

# **IQRF OS**

## **Operating System**

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

## **User's Guide**



**Simple way to smarter wireless solutions**

## Content

Compatibility.....	3
OS Principles.....	4
Concept of OS plug-ins.....	4
IQRF OS Architecture .....	5
RF circuitry.....	6
Microcontroller.....	6
Memories.....	7
Program memory (Flash).....	7
Data memory (RAM).....	7
Data memory (EEPROM).....	8
Identification.....	8
Module data.....	8
Application data.....	8
Control.....	9
Operation modes.....	9
Real time.....	9
Communication control, checking and timeouts.....	9
Watchdog.....	10
TR module Sleep.....	10
RF Sleep.....	10
RX chain.....	10
Other PIC peripherals.....	10
Reset.....	11
Temperature measurement.....	11
Battery check.....	11
LED indication.....	11
Debug.....	11
SPI.....	11
RF.....	12
RF overview.....	12
RF networking.....	13
Filtering.....	14
RF transmitting.....	14
RF receiving.....	14
Routing.....	15
Bonding.....	16
Appendix 1.....	17
EEPROM map.....	17
RAM map (PIC16F886).....	18
Appendix 2.....	19
868 MHz band channel map.....	19
916 MHz band channel map.....	20
Documentation and Information.....	21
Document revision.....	21
Sales and Service.....	22

## Compatibility

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

Communication is possible among TR modules with the same type of modulation only. FSK also supports multiple channels.

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

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

### IQRF OS versions and history:

Version	Main differences	Release	Status
v3.00 for 5xB	<ul style="list-style-type: none"> <li>Up to 65 000 devices and up to 240 hops in IQMESH network</li> <li>New power management - 1 uA in Sleep, 35 uA in XLP RX</li> <li>Discovery, real time transparent routing, low power routers</li> <li>Many other outstanding features</li> </ul>	Jan 2011	<b>current for TR-52B and TR-53B</b>
v2.11 for 5xB	<ul style="list-style-type: none"> <li>RF power management supported (<code>setRFmode</code>, <code>checkRF</code>, ...)</li> <li>RF channels available</li> <li>selectable RF bit rate (provisionally for experimental purpose)</li> </ul>	Mar 2010	not for new designs for <b>TR-52B</b> and <b>TR-53B</b>
v2.10 for 5xB	In addition to that for 3xB: <ul style="list-style-type: none"> <li>868 MHz or 916 MHz <b>band</b> software selectable</li> <li>Enhanced battery check</li> <li>RF IC sleep mode supported</li> </ul>	Jan 2010	for TR-52B and TR-53B not for new designs
v2.10 for 3xB	<ul style="list-style-type: none"> <li>concept of OS <b>plug-ins</b></li> <li>RF power not limited during bonding</li> <li>green LED support, LED functions renamed</li> <li>User <b>RAM limited</b> to 0x1CF</li> </ul>	Dec 2009	<b>current for TR-31B and TR-32B</b>
v2.09	<ul style="list-style-type: none"> <li>minor change in first falling to Sleep mode</li> <li>bonding robustness increased</li> </ul>	Jul 2009	not for new designs
v2.08	<ul style="list-style-type: none"> <li><b>broadcast</b> message support added</li> <li>implemented in <b>TR-31B</b> modules</li> </ul>	Oct 2008	<b>current for TR-11A and TR-21A</b>
		Nov 2008	
v2.07	<ul style="list-style-type: none"> <li>bug in the <code>setLoggingOff()</code> function fixed</li> <li><b>Wake-up on pin change</b> improved. To utilize it, the sequence GIE = 0; RBIE = 1; is required just before <code>iqrfSleep()</code>.</li> </ul>	Sep 2008	not for new designs
v2.06	<ul style="list-style-type: none"> <li>minor change in routing</li> </ul>	Aug 2008	not for new designs
v2.05	<ul style="list-style-type: none"> <li>higher <b>RF noise immunity</b></li> <li>corrected transfer of MPRWx while not routing</li> <li>several minor bugs not affecting module functionality corrected</li> </ul>	Aug 2008	not for new designs
v2.04	<ul style="list-style-type: none"> <li><code>setNetworkFilteringOn()</code> switches just packet from active network (1 or 2), non-networking communication ignored</li> <li><b>Wake-up on pin change</b> under user's control. Default disabled. To enable, set RBIE = 1 before <code>iqrfSleep()</code>. Not compatible with previous versions (permanently enabled in Sleep up to v2.03).</li> </ul>	Jul 2008	internal release only
v2.03	<ul style="list-style-type: none"> <li>BufferCOM size increased from 35B to 41B</li> <li>Number of nodes in one network increased from 128 to 239</li> <li>Minor bug in routing fixed</li> </ul>	Jul 2008	not for new designs
v2.02	<ul style="list-style-type: none"> <li>minor SPI bug fixed</li> </ul>	May 2008	not for new designs
v2.01	<ul style="list-style-type: none"> <li>function <code>wipeBondNR()</code> added</li> <li>function <code>batteryValueOK()</code> added</li> </ul>	Mar 2008	not for new designs
v2.00	<ul style="list-style-type: none"> <li>Much more <b>effective, easier</b> to use, higher <b>performance</b></li> <li><b>Networking</b> totally reworked. Extended capability. <b>Complete IQMESH.</b></li> <li><b>SPI on background</b></li> <li><b>Encoded</b> network communication</li> <li>Indirect RAM access</li> <li>Temperature measurement supported by OS</li> <li>Supports user application debugging directly by IQRF OS</li> <li>Many other improvements</li> <li><b>IDE</b> – complete <b>development environment</b> with all SW tools integrated including effective debug tools</li> </ul>	Jan 2008	not for new designs
v1.14	previous generation	Jul 2007	not for new designs

