SPI

Implementation in IQRF

For (DC)TR-7xD

Technical guide

Smarter wireless. Simply.

This document is intended just for those implementing their own SPI master connected to an IQRF TR module.

This document is valid for TR as well as DCTR transceivers. For simplicity, only TR is used further on throughout the document (with minor exceptions where needed).

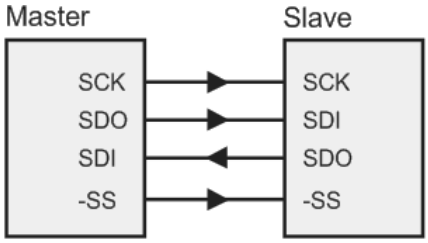
SPI general overview

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

The SPI bus with a single slave:



SPI in TR modules

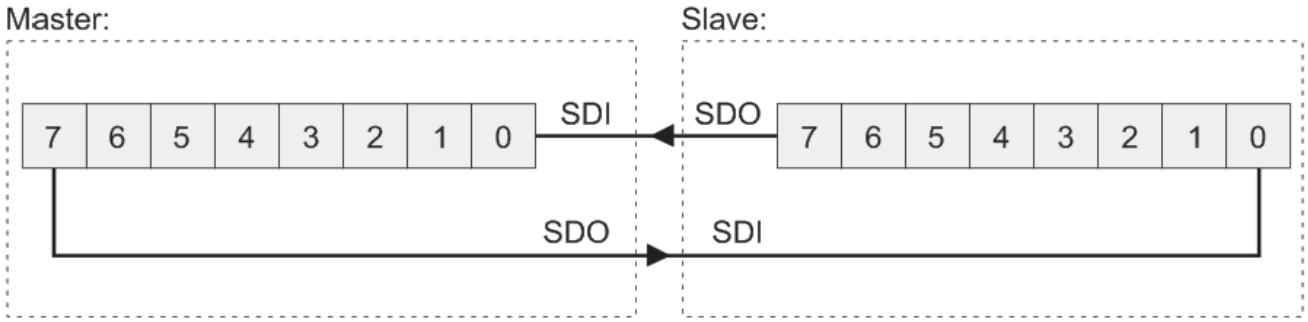
IQRF OS supports TR modules working as SPI slaves, full or half duplex. Master mode can also be realized but it is not supported by OS and must be programed by the user. The hardware module for serial communication (MSSP) inside the microcontroller can be used for this. Using of interrupt is not allowed.

The rest of this document concerns SPI slave implemented in IQRF OS.

Data transfer

SPI communication is fully synchronous without any timeouts. It is packet oriented and works in OS background. Packets consist of selectable number of bytes.

Data is transferred using two internal shift registers to form a circular buffer, MSb first. After the registers have been shifted, the master and slave have swapped register values.



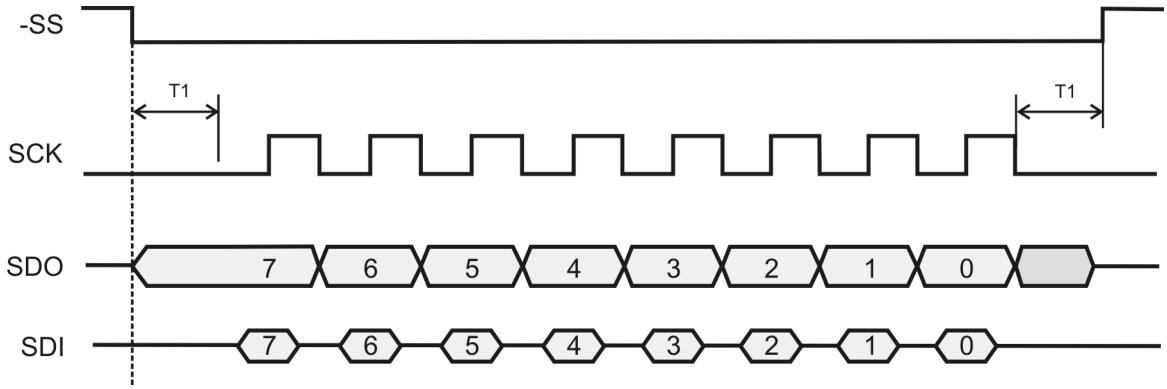
If the slave is configured to receive data, the received byte is copied to the `bufferCOM` then. More bytes can be transferred by this repeated process in single packet and stored to the `bufferCOM`, starting from `bufferCOM[0]`.

