Working ME218C Communications Protocol

Spring 2013

May 10, 2013

Communication Committee

Revision History

Date	What	Who
May 8, 2013	Initial draft of protocol	Communications Committee
May 9, 2013	Revised draft of protocol	Michael Bunne, …
May 10, 2013	State diagrams updated	David Stonestrom
May 10, 2013	State diagrams updated. Corrected Typos	Ramanan Sampath
May 10, 2013	Undid state diagram changes, fixed actual state diagram errors, added explanation	David Stonestrom
May 10, 2013	Finalized for second submission	

Overview

This communications protocol details the communications for ME218C in spring 2013. The purpose is to allow each team's POD (creative input device) to control every other team's ROAMER (mobile platform with gripper to move safety hatch and open airlock).

This communication works over a wireless Xbee network. The PODs request to take over individually numbered (1-3) ROAMERs through a broadcast transmission. If a ROAMER is not connected to another POD, it allows incoming POD requests to connect. When the POD is about to get too far out of orbit, it disconnects from the ROAMER, allowing the next POD to connect to it and take control. While connected, ROAMERs send back their status each time they receive a message directed at them from the correct POD.

If either the ROAMER or the POD misses five messages in a row (one second of communication) it will treat the connection as lost.

Note for the next few weeks

For testing purposes, use a ROAMER number of (Team # + 3) unless you are deliberately testing on ROAMER number 1-3 for interoperability. This is to avoid one group accidently driving another group's ROAMER off the work bench.

State Charts

There is an important distinction for working on the events and services framework:

- Messages are the full Xbee protocol 13 or 14 bytes that start with 0x7E and end with the checksum
- Message Events are ES_Event events which tell state machines about messages

On each of the POD and the ROAMER, there are three state machines running to handle communication, as well as interrupt responses for the asynchronous transmit and receive interrupts (this can be done with polling if you prefer).

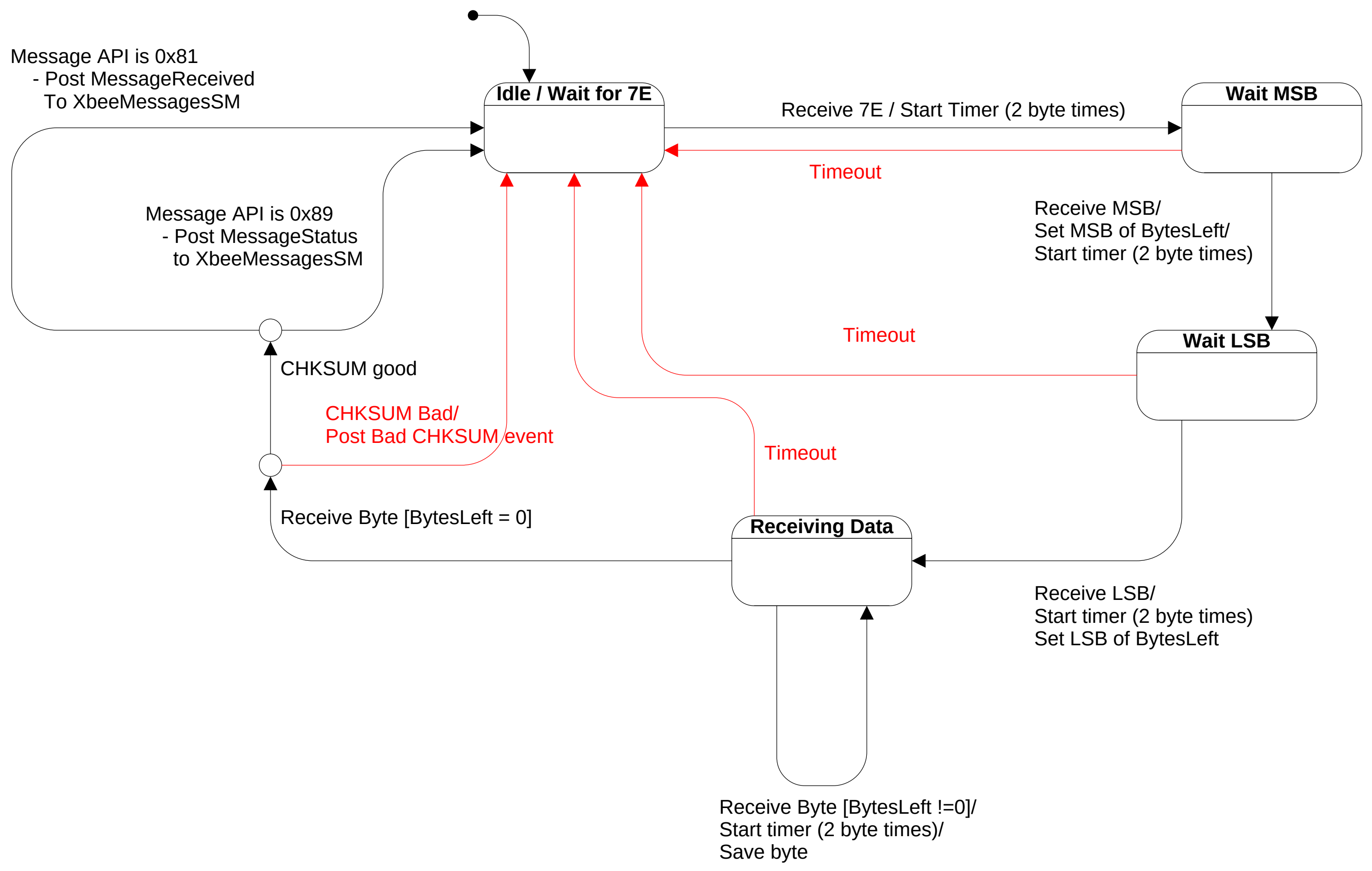
The interrupt response for receive is always active. It pulls in bytes as they arrive and interacts directly with the AsyncReceiveSM to compose incoming messages from the incoming bytes. After a correct and complete message has arrived, there should be a post to one of the other two state machines. The XbeeMessagesSM should be alerted about message status replies from the Xbee with the event MessageStatus. Incoming messages from another Xbee should be sent along to the CommunicationSM for the device with the event MessageReceived. In each case, the events and services framework will not pass the whole message in an event, so you will need to make your own arrangements for passing the message around.

