IQRF OS

Operating System

Version 3.03D
for TR-5xD

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 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		
TR–31B	v2.10 for 3xB	
TR–32B		
TR–52B	v3.00	FSK
TR–53B		
TR-52D	v3.03D	
TR-54D		
TR-55D		
TR-56D		

Communication is possible among TR modules with the same type of modulation only. IQRF OS v3.0x is compatible with 2.11 in STD mode only.

It is possible to mix TR-5xB (with IQRF OS v3.00) and TR-5xD (with IQRF OS v3.01D or highers) within a single IQMESH. It is just recommended to use TR-5xD with IQRF OS 3.02D or higher as a network coordinator.

IQRF OS v3.03D or higher requires IQRF IDE v4.11 or higher.

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.

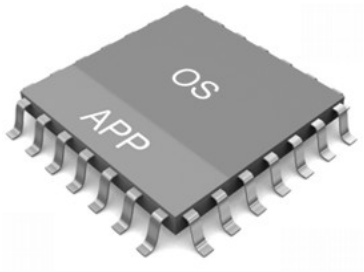
IQRF OS versions and history

Version	Main differences	Release	Status
v3.03D for 5xD	• Function getRSSI. • DPA (Direct Peripheral Addressing) support with network visualization • Some other improvements • **IQRF IDE v4.11 or higher required**	Nov 2012	**current** for **TR-52D**, **TR-54D, TR-55D** and **TR-56**
v3.02D for 5xD	• User interrupt, Timer6 user available • Lower power consumption in LP/XLP • Functions `setINDFx` and `getINDFx` • Series of other enhancements, see Appendix 4	Sep 2012	not for new designs for **TR-52D** and **TR-54D**
v3.01D for 5xD	• **Powerful MCU** PIC16LF1938: **extended memories** (RAM 1024 B, Flash 16 kwords), **16-level stack**, efficient architecture, faster interrupt. • External **serial EEPROM** 16 kb user available (not for Coordinator) • Several additional improvements	Apr 2012	not for new designs for **TR-52D** and **TR-54D**
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 significantly reduces time and development costs.

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, ...) utilizing HW communication modules inside the MCU.

Complex standard communication interfaces (**USB**, **Ethernet**, **GSM**, …) can not be realized by a single TR module but using IQRF gateways.

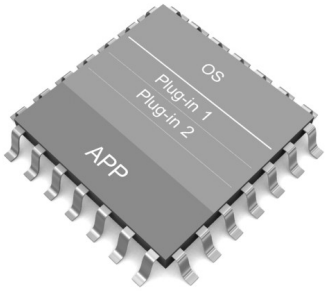
OS supports low power consumption of IQRF transceivers with the **Sleep** functions when operation of TR module or RF IC is reduced/stopped. Power management RF modes optimize consumption when RF is active.

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, but can also be created by the user in collaboration with 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.

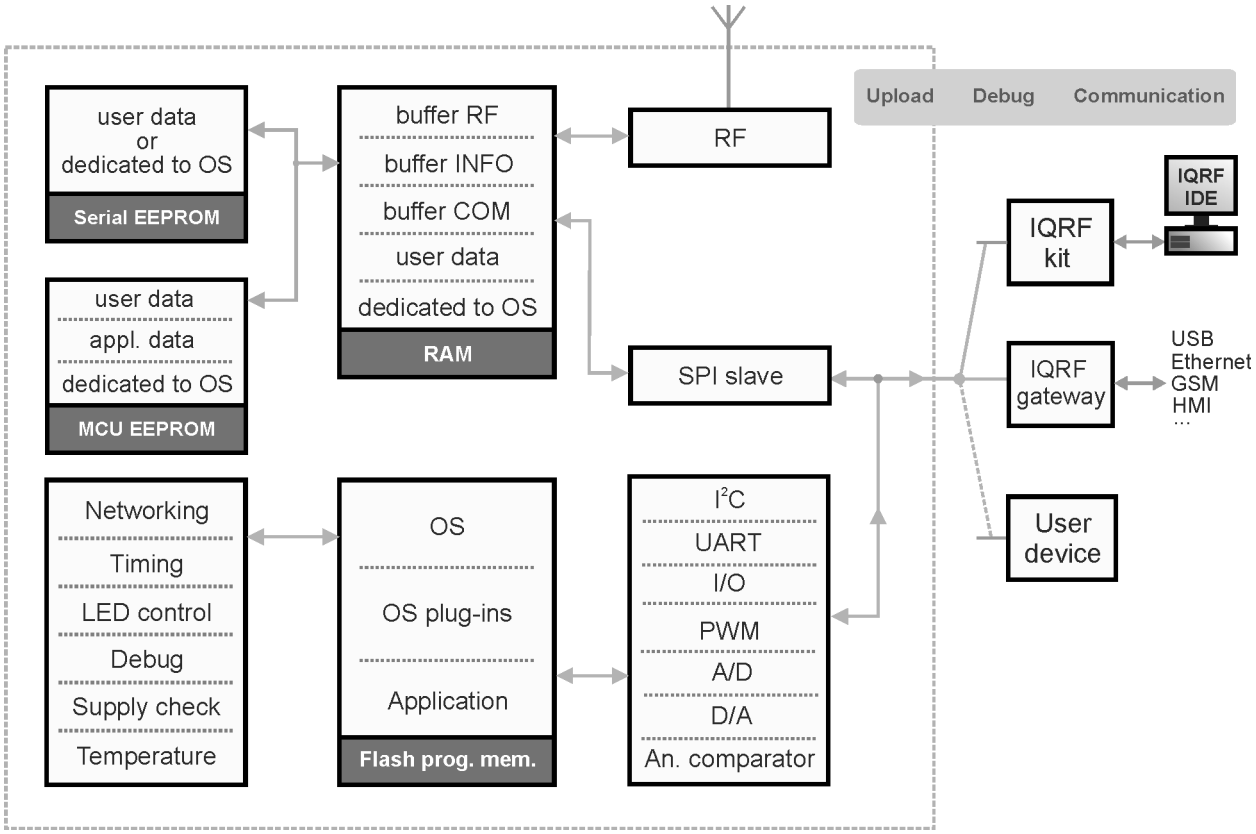
Example: `#include "plug-ins/PlugInXY.h"`

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

Flash memory area dedicated to user OS plug-ins can be partly or completely used as an extension of user code (program) memory. See chapter *Flash memory* and IQRF IDE 4 Help.

IQRF OS Architecture

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



Individual blocks:

- Memories:
  - program memory (Flash – inside MCU)
  - data memory (RAM – inside MCU)
  - data memory (EEPROM – inside MCU)
  - data memory (EEPROM – external serial)
- Communication interface:
  - RF (wireless)
  - SPI (standard serial, 4-wire, slave, running in OS background)
- Temperature sensor (optional)
- Power supply check
- Digital I/O (input/output)
- A/D converter
- D/A converter
- Analog comparator
- PWM output
- Time base support: 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 is 868 MHz, may depend on delivery destination. Default 916 MHz on request.
  - **433 MHz** is implemented for several TR types by TR hardware modification.
- **Channels:** Within a given band, there are separate frequency channels (SW selectable). 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 preliminary, provisionally intended for experimental purposes only. All OS and IQMESH functionality and 433 MHz TR modules are 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.

Users have to ensure observing local provisions and restrictions relating to the use of short range devices by software, e.g. the CEPT ERC/REC 70-03 Recommendation and subsequent amendments in EU.

Microcontroller

IQRF OS for TR-52D, TR-54D and compatibles is implemented in the **PIC16LF1938** MCU (8-bit microcontroller by Microchip) – datasheet see [8].