The TR module operates according the following **slave specification:**

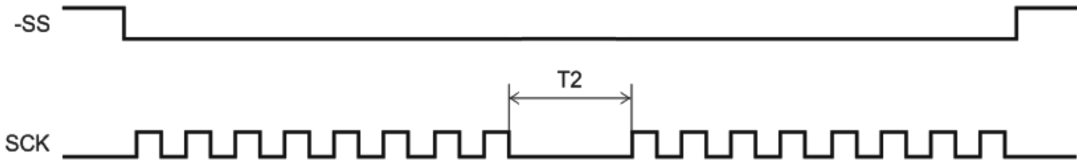
Slave select	Active low
Idle clock polarity	Low
Clock edge	Output data from TR on rising SCK edge

Timing (from the slave point of view):

To start the communication, the master pulls the Slave select low. Then the master must wait for at least the T1 period before starting to issue clock cycles. During each SPI clock cycle, a duplex data bit can be transmitted to complete a full duplex byte transmission in 8 clocks.



Then the master must wait for at least the T2 period before sending next byte.



SCK	SPI clock	250 kHz	max.
T1	–SS to SCK (falling edge), SCK (falling edge) to –SS	5 µs	min.
T2	Delay between bytes	30 µs – in case of non-networking RF communication	min.
		150 µs – in case of networking RF communication	

Delay between bytes (T2) is very important parameter for IQRF SPI communication. Due to RF communication and possible peripheral processes running in background (e.g. writing to EEPROM), proper value depends on the application. RF communication has higher priority than SPI (RF interrupts should be served first), is time-demanding (depending on RF duty cycle) and runs in OS foreground. That is why RF RX has the strongest impact to SPI throughput. Additionally, it strongly depends even on RF noise.

All IQRF standard USB devices (CK-USB-04A, GW-USB-06, …) use T2=150 µs.

For detailed information about SPI implementation in the MCU see datasheet of respective microcontroller, the MSSP module.

SPI status

IQRF SPI is implemented as a state machine. Thus, TR module stays always in one of given states and the module can be polled for this state at any time:

- by the slave using the `getStatusSPI()` function. See the IQRF OS Reference guide.
- by the master using the `SPI_CHECK` command. See the table below (SPI status).

After the `SPI_CHECK` command sent from the master to the slave via SPI the following answer will be sent from the slave to the master:

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.). Data in `bufferCOM` is protected against overwriting by next transmission from the master. `startSPI(0)` can be used to enable further transmission.
3E	SPI not ready (buffer full, last CRCM error). `startSPI(0)` can be used to recover.
40 to 40+N_{max}	SPI data ready. For 41 to 40+N_{max}: Data length = this value – 0x40. For 40: Data length = 64. Example 1: After the `startSPI(10)` the TR module returns state `0x4A` until SPI transfer is started. It means that 10 B data is prepared in `bufferCOM` to be sent from the slave. Example 2: After the `startSPI(41)` the TR module returns state `0x69`. Example 3: After the `startSPI(64)` the TR module returns state `0x40`.
80	SPI ready – communication mode
81	SPI ready – programming mode
82	SPI ready – debugging mode
FF	SPI not active (HW error)

SPI status of the module is indicated by the IQRF IDE when used with related IQRF development tools (e.g. CK-USB-04A):

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 check

`SPI_CMD:` `0xF0` Data read/write.

`SPI_CMD:` `0xF5` Get TR Module Info. It can be used for TR module with OS v3.02D and higher in communication mode only. 16 B is returned, only first 8 B is implemented in OS v3.02D. Refer to the IQRF OS Reference guide for Module Info format. See Example 2.

`SPI_CMD:` `0xFA` DPA data write (for DPA data read use `0xF0`, see above).

`SPISTAT:` SPI status (see the table above)

`PTYPE:`

b7	b6	b5	b4	b3	b2	b1	b0
CTYPE	SPIDLEN						

`SPIDLEN`: data length (from 1 to N_{max})

N_{max} = *64*

`CTYPE`: communication type

b7 = 1: `bufferCOM` changed (**`DM`** → **`DS`** or **`DM`** ↔ **`DS`**).

b7 = 0: `bufferCOM` unchanged (**`DS`** → **`DM`**)

Typical applications often use just unidirectional transmissions (**`DM`** → **`DS`** and **`DS`** → **`DM`**) but in fact, physical communication is always full duplex:

- **`DM ↔ DS:`** Master sends useful data (to `bufferCOM`) and slave sends useful data (from `bufferCOM`).
- **`DM → DS:`** Master sends useful data (to `bufferCOM`).
  Slave sends dummy data (currently meaningless data from `bufferCOM`).
  Master can ignore incoming data (should be arranged by the user).
- **`DS → DM:`** Master sends dummy data (e.g. zeroes, see example below) just to generate clocks.
  Slave sends useful data (from `bufferCOM`) and ignores incoming data (arranged by OS).