The interrupt response for transmit is only enabled when the XbeeMessagesSM is in particular states, and only until it has finished sending one message. As it transfers the checksum to the asynchronous hardware, the interrupt should disable itself.

The XbeeMessagesSM on each device is responsible for getting each message sent. It is moved from its idle state by the device's CommunicationSM posting TransmitMessage or BroadcastMessage. No two of these posts should come more frequently the 5Hz due to the structure of the CommunicationSM. Once a TransmitMessage or BroadcastMessage event has been posted to the XbeeMessagesSM, it will attempt to send the message three times before declaring failure. It is up to each group to handle the debugging response for losing a message. Due to the slow message rate, there should be no issues with the XbeeMessagesSM finishing its three attempts before another message needs to be sent.

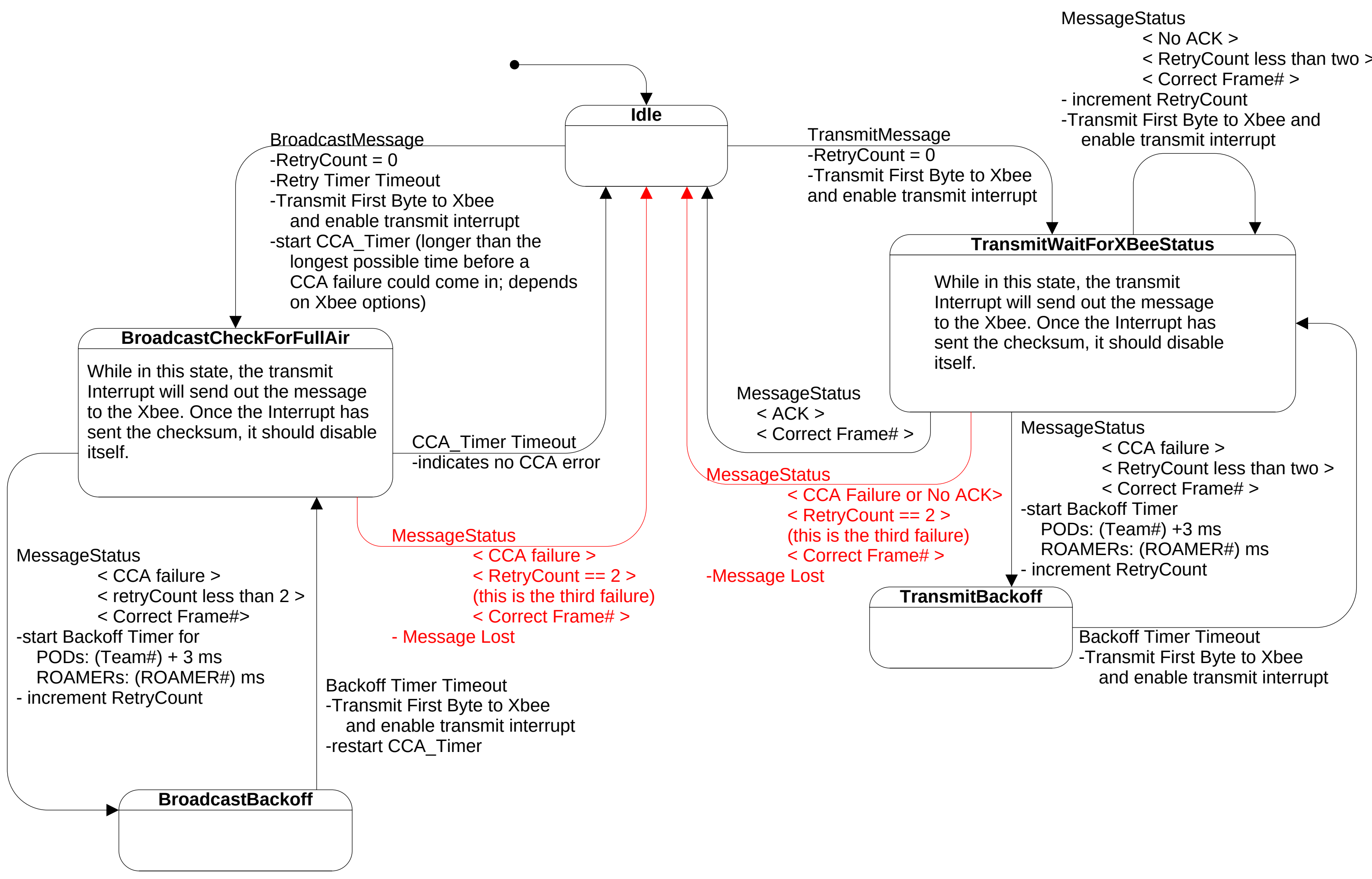
The CommunicationSM for each device is responsible for the highest level of the communication protocol. It should get MessageReceived events when the Xbee receives a message, and should post to the XbeeMessagesSM when it wants to send messages. It is responsible for handling connection requests, disconnect requests, ROAMER status messages, and POD command transmissions between PODs and ROAMERs.

