Title	RFm module
Doc. Number	BCC-611-004-2003
Short description	RFm RF network module Alpha phase

Document history:

Revision	Date	Author/Who	What
1	16-Nov-03	Stp	Created

Related documents:

Doc	Date	Author/Who	Revision
rfq-icu-0.9	16-Sep-03	SR	0.9
RF2200_B0_1.0x	15-Sep-03	Michel Benoit	1.0
BCC-611-001-2003	4-Oct-03	STP	1.0
BCC-611-002-2003	6-Oct-03	STP	1.0

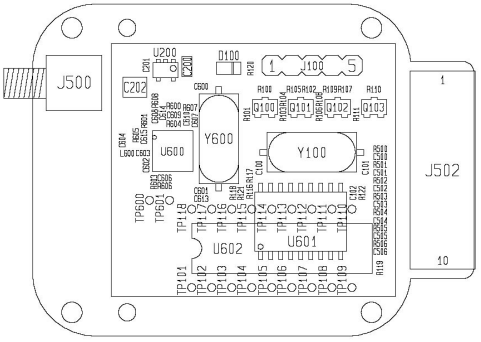
High performance, low cost

Product Description

The Hermes networking module is a fully integrated surface mountable transceiver with star networking capabilities. The module is intended for use as low cost FSK transceiver to establish a frequency-programmable, half-duplex bidirectional RF link. The FHSS multi channelled FSK transceiver is intended for UHF radio equipment in compliance with the North American Federal Communications Commission (FCC) part 15.247 and the European Telecommunication Standard Institute (ETSI) specification EN300 220.The transmitter consists of a VCO, a frequency synthesizer and a programmable power amplifier. The integrated receiver is a direct conversion (Z-IF) receiver allowing low power operation. It consists of a low noise amplifier, mixers, internal filters, limiters and a FSK demodulator.

Features

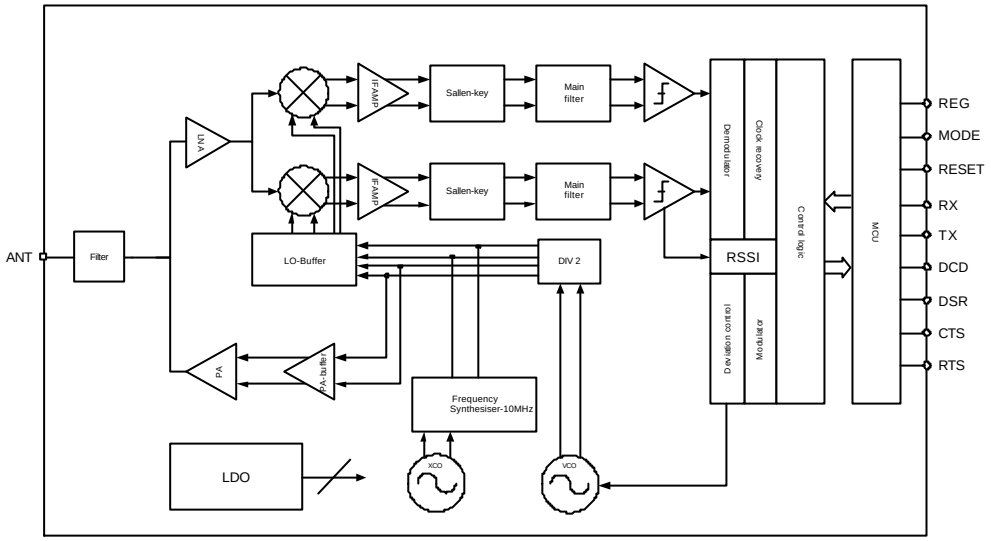
- **915 / 868MHz FHSS**
- **Networking**
- **Low power operation**
- **Programmable protocol**
- **UART interface**
- **ICD2 programmable**



