Watchdog Timers Keep Computer Failures from Having Catastrophic Results

Digital computers do only what they're told - or what they think they're told. It's not unheard of for a computer to suddenly latch up or get lost in a program with potentially catastrophic results if it's controlling a critical process. To minimize the damage of a computer failure, good designs include a watchdog timer - a device that monitors host activity and triggers appropriate action if it notices that the computer isn't functioning as expected. These devices come as low-cost ICs or modules and are not only easy to integrate into a system, but it's highly recommended you do so in any control application.

Watchdog timers easily integrate with Industrial I/O Control Systems

Implementation strategies

You can interface your system to a watchdog timer in several configurations. An easy yet effective way to detect a computer fault is by programming one or more digital output bits to toggle On/Off continuously, and if that bit stops at one state longer than a designated time period, then the watchdog timer monitoring the signal assumes a computer fault. An easy way to hook up a watchdog in a PC-based system is to add a digital I/0 board and configure one of its output lines to toggle the watchdog.

When any watchdog detects a failure, there are three major courses of action it can take: reset, alarm or shutdown.

Reset or an attempt to reset is a common approach in less-than-critical applications such as running a business application or simple datalogging where a reset can establish system recovery. In such cases where a computer fault is recoverable, the fault and the action of the watchdog timer are often transparent to the user. On a PC, for instance, the watchdog could activate the reset line (the same as hitting <ctrl> <alt> <del>.

When implementing a watchdog reset, especially in control applications, examine the overall impact of the sudden initialization of the I/O (hot start) and loss of data in RAM. Consider, for example, the simple case where a microcontroller acts as the brain of a toaster oven and controls temperature and toasting time. If the computer latches up and is reset by a watchdog while toasting a melted-cheese sandwich, the fact that it was in the process of toasting and the elapsed toasting time would likely be information previously contained in RAM and might be difficult to ascertain. Thus, such a design should also contain circuitry to monitor the temperature to see if the oven is already hot when the system resets, or use a nonvolatile memory to store timing data.

Alarming is generally used along with other watchdog functions and is common in applications where a human operator must determine why a computer fault occurred and manually clear the alarm. A good example is a controller for a hydroelectric power plant. Regulation of turbine flow and other parameters likely requires PID (proportional integral derivative) algorithms and special arithmetic functions. Here, a watchdog alarm should alert a skilled operator to manually control the situation until the computer again comes up to speed. This action is the most appropriate because it's unlikely you'd want to shut down the power plant because of a computer fault. However, if the computer is reset with a hot start such as noted above, it might initialize variables in the control algorithms and lose track of phase, flow and other vital parameters.

Shutdown in response to a watchdog signal is common for critical industrial-control applications where a computer fault could have unpredictable results, and it's generally used in conjunction with the Alarm function. Shutdown implies that a human operator must investigate the cause of the computer fault and manually clear it. Note in this mode that the watchdog should deactivate devices being controlled, such as motors or heaters, but not necessarily the computer. Depending on the type of equipment being controlled, it might be wise to have the watchdog initiate a predetermined shutdown sequence to prevent damage to a process or equipment. Which level of response you want—reset, alarm or shutdown—also plays a role in how you implement the watchdog timer. Consider, for example, watchdog ICs that are sometimes included on function cards for industrial PCs. These devices might monitor power-supply voltages and have a toggle input. When they detect a computer failure, they simply strobe a Reset line and don't have any means of activating an alarm or shutting down a process. Further, most watchdog IC designs place the device local to the system CPU, and because they don't monitor the control output, they leave much of the output hardware unsecured.

Watchdogs with relay outputs are better suited to control applications because they can be implemented local to the control outputs. They can provide fault indication and the correct response to most failure modes, including loss of power or control signal. It's also easy to cascade multiple watchdogs with relay outputs. However, these devices are relatively large and may be costly for systems that need only low levels of system integrity.

When implementing a watchdog timer, take care when determining the system output it monitors. You should decode that signal to at least the level of the control outputs, if not beyond. Many control systems require much less I/O than the computer is capable of, so you might be tempted not to perform address decoding to the highest possible level. For instance, assume that a watchdog timer monitors an output bit mapped at binary address X111XX00 (where X indicates a don't care). Because for each one of those undefined bits you can come up with an absolute address, the watchdog—rather than reading just one address—is toggled by a number of addresses, specifically hex $70, $74, $78, $7C, $F0, $F4, $F8 and $FC. The chances of a runaway program hitting the one absolute address that toggles the watchdog timer and fools it into thinking everything is functioning properly is relatively low, but chances of a false signal to the timer are greater as you increase the number of addresses to which it responds. Thus, it's recommended that the output to the watchdog timer be a control output from one or more data bits of a decoded address (*see Figure 1*).

