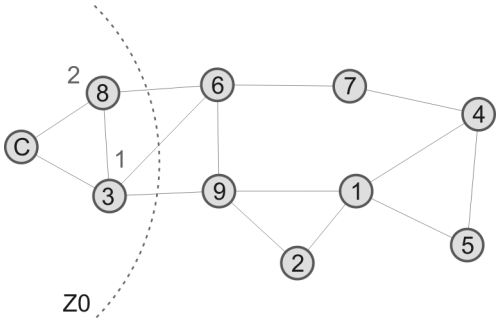


Sending a packet to Node 5
- C sends a packet. It is received by nodes N3 and N8.
- N3 waits 2 slots and then (in the third one) resends the packet.
- N8 waits 7 slots and then (in the 8th one) resends the packet.
- In slots 1 and 2 no packets are resent because nodes N1 and N2 have not received the packet.
- The packet resent by N3 in slot 3 is received by N8, N6 and N9. (C ignores the packet.)
- N8 receives the packet for the second time and is still waiting for its time slot.
- N6 receives the packet for the first time and is waiting for its time slot.
- N9 receives the packet for the first time and is waiting for its time slot.
- etc.
- In slot 9 N9 resent the packets to N1 and N2 but no one of them can route because their slots are already expired. Thus, destination N5 can not be reached from N1.
- Similarly, neither N4 can route the packet to N5 since it received this in slot 7 (from N7) after expiration (slot 4).

It is evident that N5 can never be accessed in such configuration and this MESH is not well designed. This MESH would work well if nodes are placed according to their logical addresses (ascending from the coordinator). This can be used in special cases only (e.g. street light lamps in a straight line without branches, with the Coordinator on one edge – see the figure on page 17, SFM). In such cases the SFM routing algorithm (without Discovery) can be used.

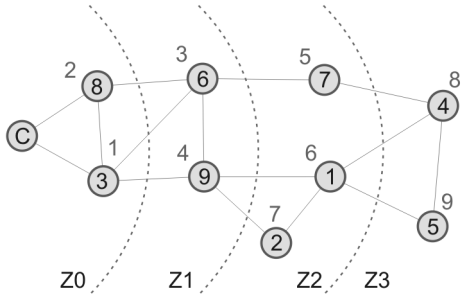
In example above it is necessary to use the DFM routing. To utilize DFM, an additional step (Discovery) must be performed. From the source code point of view it means that to call the `discovery(z)` function and wait for the result returning number of discovered nodes. Duration takes some time (it can be from tens of seconds to tens of minutes) depending on number of nodes, topology etc.

Discovery
After the `discovery(z)` is called the following system communication on background is invoked:
- C sends a request packet: "Whoever is in range, answer me".
- N3 and N8 answers in this example. They are nodes in direct range from the Coordinator, creating the Zone 0 (Z0).
- C sends a packet to N3 (lower logical address has always priority): "You have assigned virtual routing number VRN = 1"
- C sends a packet to N8: "You have assigned VRN = 2"



- Now C passes control to N3 (lower logical address has also priority): "Discover your neighbors which have not been discovered yet, assign VRNs to them and give results back to me. The first free VRN is 3."
- N3 discovers new neighbors N6 (assigning VRN = 3 to it) and N9 (assigning VRN = 4).
- C passes control to N8 but it discovers no new neighbors.
- Now Zone 1 (N6 and N9) is discovered (nodes accessible from/to Coordinator via two hops.).
- C passes control to N6 which discovers N7 assigning VRN = 5 to it.
- etc.

Resulting renumbered network is arranged in ascending order according VRNs (DFM routing).



Now routing according the rule mentioned above will work. But the user does not know VRN addresses. It is not needed. To address nodes, the user uses logical addresses assigned by bonding.

