User guide RFMV0.4

Product	RFM
Product No	RFM-LS-B-US, RFM-LS-A-US
Revision	V0.4
Short description	Radio Frequency networking Module

Document history:

Revision	Date	Author/Who	What
1.0	16-Feb-04	STP	Document created.

1 Contents

2 Release information

RFM, Radio Frequency networking Module				
FCC ID	**Part Number**	**HW revision**	**SW revision**	**Comments**
RTN-BCC-RFMV04	RFM-LS-A-US	V0.4	OD4	Integrated antenna
RTN-BCC-RFMV04	RFM-LS-B-US	V0.4	OD4	Antenna connector

3 Definitions

Beacon: A FHSS synchronization message
Binding: Association of Master and Slave id
Cluster: One Master and multiple slaves
RF-ID Unique ID stored in every RFm_x
Star network: A network with one master and multiple slaves. Slaves are only allowed to transmit to one master. Master can transmit to a specific slave.

RFm_M: RF module Master
RFm_S: RF module Slave
RFm or RFm_x: General term for a RF device
Source-RFm: An RFm_x transmitting a frame
Destination-RFm: An RFm_x receiving a frame

User_M: User device connected to RFm_M
User_S: User device connected to RFm_S
User or User_x General term for a User device
Source-User: An User_x transmitting a frame
Destination-User: An User_x receiving a frame

RFm_Retries Number of retransmissions if no ack (or 0)

4 General Description

This document describes the use of RFm_x. How to connect a user device to an RFm_x device, how to associate master/slave devices and how to transfer data to/from user devices is described.

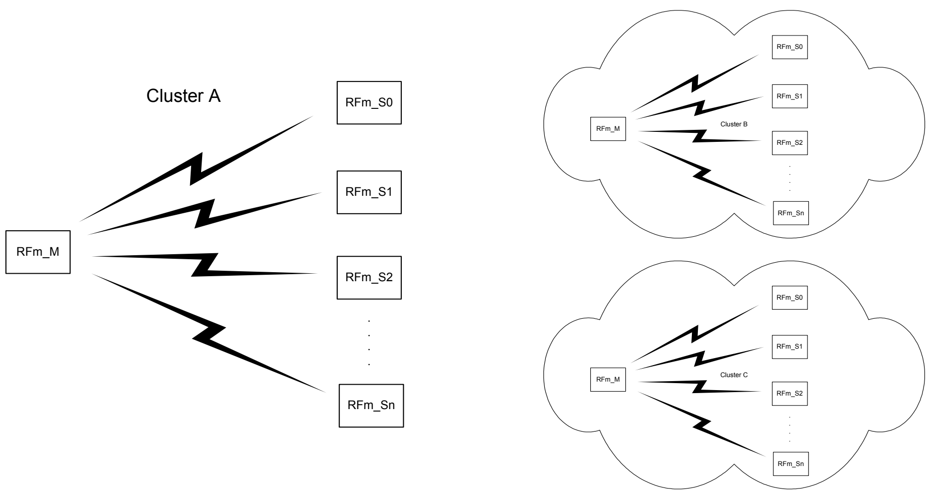
A network is built of a number of RF modules. A RF module is called “RFm_M” or “RFm_S”. Every RFm_x has a unique address. To every RFm_x, one “User-device” is connected.

The purpose of the network is to let the user devices exchange data.

The network has a star topology:

- 1 master RF unit, called RFm_M. Connected to a master user device called User_M.
- 1...64 slave RF units, called RFm_S. Connected to a slave user device called User_S.
- RFm_M can talk to a specific RFm_S
- RFm_S can only talk to one RFm_M

The combination of a master and the associated slaves is called a “cluster”.



A User_x device can set the connected RFm_x in “Programming mode”. In programming mode, parameters can be changed/read and commands can be given to the RFm_x.

A new slave is included in the cluster through an association process. A new RFm_S can use destination address = universal address and RFm_M can be set in “binding mode”. In this mode, the RFm_M will accept all frames with destination address = the universal address.

In “Active mode”, data traffic from slave to master is transparent, except that User_M will get the RF-ID of the source-RFm_S before any data bytes. User_S enters data bytes only (destination is always the master).

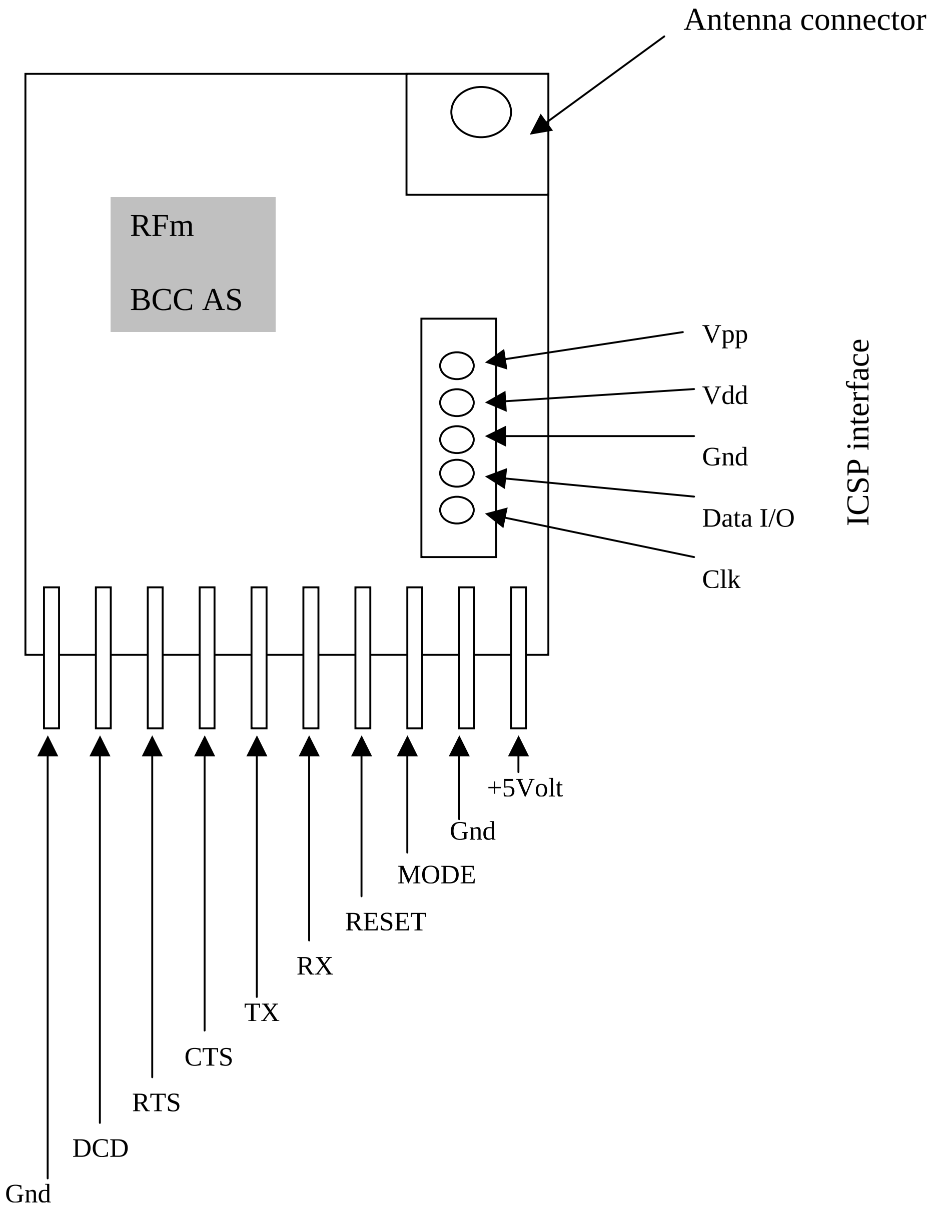
Data traffic from master to slave is also transparent, except that User_M must enter the RF-ID of the destination-RFm_S before any data bytes. If User_M brings RTS inactive and then active again, User_M must enter the RF-ID of the destination RFm again (the same or a new destination). If RTS is kept active, User_M enters only data bytes after the address is entered 1 time. User_S gets data bytes only (the source is always the master).

