

Windows CE 5.0 for real-time systems

Although Windows CE exposes the same Win32 Application Programming Interfaces (APIs) present on Microsoft desktop and server operating systems, its underlying operating system architecture is completely different from its desktop cousins. Windows CE – designed from the ground up to be a small footprint, componentized, hard real-time embedded operating system – combines support for desktop application development frameworks including Win32, MFC, AT, and .NET with a real-time kernel providing the operating system primitives needed to support today's real-time embedded system designs. This article focuses on the operating system tools used to verify Windows CE 5.0 real-time behaviors.

By Mike Hall

It is understood that real-time systems are not tested with a single analysis that pronounces them correct. Testing of real-time systems is a proof by exhaustion. Developers work to gain confidence in the solution. Microsoft tools work together to further the developer's understanding of a real-time system by providing complete timing explanations of the application and operating system interactions.

There is much discussion about the definition of real time, so it's probably good to take a look at that definition. A quote from comp.realtime FAQ, by Donald Gilles at the University of British Columbia, gives the canonical definition of real time as follows:

"A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation, but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred."

CE fits 95 percent of real-time needs

Since the industrial automation applications of high speed I/O, robotics, and machinery controls often create the most demanding timing constraints, Microsoft chose to ask that community for their specific requirements for a real-time embedded operating system. General Motors Powertrain Group (GMPTG) has been the leader in implementing Open Modular Architecture Controls (OMAC) technologies in its manufacturing applications since 1986, and GMPTG later drove the formation of an OMAC User Group. OMAC reviewed hundreds of applications and found that 95 percent of systems require cycle times of one millisecond and greater. A tolerable variation on this one millisecond cycle time would be 10 percent, or 100 microseconds (μ s). With a response time averaging 50 μ s on a 200 MHz x86 system, Windows CE meets or exceeds the requirements of all but 5 percent of applications OMAC reviewed.

Large portions of industrial automation applications reviewed were driven by an external signal from a piece of machinery. This signal is presented to hard real-time applications

in the form of an interrupt. Microsoft has been encouraging Windows CE developers to put as much application code into the Interrupt Service Threads (ISTs) as possible. This leads the OMAC jitter definition to become a timing constraint on IST latencies of no more than 100 μ s. Windows CE typically provides latencies in the sub 10 μ s range. The remainder of the applications reviewed use timers to create their cycle times. This led to the requirement of 1 millisecond timers with no more than 100 μ s latency or jitter. In summary, the OMAC definition provided the following design and test requirements:

- IST latencies of no more than 100 μ s latency
- 1 millisecond timers with maximum of 100 μ s latency

It is also important to distinguish between a real-time system and a real-time operating system. A real-time system contains all elements, including hardware, operating system, and applications. A real-time operating system is just one element of the complete real-time system. For more information, take a look at *Designing and*

Optimizing Microsoft Windows CE for Real-Time Performance by Microsoft.

Servicing interrupts

A key feature of kernel real-time performance is the ability to service an interrupt within a specified amount of time. Interrupt latency refers primarily to the software interrupt handling latencies – that is, the amount of time that elapses from the time that an external interrupt arrives at the processor until the time that the interrupt processing begins.

Windows CE interrupt latency times are bounded for threads locked in memory, if paging does not occur. This makes it possible to calculate the worst-case latencies – the total times to the start of the ISR and to the start of the IST. The total amount of time until the interrupt is handled can then be determined by calculating the amount of time needed within the ISR and IST.

ISR latency

ISR latency is the time from when an IRQ is set at the CPU to when the ISR begins to run. Three time-related variables affect the start of an ISR:

- A = Maximum time that interrupts are off in the kernel. The kernel seldom turns off interrupts, but when they are turned off, it is for a bounded amount of time.
- B = Time between when the kernel dispatches an interrupt and when an ISR is actually invoked. The kernel uses this time to determine what ISR to run and to save any register that must be saved before proceeding.
- C = Time between when an ISR returns to the kernel and the kernel actually stops processing the interrupt. This is the time when the kernel completes the ISR operation by restoring any states, such as registers, that were saved before the ISR was invoked.

