IQRF OS

Operating System

version 2.10
for TR-31B and TR-32B

User's Guide

Simple way to smarter wireless solutions

Content

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	v2.10 for 5xB	**FSK**
TR–53B		**FSK**

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 OS versions and history:

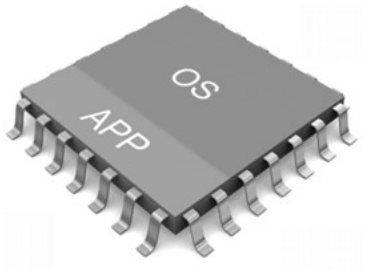
Version	Main differences	Release	Status
v2.10 for 5xB	In addition to that for 3xB: • 868 MHz or 916 MHz **band** software selectable • Enhanced battery check	Jan 2010	**current** for **TR-52B** and **TR-53B**
v2.10 for 3xB	• concept of OS **plug-ins** • RF power not limited during bonding • green LED 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

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

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. This approach reduces time and development costs significantly when creating connectivity applications.

OS helps with development of IQRF applications very effectively. It 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. Moreover even this user part can be very simple thanks to the OS. The user works on much higher level of OS functions. He need not solve partial issues, e.g. how to read a data from equipment connected in a standard way and send it via the RF. He can fully concentrate to implementation of his own demands.

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.

There are two types of OS functions:

- functions intended for using in application programs – block memory copying, writing of a byte or a block to EEPROM, sending/receiving block data via selected interface, starting/stopping of LED blinking, ...
- supportive functions, i.e. tools which support creation and development of the application but do not directly participate in user program, e.g. functions for debugging and uploading the program into the microcontroller.

Operation not supported with OS can be realized by functions located in user program memory area. Specific function can also be written by the user oneself.

OS functions need not run sequentially (next function invoked not until the preceding one is finished) but some operations can run so called "on 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 "on foreground" up to several tasks on background can be running. IQRF OS allows to run even very complex operation including complete communication protocols on 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-02/03).

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

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

OS supports low power consumption of IQRF transceivers with the **Sleep** function when operation of TR module is reduced/stopped until setup time is elapsed or selected inputs are changed.

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-02). The procedure is similar to uploading a user program.

Plug-ins can be located in the part of memory dedicated to OS (2 Kword) or in user part of memory (up to 1 Kword). 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.

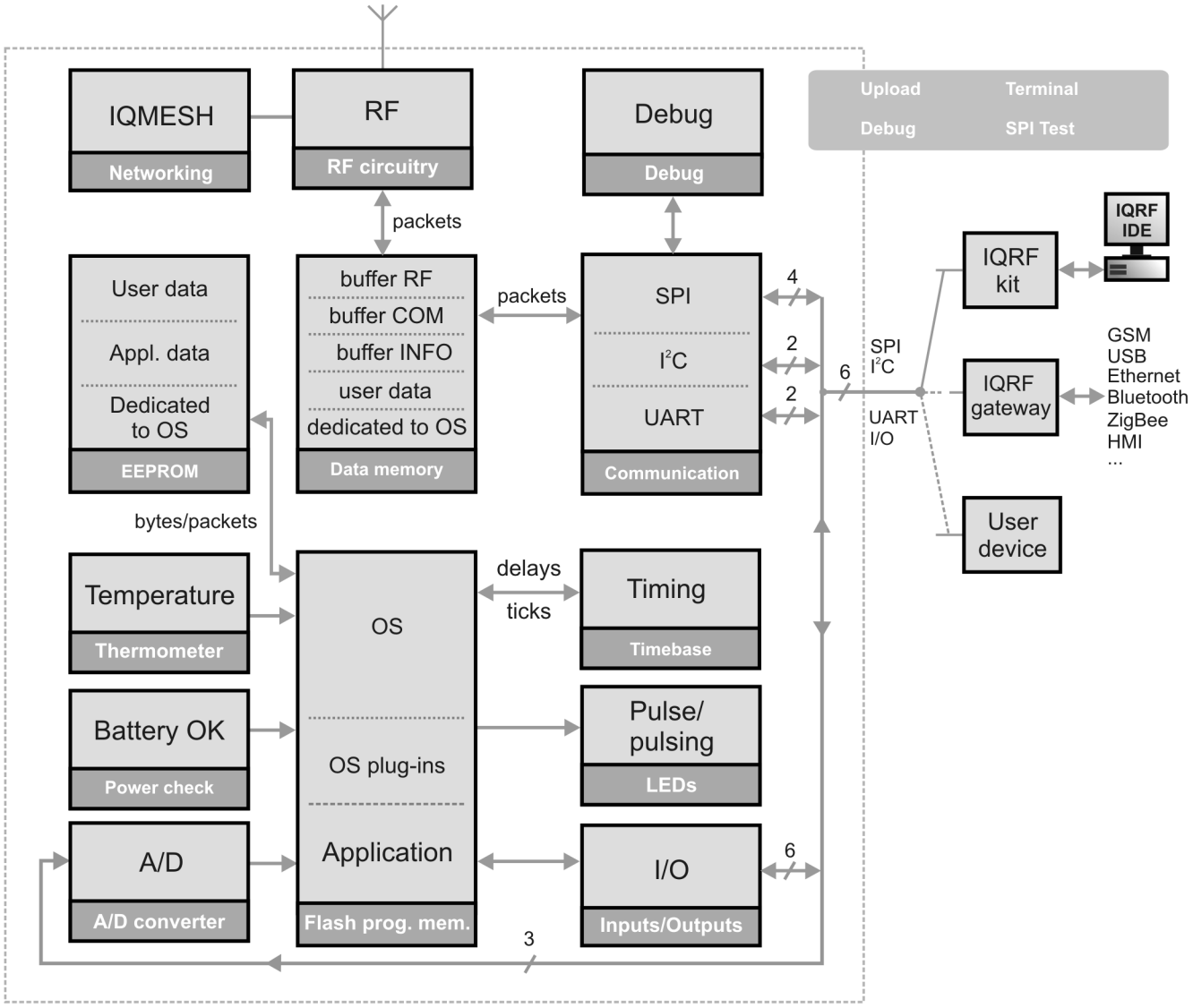
Example: `#include "plug-ins/PlugInXY.h"`