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

PIC HW resources		Utilization
Program memory	Flash	1536 instructions
Data memory	RAM	96 B – user data 160 B – communication/system buffers
	MCU EEPROM	Node: 160 B user data + 32 B application data
		Coordinator: 0 B user data + 32 B application data
	Serial EEPROM	2 KB, SPI serial interface, 16 B block access. Not available for Coordinator utilizing Discovery.
I/O pins	TR-52D	6 × I/O
	TR-54D, TR-56D	11 × I/O, 1 input only
	TR-55D	9 × I/O, 1 input only
A/D converter		10 b. Multiplexed S&H. Number of pins depends on TR type.
D/A converter		5 b. See Application examples [10]. Allowed range is from VSS to VDD only.
Analog comparator		See Application examples [10]. Allowed CIN+ input is GND only.
Voltage reference (FVR)		Dedicated to OS. Do not change.
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]
PWM output		Realized by PIC HW module – see Application examples [10]
Interrupt		User available. Global interrupt can be disabled just for a short period if necessary.
Stack (for subroutines)		Max. 5 levels of subroutine calling is allowed.
Timer6		Fully user available.
Power-on reset		Utilizes HW filter to eliminate improper power-up rising and spikes to some extent.
Brown-out reset		Default disabled, can be enabled and disabled by SW.
Power-up timer		Enabled
Watchdog		Default disabled, can be enabled by SW (`SWDTEN = 1`). Time-out can be set from 1 ms to 256 s, 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 is not derived from this but from RF IC with crystal precision.
Configuration words ("Fuses")		`CONFIG1 = 0x2A0C, CONFIG2 = 0x1EFE`

These resources partly depend on TR type and 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 `EECONx` registers) to the EEPROM and Flash is not allowed at all, extra OS functions are intended for access to EEPROM. 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 `INDFx` registers 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 and an on-board serial EEPROM.

Individual parts of memories are:
- Dedicated to the MCU core and MCU peripherals
- 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 TR module and OS implementation.

Illegal modification of dedicated memory locations can cause system malfunction.

Several header files (with the .H extension) are included in installation package of IQRF IDE development environment. They are intended for C compiler to provide 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 `16F1938.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 (access via registers `INDFx` or `EECONx` is restricted). The program remains stored there even after power off. Overwriting is not unlimited, number of erase/write cycles is about 10 000 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.

- Area `0x3A00` – `0x3FFF` (1536 machine instructions) is available for user program.
- Area `0x3000` – `0x37DF` (2016 machine instructions) is a part of locations dedicated to plug-ins but can also be used (partly or completely) as additional space for user program. It is available under the licence based on NDA (Non-disclosure agreement).
- Remaining area is dedicated to OS and OS plug-ins.

User program should begin from address `0x3A00`. It is automatically arranged by the IQRF header files. `void APPLICATION()` must be the first function in user code. See IQRF code examples.

Data memory (RAM)

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

RAM access implemented in MCU

Direct RAM access

The MCU logical RAM space is devided to 32 banks (0 – 31, up to 128 B each) with the same architecture:
- Core: dedicated to MCU core, shared (12 identical registers) in all 32 banks
- SFR: Special function registers: dedicated to MCU peripherals
- GPR: General purpose registers
- Common: like GPR but shared (16 identical registers) in all 32 banks

The Core and Common registers are directly accessible without bank preselection. But to access individual SFR or GPR registers directly (e.g. `userVariable = 0`), just the address offset (see the table) is used and given bank must be preselected first by the compiler. See the table of one bank on the right and PIC datasheet [8] for details.

Address offset [hex]	Memory region
00 0B	Core
0C 1F	SFR
20 6F	GPR
70 7F	Common

Indirect RAM access

MCU allows to access GPRs also indirectly in a non-banked methods using addressing via FSRx register pairs. There are two possibilities:

- **Traditional access** using the absolute addresses containing the bank as well as the address offset (FSRx = 0x000 to 0xFFF). It is available also for the Core, SFR and Common registers. GPR locations available by OS for the user are FSRx = 0x5C0 to 0x5EF and FSRx = 0x620 to 0x64F (see below).
- **Linear memory:** Besides of absolute addresses, GPR registers can also be accessed by virtual addresses (FSR = 0x2000 to 0x29AF) completely independent on banking. Among others, this enables linear access to long user arrays. GPR locations available by OS for the user are FSRx = 0x2390 to 0x23EF (see below).

RAM architecture implemented in TR modules by OS

- RAM dedicated to **OS**:
  - IQRF **communication** (RF and SPI) is packet oriented therefore buffer servicing is supported. Specialized OS functions are available for comfortable buffer clearing and buffer to buffer data copying. There are four basic buffers primarily dedicated to communications and block operations.
    - **bufferRF** (0x4A0 – 0x4EF), 128 B – for RF communication. Functions RFTXpacket and RFRXpacket works with lower 64 B of bufferRF only.
    - **bufferCOM** (0x3A0 – 0x3DF), 64 B – for serial wired communication (SPI, …).
    - **bufferINFO** (0x320 – 0x35F), 64 B – for OS and user block operations

    These communication buffers are especially intended for transferred data and not for other user data. Additionally, OS uses them as work buffers in some cases. E.g. bufferINFO can be modified in some cases during RFRXpacket() and bufferCOM can be destroeyd in background by SPI master if SPI is not disabled. See also side effects in IQRF OS Reference guide [1].

    Tip: If there is a need to avoid possible bufferCOM destroying in background, SPI must not be active that time.
    - **bufferAUX** (0x420 – 0x45F), 64 B – auxiliary buffer, to store bufferINFO temporarily (by function **swapBufferINFO**) to make it free for other operations.
  - Array **networkInfo** (0x2A0 – 0x2BF), 32 B is an area dedicated to network system information, partly accessible to the user in accordance with IQMESH specification.
  - System variables
    - Some of them can be modified by the user (toutRF, userStatus. …)
    - Some of them are read only (RFchannel, …)
    - Others are not documented, the user need not access them
  - OS work variables (not documented, the user need not access them)
- RAM available for the **user**:
  - Areas 0x5C0 – 0x5EF (48 B) in RAM bank 11 and 0x620 – 0x64F (48 B) in RAM bank 12. They can be accessible in a single block up to 96 B using linear data memory access ranging 0x2390 to 0x23EF (see above). Preselection of bank 11 is automatically arranged by OS. All OS functions return with the bank 11 preselected.
  - Additionally, two userReg registers (0x70 and 0x71) are available in the Common area shared for all banks.

Mapping of GPR locations available by OS:

Bank	Address		Space
	Linear	Traditional	
11	2390	5C0	user 48 B
	23BF	5EF	
12	23C0	620	user 48 B
	23EF	64F	

See Appendix 1, RAM map.

Tip: To minimize code length due to bank switching by the compiler implied from access to user RAM, it is recommended:

- Map all individual variables (needed to be accessed in random way) to bank 11.
  The compiler arranges it automatically. Default mapping of user variables starts from the beginning of user part of bank 11 and bank 12 is used not until bank 11 is full, unless otherwise stated.
- Map possible arrays, user buffers etc. to bank 12. Example:

```
uns8 i, j;                // This will be mapped in bank 11
#pragma rambank = 12
uns8 xx[37];              // This will be mapped in bank 12
#pragma rambank = 11
uns16 x, y;               // This will be mapped in bank 11
```

- Access bank 12 just by indirect addressing (see below) using dedicated OS functions writeToRAM, readFromRAM, and copyMemoryBlock.
- Then bank 11 is always kept as the current one in user program. It is automatically arranged by OS (bank 11 is selected after OS boot as well as after finishing of every OS function). Thus, in such arrangement the compiler does not need to switch banks at all.

Indirect RAM access

- Due to security reasons, indirect RAM access using registers `INDFx` is denied (for either traditional or linear addressing). Thus, indexes of arrays are not allowed to be variables. (A[1]=0 is allowed, A[i]=0 is restricted due to using `INDFx` by the C compiler). IQRF IDE issues a warning and OS performs the `reset` instruction in case of such denied access. See IQRF OS Reference Guide [1].
- Instead of this IQRF OS allows to write to indirectly addressed RAM using extra system functions `setINDF0` and `setINDF1` to write and `getINDF0` and `getINDF1` to read. Operation is the same like indirect addressing using `INDF0` and `INDF1` but without the restriction above. Another possibility is using functions `readFromRAM` and `writeToRAM`. See IQRF OS Reference Guide [1] and example E06–RAM [10].