## OS Principles

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

Patented IQRF transceiver module architecture has two software layers:

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



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

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

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

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

IQRF OS supports communications:

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

Other communications can be realized with a user program (I<sup>2</sup>C, UART, ...).

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

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

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

## Concept of OS plug-ins

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

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

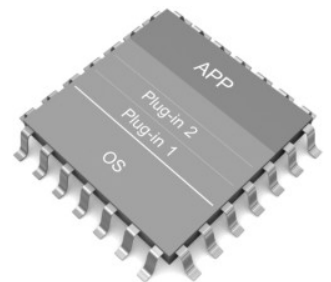
More plug-ins can be used at the same time.

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

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

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

Users are allowed to create their own plug-ins. The IQRF manufacturer offers custom plug-in conversion from `.HEX` to encrypted `.IQRF` format.





## RF circuitry

Main features:

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

## Microcontroller

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

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

PIC HW resources		Utilization
Program memory	Flash	1024 instructions
Data memory	RAM	40 B – user data 172B – communication/system buffers
Data memory	EEPROM	Node: 160 B (+32B application data) Coordinator: 0 B (+32B application data)
I/O pins	TR-52B	6 × I/O
	TR-53B	13 × I/O, 1 × input only
A/D converter (10b)		2 (TR-52B) or 3 (TR-53B) external analog inputs
Serial communication	SPI (slave)	Supported by OS in background
	I <sup>2</sup> C (slave)	Realized by PIC HW module and user function – see Application examples [10]
	UART	Realized by PIC HW module and user function – see Application examples [10]
Interrupt		Not user available. It can be disabled just for a short period if necessary.
Stack (for subroutines, shared with interrupt)		Max. 2 levels of subroutine calling is allowed except of <code>RFTXpacket()</code> and <code>RFRXpacket()</code> (1 level allowed) and <code>bondNewnode()</code> , <code>bondRequest()</code> and <code>answerSystemPacket()</code> (must not be called in subroutines at all).
Power-on reset		Utilizes HW filter to eliminate improper power-up rising and spikes to some extent.
Brown-out reset		Disabled. Can be enabled during operation and disabled in Sleep by the manufacturer on request.
Power-up timer		Disabled
Watchdog		Time-out can be set from 1 ms to 268s or disabled at all by SW. Default is enabled, time-out ~ 4 s. WDT time-out varies with temperature, supply voltage and other conditions from part to part. See PIC datasheet [8].
Oscillator		Internal RC, 8MHz (500ns instruction). Switching to lower clock is allowed but not recommended for keeping correct OS functionality especially during communication. IQMESH timing precision is not limited by precision of this oscillator when calibrated by the <code>calibrateTimer()</code> OS function.
Configuration words ("Fuses")		CONFIG1 = 0x2014, CONFIG2 = 0x38FF

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

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

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

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

## Memories

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

Individual parts of memories are:

- Dedicated to the microcontroller
- Dedicated to the OS
- Other areas are available for the user

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

*Illegal modification of dedicated memory locations can cause system crash.*

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

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

Refer to Appendix 1 - RAM and EEPROM maps.

