Improving Speed and Performance*

CJ Ganier

This work is produced by The Connexions Project and licensed under the
Creative Commons Attribution License †

Abstract

In this lab we will focus on improving the performance of the MSP and attempt to optimize your code.

So far in this course, programming assignments have focused on functionality. In most applications of embedded programming, speed and power performance are equally important. Long battery life is won through judicious hardware and software design. Skillful programming will allow the same job to be done with cheaper parts to improve the bottom line. This lab will introduce the basic concepts behind power and speed performance improvement.

1 Speed Performance

It is well known from the consumer PC market that the speed of computers can be measured in hertz. It is less well known that the frequency of the computer's processor does not adequately indicate a computer's performance or even the performance of the processor itself. By using the ELEC226 board, the choice of processor and speed has been made, but the question of speed is a different one in embedded programming. While the dominant paradigm in consumer personal computing is to increase the performance of the computer to allow the system to do more with each generation, embedded processors are chosen to be able to perform specific tasks. The cheapest processor that can meet the specifications for the design will be chosen. While the issue and business conditions make the situation much more complicated than just price, the pressure is still toward choosing a part with **less** performance, not more.

In order to improve the performance of a software application, it is necessary to understand the way performance is measured. Measuring performance between platforms and software packages is a problematic endeavor, improving the performance of a single program on a single platform is much simpler. a detailed explanation of the nuances of performance measurement in computing is beyond the scope of this lab, simple way to gauge the amount of time a program will take to perform a task is to count the number of processor cycles that the code will take. On the MSP430, each CPU instruction, jump, and interrupt takes a fixed number of cycles as explained in the MSP430 User's Guide. Taking into account branching, function calls, and interrupts the assembly code of a program can be used to calculate the time needed for a section of code.

2 Performance Tips

As mentioned above, embedded programming has different priorities from personal computing. Because the embedded programmer is usually trying to accomplish a specific task within a certain amount of time, the

*Version 1.5: Aug 21, 2005 6:41 pm -0500
†http://creativecommons.org/licenses/by/1.0

most important test of performance is whether the program is performing calculations on the inputs as fast as the inputs can enter the system. The goal is to make applications "real time[1]."

When the first draft of a program is unable to keep up with the required sampling, it is necessary to reduce execution time. Often, changing the hardware configuration is not possible; and software speed gains are almost always more cost effective.

There are many approaches to improving speed performance. Incredible amounts of research go into new algorithms for common problems to improve the way that problem is solved. However, simply eliminating unnecessary instructions is a good start to improving performance. Test code left in a final version, any unnecessary instructions in a loop, and can all significantly increase the time in a section of code.

In C, unnecessary code takes the form of too many method calls inside of a loop because each function call costs additional instructions. While this is not an important loss for code that is only executed once per sample, in loops that run often, every little gain counts much more. When trying to reduce execution time, it is best to start with the regions of the code where the processor spends the most time. Parts of the program that are only executed rarely have only a small effect on the speed compared to a loop that might run 100 times per sample. If something can be done once, outside of the loop, do not do it many times inside the loop.

Another straightforward way to maximize performance on the hardware provided is to make judicious use of timers and other instruction saving interrupts. The timer interrupts allow the processor to periodically check on the status of the program without the use of slow while () loops. However, for correct program behavior, it is important to do the minimum possible in the interrupt. This is most important with interrupts that happen frequently because the control flow of the program can be thrown off when interrupts happen faster than the system can handle them. If the same interrupt occurs a second time before the first occurrence of the interrupt has exited there, program behavior is much more difficult to control. It is much easier to simply ensure that the interrupt is short enough to avoid the danger all together.

Avoid recalculating values. If a piece of information is reusable, save it rather than recalculating it to save time. Sometimes memory is so scarce that this may not be possible.

Don't use the printf function unless you absolutely must. It can be quite slow if used repeatedly. Use breakpoints instead.

Don't leave legacy code from previous revisions running. If you believe you may no longer need a part of the program, comment it out and note what you did in the comments.

[1] "Real Time" <http://cnx.org/content/m11994/latest/>