RAM access restrictions

In addition to `INDF0`, `INDF1`, `EECON1` and `EECON2` which are inaccessible at all, from OS v3.02D the following MCU registers are protected against direct writing for security reasons: `TMR0`, `TMR1L`, `OPTION_REG`, `PCON`, `CMOUT`, `BORCON`, `SSPCON1`, `SSPCON2`, `CCP1AS`, `PSTR1CON`, `CCP3AS`, `PSTR3CON`, `IOCBN` and `IOCBF`. They can be directly read but writing is allowed by the `writeToRAM()` function only. For several more frequent registers (e.g. to control BOR and wake-up on pin change) there are macros to make this easier. See the `IQRF-macros.h` header file.

When attempted to access locations in an illegal way, restricted instructions are replaced by the reset instruction.

Tip: Such type of MCU reset can be identified by the `_RI` flag in the `userReg0` register (see chapter Reset).

Caution
RAM dedicated to MCU core, MCU peripherals or OS should be accessed by the user in accordance with MCU datasheet, requirements resulting from hardware construction of the TR module, OS implementation and this manual. Illegal modification of dedicated memory locations can cause system malfunction.

Data memory (EEPROM inside the MCU)

EEPROM data remains stored even after power off. Overwriting is not unlimited, number of erase/write cycles is 1 000 000 min., (typically 10 000 000). EEPROM is especially intended for configuration parameters and data.

Individual parts of internal EEPROM:

- **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 `0xF000-0xF0FF` instead of `0x00-0xFF` when using `cdata` and similar C statements (e.g. `__EEAPPINFO = 0xF0A0`).
- 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 `EECONx` 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.

Data memory (serial EEPROM)

External optional 16 kb EEPROM with serial I2C interface. It must not be utilized by the user in case of network Coordinator. Data is accessible for read and write in 16 B blocks using dedicated OS functions `eeeWriteData` and `eeeReadData`. Refer to the IQRF OS Reference guide [1].
Data remains stored even after power off. Overwriting is not unlimited, number of erase/write cycles is 1 000 000 typically.

Identification

Module identification

Every IQRF module contains information about itself. This is accessible via the `moduleInfo()` function storing data to the `bufferINFO`. See the IQRF OS Reference guide [1], `moduleInfo` for Module data format details.

Module identification is displayed by the IQRF IDE development environment.

Application Info

It is a 32 B block in EEPROM inside the MCU (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 1 as well.

Control

Operation modes

The TR modules can work in four 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 the IQRF Quick start guide [11].
- **RFPGM:** The user program or OS plug-in can be wirelessly uploaded to the TR (including EEPROM content). More TRs can be uploaded at the same time. See Appendix 3 – RFPGM RF Programming ™.
- **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 period (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. Ticks are derived from internal RC oscillator if RF IC is off (e.g. after `setRFsleep()`) and from crystal otherwise (e.g. after `setRFready()`).

Tip: Use `setRFready()` function if you need precise ticks even when you do not use RF right now.

- The 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 also provides functions for waiting on foreground.
- Short time intervals for timing on foreground can be derived also from instruction timing. The MCU is clocked with internal 8 MHz 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. IQMESH timing is derived from RF IC with quartz precision. It is not necessary to calibrate timing from OS v3.01D. Thus, do not use the `calibrateTimer()` function. See the IQRF OS Reference guide [1].

Timer6

Timer6 is a HW peripheral of MCU intended as a programmable timer. It is fully user available from OS v3.02D. It works in background and can also generate an optional MCU interrupt. It cannot be operated while the MCU is in Sleep mode. Refer to the PIC datasheet, chapter *Timer2/4/6 modules* [8] for handling.

Watchdog

To increase the reliability, OS allows to use hardware watchdog of the MCU to recover the system from unexpected events. 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 by the C compiler). If an overflow occurs it is regarded as a program execution failure and the microcontroller responds with reset. If the failure is not a permanent one, it can lead to system recovering. It is up to the user to place `clrwdt()` at proper location(s). 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 is default disabled from OS v3.01D and can be enabled by SW via the `SWDTEN` bit (0 – disabled, 1 – enabled). Overflow period is user selectable (even while the program is running) from 1 ms to 256 s by the `WDTPSx` bits in the `WDTCON` register – see PIC datasheet [8] and the `IQRF-macros.h` header file.

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.

Switching 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. SPI must be disabled if there is a possibility of unintentional wake-up from SPI master. 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. Either one or both rising or falling edge can be used.
  - To enable, the following sequence is required (see also the `IQRF-macros.h` header file):

```
GIE = 0;                              // Global interrupt disabled
writeToRAM(&IOCBN, IOCBN | 0x10);     // Negative edge active. Instead of IOCBN.4 = 1;
                                      // IOCBN cannot be accessed directly due to OS
                                      // restriction.
IOCBP.4 = 1;                          // Positive edge active (clear if not required)
IOCIE = 1;                            // Interrupt on change enabled
SWDTEN = 0;                           // Watchdog disabled
iqrfSleep();                          // Sleep
writeToRAM(&IOCBF, IOCBF & 0xEF);     // Clear interrupt on change flag. Instead of IOCBF.4=0.
                                      // IOCBF cannot be accessed directly due to OS
                                      // restriction.
```

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 see the TR datasheets [7].

Other PIC peripherals

There are PIC HW resources (I^2C, UART, PWM output, A/D and D/A converters, analog comparator, …) user accessible but not supported by the OS. Refer to datasheet of given TR module [7], the microcontroller [8] and application examples [10]. Due to pin sharing with more peripherals possible conflicts must be avoided by the user. E.g. TR-54D D/A converter must not be used at the same time with OS functions for red LED.

Reset

Initialization of MCU, OS and application. The following reset types available:

- **Power-on reset (POR)**: Utilizes HW filter to eliminate improper power-up rising and spikes to some extent. A lot of applications need no external reset circuitry.
- **Brown-out reset (BOR):** If power supply falls below given level for given time the reset is invoked. This option is default disabled and can be enabled by user SW. BOR should be disabled in Sleep otherwise additional 7.5 µA is consumed. Direct writing to Brown-out control register `BORCON` is restricted (see above). Thus, `SBOREN = …` is not allowed and macros `setBORon()` and `setBORoff()` should be used instead.
  Tip: If possible problems with unstable power supply may be expected, it is recommended to call `setBORon()` as the first command in the application, otherwise unpredictable behavior of TR module can occur.
- **Watchdog (WDT) reset:** After WDT time-out.
- **Reset instruction:** Reset can also be invoked by SW by the `reset()` command. It is not an OS function but MCU machine instruction accessible due to a macro in the `IQRF-macros.h` header file.
  Example:

```
if (Error == 1)
  reset();
```

- **Stack overflow / underflow reset:** After MCU Stack overflow / underflow.

Determining the cause of a Reset

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

Bit	7	6	5	4	3	2	1	0
Status flags	**_STKOVF**	**_STKUNF**	**_RMCLR**	**_TO**	**_PD**	**_RI**	**_POR**	**_BOR**
Power on reset	0	0	1	1	1	1	0	x
Brown-out reset	0	0	1	1	1	1	u	0
WDT reset	u	u	u	0	u	u	u	u
WDT Wake-up from Sleep	u	u	u	0	0	u	u	u
Interrupt Wake-up from Sleep	u	u	u	1	0	u	u	u
Reset instruction executed	u	u	u	u	u	0	u	u
Stack overflow reset	1	u	u	u	u	u	u	u
Stack underflow reset	u	1	u	u	u	u	u	u

u = unchanged, x = undefined

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

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

_RI Reset instruction flag
- _RI = 0 After `Reset` instruction executed
- _RI = 1 `Reset` instruction has not been executed

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

_TO Watchdog time-out flag
- _TO = 0 after WDT overflow
- _TO = 1 after power-up, clrwdt or iqrfSleep

_RMCLR –MCLR reset flag
- _RMCLR = 0 after –MCLR reset
- _RMCLR = 1 –MCLR reset has not occured

_STKUNF Stack underflow flag
- _STKUNF = 1 after Stack underflow
- _STKUNF = 0 Stack underflow has not occured

_STKOVF Stack underflow flag
- _STKOVF = 1 after Stack overflow
- _STKOVF = 0 Stack overflow has not occured

To fully utilize all these features, reset flags should be properly handled in SW after respective event occurred. Refer to the PIC datasheet [8], (`STATUS` and `PCON` registers and Reset) for details.