5 Pinout and User – RFm Interface

The User controls the RFm through a number of pins. These pins are:

- RESET: Input to RFm. User can restart the program (parameters stored in EEPROM are not changed)
- MODE: Input to RFm. Setting this pin active puts the RFm in programming mode. Bringing the pin inactive: RFm enters a user-specfied mode of operation
- RX: Input to RFm. Serial data/commands from User
- TX: Output from RFm. Serial data/commands from RFm
- DCD: Output from RFm. Indicates that the last txed data was ack'ed by the destination-RFm
- CTS: Output from RFm. Indicates RFm is ready for data from User
- RTS: Input to RFm. Indicates User wants to transfer data, or User is ready for data from RFm

- RESET is active low
- MODE, CTS, RTS and DCD are active low
- UART: RX and TX are idle high. Start-bit is low, data bits are "1:1", stop bit is high
- UART bitrate and bitformat: 57600-8-N-1



6 Using the Number of retransmissions Parameter

The number of retransmissions is a User-programmable parameter, referred to as “RFm_Retries”.

There are 2 special cases for this parameter: “No retransmissions” and “Retransmissions until ack’ed”.

If RFm_Retries = 0 (0x00): A source-RFm will not expect ack from the destination-RFm when a data frame is transmitted. And: A destination-RFm will not transmit ack to the source-RFm when a data frame is received.

If RFm_Retries = 255 (0xFF): A source-RFm will re-transmit a packet until ack is received from the destination-RFm, or until power-down. And: A destination-RFm will transmit ack to the source-RFm when a data frame is received.

If RFm_Retries = n (!= 0 and != 255): A source-RFm will re-transmit a packet until ack is received from the destination-RFm, or until n transmissions are made. And: A destination-RFm will transmit ack to the source-RFm when a data frame is received.

Setting RFm_Retries = 255 is not recommended (but kept as an option) because of the possible lock-situation (if destination is not present or the destination address is incorrectly entered by User).

In some cases, it might be advantageous to set RFm_Retries = 0 (especially if data is ack’ed at the User-level). In this case, the RF traffic is reduced. Example: A source-User sends n packets (via the source-RFm) without waiting for ack between packets. After the nth packet, the source-User expects ack. Destination-User gets the packets (from the destination-RFm), and acks all frames or requests a retransmission of 1 or more packets after the nth packet is received (Suggested exercise: Set n=1 in this example). This is a User-protocol issue.

Note this special case:

- If the value of parameter RFm_Retries > 0, a transmitted frame should be ack’ed, or else it will be retransmitted. If RFm_Retries = 0, then no ack is expected by source-RFm, and no ack is sent by destination-RFm. If the source-RFm has RFm_Retries =n (n> 0), but the destination-RFm has RFm_Retries = 0: The source-RFm will transmit the packet n times

7 Use of RTS/CTS and DCD

RTS/CTS are handshake signals between a User and an RFm. That is: They are not handshake signals between User_M and User_S. Example: “RFm_M ready for receiving bytes from User_M” does not imply ”User_S ready to get bytes from RFm_S”.

While RTS/CTS are used for starting/stopping the data stream, DCD indicates “transmitting link ok”. In practice, it will confirm that the last txed data did get through to the destination - RFm. (DCD goes active or stays active) or it will tell User that the last txed data (probably) did not get through (DCD goes inactive or stays inactive).

The User may select to ignore the DCD pin and the “link ok” function.

RTS is User - controlled. When User brings RTS active, it says “User is active” to the connected RFm.

CTS is RFm - controlled. When RFm brings CTS active, it says “RFm is active” to the connected User.

DCD is RFm - controlled. It is only used if “Number of retries” > 0 (refer to section “Using the Number of retransmissions Parameter”). If the last frame was ack’ed, RFm brings DCD active or keeps it active. Else, RFm brings DCD inactive or keeps it inactive.

If a LED is connected to the DCD pin, the state of this line can be monitored visually.

Observe this special case: A frame is successfully received by the destination, but no ack is received by the source. Then DCD will indicate “No success”, although the frame in fact is successfully received by destination.

Note: In “Test-mode RX” (Test1) the DCD line will be inverted whenever a frame with correct CRC is received. This can be used as a communication-link test.

Principle of RTS/CTS from User → RFm:

- User brings RTS active and keeps it active until all bytes are sent or until quitting
- User enters bytes into RFm when CTS is active, and stops entering bytes when CTS is not active or when finished

Principle of RTS/CTS from RFm → User:

- RFm tests if RTS is active or not
- While RFm has data to give to User: RFm gives bytes to user while RTS is active
- CTS is not used by RFm here

7.1 Timing of RTS/CTS

User-controlled timing:

- CTS detected active -> start entering bytes: 0 msec
- Last byte completely entered -> Bring RTS inactive: > 1 msec
- RTS brought inactive -> Bringing RTS active: > 2 msec

RFm-controlled timing:

- RTS brought active -> CTS brought active:
    - If no activity (nothing being txed or rxed or some programming action carried out): < 2 msec
    - If activity: depends on number of retries/length of data to be sent/received etc. Typical example: If txing a 32-byte data packet: > 30 msec
- User has entered < 32 data bytes, then brought RTS inactive -> CTS brought inactive: < 3 msec
- User has completely entered 32 data bytes -> CTS inactive: < 1 msec

7.2 *Detailed Procedure User_x → RFm_x*

Refer to the “cases” described below. Note: User_Master to RFm_Master is described. For a slave, the same procedure is used, except for the entering of an address. Binding mode is not used for a slave. Refer to “Active Mode”, “Binding Mode” and “Programming Mode” as well.

- User_M brings RTS active, indicating “User_M wants to transfer data”
- IF RFm_M is ready to get bytes from User_M, it detects RTS active and brings CTS active, indicating “RFm_M ready”
- User_M detects CTS active and enters address of destination (4 bytes) (if “Active mode” or “Binding mode”) and a number of data bytes, max 32. If User_M enters 32 data bytes, RFm_M tells User_M to stop entering bytes by bringing CTS inactive. If User_M wants to transfer < 32 data bytes, User_M brings RTS inactive after the last byte is completely entered into RFm. In the last case, RFm_M detects RTS inactive and brings CTS inactive.

- If Active Mode or Binding mode: RFm_M now adds overhead (like address and CRC) to the data bytes (making a “frame”), and transmits the frame.
- If programming mode: “Action” is started based on the entered bytes (Examples of “Action”: update a parameter, read a parameter, reset RFm).

- If ack is expected (RFm_Retries > 0):
  - RFm_M searches for ack. If no ack is received before “timeout”, the frame is retransmitted. This is repeated “RFm_Retries” times. The timeout is a random time between (approx) 26 and 100 msec.
  - If no ack after all retransmissions: The DCD pin is brought (or kept) inactive. CTS is brought active if RTS is still active. User must decide if he wants to send more data or not, knowing that the last data entered (probably) did not get through.
  - If ack: The DCD pin is brought (or kept) active. CTS is brought active if RTS is still active.

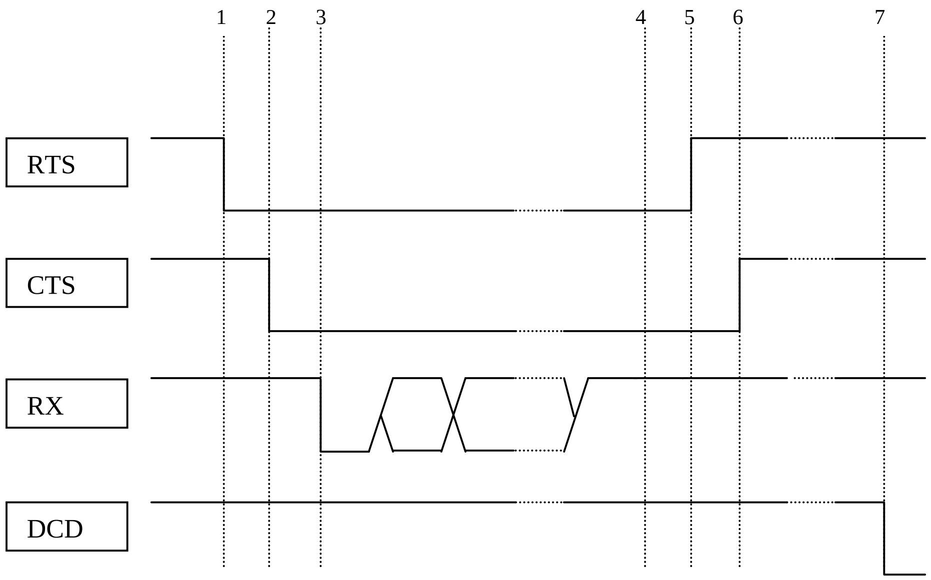
- If ack is not expected (RFm_Retries = 0):
  - DCD is not used (not changed)
  - CTS is brought active if RTS is still active

- If User_M keeps RTS active and detects CTS active again: He can enter more bytes (without entering the address).

