Communication client-datanetwork-minibees

Marije Baalman

September 20, 2011

1 Communication between DataNetwork host and MiniBee nodes

Several stages of sending data to the MiniBee:

1. Client to DataNode
2. DataNode to MiniBee
3. Host (server) to node

On one hand mapping of DataNode to MiniBee node, host should be repeating the sending of the data at a regular interval, so that messages are still arriving, even if they may be lost depending on the network density.

So, the MiniBee firmware has to check for:

- type of message (pulse width modulation, or digital out)
- node ID (is this for me?)
- msg ID (did I already parse this message?)

2 Wireless configuration - startup sequence of MiniBee nodes

1. Wake up XBee
2. Read SH and SL
3. Send SH and SL to coordinator (msgtype 's')
4. Coordinator sends node ID (msgtype 'I') with an optional configuration ID. If so, the board will receive a new configuration.
5. *if new config* Node sends waiting message (msgtype 'w').

description	type	data	sender
Data output	'O'	node ID + msg ID + N values	server
Data	'd'	node ID + msg ID + N values	node
Active	'a'	node ID + msg ID	node
Pausing	'p'	node ID + msg ID	node
Loopback	'L'	node ID + msg ID + onoff	server
Running	'R'	node ID + msg ID + onoff	server
Custom message	'E'	node ID + msg ID + (*data*)	server
Announce	'A'		server
Quit	'Q'		server
Serial number	's'	Serial High (SH) + Serial Low (SL) +	
		+ library version + board revision + capabilities	node
ID assignment	'I'	msgID + SH + SL + node ID + (*config ID*)	server
Wait for config	'w'	node ID + config ID	node
Configuration	'C'	msg ID + node ID + config ID + configuration bytes	server
Confirm config	'c'	node ID + config ID + smpMsg + msgInt +	
		+ datasize + outsize + (*custom*)	node
(*custom*)		customInputs + customDataSize + N x (custom pin, data size)	

Table 1: Message protocol between host and MiniBee nodes. Type is preceded by the escape character (92), and messages are delimited with the delimiter character (10). Furthermore character 13 needs to be escaped as well. General convention: server message in capital, node message small.

6. *if new config* Wait for configuration message.
7. *if new config* Write new config to EEPROM and confirm received configuration message.
8. Load config from EEPROM.
9. Node sends config summary (msgtype 'c').
10. Start sending data, or wait for receiving data (depending on config).

NOTE: Configuration message can go to several MiniBees at the same time, if they share the same configuration. That's why we set the destination address, so that only those MiniBees will receive the message. We'll do some magic on the host's side to determine whether MiniBees have the same config or not.

NOTE: When the host comes up, it should send an announce message, so that all MiniBees can announce their presence (basically do the startup cycle from point 3 onwards). I've added the destination address as an optional parameter, so the MiniBees could switch to that destination address (we'll have to discuss whether that is useful or not; maybe it's not). Similarly, the host could send a Quit message to tell the MiniBees that they can stop doing what they do... this may or may not be useful, since some projects might just want to run MiniBees by themselves...

2.1 Configuration message

msg time interval	2 bytes (interval in ms)
samples per message	1 byte
19 bytes pin configuration	

```
enum MiniBeePinConfig {
  NotUsed,
  DigitalIn, DigitalOut,
  AnalogIn, AnalogOut, AnalogIn10bit,
  SHTClock, SHTData,
  TWIClock, TWIData,
  Ping,
  Custom=100
}
```

Version 2: pins 18 and 19 are not relevant.

Version 3: Two wire interface supports various devices, which are selected after the 19 pin configuration bytes.

msg time interval	2 bytes (interval in ms)
samples per message	1 byte
19 bytes pin configuration	1 byte each
number of I2C devices	1 byte
N x I2C device ID	1 byte each

```
enum TWIDeviceConfig {
  TWI_ADXL345=10,
  TWI_LIS302DL=11,
  TWI_BMP085=20,
  TWI_TMP102=30
}
```

3 Using API mode (version 4)

1. Wake up XBee
2. Read SH and SL *(currently only SL, as SH tends to be 00130020, and the 0x13 is not parsed right)*
3. Send SH and SL to coordinator (msgtype 's') *(currently only SL)*
4. Coordinator sets ATMY of MiniBee with a remote-at command
   Coordinator sends node ID (msgtype 'I') with an optional configuration ID. If so, the board will receive a new configuration.
5. *if new config* Node sends waiting message (msgtype 'w').
6. *if new config* Wait for configuration message.
7. *if new config* Write new config to EEPROM and confirm received configuration message.
8. Load config from EEPROM.
9. Node sends config summary (msgtype 'c').
10. Start sending data, or wait for receiving data (depending on config).

description	type	data	sender
Data output	'O'	node ID + msg ID + N values	server
Data	'd'	node ID + msg ID + N values	node
Active	'a'	node ID + msg ID	node
Pausing	'p'	node ID + msg ID	node
Loopback	'L'	node ID + msg ID + onoff	server
Running	'R'	node ID + msg ID + onoff	server
Custom message	'E'	node ID + msg ID + (*data*)	server
Announce	'A'		server
Quit	'Q'		server
Serial number	's'	node ID + msg ID + (Serial High (SH)) + Serial Low (SL) + + library version + board revision + capabilities	node
ID assignment	'I'	node ID + msgID + (SH) + SL + (*config ID*)	server
Wait for config	'w'	node ID + msg ID + config ID	node
Configuration	'C'	node ID + msg ID + config ID + configuration bytes	server
Confirm config	'c'	node ID + msg ID + config ID + smpMsg + msgInt + + datasize + outsize + (*custom*)	node
(*custom*)		customInputs + customDataSize + N x (custom pin, data size)	

Table 2: Message protocol between host and MiniBee nodes in API mode. Character 13 needs to be escaped. General convention: server message in capital, node message small.