Content of RAM registers after resets is strictly defined (set / cleared / not affected / unknown). It partly depends on reset type. Refer to MCU datasheet [8]. Additionally, OS completely clears the user memory area except of the `userStatus` register (address `0x5AD`) 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.
- You can also use this register as a counter or timer to reveal the cause of the reset.

Interrupt

MCU offers a sophisticated interrupt system supporting various asynchronous events allowed to interrupt the running program, call specified procedure and then continue from the previous location. In brief, all interrupts have the same handling: it can be enabled / disabled by the flag …E and generates a request for interrupt by the flag …F. Additionaly, all interrupts can be enabled / disabled by the flag GIE (global interrupt enable). In fact, due to various peripherals involved, the interrupt system is slightly more complex. That is why refer to the PIC datasheet [8].

OS and IQMESH use the interrupt system to generate system ticks, communication control etc. Additionally, interrupt can also be used by the user from OS v3.02D.

Disabling interrupt

Interrupt can be disabled by the user (`GIE = 0;`) to gain control over precise timing, e.g. to generate a signal with desired waveform. To avoid OS malfunction, disabling is allowed for very short period only (depending especially on communication in the application).

User interrupt

Proper timing is fundamental for correct OS functionality, that is why the user should use interrupt very considerately and in well reasoned cases only.

Specified user routine can be called whenever an interrupt occurs. The following rules must be kept:

- User interrupt must be enabled (`_enableUserInterrupt = 1`) when desired and disabled (`_enableUserInterrupt = 0`) when not desired. Default OS condition is disabled.
- The user interrupt routine must start from address `0x3F00`.
- The user interrupt routine must take less than 50 µs.
- User interrupts must not be called in period shorter then 800 µs.

If enabled, user interrupt routine is invoked within any interrupt handling. If the interrupt is caused by OS, the user routine is invoked after all OS interrupt handling is finished. Thus, timing accuracy of user interrupt is limited (the order of magnitude of possible difference is 1 ms). Among others, OS generates an interrupt at least every 10 ms to generate system tick. There are more sources able to generate an interrupt. For TR modules especially interrupt on pin change (for fast response to external events) and interrupt on Timer6 overflow (for user timing) are predetermined for wide usage.

See `E013-INTERRUPT` example [10] .

Example using Timer6:

```
…
TMR6IE = 1;                    // Enable interrupt from Timer6
_enableUserInterrupt = 1;
…

#pragma origin 0x3F00          // User interrupt routine must start here
void userIntRoutine()
{
    if (TMR6IF) {              // Ignore all other interrupt causes
      _LEDG = 1;               // LED on when Timer6 overflowed
      TMR6IF = 0;              // Clear to reinitialize Timer6 for next interrupt
    }
}
```

Temperature measurement

Temperature is measured by precise optional digital on-board sensor TMP112 (TI) with ~0.0625 ºC resolution and 0.5 ºC accuracy in 12 b data format connected to MCU via the I2C bus. See `E08-TEMPERATURE` example [10] and IQRF OS Reference guide [1].

Battery check

Supply voltage can be measured by the `getsupplyVoltage()` function. See IQRF OS Reference guide [1].

LED indication

Two on-board LEDs (red and green) can be served by the set of specialized functions (e.g. `pulseLEDR()`) running in OS background or direclty by handling the appropriate pins (e.g. `_LEDR = 1`). But both these approaches should not be combined each other. Directly changed pin levels can be modified in background by OS LED functions.

SPI

Standard serial 4-wire bus in slave mode running in OS background. See separate document *SPI implementation in IQRF TR modules* [5]. SPI is widely used also by the IQRF IDE.

Debug

The IQRF platform provides user with an efficient debugging tool. To use it, the following configuration should be used: The transceiver module plugged e.g. in 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.

Debug utilizes SPI to communicate with TR module. Thus, SPI pins must be configured as follows: C5, C6, C7 as inputs and C8 as output. This is arranged by OS after reset. If changed by the user, this state must be restored before calling the `debug()` function.

Possible user circuitry conected to SPI pins must avoid conflicts otherwise failure or damage can occur.

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:

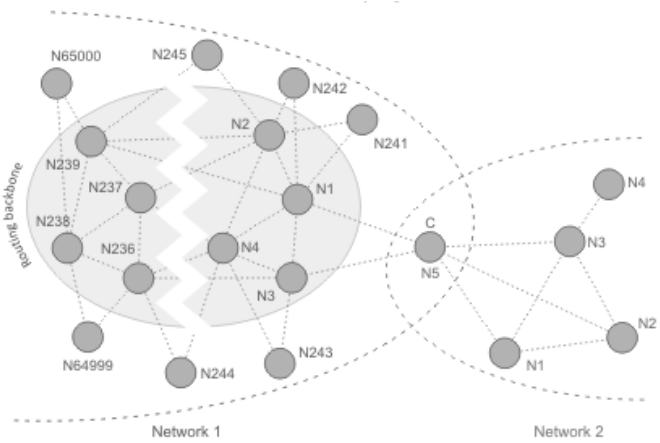
- **Peer-to-peer:** Two or more peer-to-peer devices, without a network Coordinator. Packets are available for all devices in range and completely managed by the user program. Number of devices is unlimited. Keep `PIN=0` (see below) in this mode. This is the default mode.
- **IQMESH:** Topology with 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 ("networking").

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 (~40 ms in LP RX or ~500 ms in XLP RX, see below, Low power modes). Default value is 50 (500 ms in STD mode, 2 s in LP or 30 s in XLP).

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 programmed 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. See below (Filtering).



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 and the `networkINFO` array.

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.

<table>
<tr><td>PIN</td><td>DLEN</td><td>CRCH</td><td>Network info</td><td>CRCN</td><td>User data</td><td>CRCD</td><td>CRCS</td></tr>
<tr><td colspan="3">PAH</td><td colspan="2">NTWINFO</td><td colspan="2">DATA</td><td>CRC</td></tr>
</table>

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 address 0x2A0

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 Networking mode (See above, chapter *RF overview*)

_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 (low byte of Node address in case of 2 B user addressing. High byte see DFM2B.)
- 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 DPA (Direct Peripheral Addressing)

User data

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

User interface

Some information about curent system, RF and network parameters is available in the userInterface register (address 0x7F). Bits marked with “R/W“ can be changed by the user. Bits marked with “R“ are **read only**. Changing of them is allowed via respective OS functions only.

address 0x07F

7	6	5	4	3	2	1	0
R	R	R	R			R/W	
_networkingMode	_networkTwo	_filterCurrentNetwork	_916MHz	-	-	_enableUserInterrupt	-

- _networkingMode
  - 0: Peer-to-peer (non-networking) topology. Relates to the setNonetMode function.
  - 1: IQMESH selected. Relates to the setCoordinatorMode and setNodeMode functions.
- _networkTwo
  - 0: Network 1 selected. Cleared by the setCoordinatorMode function.
  - 1: Network 2 selected. Relates to the setNodeMode function.
- _filterCurrentNetwork
  - 0: Network filtering disabled. Relates to the setNetworkFilteringOff function.
  - 1: Network filtering enabled. Relates to the setNetworkFilteringOn function.
- _916MHz
  - 0: RF band 868 MHz selected. Corresponds to the setRFband(0) function.
  - 1: RF band 916 MHz selected. Corresponds to the setRFband(1) function.
- _enableUserInterrupt
  - 0: User interrupt disabled.
  - 1: User interrupt enabled.

Other RF and network parameters

Other parameters are available in RAM registers strarting from `0x5A0`, see Appendix 1 (OS, RF and network parameters). Writing is allowed to the registers marked with R/W, the others are read only. All registers with names starting with `ntw` (addresses `0x5A0` to `0x5AA`) are updated after calling the `getNetworkParams()` function only (e.g. `ntwADDR`).

The `_disabledRouting` flag (`ntwCFG.2`) relates to the `setRoutingOff()` and `setRoutingOn()` functions and is valid after calling the `getNetworkParams` function.