- If User_M brings RTS inactive, it must be kept inactive for > 30 msecs. Then, after bringing RTS active again, he must wait for CTS active, enter the destinations address and then the databytes.

Handshake case 1:

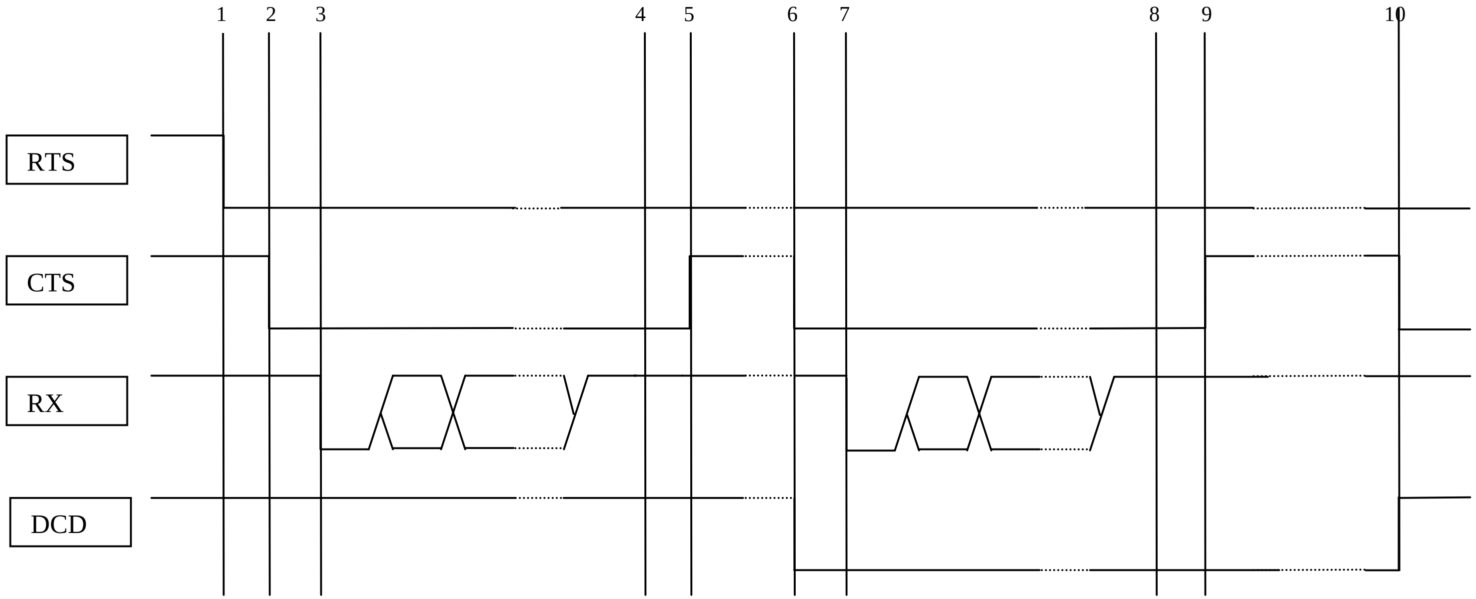
- User wants to enter <= 32 bytes
- DCD is inactive before entering bytes
- “RFm_Retries” > 0
- Data is successfully acked by destination-RFm



1: User brings RTS active
2: RFm brings CTS active
3: User enters start-bit of 1st data byte (or of destination-address if it is master)
4: User has entered stop-bit of last data byte
5: User brings RTS inactive
6: RFm brings CTS inactive and starts processing the entered bytes
7. RFm has received ack from destination-RFm and brings DCD active. Since RTS is inactive, CTS is kept inactive.

Handshake case 2:

- User wants to enter > 64 bytes and keeps RTS active
- DCD is inactive before entering bytes
- "RFm_Retries" > 0
- The 1st entered 32 data bytes are successfully acked by destination-RFm, but
- The last entered 32 data bytes are not acked after "RFm_Retries"



1: User brings RTS active
2: RFm brings CTS active
3: User enters start-bit of 1st data byte (or of destination-address if it is master)
4: User has entered stop-bit of byte #32 (or of #36 if it is master)
5: RFm brings CTS inactive and starts processing the entered bytes
6: RFm has received ack from destination-RFm and brings DCD active. Since RTS is still active, CTS is brought active as well
7: User enters start-bit of data byte #33 (not address, regardless of master/slave -type)
8: User has entered stop-bit of data byte #64
9: RFm brings CTS inactive and starts processing the entered bytes
10. RFm has not received ack from destination-RFm after "RFm_Retries" attempts. It brings DCD inactive. Since RTS is still active, CTS is brought active (user must then decide if he wants to enter more bytes or not).

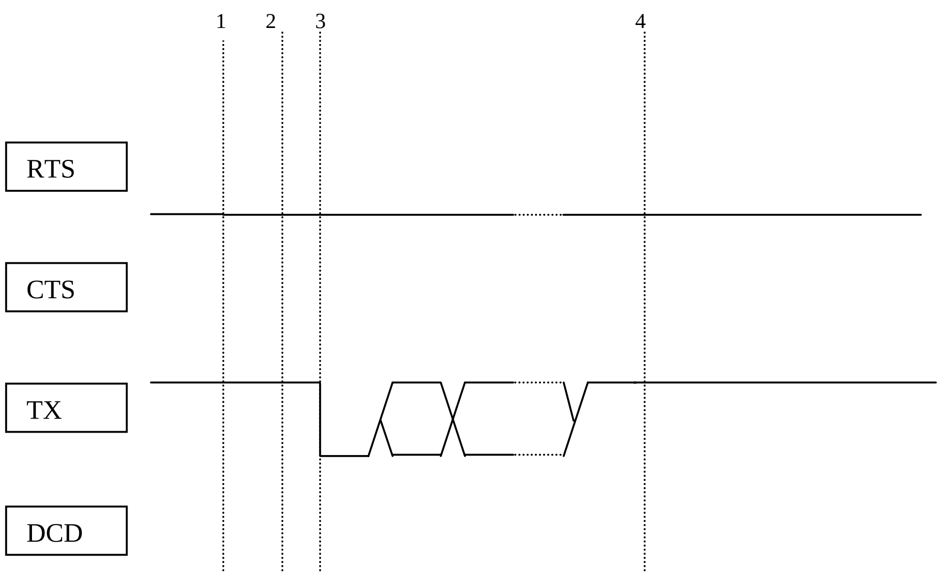
7.3 Detailed Procedure RFm_x → User_x

In the detailed listing below, a RFm_M to User_M transfer is described. The same procedure is used for a RFm_S to User_S transfer, except that RFm_S does not give the address of the source to the User_S (for a slave, the source is always the master).

- RFm_M has received a datapacket (address and CRC OK)
- RFm_M tests if RTS is active or not. If User_M is ready to get bytes, RTS should be active (although active, User_M does not have to enter any bytes)
- RFm_M detects RTS active and transfer the source address and data to User_M
- User_M can stop the transfer by bringing RTS inactive.
- When RFm_M has given source address and data: RFm_M action finished
- Note: CTS and DCD are not changed by this process

Handshake case 3:

- RFm has data to give to User
- User is ready for receiving, indicated by RTS active
- DCD/CTS are not changed by this process

