Rick DeV
Paul K
Andy P
Mike S
Ed W

Draft - please review and comment.

BASE COMMUNICATOR PROTOCOLS - Version 1.0
Mike Saari 1/3/84
File: protocol.tp2

This document will deal strictly with base communicator (BACOM or B.C.) protocols. Commands passing between the Apple and Topo are covered in a separate document.

IR PROTOCOLS

Packet Format

All messages are sent over the IR link in packets. The packet format is one bit every 256 usec, starting with 5 on's and 1 off (the "start sequence"), then characters coded as 8 data bits (LSB first) followed by 1 parity bit - odd parity. This gives a start sequence time of 1.5 msec, a character time of 2.3 msec and a packet time of 20.0 msec. A high bit is a 5 usec IR pulse, a low bit is no IR pulse sent. All packets in the system are always 8 characters, expect for short ACK's which are only a single character. The 8 characters in the packet are ordered as follows:

ch#, proc#, cmd#, d,d,d,d, cksum .

d,d,d,d are 4 characters of data; ch# is the channel number to indicate which device should receive the message; proc# is the on-board process number; cmd# is the command for the given process; and cksum is the twos complement of the sum of all the previous characters.

All packetizing, retries, and IR error handling is handled by the H.D.

IR Timing Specs

For timing purposes, "packet start" occurs on the first bit of the start sequence. "Packet end" occurs at the parity bit for the last byte transmitted.

-Packet end to packet start "dead time" = 2 msec minimum

-Packet end to response packet start = 50 msec maximum

-Packet end to saywhat? start = 55 msec minimum

As implemented, packet end to saywhat? start is actually 60 msec. Also, if the B.C. receives a valid start sequence and a valid ch# (which also indicates the length of the forthcoming message) before the 60 msec timeout, then the B.C. will wait until the expected end of the message before sending a saywhat? - even if the total wait time exceeds the 60 msec timeout time.

ACK0,ACK1, and Saywhat Protocol

Every message received by any given Topo (except for public channels), is acknowledged by an ACK0 or an ACK1. These must alternate to insure valid transmissions. The MSB of the channel# being transmitted by the B.C. indicates whether Topo should send an ACK0 or ACK1 - 0 means ACK0, 1 means ACK1. Similarly, the return ACK is indicated by the MSB of the returning channel#. The B.C. is required to keep track of the current ACK0/ACK1 status for each private channel.

A valid ACK should be received by the B.C. within 50 ms of the end of the message being sent. If no ACK is received within that time, or a garbled transmission or the wrong ACK is detected, then a "saywhat?" command (see CONFIGURE BACOM) is given, telling the Topo to repeat its last ACK. Receiving the correct ACK means to continue normally, i.e. the original message has been correctly handled. Receiving the wrong ACK means the original message was mishandled and should be resent. Receiving nothing or a garbled reply should prompt another "saywhat?". After 5 or more saywhat's without a valid reply, the appropriate QUERY error flag (Topo not responding) is set while the saywhat's continue. The error flag will only be reset after the message is eventually received, or by doing a RESTART BACOM.

If Topo is ever unable to determine the answer to a status request within the ACK timeout time, it simply sends the appropriate ACK with no message. The Apple may then deal with the error of "message not received". This timeout handler could change with future product revisions.

When Topo sends a message to the B.C. (generally a status requested by the Apple), the format is:

long ACK B.C. channel #,
0 (don't care)
0 (don't care)
d,d,d,d,
cksum.

For a short ACK, the message is a single character, simply:
short ACK B.C. channel #

IR Carrier

A carrier signal tells all Topos if they go out of range of the B.C. The B.C. is required to send out some command at least once every 250 msecs. If no command has been sent after 250 msecs has elapsed, the B.C. sends out a dummy carrier message (see CONFIGURE BACOM). Any Topo which hears no valid commands for 1.28 seconds will PARK (abort the current motion command and flush the motion command queue).

APPLE/B.C. PROTOCOLS

8-bit Data and Nibbling

All data in the system, whether on the Apple or onboard Topo, consists of full 8-bit words. There is a problem, however, in that most serial cards deal with 7-bit ASCII, and also trap certain control codes. To avoid this, all data sent between the Apple and B.C. will be broken up into 4-bit nibbles, sent through the serial card, and then reassembled. (The full 8-bits are sent over the IR.) The 4-bit nibbles are encoded as ASCII 0-9 and A-F. For example, data 10100011 would be sent as A3 (hex 41, hex 33). A letter G (hex 47) would be sent as 47 (hex 34, hex 37).

Apple/B.C. Command Syntax

Handshaking commands between the Apple and B.C. are not broken up into nibbles. All commands and responses are standard ASCII characters, excluding 0-F and excluding all control codes. The actual range of values is hex 50-7F.

Apple/B.C. Serial Bus Communication

Although the baud rates are setable on both the serial card and the B.C., the standard configuration is: 9600 baud, no parity, 1 start bit, 8 data bits, and 1 stop bit.

Apple/B.C. Handshaking

A partial handshaking system is used between the Apple and the B.C. Before sending any message or command to the B.C., the Apple should first send a

QUERY (ASCII Q).