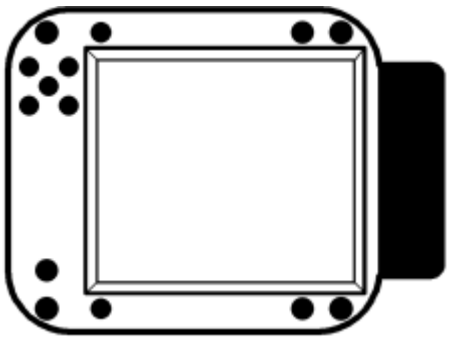
Parameter	Value	Unit	Conditions
Frequency	868/9915	MHz	
Modulation	FSK		
RF output power	10	DBm	2.5V
Current consumption, TX	33	MA	2.5V
Sensitivity	–105	DBm	(10kbps , BER=10^{-3})
Current consumption, RX	15	MA	5V
Maximum data rate	10	kbps	

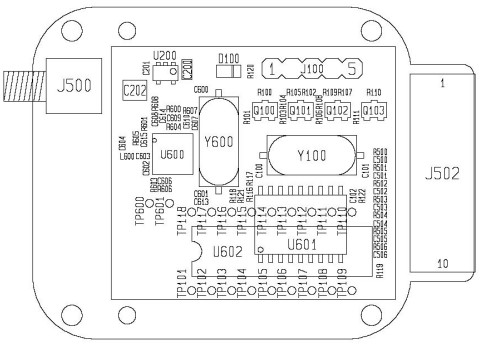
Contents

Block diagram



Pin Assignment





Pin	Name	Function	I/O
1	VCC	+5V	-
2	GND	GND	-
3	MODE	Mode of operation	I
4	RESET	HW Reset	I
5	RX	Serial data in	I

Pin	Name		I/0
6	TX	Serial data out	I
7	CTS	Clear To Send	O
8	RTS	Request To Send	I
9	DCD	Data Carrier Detect	O
10	GND	GND	-

Absolute Maximum Ratings

Parameter	Conditions	Value		Units
		Min	Max	
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**
ESD-Human body model			**t.b.d.**	**kV**

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

Electrical characterization

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

Parameter	Conditions	Values			Units
		Min.	Typ.	Max.	
Overall					
RF frequency operating range		868		927	MHz
Number of Channels	868MHz		4		
	915MHz		25		
Start-up Time			10		mS
Switching time	Rx-Rx		500		µS
	Rx-TX		800		µS
Synchronization time				<<<1	S
Power supply			3		V
Temperature range		-20		70	°C
Transmit section					
Output Power	R_{load}= 50?, Pa2-0=111		10		dBm
	R_{load}= 50?, Pa2-0=000		-11		dBm
Output power tolerance	over temperature range		2		dB
	over power supply range		3		dB
Tx current consumption	R_{load}= 50?, Pa2-0=111		33		mA
	R_{load}= 50?, Pa2-0=000		15		mA
Binary FSK frequency separation	bitrate=200kbps			240	kHz
Data rate 915MHz	VCO modulation		200		kbps
Data rate 868MHz	VCO modulation		40		kbps
Receiver section					
Rx current consumption			15		mA
Receiver sensitivity	38kbaud/s, ß=4		-104		dBm
	200kbaud/s, ß=2		-97		dBm
Receiver maximum input power				-12	dBm
Adjacent channel rejection	200kHz spacing		t.b.d.		dB
	1MHz spacing		t.b.d.		dB
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		?

Parameter	Condition	Values			Units
		Min.	Typ.	Max.	
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			115.2		kbps
-Data bits			8		bit
-Parity			NONE		bit
-Stop bits			1		bit
-Flow control			HW		
CRC			CRC-CCITT		
Network topology			Star		
Addressing					
- Source			2		Byte
- Destination			2		Byte

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)

User-RFm interface
USART

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.

