

# Re: Device Drivers and all for one interrupts, can it be done.

---

*Source:*

<http://www.tech-archive.net/Archive/WindowsCE/microsoft.public.windowsce.embedded/2007-07/msg00113.html>

---

- *From:* "Steve Maillet \ (MVP\)" <nospam@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>
  - *Date:* Thu, 12 Jul 2007 16:40:00 -0400
- 

What you have describe is the scenario generally called a Shared IRQ, however since it is a GPIO pin on a port of multiple(16) pins it is a special case. The BSP must be written to support that. There are 2 approaches to how that is generally handled.

1) The OAL is written to know about the 16 bit GPIO port and actually figures out which pin is interrupting the system (when more than one is used the oal can use a count leading zeros approach to define a "priority"). It then translates that to an internal IRQ number that a device driver can request a SYSINTR for which the OAL will assign dynamically at runtime. This approach keeps things simple and keeps the driver from having to deal with any hardware specific issues. We highly recommend this approach for all processor internal GPIO ports and any external ones known to the BSP writer. It allows the greatest re-usability of code across systems. (This approach is implemented in all of our FusionWare::BSP implementations and has worked very well for a number of years)

2) The second approach involves the use of an installable ISR. Basically the OAL has an ISR for the GPIO port itself and then calls all Installable ISR routines registered for that shared interrupt. Each one in turn can check the GPIO port register status to see if it's device is interrupting the system and return the SYSINTR for the device (The SYSINTR is provided by the system when you register for the installable ISR). Microsoft provides the Generic Installable ISR (GIISR) for this. Unfortunately it's not all that generic. It's quite useful for PCI based devices but not all that useful for most SOC internal devices nor for external direct memory mapped devices either. So you need to create your own drive Installable driver. The difficult part of that (and why we recommend and implemented #1 ourselves) is that the UART device driver now needs to "KNOW" about the IISR and provide it the correct information it needs to run on that platform. But there is no standardized method for doing that. (At least not one defined by MS. All of our BSPs have a defined method of storing a BLOB of data in the registry that a driver can detect and if present pass it on blindly to the installable ISR to deal with thus the driver need not "know" about the details of the IISR) So the driver for a UART now has to become very platform specific instead of being platform agnostic.

"Shamus" <Shamus@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message  
<news:A389A91B-7BBE-456F-A275-D1067D39CC8C@xxxxxxxxxxxxxxxxxxxx>

The Info:

Let s say you have a processor (Arm 9315) running Windows CE 5.0 and you have general purpose IO (GPIO) configured as an interrupt.

In other words you have a 16 bit port that triggers a single hardware interrupt if one of the GPIO bits goes high. So when the interrupt occurs you read the GPIO and figure out which one, piece of cake.

Re: Device Drivers and all for one interrupts, can it be done.

Ok now let s say you have several serial UARTS (say 4) attached to 4 of the GPIO bits. You then write a driver for each of the 4 UARTS except in the registry for the driver you have the same interrupt number assigned to each UART.

The question:

How does the driver know which software interrupt vector to use when the interrupt goes off? It almost appears that when a serial port is opened that becomes the interrupt vector driver of choice and no other serial ports that use that interrupt are allowed to open. However closing the serial port seems to un-allocate the interrupt vector and allow it to be assigned again to say another serial port. I cant say for sure what causes the other serial ports not to open when one is already open, they all appear to all have there own handles when you open them up one at a time.

Because there is only one hardware interrupt that must service several serial port UARTS do I need to do the GPIO parsing in the interrupt.c file itself and then trigger some sort of software based interrupt that the serial drivers act upon? If so how does one go about this?

I tried using the answer to life, the universe, and everything else (42) but it does'nt seem to be working.