The B.C. must respond with one of several answers to indicate readiness to receive the message. The required response time is within 3 ms + 1 character time (dependent on the baud rate). (This protocol will run at 9600 baud, which gives a character time of 1 ms.) The possible responses are:

```
   Bit 3 - BUSY  - Apple may not send more data yet
       2 - MESSAGE WAITING (from Topo to Apple)
       1 - ERROR - Topo not responding
       0 - ERROR - invalid message from Apple to B.C.
 all off - READY - OK to send
```

The response is a single character, where the indicated bit is active. The four high-order bits are always 1110, so responses are E0-EF (ASCII lowercase blank,a,b...m,n,o). Thus MESSAGE WAITING is 11100100, or hex E4, and BUSY is 11101000 or hex E8 .

The B.C. may interrupt a long message coming in by sending an

INTERRUPT (ASCII U) back to the Apple, which must then stop sending and QUERY until a READY is received. (The Apple always checks before sending any character to see if an INTERRUPT has been received.)

(The B.C. will accept data even if MESSAGE WAITING is set. However, the Apple will generally read the message first.)

Apple/B.C. Command Formats

The standard message format from Apple to B.C. is:

MESSAGE START (ASCII S)
Topo process number (nibbled)
Topo command number (nibbled)
d,d,d ... d (nibbled)
MESSAGE END (ASCII Z).

This is used for every message which is to be sent from the Apple to Topo. If the B.C. detects an extra MESSAGE START, an extra MESSAGE END, a MESSAGE END before proc# and cmd# have been received, a framing error or other serial error, or a MESSAGE END after only half of a character (one nibble) has been sent, then the invalid message error code is set for the next query. Remaining characters at the end of a message, which fall short of filling a packet, are sent out in a final packet padded with trailing spaces (hex 20). If the data for a message spans more than one packet, then the Topo process number and command number will be duplicated for each packet.

CONFIGURE PACKET (ASCII P) tells the B.C. the channel # for further packets, and a public/private flag to indicate whether to expect a return ACK. When the public flag is set, no ACK's are expected. The format is:

CONFIGURE PACKET (ASCII P)

20 channel # (nibbled)
00 public flag - 1=public, 0=private (nibbled)

The left column shows the power-on default values (in hex).

CONFIGURE BACOM (ASCII Y) tells the B.C. its own IR channel #'s, the content of the carrier dummy message, and the content of the "saywhat" message. The format is:

CONFIGURE BACOM (ASCII Y)
10 long ACK B.C. channel # (nibbled)
0F short ACK B.C. channel # (nibbled)
1F carrier channel # (nibbled)
50 carrier proc# (nibbled)
41 carrier cmd# (nibbled)
82 "saywhat" proc# (nibbled)
FF "saywhat" cmd# (nibbled)

The left column shows the power-on default values (in hex).

GET LAST RESPONSE (ASCII R) will cause the B.C. to return (0,0,d,d,d,d) (in nibbles) that was last received back from Topo. The 0's are don't care characters returned with the packet. They

may be used in the future as a response identifier, or for other information.

RESTART BACOM (ASCII X) cancels the current packet being attempted and resets all four QUERY status flags. Channel settings and other parameters are not affected.

GET BACOM REVISION (ASCII V) causes the B.C. to return two don't care bytes, then four bytes (in nibbles) of the B.C.'s version number and prom number, i.e. (0,0,V.V,P.P). Each byte has values 0-99. The first byte is integral version, the second is fractional version, i.e. 1.00, 2.05, 99.99 . The third and fourth bytes are integral and fractional prom numbers. The initial release will be 1.00.

In the middle of a message the only legal B.C. commands are QUERY and RESTART BACOM, otherwise just data and END-OF-MESSAGE (after READY has been received.) In fact, whenever an INTERRUPT or BUSY has been received, the only legal commands are QUERY and RESTART BACOM.

Handshake Command Summary

P - configure Packet. Takes (ch#, public flag) in nibbles.
Q - Query bacom. Returns E0-EF (four status bits).
R - get last Response. Returns (0,0,d,d,d,d) in nibbles.
S - Start message.
U - interrUpt character to host.
V - get bacom reVision. Returns (0,0,V.V,P.P) in nibbles.
X - restart bacom.
Y - configure bacom. Takes (HD ch#, short ACK#, carrier ch#, carrier proc#, carrier cmd#, saywhat proc#, saywhat cmd#) in nibbles.
Z - message end.

Status Bit Values

Bit 3 - BUSY - Apple may not send more data yet
2 - MESSAGE WAITING (from Topo to Apple)
1 - ERROR - Topo not responding
0 - ERROR - invalid message from Apple to B.C.
all off - READY - OK to send

Real-time Joystick Considerations

Since the packets are relatively short (20 ms per packet, plus a 6 character message at 9600 baud gives 15 ms), this should work for real-time joystick response. Since short ACK's take only another 4 ms (plus computation time), it may well work with ACK's, too. The typical configuration will be to use a public channel so as to ignore any message errors, since we are better off simply sending the newest command, instead.