RF IC modes

RF IC can operate in four modes. Power consumptions mentioned here are just for guidance and relate to RF IC only.

RF Sleep <1 µA

RF IC is completely switched off (while the MCU can continue running). OS system clock (tick) is derived from MCU internal RC oscillator. This mode is established after reset of the TR module or after `iqrfSleep` or `setRFsleep`.

RF Ready ~600 µA

RF IC is switched on and crystal oscillator is running. OS system clock (tick) is derived from from the crystal oscillator. RF receiver is not active (RX chain is off). This mode is established after `setRFready()`, `checkRF(x)`, `RFRXpacket()`, `RFTXpacket()` or `getSupplyVoltage()`. Switching from RF Sleep to RF Ready takes 2 ms typ., 7 ms max. Thus, `RFTXpacket()` is prolonged by 7 ms when called in RF Sleep mode.

RF Receiving ~12 mA

RF receiver is active (RX chain is on). RF IC works in this mode during `checkRF(x)` or `RFRXpacket()`. After finishing, RF IC is switched to RF Ready unless the `_STAYRX` flag (`setRFmode(_STAYRX)`) is set or `checkRF(x)` returns true. In these cases RF IC stays still in RF Receiving.

Tip: The `setRFRXchain()` macro is intended for fast RX chain start.

RF Transmiting up to 23 mA

RF transmitter is active. RF IC works in this mode during `RFTXpacket()`. After finishing, RF IC is switched to RF Ready or RF Receiving according to the `_STAYRX` flag (`setRFmode(_STAYRX)`) .

Code example illustrating consumption in RF IC modes

Consumptions mentioned here are just for guidance, relate to complete TR module and depend on the TR type.

```
reset();          // MCU running, RF Sleep, ~1 mA total
waitDelay(255);
setRFready();     // RF Ready, ~1.6 mA
waitDelay(255);
setRFRXchain();   // RF Receiving, ~13 mA
waitDelay(255);
iqrfSleep;        // MCU Sleep, RF Sleep, ~1 µA
```

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 with respect to a receiver power managing mode.

RF packet propagation time

DLEN	Peer-to-peer (PIN = 0x00) [ms]	Networking without routing (PIN = 0x80) [ms]	Networking with routing (PIN = 0xA0) [ms]
1	10	13	16
10	12	15	18
64	35	39	42

Times indicated above are for brief guidance only, rounded to ms and valid for bit rate 19.2 kb/s

Actual time consumed by sending a packet (`RFTXpacket` duration) can be measured on a TR pin by an oscilloscope:

```
…
setCoordinatorMode();   // Remove for Peer-to-peer
PIN = 0;
DLEN = 10;              // Set required length (payload in bytes)
_C1 = 1;                // Rising edge on pin C1
RFTXpacket();           // Send the packet and wait until finished
_C1 = 0;                // Falling edge on pin C1
…
```

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 case if the *Wait packet end* option is enabled.. The user has full control on timing as the timeout can be set (in ticks ~10 ms in STD RX mode or in cycles in LP and XLP RX modes, see below, *Low power modes*) prior to the `RFRXpacket()` (by `toutRF = ...`).

Resulting `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 (in case of IQMESH) all other networking information.

To minimize 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].

Low power modes

RX and corresponding TX modes

STD RX

- Continuous receiving during `RFRXpacket`
- `toutRF` is in 10 ms ticks
- ~13 mA power consumption

STD TX

- 3 ms preamble

LP RX

- TR module is periodically switched between receiving (for a short period) and sleep in `RFRXpacket`. This defines a cycle = ~46 ms.
- `toutRF` is in cycles (see above)
- 330 µA average power consumption

LP TX

- 50 ms preamble

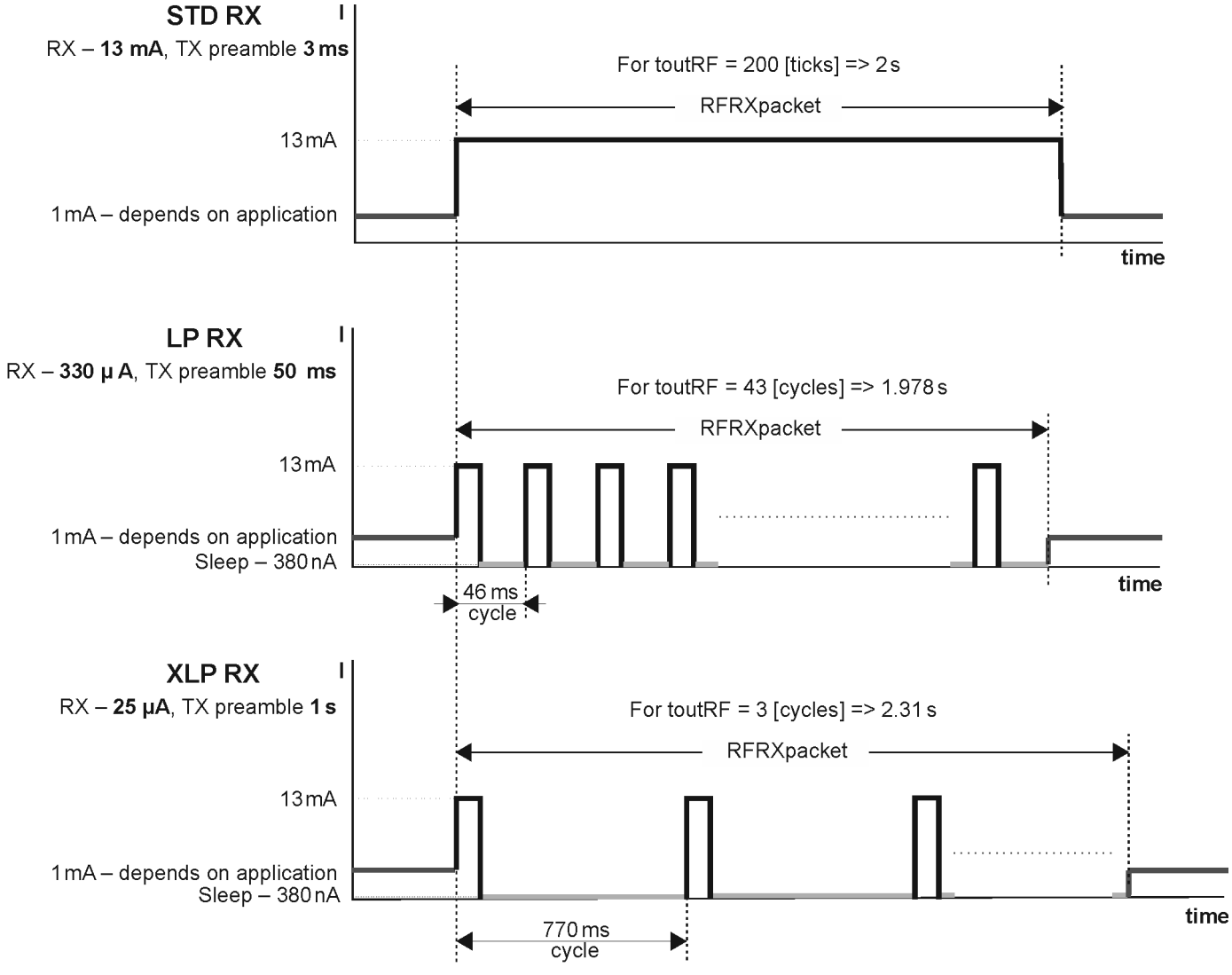
XLP RX

- TR module is periodically switched between receiving (for a short period) and sleep in `RFRXpacket`. This defines a cycle = ~770 ms.
- `toutRF` is in cycles (see above)
- 25 µA average power consumption

