GoGo Board Tethered Mode Protocol

Arnan (Roger) Sipitakiat
Updated: July 8, 2011
http://www.gogoboard.org

General Information

This document describes only the serial protocol of the tethered mode. The protocol for Cricket Logo (autonomous mode) can be found on the handy-cricket website (http://www.handyboard.com/cricket)

The serial port setting for the GoGo board:

- Baud rate 9600 bps.
- No parity, 8 data bits, 1 stop bit (N,8,1).
- No flow control; neither in software (XON/XOFF) nor hardware (CTS/RTS, DTR/DSR).

Here is a typical sequence of bytes flowing in a GoGo board command session

PC to GoGo	0x54	0xFE	Command Byte(s)	
Reply from GoGo to PC	Echo bytes	0x55	0xFF	Reply Byte(s)

- There are two header bytes with values 0x54, 0xFE preceding every command.
- Following the header bytes are command bytes. Most commands require only one byte, but some require a second parameter byte. An extended command can contain multiple bytes. See command chart for details.
- After receiving each command, the GoGo board will respond by sending echo bytes. That is, the gogo will send back whatever it has received. Then, it will send an acknowledgment header. This header consists of two bytes with values 0x55 and 0xFF respectively. Other bytes may follow as described in the Command chart.

GoGo Board Command Chart

Note that every command byte is always preceded by two header bytes (0x54, 0xFE) and the reply byte(s) are also preceded by two header bytes (0x55, 0xFF).

Command	CMD Len	CMD byte			2nd byte	Reply
		CMD	Parm	Ext		
Ping	1	000	Board ID		–	Ack,BID,BV,FV
Read sensor	1	001	SSS	RM	–	SSh,SSl
Motor control	1					
ON		010	000	XX	–	Ack
OFF		010	001	XX	–	Ack
Reverse direction		010	010	XX	–	Ack
This way		010	011	XX	–	Ack
That way		010	100	XX	–	Ack
Coast		010	101	XX	–	Ack
Set motor control	1	011	PPP	XX	–	Ack
Talk to motor	2	100	XXX	XX	Motor bits	Ack
Set burst mode	2	101	XXX	BM	Burst bits	Ack + burst cycles*
Misc controls	2					
User LED on		110	000	00	XX	Ack
User LED off		110	000	01	XX	Ack
Beep		110	001	XX	XX	Ack
Set PWN duty		110	010	XX	Duty cycle	Ack
Upload EEPROM		110	011	EL1	EL2	Ack + Data**
I2C start		110	100	00	XX	–
I2C stop		110	100	01	XX	–
I2C write		110	100	10	I2C data	–
I2C read		110	100	11	Ack indicator	–
Autorun OFF		110	101	00	00	Ack
Autorun ON		110	101	00	01	Ack
Extended CMD byte	2	111	111	11	-	Depends on command

* See Burst Mode section for more info
** See EEPROM upload section for more info

Extended Commands

An Extended command begins with a command length byte, which determines the length in bytes of the extended command. The second byte is the command byte. Any remaining bytes are parameter bytes. Therefore, the entire command stream (including the first command byte) is as follows:

0xFF	Length	Ext CMD	Parameter bytes..

Oxff = First command byte. Notifies the board that this is an extended command
Length = Length of the extended command in bytes.
Ext CMD = The extended command byte. See the table below for a description.
Parameter bytes = The remaining Length-1 bytes used as parameters.

Command Byte definition

Command	Len	CMD byte		Extended byte(s)	Reply
		CMD	Parm		
Read from RTC	1	0000	0000	–	RTC packet***
Write to RTC	8	0000	0001	7-byte RTC packet***	Ack

*** A Real-time-clock (RTC) packet consists of a 7 byte stream as follows:
Byte1 = Seconds, Byte2 = Minutes, Byte3 = Hours, Byte4 = Day of week (1-7)
Byte5 = Date, Byte6 = Month, Byte7 = Year (where 00 = year 2000)

Legend

Symbol	Values	Note
X		Don't care; this should always be 0.
Ack	0xAA	Acknowledge byte.
Board ID	0-31	Applicable only when using multiple GoGo boards (via RF, or I^2C). 0 = local board (default).
BID		Board ID. 01 = GoGo board
BV		Board version 0x40 = version 4.0
FV		Firmware version (runs from 0 to 255)
SSh, SSl		Sensor value bytes (high and low, respectively).
SSS	0-7	Sensor number.
RM* (read mode)	0	Default. Gets current sensor value.
	1	Read-max. Gets maximum sensor value sensed since the last max-value read operation. The board resets this max-value after each read.
	2	Read-min. This is the opposite of read-max.
PPP	0-7	Motor power level.
Motor bits		Each bit in this byte represents a motor port (bit 0 is port A, bit 1 is B, and so on). If a bit is set, its corresponding motor port will be active.
BM	0, 1	0 = normal mode (~30 Hz per channel), 1 = slow mode (~10 Hz).
Burst bits		Each bit in this byte represents a sensor port (bit 0 is sensor 1, bit 1 is sensor 2, and so on.) If a bit is set, its corresponding sensor will be included in the burst mode stream.
Burst cycle		0x0C + high byte + low byte. Bits 5-7 of the high byte contain the sensor number.
Duty cycle	0-255	Sets the PWM (pulse width modulation) duty cycle for the active motors.
EL1 & EL2	0-1023	Number of blocks to upload from the EEPROM. One block is 32 bytes (16 sensor values). EL1 is the upper 2 bits and EL2 is the remaining 8 bits.
I^2C data		One data byte. For I^2C write, this byte may contain a 7-bit slave address, in which case the final bit is used to specify whether the slave is going to receive (0) or transmit (1).
Ack indicator	0, 1	Specifies whether the device that issued the i2c_read command will ack after receiving data. i2c_read(0) = no ack, i2c_read(1) = ack.