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.

IQRF OS Architecture

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



Individual blocks:

- Memories:
  - program memory (Flash)
  - data memory (RAM)
  - data memory (EEPROM)
- Communication interface:
  - RF (wireless)
  - Standard serial (SPI, I²C, UART)
- Temperature sensor
- 2 LEDs
- Power supply check
- Digital I/O (input or input/output). Up to 6 pins (4 pins are shared with SPI).
- A/D converter
- Real time support: 10 ms interval (tick) generator on background and supporting functions
- Pulse generator (for LED driving on OS background)
- IQMESH networking
- Debug: OS support for testing and debugging

Resources partially depend on transceiver module type.

Microcontroller

The IQRF OS is intended for transceiver modules with the **PIC16LF88** (TR-11A/21A, up to OS v2.08) and **PIC16F886** (TR-31B/32B/52B/53B) MCUs (8-bit microcontrollers by Microchip) – datasheets see [8].

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

PIC HW resources		Utilization
Program memory	Flash	512 instructions (TR-11A/21A) 1024 instructions (TR-31B/32B/52B/53B)
Data memory	RAM	64 B – user data 140B – communication buffers
Data memory	EEPROM	Node: 160 B (+32B application data)
		Coordinator: 0 B (+32B application data)
A/D converter (10b)	internal	Temperature measurement, battery check
	pins	3 external analog inputs (TR-31B/32B/52B/53B)
Serial communication	SPI (slave)	Supported by OS on background
	I^2C (slave)	Realized by PIC HW module and user function
	UART	Realized by PIC HW module and user function
Interrupt		Not user available. It can be disabled just for a short period if necessary.
Brown-out reset		Disabled
Power-up timer		Enabled (TR-11A/21A), disabled (TR-31B/32B/52B/53B)
Watchdog		Timeout 1 ms – 268s (user selectable and programmable) Enabled (TR-11A/21A), SW selectable (TR-31B/32B/52B/53B)
Oscillator		Internal RC, 8 MHz (500 ns instruction). It can be lowered by the user but it is not recommended for keeping correct OS functionality.
Configuration words ("Fuses")	PIC16LF88	`CONFIG1 = 0x0E14, CONFIG2 = 0x3FFF`
	PIC16F886	`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 to the microcontroller using the IQRF development kits only. Indirect RAM access using the FSR register is not allowed due to security reasons. Instead of this IQRF OS provides complete support for indirect addressing using extra system functions. Not dedicated user inputs/outputs, peripherals (e.g. I^2C and UART) and RAM locations can be accessed directly according to user's needs.