Async Receive State Machine:



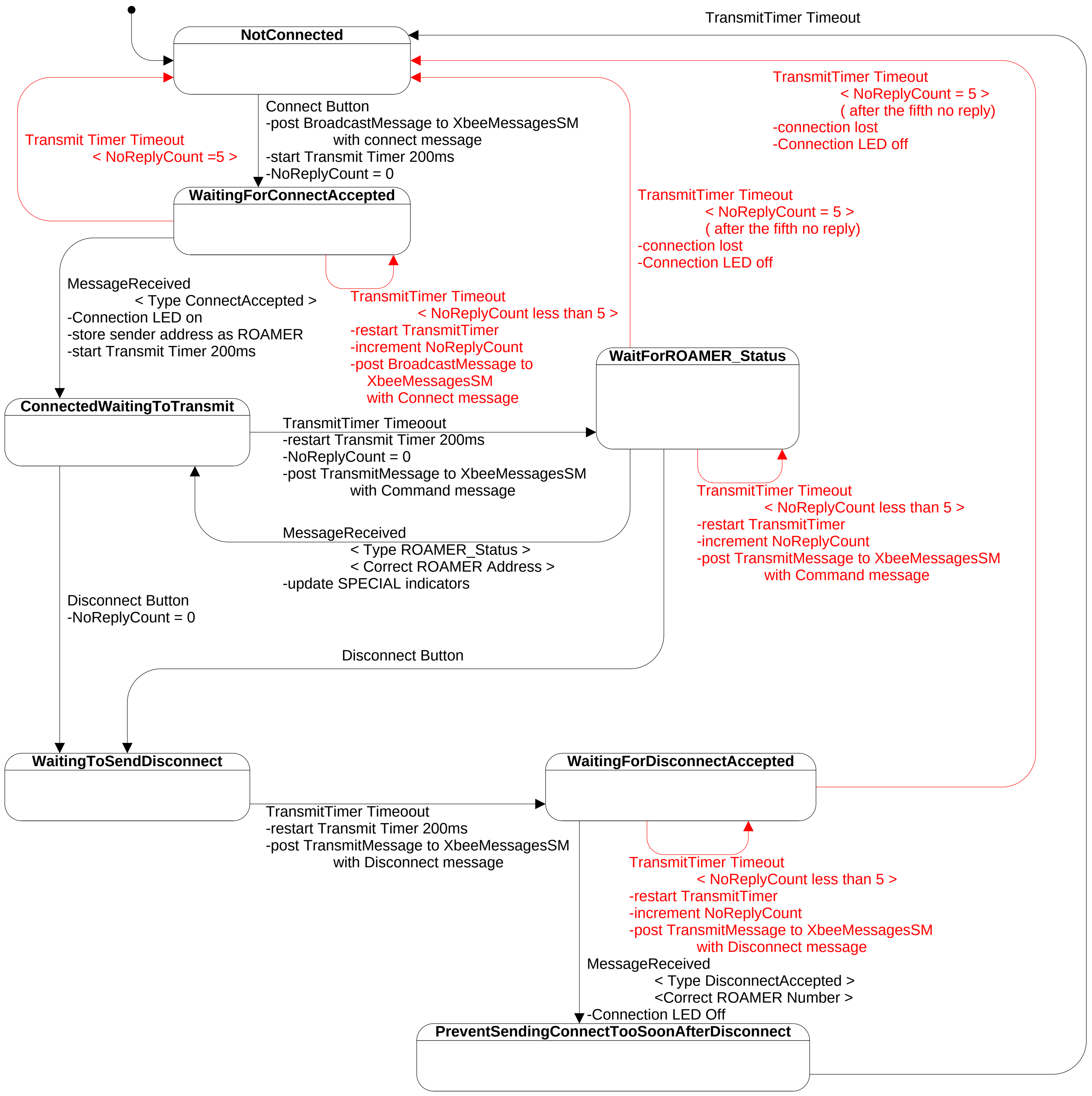
This is a simple state machine to process incoming messages. It should work with the async receive interrupt, putting bytes in the correct location as they come in. Once the message is complete and the checksum is checked, the state machine should route the message to either the XbeeMessagesSM or the CommunicationSM. Red transitions represent error is receiving the message, and should be handled for debugging.