* The read mode feature has been removed starting from GoGo 2.3 firmware MS2. The board will no longer keep the max and min values.

Burst Mode.

Burst Mode is designed for applications that need maximum sensor refresh rate. With out the burst mode, you need to make a request to the GoGo board (with the read sensor command) every time you want to refresh your sensor data. With the burst mode, the GoGo board will actively stream new sensor data to you.

To activate the burst mode, you need to send a set-burst -mode command together with a burst-cycle byte (see command chart). The burst-cycle byte tells the GoGo board which sensor ports to include in the burst cycle. A burst-cycle byte of value 0 deactivates the burst mode.

A burst cycle consists of X sensor chunks where X is the number of activated sensors. Each sensor chunk is three bytes long and contains a sensor value. Here's the format of a sensor chunk:

Byte 1	Byte 2	Byte 3
0x0C	Sensor Value (high byte)*	Sensor Value (low byte)

0x0C is the header of a sensor chunk
Sensor Value bytes contain the 10 bits sensor value
*Bit 5-7 of the high byte sensor value contains the sensor number.

Example 1:

A burst-cycle byte of 0000 0010 will tell the GoGo board to stream sensor two. Here's an example of a burst cycle:

Byte 1	Byte 2	Byte 3
0000-1100	0010-0001	0100-0100

Bits 5,6,7 of the sensor value high byte (underlined) is 001 indicating that this is sensor 2. The actual sensor value is 01 0100 0100 = 324

Example 2:

A burst-cycle byte of 1001 0010 will tell the GoGo board to stream sensor 2, 5, and 8. Thus, one burst cycle will consist of 9 (3x3) bytes. Here's an example of a burst cycle:

0000-1100, 0010-0001, 0100-0100,
0000-1100, 1000-0000, 0000-0010,
0000-1100, 1110-0010, 0000 1001

Bits 5,6,7 of the second, fifth, and eighth byte are 001, 100, and 111 indicating that they are sensor values of port 2, 5, and 8 respectively.
Their sensor values are 01 0100 0100, 00 0000 0010, and 10 0000 1001 accordingly.

EEPROM Upload Protocol

Follow these steps to trigger an auto-upload.

1. Send 0x54, 0xFE, 0xCC, 0x00 to the gogoboard. This puts the gogo in an auto-upload mode.
2. The Gogo board will send echo the bytes then send ack bytes (0x55,0xFF, 0xAA).
3. The Gogo will send four bytes. 0xEE, 0x11, uploadLen (low byte), uploadLen (high byte). The first two bytes are just headers. The latter two are the upload length in bytes.
4. The GoGo will then start sending the actual data. Each value is a 16-bit number. The order is low-byte then high-byte.

Here's an example session:

```
Computer->GoGo  :  0x54, 0xFE, 0xCC, 0x00   // command
GoGo->Computer  :  0x54, 0xFE, 0xCC, 0x00   // echo
GoGo->Computer  :  0x55, 0xFF, 0xAA   // ack
GoGo->Computer  :  0xEE, 0x11, 0x06, 0x00   // upload header
GoGo->Computer  :  0x00, 0x00, 0x01, 0x00, 0x02, 0x00  // the data
```

In this example the gogoboard uploaded three 16-bit values: 0, 1, 2 respectively.

Note. You may find that the gogo seems to be excessively sending stuff back to the computer with all the echos and acks. This was done to provide compatibility with the Cricket download protocol. In practice, once you send the command bytes to the gogo you can just wait for the upload headers (0xEE, 0x11) while ignoring the others.

Example commands:

Command	Bytes to send	Reply bytes (After Echo Bytes)
Ping	**0x54 0xFE 0x00**	**0x55 0xFF 0xAA 0x01 0x40 0x0A** **Found board type 1 (gogo),** **Board version 4.0,** **Firmware version 10**
Turn on motor A	**0x54 0xFE 0x80 0x01** **(Activate motor A)**	**0x55 0xFF 0xAA**
	0x54 0xFE 0x40 **(Turn motor on)**	**0x55 0xFF 0xAA**
Turn on motor C	**0x54 0xFE 0x80 0x04** **(Activate motor C)**	**0x55 0xFF 0xAA**
	0x54 0xFE 0x40 **(Turn motor on)**	**0x55 0xFF 0xAA**
Turn on motor A and C	**0x54 0xFE 0x80 0x05** **(Activate motor A,C)**	**0x55 0xFF 0xAA**
	0x54 0xFE 0x40 **(Turn motor on)**	**0x55 0xFF 0xAA**
Read sensor1	**0x54 0xFE 0x20**	**0x55 0xFF 0x01 0x37** **(Sensor value is 0x137)**
Turn on burst mode for sensor 1 and 8	**0x54 0xFE 0xA0 0x81**	**0x55 0xFF 0xAA + Burst Cycles**