### Program memory (Flash)

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

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

- OS occupies the memory from 0x0000 to 0x13FF
- Area 0x1400 to 0x1BFF is dedicated for OS plug-ins
- Remaining area 0x1C00 – 0x1FFF (1024 machine instructions) is available for user program .

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

### Data memory (RAM)

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

Individual RAM parts:

- dedicated to the **microcontroller** and its **peripherals** (PIC special function registers – SFRs). Direct using is mostly restricted, e.g. the application need not use registers INDF, EECON1, EECON2, EEADR, PIR2, PIE1 and PIE2. In doubt, refer to IQRF support by the manufacturer [6].
- dedicated to **OS**:
  - IQRF **communication** (RF and SPI) is packet oriented therefore buffer servicing is supported. There are four basic buffers primarily dedicated to communication and block operation:
    - **bufferRF** (0x110 – 0x14F), 64B – for RF communication
    - **bufferCOM** (0xA0 – 0xC8), 41 B – for serial communication (especially SPI). Use 35 B (0xA0 – 0xC2) only, the others are reserved for OS and future compatibility.
    - **bufferINFO** (0x20 – 0x42), 35B – for OS and user block operations
 These communication buffers are especially intended for transferred data but can be used according user's need in specific cases as well. There are specialized OS functions for comfortable buffer to buffer data copying.
    - **bufferAUX** (0x1C0 – 0x1DF), auxiliary buffer to store bufferINFO temporarily for using it for other operations.
  - **buffer networkInfo** (22B) is an area dedicated to network system information.
  - **system variables**. Some of them (toutRF, userStatus etc). can be directly accessed by the user.
  - **OS work variables** (not documented, the user need not modify them).
- Area (0x190 – 0x1B7) is **available for the user** (40 B). It is located in the RAM bank 3. Selection of bank 3 is automatically arranged by the IQRF header files. Additionally, two **userRegx** registers (0x1F0 and 0x1F1) are available in area shared for all banks. If needed, refer to the PIC datasheets [8] for information about RAM banking.

See Appendix (RAM map).

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

## Data memory (EEPROM)

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

Individual EEPROM parts:

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

EEPROM access:

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

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

## Identification

### Module data

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

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

Coordinator / Node: reserved for future OS versions. Coordinator / Node is SW selectable in IQRF this OS version.

- 0: Node
- 1: Coordinator

OS version:

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

PIC type:

- 3: PIC16F886

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

Example (all in hexadecimal):

```

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

```

Meaning: Coordinator, Module ID = 0100101C, IQRF OS version 3.00, PIC16F886, build # 0x0330.

Module ID is displayed by the IQRF IDE development environment.

### Application data

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

Refer to memory maps in Appendix as well.



## Control

### Operation modes

The TR modules can work in three modes:

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

### Real time

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

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

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

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

**Tip:** If there is a gateway in the network, it can be used also as a timebase with crystal precision. Time information can be distributed via RF.

### Communication control, checking and timeouts

Good programming practice requires to have supervision under times for communication establishing and data transfers. It is convenient for synchronization with the transmitter, securing the program against transmission failures, noise reduction, keeping local regulation according to duty cycles etc.

The OS supports the following timeouts and checking:

- **during receiving:** via functions for waiting on background. It is checked whether the requested operation passed during the user setup time. Otherwise attempts for receiving are terminated and receiving function returns control to superordinate program. OS checks the following times:
  - Receiving an RF packet in time (via the `toutRF` variable, if `Wait packet end` not selected)
  - Starting receiving an RF packet in time (via the `toutRF` variable, if `Wait packet end` selected)
  - RF reception can be terminated if signal strength falls under predefined level in RFIM mode (RF Immunity Mode)
  - RF signal strength can be checked which enables to detect incoming packets, select a free channel, FHSS (Frequency-Hopping Spread Spectrum), LBT (Listen Before Talk) etc.
  - Starting receiving via SPI
  - Receiving next word via SPI (i.e. after successfully established communication)
- **during transmitting:**
  - Packet preambles are automatically prolonged for low power RX modes if specified by the user
  - Other control (e.g. repeated packet transmitting with number of repetitions specified in advance) is NOT ruled by the OS but it is fully under user’s handling.
- **during receiving and transmitting:** RF and SPI communication is ensured with check „sums“ CRC. Complete SPI packet (not only „significant data“) is ensured with a single byte CRC, the RF packet with more various CRCs.
- Even higher reliability can be achieved with additional user verification.

Refer to SPI [5] and IQMESH [4] specifications for details.

## Watchdog

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

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

## TR module Sleep

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

### Transition to the Sleep mode:

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

Before switching to the Sleep mode:

- Power consumption should be minimized even for hardware resources of TR controlled by the user (PIC pins, possible PIC internal peripherals) and possible external peripherals connected. It must be done in user program. See the TR [7] and PIC [8] datasheets.
- The microcontroller should be configured for subsequent wake-up on pin change (if required):
  - Wake-up on pin change is under user's control, default disabled.
  - To enable, the sequence `GIE = 0; RBIE = 1; iqrfsleep(); RBIF = 0;` is required.
  - This is not compatible with previous IQRF OS versions. Wake-up on pin change was default disabled by OS up to v2.03, automatically enabled before `iqrfsleep` and automatically disabled after `iqrfsleep`. Now wake-up on pin change is fully under user's control.

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

- after watchdog overflow (non-maskable, but the watchdog can be disabled for TR-31B and higher)
- 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:** all wake-up types can be distinguished via the `-TO` and `-PD` status flags – see PIC datasheets [8].

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

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

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

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

## RF Sleep

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

## RX chain

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

## Other PIC peripherals

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

## Reset

The following reset types available:

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

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

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

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

-TO Watchdog time-out flag

-TO = 0 after reset execution, WDT overflow or `IQRReset()` function.

-TO = 1 after power-up, `clrwdt` or `iqrfsleep`

-PD Power-down flag

-PD = 0 after `iqrfsleep`

-PD = 1 after power-up or `clrwdt`

-POR Power-on reset flag

-POR = 0 after power-on reset (must be set in software then)

-POR = 1 no power-on reset occurred

-BOR Brown-out reset flag

-BOR = 0 after Brown-out reset (must be set in software then)

-BOR = 1 no Brown-out reset occurred

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

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

**Tip:** `userStatus` can be used to help to debug unexpected resets in user programs.

## Temperature measurement

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

## Battery check

For power supply check see the `E10-BATTERY` example [10] and IQRF OS Reference guide [1].

## LED indication

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

## Debug

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

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

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

## SPI

See separate User's guide Implementation in IQRF TR modules.

## 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 implementation:

- 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, ...

Related memory locations and registers:

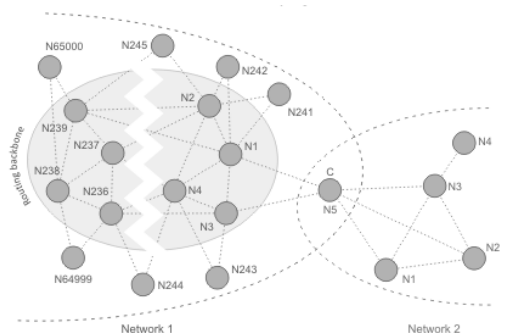
uns8	bufferRF[64]	buffer for RF routines, 64B long
uns8	DLEN	packet length, 0-64 (specify before transmitting, automatically set after receiving)
uns8	toutRF	timeout for packet receiving (in ticks). 1-255, default value is 50 (500 ms).
uns8	PIN	packet information. See below.

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 in this mode. This is the default mode.
- **IQMESH:** Topology with one Coordinator mastering the network and up to 239 end devices (Nodes) 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.

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, e.g. with time information) are supported, user addressing is supported. IQMESH protocol was defined as a light and portable to the inexpensive microcontrollers with limited resources. Therefore, one or two byte addressing is chosen, enabling to address up to 65 000 devices.

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 supported. 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.