Resume:

Routing algorithm	Addressing by	Routing by
SFM	logical address	logical address
DFM	logical address	VRN

The discovery(z) function has an input parameter z. It is max. number of zones to be discovered. Return value is number of discovered Nodes. To analyse the network more detailed, discovery can be called repeatedly with increased z. So 2 nodes in Z1, 2 nodes in Z2 etc. can be found in this example.

- If all nodes operate also as background routers the most commonly z value is number of bonded nodes: discovery(eeReadByte(0x00)). (The Coordinator stores the number of bonded nodes in EEPROM at address 0.) It is used also in E11-IQMESH example [27]. This solves even the worst case when there is just one node in every zone – the Chain topology which is the worst MESH case (the less robust one) without redundant paths at all.
- If some nodes do not route number of routers is used as the z parameter.

Example: Communication with N1. Topology see the picture above, DFM.

Packet from the Coordinator to the node (request):

```
...
setCoordinatorMode();      // To select the Coordinator and networking communication
RX = 1;
DLEN = 10;                 // Data are prepared in buffer RF
PIN = 0;                   // Preclearing
//_NTWF = 1; // not necessary. This flag is set automatically by setCoordinatorMode()
_ROUTEF = 1;               // To route outgoing packet (the PIN.5 flag)
RTDEF = 2;                 // DFM
RTDT0 = eeReadByte(0x00); // Number of hops = number of bonded nodes
                           // (only if all nodes operate also as background routers)
RTDT1 = 2;                 // 20 ms is enough for DLEN=10. See the formula at page 17.
RFTXpacket();
...
```

This code induces so called network flooding. It means that every discovered node (with a VRN assigned) routes the packet in its time slot. So routing vector 1, 2, 3, 4, 5, 6, 7, 8, 9 is used. Flooding is used especially in networks with moving nodes. They should not be in a routing structure. (It is not known which routers are in range. Moreover, it is changing in time.)

Besides of nodes routing in background, the network can also include nodes which do not route at all. For such nodes the following conditions should be kept:

- The `setRoutingOff()` function must be called once during initialization (default OS value is Routing On).
- The answerSystemPacket() function should not be called
- It is no use calling the `wasRouted()` function.

N1 receives the packet four times in total:

- In 4. slot from N9
- In 7. slot from N2
- In 8. slot from N4
- In 9. slot from N5

Note: Addressed Node does not route.

Thus, the connection with N1 is quite robust. E.g. after failure in 4. slot the packet is still received in 8. and 9. slot.

In case of flooding the communication takes time according the formula:

Total time = time slot duration x number of hops (RTDT1 x RTDT0).

If the answer is required the time is twice longer.
IQMESH offers speeding up using optimizing number of hops – DOM (Discovered Optimized MESH). The optimizeHops() function called before sending a packet sets the number of hops (register RTDT0) to specified value. Parameter 0xFF means to copy VRN of addressed node to RTDT0. It should be tested whether the addressed node has been discovered (otherwise it has no VRN).

```
...
if (isDiscoveredNode(RX))
    optimizeHops(0xFF);       // DOM
RFTXpacket();
...
```

After this optimizing number of hops is reduced to 6 (including a packet from the Coordinator). Only nodes with VRNs 1 to 5 route packets. Routing vector is 1, 2, 3, 4, 5. N1 receives the packet only once (in 4. slot from N9).
Optimizing leads to faster communication (less time slots) but the robustness is reduced due to less redundant paths. Routing method should be specified according to particular needs of an application. E.g. a packet can be sent with optimization at first and if no answer is returned the packet is sent once more not optimized.

Packet from the node to the Coordinator (answer):

Registers RTDEF and RTDT0–3 are directly included in RF packet. That is why the routing algorithm, current time slot, number of hops and time left to end of routing are known for every node which has received the packet. OS automatically decrements RTDT0 in every hop. This is used for answers. Addressed node can receive the packet (RFRXpacket returns 1) e.g. in 2. slot of 9 slots but it should respond not until all routing is finished otherwise a collision may occur. Waiting can be ensured by a simple delay loop. Delay time is: number of hops left x time slot duration (RTDT1 x RTDT0)

```
while (RTDT0)        // RTDT0 - the rest of hops
{
  waitDelay(RTDT1);  // RTDT1 - timeslot
  RTDT0--;
}
```

The answer can be sent not until this time is elapsed.
Additionally, this can be used for synchronization of all nodes in the network. E.g. a broadcast packet with a command to switch all lamps on in street lighting is received by individual lamps in different time slots. After this delay all lamps are synchronized and can be switched on at the same time.

