Enhanced Firmware

Quick Start Guide

Dec 10, 2004

Table of Contents

Document Change History

Date/Version	Contents
18 Nov 2004	Initial Release describing Beta firmware version
10 Dec 2004	Updated to reflect first general firmware release. Some material still remains outdated. Removed content describing new opcodes and firmware features and placed in separate files that are also web accessible.

Enhanced Firmware Compatibility and Operation

Overview

The objective of this document was to provide a "How To" guide and overview of the replacement firmware.

The design objective was to provide backwards compatibility with standard Lego firmware. This has been achieved. It has been tested with both the BricxCC/NQC and Robolab development environments (IDEs). There is active engagement with the developers of both of these IDEs to provide good support for the new firmware.

The simplest way to get started is simply to load the firmware into the RCX and seamlessly start using it with your existing programming environment and applications. Naturally, capabilities are limited to the functionality supported in the IDE (Bricx, Robolab, etc.) and not all of the features supported in the new firmware (e.g. expanded number of variables) will be available.

The firmware distributed is the 10 times faster version. It does not include support for floats and longs. There are about 8K bytes of space for user programs and datalog vs. 6K in the standard firmware. When floats and longs are included in the firmware available space drops to about 4K.

The remainder of this document contains section on:

- Operation and compatibility of the new firmware.
    - How to download the new firmware and possible problems.
    - Minor known incompatibilities.
    - Functional areas that may be untested.
- Compatibility with current BricxCC/NQC development environment
- Compatibility with Robolab

Downloading Problems

Filename of new firmware file is *fastfirm0101.lgo* (or something similar). Some programming environments allow you to specify the name of the firmware file and you simply point to this file when downloading. In other environments (Lego's software, Robolab) the filename is not user specifiable and you will have to find the existing file (*firm0328.lgo* for Lego's software, *firmware.txt* for Robolab), archive it so you can restore it, and then copy the *fastfirm0101.lgo* to this location and rename it to the name expected by the environment.

Some programming environments use a fast firmware downloader that operates at four times normal IR transmission speed. The *fastfirm0101.lgo* file is more susceptible to intermittent download failures and you may need to disable fast downloader if this occurs. Technically this is due to a couple of large blocks of contiguous zero values in the download file; long messages containing mostly zero can cause infrared transmission failures when operating at quad speed transmission because quad speed transmission works best when there is a balance of zeroes and ones in the bits.

A problem that occurs in Bricx is that after quad speed downloading is finished, there is a failure message stating "Firmware download failed. Put the RCX closer and try again". This message also occurs when downloading the standard firmware. I suspect that this is because Bricx has configured serial link for 4800 baud for fast downloading and the RCX resets itself back to 2400 baud as part of its firmware initialization sequence, but BricxCC is still talking at the 4800-baud rate. If you hear a beep from the RCX, and the display is a fast clock (in M.SS.S – minutes, seconds and tenths of a second) format, then downloading was successful.

Known Incompatibilities with Standard Firmware Features

There are some minor incompatibilities with existing firmware. This includes:

- "Calibrate" opcode for 'events' is currently a no-op. This opcode is designed to automatically determine the high/low ranges for events based on a short series of measurements. However, initial event thresholds follow the same default values as defined in the standard firmware and the opcodes to 'manually' configure the thresholds are supported.
- Global motor operation opcodes are not supported. A no-op is implemented.
- The default display format is M.SS.S format. It can be reset to Lego standard HH.MM format within your program.
- An exception handler has been implemented that traps about 30 different kinds of exceptions in the byte code interpreter. Standard Lego firmware simply ignores "exceptions". Exceptions include bad parameters to an opcode, invalid opcode, address out of range of current program/task, stack underflow/overflow when calling nested subroutines, etc. Hopefully, you won't notice these. When an exception occurs you'll hear a funny set of tones (upward tones and a buzz) and the display will change to a number indicating the type of exception.
- I think the buttons on my RCX may be "wearing out" because I have trouble getting them to work; I have to press hard and perhaps hold them down too long. But it could also be because I'm not scanning them fast enough. Let me know if you have similar problems. There were no reports of this problem from any of the early testers so it's probably a fault with my RCX.
- I've found a few programs that stopped working reliably because they were dependent on the slower speed of the standard firmware and didn't worry about inserting delays. For example, a program that contained a loop generating IR messages where the time to execute one pass of the loop was longer than the time to transmit a message; with faster firmware, program tries to send new message before the previous one is completed.
- Sensor handling in the Beta version of the new firmware was too responsive for many programs. Programs would set up an active sensor (e.g. a light sensor) and then immediately try to read the sensor value. Active sensors are powered from the RCX with the power temporarily removed (for about 100 microseconds or less) while the value of the sensor is read. It takes quite a settling time for a freshly "activated" sensor to reach its steady state value – I did detailed measurements and found standard light sensors took 60 milliseconds to reliably reach their steady state levels although they get to be very close after about 5 – 10 milliseconds. This is a "lifetime" of program execution with the new firmware. The solution I implemented was to internally suspend a task for 10 milliseconds (actually a user configurable parameter) when an active sensor is initially configured. This (hopefully) avoids the need for end user program changes. A byproduct of this change is its best to do sensor setup once at the start of your program rather than imbed in a loop so that delay is only experienced once.
- Some reports that tones/sounds aren't quite the same as with the old firmware. This has not been investigated as of yet. The objective was that they were the same.

"Untested" Areas

The bulk of current testing has been using the programs developed for my robotics applications. This covers arithmetic opcodes, branching opcodes and motor and sensor opcodes, event handling and multiple tasks.

Datalog support has been implemented but only briefly tested. It appears to work fine with the BricxCC utility.

Multiple tasks and event variables appear to perform at least as well as existing standard firmware implementation. Recent stress testing has exposed several weaknesses in the implementation that should be addressed in an upcoming release. Stress testing involves generating 1,000 events per second and ensuring that they are all getting properly processing (i.e. recognized by the firmware and reported to appropriate "monitoring" tasks. This appears to be working fine. However, programs that have a "do forever" loop to "monitor" events, handle them in a "catch" handler and then repeat the loop may miss some events that occur between when the catch handler first starts executing and the program gets back to starting the next "monitor" cycle. It's not a bug in the implementation; simply that events are happening too fast.

Have implemented, but not tested, the ability to send a single IR messages with custom configuration (no preamble, 4800 baud, no complement bytes). However, I have thoroughly tested this customization with some new capabilities that allow the customization to be set on a permanent basis (i.e. until the next power down or user reconfiguration); this is useful with PC programs that want to always communicate with the RCX at a faster speed.

BricxCC / NQC Compatibility

This section is likely to be out of date. John Hansen has made terrific progress on upgrading NQC to support the new firmware. He's added support for 256 variables and for many of the new "intrinsic" variables.

There are currently no known issues with NQC compatibility and the new firmware.

Robolab Compatibility

New firmware has been through preliminary compatibility testing by the Robolab development team. A few minor incompatibilities were discovered and fixed.

One of the fixes for Robolab compatibility was a "kludge" that affects the "upload memory opcode. In the old firmware address 0cC888 contained a variable indicating whether the "running man" LCD display was "running" or "stopped". Robolab interrogates this memory location to see if a program is running. Of course, in the new firmware this won't work. The kludge was to change the "upload memory opcode" so that when this address is uploaded it contains the running man Boolean rather than the real memory location.

Other than the items mentioned above, there are currently no known issues with Robolab compatibility and the new firmware.