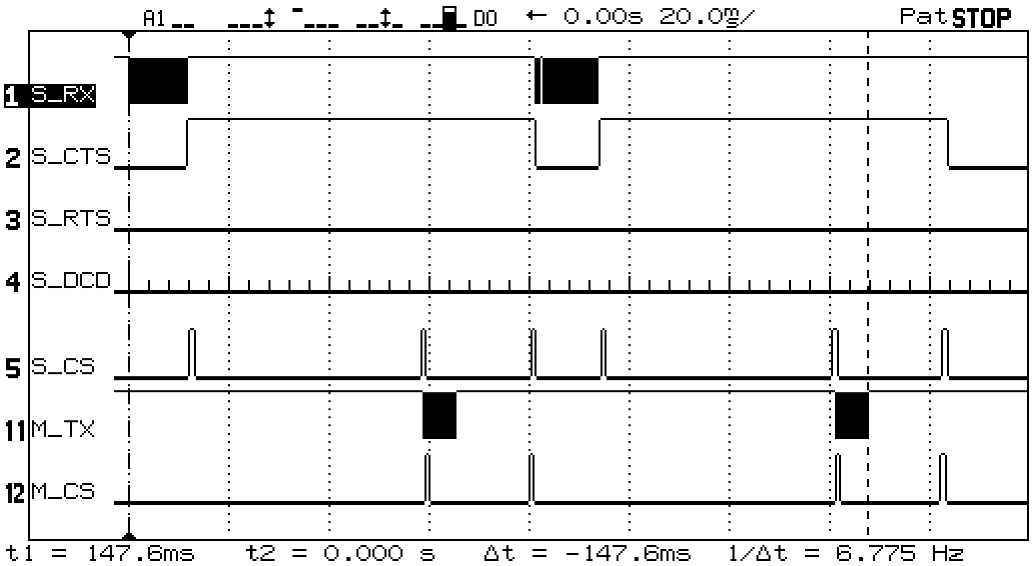


1. RFm receives (via RF) an OK data frame
2. RFm detects RTS active
3. RFm gives out startbit of 1st data byte (or of source-address if it is master)
4. RFm has finished the stop-bit of the last data byte

Suggested exercise: Construct a case where RTS stops and starts data stream on TX line by bringing RTS inactive/active

7.4 Example of Data Transfer from User_S to User_M

In the plot and description below: slave ("S_xxx") and master ("M_xxx") pins are shown.



Note the sequence of events:

- S_RTS is active
- S_CTS is active
- S_RX: User_S enters bytes to RFm_S
- S_CTS goes inactive after the 32nd byte (bytes are entered with a random delay between bytes)
- S_CS goes high, indicating S-RF chip is programmed to tx-mode
- A data-frame is constructed and transmitted
- When RFm_S has transmitted the data frame, these events occur simultaneously:
    - S_CS goes high: S-RF chip is programmed to rx-mode (waiting for ack)
    - M_TX: RFm_M gives data bytes to User_M
    - M_CS goes high: M-RF chip is programmed to tx-mode, an ack-frame is constructed and transmitted
- When RFm_M has transmitted the ack frame, these events occur simultaneously:
    - S_DCD remains low, indicating "data acked"
    - S_CTS goes active, User_S enters 32 more bytes on the S_RX line
    - M_CS goes high: M-RF chip is programmed to rx-mode
    - S_CS goes high – and enters rx and the "correct" frequency
    - The process is repeated for the new 32 bytes
- In this case, User_S gives bytes to RFm_S with a random delay. Including these delays, the total transfer time for 64 bytes (from start of entering the 1st startbit into RFm_S to finished entering the last stop-bit to User_M) is approx 148ms. The effective transfer rate in this case is 640/0,148 = 4300bps.

Note that max 32 data bytes are transferred to RFm, then RFm enters tx mode and transmits the packet. RFm cannot enter tx mode when RFm starts to enter bytes, because there may be random delays between bytes from User, and the "correct" frequency may be another at start-of-entering bytes and finished-entering-bytes.

When a complete frame is received, CRC is tested and then bytes are given to User. RFm cannot give bytes to User before the CRC is read, because the RFm will not give data with errors to User.

Data bytes are packed into a "frame" and transmitted. Included in this frame are source address, destination address, CRC checksum, frame type, frame length, frame ID and frequency/timing info. In addition, a "preamble" and "start of frame delimiter" are transmitted. Therefore, the tx'ing of a frame takes more time than simply txing the data. It takes approx 45 msec to completely transmit a frame with 32 data bytes. A successful ack'ing will take approx 20 msec.

8 Programming Mode

A number of commands ("Requests") can be entered into the RFm, and the RFm can answer (with "Confirms"). "Requests" and "Confirms" are only available in programming mode. "Programming mode" can be entered any time. If the RFm is busy giving data to User, the user should wait until no more bytes are coming before entering programming mode (to separate any "confirms" from "data").

To enter programming mode: Set the MODE pin active. Bring it inactive to exit programming mode. CTS/RTS must be used as described in "Use of RTS/CTS and DCD". The DCD pin is not used in programming mode.

In addition, RFm can give "Indications" to User, and User can answer (with "Response"). Presently, no "Indications" and "Response" are available.

The "primitives" are categorized as "Requests", "Confirms", "Indications" and "Responses":

> "Requests": from User_x to RFm_x
> "Confirms" to requests: from Rfm_x to User_x
>
> "Indications": from RFm_x to User_x
> "Response" to indications: from User_x to RFm_x

The set of available or future primitives is different for an RFm_S and an RFm_M.
When transferring a primitive:
First, enter the start-of-transfer character "'*". The first byte in the primitive is a number (1, 2, 3 or 4) indicating Request, Confirm, Indication or Response, respectively.

The 2nd and 3rd bytes complete the primitive value.

Following the primitive value, a number of parameters may be given (depends on primitive).

After changing parameters (through "Set…" requests), the User should test that the parameter update was successful by reading it back (through "Get…" requests).

Note: If RTS is kept active: User can enter a request by adding additional characters (any char will do) until CTS goes inactive.

If, for a slave, Programming mode is selected at power-on:

- The slave is not in sync with master, and will not try to get in sync with it until programming mode is left. It is suggested to either
  a) wait for the slave to get in sync with master before entering programming mode (see procedure below), or
  b) enter a "reset" command before leaving programming mode (refer to a special note on the reset in the section "How to Enter Requests").
- If programming mode is selected at power-on, and no reset-command is entered, it may take up to 25 seconds for a slave to get sync'ed to a master after programming mode is left.

Method for testing slave-master sync after power-on:

- Make sure MODE and RTS are inactive
- Power-on the RFm
- Wait to make sure CTS inactive
- Bring RTS active
- Wait for CTS active ← This will indicate RFm ready for data bytes from user, and therefore, sync'ed to master.

8.1 Format of the Primitives

Refer to the section “How to Enter Requests” as well.

In every transfer, 3 fields are sent. These are:

1. Start-of-transfer character: From user: Ascii character ‘*’. To user: Ascii character “#”
2. Type of primitive, 3 ascii characters, every character is in the range ‘0’-‘9’
3. Parameters (if any). All parameter octets (bytes) are coded into 2 ascii characters ‘0’-‘9’, ‘A’-‘F’. Every octet is considered a hex number, and high and low nibble of the octet are transferred as ascii characters.

Then, after these 3 fields are entered, the User has 2 options:

1. The RTS line can be kept active. Then, an additional number of characters can be entered until CTS is brought inactive (a total of 32 or 36 bytes)
2. The RTS line can be brought inactive after the parameter-field and then active again when CTS is detected inactive. The time in inactive state should be > 2 msec.

The start-of-transfer character (‘*’) is used to reset the byte-counter in RFm. If User makes a mistake when entering the bytes, he can start over by entering a new ‘*’ (assuming < 32 characters entered and RTS is kept active).

Examples of parameter coding:

Parameter value = 1 => 0x01 => ascii characters ‘0’,‘1’
Parameter value = 56 =>0x38 => ascii characters ‘3’,’8’
Parameter value =255 => 0xFF => ascii characters ‘F’,’F’

Examples of complete primitive-transfer:

Example 1. User wants to set type = Slave:

Start-of-transfer * => ‘*’
Type of primitive 101 => ‘1’,‘0’,‘1’
Parameter value 2 => 0x02 => ‘0’,’2’

Total transfer, in ascii-character notation: *10102
After adding characters to get a total of 32/36, or bringing RTS inactive after the transfer of the primitive, the RFm will update the parameter.

Example 2. User wants to set number of retries = 25:

Start-of-transfer * => ‘*’
Type of primitive 131 => ‘1’, ‘3’,‘1’
Parameter value 25 => 0x19 => ‘1’,’9’

Total transfer, in ascii-character notation: *13119
After adding characters to get a total of 32/36, or bringing RTS inactive after the transfer of the primitive, the RFm will update the parameter.