POD Xbee Messages State Machine:


Idle
BroadcastMessage
-RetryCount = 0
-Retry Timer Timeout
-Transmit First Byte to Xbee
and enable transmit interrupt
-start CCA_Timer (longer than the
longest possible time before a
CCA failure could come in; depends
on Xbee options)
TransmitMessage
-RetryCount = 0
-Transmit First Byte to Xbee
and enable transmit interrupt
MessageStatus
< No ACK >
< RetryCount less than two >
< Correct Frame# >
- increment RetryCount
-Transmit First Byte to Xbee and
enable transmit interrupt
TransmitWaitForXBeeStatus
While in this state, the transmit
Interrupt will send out the message
to the Xbee. Once the Interrupt has
sent the checksum, it should disable
itself.
BroadcastCheckForFullAir
While in this state, the transmit
Interrupt will send out the message
to the Xbee. Once the Interrupt has
sent the checksum, it should disable
itself.
CCA_Timer Timeout
-indicates no CCA error
MessageStatus
< ACK >
< Correct Frame# >
MessageStatus
< CCA Failure or No ACK>
< RetryCount == 2 >
(this is the third failure)
< Correct Frame# >
-Message Lost
MessageStatus
< CCA failure >
< RetryCount less than two >
< Correct Frame# >
-start Backoff Timer
PODs: (Team#) +3 ms
ROAMERs: (ROAMER#) ms
- increment RetryCount
MessageStatus
< CCA failure >
< RetryCount == 2 >
(this is the third failure)
< Correct Frame# >
- Message Lost
TransmitBackoff
Backoff Timer Timeout
-Transmit First Byte to Xbee
and enable transmit interrupt
MessageStatus
< CCA failure >
< retryCount less than 2 >
< Correct Frame#>
-start Backoff Timer for
PODs: (Team#) + 3 ms
ROAMERs: (ROAMER#) ms
- increment RetryCount
Backoff Timer Timeout
-Transmit First Byte to Xbee
and enable transmit interrupt
-restart CCA_Timer
BroadcastBackoff

The receive interrupt should post MessageStatus events to the XbeeMessagesSM when the incoming message API is a message status (0x89). The TransmitMessage and BroadcastMessage events come from the POD_CommunicationSM. The 5Hz limit on transmit rates and the structure of the state machines means you should never be posting these events when the XbeeMessagesSM is not in the idle state, but it is up to each group to implement their own error checking and debugging. The designed function of this state machine is that it ignores the second message posted.

