Performance Testing Results

The following table summarizes the results of performance testing of the replacement firmware compared to the Lego standard firmware. Several different NQC programs were downloaded from the web and their performance measured. Results were normalized so that Lego firmware had a value of 1X. Overall, the performance is comparable to 10X and 100X.

Program	Lego	10X	100X	Tweaked
Light Rover	1X	7.9X	58X	86X
Peeves PID	1X	8.0X	50X	87X 185X
Lepomux	1X	9.4X	93X	41X
Remote Car	1X	7.6X	79X	139X
Empty Loop			80X	170X
Average	1X	8.2X	88X	118X

The short video (600K, 'wmf' format) illustrates the speed difference of the “empty loop” program as it updates the LCD display. You may have to save the file on your computer and then open it from there; doesn't always seem to open and play when directly accessed from the web.

The first four were NQC programs downloaded from the web. Each contained 100 to 1,000 lines of user code. The four columns of data represent real time performance for four different firmware scenarios.

- The first column is the performance of the standard Lego firmware.
- The next two columns contain the performance of the 10X and 100X firmware versions when downloaded by existed BricxCC IDE and compiled by existing NQC. No source code changes.
- The tweaked column contains results after only a few lines of changes in each program to better utilize replacement firmware capabilities. The changes include:
  - Programs were simply recompiled to use new firmware opcodes. These new opcodes pass compile time information to the interpreter to take advantage of rather than it re-calculating at runtime. For example:
  - Separate opcode for A += 5 (constant parameter) than for A+= B (variable parameter). 30% performance boost on arithmetic opcodes for this.
  - Avoid the value check on every assignment opcode to see whether it is one of the three special counter variables which, when changed, may trigger an event.

  Recompilation was with a new compiler.

- The Lepomux program sends messages to an I/O expander using changes in a motor output. Delay statements had to be inserted because it was running too fast for the hardware.
- One line was inserted into the “Remote Car” program to use the new “wait for message” opcode. In the new firmware, this takes a task out of the “ready to run” list until a message arrives preventing it from being given time slices.
- Variables in the Peeves program were redefined as 32-bit integers to eliminate a time consuming subroutine. Boosts performance from 87X to 185X.

The variations in performance boost are due to different opcode mixes among the programs. The biggest performance gain is in the arithmetic opcodes; intrinsic opcodes, like motor control, only have about half the speedup.

Program Descriptions:

A brief description of the four NQC programs follows.

Light Rover: Robot uses a light sensor to detect and avoid collisions.

Peeves PID: “Peeves” is a dead reckoning (using rotation counter) robot. It needs 16-bit precision on the result of “A*B/C” and uses a subroutine to perform this specialized calculation. Measurements on this subroutine were performed.

Lepomux Signaling: Lepomux is a motor/sensor expander that sends serial messages out a motor port. The NQC program updates the motor direction and power so that message bits are “clocked” out on the motor PWM waveforms.

Remote Car: This was a large NQC program from a senior’s thesis at CMU. The program tries to have two Mindstorms based “cars” cooperatively driving the same route. Multi-byte messages were built up from standard 8 bit messages. Program has two tasks; one to receive messages and one to interpret and drive cars. Timing was performed on the second task.

Empty Loop: A simple program to count the most number of “idle loops” that can be executed in 10 seconds. Code for the “empty loop” program is below. Standard firmware reaches 12 at end of 10 seconds; tweaked firmware counts to 1809!

```
// Simple program to demonstrate the performance comparison between
// standard and fast firmware.
//
// Count the number of times a 100 item "do nothing" loop can be
// executed in ten seconds.

int nCnt;
int nLoops;

task main()
{
  nLoops = 0;
  SetUserDisplay(nLoops, 0);  // display results on LCD

  ClearTimer(0);
  while (FastTimer(0) < 1000) // loop for 10 seconds
  {
    for (nCnt = 0; nCnt < 100; ++nCnt)
     {} // waste some time. Otherwise fast f/w will overflow 16-bits
    ++nLoops;
  while (true)
  {} // Delay forever to keep 'nLoops' on LCD display
}
```