You Can't Control what you Can't Measure, OR Why it's Close to Impossible to Guarantee Real-time Software Performance on a CPU with on-chip cache

Nat Hillary
Manager of Technical Marketing
Applied Microsystems Corp.
nath@amc.com

Ken Madsen
Manager, Product Marketing
Wind River Systems, Inc.
ken.madsen@windriver.com

June 3, 2002.

1. Abstract

Steady increases in CPU core speeds continue to extend the range of applications for computer-based solutions, resulting in the creation of ever more responsive systems. At these higher core speeds, on-chip cache architectures are used to prevent the CPU from stalling when accessing relatively slow off-chip memory. In normal operation, most fetch-execute cycles occur internally, guaranteeing the execution of the maximum instructions per second. However, this also serves to hide the state of executing code from the user. Given the fact that it is not possible to directly monitor the execution of code within such a CPU when it is running at full speed, is it possible to guarantee and control the performance of Real-Time software on these cache-based CPU architectures?

This paper investigates this issue by first offering a definition of Real-Time software, together with a discussion on what must be done to prove that the system will meet its performance objectives in all circumstances. In addition, the range of software performance monitoring techniques that are currently available will be discussed, together with a summary of the pros and cons of each measurement technique. The conclusion of this paper is that it is close to impossible to make deterministic software performance measurements using traditional techniques on a CPU that heavily utilizes on-chip cache, so it is therefore almost impossible to guarantee the performance of Real-Time software based on these styles of microprocessor architectures unless new measurement techniques are utilized.

Real-Time software is distinguished from any other application by the fact that performance criteria are included in the specifications. This means that for Real-Time software to be verified as being correct, it has to be proven that the software will always meet its performance objectives. With cache based microprocessor architectures, software performance measurements are extremely difficult to do without causing serious perturbations, affecting measurement accuracy. This holds true across all possible measurement techniques. Accurate, hardware only performance measurements are generally not possible on architectures utilizing on-chip cache.

Software performance measurement techniques range from hardware assisted solutions, to software only ones. All techniques, however, rely on being able to get information about the current state of the executing code off-chip.

Pure hardware or hardware assisted solutions such as Logic Analyzers (LAs), In Circuit Emulators (ICEs), or dedicated pure software performance monitoring devices cannot determine what code is executing in the cache-based CPU core by monitoring external microprocessor signals. It is therefore necessary to force the activation of off-chip signals (such as an off-chip write or an assertion of a hardware signal) in order for the monitoring hardware to determine the state of the executing code in the CPU core. Due to the performance limitations on the external busses of cache-based CPUs, these external writes have the side effect of stalling the CPU, affecting the accuracy of any performance measurements based on them.

Software only solutions (such as instruction pointer sampling) do not require off-chip writes during measurement, but they introduce their own limitations. As they necessarily require extra software components to be executing on the CPU in addition to the application under test, they cause their own perturbations that affect software performance measurements. By placing extra demands on the CPU, these software measurements are limited in their accuracy. In addition, the introduction of additional code will affect cache flush and update intervals, significantly impacting the accuracy of any performance measurements.

Guaranteeing the performance of real-time software relies on being able to prove that the software will meet its performance objectives in all circumstances. As this paper suggests, obtaining accurate timing measurements is very difficult for systems utilizing CPUs with on-chip cache based architectures. Does this mean, then, that this type of CPU should not be used for real-time systems? The answer is no. This type of CPU is often ideally suited for maximum performance real-time systems and must often be used by system designers in order to build a system possessing top competitive performance characteristics. However, the full maximum real-time performance of these systems cannot be easily guaranteed with any single measurement approach.

This paper will show how the best approach to measuring real-time responsiveness for a system with a CPU containing on-chip cache is not a single measurement approach, but is in fact an approach based on the intelligent, clever, and often simultaneous, use of multiple measurement techniques, from pure hardware based techniques, to hardware assisted based techniques to pure software based techniques.

2. Real-Time Software

Because Real-Time software has performance criteria included in its specifications, it is essential that software execution performance be monitored at every step of the way while it is being created, from the writing of Interrupt Service Routines (interrupt service routines) to time-critical sections of application code. So what techniques may be used to measure software execution performance, and what are the implications of using them with a cache-based CPU?

Starting from board bring-up, the measurement technologies most commonly employed are:

- Logic Analyzers
- In Circuit Emulators
- Hardware-assisted software performance monitors
- Software-assisted software performance profilers

3. Logic analyzers

Typically used to monitor hardware signals, logic analyzers may also be used to make high-resolution measurements of software performance, normally for point-to-point type timing measurements.

All software performance measurements made with a logic analyzer require external CPU signal lines to be asserted when particular lines of code are reached. This results in very high-resolution timing for specific sections of executed code, and when measuring response time against external stimulus or events.