POD Communication State Machine

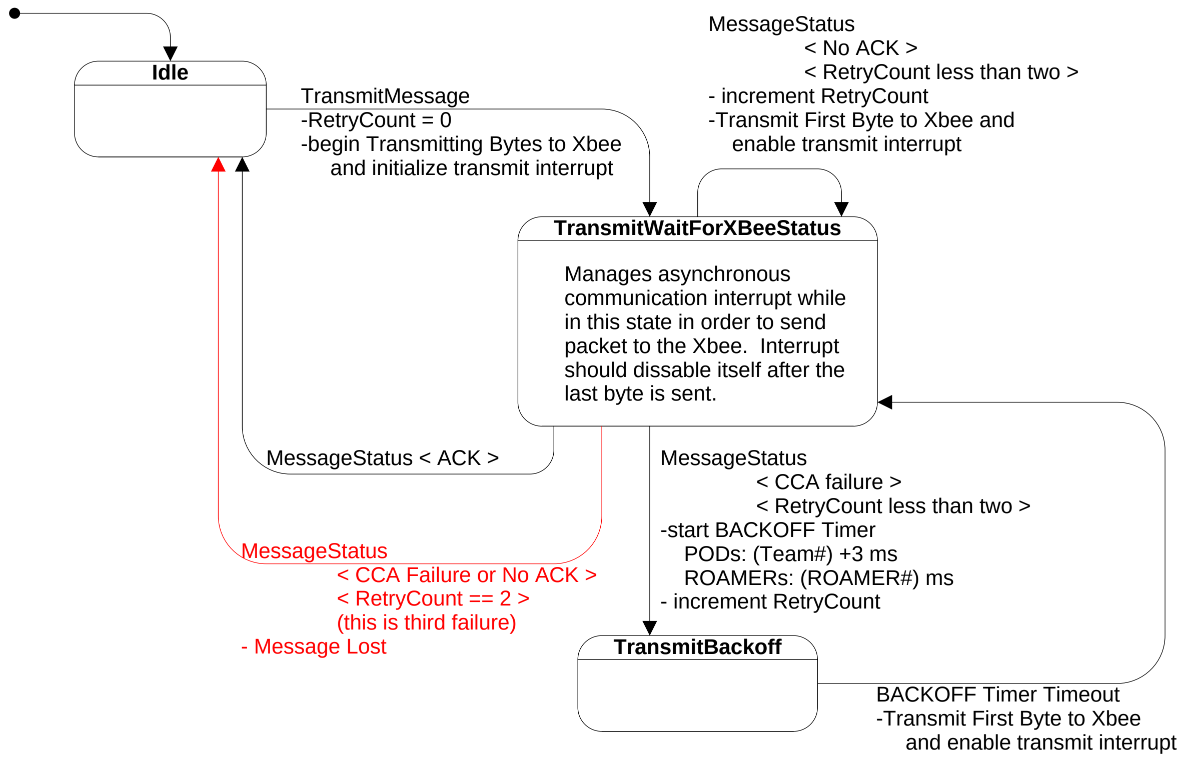


NotConnected
TransmitTimer Timeout
TransmitTimer Timeout < NoReplyCount = 5 > (after the fifth no reply) -connection lost -Connection LED off
Transmit Timer Timeout < NoReplyCount =5 >
Connect Button -post BroadcastMessage to XbeeMessagesSM with connect message -start Transmit Timer 200ms -NoReplyCount = 0
WaitingForConnectAccepted
TransmitTimer Timeout < NoReplyCount = 5 > (after the fifth no reply) -connection lost -Connection LED off
MessageReceived < Type ConnectAccepted > -Connection LED on -store sender address as ROAMER -start Transmit Timer 200ms
TransmitTimer Timeout < NoReplyCount less than 5 > -restart TransmitTimer -increment NoReplyCount -post BroadcastMessage to XbeeMessagesSM with Connect message
WaitForROAMER_Status
ConnectedWaitingToTransmit
TransmitTimer Timeout -restart Transmit Timer 200ms -NoReplyCount = 0 -post TransmitMessage to XbeeMessagesSM with Command message
TransmitTimer Timeout < NoReplyCount less than 5 > -restart TransmitTimer -increment NoReplyCount -post TransmitMessage to XbeeMessagesSM with Command message
MessageReceived < Type ROAMER_Status > < Correct ROAMER Address > -update SPECIAL indicators
Disconnect Button -NoReplyCount = 0
Disconnect Button
WaitingToSendDisconnect
WaitingForDisconnectAccepted
TransmitTimer Timeoout -restart Transmit Timer 200ms -post TransmitMessage to XbeeMessagesSM with Disconnect message
TransmitTimer Timeout < NoReplyCount less than 5 > -restart TransmitTimer -increment NoReplyCount -post TransmitMessage to XbeeMessagesSM with Disconnect message
MessageReceived < Type DisconnectAccepted > <Correct ROAMER Number > -Connection LED Off
PreventSendingConnectTooSoonAfterDisconnect

The receive interrupt should post MessageReceived to the POD_CommunicationSM when it receives an incoming message API (0x81). The POD Communication SM will ignore any other POD's messages due to the message type guard conditions. It will also ignore any ROAMER except the one it is in control of due to the ROAMER address guard condition. If two ROAMERs share a ROAMER number, the POD will only command the first to respond, and the second one will have to time out in order to get disconnected from the POD it thinks is controlling it.