`DM:` data from the master

`DS:` data from the slave

`CRCM` = `SPI_CMD xor PTYPE xor` DM_1 `xor` DM_2 `... xor` $DM_{SPIDLEN}$ `xor 0x5F`
`CRCS` = `PTYPE xor` DS_1 `xor` DS_2 `... xor` $DS_{SPIDLEN}$ `xor 0x5F`

CRCM must be calculated and sent by the master. OS verifies received CRCM and:

- returns 0x3F (O.K.) or 0x3E (error) to the master as an answer to the SPI_CHECK command.
- updates the `SPICRCok` flag for the slave after the `getStatusSPI()`.

CRCS is calculated and included to outgoing packet by OS and should be verified by the master.

IQRF OS functions related to SPI

`enableSPI()`, `disableSPI()`, `startSPI(x)`, `restartSPI()`, `stopSPI()` and `getStatusSPI()`. See the IQRF OS Reference guide.

Application

All communication is under the master's control. The slave can not initiate the communication at all, it can only offer one's own state to the master including a request to send data. Therefore the master should watch the state of the slave continuously and respose in corresponding way (comply with slave requirements, read offered data, calculate CRCM and check CRCS, verify successful reception by the slave, repeat unsuccessful transfers etc.)

Master should periodically send `SPI_CHECK` to get status of TR. Good practise for master MCU is to send `SPI_CHECK` every 10 ms.

If the slave requests to send data, the master should generate the "reading" packet for it. If it does not, the slave still stays in the state 0x40 to 0x7F and can be recovered e.g. by `startSPI(x)`.

If the master needs to send data it should check whether the slave has processed previous data (so that the slave is in the communication mode again). If omitted the packet is lost and the bufferCOM is not overwritten. To refuse data by the slave for other reasons the stopSPI() function is intended. If the slave is not ready (SPI is stopped or disabled by the slave or the `bufferCOM` is full) it is useless to send packets by the master.

Stopped SPI communication can also be restarted by `startSPI()` to continue. Thus, current packet can be completed later on.

Useful data can be exchanged at the same time (full duplex) but it is not usual in typical applications.

The slave shoud also take into consideration its own state using `getStatusSPI()`. Return value and output flags are optimized for user program requirements and that is why they differ from the SPI status format sent to the master.

If a communication error occurs during reading the `bufferCOM`, reading can immediately be repeated. Data still remains valid in the `bufferCOM` (unless it is overwritten by the user application). See Example 3.

If the CRCM / CRCS check is not sufficient, for further security increase, the user should include additional verification in user data.

See IQRF OS User's guide, IQRF OS Reference guide, Application examples, Application notes and www.iqrf.org.

Demo example how to implement SPI master in MCU can be downloaded from www.iqrf.org/158.

Example 1

Communication with TR module via SPI

Configuration:
- The E07-SPI example uploaded in a TR module
- The IQRF IDE 4 – SPI Test running on PC. Values "from master" should be typed to the *Data to send* field and values "from slave" can be watched in the *Terminal Log* window.

```
from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS – the module is in communication mode
.
.                            // Master periodically sends SPI_CHECK to get status of TR.
.
from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS – the module stays in communication mode

from master: F0.81.69.47.00  // Now the master wants to send data: "i"
                             // F0 81: SPI_CMD (PTYPE = 81: DM → DS, SPIDLEN = 1)
                             // 69: "i" in ASCII
                             // 47: CRCM (F0 xor 81 xor 69 xor 5F)
                             // 00: SPI_CHECK added to get SPI_STATUS: CRCM ok or error

from slave:  80.80.30.EE.3F  // 5 B answer for 5 B issued by master
                             // 80 80: 2 x SPI_STATUS – the module is in communication mode
                             // 30: dummy - current (undefined) content of bufferCOM
                             // EE: checksum (81 xor 30 xor 5F)
                             // 3F: SPI_STATUS: CRCM is ok

from master: 00              // SPI_CHECK
from slave:  4A              // SPI_STATUS - the module offers 10 B in bufferCOM

from master: F0.0A.00.00.00.00.00.00.00.00.00.A5.00
                             // F0 0A: SPI_CMD (PTYPE = 0A: DS → DM, SPIDLEN = 10)
                             // 10 x 00: dummy (to generate master clock only)
                             // A5: checksum (F0 xor 0A xor 00 ... xor 00 xor 5F)
                             // 00:  SPI_CHECK
from slave:  4A.4A.30.31.32.33.34.35.36.37.38.39.54.3F
                             // 4A 4A: 2 x SPI status (ready to sent 10 B)
                             // 30 ... 39: data from bufferCOM (10 B)
                             // 54: checksum (0A xor 30 ... xor 39 xor 5F)
                             // 3F: SPI_STATUS: CRCM is ok

from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS - the module is in communication mode
.
.                            // Master periodicaly sends SPI_CHECK to get status of TR module
.
```

