Response Time Testing of Various Beaglebone Black Programming Methods

David Mehl

Overview: Response time testing was performed with various methods of copying one GPIO on the beaglebone black to another GPIO. These methods included a program written in Node.js, a program written in C using mmap, a linux kernel module, and a program written for the PRU. In all cases a 1 kHz square wave was inputted into a pin, and the programs were to copy the input to a separate output. Table 1 displays the general results.

1 kHz Square Wave Testing				
Method	CPU Usage	Max Delay	Min Delay	Avg Delay
Node.js	89.00%	178 ms	120 us	324 us
MMAP	95.80%	5.8 ms	22 us	24 us
Kernel	6.50%	606 us	32 us	132 us
PRU	4.70%	27.6 ns	23 ns	24.8 ns

Table 1: Test results for different methods

Node.js: The results of the JavaScript program were less than ideal. Not only was the copied waveform slow and of a lower frequency than the input, it was inconsistent and the program itself significantly hampered the CPU. The program was interrupted by other processes, which led to the massive maximum delay time, and even on a normal basis the output was varying consistently.

MMAP: The results of the C program were a marked improvement over the Node.js program. While the C program did use more CPU, it's output was far more consistent than the Node.js program, with the minimum delay essentially equaling the average delay. However, there was still evidence of the program being interrupted by other threads, with a worst case delay being much lower than the worst case for the Node.js program. That being said, this program was much improved over the Node.js program when it comes to consistency when not being interrupted.

Kernel: The kernel module was actually slower than the C program, which was surprising. However, considering it is still software based, this is a reasonable outcome as the code is often interrupted by other kernel modules with a higher priority. The consistency of the waveform left much to be desired. There was a massive standard deviation, and the overall functionality is most close to the Node.js solution. However, the CPU usage was minimal. Where the first two CPU percentages were the singular processes that were running the programs, this CPU percentage is overall for the entire system, thus showing a drastic improvement.

PRU: This method was the best of all three worlds. It was consistent, it was fast, and it didn't hamper the CPU whatsoever. The delay was multiple orders of magnitude less than the other methods, at a level where the delay was impossible to see without increasing the zoom on the scope to almost the maximum. There were no interruptions to the output, and the program didn't affect the CPU at all, with the lowest percentage of overall CPU usage in the entire experiment.

Conclusion: The PRU is the ideal method for performing this task, as it outperforms the other methods absolutely in every category. The only other feasible method would be the kernel module. The C and JavaScript methods simply hamper the CPU too much for the device to remain effective and thus should never be used for this purpose, unless this was the only task intended to be executed.