Example 3. User wants to get the value of "number of retries" (RFm_Retries):

Start-of-transfer * => '*'
Type of primitive 132 => '1', '3','2'
No Parameters

Total transfer, in ascii-character notation: *132

After adding characters to get a total of 32/36, or bringing RTS inactive after the transfer of the primitive, the RFm will give this confirm back:

Start-of-transfer # => '#'
Type of primitive 232 => '2', '3','2'
Parameter value 25=0x19 => '1','9'

Total transfer from User, in ascii-character notation: #23219

Example 4. User wants to set "Master ID" = 0x41414141 (ascii char: AAAA)

Start-of-transfer * => '*'
Type of primitive 121 => '1', '2','1'
Parameters:
ID_Source=Master_ID=2=0x02 => '0','2'
ID=0x41414141 => '4','1', '4','1', '4','1', '4','1'

Total transfer, in ascii-character notation: *1210241414141

8.1.1 Summary of Primitives

*100	RESET_REQ
*101	SET_TYPE_REQ
*102	GET_TYPE_REQ
*108	GET_FW_REQ
*111	SET_MODE_REQ
*112	GET_MODE_REQ
*115	GET_LINKQUAL_REQ
*116	RESTART_LQ_REQ
*117	SET_PWRLVL_REQ
*118	GET_PWRLVL_REQ
*119	SET_LNABIT_REQ
*120	GET_LNABIT_REQ
*121	SET_ID_REQ
*122	GET_ID_REQ
*126	SET_FREQBAND_REQ
*127	GET_FREQBAND_REQ
*129	SET_DEFFREQ_REQ
*130	GET_DEFFREQ_REQ
*131	SET_RETRIES_REQ
*132	GET_RETRIES_REQ

#202	TYPE_CNF
#208	FW_CNF
#212	MODE_CNF
#215	LINKQUAL_CNF
#218	PWRLVL_CNF
#220	LNABIT_CNF
#222	ID_CNF
#227	FREQBAND_CNF
#230	DEFFREQ_CNF
#232	RETRIES_CNF

8.1.2 Primitives and Parameters

In the descriptions below, "Ascii value" means the 3 ascii characters to enter for the primitive.
"Parameters": The ascii chars to enter are shown.

Primitive: Reset_Req()

Ascii value:	100
Parameters:	None
To RFm Master/Slave :	Both
Expected confirm from RFm:	None
Comments:	This is a sw-method to restart the RFm. EEPROM values are not changed. After restarting a RFm_M: It will be busy approx 5 sec sync'ing up all slaves. After restarting a RFm_S: It will sync to the master, typically busy for 2-3 seconds. It is recommended to restart the RFm after changing parameters like Type and Mode (restart when all changes are requested and confirmed). If, due to malfunction, RFm will not talk to the User: A hardware reset is necessary. A 50 msec delay is included after a reset-command is entered, before the program restarts

Primitive: Set_Type_Req(Type)

Ascii value:	101	
Parameters:	Type	
	01	Sets RFm as a RFm_Master
	02	Sets RFm as a RFm_Slave
To RFm Master/Slave :	Both	
Expected confirm from RFm:	None	
Comments:	Type stored in EEPROM, value used until changed by a new Set_Type request	

Primitive: Get_Type_Req()

Ascii value:	102
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	Type_Cnf(Type)
Comments:	

Primitive: Get_Fw_Req()

Ascii value:	108
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	Fw_Cnf(FWversion)
Comments:	Used to get the firmware version used in the RFm

Primitive: Set_Mode_Req(Mode)

Ascii value:	111	
Parameters:	Mode	
	01	Active
	02	Binding
	03	Promiscuous
	04	Test1 (RX on 1 freq)
	05	Test2 (TX carrier on 1 freq)
	06	Test3 (TX 1010… on 1 freq)
	07	Test4 (TX test-packets on 1 freq)
To RFm Master/Slave:	Both	
Expected confirm from RFm:	None	
Comments:	Mode stored in EEPROM, value used until changed by a new Set_Mode_Req request	

Primitive: Get_Mode_Req()

Ascii value:	112
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	Mode_Cnf(Mode)
Comments:	

Primitive: Get_LinkQual_Req()

Ascii value:	115
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	LinkQual_Cnf(LinkQuality)
Comments:	LinkQuality is the average number of transmissions necessary to get ack. (Refer to the excel file “link quality.xls” for a description as to how the average is calculated)

Primitive: Restart_LQ_Req()

Ascii value:	116
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	None
Comments:	After resetting, “LinkQuality” parameter will be read as “00” until some data are txed

Primitive: Set_PwrLvl_Req(PowerLevel)

Ascii value:	117	
Parameters:	PowerLevel	
	00	Power amplifier (PA) off
	01…07	PA is used (01=min., 07=max. power level)
To RFm Master/Slave :	Both	
Expected confirm from RFm:	None	
Comments:	PowerLevel stored in EEPROM, value used until changed by a new Set_PwrLvl request As a general rule, PowerLevel should be set to the lowest possible value to reduce interference on neighboring clusters	

Primitive: Get_PwrLvl_Req()

Ascii value:	118
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	PwrLvl_Cnf(PowerLevel)
Comments:	

Primitive: Set_LNAbit_Req(LNAbit)

Ascii value:	119	
Parameters:	LNAbit	
	00	Include LNA (low noise amplifier)
	01	Bypass LNA
To RFm Master/Slave :	Both	
Expected confirm from RFm:	None	
Comments:	LNAbit stored in EEPROM, value used until changed by a new Set_LNAbit request It might be an advantage to bypass LNA if the unit is close to a strong transmitter. This will prevent saturation in the receiver.	

Primitive: Get_LNAbit_Req()

Ascii value:	120
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	LNAbit_Cnf(LNAbit)
Comments:	

Primitive: Set_ID_Req(ID_Source, ID)

Ascii value:	121	
Parameters:	ID_Source	
	01	(reserved)
	02	Set my Master's ID
	03	(reserved)
	ID	
		Value of the selected ID
To RFm Master/Slave:	Both	
Expected confirm from RFm:	None	
Comments:	The ID must be entered as 8 ascii characters, refer to example in start of this section.	

Primitive: Get_ID_Req(ID_Source)

Ascii value:	122	
Parameters:	ID_Source	
	01	Own_ID,
	02	My Master's ID
	03	(reserved)
To RFm Master/Slave:	Both	
Expected confirm from RFm:	ID_Cnf(ID)	
Comments:		

Primitive: Set_FreqBand_Req(FreqBand)

Ascii value:	126	
Parameters:	FreqBand	
	00	868 MHz band (3 different freqs)
	01	915 MHz band (25 different freqs)
To RFm Master/Slave :	Both	
Expected confirm from RFm:	None	
Comments:		

Primitive: Get_FreqBand_Req()

Ascii value:	127
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	FreqBand_Cnf(FreqBand)
Comments:	

Primitive: Set_DefFreq_Req(DefFreq)

Ascii value:	129	
Parameters:	DefFreq	
	xx	xx = wanted frequency (channel) (0…24)
To RFm Master/Slave :	Both	
Expected confirm from RFm:	None	
Comments:	Used in test modes only “xx” is a hexadecimal number: 0x00 – 0x18 For 868 MHz: Freq 0-11 are equal, 12-23 are equal, 24 is unique For 915 MHz: Freq 0-24 are unique	

Primitive: Get_DefFreq_Req()

Ascii value:	130
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	DefFreq_Cnf(DefFreq)
Comments:	Used in test modes only

Primitive: Set_Retries_Req(Retries)

Ascii value:	131	
Parameters:	Retries	
	n	Max number of retransmissions
To RFm Master/Slave:	Both	
Expected confirm from RFm:	None	
Comments:	2 special cases: 0X00 => no ack/retransmissions are done 0xFF => retransmissions until ack’ed or power-off (0xFF should be used with care) Refer to section “Using the Number of retransmissions Parameter”	

Primitive: Get_Retries_Req()

Ascii value:	132
Parameters:	None
To RFm Master/Slave:	Both
Expected confirm from RFm:	Retries_Cnf(Retries)
Comments:	

Primitive: Type_Cnf(Type)

Ascii value:	202	
Parameters:	Type	
	01	I am an RFm_Master
	02	I am an RFm_Slave
From RFm Master/Slave:	Both	
Result of request from User:	Get_Type_Req()	
Comments:		

Primitive: Fw_Cnf(FWversion)

Ascii value:	208	
Parameters:	FWversion	
	n	FirmWare version used in the RFm
From RFm Master/Slave:	Both	
Result of request from User:	Get_Fw_Req()	
Comments:		

Primitive: Mode_Cnf(Mode)

Ascii value:	212	
Parameters:	Mode	
	01	Active
	02	Binding
	03	Promiscuous
	04	Test1 (RX on 1 freq)
	05	Test2 (TX carrier on 1 freq)
	06	Test3 (TX 1010… on 1 freq)
	07	Test4 (TX test-packets on 1 freq)
From RFm Master/Slave:	Both	
Result of request from user:	Get_Mode_Req()	
Comments:		

Primitive: LinkQual_Cnf(LinkQuality)

Ascii value:	215	
Parameters:	LinkQuality	
	n	Average no of transmissions to get ack
From RFm Master/Slave:	Both	
Result of request from User:	Get_LinkQual_Req()	
Comments:		

Primitive: PwrLvl_Cnf(PowerLevel)

Ascii value:	218	
Parameters:	PowerLevel	
	n	Selected power level (00...07)
From RFm Master/Slave:	Both	
Result of request from User:	Get_PwrLvl_Req()	
Comments:		

Primitive: LNAbit_Cnf(LNAbit)

Ascii value:	220	
Parameters:	LNAbit	
	00	Include LNA (low noise amplifier)
	01	Bypass LNA
From RFm Master/Slave:	Both	
Result of request from User:	Get_LNAbit_Req()	
Comments:		

Primitive: ID_Cnf(ID)

Ascii value:	222	
Parameters:	ID	
		4-byte unique RF_ID
From RFm Master/Slave:	Both	
Result of request from user:	Get_ID_Req(ID_Source)	
Comments:	The ID will be given as 8 ascii characters.	