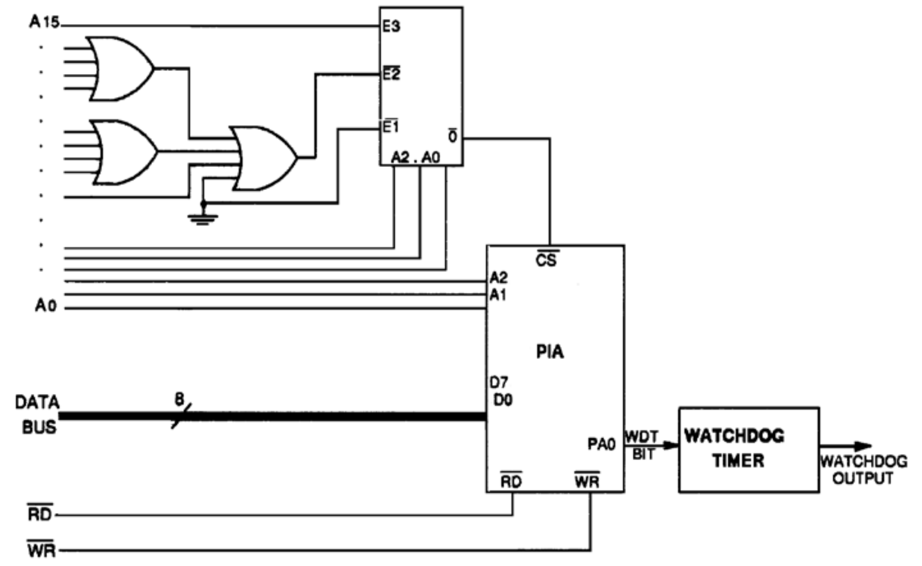

Figure 1 - Take care when decoding the address that activates the watchdog timer so that it does not receive erroneously TRUE signals.

In control applications with more than one I/O rack, you can increase system integrity by configuring a watchdog on an output of each rack and cascading their outputs. This approach assures computer control at each control branch, and thus it indicates loss of control at any rack *(see Figure 2).*

Another consideration when implementing watchdogs in control

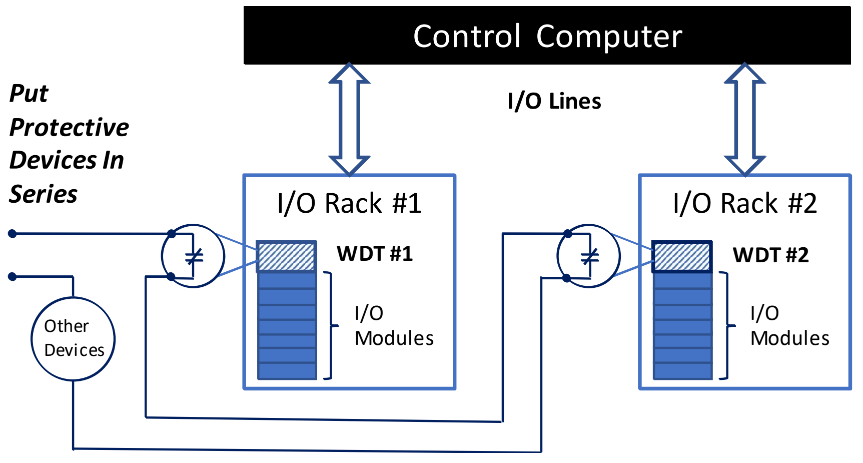

Figure 2 - When working with multiple I/O racks, it is a good idea to cascade the outputs from a watchdog timer located on each rack.

applications is to use an energized dry contact as the output. One advantage of this approach is that loss of power is detected as a fault condition, and Fail-Open is common in fault reporting because of constant continuity.

Finally, note that some designers combine watchdogs with other means of securing a computer. You can, for instance, make frequent checksums of memory, look for bus errors or feed back control outputs to verify their presence. The bottom line, though, is to never design a control computer without a watchdog timer.

Software aspects

When setting up a program with watchdog-timer capability, keep several things in mind. Foremost, remember that perhaps the most useless type of watchdog is one implemented completely in software. It should be apparent even to a casual observer that such an approach implies that the microprocessor is alive enough to run the code section that indicates a computer fault. Likewise, view with care any shutdown sequencing implemented in software if you need the highest level of system integrity.

Next, avoid toggling the watchdog with an address-decode strobe. As the above example indicates, software gone astray might coincidentally hit that address and convince the watchdog that everything's satisfactory.