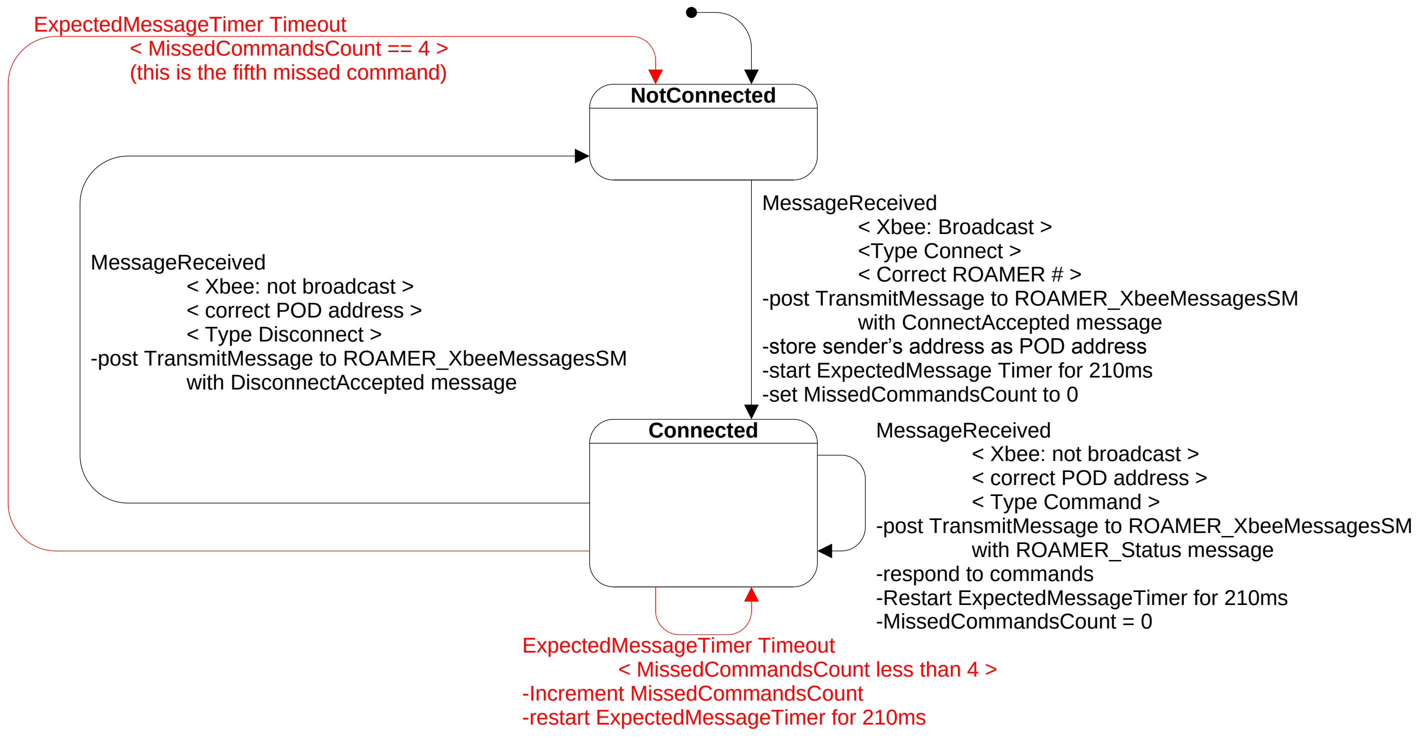
ROAMER Xbee Messages State Machine

This is just the POD version without the loop for handling broadcast messages.


Idle
TransmitMessage
-RetryCount = 0
-begin Transmitting Bytes to Xbee
and initialize transmit interrupt
MessageStatus
< No ACK >
< RetryCount less than two >
- increment RetryCount
-Transmit First Byte to Xbee and
enable transmit interrupt
TransmitWaitForXBeeStatus
Manages asynchronous communication interrupt while in this state in order to send packet to the Xbee. Interrupt should dissable itself after the last byte is sent.
MessageStatus < ACK >
MessageStatus
< CCA failure >
< RetryCount less than two >
-start BACKOFF Timer
PODs: (Team#) +3 ms
ROAMERs: (ROAMER#) ms
- increment RetryCount
MessageStatus
< CCA Failure or No ACK >
< RetryCount == 2 >
(this is third failure)
- Message Lost
TransmitBackoff
BACKOFF Timer Timeout
-Transmit First Byte to Xbee
and enable transmit interrupt

The receive interrupt should post MessageStatus events to the XbeeMessagesSM when the incoming message API is a message status (0x89). The TransmitMessage event comes from the ROAMER_CommunicationSM. The 5Hz limit on transmit rates and the structure of the state machines means you should never be posting these events when the XbeeMessagesSM is not in the idle state, but it is up to each group to implement their own error checking and debugging. The designed function of this state machine is that it ignores the second message posted.