Measuring hardware/software interactions (such as interrupt latency times) is fairly straightforward; a single command is placed at the entry to the software routine in question that asserts a signal on an external CPU pin (e.g. a spare chip select or programmable I/O pin, which will not stall the CPU). The logic analyzer is then used to measure the interval between the external hardware event and the signal marking entry to the software routine being asserted. The fine-grained timing measurements obtained using this technique may also be used for monitoring critical sections of code. Modern Logic Analyzers (such as the TLA series from Tektronix) extend this technique, allowing specific networking signals (such as Ethernet packets or ATM cell contents) to be used as hardware trigger events.

By using instrumentation (e.g. adding statements at salient points in code), logic analyzers may also be used for monitoring the performance of a Real-Time application.

For some applications, using spare off-chip signals are restrictive, as a prohibitive number would be needed in order to correctly identify each unique point in code. Instead, external writes are required to ensure that enough unique instrumentation values are included for the measurements to be meaningful. However, the resolution of this type of solution is slightly less than the technique described above for point-to-point measurements.

Although Logic Analyzers provide a means of making deterministic A to B type measurements of code, they typically do not gather profiling data over a statistically long period. It is therefore necessary to use analytical techniques to ensure that the correct conditions are created so that particular measurements accurately reflect the worst-case execution time of a particular section of code.

Logic Analyzers are an excellent solution for making controllable, minimally intrusive measurements of critical sections of code, particularly those associated with external hardware events.

In some rare cases, inserting additional code into an application degrades the performance to a point where the system's Real-Time characteristics are not being met. In this case, 'black box' performance testing techniques are required, where measurements are made at points external to the CPU. E.g. the response time between a particular Ethernet packet arriving and the system responding might be measured using a Tektronix TLA Logic Analyzer.

4. In circuit emulators

Typically used in the early debug stages of target board bring-up, in circuit emulators may also be used for software performance measurements.

Traditionally, the Real-Time bus trace capability was the most significant feature of an ICE. For non cache-based CPUs, bus trace can be used to monitor the timing of higher-level application code, including those that need to respond to external hardware events.

Aside from the lack of profiling data, this would be the ideal solution for performance monitoring of true Real-Time software, but it requires an off-chip fetch-execute cycle to occur in order to monitor what's going on.

Modern cache-based CPUs tend not to have full ICE solutions available. Instead, most modern CPUs have emulators that use serial test access points such as JTAG.

Most JTAG emulation solutions do not have Trace measurements. Triggering timing measurements with a JTAG emulator requires the use of hardware or software breakpoints, which are intrusive. In addition, the serial JTAG bus is slow by comparison to processor speed and events are detected asynchronously to their occurrence. Any timing measurements made via this bus are going to be subject to inaccuracies; monitoring the execution speed of a 400 MHz CPU core by sending information through a significantly slower serialized communications bus is not an ideal solution.

Traditionally the ideal solution for making software performance measurements on the fly, contemporary ICE solutions rarely support the features required to make deterministic timing measurements of code.

5. Hardware-Assisted Software Performance Monitors

An extension of in circuit emulation technology, hardware assisted software performance monitors, such as the CodeTEST product from Applied Microsystems, are designed specifically to measure software execution performance.

This technology requires the combination of software instrumentation and hardware data collection, with time stamping. It may be used to monitor low-level code (such as interrupt service routines), application level code and also RTOS activity. In addition, time stamping may be triggered by external events, making the timing of hardware/software interactions (such as interrupt latencies) possible.

Although the instrumentation of code is automatic, this technique requires an off-chip write for each measurement point, producing the same inaccuracies as with the Logic Analyzers above. However, the instrumentation required for monitoring the worst-case execution time of critical sections of code may be limited to a single manually inserted statement. This technique may also be used to gather performance data over a significant period of time, with the automatic collation of minimum, maximum and average execution times. The overhead of a single off-chip write is minimal, and is easy to calculate, making the measurements that this technique provides highly accurate and deterministic.

This technology provides the best method for monitoring critical sections of code and for general code optimization, by providing application level profiling data that identifies where the system is spending its time, ensuring that optimization efforts are focused on the right areas.

The 'call-pair' data provided by this technology may also be used to improve software performance. 'Call-pairs' measurements identify highly inter-dependent functions that make good candidates for either inlining, fixing in cache, being located close to one another in the link map of the application.

This technique has been successfully used in the CodeTEST product for Performance, Coverage, and Memory analysis, in addition to Software Execution Trace.

6. Software-Assisted Software Performance Profilers

Worthy of mention because of their dominance in the desktop marketplace, software-assisted performance profilers use a variety of techniques for monitoring where an application is spending its time. If this technology is ever used during the development of a Real-Time system, it is used to aide optimization efforts, and not to measure any of the Real-Time characteristics of the code.