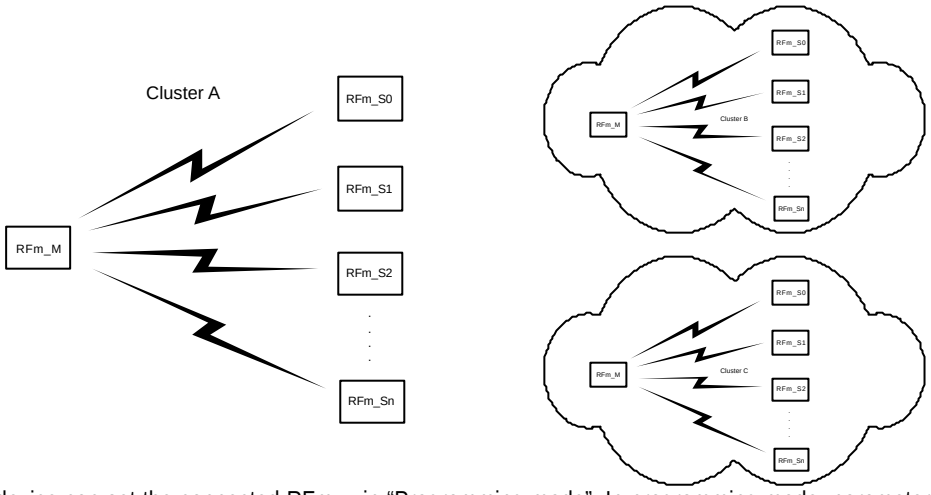
A network is built of a number of RF modules. A RF module is called “RFm_M” or “RFm_S”. Every RFm_x has a unique address. To every RFm_x, one “User-device” is connected.

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 any RFm_S
- RFm_S can only talk to 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 called “binding”. The User_M device must accept the new slave. In addition, User_S must accept the master to complete the binding.

If binding mode is not available, User can do the association process in “Programming mode”. Then User program the RFm’s own RF-ID and the RF-ID of the master RFm (i.e the associated master).

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).

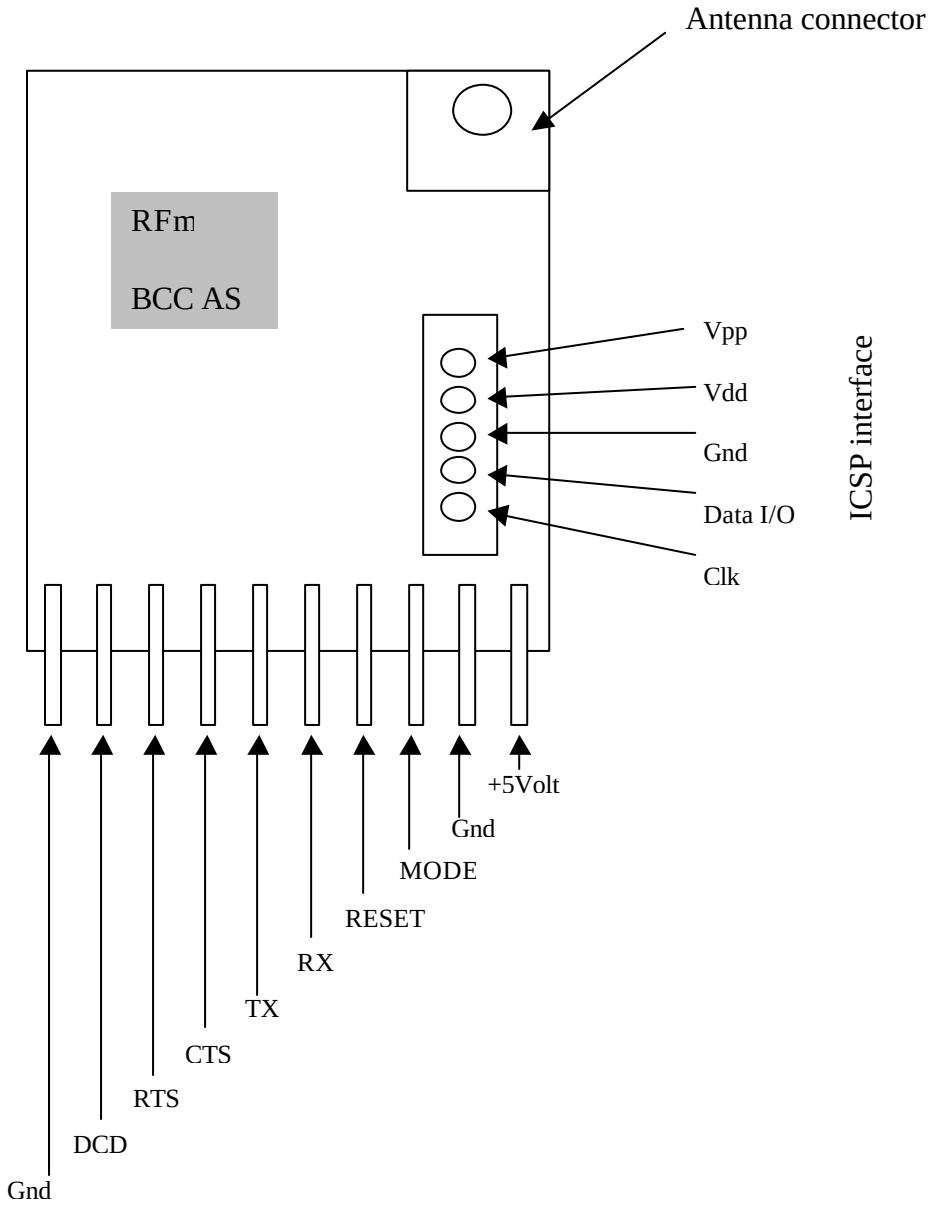
Data traffic from master to slave is also transparent, except that User_M must enter the RF-ID of the destination-RFm_S before any data bytes. If User_M brings RTS inactive and then active again, User_M must enter the RF-ID of the destination RFm again (the same or a new destination). If RTS is kept 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).

