The Exar UART drivers and Power Management
June 27, 2005

The Exar UART driver (xr788.o) along with the 'core' serial driver which is built into the Linux kernel are responsible for the behavior of the serial ports during sleep mode. The first thing to realize is that the core layer, by default, closes all serial ports that are registered with it when 'suspend' mode is entered. This has always been the way that the internal UARTS (ttySA0, ttySA1, ttySA2) on the SA1110 operate, and due to their hardware configuration, they are incapable of waking up the CPU from sleep anyway. So this is a perfectly reasonable thing to do.

Wakeup on serial character received

Enter the requirement of waking up from sleep. It is certainly possible, since an interrupt edge from one of the Exar UARTs can bring the SA1110 out of sleep. This, however, means that we have to disable the default mechanism in the 'core' driver that closes serial ports, since this would disconnect the interrupt and we would not receive a wakeup indication. This is the most significant modification of the core and xr788 drivers, and is one of two steps that you must take to enable wakeup on a serial character.

The current version of the driver operates in this mode by default, so no action here is necessary. The second thing that you must do is set the PWER register properly in your ./suspend.sh script so that the interrupts associated with the Exar serial ports are allowed to wakeup the CPU. This must happen because once the Exar chip receives an interrupt, it must be serviced before the CPU is to fully exit from sleep. This is because of a 'feature' in the linux power management support for the SA1110 that clears all edges encountered while sleeping. This feature was originally intended to clear an edge that was generated by a push button, which was a perfectly reasonable thing to do on a PDA. Our case is a bit more complicated in that since the edge is cleared on exit from sleep, the condition that caused the edge must be completely handled in the 'resume' handler of the driver. In order for this to happen, the driver has to run when an edge occurs and in order for this to happen, the PWER bits have to be set properly. The default value for CIMT was 0x8000_0001. You must modify this value to 0x8000_000D (without the '_' -- I added this for readability). So to summarize for **wakeup on character received capability:**

[1] allow the driver to load with it's default settings
[2] **set PWER to 0x8000000D in your suspend script**

Sleep and ignore spurious serial traffic

Instruments like the chatty GPS can keep a SideARM from sleeping and introduce a lot of extra and unnecessary system state transitions (awake->sleep->awake). This will reduce the overall responsiveness of the system because there is overhead associated with going in and out of sleep mode so frequently. Waking up from sleep mode is actually a flavor of reset which exercises code from the bootloader, all the way up to the Linux kernel, and into the serial driver. In this case, the default action of closing the serial port and preventing it from interrupting the sleeping SideARM is preferable.

Currently, the Exar drivers are loaded by the **`/etc/init.d/loadxr`** script, with one invocation of insmod for each of the octal UART chips. In order to allow the core serial driver to put the Exar uarts to sleep, as it does the internal uarts in the SA1110, you need to add the following text to each insmod invocation after the driver name:

`no_core_pm_ops=0`

This will be among options, such as **`chipirq=3`** for the second invocation of the driver, however, you must add it to **both** invocations in loadxr. This represents the CIMT configuration, and has been the configuration that has been running robustly in building G for a few weeks.

While wakeup on serial character is supported to the best degree that I can support it with the current Linux kernel, the power management code was clearly not designed for this eventuality in mind. I would recommend it's use on low volume, low periodicity serial traffic occasions, like two BINs communicating with each other over RS485, but I would not recommend running it on a surface node, for instance, where a chatty GPS will be waking a sleeping SideARM with 1 second periodicity.