Opening up new opportunities

How software developers can help to optimise the power consumption of an embedded system. By Lotta Frimanison and Anders Lundgren.

Power consumption has traditionally been something influenced only by hardware developers. But power consumption depends not only on the hardware, but also on how it is used and how it is controlled by the system software. Through power debugging – coupling source code to power consumption – it becomes possible to test and tune for power optimisation.

The approach is based on the ability to sample power consumption and to correlate each sample with the program’s instruction sequence and, hence, with the source code. One difficulty is achieving high sampling precision. Ideally, power consumption should be sampled at the system clock’s frequency, but power system capacitances reduce the reliability of such measurements. From the software developer’s perspective, it is more interesting to correlate power consumption with source code and various events in the program execution than with individual instructions, so the resolution needed is much lower.

Power is measured by the debug probe – IAR’s J-Link Ultra, for example (see fig 1). This measures the voltage drop across a small resistor in series with the supply power to the device. The voltage drop is measured by a differential amplifier and sampled by an a/d converter.

The key to accurate power debugging is a good correlation between the instruction trace and the power samples. The best correlation is with a complete instruction trace, but this is not available in all devices and, even if it is, a special debug probe is often required.

Less accurate, but still giving good correlation, is to use the PC (program counter) sampling facility found in some on chip debug architectures (see fig 2). The PC is sampled periodically and each sample given a time stamp. The debug probe samples device power consumption using an a/d converter and, by time stamping the sampled power values and the PC samples, it is possible for the debugger to present power data on the same time axis as graphs like interrupt log and variable plots, and to correlate power data to source code.

In general, optimising for power is similar to optimising for speed. The faster a task is executed, the more time can be spent in a low power mode. So by maximising idle time, power consumption is reduced.

Frimanson: “...polling device status change should be solved using interrupts.”

Lundgren: “The faster a task is executed, the more time can be spent in a low power mode.”
Battery life can be extended significantly.

Active for a small amount of the total time, so by produced while little is being accomplished. In running at full speed when idle, battery life is ridgeting for something to happen. If the processor waits for something to happen. If the processor

Embbeded applications spend most of their time compared to a cpu driven polled approach. ebugger, see what effects DMA techniques will allows the developer to experiment and, via the mode during the DMA transfer. Power debugging makes it easier to discover the increase in power consumption when the interrupt hits and to identify it as abnormal.

• DMA versus polled I/O

Direct memory access (DMA) has traditionally been used to increase transfer speed. In some architectures, the cpu can even be put into sleep mode during the DMA transfer. Power debugging allows the developer to experiment and, via the debugger, see what effects DMA techniques will have compared to a cpu driven polled approach.

• Low power mode diagnostics

Embedded applications spend most of their time waiting for something to happen. If the processor is running at full speed when idle, battery life is reduced while little is being accomplished. In many applications, the microprocessor is only active for a small amount of the total time, so by placing it in a low power mode during idle time, battery life can be extended significantly.

A good approach is to use task oriented design and an rtos. In task oriented design, a task defined with the lowest priority will only run when no other task needs to run. This idle task is the perfect place to implement power management. In practice, every time the idle task is activated, it puts the processor, or parts of it, into one of a number of low power modes.

• CPU frequency

Power debugging allows the developer to verify power consumption as a factor of the clock frequency. A system that spends little time in sleep mode at 50MHz is expected to spend 50% of the time in sleep mode when running at 100MHz. The power data in the debugger will allow the developer to verify expected behaviour and, if there is a non linear dependency, to choose the operating frequency that gives the lowest power consumption.

• Interrupt handling

Figure 3 shows a schematic diagram of the power consumption of an event driven system where the system at t_3 is in an inactive mode and the current is I_0. At t_1, the system is activated and current rises to I_1 which is the system’s power consumption in active mode with one used peripheral device. At t_2, execution is suspended by an interrupt handled with higher priority. Active peripherals are not turned off, although the thread with higher priority is not using them. Instead, more peripherals are activated by the new thread and the current rises to I_2 between t_2 and t_3, when control is handed back to the thread with lower priority.

While system functionality could be excellent and maybe further optimised in terms of execution speed and code size, more optimisations can be done in the power domain. The highlighted area represents the energy that could be saved if the peripherals that are not used between t_2 and t_3 are turned off, or if the priorities of the two threads can be changed.

Power debugging makes it easier to discover the increase in power consumption when the interrupt hits and to identify it as abnormal.

• Conflicting hardware

It is common practice to avoid floating inputs by tying unused mcu I/O pins to ground. If, by mistake, the software configures a grounded I/O pin as a logical ‘1’ output, a current as high as 25mA may be drained on that pin. This is easily observed by reading the current value from the power sampling display; it is also possible to find the corresponding erratic initialisation code by looking at this display during application startup.

Conclusion

Power debugging provides embedded developers with the ability to see inside their application and to determine how program code and flow influences power consumption. With this knowledge, source code can be tuned and optimised to minimise the power required, making designs as energy conservative as possible without impacting performance.

Author profiles:

Lotta Frimanson and Anders Lundgren are with IAR Systems (www.iarsystems.com).

For further information on any subject visit: www.newelectronics.co.uk