Primitive: FreqBand_Cnf(FreqBand)

Ascii value:	227	
Parameters:	FreqBand	
	00	868 MHz band (3 different freqs)
	01	915 MHz band (25 different freqs)
From RFm Master/Slave:	Both	
Result of request from User:	Get_FreqBand_Req()	
Comments:		

Primitive: DefFreq_Cnf(DefFreq)

Ascii value:	230	
Parameters:	DefFreq	
	xx	Default frequency
From RFm Master/Slave:	Both	
Result of request from user:	Get_DefFreg_Req()	
Comments:	Used in test modes only	

Primitive: Retries_Cnf(Retries)

Ascii value:	232	
Parameters:	Retries	
	n	Max number of retransmissions
From RFm Master/Slave:	Both	
Result of request from user:	Get_Retries_Req()	
Comments:		

8.2 *How to Enter Requests*

- Make sure all requests starts with the ascii character '*'
- Other chars will be ignored until '*' is found
- If a new '*' is entered before the request is completely entered, the byte-counter in RFm is reset

- Make sure the 3-byte request number is correct (each of the 3 bytes is an ascii-char '0'...'9')
- Make sure parameters are correctly entered. Legal bytes are the ascii chars '0'...'9' and 'A'...'F'.
- Suggestion: Always confirm the programmed params and confirm all params before leaving programming mode

Two methods:

1): Use RTS to separate the requests
2): Keep RTS and MODE active all the time

8.2.1 Method 1: Use RTS to Separate Requests

Bring MODE active

REPEAT_FOR_ALL_REQUESTS
  Bring RTS active
  Wait for CTS active
  Enter the request
  Bring RTS inactive
  Wait for CTS inactive

  If a confirm is expected:
    Bring RTS active
    Wait for and read bytes from RFm
    Bring RTS inactive
END_REPEAT

Bring MODE inactive

Special note on software reset command: Use the method above, and bring MODE inactive immediately (< 50 usec) after detected CTS inactive to avoid re-entering programming mode when the program starts over.

8.2.2 Method 2: Keep RTS and MODE Active All the Time

Bring MODE active
Bring RTS active

REPEAT_FOR_ALL_REQUESTS
  Wait for CTS active
  Enter the request and fill up dummy-bytes until CTS goes inactive
  If a confirm is expected:
    Wait for and read bytes from RFm
END_REPEAT

Bring RTS inactive
Bring MODE inactive

8.2.3 Example: Logged Primitives

In the log below, “Method 2” is used. “Space” is used as dummy-byte.

Note that “get-requests” are followed by a “confirm”. “Set-requests” are not followed by a “confirm”.

```
*102                              #20202
*108                              #20804
*112                              #21201
*115                              #21501
*118                              #21807
*120                              #22000
*12201                            #222504B4232
*12202                            #222504B4233
*132                              #2321F

*102                              #20202
*10101
*102                              #20201
*10102
*102                              #20202

*112                              #21201
*11103
*112                              #21203
*11101
*112                              #21201

*115                              #21501
*116
*115                              #21500

*118                              #21807
*11700
*118                              #21800
*11707
*118                              #21807

*120                              #22000
*11901
*120                              #22001
*11900
*120                              #22000

*12201                            #222504B4232

*12202                            #222504B4233
*1210229292929
*12202                            #22229292929
*12102504B4233
*12202                            #222504B4233

*132                              #2321F
*13105
*132                              #23205
*1311F
*132                              #2321F
```

9 Modes of Operation, Overview

If not in “Programming mode”, the RFm_x will be “operational”. The modes of operation are described in detail in later sections. The modes of operation are:

01: Active mode
For data transfer between master and slave users

02: Binding mode
For master-slave association

03: Promiscuous mode or “Sniffer” mode
For test/debug of transmitted frames within a cluster

04…07: Test mode
For debugging and testing the RFm_x

10 Active Mode

Active mode is also referred to as “traffic mode” or “normal mode”. This is the mode of operation where data is transferred between a User_Slave and a User_Master.

RFm_x must be programmed to active mode through the “Set_Mode_Req(Mode)” request, with Mode = “Active”). Refer to “Programming Mode”.

In active mode, it is possible for User_M to transfer data to a specific User_S, or any User_S to transfer data to the User_M.

CTS/RTS must be used as described in “Use of RTS/CTS and DCD”. The DCD pin is a help to the user and the user can choose to ignore it.

Traffic from User_M to User_S:

- User_M must enter the 4-byte address of the receiving RFm_S, followed by data, into the RFm_M. If User_M has more than 32 bytes of data, the data flow must be stopped when the RFm_M says “stop”. When RFm_M says “continue”, User_M can enter more bytes. Refer to “Use of CTS/RTS and DCD”.

- The receiving RFm_S will give out data bytes (not address of source-RFm) to User_S when User_S is ready to get bytes. Refer to “Use of CTS/RTS and DCD”.

Traffic from User_S to User_M:

- User_S enters only the data to send to User_M into the RFm_S. If User_S has more than 32 bytes of data, the data flow must be stopped when the RFm_S says “stop”. When RFm_S says “continue”, User_S can enter more bytes. Refer to “Use of CTS/RTS and DCD”.

- The receiving RFm_M will give out the 4-byte address of the source-RFm to User_M, followed by 1 - 32 bytes of data, when User_M is ready to get bytes. Refer to “Use of CTS/RTS and DCD”.

10.1 Use of I/O Pins

Refer to “Use of CTS/RTS and DCD” for a detailed description.

10.1.1 User -> RFm

Bring RTS active

If Master:
    Wait for CTS active
    Enter the 4-byte RF-ID of the destination-slave (TX-pin)

REPEAT_FOR_ALL_DATABYTES
    Wait for CTS active
    Enter data byte (TX-pin)
END_REPEAT

Wait for last data byte to be completely entered

Bring RTS inactive ← This will start transmission of the last entered bytes

10.1.2 RFm -> User

WHILE_READY_TO_GET_BYTES
    Bring or keep RTS active
    Read byte, if any (RX-pin)
END_WHILE

Bring RTS inactive to stop bytes from RFm

10.2 Universal Address

If a slave is not associated to a master, it has master_ID = universal address.

The value of the universal address is 0x00000000.

A slave in active mode, with master_id = universal address has a special functionality:

- If User_S gives RFm_S data to transmit, the RFm_S starts to transmit the data. Frequencies to transmit on (the “jump-pattern”) are based on the unique RF-ID of the slave.

- If a RFm_M receives and accepts the frame (RFm_M must be in binding mode to accept frames with destination-address = universal address), the RFm_M transmits an ack back to the RFm_S.

- If the RFm_S receives an ack, it gets sync’ed to the RFm_M sending the ack. Then RFm_M can transmit data frames to RFm_S “without delay”.

- If, however, User_S enters more bytes into RFm_S, and master_id is still equal to universal address: the RFm_S ignores the sync to the previous master and restarts the procedure.

Note: Slaves will not accept any frames with destination id = universal address.

Suggestion: If a slave has master_ID = universal address, then set the “number of retries” parameter (“RFm_Retries”) to > 100.

Refer to “Binding Mode” as well.

10.3 Special Features in Active Mode

ARQ: If the value of parameter “RFm_Retries” > 0, a transmitted frame must be ack’ed by the destination-RFm, or else it will be retransmitted. If RFm_Retries = 0, then no ack is expected by source-RFm, and no ack is sent by destination-RFm. If the source-RFm has RFm_Retries =n (n> 0), but the destination-RFm has RFm_Retries = 0: The source-RFm will transmit the packet n times. Refer to the section “Using the Number of retransmissions Parameter”.