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 should not 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 being readable in given network only. In addition to "user" packets, IQMESH uses also system packets with auxiliary information (e.g. for bonding, routing etc.). Such system packets are completely transparent from the user's point of view.

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

## RF networking

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

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

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

### PAH

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

### PIN

bit	7	6	5	4	3	2	1	0
	NTWF		ROUTEF		MPRWF			

NTWF:

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

ROUTEF (for IQMESH mode only):

- ROUTEF = 0 Routing not required for outgoing packets.
- ROUTEF = 1 Routing required for outgoing packets, routing vector RTV0-3 must be defined.

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.

Others PIN flags should not be used by the user in this OS version.

### NTWINFO (applies for IQMESH mode only)

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

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

RX Address of the device the packet is intended to in current network:

- 0 Coordinator
- 1 - 239 Nodes
- 240 - 254 Groups
- 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 after RFTXpacket().

PID Packet identification (can be set by the user, e.g. not to respond twice to the same packet)

RT0TX Originated sender of packet

RTDEF Routing algorithm.

RTDT0-3 Routing data. See below.

MPRW0-2 See Direct peripheral access

### User data

Data length can vary between 0 and 64 B.

## 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.

Signal strength filtering see below (RF receiving).

## RF transmitting

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

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

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

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

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

## RF receiving

The `RFRXpacket()` function attempts to receive a packet and returns control to application after successful reception or after the timeout. Timeout during packet receiving terminates the reception except of the Wait packet end option is enabled.. The user has full control on timing as the timeout can be set in ticks (~10ms in STD RX mode or in cycles ~30 ms in LP RX or ~600 ms in XLP RX) prior to the `RFRXpacket()` function call (`toutRF = ...`).

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

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

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

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

- STD: standard
- LP (low power): receiving combined with standby mode. Incoming 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. Incoming 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.

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

*Example:*

RX	TX
<code>setRFmode(0x01); // LP_RX 30ms off period</code>	<code>setRFmode(0x10); // LP_TX preamble 30ms</code>
<code>touRF = 5; // [cycle]</code>	<code>touRF = 5; // [tick]</code>
<code>while (1)</code>	<code>while (1)</code>
<code>{</code>	<code>{</code>
<code>clrwdt();</code>	<code>...</code>
<code>if (RFRXpacket())</code>	<code>PIN = 0;</code>
<code>{</code>	<code>DLEN = 22;</code>
<code>...</code>	<code>RFTXpacket();</code>
<code>waitMS(5);</code>	<code>waitMS(5);</code>
<code>PIN = 0;</code>	<code>if (RFRXpacket())</code>
<code>DLEN = 22;</code>	<code>{</code>
<code>RFTXpacket();</code>	<code>pulseLEDG();</code>
<code>}</code>	<code>}</code>
<code>}</code>	<code>}</code>

## Routing

Routing allows sending packets to addressees out of the sender's range using "hops" via devices which are in range each other. This IQRF OS supports up to 240 routing devices for a packet. There are several routing algorithms specified in the `RTDEF` register according to network topology. Packet is routed via devices in specified order (routing vector) in defined time slots with specified period each. Retransmitting passes in reverse order. OS ensures that the packet is ignored by all devices except of the addressee and the devices specified in the routing vector.

For effective IQMESH the topology (placement of devices with respect to the range) should be designed in a redundant way - every device should have sufficient number of devices in range. Routing algorithm should be specified with respect to reliability and speed requirements. Due to time slots the efficiency should considerably depend on the order in the routing vector as well.

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.

Nodes can be placed according to their addresses (with respect to fixed routing vector 1, 2, 3, ...) or in a random order utilizing Discovery when routing paths are found automatically. Nodes can have user addresses too, either short (up to 240, `RTDEF = 4`) or long (up to 65 000, `RTDEF = 5`).