The transmitting should be designed as follows:

```
...
setNodeMode();
RX = 0;               // To Coordinator
DLEN = 10;            // Data are already prepared in buffer RF

// for SFM
  RTDEF = 1;          // SFM
  getNetworkParams(); // Returns logical Node address in param2
  RTDT0 = param2;

// for DFM:
  RTDEF = 2;          // DFM
  RTDT0 = 0xFF;       // OS includes VRN in the packet into RTDT0

RTDT1 = 2;            // 20 ms is enough for DLEN=10, see the formula on page 17.
RFTXpacket();
...
```

Note:
If immediate answering (without calling another RFRXpacket meanwhile), it is not necessary to set the PIN (it remains stored from the received packet).

Besides of the RTDEF register, the main difference between SFM and DFM is in setting of number of hops (RTDT0):
- For SFM: RTDT0 = one's own logic address
- For DFM: RTDT0 = one's own VRN

Routing works in similar way like requests from Coordinator but time slot order is automatically reversed. Thus, for DFM answer from N1 (VRN=6) routing vector 5, 4, 3, 2, 1 is used:

- N1 (VRN 6) sends the answer, received by N2, N9, N4 and N5.

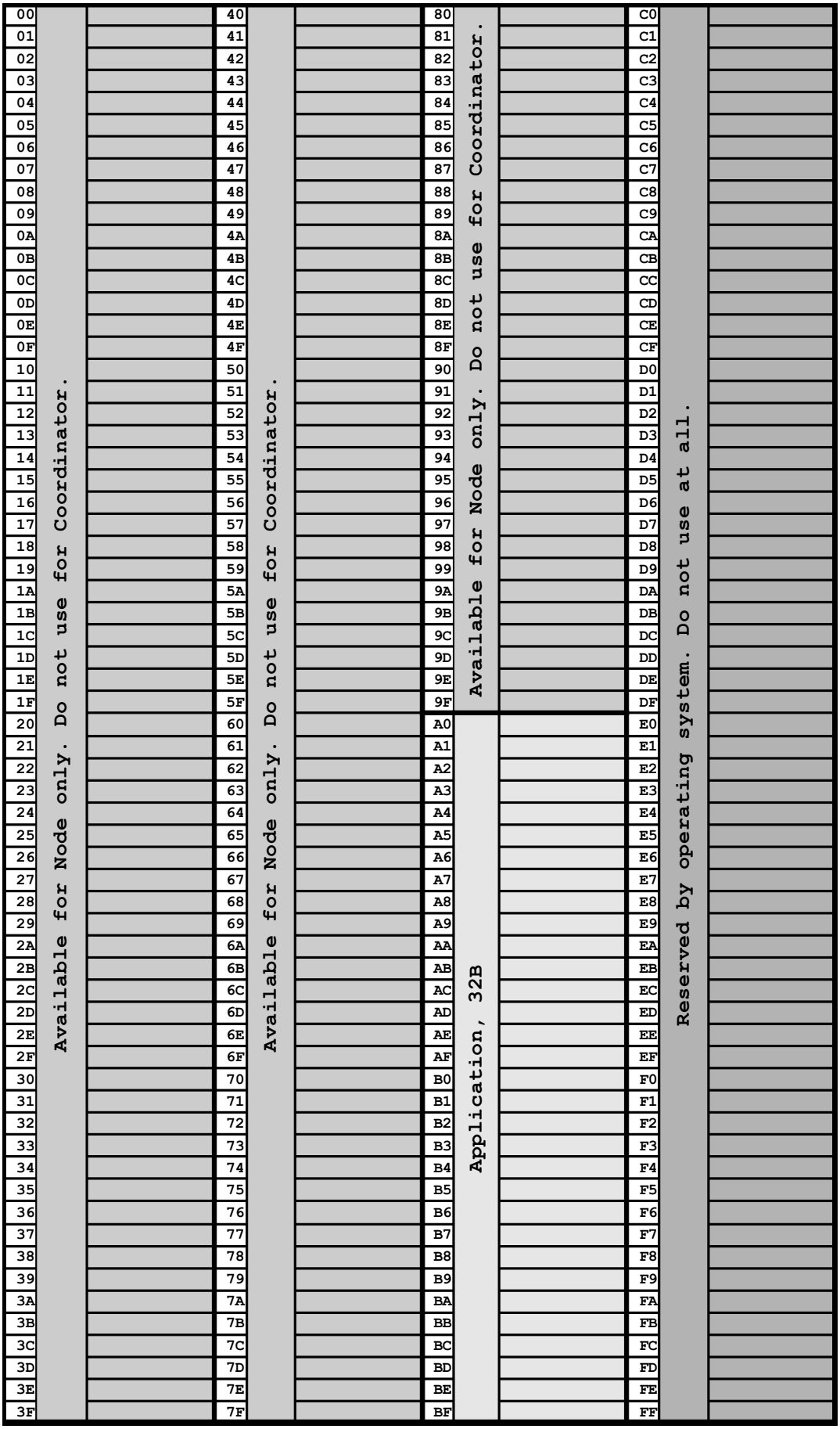
- N2, N4 and N5 do not resend the packet because their VRNs (7, 8, 9) are higher than sender's VRN (6).
- N9 (VRN = 4) waits one time slot and then resend the packet. This skipped slot should belong to N7 (VRN = 5), but it has not received the packet.
- The packet from N9 is received by N6, N3, N1 and N2.
- N1 and N2 do not respond because their VRNs (6 and 7) are higher than sender's VRN (4).
- N3 (VRN = 1) waits 2 slots
- N6 (VRN = 3) resends the packet in the next slot
- etc.