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 a link is established between master and slave
- CTS: Output from RFm. Indicates RFm is ready for data from User
- RTS: Input to user. 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: 57600-8-N-1



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: 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).

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 must 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

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 data did get through to the destination - RFm. (DCD goes active or stays active) or it will tell User that the last 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 white 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

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

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.

- 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") and a number of bytes, max 32. If User_M enters 32 bytes, RFm_M tells User_M to stop entering bytes by bringing CTS inactive. If User_M wants to transfer < 32 bytes, User_M brings RTS inactive after the last byte. In the last case, RFm_M detects RTS inactive and brings CTS inactive.

- If Active 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
  - 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.

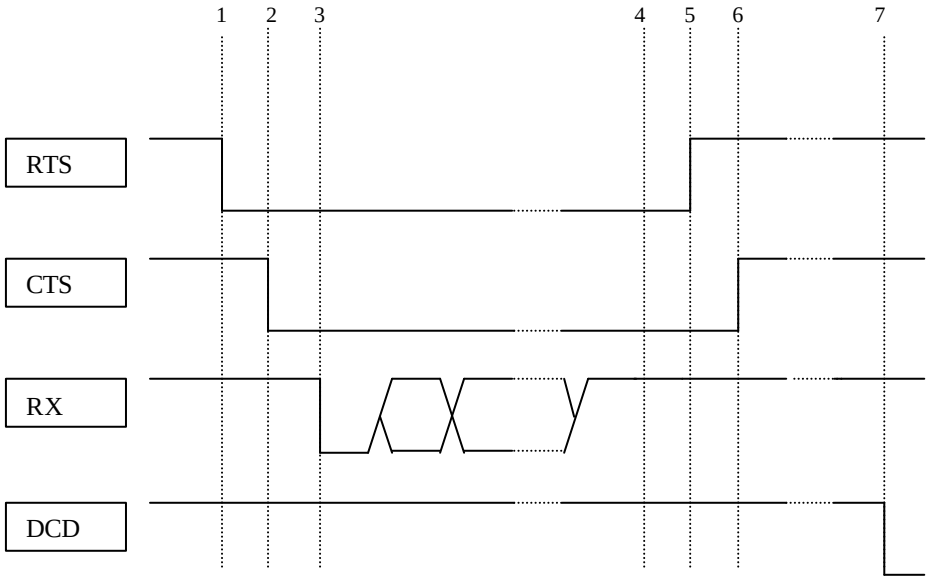
- If ack is not expected (RFm_Retries = 0):
  - DCD is not used
  - CTS is brought active if RTS is still active

- 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, User_M must enter the destinations address first, then the databytes.