ROAMER Communication State Machine


ExpectedMessageTimer Timeout
< MissedCommandsCount == 4 >
(this is the fifth missed command)
NotConnected
MessageReceived
< Xbee: Broadcast >
<Type Connect >
< Correct ROAMER # >
-post TransmitMessage to ROAMER_XbeeMessagesSM
with ConnectAccepted message
-store sender's address as POD address
-start ExpectedMessage Timer for 210ms
-set MissedCommandsCount to 0
MessageReceived
< Xbee: not broadcast >
< correct POD address >
< Type Disconnect >
-post TransmitMessage to ROAMER_XbeeMessagesSM
with DisconnectAccepted message
Connected
MessageReceived
< Xbee: not broadcast >
< correct POD address >
< Type Command >
-post TransmitMessage to ROAMER_XbeeMessagesSM
with ROAMER_Status message
-respond to commands
-Restart ExpectedMessageTimer for 210ms
-MissedCommandsCount = 0
ExpectedMessageTimer Timeout
< MissedCommandsCount less than 4 >
-Increment MissedCommandsCount
-restart ExpectedMessageTimer for 210ms

The receive interrupt should post MessageReceived to the ROAMER_CommunicationSM when it receives an incoming message API (0x81). The ROAMER will ignore anything except Connect requests when not connected. It will then respond to the first POD to send a connection request with its ROAMER number. Once connected, it will ignore any connection requests and any messages not from its POD.

Byte Specifications, Overall Format
This protocol is meant to be included inside of an Xbee packet. A brief explanation of the Xbee format follows. Our data packets are included in the Data section.

Sending a Packet of Data
When you want to tell your Xbee to send a message into the world:

Start Delimiter	Length HI	Length LO	API ID	Frame	Target Addr HI	Target Addr LO	Options	Data <6 or 5 bytes>	Checksum

Start Delimiter:
0x7E for all communications

Length:
HI – 0x00 for all communications

LO – 0x0B if sending from POD
– 0x0A if sending from ROAMER

API ID:
0x01 for all outbound communications

Frame ID:
Start at any number of your choosing, and increment with each sending operation
-Note: if you allow the Frame ID to be 0, it will disable the response frame from your Xbee to your PIC (you won't get a message with API ID of 0x89 from your Xbee for that command)

Target Address:
For PODs:
If NOT already connected to a ROAMER: Set to 0xFFFF to broadcasting a "Connect" message
If already connected to a ROAMER: Set both bytes to Address of that ROAMER
-Note: Every time you receive a response from a ROAMER, your Xbee will include the Source Address of the response. This is the Target Address that you should use to communicate directly to that ROAMER for all future messages until Disconnection occurs.
For ROAMERs:
If NOT already connected to a POD: You should not be sending responses
If already connected to a POD: Set both bytes to Address of that POD
-Note: Every time you receive a message from a POD, your Xbee will include the Source Address of the message. This is the Target Address that you should use to communicate directly to that POD for all future responses until Disconnection occurs. **The ROAMER sends messages only in response to messages from the POD. It does not initiate a transmission (a broadcast) on its own.**

Options:
0x00 for all outbound communications

Data:
6 bytes if sending from POD (Type, Message bytes 1-5)
5 bytes if sending from ROAMER (Type, Message bytes 1-4)

Data Packet types: POD to ROAMER

"Connect" to ROAMER:

0x00	ROAMER # (0x01, 0x02, or 0x03)	0x00	0x00	0x00	0x00

-The ROAMER # should be controlled by an input on your POD and the command will be broadcasted to all devices activated. The ROAMER whose # matches your request (also indicated by a switch on the ROAMER) will reply (with an "Accepted Connection" response) and then you will proceed to talk only to that ROAMER based on the Source Address of that message.

-Note: Other PODs activated will also receive this message. It is the duty of each POD to simply ignore messages from other PODs.

"Disconnect" from ROAMER:

0x01	0x00	0x00	0x00	0x00	0x00

-The ROAMER must still send a reply of "Disconnect Accepted" to acknowledge the disconnect. There is also a predetermined period of no response time that will also signal a disconnect. If any device (ROAMER or POD), while believing to be connected, experiences 5 full cycles (running at 200ms each = 1 sec) without receiving message or response from the other side, then the device will assume the connection has been broken and will return to their Disconnected state.

Sending a "Command" to ROAMER:

0x02	Left Wheel	Right Wheel	Gripper	Camera	Digital I/Os

-Left Wheel:

0x00 – Left Wheel Full Reverse
0x40 – suggested cutoff for ROAMERs which only drive at full speed
0x80 – Left Wheel stopped
0xC0 – suggested cutoff for ROAMERs which only drive at full speed
0xFF – Left Wheel Full Forward

-Right Wheel:

0x00 – Right Wheel Full Reverse
0x40 – suggested cutoff for ROAMERs which only drive at full speed
0x80 – Right Wheel stopped
0xC0 – suggested cutoff for ROAMERs which only drive at full speed
0xFF – Right Wheel Full Forward

-Gripper:

Bit 7 (MSB):	Digital bit devoted to Gripper Actuation #1
Bits 0-6:	Analog bits devoted to Gripper Actuation #2

-Note: Each POD should have two inputs for the gripper (one for each potential actuation), but neither of these inputs necessarily have to be analog. If a group decides to use 2 digital inputs for the gripper control, they may. The 2nd digital input will be sent as the 7 analog bits railed high or railed low, and it is up to the ROAMER to interpret this however necessary to complete the desired action.