Example 2

Reading Module Info from TR module via SPI

Configuration:

- TR-7xDx transceiver, plugged in CK-USB-04A or similar IQRF kit
- The IQRF IDE 4 – SPI Test running on PC. Values "from master" should be typed to the *Data to send* field and values "from slave" can be watched in the *Terminal log* window.

```
From master: 00         // SPI_CHECK
From slave:  80         // SPI_STATUS - the module stays in communication mode

From master: F5.10.00.00.00.00.00.00.00.00.00.00.00.00.00.00.00.BA.00
                        // F5: command get Module Info
                        // 10: PTYPE - read 16 B
                        // 16 x 00: dummy
                        // BA: CRCM
                        // 00: SPI_CHECK added to get SPI_STATUS: CRCM ok or error

From slave:  80.80.81.00.2B.E1.37.24.41.07.00.00.00.00.00.00.00.51.3F
                        // 80 80: 2 x SPI_STATUS – the module is in communication mode
                        // 81.00.2B.E1: Module ID = 81002BE1, DCTR
                        // 37: IQRF OS version 3.07D
                        // 24: TR type = TR-72D, FCC not certified, MCU type = PIC16LF1938
                        // 41.07: build 0x0741
                        // next 8 B is not defined
                        // 51: CRCS
                        // 3F: SPI_STATUS: CRCM is ok

from master: 00         // SPI_CHECK
from slave:  80         // SPI_STATUS - the module is in communication mode
```

Example 3

Repeated reading after communication failure

```
from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS - the module stays in communication mode

from master: F0.81.69.47.00  // Write the "i" byte
from slave:  80.80.30.EE.3F

from master: 00              // SPI_CHECK
from slave:  4A              // SPI_STATUS - the module offers 10 B in bufferCOM

from master: F0.0A.00.00.00.00.00.00.00.00.00.A4.00
                             // Reading with intentional error in CRCM (A4 instead of A5)
from slave:  4A.4A.30.31.32.33.34.35.36.37.38.39.54.3E
                             // Trailing 3E indicates wrong CRCM
...                          // Send SPI_CHECK unless TR is in communication mode again

from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS - the module is in communication mode

from master: F0.0A.00.00.00.00.00.00.00.00.00.A5.00
                             // Repeated reading with correct CRCM (It is not needed to
                             //   write "i" again, data remains valid in bufferCOM)
from slave:  80.80.30.31.32.33.34.35.36.37.38.39.54.3F
                             // Trailing 3F indicates correct CRCM
from master: 00              // SPI_CHECK
from slave:  80              // SPI_STATUS - the module is in communication mode
```

Migration notes

Migration from TR-5xD to TR-7xD

Features	TR-5xD	TR-7xD
–SS to SCK (falling edge), SCK (falling edge) to –SS (period T1)	T1 = 10 µs	T1 = 5 µs
Min. delay between bytes (period T2)	T2 = from 100 µs to 700 µs	T2 = 30 µs (non-networking RF)
		T2 = 150 µs (networking RF)
Slave select deactivating between bytes (period T3 *)	Necessary (T3 = min. 20 µs)	Not necessary (T3 = 0)

* Refer to the Tech_Guide_SPI_TR-5x (SPI implementation in TR-5xD).

Document history

- 151106 Min. delay between SPI bytes changed for networking RF communication.
- 150608 First release for (DC)TR-7xD

Sales and Service

Corporate office:

MICRORISC s.r.o., Prumyslova 1275, 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 and MICRORISC name are registered trademarks of MICRORISC s.r.o.
PIC, SPI, Microchip and all other trademarks mentioned herein are property of their respective owners.

Legal:

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

Without written permittion 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)

On-line support: support@iqrf.org

Smarter wireless. Simply.