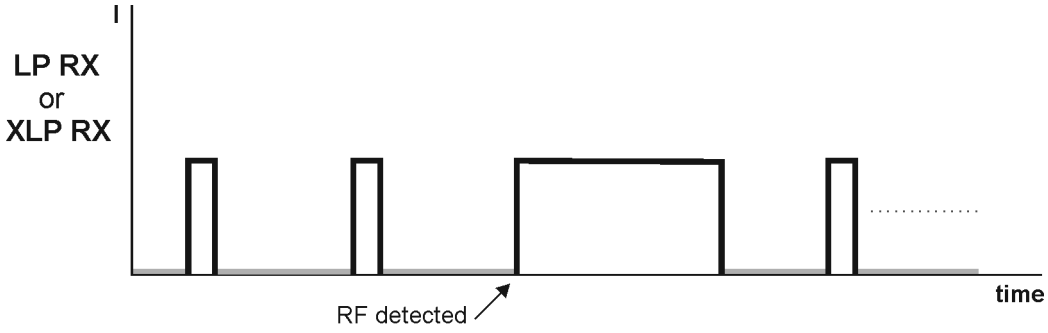
XLP TX

- 1 s preamble

Waiting for incoming RF packets in STD, LP and XLP modes



Response to incoming RF signal in LP RX or XLP RX



Thus, the following consequences are clear from above regarding RX modes:

- The only difference between LP and XLP is the sleep time duration (cycle period)
- TR module saves power only in the `RFRXpacket` function
- The checkRF function must not be used in LP/XLP
- Using of LP/XLP decreases RF range of 20%. This corresponds to checkRF(5).
- Using LP/XLP in noise environment increrases power consumption (TR tries to receive a packet even when it is a noise).
- `setRFmode` offers a possibility to set stronger RF filter to increase immunity against a noise (but the range will be shorter)
- When the receiver is in LP/XLP mode the transmiter must prolong the preamble according to the RX mode. It is arranged automatically but the user must specify this in the `setRFmode` function.
- RX and TX modes can be combined in user application as needed. For example, a TR can receive in XLP RX and transmit in STD TX. It depends on the application and endurance of batteries.
- Bonding, discovery, RF programming can be done only in STD mode. That is why the IQMESH-XLP demo example uses two RX loops – one for service (STD) and the other for normal operation (XLP).

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.

IQRF routing is based on directional flooding of the network and TDMA (Time division multiple access) [12]. This IQRF OS supports up to 239 routing devices for a network. Routing is separated from addressing and is transparent from the user's point of view. There are several routing algorithms specified in the `RTDEF` register according to network topology. A 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 routing devices.

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. Both Discovery as well as AnswerSystemPacket must be performed in STD RX modes only.

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 the 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 is assumed that 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.
- Not all nodes must be assigned to be routers.
- 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:

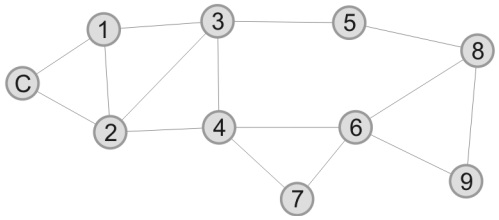
- `RTDT0`: number of hops per packet (0 – 239). 0 means direct delivery (without routing), e.g. 2 means 3 packets sent (sender + 2 routers). The user can set number of hops according to specific needs. But the maximal reasonable value is the number of routing nodes in the network. If an RF packet is 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, see example `IQMESH-XLP` [10]).
- `RTDT2`: For OS only
- `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)**



SMF algorithm is intended for cases where the sender knows the network topoplogy (the paths to the addressee). 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). There is no reason to perform Discovery in this case.

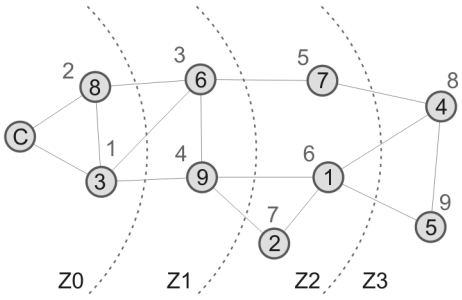
If the logical address of a node is greater than number of hops specified in RTDT0 of given packet, it will not be routed by this node. Thus, a node can be excluded from the routing structure by assigning the logical address out of the address space dedicated to routers. In SFM, the `setRoutingOff()` function does not exclude the node from routing. Excluded nodes can be placed in random way but they must be in range with some router(s).

Setting of number of hops (`RTDT0`):

- For a packet from the coordinator to a node: `RTDT0 =` number of bonded Nodes
- For a packet from the node to the coordinator (answer):
  - For routing nodes: `RTDT0 =` logical address of the sender
  - For non-routing nodes: `RTDT0 =` the highest logical address of all routers in the network.

Example:
A network containing 100 nodes. According to given topology, just 20 routers are enough for sufficiently robust coverage. These routers are bonded with logical addresses 1 to 20 and placed in ascending order from the coordinator. The other nodes are bonded with addresses 41 to 120 and placed in random way. (Therefore, there are 20 free positions in address space of routers reserved for possible increasing of number of routers in the future.) For answer, every router has set `RTDT0 =` its'own logical address and all non-routing nodes `RTDT0 = 20`.

- **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. See the picture.

To disable routing for a node:

- Call the `setRoutingOff()` function. Then no VRN will be assigned to this node during Discovery.
- Do not call the `answerSystemPacket` function.

See the IQMESH-DFM2B-N example [10].

Setting of number of hops (`RTDT0`):

- For a packet from Coordinator to Node: `RTDT0 =` number of bonded Nodes (flooding)
- For a packet from Node to Coordinator (answer):
  - For routing nodes: `RTDT0 =` VRN of given node, or even better `RTDT0 = 0xFF` (OS sets the VRN automatically)
  - For non-routing nodes: `RTDT0 = 0xFF`. Then OS automatically replaces this value with `RTDT0 =` VRN of the node (or possibly the coordinator) which the packet has been received from.
- **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.

  The sender's address is available in following registers of the Coordinator:
  - `RTDT3` High byte of user address
  - `RX` Low byte of user address
  - `TX` Logical address (1 - 240 or 0xFE)

  Unlike SFM and DFM, addresses route packets in DFM2B routing.

- **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.

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. Bonding must be performed in STD RX mode only.

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.

- `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.

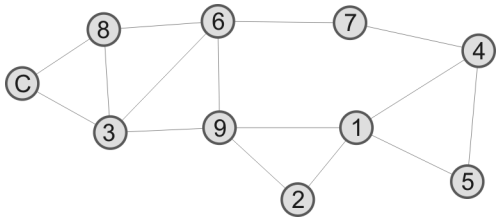
Static systems that suit IQRF best have moderate requirements for changes in bonding.

IQMESH in practise

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.

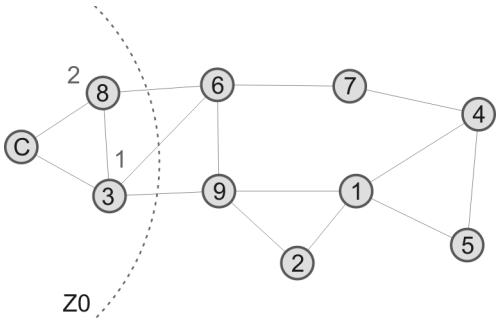


Sending a packet to Node 5
- C sends a packet. It is received by nodes N3 and N8.
- N3 waits 2 slots and then (in the third one) resends the packet.
- N8 waits 7 slots and then (in the 8th one) resends the packet.
- In slots 1 and 2 no packets are resent because nodes N1 and N2 have not received the 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 27, **SFM**). In such cases the SFM routing algorithm (without Discovery) can be used.