Routing is possible under all following conditions:

- The routing device is bonded to respective network
- The packet was sent by the original sender with routing requirement (`ROUTEF = 1`)
- The routing device is included in the routing vector of the packet to be routed
- 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:

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

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

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

IQMESH allows up to 65 000 devices in single network but only 240 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). It should be set according to topology (e.g. number of bonded Nodes in vase of chai MESH). `RTDT0` is decremented in each hop.
- `RTDT1`: time slot duration (in ticks). It should be longer than the transmit time for given packet. It depends on number of user data to be sent, `PIN`, `DLEN`, RF mode and RF speed. It can be approximately evaluated (in ticks) for 19.2 kb/s:
  - STD: (1 or 2) + 1 for every 24 B of user data
  - LP: (5 or 6) + 1 for every 24 B of user data
  - XLP: 120
- `RTDT2`: DID (Discovered ID) – identification of Discovery. Not set by the user.
- `RTDT3`: Upper byte of user address if user addressing is used.

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

There are two kinds of addresses:

- Logical address: address assigned during bonding
- VRN (Virtual Routing number): address assigned during Discovery

Routing is transparent from the user's point of view in both cases – the user always uses logical addresses (`RX = ...`).

## Routing algorithms

- **SFM (Static Full MESH):** Routing vector is fixed (1, 2, ..., 240), logical addresses are used for routing as well as for addressing ( $R_X = \dots$ ). Bonding can be done in advance, addresses must be known and placement should be done with respect to topology (addresses increase with the distance from the Coordinator). Discovery is no use.  
*Example:* Selecting  $RTDT0=5$  (5 hops) and  $RTDT1=50$  (50 ms time slot) is analogic to the only routing algorithm in OS v2.xx.
- **DFM (Discovered Full MESH):** Random placement is allowed and addresses need not be known for routing purpose but Discovery has to be performed after 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 divided to several zones. VRNs increase with the distance from the Coordinator (Virtual Routing Structure). Logical addresses are maintained. Then routing is transparent from the user's point of view as well. After a change in topology Discovery should be repeated. DFM is analogic to SFM but VRNs are used instead of logical addresses. User addressing is completely the same ( $R_X = \dots$ ).
- **DOM (Discovered Optimized MESH):** Unlike DFM, number of hops can be optimized by the `optimizeHops()` function. It sets the  $RTDT0$  (number of hops) according to Discovery results to speed up the transmission at the cost of limited redundancy.
- **Tree:** not fully implemented yet.

RTDEF	Routing algorithm
0	Not implemented
1	SFM
2	DFM
4	1B user addresses
5	2B user addresses
8	Tree

## 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 serves 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 during bonding. To avoid possible influence on other modules, bonding can be performed on minimal distance with RF power lowered by the user.

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

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

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

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



## Appendix 1

### EEPROM map

00		40		80		C0
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		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

Available for Node only. Do not use for Coordinator.

Available for Node only. Do not use for Coordinator.

Available for Node only. Do not use for Coordinator.

Application, 32B

Reserved by operating system. Do not use at all.