The start of the ISR being measured can be calculated based on the current status of other interrupts in the system. If an interrupt is in progress, calculating the start of the new ISR to be measured must account for two factors: the number of higher-priority interrupts that will occur after the interrupt of interest has occurred and the amount of time spent executing an ISR.

Both Windows CE and OEM code affect the time to execute an ISR. Windows CE and OEM code combined are in control of the variables A, B, and C, all of which are bounded.

IST latency

IST latency is the time from when an ISR finishes execution – that is, signals a thread – to when the IST begins execution. Four time-related variables affect the start of an IST:

- B = Time between when the kernel dispatches an interrupt and when an ISR is actually invoked. The kernel uses this time to determine what ISR to run and to save any register that must be saved before proceeding.
- C = Time between when ISR returns to the kernel and the kernel actually stops processing the interrupt. This is the time when the kernel completes the ISR operation by restoring any state, such as registers, that were saved before the ISR was invoked.
- L = Maximum time in a kernel call (Kcall).
- M = Time to schedule a thread.

The start time of the highest-priority IST begins after the ISR returns to the kernel and the kernel performs some work to begin the execution of the IST. The IST start time is affected by the total time in all ISRs after the ISR returns and signals IST to run. The resulting start time is as follows:

Start of highest priority IST =

$$C + L + M + \sum_{x=0}^{N_{ISR}} (E + C + T_{ISR_N})$$

Both Windows CE and the OEM affect the time required to execute an IST. Windows CE is in control of the variables B, C, L, and M, all of which are bounded. The OEM is in control of N_{ISR} and T_{ISR_N} – both of which can affect IST latencies.

Windows CE also adds restrictions to ISTs: The event handle that links the ISR and IST can only be used in the WaitForSingleObject function. Windows CE prevents the ISR-IST event handle from being used in a WaitForMultipleObjects function, which means that the kernel can guarantee an upper bound on the time to trigger the event and time to release the IST.

Microsoft analysis tools

A number of tools are included with Windows CE that can be used to determine the interrupt timings, application execution behavior, operating system feature timings, and scheduling timing.

- Kernel Tracker: This tool provides a visual representation of the execution of the Windows CE .NET operating system on a target device. This tool can be used to view thread interactions, internal dependencies, and system state information in a real-time environment. For the purposes of this article, the interaction between interrupts, threads, and processes is examined.
- OSBench: This tool enables developers to collect timing samples to measure the performance of the kernel by conducting scheduler performance-timing tests.
- ILTiming: This tool determines the ISR and IST latencies for the target platform. ISR latency is the time from a hardware interrupt to the time of the first Interrupt Service Routine instruction. IST latency is the time from the ISR exiting and the Interrupt Service Thread starting.

Several places within a device can affect real-time performance, including hardware, drivers, and applications. Any thread that requires real-time behavior must prepare all of its resources at startup. That includes the code it runs – it must be non-pageable, and it must also be careful not to share resources (such as critical sections) with non-real-time threads or to call operating system APIs except for those that are defined for use in real-time situations.

Kernel Tracker

The Remote Kernel Tracker can be used to examine the interaction between processes, threads, and interrupts on a running device. Figure 1 depicts an example application that walks the file system of the Windows CE device, one of the folders within a Windows CE operating system image that is mapped to the build release folder on the development desktop machine. The application generates a Kernel Independent Transport Layer (KITL) interrupt for each file residing on the desktop PC. The interaction of the

interrupts and application on the running operating system image can be seen, and the time delta between the KITL interrupt and the application thread can also be determined.

In Kernel Tracker, the user interface is divided into three areas, the left pane shows interrupts and processes, the center pane shows the thread/process interaction, and the right pane (not shown) is a key to the symbols used in the center pane. The walktree application is running (at the bottom of the image), but exactly how much time is being spent in the context of the application and the kernel is unknown.

Kernel Tracker provides the ability to set time markers between events and displays the time difference in the status bar of the Kernel Tracker application. There are a number of predefined events, covering synchronization, miscellaneous, and user defined. The thread state is also displayed, which shows threads running, blocked, sleeping, and migrating. In Figure 2, the first time marker is set when the execution of the thread returns from the kernel, and the second time marker is set at the point when the thread context switches back to the kernel.