CRC: Before transmitting a frame, a CRC calculation is made by the RFm_x and a 16-bit FCS is included in the frame. When a frame is received, the FCS is tested. If this CRC fails, the received frame is ignored.

Frequency jumping: 25 channels in the 902-928 MHz band are used.

Sync info: To obtain frequency sync between master and slave, the RFm_M adds sync-info to the frame before transmitting it. In addition, the RFm_M transmits “beacons” to maintain the sync. Refer to “Master/Slave Sync Description” in “Sniffer Mode” as well.

11 Binding Mode

Binding mode is only used for a RFm_M.

Binding mode is equal to active mode, except:

- A master in binding mode will accept data frames from any slave if the destination address of the frame is equal to the universal address

In addition to frames with “universal address”, a RFm_M in binding mode will accept frames with destination address equal to the RFm_M’s own unique ID (that is: “Active Mode” operation).

Refer to the section “Universal Address” in “Active Mode” as well.

12 Sniffer Mode

The purpose of this mode is to give the system developer a tool for monitoring RF-traffic.

Typically, a RFm in “sniffer mode” is connected to a PC running e.g. HyperTerminal (that is: “User” = “PC”). Then, on the PC screen, the received messages are shown.

The RFm must be programmed with the following values (refer to “Programming Mode”):

Type = Slave
Mode = Promiscuous
Master_Address = Address of Master in "cluster to sniff"

After programming, it is suggested to give a reset-command or to power off – power on.

Note: Bytes are given to the User without testing RTS line. All bytes are given as ascii characters.

Format of bytes given to User:

Source-RFm (space)
Frame_type (space)
Destination-RFm (space) ← only for data or ack frames
Data bytes (space) ← only for data frames
If crc fails: ? (space)
Carriage return

The RFm give user all received RF messages (even if from another master than the one associated to).

Frame types:

FRAME_DATA	0x01
FRAME_ACK	0x02
FRAME_BEACON	0x03
FRAME_BEACON_REQ	0x04 (used in slave’s sync-procedure)
FRAME_RESET_BEACON	0x05 (used in master’s sync-procedure)

12.1 Master/Slave Sync Description

As soon as a slave's master exchanges frames with any slave, the slave should be synched to the master.

Upon reset/power-on, a RFm_M will transmit 250 “FRAME_RESET_BEACON” to sync up all slaves. This will take 4-5 seconds (CTS is held inactive during this activity). A slave will transmit “FRAME_BEACON_REQ” until got a beacon. This should take 2-3 seconds (note: MODE active will stop the slave activity).

When Master transmits “FRAME_RESET_BEACON” 250 times, it follows a device-specific jump-pattern. The frame contains time/freq info. The number of frames is chosen to ensure all slaves listening should receive at least 1 frame. When a slave receives “FRAME_RESET_BEACON”, the sync-info is used, and the slave is in sync (it stops any on-going beacon-requests). Note: There is no difference between a “reset-beacon” and a “normal beacon”.

After reset/power-on, master and slaves are at a random frequency (1 of 25 freq's). Since the master does not know which freq the slaves are on, a number of beacons must be transmitted, and slaves will not be synced immediately.

After the initial sequence, master transmits sync info at least every 8 secs (approx).

Slave reset/power-on (and not in programming mode): Slave tx “FRAME_BEACON_REQUEST” and waits (a random time) for sync-info from master. If a frame from master is received, the sync-info is extracted and the slave is in sync. It then stops requesting sync info.

If no sync-info for approx 25 seconds, the slave starts to request sync info.

12.2 Example: Output in Sniffer Mode

In the log below, RFm_M has the RF-ID 0x504B4233
Only one slave is used. It has this RF-ID: 0x504B4232

```
504B4233 03
504B4233 03
504B4232 01 504B4233 6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A6A
504B4233 02 504B4232
504B4233 03
504B4233 03
504B4233 03
504B4232 01 504B4233 696969696969696969696969696969696969
504B4233 02 504B4232
504B4232 01 504B4233 6969696969696969696969696969696969696969696969
504B4233 02 504B4232
```

Example:

504B4233 03 ===> RFm_M transmitted a beacon-frame

13 Test Modes

Several test-modes are included for BCC firmware development and hardware testing.

Test1 (Mode04):

The radio chip is programmed to RX mode. The RFm will use 1 frequency only. If a frame is received (correct CRC) and number of bytes in the frame is <= 36, the bytes are given to User. The DCD pin is inverted every time an OK frame is received.

Test2 (Mode05):

The radio chip is programmed to TX mode and transmits a “carrier” signal on this frequency. This can be used for output power, frequency and power consumption measurements.

Test3 (Mode06):

The radio chip is programmed to TX mode and transmits a “1010…” signal on this frequency. This can be used for deviation measurements. In combination with Test1, it can be useful to see that a 1010… signal is transmitted/received correctly.

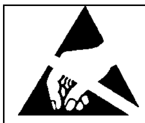
Test 4 (Mode07):

The radio chip is programmed to TX mode and transmits “test-packets” on 1 frequency. The test-packets are made of the alphabet (A…Z) followed by a running number (ascii ‘0’ to ascii ‘9’) and the carriage return character (0x0D). This can be combined with Test1 to test the communication link.

14 Electrical Specifications and Maximum rating

Parameter	Conditions	Value Min	Value Max	Units
Supply voltage, VDD			**5**	**V**
Voltage on any pin	**GND=0**	**-0.3**	**5**	**V**
Storage Temperature range		**-50**	**150**	**°C**
Lead Temp			**250**	**°C**

Note: " Absolute Maximum Rating" indicate the limit beyond witch damage to the device may occur. Recommended Operating conditions indicate conditions for which the device is intended to be functional, but do not guarantee specific performance limits. Electrical Characteristics document specific minimum and/or maximum performance values at specified test conditions. Typical values are for information purposes only- based on design parameters or device characterization and are not guaranteed.



ATTENTION
STATIC SENSITIVE DEVICE
HANDLE ONLY AT STATIC SAFE WORK STATION

f_{RF} = 915MHz, Data-rate=10kbps,, Vdd=5V, T=25°C, unless otherwise specified

Parameter	Conditions	Values Min.	Values Typ.	Values Max.	Units
Overall					
RF frequency operating range		902		927	MHz
Number of Channels	915MHz		25		
Power supply		4.5		5.5	V
Temperature range		-10		70	°C
Transmit section					
Output Power	R_{load}= 50Ω, Pa2-0=111		10		dBm
Output power tolerance			4		dB
Tx current consumption	R_{load}= 50Ω, *11707		33		mA
FSK deviation			130		kHz
Data rate 915MHz	Divider modulation		10		kbps
Receiver section					
Rx current consumption			15		mA
Receiver sensitivity	BER=10^{-3}		-109		dBm
Receiver maximum input power				-20	dBm
Blocking	±1MHz		42		dB
	±2MHz		47		dB
	±5MHz		38		dB
	±10MHz		41		dB
1dB compression			-35		dB
Input IP3	2 tones with 1MHz separation		-25		dBm
Input impedance			~ 50		Ω
Digital Inputs/Outputs					
Logic high input, Vih		0.7*VDD		VDD	V
Logic low input, Vil		0		0.3*VDD	V
User Interface and networking					
User interface			UART		
Data format					
-Bits per second			57.6		kbps
-Data bits			8		bit
-Parity			NONE		bit
-Stop bits			1		bit
-Flow control			HW		

15 Warranty and registration

15.1 FCC Statement:

This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

Reorient or relocate the receiving antenna.
Increase the separation between the equipment and receiver.
Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
Consult the dealer or an experienced radio/TV technician for help.

15.2 FCC Caution

The manufacturer is not responsible for any radio or TV interference caused by unauthorized modifications to this equipment; such changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation

15.3 IMPORTANT NOTE

This equipment complies with FCC radiation exposure limits set forth for an uncontrolled environment. The antenna(s) used for this equipment must be installed to provide a separation distance of at least 8 inches (20cm) from all persons.

The equipment must not be operated in conjunction with any other antenna.