Thus, the Coordinator receives the packet twice:
- In penult slot from N8 (VRN = 2)
- In last slot from N3 (VRN = 1)

Note:
If a sender of DFM answer is not discovered (having no VRN) OS automatically ensures sending to the node the request has been received from. Then routing normally goes on.

Appendix 1

EEPROM map



RAM map (PIC16F886)

IRP = 0				IRP = 1			
Bank 0		Bank 1		Bank 2		Bank 3	
00	Ind. addr.	80	Ind. addr.	100	Ind. addr.	180	Ind. addr.
01	TMR0	81	OPTION_REG	101	TMR0	181	OPTION_REG
02	PCL	82	PCL	102	PCL	182	PCL
03	STATUS	83	STATUS	103	STATUS	183	STATUS
04	FSR	84	FSR	104	FSR	184	FSR
05	PORTA	85	TRISA	105	WDTCON	185	SRCON
06	PORTB	86	TRISB	106	PORTB	186	TRISB
07	PORTC	87	TRISC	107	CM1CON0	187	BAUDCTL
08	–	88	–	108	CM2CON0	188	ANSEL
09	PORTE	89	TRISE	109	CM2CON1	189	ANSELH
0A	PCLATH	8A	PCLATH	10A	PCLATH	18A	PCLATH
0B	INTCON	8B	INTCON	10B	INTCON	18B	INTCON
0C	PIR1	8C	PIE1	10C	EEDAT	18C	EECON1
0D	PIR2	8D	PIE2	10D	EEADR	18D	EECON2
0E	TMR1L	8E	PCON	10E	EEDATH	18E	–
0F	TMR1H	8F	OSCCON	10F	EEADRH	18F	–
10	T1CON	90	OSCTUNE	110	00	190	
11	TMR2	91	SSPCON2	111	01	191	
12	T2CON	92	PR2	112	02	192	
13	SSPBUF	93	SSPADD	113	03	193	
14	SSPCON	94	SSPSTAT	114	04	194	
15	CCPR1L	95	WPUB	115	05	195	
16	CCPR1H	96	IOCB	116	06	196	
17	CCP1CON	97	VRCON	117	07	197	
18	RCSTA	98	TXSTA	118	08	198	
19	TXREG	99	SPBRG	119	09	199	
1A	RCREG	9A	SPBRGH	11A	10	19A	
1B	CCPR2L	9B	PWM1CON	11B	11	19B	
1C	CCPR2H	9C	ECCPAS	11C	12	19C	
1D	CCP2CON	9D	PSTRCON	11D	13	19D	
1E	ADRESH	9E	ADRESL	11E	14	19E	
1F	ADCON0	9F	ADCON1	11F	15	19F	
20	00	A0	00	120	16	1A0	
21	01	A1	01	121	17	1A1	
22	02	A2	02	122	18	1A2	
23	03	A3	03	123	19	1A3	
24	04	A4	04	124	20	1A4	
25	05	A5	05	125	21	1A5	
26	06	A6	06	126	22	1A6	
27	07	A7	07	127	23	1A7	
28	08	A8	08	128	24	1A8	
29	09	A9	09	129	25	1A9	
2A	10	AA	10	12A	26	1AA	
2B	11	AB	11	12B	27	1AB	
2C	12	AC	12	12C	28	1AC	
2D	13	AD	13	12D	29	1AD	
2E	14	AE	14	12E	30	1AE	
2F	15	AF	15	12F	31	1AF	
30	16	B0	16	130	32	1B0	
31	17	B1	17	131	33	1B1	
32	18	B2	18	132	34	1B2	
33	19	B3	19	133	35	1B3	
34	20	B4	20	134	36	1B4	
35	21	B5	21	135	37	1B5	
36	22	B6	22	136	38	1B6	
37	23	B7	23	137	39	1B7	
38	24	B8	24	138	40	1B8	
39	25	B9	25	139	41	1B9	
3A	26	BA	26	13A	42	1BA	
3B	27	BB	27	13B	43	1BB	
3C	28	BC	28	13C	44	1BC	
3D	29	BD	29	13D	45	1BD	
3E	30	BE	30	13E	46	1BE	
3F	31	BF	31	13F	47	1BF	

bufferINFO (20–31) · bufferCOM (A0–BF) · bufferRF (100–13F)