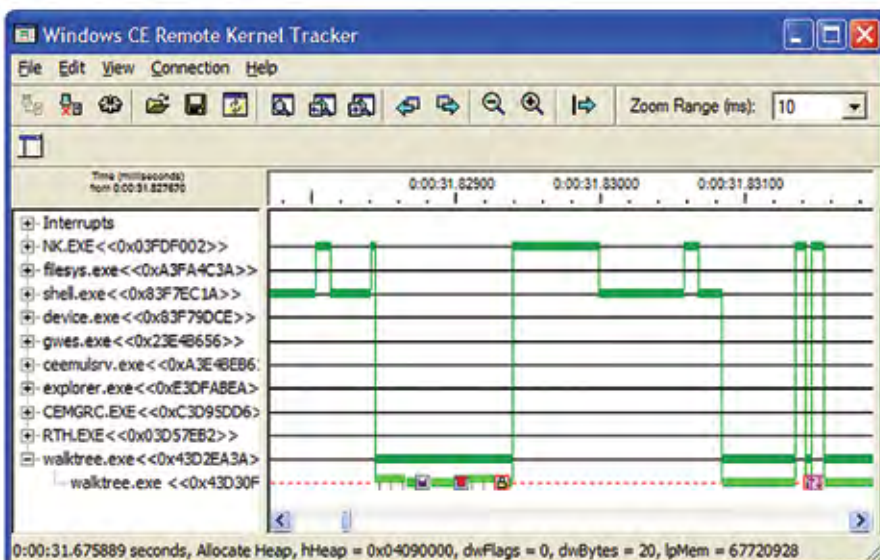


Figure 1

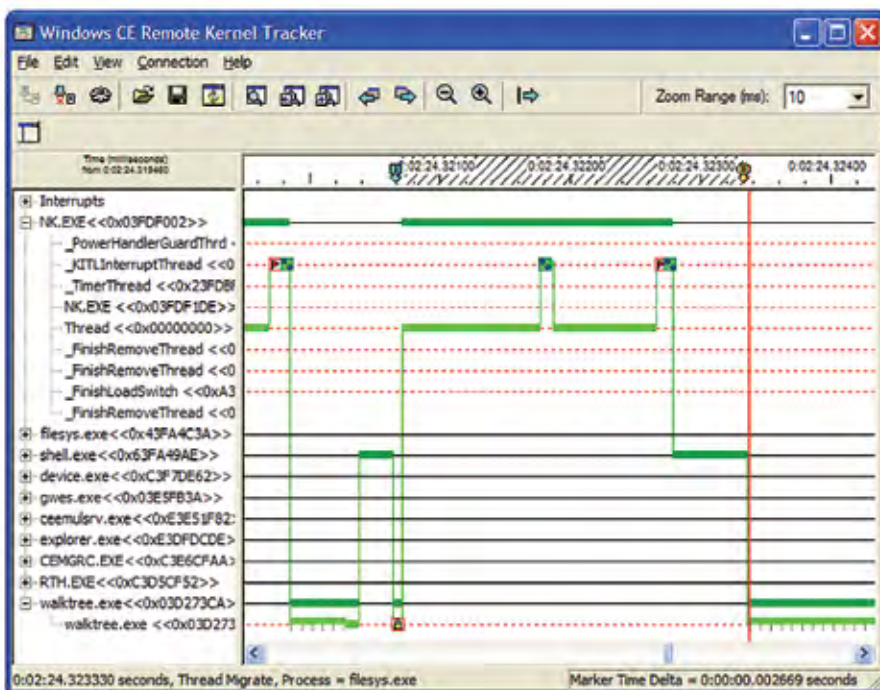


Figure 2

The Kernel Tracker tool can be used to locate and examine deadlock situations, and to examine the amount of time being spent in your application and driver threads. Running Kernel Tracker does affect the overall timing of the operating system, perhaps adding an additional 2 to 3 percent of overhead to the system.