Details see datasheets of the transceiver modules [7], PIC datasheets [8] and Appendix - 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.

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 `16F88.h` and `16F886.h` are based on standard files made by the C compiler manufacturer that is why they contain some 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 - 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-02(03) and IQRF IDE servicing program [9]. Codes in standard .HEX format or scrambled codes in the .IQRF format can be uploaded.

- OS occupies the memory from `0x000` to `0xDFF` (TR-11A and 21A) or to `0x1BFF` (TR-31B/32B/52B/52B)
- Remaining area `0xE00h` – `0xFFF` (512 machine instructions) for TR-11A and 21A or `0x1C00h` – `0x1FFF` (1024 machine instructions) for TR-31B/32B/52B/53B is available for user program .

User program should begin from address `0xE00` (TR-11A/21A) or `0x1C00` (TR-31B/32B/52B/53B). It is automatically arranged by the IQRF header files.

Data memory (RAM)

RAM is a fast memory accessible in a comfortable way. 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`. `PIR1` is partially accessible via special OS function only. 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 three basic buffers primarily dedicated to communication and block operation:
    - **`bufferRF`** (`0x110` – `0x14F`), 64 B – for RF communication
    - **`bufferCOM`** (`0xA0` – `0xC8`), 41 B – for serial communication (especially SPI). Use 35 B (`0xA0` – `0xC2`) only, the others are reserved for OS and future compatibility.
    - **`bufferINFO`** (`0x20` – `0x42`), 35 B – for OS and user block operations

    These communication buffers are especially intended for transferred data but can be used according user's need in specific cases as well. There are specialized OS functions for comfortable buffer to buffer data copying.
  - **buffer `networkInfo`** (22B) is an area dedicated to network system information.
  - system variables. Only the `toutRF` register should be directly accessed by the user.
  - OS work variables (not documented, the user need not modify them).
- Remaining area (`0x190` – `0x1CF`) is **available for the user** (64 B). It is located in the **RAM bank 3** excluding the shared area (`0x1F0` – `0x1FF`) where only two more `userRegx` registers (`0x1F0` and `0x1F1`) are available. Selection of bank 3 is automatically arranged by the IQRF header files. Refer to the PIC datasheets [8] for information about RAM banking.

See Appendix (RAM map).

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

Data memory (EEPROM)

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

Individual EEPROM parts:

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

EEPROM access:

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

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

Identification

Module data

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

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

Coordinator / Node: reserved for future OS versions. Coordinator / Node is SW selectable in IQRF OS v2.02.

0: Node
1: Coordinator

OS version:

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

PIC type:

2: PIC16LF88
3: PIC16LF886

OS build: for the manufacturer only. Differences among various builds has no effect for 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  21  02  30  03
```

Meaning: Coordinator, Module ID = `0100101C`, IQRF OS version 2.01, PIC16LF88, build # `0x0330`.

Module ID is displayed with 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 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 required 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. Using number of ticks times of appropriate processes (delay, LED blinking, communication timeout check, ...) can be specified, the timebase can be realized, ...

- Capture is another efficient timing tool. It is an independent resettable timer (16-bit counter of ticks) freely running on background. It is suitable especially for working with long periods (up to 655 s).
- OS provides functions even for waiting on foreground.
- Short time intervals for timing on foreground can be derived also from instruction timing. The PIC16(L)F88(6) is clocked with internal 8 MHz RC oscillator. Thus, instruction cycle is 500 ns (1 µs for some instructions) – see PIC datasheets [8].

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.

Tip: If there is a gateway in the network, it can be used even 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, 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:
  - starting receiving a packet via RF
  - starting receiving via SPI
  - receiving next word via SPI (i.e. after successfully established communication)
- **during transmitting:** 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 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 (TR-31B and higher TR modules only). 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; for TR-31B and highers.

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 ~2.5 μA can be reached with all peripherals off – see the TR datasheets [7].

Other PIC peripherals

There are PIC HW resources (I^2C, UART) not supported with the OS but accessible directly via PIC special function registers. One of them (PIR1) is not directly accessible due to security reasons. To allow using appropriate PIR1 flags, the specialized OS function is available. Refer to the IQRF OS Reference guide [1] (`setPIR1()`) and datasheet of the microcontroller [8].

Reset

If needed, it is possible to run the application program from the very beginning again. It can be accomplished via the `reset()` function used in the user program in given location. It is a real microcontroller reset (the WDT type – see PIC datasheets [8]), automatically followed by reinitialization of OS and user program.

Temperature measurement

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

Battery check

Power supply check uses internal A/D converter of the microcontroller. See `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-02 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 can be 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 distinguish individual debug breakpoints the W register can be used. See IQRF IDE Help and `E06-RAM` example [10] for details.

SPI

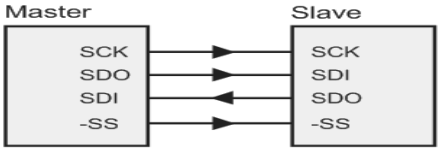
IQRF transceiver modules can communicate with external peripherals via the SPI interface.

SPI™ (Serial Peripheral Interface, introduced by Motorola) is a standard serial four wire synchronous data bus that can operate in full duplex. Devices communicate in master/slave mode with a single master initiating data frames. Multiple slave devices are allowed with individual slave select lines.

The **SPI bus** specifies four logic signals:

SPI signal	TR pin	function	
SCK	C6	Serial Clock	issued by master
SDI	C7	Serial Data In	
SDO	C8	Serial Data Out	
–SS	C5	Slave Select	issued by master, active low

The SPI bus with a single slave:



The IQRF transceiver modules can communicate as SPI slaves. Full as well as half duplex is supported. The SPI protocol is implemented in IQRF operating system. Thanks to the state machine architecture the communication is fully synchronous without any timeouts. It is packet oriented and works on OS background. Packets consist of selectable number of bytes (`0` to N_{max}). In time constrained cases (e.g. during RF receiving) the communication can be slowed down to work on OS foreground with longer delays between individual bytes (SPI slow mode). Data stream can even be suspended at all.

Packet structure:

The master can send two types of packets with the following structure:

Master checks the SPI status of the module:

Master	**SPI_CHECK**
Slave	SPISTAT

Master reads/writes a packet from/to the module:

Master	**SPI_CMD**	PTYPE	DM_1	DM_2	---	$DM_{SPIDLEN}$	CRCM
Slave	SPISTAT	SPISTAT	DS_1	DS_2	---	$DS_{SPIDLEN}$	CRCS

Where:

```
SPI_CHECK = 0x00
SPI_CMD   = 0xF0
```

`SPISTAT`: SPI status of the module

hex value	SPI status
`00`	SPI not active (disabled by the `disableSPI()` command)
`07`	SPI suspended by the `stopSPI()` command
`3F`	SPI not ready (buffer full, last CRCM O.K.)
`3E`	SPI not ready (buffer full, last CRCM error)
`40` to `63`	SPI data ready. Value - `0x40` = number of bytes to be sent from the slave (1 to N_{max})
`80`	SPI ready (communication mode)
`81`	SPI ready (programming mode)
`82`	SPI ready (debugging mode)
`83`	SPI working in Slow mode (e.g. during receiving of RF packet). The Master should prolong the delay between individual bytes when this status is received.
`FF`	SPI not active (HW error)

SPI status of the module is indicated by the IQRF IDE when used together with related IQRF development tools, e.g. CK-USB-02:

PTYPE:

b7	b6	b5	b4	b3	b2	b1	b0
CTYPE		SPIDLEN					

CTYPE: communication type
- 10: full duplex (the master reads/writes from/to the module, bufferCOM changed)
- 00: half duplex (the master reads from the module, bufferCOM unchanged)

SPIDLEN: data length (from 1 to N_{max})
- TR-xxA: N_{max} = 35
- TR-xxB: N_{max} = 35

DM: data from the master

DS: data from the slave

$CRCM = SPI\_CMD \text{ xor } PTYPE \text{ xor } DM_1 \text{ xor } DM_2 \text{ ... xor } DM_{SPIDLEN} \text{ xor } 0x5F$

$CRCS = PTYPE \text{ xor } DS_1 \text{ xor } DS_2 \text{ ... xor } DS_{SPIDLEN} \text{ xor } 0x5F$

Because of SPI runs on OS background information about current SPI state is available via the `getStatusSPI()` function (packet length, busy flag etc.).

Refer to IQRF SPI Specification [5] for detailed information.

See example E07–SPI [10].

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 (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, ...

Related memory locations and registers:

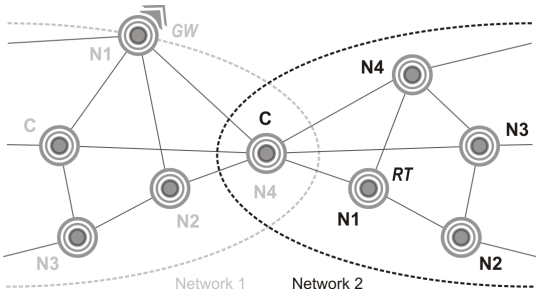
`uns8`	`bufferRF[64]`	buffer for RF routines, 64B long
`uns8`	`DLEN`	packet length, 0-64 (specify before transmitting, automatically set after receiving)
`uns8`	`toutRF`	timeout for packet receiving (in ticks). 1-255, default value is 50 (500 ms).
`uns8`	`PIN`	packet information. See below.

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.

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 one or more groups. Individual and broadcast packets (for all network members, e.g. with time information) are supported, group addressing will be available in future OS versions. IQMESH protocol was defined as a light and portable to the inexpensive microcontrollers with limited resources. Therefore, one byte internal addressing scheme was chosen, enabling to address up to 240 devices and up to 15 groups.

Every IQRF device can simultaneously work in two or more independent networks. This OS version supports two networks for every device, working as a Coordinator in Network 1 and as a Node in Network 2. It allows chaining networks up to unlimited number of devices and easy data sharing. Packets coming from other network(s) can be filtered. Background routing is fully suppported. Each Node can provide background routing service for network packets or can be programed as a dedicated router (RT). Both Coordinator and Node can be realized by a more complex device, a Gateway (GW), 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 almost 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 beeing readable in given network only. In addition to "user" packets, IQMESH uses also system packets with auxiliary information (e.g. for bonding, routing etc.). Such system packets are completely transparent from the user's point of view.

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

RF networking

IQMESH packet transmission is supported with 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	NTW INFO	CRCN	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 should be add to NTWINFO.