-Camera:

0x00 – From the Camera's Perspective, turned LEFT from default, 180° from default position
0x40 – suggested cutoff for ROAMERs that turn camera LEFT with digital response
0x80 – Camera Default Position
0xC0 – suggested cutoff for ROAMERs that turn camera RIGHT with digital response

0xFF – From the Camera's perspective, turned RIGHT from default, 180° from default position

-Note: for ROAMERs and PODs with only 2 camera positions or only 1 direction that the camera turns from default, use 0x00 to 0x80 regardless of camera orientation

-Digital I/Os:

Bit 0: Digital bit devoted to controlling SPECIAL's lower RIGHT light

Bit 1: Digital bit devoted to controlling SPECIAL's upper CENTER light

Bit 2: Digital bit devoted to controlling SPECIAL's lower LEFT light

Bits 3-7 (MSB): Digital bits devoted to any extra (non-mission critical) commands that may be unique to a particular ROAMER (fans, beacons, self destruct sequence, Harlem Shake). PODs with fewer than 5 additional inputs should route whatever inputs they have to the address starting with Bit 3 (i.e. the most critical of any extra functions a ROAMER can perform should be controlled by Bit 3 with lesser important functions being controlled by successively higher bits)

-Note: while it is not required for any POD to have all 5 additional inputs for these optional extra commands, it is STRONGLY encouraged (as in nearly required) to provide at least 1 of these inputs.

Data Packet types: ROAMER to POD

Sending "ROAMER Status" to POD:

0x03	SPECIAL Battery %	SPECIAL charging/discharging status	SPECIAL LED states	Last Camera Command

-The 3 bytes corresponding to the SPECIAL are exactly the same as generated from the SPECIAL when queried from the ROAMER's PIC (it is simply passing this data back to the POD). The last byte is information about the previous camera position that can be used by the POD if the camera control is rate-based (as opposed to position based).

Sending "Connect Accepted" to POD:

0x04	SPECIAL Battery %	SPECIAL charging/discharging status	SPECIAL LED states	Last Camera Command

-The ROAMER will always send the 3 bytes corresponding to the SPECIAL, and will simply change the Type (1st bit) to indicate that it is responding to a Connection, Disconnection, or a standard Command.

Sending "Disconnect Accepted" to POD:

0x05	SPECIAL Battery %	SPECIAL charging/discharging status	SPECIAL LED states	Last Camera Command

-The ROAMER will always send the 3 bytes corresponding to the SPECIAL, and will simply change the Type (1st bit) to indicate that it is responding to a Connection, Disconnection, or a standard Command.

Checksum:

-Standard Checksum procedure for talking to the Xbee

-There is no additional Checksum to maintain congruence from PIC to PIC (only the standard PIC to Xbee will be used)

Result from a Transmit Packet

After you tell your Xbee to send a message into the world, it will (almost) immediately reply with:

Start Delimiter	Length HI	Length LO	API ID	Frame	Status	Checksum

Start Delimiter:

0x7E for all replies

Length:

0x03 for all replies

API ID:

0x89 for all replies

-Note: For broadcasts, there is only a 0x89 if the message did not send. If you successfully broadcast a message, expect no 0x89 back from your Xbee.

Frame ID:

The same Frame ID of the message you just sent the Xbee

-Note: if you allow the Frame ID to be 0, it will disable the response frame from your Xbee to your PIC (you won't get a message with API ID of 0x89 from your Xbee for that command)

Status:

0x00 = Success
0x01 = No ACK received (meaning it failed to transmit the message, after multiple retries)
0x02 = CCA failure
0x03 = Purged

Checksum:

-Standard Checksum procedure for talking to the Xbee
-There is no additional Checksum to maintain congruence from PIC to PIC (only the standard PIC to Xbee will be used)

An Incoming Packet

When your Xbee receives a message addressed to you (or broadcasted) from the world, it sends you:

Start Delimiter	Length HI	Length LO	API ID	Source Addr HI	Source Addr LO	RSSI	Options	Data <6 or 5 bytes>	Checksum

Start Delimiter:

0x7E for all communications

Length:

HI – 0x00 for all communications

LO – 0x0B if packet is from POD
– 0x0A if packet is from ROAMER

API ID:

0x81 for all incoming communications sent from the outside world (not our Xbee responding to a command we just tried to send)

Source Address:

These two bytes represent the unique address of the device that just sent you the message. If you determine that you wish to communicate with this device directly in the future, you should probably record these bytes and set them as the Target Address bytes for your next Outgoing Packet

RSSI (Received Signal Strength Indicator):

Hexadecimal equivalent of signal strength (probably not important for our purposes)

Options:

0x01 = Address broadcast
0x02 = PAN broadcast

Data:

6 bytes if received from POD (Type, Message bytes 1-5)
5 bytes if received from ROAMER to POD (Type, Message bytes 1-4)