IRP = 0				IRP =1			
Bank 0		Bank 1		Bank2		Bank3	
40	32	C0	32	140	48	1C0	00
41	33	C1	33	141	49	1C1	01
42	34	C2	34	142	50	1C2	02
43		C3	35	143	51	1C3	03
44		C4	36	144	52	1C4	04
45		C5	37	145	53	1C5	05
46		C6	38	146	54	1C6	06
47		C7	39	147	55	1C7	07
48		C8	40	148	56	1C8	08
49		C9		149	57	1C9	09
4A		CA		14A	58	1CA	10
4B		CB		14B	59	1CB	11
4C	toutRF	CC		14C	60	1CC	12
4D		CD		14D	61	1CD	13
4E		CE		14E	62	1CE	14
4F		CF		14F	63	1CF	15
50		D0		150		1D0	16
51		D1		151		1D1	17
52		D2		152		1D2	18
53		D3		153	PIN 00	1D3	19
54		D4		154	DLEN 01	1D4	20
55		D5		155	02	1D5	21
56		D6		156	RX 03	1D6	22
57		D7		157	TX 04	1D7	23
58		D8		158	05	1D8	24
59		D9		159	06	1D9	25
5A		DA		15A	PID 07	1DA	26
5B		DB		15B	RTOTX 08	1DB	27
5C		DC		15C	RTDEF 09	1DC	28
5D		DD	SPIpacketLength	15D	RTDT0 10	1DD	29
5E		DE		15E	RTDT1 11	1DE	30
5F		DF		15F	RTDT2 12	1DF	31
60		E0		160	RTDT3 13	1E0	
61		E1		161	MPRW0 14	1E1	
62		E2		162	MPRW1 15	1E2	
63		E3		163	MPRW2 16	1E3	
64		E4		164	17	1E4	
65		E5		165	18	1E5	
66		E6		166	19	1E6	
67		E7		167	20	1E7	
68		E8		168	21	1E8	
69		E9		169		1E9	
6A		EA		16A		1EA	
6B		EB		16B		1EB	
6C		EC		16C		1EC	
6D		ED		16D		1ED	userStatus
6E		EE		16E		1EE	memoryOffsettTo
6F	lastRSSI	EF		16F		1EF	memoryOffsetFrom
70	userReg0	F0	userReg0	170	userReg0	1F0	userReg0
71	userReg1	F1	userReg1	171	userReg1	1F1	userReg1
72	param1	F2	param1	172	param1	1F2	param1
73	param2	F3	param2	173	param2	1F3	param2
74	param3	F4	param3	174	param3	1F4	param3
75		F5		175		1F5	
76	param4	F6	param4	176	param4	1F6	param4
77		F7		177		1F7	
78		F8		178		1F8	
79		F9		179		1F9	
7A		FA		17A		1FA	
7B		FB		17B		1FB	
7C		FC		17C		1FC	
7D		FD		17D		1FD	
7E		FE		17E		1FE	
7F		FF		17F		1FF	

bufferCOM (40–48) · bufferRF (140–148) · networkInfo (15A–168) · bufferAUX (1C0–1CF)

– reserved for PIC HW · – OS buffers · – reserved for OS · – user available

Smarter wireless. Simply.

Appendix 2

868 MHz band channel map

Channel	Bit rate		
	BR1	BR2	BR3
	1.2 - 19.2	**57,6**	**86,2**
	Frequency [MHz]		
0	863.15	863.15	863.15
1	863.25	863.35	863.55
2	863.35	863.55	863.95
3	863.45	863.75	864.35
4	863.55	863.95	864.75
5	863.65	864.15	865.15
6	863.75	864.35	865.55
7	863.85	864.55	865.95
8	863.95	864.75	866.35
9	864.05	864.95	866.75
10	864.15	865.15	867.15
11	864.25	865.35	867.55
12	864.35	865.55	867.95
13	864.45	865.75	**868.35**
14	864.55	865.95	868.75
15	864.65	866.15	
16	864.75	866.35	
17	864.85	866.55	
18	864.95	866.75	
19	865.05	866.95	
20	865.15	867.15	
21	865.25	867.35	
22	865.35	867.55	
23	865.45	867.75	
24	865.55	867.95	
25	865.65	868.15	
26	865.75	**868.35**	
27	865.85	868.55	
28	865.95	868.75	
29	866.05	868.95	
30	866.15		
31	866.25		
32	866.35		
33	866.45		
34	866.55		
35	866.65		
36	866.75		
37	866.85		
38	866.95		
39	867.05		
40	867.15		
41	867.25		
42	867.35		
43	867.45		
44	867.55		
45	867.65		
46	867.75		
47	867.85		
48	867.95		
49	868.05		
50	868.15		
51	868.25		
52	**868.35**		
53	868.45		
54	868.55		
55	868.65		
56	868.75		
57	868.85		
58	868.95		
59	869.05		
60	869.15		
61	869.25		

g band	863.000 - 868.000 MHz	duty 0.1%
g1 band	868.000 - 868.600 MHz	duty 1%
g2 band	868.700 - 869.200 MHz	duty 0.1%