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 PAH 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.

RX	TX	CLID0-1	PID	...	RTDEF	RTV0-3	...	MPRW0-2	...

RX Address of the node 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().

CLID0-1 Network identification. Added to packets by OS. See IQRF OS Reference guide [1], getNetworkParams.

PID System information about packet and hops. Dedicated to OS, not intended for users.

RTDEF Routing definition. Reserved for future use. Keep RTDEF=0 (standard routing, 4+1 hops).

RTV0-3 Routing vector (for ROUTEF = 1 only). Addresses of devices to route the packet. It must be specified by the user before sending a packet. Receiving device stores it in reversed order to support seamless acknowledge. See below.

MPRW0-2 See Direct peripheral access

DATA
Data length can vary between 0 and 64 B.

Detailed IQMESH protocol description will be publicly open. See IQMESH specification [4] for details.

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.

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 (CLID0, CLID1, ...) is added to the packet by OS automatically.

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

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. The user has full control on timing as the timeout can be set in ticks (~10ms) prior to the `RFRXpacket()` function call (`toutRF = ...`).

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

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

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

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 (and lower versions from v2.00) supports 4 routing devices for a packet. In standard routing mode (`RTDEF=0`) addresses of routing devices (routing vector) are specified in user program before sending the packet. OS includes the routing vector into the packet and ensures that the packet is ignored by all devices except of the addresse and the devices specified in the routing vector. Routing devices retransmit the packet in specified order (`RTV0` device first) in defined time slots, fixed period each (duration is depended on TR type). During these 1+4 hops the packet should be delivered to the addressee.

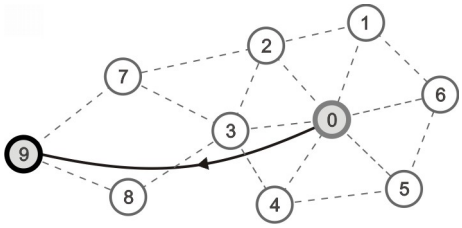
For effective IQMESH the topology (placement of devices with respect to the range) should be designed in a redundant way - every device should have sufficient number of devices in range. Routing vector should be specified with respect to maximal redundancy requirement. Due to time slots the efficiency 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, objects or devices themselves) which results in better throughput and reliability.

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

Example:

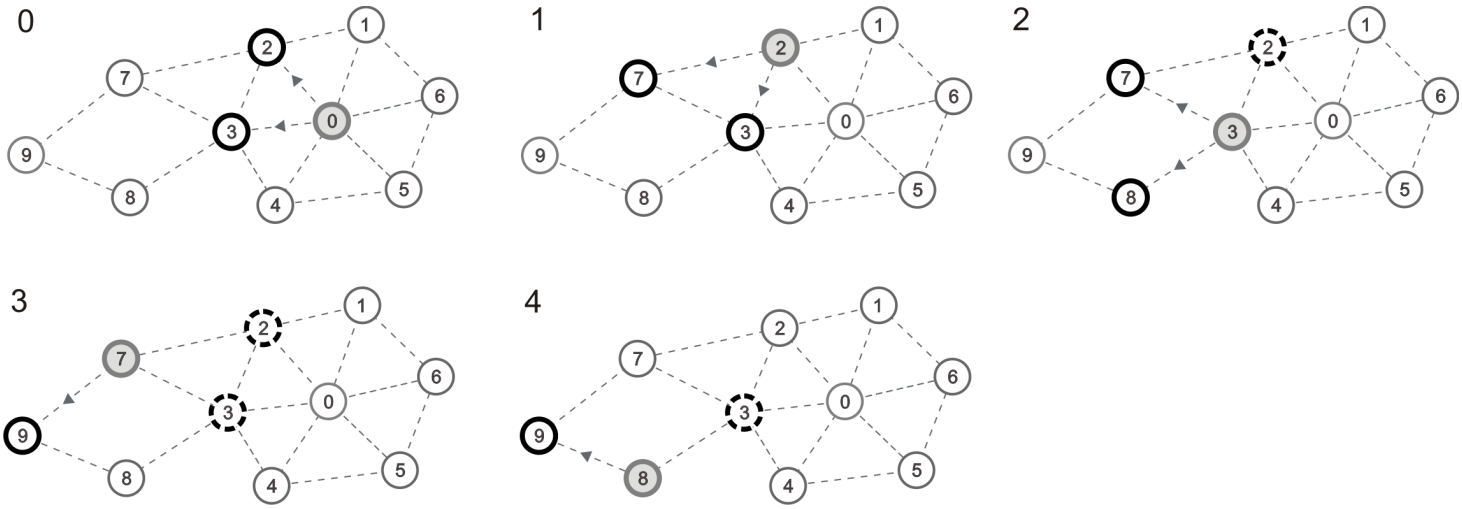
Task: Sending a packet from the device 0 to device 9 in network topology according the picture. `TX=`**`0`**`, RX=`**`9`**, for TR module with 1+4 hops and 50 ms time slots.



Solution: Delivering via devices 2, 3, 7 and 8. `RTV0=`**`2`**, `RTV1=`**`3`**, `RTV2=`**`7`**, `RTV3=`**`8`**.

Situation in individual time slots (**`0`** – **`4`**):

Time slot		Transmitting device	Receiving devices		
			successful	not in routing vector	too late, time slot expired
0	0 – 50 ms	**0**	2 3	1 4 5 6	
1	50 – 100 ms	**2**	3 7	0 1	
2	100 – 150 ms	**3**	7 8	0 4	2
3	150 – 200 ms	**7**	**9**		2 3
4	200 – 250 ms	**8**	**9**		3



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. But just a standard routing (`RTDEF = 0`), 4+1 hops with routing vector specified by the user is supported by this OS version. As a part of the packet, routing vector is stored in receiving device in reversed order to support acknowledgement via the same routing path. *The addresse must not be included in routing vector.*

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 doing nothing but background routing can be realized very simply by a neverending loop:

```
setRoutingOn();
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.

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 from OS v2.10. 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.

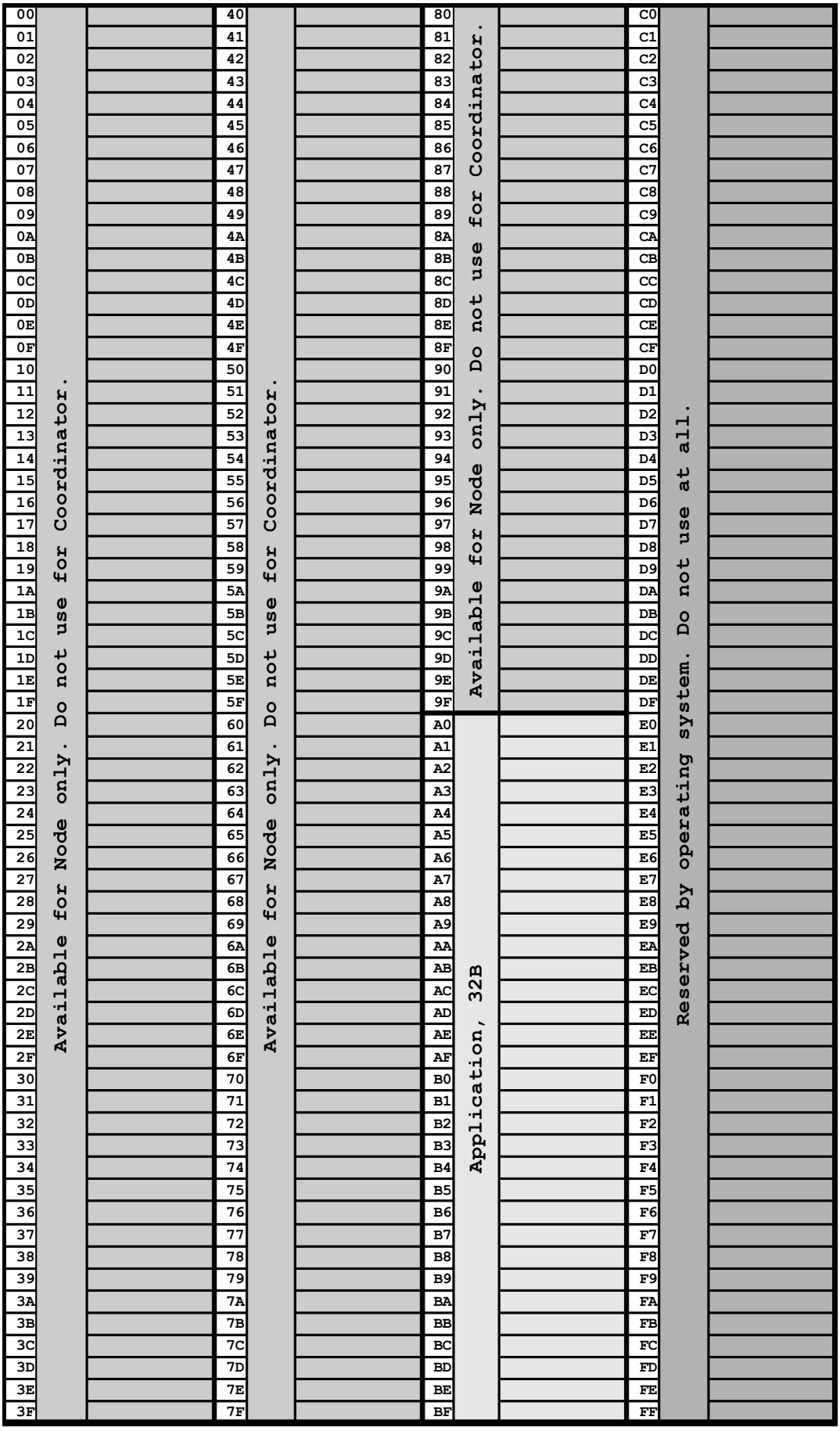
In short, only `bondRequest` and `bondNewNode` exchange RF system packets between Coordinator and Node. All subsequent changes in bonding by either Coordinator or Node are written to EEPROM just on one side. For example, `removeBondedNode()`, `rebondNode()` and `clearAllBonds()` operate with the Coordinator 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.

Logging

Logging means monitoring of communication to register adjacent devices beeing in range each other. OS has the logging capability but it is not intended to be used in this OS version. It is reserved for future usage (Discovery service etc.). Do not use the `setLoggingOn()` and `setLoggingOff()` function in this OS version.

