



## Using an Emulator to Debug Interrupts in Windows CE

One of the major issues in developing a new embedded system is working out the complexities of the interrupt system. The system depends on the interrupt system to "connect" it to the real world. This happens because it is through the interrupt system that the embedded system is made aware of external stimulus.

In a motion control system it is the interrupt system that notifies the system that the position requested has been reached. It is the interrupt system that notifies when a hard positional limit has been reached. In communication systems it is the interrupt system that notifies when the input has been acquired. The interrupt system also notifies when a communication line is ready to accept a new packet. In short the interrupt system is central to all of the input and output of the developer's design.

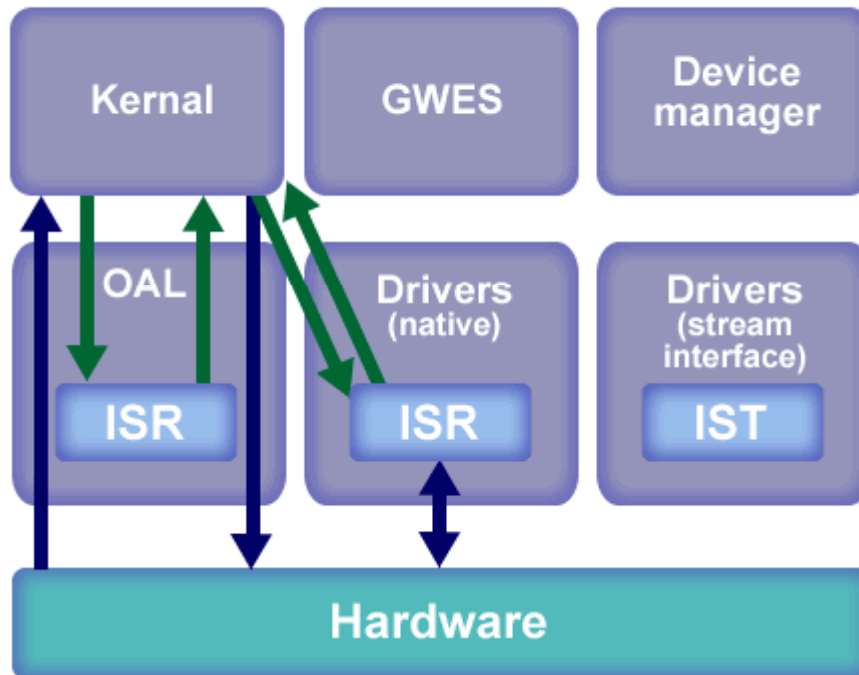
What makes this portion of the system complicated is its asynchronous nature. The designer cannot tell when an input will arrive, when a stop will be reached or when a line will clear to allow a send. This means the interrupt system is temporally overlaying the rest of the design. To work properly it must meet the external requirements in terms of time for the inputs and outputs to be accepted, while not interfering with the rest of the work the embedded system is required to perform.

The nature of the interrupt as described above makes it a particularly difficult portion of the system to debug. And in some environments it becomes the focus of the development to get the interrupts to work. In a development environment like Windows CE other complexity is present. The operating system is not only hosting the application being debugged, it is also hosting the debugger. So when you have a bug in the interrupt system, the operating system continues to run to host the debugger. The result is some of the system environment changes during your debug. This can be harmless, but in some cases the part of the system that caused the bug in the interrupt system may be masked by the debugger's presence.

## The Nature of Interrupts in Windows CE<sup>1</sup>

The nature of interrupts in the Windows CE Operating system can also mask problems associated with interrupts. An interrupt is initiated in the following manner.

### Interrupt Handling



1. The external event occurs which initiates the interrupt.
2. An interrupt is detected by the kernel, which disables this interrupt ONLY
3. The Kernel invokes an Interrupt Service Routine (ISR) within the OAL
4. The ISR returns a logical ID
5. The Kernel unblocks the IST within the driver
6. The IST services the interrupt
7. The IST signals end of interrupt to the kernel
8. The kernel re-enables this interrupt

The interrupts are implemented as tasks running on the operating system. This means that when an error occurs the operating system

will treat the errant interrupt like it does any tasks that has failed. The resulting operating system operations sweep your debug data away before you can evaluate the crash.

Our emulator can assist in simplifying these issues. First you have control of the processor. You can halt it in its tracks. This allows you to view memory, registers, pointers and all other relevant data without the operating system modifying it. This can uncover some of the problems that might otherwise be lost using a debugger. Further help comes from the trace and trigger system. By setting triggers on error routines, you can halt the processor when it has encountered an error. Then using the trace system you can look back on the executed code to determine where the problem began. Using this method you can review the code and data for problems and solve your debug issue.

1. Gareau, Jean "Writing Windows CE Device Drivers" presented at the 2000 Embedded Systems Conference October, 2000.