

Task and Interrupt LEDs

Optimization of access

Thu, Jul 11, 2002

System software operates digital output lines that can be monitored for timing of software activities. In order to turn on or turn off a LED, the usual code reads from the LED register, sets or clears the target bit, and writes to the LED register. In the IRM code, this is normally done with a BSET or BCLR (68K) instruction, which performs the read and write cycles that are necessary.

For the PowerPC implementation, however, the RISC architecture includes no such read-modify-write instructions, so the C code explicitly reads from the LED register, modifies the bits as appropriate, and writes to the LED register. In an interrupt routine, for example, this can mean four memory accesses to the LED register. This would not be significant except that access to the LED register, which is part of the Digital PMC board, is comparatively slow, requiring about 1 μ s per access. Assuming that we want to continue to operate these LEDs for possible diagnostic timing purposes, even though it is seldom actually used, there may be a way to reduce the number of accesses, which is the subject of this note.

Suppose there is a mirror interrupts-LED register value that is maintained in fast memory (dynamic RAM), so that the read cycles above can be replaced with a read of the mirror value. The logic would proceed as follows to set/clear a bit:

- Read mirror register in fast memory
- Set/clear target bit
- Write to mirror register
- Write to LED register

The accesses to fast memory are comparatively free, so that the only time-consuming part of this sequence is the write to the LED register. Of course, for interrupt timing, there would be one such sequence at the start of the routine and one at the end.

One may wonder whether this can be done accurately, since there are separate steps to set/clear a bit that can be interrupted by a higher priority interrupt. But that is a property of the CPU architecture and not a property of this particular sequence of operations.

During task switching, a similar procedure can be followed. A global variable of the current tasks-LED register is kept in fast memory. First modify the memory variable, then write to the hardware LED register.

To prevent concern about somehow "getting out of sync," one may choose to clear the two global mirror variables during the lowest priority task. If the CPU is actually executing this task, none of the LEDs should be in use, at least if we restrict their usage to timing the activity of interrupts and tasks.