Handshake case 1:

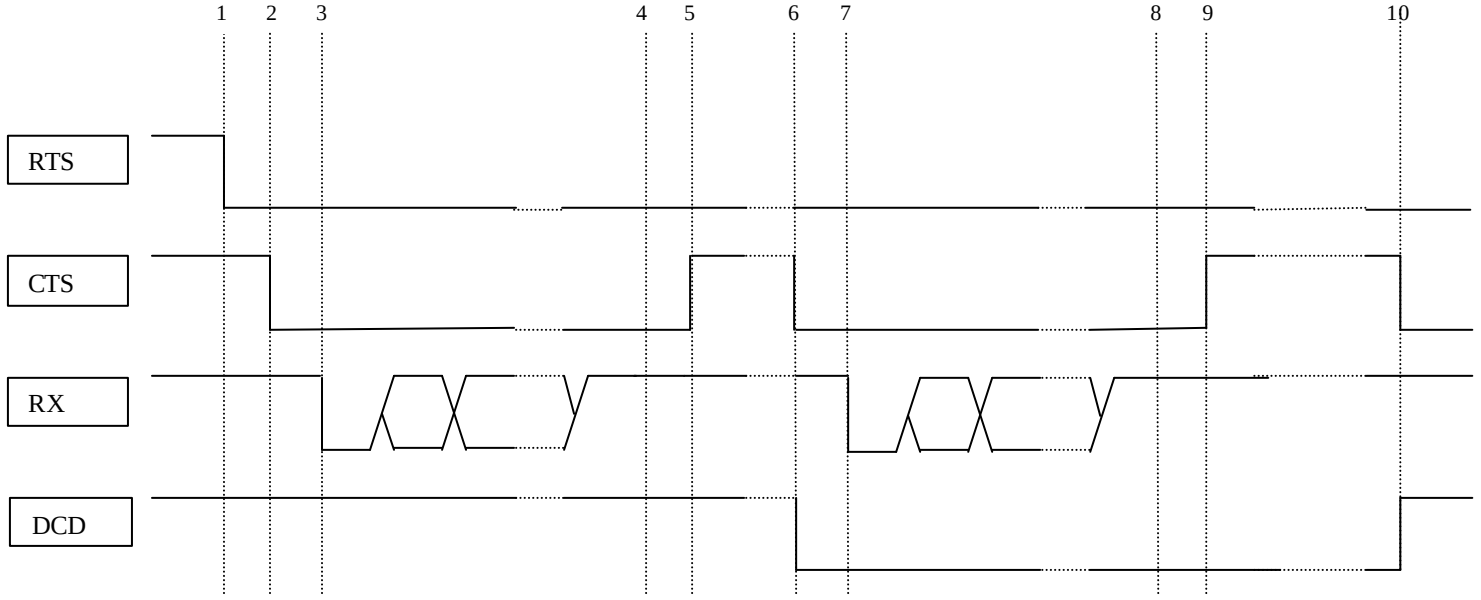
- User wants to enter <= 32 bytes
- DCD is inactive before entering bytes
- "RFm_Retries" > 0
- Data is successfully acked by destination-RFm



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).

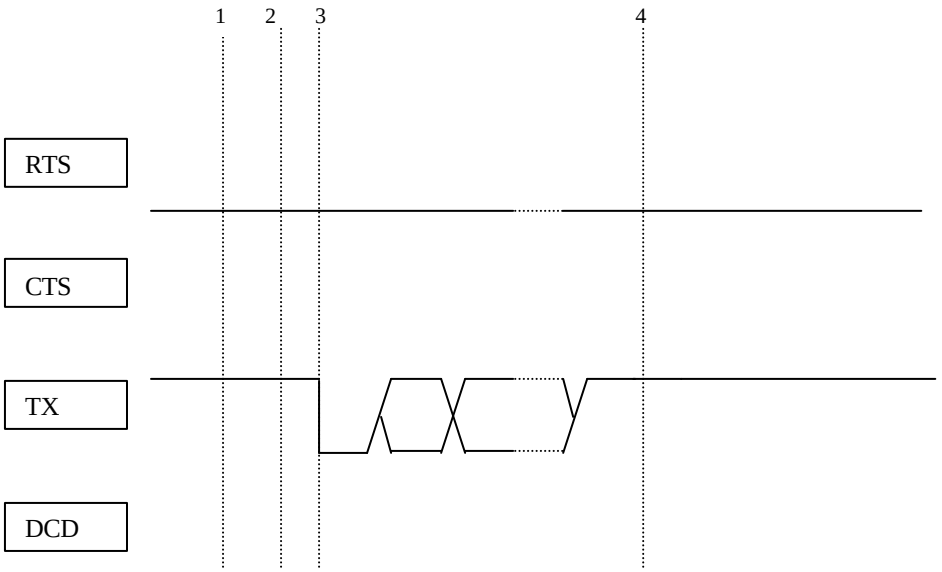
Detailed procedure RFm_x → User_x:

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 outstartbit of 1st data byte (or of source-address if it is master)
4. RFm has finished the stop-bit of the last data byte

(Another case: RTS stops and starts data stream on TX line by bringing RTS inactive/active)

Programming Mode

A number of commands ("Requests") can be entered to 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, except when the RFm is busy giving data to User or the RFm is busy transmitting a frame/waiting for ack. Note: If RFm_Retreies = 255, programming mode is not reachable before the last data packet is ack'ed.

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", but the DCD pin is not used in programming mode.

In addition, RFm can give "Indications" to User, and User can answer (with "Response"). Presently, "Indications" and "Response" are only available in binding mode (they are used in the association process, refer to section "Binding Mode").

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 primitives is different for an RFm_S and an RFm_M.
When transferring a primitive:
The first byte 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 primitive value, a number of parameters may be given (depends on primitive value).

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, a request can be entered by adding additional characters (any char will do) (the other method is to bring RTS inactive after the last byte of the request), then bringing RTS active again.

Format of the primitives

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 must be entered to make a total of 32 characters (where "*" is the 1st character)
2. The RTS line can be brought inactive after the parameter-field and then active again. The time in inactive state should be > 30msec.

The start-of-transfer character ('*') is used to reset the character-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, 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, 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, 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 "own ID" = 0x41414141 (ascii char: AAAA)
Start-of-transfer * => '*'
Type of primitive 121 => '1', '2','1'
Parameters:
ID_Source=Own_ID=1=0x01 => '0','1'
ID=0x41414141 => '4','1', '4','1', '4','1', '4','1'

Total transfer, in ascii-character notation: *1210141414141

Note: In Active mode, if User_Master wants to transmit a message to User_Slave connected to the RFm_Slave with address 0x41414141: User_M first enters the 4 bytes 0x41, 0x41, 0x41, 0x41 (= AAAA in ascii notation), then the data bytes to transfer to this slave.

Below: First, the primitives are summarized. Then every primitive is described in more detail. "Ascii value" means the 3 ascii characters to enter for the primitive. "Parameters": The ascii chars to enter are shown.

REQUEST Primitive	**Parameters**
Reset_Req()	-
Set_Type_Req(Type)	Type = Master, Slave
Get_Type_Req()	-
Set_Mode_Req(Mode)	Mode = Active, Binding, Promiscuous, Test1, ...Test*n*
Get_Mode_Req()	-
Set_ID_Req(ID_Source, ID)	ID_Source = Own_ID, Master_ID ID = Value of the selected ID
Get_ID_Req(ID_Source)	ID_Source = Own_ID, Master_ID
Set_Retries_Req(Retries)	Retries = Max number of retransmissions
Get_Retries_Req()	-

CONFIRM Primitive	**Parameters**
Type_Cnf(Type)	Type = Master, Slave
Mode_Cnf(Mode)	Mode = Active, Binding, Promiscuous, Test1, ... Test*n*
ID_Cnf(ID)	ID = Value of the requested ID
Retries_Cnf(Retries)	Retries = Max number of retransmissions

INDICATION Primitive	**Parameters**
Slave_Bind_Ind(RF_ID, User_Field_Length, User_Field)	RF_ID= Unique RF_ID of the RFm_S sending the request User_Field_Length / User_Field: Parameters from the User_S
Bind_Ack_Ind(RF_ID, Short_Address, User_Field_Length, User_Field)	RF_ID = Unique RF_ID of the RFm_M responding to the request Short_Address = Network-unique short-address assigned to the slave by the User_M User_Field_Length / User_Field: Parameters from the User_M

RESPONSE Primitive	**Parameters**
Accept_Slave_Rsp(RF_ID, Short_Address, User_Field_Length, User_Field)	RF_ID= Unique RF_ID of the RFm_S sending the request Short_Address = Network-unique short-address assigned to the slave by the User_M User_Field_Length / User_Field: Parameters from User_M
Reject_Slave_Rsp(RF_ID)	RF_ID = Unique RF_ID of the RFm_S sending the request
Accept_Master_Rsp(RF_ID)	RF_ID =
	Unique RF_ID of the RFm_M
Reject_Master_Rsp(RF_ID)	RF_ID =
	Unique RF_ID of the RFm_M

Note:

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 (not “reset to default”). 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.