15.4 LIABILITY DISCLAIMER

BLUECHIP COMMUNICATION AS MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DO BLUECHIP COMMUNICATION AS ASSUME ANY LIABILITY ARISING OUR OF THE APPLICATION OR USE OF ANY PRODUCTS OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. ”TYPICAL” PARAMETERS CAN AND DO VARY IN DIFFERENT APPLICATIONS. ALL OPERATING PARAMETERS, INCLUDING ”TYPICALS” MUST BE VALIDATED FOR EACH CUSTOMER'S APPLICATION BY CUSTOMER’S TECHNICAL EXPERTS. BLUECHIP COMMUNICATION AS DOES NOT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.
BLUECHIP COMMUNICATION’S PRODUCTS ARE NOT DESIGNED, INTENDED OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE.
SHOULD BUYER PURCHASE OR USE BLUECHIP COMMUNICATION AS PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD BLUECHIP COMMUNICATION AS AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEE ARISING OUR OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT BLUECHIP

COMMUNICATION AS WAS NEGLIGENT, REGARDING THE DESIGN OR MANUFACTURE OF THE PART.
THE PRODUCT IS WARRANTED BY BLUECHIP COMMUNICATION AS AGAINST DEFECTS IN MATERIALS AND WORKMANSHIP FOR ONE YEAR FROM THE DATE OF ORIGINAL PURCHASE. DURING THE WARRANTY PERIOD WE WILL REPLACE OR, AT OUR OPTION, REPAIR AT NO CHARGE A PRODUCT THAT PROVES TO BE DEFECTIVE, PROVIDED THE PURCHASER RETURNS THE PRODUCT, SHIPPING PREPAID, TO AN AUTHORIZED DEALER. NO OTHER EXPRESS WARRANTY IS GIVEN. THE REPLACEMENT OR REPAIR OF A PRODUCT IS THE PURCHASER'S ONLY REMEDY. ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE YEAR DURATION OF THIS WARRANTY.
BLUECHIP COMMUNICATION SHALL IN NO EVENT BE LIABLE FOR CONSEQUENTIAL DAMAGES.

15.5 Submitting a claim

- The customer must submit with the product as part of the claim a written description of the Hardware defect or Software nonconformance in sufficient detail to allow BlueChip to confirm the same.

- The original product owner must obtain a Return Material Authorization ("RMA") number from BlueChip and, if requested, provide written proof of purchase of the product (such as a copy of the dated purchase invoice for the product) before the warranty service is provided.

- After an RMA number is issued, the defective product must be packaged securely in the original or other suitable shipping package to ensure that it will not be damaged in transit, and the RMA number must be prominently marked on the outside of the package. Do not include any manuals or accessories in the shipping package. BlueChip will only replace the defective portion of the Product and will not ship back any accessories.

- The customer is responsible for all in-bound shipping charges to BlueChip. No Cash on Delivery ("COD") is allowed. Products sent COD will either be rejected by BlueChip or become the property of BlueChip. Products shall be fully insured by the customer and shipped to **BlueChip Communication AS, Strandveien 13, NO1366 Lysaker, Norway**. BlueChip will not be held responsible for any packages that are lost in transit to BlueChip. The repaired or replaced packages will be shipped to the customer via UPS Ground or any common carrier selected by BlueChip, with shipping charges prepaid. Expedited shipping is available if shipping charges are prepaid by the customer and upon request.

BlueChip may reject or return any product that is not packaged and shipped in strict compliance with the foregoing requirements, or for which an RMA number is not visible from the outside of the package. The product owner agrees to pay BlueChip's reasonable handling and return shipping charges for any product that is not packaged and shipped in accordance with the foregoing requirements, or that is determined by BlueChip not to be defective or non-conforming.

What Is Not Covered: This limited warranty provided by BlueChip does not cover: Products, if in BlueChip's judgment, have been subjected to abuse, accident, alteration, modification, tampering, negligence, misuse, faulty installation, lack of reasonable care, repair or service in any way that is not contemplated in the documentation for the product, or if the model or serial number has been altered, tampered with, defaced or removed; Initial installation, installation and removal of the product for repair, and shipping costs; Operational adjustments covered in the operating manual for the product, and normal maintenance;
Damage that occurs in shipment, due to act of God, failures due to power surge, and cosmetic damage; Any hardware, software, firmware or other products or services provided by anyone other than BlueChip; Products that have been purchased from inventory clearance or liquidation sales or other sales in which BlueChip, the sellers, or the liquidators expressly disclaim their warranty obligation pertaining to the product. Repair by anyone other than BlueChip or an Authorized BlueChip Service Office will void this Warranty

Disclaimer of Other Warranties:

Except for the limited warranty specified herein, the product is provided "as-is" without any warranty of any kind whatsoever including, without imitation, any warranty of merchantability, fitness for a particular purpose and non-infringement. If any implied warranty cannot be disclaimed in any territory Where a product is sold, the duration of such implied warranty shall be limited to Ninety (90) days. Except as expressly covered under the limited warranty provided Herein, the entire risk as to the quality, selection and performance of the product is with the purchaser of the product.

Appendix A: Default Settings and Serial Numbers

Parameter values are programmed into EEPROM. When programming the device, it's possible to program the EEPROM as well. These are the default parameters programmed into EEPROM:

Parameter	Default value	Comments
Type	Slave	Value: 02
Mode	Active mode	Value: 01
Power level	7	Max power, Value: 07
LNA bit	0	LNA not bypassed, Value: 00
Master_adress	0x00000000	Universal address
FreqBand	1	915MHz, Value: 01
DefFreq	12	Value: 0C
Retries	31	Value: 1F

Use of serial numbers (Own ID or Self RF-ID, SRF):

- The universal address 0x00000000 is reserved, and cannot be used as SRF
- Numbers 00001 - 32767 (hex: 0x00000001 - 0x00007FFF): Reserved for BCC test/development
- Numbers 32768 - 65535 (hex: 0x00008000 - 0x0000FFFF): Reserved for Salton test/development
- Numbers 0x30300000 – 0x3030FFFF Reserved for BCC test/development
- Number 0xFFFFFFFF is reserved future features
- Remaining numbers (0x00010000 – 0x302FFFFF and 0x30310000 - 0xFFFFFFFE): Production parts.

The own RF-ID (SRF) is stored in program memory of the PIC.

Important note: A part will have an unique ID. If re-programmed, another unique ID may replace the ID.

The ID is placed in program memory locations 0xFF0-0xFF3. To keep the original unique ID: Before re-programming a part that already has a unique ID, make sure last program-memory address to program is set to 0xFEF (then the original unique ID is kept). In MPLAB menu: Select Programmer-> Settings -> Program – write 0xFEF as the end-location in the "Program Memory Addresses" field. Note: This will only work if the part is not code protected.

The IDs are programmed using a SQTP file (refer to Microchip documentation). This can only be used with PRO MATE. ICD2 can't be used to generate unique IDs in this way. But, if part is not code-protected, it's possible to write to program memory 0x000 - 0xFEF only, and then keeping the ID number (if part originally programmed with a part number).

Appendix B: Programming new Software on the RFm with ICD2

ICD2 from Microchip can be used to program EEPROM and program memory.

From microchip.com, get the latest version of MPLAB. Please follow the instructions on how to install ICD2 driver.

When MPLAB is installed, the following procedure should be followed:

Program the PIC:

- Start MPLAB
- Configure -> Select Device...-> "PIC16F648A" -> OK
- Configure -> Configuration Bits... (Config-window pops up)
- File -> Import... -> "xxx.hex" -> Open (select the wanted hex file)
- In the configuration bits window, confirm that the setting matches (as described below)
- Programmer -> Select Programmer -> MPLAB ICD 2
- Programmer -> Enable Programmer (note: this option may not be available!)
- Programmer -> Program

Oscillator	HS
Watchdog Timer	ON
Power Up Timer	ON
Brown Out Detect	Enabled
Master Clear Enable	Enabled
Low voltage Program	Disabled
Data EE Read Protect	Enabled
Code Protect	ON

Note 1: Observe that the EEPROM can be programmed as well as the program memory (that is: test the EEPROM params after a program update) (programming of EEPROM is selectable through ICD2 settings)

Note 2: It is important to program the config bits as described above. In fact, there are some combinations of config bits that will prevent the PIC to be re-programmed with the ICD2.

Appendix C: PIC Errata: Possible EEPROM Write Error

There is an error in the 1st version of the PIC 16F648A:

EEPROM write procedure may fail.

This will be a problem for parameters with > 1 byte, i.e. the ID of a slave's master.

This is a temp. problem for the PIC16F648A (the part is newly released). The solution is to write the ID until success.

Suggestion: Change 1 or several parameters, then reset the device and confirm all parameters

In the next version of PIC16F648A the problem should be solved.

Refer to microchip.com as well.