Next comes the question of not *how* to toggle the watchdog output, but *where* in the software to do so. The Main program loop proves the best place for several reasons. First, that loop runs continuously and, based on certain conditions, calls other tasks or subroutines. Many situations can arise—such as noise that inadvertently changes data in RAM or sends the program unknown address values—in which the computer might get stuck in a subroutine or roam aimlessly through the address space without getting back to the main program. It's not advisable to place code that toggles the watchdog in an interrupt-service routine, for example; even if program control gets lost in the address space, an interrupt takes higher priority and sends program control to the appropriate servicing routine, which toggles the timer so that it believes the system is operating properly. Upon servicing the interrupt, though, program control returns from where it originally left—even if it's the wrong place. Under such circumstances you could toggle the watchdog even though the main program loop never gets serviced.

To see an example of how to set up a watchdog properly, consider the program in Figure 3, which uses the ROM BASIC resident on an

```
REM  MAIN PROGRAM LOOP
10      GOSUB 1000        :  REM INITIALIZE I/O
20     IF STRT=L THEN GOSUB 2000 : REM CHECK TEMP SWITCH/CONTROL HEATER
30     IF STRT=O THEN XBY(4011H=(XBY(4011H) AND.  OBFH) : REM HEAT OFF
40     WDT=ABS(WDT-1) : REM COMPLIMENT LAST WATCHDOG STATE
50     IF WDT=1 THEN XBY(4011H)=(XBY(4011H) .OR. 01H) : REM SET WDT=1
60     IF WDT=0 THEN XBY(4011H)=(SBY(4011H) .AND.FEH) : REM RESET WDT=0
70     GOTO 20

REM  INITIALIZATION SUBROUTINE
1000   ONEXT1 3000 : REM INITIALIZE START SWITCH INTERRUPT
1010   XBY(4011H)=OOH : REM INITIALIZE OUTPUTS
1020   WDT=O : STRT=O : TEMP=O : REM INITIALIZE VARIABLES
1030   RETURN

REM  HEAT CONTROL
2000    TEMP=XBY(4010H).AND.OFEH : REM CHECK TEMP SWITCH 1=ON O=OFF
2010   REM SET HEATER ON/OFF
2020   IF TEMP=L THEN XBY(4011H)=(XBY(4011H).AND.OFBH):  REM OFF
2020   IFTEMP=OTHENXBY(4011H)=(XBY(4011H).OR.02H)  :  REM ON
2030   RETURN

REM  READ START SWITCH
3000   IFXBY(4010H<>OFFH THEN STRT=1   : REM START HEAT CYCLE
3010   TIME=0 : CLOCK1 : ONTIME 55,4000
3020   RETI

REM  TIMER INTERRUPT
4000   STRT=0 : CLOCK0 : REM TIMEOUT, STOP HEAT CYCLE
4010   XBY(4011H)=(XBY(4011H).OR. 03H) : REM BEEP
4020   XBY(4011H)=XBY(4011H).AND. 0FCH) : REM STOP BEEP
4030   RETI
```

Figure 3 – It is always a good idea to keep commands that service the watchdog timer in the main program loop.

Intel microprocessor to control a toaster oven. An important command in that chip's command set is XBY, which is a direct read/write to a virtual address (such as RAM, ROM or I/O). Key in the main loop are lines 50 and 60, which use XBY to write to an I/O address, in this case an output line (address $4011, Bit 0) connected to the watchdog timer toggle.

In more detail, the first line of the program (1000) goes to an initialization subroutine that sets the start-switch interrupt and system variables. Then it moves to Lines 20/30, which check the status of the start switch. If that switch is On (STRT=1), program control goes to the subroutine at Line 2000 to control the temperature cycle; if the switch is Off, Line 30 ensures the heater is Off. Now come Lines 50 and 60 to toggle the watchdog timer and a loop statement to send program control to Line 20. When you hit the start switch, an interrupt at Line 3000 starts the cycle but sets an internal timer for 55 seconds and sets the STRT variable to One. The program loops until the 55-sec interval has passed. During that time, if the computer suffers from noise, loses track of the program, and leaves the heating coils on, the watchdog timer doesn't get strobed and that device then takes appropriate action, such as removing power from the coils.

****

Author: Brian H. Breneman, CEO and Chief Design Engineer at Brentek International, a manufacturer of watchdog timer modules and specialized I/O modules, including dry contact modules and industrial supervisory modules.

Watchdog Timer sales and support:
Industrial Products Distributing, Inc.
(800) 543-8142
www.brentek.com