Application Note 101
Using the Secure Microcontroller Watchdog Timer

www.dalsemi.com

OVERVIEW

Microcontrollers are often used in harsh environments where power supply transients, electromagnetic interference (EMI), and electrostatic discharge (ESD) are abundant. Program corruption caused by bus corruption and electromagnetic discharges can cause a microprocessor to execute erroneous instructions. In these environments, a watchdog timer is a useful peripheral that can help catch and reset a microcontroller that has gone "out of control."

A watchdog timer is a simple countdown timer which is used to reset a microprocessor after a specific interval of time. In a properly operating system, software will periodically "pet" or restart the watchdog timer. After being restarted, the watchdog will begin timing another predetermined interval. When software or the device is not functioning correctly, software will not restart the watchdog timer before it times out. When the watchdog timer times out, it will cause a reset of the microcontroller. If the system software has been designed correctly and there has been no hardware failure, the reset will cause the system to operate properly again. The reset condition must be a "safe" state. For instance, it would not be wise to have the reset state of a magnetic stripe card reader enabling the write head.

Many systems have been designed using an external watchdog timer. The Secure Microcontroller family eliminates the need for external components by incorporating an internal watchdog timer. By moving the watchdog timer inside the microcontroller, the number of devices in the system is reduced, increasing the overall system reliability. The watchdog timer can take advantage of the high-precision crystal oscillator used by the microcontroller, rather than the imprecise RC oscillator used by most independent watchdog timers. The operation of the watchdog timer is independent of the microcontroller, unless specifically addressed via the Timed Access procedure. The possibility of an out-of-control microcontroller accidentally disabling the watchdog timer is less than 1 in 7.2×10^{16}. This application note describes the features and use of the Secure Microcontroller's watchdog timer.

GENERAL USE OF A WATCHDOG TIMER

The primary application of a watchdog timer is as a system monitor to detect and reset an "out-of-control" microprocessor. When program execution goes awry it will not properly execute the code that restarts the watchdog. In such a case the watchdog timer will timeout and cause a microcontroller reset. In a properly designed system, the reset will correct the error.

Regardless of how capable a watchdog timer might be, there are certain failures that cannot be corrected by a reset. For instance, a watchdog timer cannot prevent or detect the corruption of data memory. Unless corruption of data affects program flow, or some extra measures are taken, data corruption will not cause a watchdog timeout. Of course, self-diagnostic software can be written in such a way as to make restarting the watchdog contingent on verification of data memory. While many applications implement such a data verification scheme, it is beyond the scope of this document.

It should be remembered that a watchdog timer cannot detect a fault instantaneously. By definition, the watchdog timer must reach the end of its timeout interval before it resets the processor. The system designer should be aware of the maximum time interval that can occur between the execution of a bad instruction and the watchdog timer reset.

PLACING THE RESTART INSTRUCTIONS

In the Secure Microcontroller family the watchdog timer is driven by the main system clock. The timeout interval is fixed at 122,800 machine cycles (1,473,600 external clock cycles). When the timeout is reached a reset will occur. Table 1 shows the reset time intervals associated with different crystal frequencies.

WATCHDOG TIMEOUT INTERVALS Table 1

CLOCK FREQUENCY	TIMEOUT INTERVAL
16.0000 MHz	92 ms
14.7456 MHz	100 ms
11.0592 MHz	133 ms
7.73280 MHz	191 ms
5.52960 MHz	266 ms
1.84320 MHz	800 ms

A primary concern is the location of the watchdog timer reset command (setting the RWT bit) in the software. The most desirable approach is to have a single location within the main loop of the system software that restarts the watchdog timer periodically. The time required to pass through the main program loop must be less than the timeout interval or the device will reset itself during normal operation. In some systems, however, the program flow is not linear enough to allow the placement of a single watchdog timer reset function. Multiple reset functions should be placed in the code, corresponding to the longest software paths.

Often a system will need to know if a watchdog timer reset has occurred. The WTR bit (PCON.4) will be set whenever this occurs, and software can test for this early in the reset sequence if a system fault has occurred. If so, the system may decide to go into a “safe” mode and alert the user to an error condition.