916 MHz band channel map

Channel	Bit rate			Channel	Bit rate		Channel	Bit rate
	BR1	BR2	BR3		BR1	BR2		BR1
	1.2 - 19.2	**57.6**	**86.2**		**1.2 - 19.2**	**57.6**		**1.2 - 19.2**
	Frequency [MHz]				**Frequency [MHz]**			**Frequency [MHz]**
0	900.90	900.90	900.90	**63**	910.35	919.80	**126**	919.80
1	901.05	901.20	901.50	**64**	910.50	920.10	**127**	919.95
2	901.20	901.50	902.10	**65**	910.65	920.40	**128**	920.10
3	901.35	901.80	902.70	**66**	910.80	920.70	**129**	920.25
4	901.50	902.10	903.30	**67**	910.95	921.00	**130**	920.40
5	901.65	902.40	903.90	**68**	911.10	921.30	**131**	920.55
6	901.80	902.70	904.50	**69**	911.25	921.60	**132**	920.70
7	901.95	903.00	905.10	**70**	911.40	921.90	**133**	920.85
8	902.10	903.30	905.70	**71**	911.55	922.20	**134**	921.00
9	902.25	903.60	906.30	**72**	911.70	922.50	**135**	921.15
10	902.40	903.90	906.90	**73**	911.85	922.80	**136**	921.30
11	902.55	904.20	907.50	**74**	912.00	923.10	**137**	921.45
12	902.70	904.50	908.10	**75**	912.15	923.40	**138**	921.60
13	902.85	904.80	908.70	**76**	912.30	923.70	**139**	921.75
14	903.00	905.10	909.30	**77**	912.45	924.00	**140**	921.90
15	903.15	905.40	909.90	**78**	912.60	924.30	**141**	922.05
16	903.30	905.70	910.50	**79**	912.75	924.60	**142**	922.20
17	903.45	906.00	911.10	**80**	912.90	924.90	**143**	922.35
18	903.60	906.30	911.70	**81**	913.05	925.20	**144**	922.50
19	903.75	906.60	912.30	**82**	913.20	925.50	**145**	922.65
20	903.90	906.90	912.90	**83**	913.35	925.80	**146**	922.80
21	904.05	907.20	913.50	**84**	913.50	926.10	**147**	922.95
22	904.20	907.50	914.10	**85**	913.65	926.40	**148**	923.10
23	904.35	907.80	914.70	**86**	913.80	926.70	**149**	923.25
24	904.50	908.10	915.30	**87**	913.95	927.00	**150**	923.40
25	904.65	908.40	915.90	**88**	914.10	927.30	**151**	923.55
26	904.80	908.70	**916.50**	**89**	914.25	927.60	**152**	923.70
27	904.95	909.00	917.10	**90**	914.40	927.90	**153**	923.85
28	905.10	909.30	917.70	**91**	914.55	928.20	**154**	924.00
29	905.25	909.60	918.30	**92**	914.70	928.50	**155**	924.15
30	905.40	909.90	918.90	**93**	914.85	928.80	**156**	924.30
31	905.55	910.20	919.50	**94**	915.00	929.10	**157**	924.45
32	905.70	910.50	920.10	**95**	915.15		**158**	924.60
33	905.85	910.80	920.70	**96**	915.30		**159**	924.75
34	906.00	911.10	921.30	**97**	915.45		**160**	924.90
35	906.15	911.40	921.90	**98**	915.60		**161**	925.05
36	906.30	911.70	922.50	**99**	915.75		**162**	925.20
37	906.45	912.00	923.10	**100**	915.90		**163**	925.35
38	906.60	912.30	923.70	**101**	916.05		**164**	925.50
39	906.75	912.60	924.30	**102**	916.20		**165**	925.65
40	906.90	912.90	924.90	**103**	916.35		**166**	925.80
41	907.05	913.20	925.50	**104**	**916.50**		**167**	925.95
42	907.20	913.50	926.10	**105**	916.65		**168**	926.10
43	907.35	913.80	926.70	**106**	916.80		**169**	926.25
44	907.50	914.10	927.30	**107**	916.95		**170**	926.40
45	907.65	914.40	927.90	**108**	917.10		**171**	926.55
46	907.80	914.70	928.50	**109**	917.25		**172**	926.70
47	907.95	915.00	929.10	**110**	917.40		**173**	926.85
48	908.10	915.30		**111**	917.55		**174**	927.00
49	908.25	915.60		**112**	917.70		**175**	927.15
50	908.40	915.90		**113**	917.85		**176**	927.30
51	908.55	916.20		**114**	918.00		**177**	927.45
52	908.70	**916.50**		**115**	918.15		**178**	927.60
53	908.85	916.80		**116**	918.30		**179**	927.75
54	909.00	917.10		**117**	918.45		**180**	927.90
55	909.15	917.40		**118**	918.60		**181**	928.05
56	909.30	917.70		**119**	918.75		**182**	928.20
57	909.45	918.00		**120**	918.90		**183**	928.35
58	909.60	918.30		**121**	919.05		**184**	928.50
59	909.75	918.60		**122**	919.20		**185**	928.65
60	909.90	918.90		**123**	919.35		**186**	928.80
61	910.05	919.20		**124**	919.50		**187**	928.95
62	910.20	919.50		**125**	919.65		**188**	929.10