### RAM map (PIC16F886)

IRP = 0		IRP = 1	
Bank 0	Bank 1	Bank 2	Bank 3
00 Ind. addr.	80 Ind. addr.	100 Ind. addr.	180 Ind. addr.
01 TMR0	81 OPTION_REG	101 TMR0	181 OPTION_REG
02 PCL	82 PCL	102 PCL	182 PCL
03 STATUS	83 STATUS	103 STATUS	183 STATUS
04 FSR	84 FSR	104 FSR	184 FSR
05 PORTA	85 TRISA	105 WDTCON	185 SRCON
06 PORTB	86 TRISB	106 PORTB	186 TRISB
07 PORTC	87 TRISC	107 CM1CON0	187 BAUDCTL
08 -	88 -	108 CM2CON0	188 ANSEL
09 PORTE	89 TRISE	109 CM2CON1	189 ANSELH
0A PCLATH	8A PCLATH	10A PCLATH	18A PCLATH
0B INTCON	8B INTCON	10B INTCON	18B INTCON
0C PIR1	8C PIE1	10C EEDAT	18C EECON1
0D PIR2	8D PIE2	10D EEADR	18D EECON2
0E TMR1L	8E PCON	10E EEDATH	18E -
0F TMR1H	8F OSCCON	10F EEADRH	18F -
10 T1CON	90 OSCTUNE	110 -	190 -
11 TMR2	91 SSPCON2	111 -	191 -
12 T2CON	92 PR2	112 -	192 -
13 SSPBUF	93 SSPADD	113 -	193 -
14 SSPCON	94 SSPSTAT	114 -	194 -
15 CCP1L	95 WPUB	115 -	195 -
16 CCP1H	96 IOCB	116 -	196 -
17 CCP1CON	97 VRCON	117 -	197 -
18 RCSTA	98 TXSTA	118 -	198 -
19 TXREG	99 SPBRG	119 -	199 -
1A RCREG	9A SPBRGH	11A -	19A -
1B CCP2L	9B PWM1CON	11B -	19B -
1C CCP2H	9C ECCPAS	11C -	19C -
1D CCP2CON	9D PSTRCON	11D -	19D -
1E ADRESH	9E ADRESL	11E -	19E -
1F ADCON0	9F ADCON1	11F -	19F -
20	00 A0	120	16 1A0
21	01 A1	121	17 1A1
22	02 A2	122	18 1A2
23	03 A3	123	19 1A3
24	04 A4	124	20 1A4
25	05 A5	125	21 1A5
26	06 A6	126	22 1A6
27	07 A7	127	23 1A7
28	08 A8	128	24 1A8
29	09 A9	129	25 1A9
2A	10 AA	12A	26 1AA
2B	11 AB	12B	27 1AB
2C	12 AC	12C	28 1AC
2D	13 AD	12D	29 1AD
2E	14 AE	12E	30 1AE
2F	15 AF	12F	31 1AF
30	16 B0	130	32 1B0
31	17 B1	131	33 1B1
32	18 B2	132	34 1B2
33	19 B3	133	35 1B3
34	20 B4	134	36 1B4
35	21 B5	135	37 1B5
36	22 B6	136	38 1B6
37	23 B7	137	39 1B7
38	24 B8	138	40 1B8
39	25 B9	139	41 1B9
3A	26 BA	13A	42 1BA
3B	27 BB	13B	43 1BB
3C	28 BC	13C	44 1BC
3D	29 BD	13D	45 1BD
3E	30 BE	13E	46 1BE
3F	31 BF	13F	47 1BF

IRP = 0		IRP = 1	
Bank 0	Bank 1	Bank 2	Bank 3
40	32 C0	140	48 1C0
41	33 C1	141	49 1C1
42	34 C2	142	50 1C2
43	35 C3	143	51 1C3
44	36 C4	144	52 1C4
45	37 C5	145	53 1C5
46	38 C6	146	54 1C6
47	39 C7	147	55 1C7
48	40 C8	148	56 1C8
49	41 C9	149	57 1C9
4A	42 CA	14A	58 1CA
4B	43 CB	14B	59 1CB
4C	44 CC	14C	60 1CC
4D	45 CD	14D	61 1CD
4E	46 CE	14E	62 1CE
4F	47 CF	14F	63 1CF
50	48 D0	150	64 1D0
51	49 D1	151	65 1D1
52	50 D2	152	66 1D2
53	51 D3	153	67 1D3
54	52 D4	154	68 1D4
55	53 D5	155	69 1D5
56	54 D6	156	70 1D6
57	55 D7	157	71 1D7
58	56 D8	158	72 1D8
59	57 D9	159	73 1D9
5A	58 DA	15A	74 1DA
5B	59 DB	15B	75 1DB
5C	60 DC	15C	76 1DC
5D	61 DD	15D	77 1DD
5E	62 DE	15E	78 1DE
5F	63 DF	15F	79 1DF
60	64 E0	160	80 1E0
61	65 E1	161	81 1E1
62	66 E2	162	82 1E2
63	67 E3	163	83 1E3
64	68 E4	164	84 1E4
65	69 E5	165	85 1E5
66	70 E6	166	86 1E6
67	71 E7	167	87 1E7
68	72 E8	168	88 1E8
69	73 E9	169	89 1E9
6A	74 EA	16A	90 1EA
6B	75 EB	16B	91 1EB
6C	76 EC	16C	92 1EC
6D	77 ED	16D	93 1ED
6E	78 EE	16E	94 1EE
6F	79 EF	16F	95 1EF
70	80 F0	170	96 1F0
71	81 F1	171	97 1F1
72	82 F2	172	98 1F2
73	83 F3	173	99 1F3
74	84 F4	174	100 1F4
75	85 F5	175	101 1F5
76	86 F6	176	102 1F6
77	87 F7	177	103 1F7
78	88 F8	178	104 1F8
79	89 F9	179	105 1F9
7A	90 FA	17A	106 1FA
7B	91 FB	17B	107 1FB
7C	92 FC	17C	108 1FC
7D	93 FD	17D	109 1FD
7E	94 FE	17E	110 1FE
7F	95 FF	17F	111 1FF