OSBench

OSBench provides benchmarking for an extensive set of operating system conditions from the overhead of an internal Protected Server Library (PSL) call from an application into one of the processes of the operating system (FileSys.exe, Device.exe, and so on). The tests are broken down into seven basic groups as follows:

- CriticalSections
- Event set-wakeup
- Semaphore release-acquire
- Mutex
- Voluntary yield
- PSL API call overhead
- Interlocked APIs (decrement, increment, testexchange, exchange)

As an example, let's look at the output of a couple of tests:

```
=====
| 0.01 | IP = NO | CS = NO | 1 IPS
-----
EnterCriticalSection traditional (blocking) without priority inversion :
Time from a higher priority thread calling EnterCS (blocked) to a lower priority runnable
thread getting run
-----
| Max Time = 13.409 µs
| Min Time = 7.543 µs
| Avg Time = 8.389 µs
=====
```

```
=====
| 0.02 | IP = NO | CS = NO | 1000 IPS
-----
EnterCriticalSection fastpath (uncontested)
-----
| Subtracting out base result of 12 ticks
| Max Time = 0.064 µs
| Min Time = 0.061 µs
| Avg Time = 0.061 µs
=====
```

These tests compare the amount of time it takes to process a call to EnterCriticalSection. There are two paths through that function. The first exercises the critical section with contention and causes the implementation to transition to the kernel to handle the contention. The second remains entirely in the calling process when there is no contention for the critical section and is obviously a lot faster. (This explains why critical sections are preferred for synchronization.)

ILTiming

ILTiming measures the Interrupt Latencies in a system. It uses a number of OEM Adaptation Layer (OAL) support capabilities to measure the response times to interrupts for the ISR and IST. These numbers are critical for understanding the constraints of the system, and come as close as possible, with a software-only solution, to measuring the results that can otherwise only be obtained by using an external hardware probe.

Table 1 shows the numbers achieved on an AMD K6 500 MHz-based Windows CE PC (CEPC) system.

	ISR Starts	IST Starts
Minimum	1.6 μ s	7.5 μ s
Average	1.6 μ s	8.4 μ s
Maximum	2.5 μ s	36.0 μ s

Table 1

Windows CE conforms to, and often exceeds, the OMAC definition of a hard real-time operating system, and ships with the tools and resources needed to build, test, and deploy a real-time device. All of these tools – Kernel Tracker, OSBench, and ILTiming – work together to help evaluate the real-time capabilities of Windows CE on a target platform.

ECD

Mike Hall is a technical product manager in the Mobile and Embedded Devices Group at Microsoft. Mike has been working at Microsoft for 10 years, originally in Developer Support, focusing on C/C++, MFC, COM, device driver development, Win32, MASM, and Windows CE operating system development, and then as a systems engineer in the Embedded Devices Group. For the last three years, Mike has been in the Embedded Devices Platforms Group, working with Windows CE and Windows XP Embedded. Mike writes a monthly MSDN column on embedded systems development and presents at a number of Microsoft and third party events on Microsoft embedded technologies.



To learn more, contact Mike at:

Microsoft
 One Microsoft Way • Redmond, WA 98052
 Tel: 425-707-5223
 E-mail: mikehall@microsoft.com • Website: www.microsoft.com

Further Reading

Real-Time Behavior of the .NET Compact Framework

Maarten Struys

Michel Verhagen

PTS Software

http://msdn.microsoft.com/library/en-us/dncenet/html/Real-Time_NETCF.asp

Dedicated Systems, Windows CE 5.0 Real-Time x86 Processor

<http://download.microsoft.com/download/7/2/f/72fef3b0-9545-46a4-8886-a94f265df9c4/EVA-2.9-TST-CE-x86-01-Iss1.00.pdf>

Dedicated Systems, Windows CE 5.0 Real-Time ARM Processor

<http://download.microsoft.com/download/7/2/f/72fef3b0-9545-46a4-8886-a94f265df9c4/EVA-2.9-OS-CE-01-I01.pdf>

Windows CE 5.0 Real-Time Podcast interview with John Eldridge, Architect, Windows CE

<http://blogs.msdn.com/mikehall/archive/2005/09/01/459443.aspx>

Real-Time Determinism in Windows CE

<http://www.windowsfordevices.com/articles/AT6761039286.html>