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: 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: Set_ID_Req(ID_Source, ID)

Ascii value:	121	
Parameters:	ID_Source	
	01	Set my Own ID
	02	Set my Master's ID
	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
To RFm Master/Slave:	Both	
Expected confirm from RFm:	ID_Cnf(ID)	
Comments:		

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: 0 => no ack/retransmissions are done 255 => retransmissions until ack'ed or power-off (255 should be used with care)	

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: 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: 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: 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:		

Primitive: Slave_Bind_Ind(RF_ID, User_Field_Length, User_Field)

Ascii value:	301	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_S sending the request
	User_Field_Length	
	n	Length (# bytes) from User_S in User_Field
	User_Field	
	xxx…xx	User data from User_S
To User Master/Slave:	To User_M only	
Expected response from user:	Accept_Slave_Rsp(RF_ID, …) or Reject_Slave_Rsp(RF_ID)	
Comments:		

Primitive: Bind_Ack_Ind(RF_ID, Short_Address, User_Field_Length, User_Field)

Ascii value:	311	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_M responding to the request
	Short_Address	
	n	Network-unique short-address assigned to the slave by the User_M
	User_Field_Length	
	n	Length (# bytes) from User_M in User_Field
	User_Field	
	xxx…xx	User data from User_M
To User Master/Slave:	To User_S only	
Expected response from user:	Accept_Master_Rsp(RF_ID) or Reject_Master_Rsp(RF_ID)	
Comments:		

Primitive: Accept_Slave_Rsp(RF_ID, Short_Address, User_Field_Length, User_Field)

Ascii value:	401	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_S sending the request
	Short_Address	
	n	Network-unique short-address assigned to the slave by the User_M
	User_Field_Length	
	n	Length (# bytes) from User_M in User_Field
	User_Field	
	xxx…xx	User data from User_M
To RFm Master/Slave:	To RFm_M only	
Result of indication from RFm:	Slave_Bind_Ind(RF_ID, ...)	
Comments:		

Primitive: Reject_Slave_Rsp(RF_ID)

Ascii value:	402	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_S sending the request
To RFm Master/Slave:	To RFm_M only	
Result of indication from RFm:	Slave_Bind_Ind(RF_ID, ...)	
Comments:		

Primitive: Accept_Master_Rsp(RF_ID)

Ascii value:	411	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_M responding to the request
To RFm Master/Slave:	To RFm_S only	
Result of indication from RFm:	Bind_Ack_Ind(RF_ID, ...)	
Comments:		

Primitive: Reject_Master_Rsp(RF_ID)

Ascii value:	412	
Parameters:	RF_ID	
	xxxx	Unique RF_ID of the RFm_M responding to the request
To RFm Master/Slave:	To RFm_S only	
Result of indication from RFm:	Bind_Ack_Ind(RF_ID, ...)	
Comments:		

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:

Active mode
For data transfer between master and slave users

Binding mode
For master-slave association

Special modes: "Sniffer", "Testmodes"
For debugging and testing the RFm_x

Active mode

Active mode is also referred to as "traffic 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 (assuming User_M and User_S are "associated" through a binding mode operation or by using the Set_ID_Req(...) request).

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".

ARQ: If the value of parameter "RFm_Retries" > 0, a transmitted frame must 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. 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.

Binding Mode

User_M keeps a list of the associated User_S devices. In addition, all User_S devices must know which User_M they are connected to.

In "binding mode", slaves are associated with a master. User_M and User_S have different tasks in this process. Refer to "User_M Action" and "User_S Action" below.

In binding mode, only "binding-operations" are enabled. No data from previously bound slaves will get through. CTS/RTS must be used as described in "Use of RTS/CTS and DCD". DCD pin is not used in binding mode.