Appendix

EEPROM map



RAM map (PIC16F886 – TR-31B, TR-32B, TR-52B and TR-53B)

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

bufferINFO (20–3F) · bufferCOM (A0–BF) · bufferRF (120–13F)

IRP = 0				IRP =1			
Bank 0		Bank 1		Bank2		Bank3	
40	bu.. 32	C0	32	140	48	1C0	
41	33	C1	33	141	49	1C1	
42	34	C2	34	142	50	1C2	
43		C3	35	143	51	1C3	
44		C4	36	144	52	1C4	
45		C5	37	145	53	1C5	
46		C6	38	146	54	1C6	
47		C7	39	147	55	1C7	
48		C8	40	148	56	1C8	
49		C9		149	57	1C9	
4A		CA		14A	58	1CA	
4B		CB		14B	59	1CB	
4C	toutRF	CC		14C	60	1CC	
4D		CD		14D	61	1CD	
4E		CE		14E	62	1CE	
4F		CF		14F	63	1CF	
50		D0		150		1D0	
51		D1		151		1D1	
52		D2		152		1D2	
53		D3		153	PIN 00	1D3	
54		D4		154	DLEN 01	1D4	
55		D5		155	02	1D5	
56		D6		156	RX 03	1D6	
57		D7		157	TX 04	1D7	
58		D8		158	05	1D8	
59		D9		159	06	1D9	
5A		DA		15A	07	1DA	
5B		DB		15B	RTOTX 08	1DB	
5C		DC		15C	RTDEF 09	1DC	
5D		DD		15D	RTV0 10	1DD	
5E		DE		15E	RTV1 11	1DE	
5F		DF		15F	RTV2 12	1DF	
60		E0		160	RTV3 13	1E0	
61		E1		161	MRW0 14	1E1	
62		E2		162	MRW1 15	1E2	
63		E3		163	MRW2 16	1E3	
64		E4		164	17	1E4	
65		E5		165	18	1E5	
66		E6		166	19	1E6	
67		E7		167	20	1E7	
68		E8		168	21	1E8	
69		E9		169		1E9	
6A		EA		16A		1EA	
6B		EB		16B		1EB	
6C		EC		16C		1EC	
6D		ED		16D		1ED	
6E		EE		16E		1EE	
6F		EF		16F		1EF	
70	userReg0	F0	userReg0	170	userReg0	1F0	userReg0
71	userReg1	F1	userReg1	171	userReg1	1F1	userReg1
72	param1	F2	param1	172	param1	1F2	param1
73	param2	F3	param2	173	param2	1F3	param2
74	param3	F4	param3	174	param3	1F4	param3
75		F5		175		1F5	
76	param4	F6	param4	176	param4	1F6	param4
77		F7		177		1F7	
78		F8		178		1F8	
79		F9		179		1F9	
7A		FA		17A		1FA	
7B		FB		17B		1FB	
7C		FC		17C		1FC	
7D		FD		17D		1FD	
7E		FE		17E		1FE	
7F		FF		17F		1FF	

bufferCOM (C0–DF) · bufferRF (140–168) · networkInfo (15B–168)

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

Documentation and Information

1 **IQRF** OS Reference guide http://www.iqrf.org/weben/downloads.php?id=84
2 **RAM map** and **EEPROM map**, IQRF OS User's guide, Appendix [1]
3 **IQRF** home page www.iqrf.org
4 **IQMESH** specification www.iqmesh.org/iqmesh
5 **SPI** specification www.iqrf.org/weben/downloads.php?id=85
6 **IQRF support** site www.iq-esupport.com
7 **TR-31B** datasheet: www.iqrf.org/weben/downloads.php?id=92
TR-32B datasheet: www.iqrf.org/weben/downloads.php?id=94
8 **PIC16F886** datasheet: www.iqrf.org/weben/downloads.php?id=126
9 **IQRF IDE:** www.iqrf.org/weben/downloads.php?id=86
10 **Basic examples** (included in the StartUp Package): www.iqrf.org/weben/downloads.php?id=112
11 **AN003 – IQRF development tools Installation guide**: 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

- 100118 OS v2.10, for TR-52B and TR-53B
  Concept of plug-ins
  User RAM limited to `0x1CF`
  Documentation and Information updated
- 091201 User's/Reference guide supplement for OS v2.10
- 090819 First release, OS v2.09

If you need a help or more information please visit IQRF support pages [1] and Submit a Ticket with your request. A lot of information is also available on the IQRF web site [3].

Sales and Service

Corporate office:

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

Partners and distribution:

please visit www.iqrf.org/partners

Quality management:

ISO 9001 : 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)

Website	**www.iqrf.org**
E-mail	**sales@iqrf.org**
On-line support	**http://iq-esupport.com**

Simple way to smarter wireless solutions