

Re: calculation of execution time of assembly code in a realtime program with large number of tasks

Re: calculation of execution time of assembly code in a realtime program with large number of tasks

Source: <http://coding.derkeiler.com/Archive/General/comp.arch.embedded/2007-01/msg01051.html>

- *From:* "karthikbg" <karthik.balaguru@xxxxxxxxxxxxxxxx>
 - *Date:* 22 Jan 2007 02:42:45 -0800
-

bvanevery@xxxxxxxxx wrote:

Arlet wrote:

For a software solution, make sure you have an accurate hardware timer, and take a timestamp at beginning and end, then subtract the values, and report (through printf, or any other suitable method). Use the same method to time an empty piece of code to measure the timestamping overhead.

If the task is amenable, put the assembly code in a loop and run it millions of times. This makes the calculation take much longer and then you don't need quite so accurate a timer. Of course, you have to be careful that you're not changing the basic nature of the processing, such as reading in lotsa data from main memory that otherwise would have been in a cache. Anyways, "put a loop around it" is a general trick for benchmarking.

Cheers,
Brandon Van Every

Hi,
I found something interesting w.r.t Code Composer Studio (CCS) under the topic of Benchmarking .
Pls find the info below that i got from a link . (Looks interesting !!
)

Code Composer Studio has the capability of benchmarking your code, meaning that you can analyze how long your code takes to execute. In the following example, you will be analyzing how long it takes to execute each call of the dotp() function.

Re: calculation of execution time of assembly code in a realtime program with large number of tasks

Re: calculation of execution time of assembly code in a realtime program with large number of tasks

One way of benchmarking your code is to set two breakpoints. When you run the program, the breakpoints will tell the debugger to stop execution. A clock-cycle counter, built into the debugger, can be used to count the number of cycles between these breakpoints.

DO THIS

=====

Do `DEBUG:RESTART` to set the PC at the beginning of the code.

Choose `PROFILER:START NEW SESSION...` with session name of your choice.

You can profile, count the number of clock cycles, a function by adding it to the profile session you created. To do this, left double click on the `lab1.c` file in the Project View to view the file. Place the cursor on any line inside the `dotp()` function and right click. Choose Profile Function -> in (session-name) Session, where (session-name) identifies the profiling session you created. Execute the code by `DEBUG:RUN`.

After CPU halts, click on the Functions tab of the profiling window to see the result of the profiling of the `dotp()` function.

The numbers displayed are the statistics of the execution of the `dotp()` function gathered by CCS while executing the code.

For interpretations of these numbers, read the CCS help topic `Profiler:Viewing Profile Data` by choosing the `Help:Contents` menu on the CCS menu bar.

You can also add a particular line of the code to the profile session. Learn how to do this by reading the helps. There are lots of profiling capabilities that cannot be covered fully in this manual. How many clock cycles (on average) does it take to execute the `dotp()` function?

Regards,
Karthik Balaguru

.

Re: calculation of execution time of assembly code in a realtime program with large number of task