User_M Action

- Enter programming mode (refer to section "Programming")
- Program the RFm_M mode of operation to "Binding mode"
- Exit programming mode
- RFm_M starts to search for binding requests from slaves when RFm_M exits programming mode
- RFm_M will now be in binding mode until User_M changes the mode of operation

If RFm_M gets a binding request from a RFm_S, the request is transferred to User_M.

The frame given to User_M (this type of primitive is called an "Indication"):

- An "Indication_Type" field indicating "this is a binding request from a slave"
- Unique RF_ID of the RFm_S sending the request
- Parameters from the User_S connected to the RFm_S

Indication_Type = "Slave_Bind_Ind"	Unique RF_ID of the new slave	Length of User_field (data from User_S to User_M)	User field, content is transparent to RFm_x
1 byte	4 bytes	1 byte	n byte(s)

Note: In every byte, there are 8 bit.

User_M determines if this is an OK new slave. If it is OK, User_M assigns a short-address of 1 byte (a number 1...64) to the new slave. This will be the slave's address in the network. At a later time, when User_M wants to transfer data to this User_S, User_M first enters the short-address into RFm_M, and then the data to transmit.

If slave is accepted, User_M gives the following frame to RFm_M (this type of primitive is called an "Response"):

Response_Type = "Accept_Slave_Rsp"	Unique RF_ID of the new slave	Short-address assigned to the new slave	Length of User_field (data from User_M to User_S)	User field, content is transparent to RFm_x
1 byte	4 bytes	1 byte	1 byte	n byte(s)

Response_Type = "Reject_Slave_Rsp"	Unique RF_ID of the new slave
1 byte	4 bytes

User_Slave action

- Enter programming mode (refer to section “Programming”)
- Program the RFm_S mode of operation to “Binding mode”
- (Optional) Program the RFm_S with the user-bytes to transfer to User_M as a part of the binding. Example: Serial number/device type or other user info. Refer to section “Programming” for a detailed description.
- Exit programming mode
- RFm_S starts to send binding-requests when RFm_S exits programming mode
- RFm_S will now be in binding mode until User_S changes the mode of operation
- If 2 or more RFm_S are in binding mode at the same time: Every RFm_S will compete for master’s attention. One of the RFm_S will win, and the binding process will be completed for this one before another slave gets master’s attention.

If the bind request from RFm_S is acknowledged (by an User_M), the acknowledge info is transferred to User_S.

The frame given to User_S (this type of primitive is called an “Indication”):

An “Indication_Type” field indicating “a master is responding to the binding request”
Unique RF_ID of the RFm_M responding to the request
Network-unique short-address assigned to the slave by the User_M
Parameters from the User_M

Indication_Type = “Bind_Ack_Ind”	Unique RF_ID of the RFm_M	Unique short-address to use in the network	Length of User_field (data from User_M to User_S)	User field, content is transparent to RFm_x
1 byte	4 bytes	1 byte	1 byte	n byte(s)

The User_S determines if this is an OK master. If master is accepted, User_S gives the following frame to RFm_S (this type of primitive is called an “Response”):

Response_Type = “Accept_Master_Rsp”	Unique RF_ID of the RFm_M
1 byte	4 bytes

If not OK, User_S will give a “reject” response to the RFm_S. If master is not accepted:

Response_Type = “Reject_Master_Rsp”	Unique RF_ID of the RFm_M
1 byte	4 bytes

After receiving this response, the RFm_S restarts the association process.

Other modes of operation

"Sniffer mode"

A module operating in sniffer mode must first be associated with the master. Then RFm_x must be programmed to "snifffer mode" through the "Set_Mode_Req(Mode)" request, with Mode = "Promiscuous "). Refer to "Programming Mode".

The RFm give user all received RF messages.

"Test modes"

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

Test1:

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 frame is <= 36, the bytes are given to User. The DCD pin is inverted every time an OK frame is received.

Test2:

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:

The radio chip is programmed to TX mode and transmits a "1010…" signal on this frequency. This can be used for deviation measurements.

Test 4:

The radio chip is programmed to TX mode and transmits "test-packets" on this 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.