WATCHDOG RESET EXAMPLE

A short program illustrating the initialization and basic function of the watchdog timer is shown below. It illustrates the Timed Access feature, which prevents the accidental modification of the watchdog control bits. A Timed Access operation is a sequence of steps that must be executed together, in sequence; otherwise, the access fails. The example program shows the timed access being used for restarting the watchdog and enabling its reset. Further details on Timed Access operation may be found in the Secure Microcontroller User’s Guide. The watchdog timer bits that are protected by the Timed Access procedure are the Enable Watchdog Timer Reset (EWT;PCON.2) and Restart Watchdog Timer(RWT;IP.7) bits.

```
;               WD_RST.ASM Program
;
; This program demonstrates the use of the watchdog timer.
; When running, the program counts on port 1 to indicate the device is
; running and periodically resetting the watchdog timer. After counting
; to 16, it stops resetting the watchdog timer, simulating a system fault.
;
; The program begins by checking to see if the WTR bit is set. If so, the
; reset was caused by the watchdog timer, and the program will execute
; the FAULT subroutine. Port 1 is set to F0h to indicate this condition.
; If the WTR bit is not set, the reset was caused by another source and
; execution should continue normally.
;************************************
RWT     EQU     0BFh            ;Reset Watchdog Timer bit
TA      EQU     0C7h            ;Timed Access Register
PCON    EQU     87h             ;Power Control Register
ACC     EQU     0E0h            ;Accumulator
P1      EQU     090h            ;Port 1

        ORG     00h             ;Reset Vector
        SJMP    START

;************************************
        ORG     080h            ;Program starts at 80h in this example.
START:  MOV     A, PCON         ;If reset was caused by watchdog timeout,
        JB      ACC.4, FAULT    ; (WTR bit =1) execute fault subroutine.

;**********************************
;A normal power-on reset has occurred. Start initialization sequence.
        MOV     P1, #00h        ;Clear P1 to signal start of program.

;Watchdog timer initialization sequence
        MOV     TA, #0AAh       ;First restart the Watchdog timer
        MOV     TA, #055h       ; using timed
        SETB    RWT             ; access.

        MOV     TA, #0AAh       ;Next enable the Watchdog timer reset
        MOV     TA, #055h       ; function using timed
        ORL     PCON, #04h      ; access.

;**********************************
;Main program loop. This simulates a program that is operating
; correctly and then goes awry. After the program has counted to 16
; on Port 1 it will skip over the watchdog timer reset function. This
; will simulate a fault and allow the watchdog timer reset to be asserted.
;**********************************
MAIN:   MOV     R1, #0FFh       ;Create a delay loop. This simulates
LOOP1:  MOV     R2, #0FFh       ; a device actually ”doing something.”

LOOP2:  JB      P1.4, SKIP_WD_RST ;Have we been through loop 16 times?

        MOV     TA, #0AAh       ;Watchdog timer reset. In a user application it
        MOV     TA, #055h       ; should be placed at strategic locations
        SETB    RWT             ; where it will be executed periodically.

SKIP_WD_RST:
        DJNZ    R2, LOOP2
        JNZ     R1, LOOP1

        INC     P1              ;Increment counter.
        SJMP    MAIN            ;Go back to main program loop.
;************************************
;Watchdog timeout fault. This subroutine would normally have special
; routines to be executed in the event of a system fault. In this example,
; it disables the watchdog reset and sets Port 1 to F0h to indicate a fault.
; In a real application, this routine could either clear the fault and
; restart the software, or signal a fault and halt further operation.
;************************************
FAULT:  MOV     P1, #0F0h       ;Signal a fault
        MOV     TA, #0AAh       ;Disable watchdog timer reset
        MOV     TA, #55h        ; using timed
        ANL     PCON, #0FBh     ; access.
        SJMP    $               ;Halt further operation.
```

SUMMARY

A number of considerations must go into any design that uses a watchdog as a monitor. Once the timeout period is determined, the system software must be analyzed to determine where to locate the watchdog restart instructions. For an effective design, the number of watchdog restarts should be kept to a minimum, and some consideration should be given to the likelihood of incorrectly executing a restart. As mentioned previously, some system software is too convoluted or data-dependent to ensure that all software flow paths are covered by a watchdog restart. This may dictate that a self-diagnostic software approach might be required. If there is an expected failure mechanism such as a periodic EMI burst or power supply glitch, the watchdog timeout should consider this period.