Typically consisting of an in-target data collection agent and either code instrumentation or stack/IP sampling, the potential of this technology is intriguing for two reasons. First, these techniques do not require any off-chip accesses in order to make their measurements. Secondly, solutions based on these techniques tend to be extremely easy to use.

On the other hand, these techniques rely on a target based data collection agent, which is intrusive. Any techniques based on stack/IP sampling are also prone to aliasing, and in require higher levels of intrusion to improve their accuracy.

7. What Level of measurement accuracy is required?

For Real-Time systems, 'Real-Time' does not necessarily equate to 'real-fast'. The environment in which a system must operate dictates the performance criteria of Real-Time software. A pacemaker, for instance, must respond to specific physiological events within a specific time period before permanent damage to the heart ensues (response times in the 100's of mS). Meanwhile, a commercial flight control system must process and respond to thousands of inputs a second, from pilot commands to air data (response times in the mS).

With modern CPUs capable of processing in excess of 2 billion instructions per second, is it really necessary to measure software performance on a per instruction basis?

The simple answer is no, provided that:

- Worst-case response/execution times of a system are monitored, verified and managed
- Enough information is to hand during software creation to ensure that the system performance objectives can be met.

From this, then, the question then arises whether this is achievable with CPUs utilizing on-chip cache.

For extremely high accuracy software performance measurements of worst-case execution time (e.g. nS accuracy), Logic Analyzers may be used. Alternatively, if uS accuracy of software performance is required, then hardware assisted software performance monitoring technologies show the most promise. The only question is whether the performance impact of the off-chip writes that these technologies require is prohibitive, or not. This is worth a more detailed consideration.

When measuring the worst-case execution time of a critical section of code (e.g. the main loop in a control function) using this technology, a single write statement is required. Timing is started when the write occurs the first time, and then the interval between each occurrence is timed. But what overhead does this introduce?

Consider a typical environment where a target system is using a 100 MHz external CPU bus that requires 3 clock cycles to complete a write operation. In this instance, the delay imposed by each write operation would be a deterministic 30 nS.

The impact of a 30nS delay per cycle in the time critical code of a Real-Time system is negligible. The impact of being able to deterministically measure the worst-case execution time of the software under measurement with uS accuracy, however, is not. This lends great credence to the power of hardware assisted software performance monitoring technologies, especially when these technologies may be used to gather timing information on the critical sections of code over a significant period of time, ensuring the true worst-case execution time is understood.

8. Conclusion

It is an age-old dilemma in science; how can you measure something without affecting it? When it comes to measuring the performance of Real-Time software, the simple answer to this is – you can't. Add a CPU that utilizes on-chip cache, and the situation only gets worse. It is imperative, therefore, that the right performance measurement technique be used for the software being created. If the Real-Time nature of the software under development requires a timing accuracy in the nS range, then a Logic Analyzer must be used for software performance measurements. It must be understood, however, that data can only be gathered over a limited measurement period. Therefore, careful consideration must be made in the creation of the stimulus or circumstances to make sure that the worst case scenarios are represented for measurement and analysis.

Traditionally, Logic Analyzers required intimate knowledge of memory implementations on the target hardware, thus they provided very little functionality for software engineers. However, new products such as LA Trace from Wind River Systems abstracts the bus implementation from the user making it easy for software engineers to configure the circuitry of a Logic Analyzer to make complex timing measurements. Furthermore, Wind River's LA Trace is able to leverage RTOS knowledge to present acquired information relative to RTOS threads and events.

On the other hand, if you want information in the uS range, use the type of hardware assisted software performance monitoring technology available with the CodeTEST product from Applied Microsystems. This not only provides accurate one-shot timing information, but it also gathers performance information over an indefinite period of time, ensuring that the worst-case execution time of the software being measured is encountered. In addition, the same technology provides function profiling data that greatly enhances optimization efforts, and call-pair information that enables immediate performance improvements through in-lining or prudent link-map ordering.

Real-Time bus trace data from in circuit emulators have traditionally the fall back solution for Real-Time software performance measurements. Most modern CPUs utilizing on-chip cache, however, only have serial JTAG emulation solutions without Real-Time bus trace capabilities. Emulators do not, therefore, provide the performance information that they once did.

Software only profiling solutions, popular in the desktop market, are too intrusive and/or inaccurate to make accurate worst-case execution time measurements for Real-Time systems. However, they do provide the profiling information that may be used to yield significant performance improvements during code optimization.

As with all measurements in science, it is impossible to measure the worst-case execution time of Real-Time software without affecting the system. Nevertheless, technologies are available that are appropriate for the required level of accuracy, ensuring that the Real-Time nature of software executing on a CPU utilizing on-chip cache can be controlled.