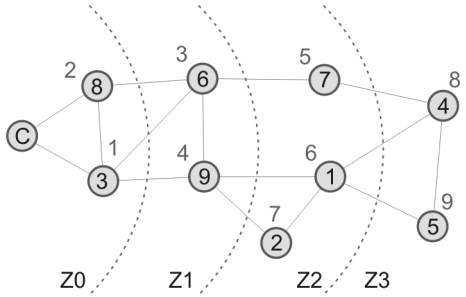
In example above it is necessary to use the **DFM** routing. To utilize DFM, an additional step (Discovery) must be performed. From the source code point of view it means that to call the `discovery(z)` function and wait for the result returning number of discovered nodes. Duration takes some time (it can vary 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 [45]. 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 26.
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
- `setRoutingOn()` and `setRoutingOff()` are intended for DFM routing algorithms only.
- There is no reason to call 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 26.
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 – Memory maps

RAM map (PIC16LF1983)

Structure of each bank

Address offset	Memory region
00 0B	Core
0C 1F	SFR
20 6F	GPR
70 7F	Common

Common registers

Addr	Register	Access	Description
70	userReg0	R/W	User
71	userReg1	R/W	User
72	RFmodeByte	R	Current RF mode (set by setRFmode)
73	param2	R/W	Parameter for OS functions
74 75	param3	R/W	Parameter for OS functions
76 77	param4	R/W	Parameter for OS functions
78		–	
79		–	
7A		–	
7B		–	
7C		–	
7D		–	
7E		–	
7F	userInterface	R/W	Some information about current system, RF and network parameters

R/W – *read/write* R – *read only*

User registers

Bank	Address		Space
	Linear	Traditional	
11	2390	5C0	user 48 B
	23BF	5EF	
12	23C0	620	user 48 B
	23EF	64F	

Communication buffers

Bank	Address		Buffer
	Direct	Linear	
6	320	21E0	bufferINFO 64 B
	35F	221F	

Bank	Address		Buffer
	Direct	Linear	
9	4A0	22D0	bufferRF 128 B
	4EF	231F	
10	520	2320	
	54F	234F	

7	3A0	2230	bufferCOM 64 B
	3DF	226F	

8	420	2280	bufferAUX 64 B
	45F	22BF	

OS, RF and network parameters

Addr	Register	User access	Relates to	Description
5A0	ntwADDR	R	Bonding	Logical Node address
5A1	ntwVRN	R	Discovery	VRN
5A2	ntwZIN	R	Discovery	Zone index (Zone number + 1) E.g. for nodes in direct range from coordinator ntwZIN==1
5A4	ntwPVRN	R	Discovery	Parent VRN
5A5	ntwUSERADDRESS	R	setUserAddress	2 B user address
5A6				
5A7	ntwID	R	Coordinator ID	Network identification – NID0
5A8		R		Network identification – NID1
5A9	ntwVRNFNZ	R	Discovery	VRN of first Node in given zone
5AA	ntwCFG	R	setRoutingon/off,…	Network configuration
5AB	memoryOffsetFrom	R/W	Buffers	Offsets for buffer copy functions. See IQRF reference guide.
5AC	memoryOffsetTo	R/W		
5AD	userStatus	R/W	User variable	Not cleared after reset (except of power on)
5AE	toutRF	R/W	RFRXpacket	Timeot for RFRXpacket() duration
5AF	RFspeed	R	setRFspeed	Current RF speed
5B0	RFpower	R	setRFpower	Current RF power
5B1	RFchannel	R	setRFchannel	Current RF channel
5B2	SPIpacketLength	R	SPI master	SPI packet length
5B6	lastRSSI	R/W	RFRXpacket	RSSI of last receipt

R/W – *read/write* R – *read only*

Network INFO

Addr	Register	Access	Description
2A0	PIN	R/W	Packet information
2A1	DLEN	R/W	Data length
2A2		–	
2A3	RX	R/W	Addressee
2A4	TX	R	Sender
2A5		–	
2A6		–	
2A7	PID	R/W	Packet identification
2A8	RTOTX	–	For OS only
2A9	RTDEF	R/W	Routing algorithm
2AA	RTDT0	R/W	Routing data
2AB	RTDT1	R/W	
2AC	RTDT2	R/W	
2AD	RTDT3	R/W	
2AE	MPRW0	–	Reserved for Direct peripheral access (HWP)
2AF	MPRW1	–	
2B0	MPRW2	–	

R/W – *read/write* R – *read only*

EEPROM map (inside the MCU, PIC16LF1983)

00	Available for Node only. Do not use for Coordinator.	Bonded Nodes *	40	Available for Node only. Do not use for Coordinator.		80	Available for Node only. Do not use for Coordinator.		C0	Reserved by operating system. Do not use at all.	
01			41			81			C1		
02			42			82			C2		
03			43			83			C3		
04			44			84			C4		
05			45			85			C5		
06			46			86			C6		
07			47			87			C7		
08			48			88			C8		
09			49			89			C9		
0A			4A			8A			CA		
0B			4B			8B			CB		
0C			4C			8C			CC		
0D			4D			8D			CD		
0E			4E			8E			CE		
0F			4F			8F			CF		
10			50			90			D0		
11			51			91			D1		
12			52			92			D2		
13			53			93			D3		
14			54			94			D4		
15			55			95			D5		
16			56			96			D6		
17			57			97			D7		
18			58			98			D8		
19			59			99			D9		
1A			5A			9A			DA		
1B			5B			9B			DB		
1C			5C			9C			DC		
1D			5D			9D			DD		
1E			5E			9E			DE		
1F			5F			9F			DF		
20			60			A0	Application Info, 32 B		E0		
21			61			A1			E1		
22			62			A2			E2		
23			63			A3			E3		
24			64			A4			E4		
25			65			A5			E5		
26			66			A6			E6		
27			67			A7			E7		
28			68			A8			E8		
29			69			A9			E9		
2A			6A			AA			EA		
2B			6B			AB			EB		
2C			6C			AC			EC		
2D			6D			AD			ED		
2E			6E			AE			EE		
2F			6F			AF			EF		
30			70			B0			F0		
31			71			B1			F1		
32			72			B2			F2		
33			73			B3			F3		
34			74			B4			F4		
35			75			B5			F5		
36			76			B6			F6		
37			77			B7			F7		
38			78			B8			F8		
39			79			B9			F9		
3A			7A			BA			FA		
3B			7B			BB			FB		
3C			7C			BC			FC		
3D			7D			BD			FD		
3E			7E			BE			FE		
3F			7F			BF			FF		

** – For Coordinator only*

Appendix 2 – Channel maps

433 MHz band channel map

Channel	Frequency [MHz]
0	433.12
1	433.22
2	433.32
3	433.42
4	433.52
5	433.62
6	433.72
7	433.82
8	**433.92**
9	434.02
10	434.12
11	434.22
12	434.32
13	434.42
14	434.52
15	434.62

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

Appendix 3 – RFPGM - RF Programming™

The Lite version

IQRF TR modules with operating system v2.11 or higher can be uploaded even in a wireless way (RFPGM – RF Programming™). During RF upload the programmed TR is not inserted in the programmer but it is connected to the programmer in a wireless way by an auxiliary TR module. This Appendix describes the Lite RFPGM version for IQRF OS v3.01D or highers and IQRF IDE 4.

RFPGM allows:
- Upload TRs also in final application boards (housed, soldered etc.)
- More TRs uploaded simultaneously (e.g. all IQMESH Nodes in one stroke)

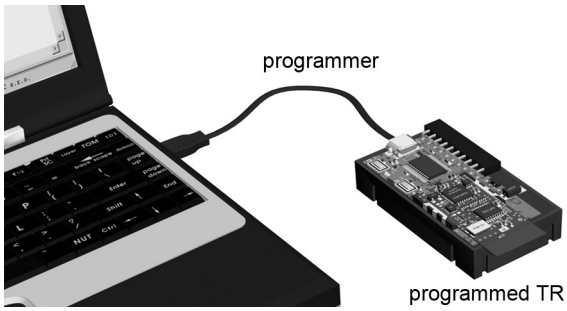


Fig. 1: Standard (wired) upload in a programmer

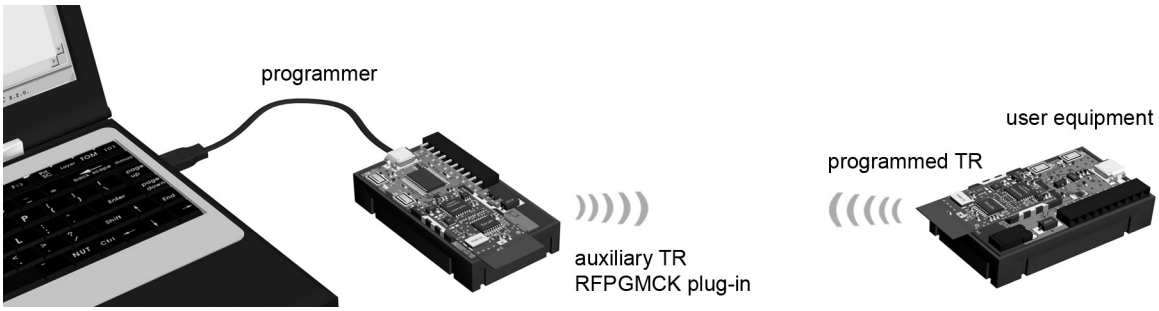


Fig. 2: Wireless upload (RFPGM)

The auxiliary TR module has to have the `RFPGMCK` plug-in (SW option extending the OS) uploaded in advance. The programmed TR module needs no plug-in for RF PGM Lite because necessary functions are included in OS from v3.01D.