- reserved for PIC HW    
  - OS buffers    
  - reserved for OS    
  - user available

## Appendix 2

### 868 MHz band channel map

Channel	Bit rate		
	BR1	BR2	BR3
	1.2 - 19.2	57,6	86,2
Frequency [MHz]			
0	863.15	863.15	863.15
1	863.25	863.35	863.55
2	863.35	863.55	863.95
3	863.45	863.75	864.35
4	863.55	863.95	864.75
5	863.65	864.15	865.15
6	863.75	864.35	865.55
7	863.85	864.55	865.95
8	863.95	864.75	866.35
9	864.05	864.95	866.75
10	864.15	865.15	867.15
11	864.25	865.35	867.55
12	864.35	865.55	867.95
13	864.45	865.75	<b>868.35</b>
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	<b>868.35</b>	
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	<b>868.35</b>		
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	<b>916.50</b>	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	<b>916.50</b>		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	<b>916.50</b>		115	918.15		178	927.60
53	908.85	916.80		116	918.30		179	927.75
54	909.00	917.10		117	918.45		180	927.90
55	909.15	917.40		118	918.60		181	928.05
56	909.30	917.70		119	918.75		182	928.20
57	909.45	918.00		120	918.90		183	928.35
58	909.60	918.30		121	919.05		184	928.50
59	909.75	918.60		122	919.20		185	928.65
60	909.90	918.90		123	919.35		186	928.80
61	910.05	919.20		124	919.50		187	928.95
62	910.20	919.50		125	919.65		188	929.10

---

---

## Documentation and Information

- 1 **IQRF OS Reference guide** [www.iqrf.org/weben/downloads.php?id=156](http://www.iqrf.org/weben/downloads.php?id=156)
- 2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix [1]
- 3 **IQRF home page** [www.iqrf.org](http://www.iqrf.org)
- 4 **IQMESH** specification [www.iqmesh.org/iqmesh](http://www.iqmesh.org/iqmesh)
- 5 **SPI** specification [www.iqrf.org/weben/downloads.php?id=85](http://www.iqrf.org/weben/downloads.php?id=85)
- 6 **IQRF support site** [www.iq-esupport.com](http://www.iq-esupport.com)
- 7 **TR-31B** datasheet: [www.iqrf.org/weben/downloads.php?id=92](http://www.iqrf.org/weben/downloads.php?id=92)  
**TR-32B** datasheet: [www.iqrf.org/weben/downloads.php?id=94](http://www.iqrf.org/weben/downloads.php?id=94)
- 8 **PIC16F886** datasheet: [www.iqrf.org/weben/downloads.php?id=126](http://www.iqrf.org/weben/downloads.php?id=126)
- 9 **IQRF IDE**: [www.iqrf.org/weben/downloads.php?id=86](http://www.iqrf.org/weben/downloads.php?id=86)
- 10 **Basic examples** (included in the StartUp Package): [www.iqrf.org/weben/downloads.php?id=112](http://www.iqrf.org/weben/downloads.php?id=112)
- 11 **AN003 – IQRF development tools Installation guide**: <http://www.iqrf.org/weben/downloads.php?id=109>

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

## Document revision

- 101223 Preliminary release, OS v3.00

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

---

## Sales and Service

---

**Corporate office:**

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

**Partners and distribution:**

please visit [www.iqrf.org/partners](http://www.iqrf.org/partners)

---

**Quality management:**

*ISO 9001 : 2000 certified*

**Trademarks:**

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

**Legal:**

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

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

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

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

---

<b>Website</b>	<b><a href="http://www.iqrf.org">www.iqrf.org</a></b>
<b>E-mail</b>	<b><a href="mailto:sales@iqrf.org">sales@iqrf.org</a></b>
<b>On-line support</b>	<b><a href="http://iq-esupport.com">http://iq-esupport.com</a></b>



Simple way to smarter wireless solutions