Documentation and Information

1 **IQRF** OS Reference guide www.iqrf.org/weben/downloads.php?id=156
2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix [1]
3 **IQRF** home page www.iqrf.org
4 **IQMESH** specification www.iqmesh.org/iqmesh
5 **SPI** specification www.iqrf.org/weben/downloads.php?id=85
6 **IQRF support** site www.iq-esupport.com
7 **TR-52B** datasheet: www.iqrf.org/weben/downloads.php?id=91
TR-53B datasheet: http://www.iqrf.org/weben/downloads.phpd?id=163
8 **PIC16F886** datasheet: www.iqrf.org/weben/downloads.php?id=126
9 **IQRF IDE:** www.iqrf.org/weben/downloads.php?id=86
10 **Basic examples** (included in the StartUp Package): www.iqrf.org/weben/downloads.php?id=112
11 **AN003 – IQRF development tools Installation guide**: http://www.iqrf.org/weben/downloads.php?id=109

If you need a help or more information please visit IQRF support pages [6] and Submit a Ticket with your request. A lot of information is also available in the IQRF OS User's guide [1] and on the IQRF home page [3].

Document revision

- 111213 More detailed routing explanation. Routing zones numbered from 0 but not from 1. Chapter IQMESH in practice added. Extended Tip to explain `userStatus`.
- 110621 MCU clock switch from 8 MHz restricted (page 7).
- 110124 Importance of time frames highlighted on page 17. Configuration words and related information updated.
- 110112 Information aded and precised
- 101223 Preliminary release, OS v3.00

If you need a help or more information please visit IQRF support pages [6] and Submit a Ticket with your request. A lot of information is also available on the IQRF web site [3].

Sales and Service

Corporate office:

MICRORISC s.r.o., Delnicka 222, 506 01 Jicin, Czech Republic, EU
Tel: +420 493 538 125, Fax: +420 493 538 126, www.microrisc.com

Partners and distribution:

please visit www.iqrf.org/partners

Quality management:

ISO 9001 : 2009 certified

Trademarks:

The IQRF name and logo are registered trademarks of MICRORISC s.r.o.
PIC, SPI, Microchip, RFM and all other trademarks mentioned herein are property of their respective owners.

Legal:

All information contained in this publication is intended through suggestion only and may be superseded by updates without prior notice. No representation or warranty is given and no liability is assumed by MICRORISC s.r.o. with respect to the accuracy or use of such information.

Without written permission it is not allowed to copy or reproduce this information, even partially.

No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

The IQRF products utilize several patents (CZ, EU, US)

Website	**www.iqrf.org**
E-mail	**sales@iqrf.org**
On-line support	**support@iqrf.org**

Smarter wireless. Simply.