To utilize RFPGM, the *RF Programming* checkbox must be active in IQRF IDE. During wireless upload the programmed TR module is not in the standard programming mode but in the RFPGM one. The standard mode is emulated by the auxiliary TR module in the programmer. From the IQRF IDE point of view all is the same as for standard programming except of the fact that the *Module Info* window and the *Reset TR Module* button in IQRF IDE relates to the auxiliary TR module.

RFPGM functions implemented in OS (see the IQRF Reference guide):

- `runRFPGM()` – switch TR to the RFPGM mode
- `enableRFPGM()` – request to enable switching TR to the RFPGM mode after every reset
- `disableRFPGM()` – request to disable switching TR to the RFPGM mode after every reset (factory default)
- `setupRFPGM(x)` – setup RFPGM parameters

RFPGM mode can be entered by reset (if enabled) or by the `runRFPGM()` function (unconditionally) and abandoned by low level on the C5 pin for at least ~0.7 s (if enabled), ~1 minute after reset (if enabled), if RFPGM is not in progress or by the *End RFPGM* button (unconditionally). RFPGM must be performed in STD RX mode only.
- If RFPGM was invoked by reset based on the `enableRFPGM()` function the original application is run from beginning.
- If RFPGM was invoked by the `runRFPGM()` function the application continues program executing.

RFPGM features

- RF band 868 MHz or 916 MHz selected automatically according to the auxiliary TR default
- Fixed channel (the default one, e.i. 52 for 868 MHz or 104 for 916 MHz bands)
- Fixed RF bit rate (19.2 kb/s).
- Fixed RF mode (STD)
- "Non-networking" communication, without addressing and broadcast.

LED indication

Operation	Auxiliary TR		Programmed TR	
	Green	Red	Green	Red
RFPGM mode	short flash in 1 s period	–	continuous	short flashes in 2 s period
Uploading	short flash in 1 s period	fast blinking	continuous	fast flashing
RFPGM mode finished	–	–	–	single flash for 1.5 s

RFPGM usage

Typical requirement used to be a different behavior during development (when modified program is uploaded repeatedly) and after finishing the development (when following-up reprogramming is not desired at all or is required for service purposes on special request only).

The `enableRFPGM()` is very important during development. Special care should especially be taken for applications with TR modules inaccessible for wired upload. E.g. an error in user program under development can cause that the `runRFPGM()` function can not be accessed any more. If the TR can not be reprogrammed externally the only way how to force programming mode is having a possibility to switch to RFPGM automatically after reset. Thus, main purposes of `enableRFPGM()` are:

- To enable entering RFPGM while routine development in a fast and reliable way.
- A loop-hole against user's errors during development.

But it is recommended to keep a way of how to invoke RFPGM mode even in completed and debugged designs on request (at least to allow services and future upgrades). That is why `disableRFPGM()` should not be used unless there is a user's defined way of how to invoke back the RFPGM mode via `runRFPGM()` or `enableRFPGM()`. It is up to the user to define the entry point into RFPGM, e.g. using a jumper, pushbutton or a special RF packet. If being omitted there is no any other way of getting back into RFPGM unless removing the module from a device and placing it into the programmer.

Thus, RF upload is fully under the user's control. Invoking the RFPGM mode, its termination, behavior after reset – everything can be adapted according to the user's needs.

Example

Wireless upload procedure can be tested using the demo program `E12-RFPGM-TST`:

1. **Preparatory phase:** Upload the `RFPGMCK` plug-in to the auxiliary TR module (not in a wireless way).
2. In case of the 916 MHz band select this in the `E12-RFPGM-TST` demo program and recompile it.
3. Insert the programmed TR to the programmer and upload (not in a wireless way) the `E12-RFPGM-TST` demo program. Then:
   - The application just starts up which is indicated by red LED flashing.
   - But thanks to the `enableRFPGM()` function used in the demo program the programmed TR is configured so that it will be switched to RFPGM after reset from this time.
4. **Development phase:** Arrange both TR modules according to Fig. 2 for RFPGM upload and activate the *RF Programming* checkbox.
5. For test purpose change LED flashing from the red LED to the green one in user program and compile this.
6. Set the programmed TR module in RFPGM mode (by the TR module reset) and invoke upload in the IDE environment by the *Upload* button (or by the *F5 key*). After RFPGM finish the red LED flashes for 1.5 s and then the application starts up automatically. (RFPGM mode can be terminated without RF uploading by the pushbutton on the kit by pressing for ~0.7 s.)
7. Repeat steps 5 and 6 until the application is debugged. Respective LED flashing indicates success.
8. **Final upload:** Upload final application with `disableRFPGM()`. After finishing the RFPGM mode is invoked for the last time. Press the pushbutton for more than 0.7 s to terminate this. The application starts to run, executes the `disableRFPGM()` function and automatical entering RFPGM after following resets is disabled.

Since now RFPGM is accessible via the pushbutton only. It makes **RFPGM upgrades** possible by `runRFPGM()` later on.

Removing the RFPGMCK plug-in

The RFPGMCK plug-in can be removed by uploading any user code into the TR module. But it can not be done just by the Upload key (*F5)* in IQRF IDE but the *Enter Programming Mode* (F6) button must be clicked first.

Tip
IQRF IDE supports funtions *Create RF Programmer* and *Remove RF Programmer*. See IQRF IDE Help, menu *Programming → RF Programmer*.

Appendix 4 – Migration from OS v3.02D

Features	OS v3.02D	OS v3.03D
Function `getRSSI`	Not implemented	Implemented
DPA network visualization	Not supported	Supported
Function `RFRXpacket` in LP and XLP modes	A/D stays on after termination	A/D is switched off after termination
RFPGM termination 1 s after reset	Even if RFPGM in progress	If RFPGM is not in progress
`NID` after `removeBond`	Not cleared	Cleared
Sender's address availability for Coordinator in DFM2B	Not available in OS registers	Available in OS registers
IQRF IDE support	v4.00 or higher	v4.11 or higher

Documentation and information

1 **IQRF** OS Reference guide www.iqrf.org/156
2 **Memory maps**, this document, Appendix [1]
3 **IQRF** home page www.iqrf.org
4 **IQMESH** specification www.iqmesh.org
5 **SPI** specification www.iqrf.org/85
6 **IQRF support** support@iqrf.org
7 **TR-52D** datasheet: www.iqrf.org/213
TR-54D datasheet: www.iqrf.org/220
TR-55D datasheet: www.iqrf.org/239
TR-56D datasheet: www.iqrf.org/278
8 **PIC16LF1938** datasheet: www.iqrf.org/214
9 **IQRF IDE:** www.iqrf.org/86
10 Examples (included in the StartUp Package): www.iqrf.org/112
11 **IQRF Quick start guide**: www.iqrf.org/quickstart
12 **IQMESH, Technology for Wireless Mesh Networks**:
www.thinkmind.org/download.php?articleid=icns_2012_4_40_10199

If you need a help or more information please contact IQRF support [6]. A lot of information is also available in the IQRF OS Reference guide [1] and on the IQRF web site [3].

Document revision

- 130501 First release for OS v3.03D. Updated for TR-55D, TR-56D and 433 MHz TR modules.
Extended description of routing algorithms.

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 and MICRORISC name are registered trademarks of MICRORISC s.r.o.
PIC, SPI, Microchip 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**
Support	**support@iqrf.org**

Smarter wireless. Simply.