PowerPC Embedded Processors
Application Note

PowerPC 40x Watch Dog Timer

IBM Microelectronics
Research Triangle Park, NC
ppcsupp@us.ibm.com
Version: 1.0

July 7, 1998

Abstract - The IBM PowerPC 400 family of embedded processors and cores include a Watchdog Timer (WDT) as part of the timer functions. This feature is useful for implementing a system recovery in case of a non-recoverable error. Watchdog timers have been implemented in quite a variety of ways and levels of sophistication. This paper is intended to explain the Watchdog Timer (WDT) function implemented as part of the core logic of the PPC 400 family of processors and cores. Examples of several methods of use are included for illustration.

Watchdog Initialization

An important part of the proper use of the WDT function is proper initialization. The use of the WDT is controlled by two registers, the Timer Control Register (TCR) and the Timer Status Register (TSR). The bit settings in the TCR and TSR following reset are shown in Figure 1. **Those bits not listed are undefined following reset.**

Register	Bits	Core Reset	Chip Reset	System Reset	Comment
TCR	2 : 3	00	00	00	Watchdog Timer reset disabled
TSR	2 : 3	Copy of TCR bits 2:3	Copy of TCR bits 2:3	Copy of TCR bits 2:3	If Reset Caused by Watchdog Timer
		Undefined	Undefined	Undefined	After Power-up
		Unchanged	Unchanged	Unchanged	If Reset not caused by Watchdog Timer

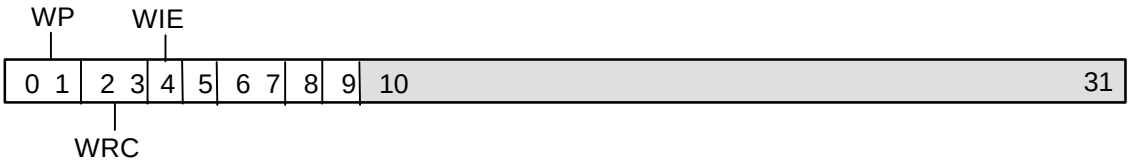
Figure 1. Register Reset Values

The various types of reset are described in detail in the 40x user's manuals. **While other bits in these registers may always power-up or reset to the same value there is no guarantee that they will in future implementations. All bits should be properly initialized.** For example the TCR[WIE] should be reset to 0 before the MSR[CE] critical interrupt bit is set to avoid a spurious interrupt**.** One of the key points of this table is that following a POR (power on reset) TSR[WRS], bits 2:3, are undefined. Without an external indication of POR it is not possible to distinguish whether the value in TSR[WRS] is random from power up or a copy of the TCR[WRC], bits 2:3, from a WDT reset. An example circuit for POR detection is described later in the paper. If the Watchdog function is to be used TSR[WRS] should be reset to 0 and the other TSR bits reset or set depending on the WDT method chosen.

Watchdog Events and Functional Description

There is a lot of flexibility in the use of the WDT **event** of the 40x. The WDT event is a pulse that is generated by decoding bits of the 40x time base registers into 4 different timer taps. The values of the timer taps and times, with at a variety of clock frequencies, is shown in Figure 2 below. WDT events always occur as long as the time base is running. The time base only stops running when programmed to do so during JTAG debugging or when one of the power down modes is used in one of the processors. A full description of the time base operation is given in each of the user's manuals. Sometimes this event

is referred to as a timeout, while for some a WDT timeout refers to an action such as reset that is the ultimate result of not servicing the Watchdog timer. For consistency in this paper a timeout will refer to a WDT event. Note that the time base increments at the core frequency. This is important to remember when running the 401GF or the 403GCX in clock doubled mode, or in ASIC designs with a 40x core running at a frequency different than the input frequency.

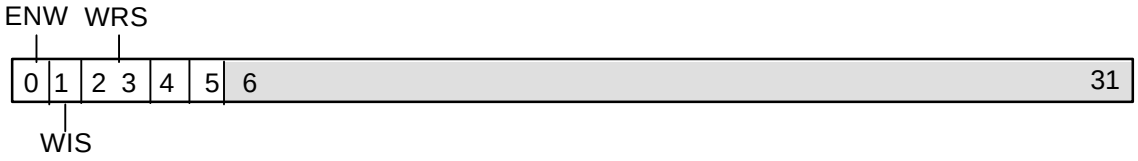


0:1	WP	Watchdog Period	25MHz	33MHz	40 MHz	50 MHz	66 MHz	80MHz
		00 - 2^{17} clocks	5.24 ms	3.972 ms	3.28 ms	2.62 ms	1.99 ms	1.64 ms
		01 - 2^{21} clocks	83.89 ms	63.55 ms	52.43 ms	41.94 ms	31.78 ms	26.21 ms
		10 - 2^{25} clocks	1.342 s	1.017s	838.9 ms	671.1 ms	508.4 ms	419.4 ms
		11 - 2^{29} clocks	21.47 s	16.27s	13.42 s	10.74 s	8.134 s	6.711 s
2:3	WRC	Watchdog Reset Control 00 - No Watchdog reset will occur. 01 - Core reset will be forced by the Watchdog. 10 - Chip reset will be forced by the Watchdog. 11 - System reset will be forced by the Watchdog. TCR[WRC] resets to 00. This field may be set by software, but cannot be cleared by software (except by a software-induced reset).						
4	WIE	Watchdog Interrupt Enable 0 - Disable WDT interrupt. 1 - Enable WDT interrupt.						

Figure 2. Timer Control Register (TCR)

The time between events is controlled by TCR[WP]. The type of reset generated by the Watchdog is controlled by TCR[WRC]. The enable for the Watchdog interrupt is controlled by TCR[WIE]. As noted in the table, the Watchdog period varies over a wide range. For short WDT periods, the time to a potential recovery is small; which is of great importance in critical applications. The cost is in the overhead of servicing the Watchdog at a high rate. Conversely, pushing out the WDT period can minimize the overhead; but the downside is that the system has more time to run amuck before a Watchdog reset.

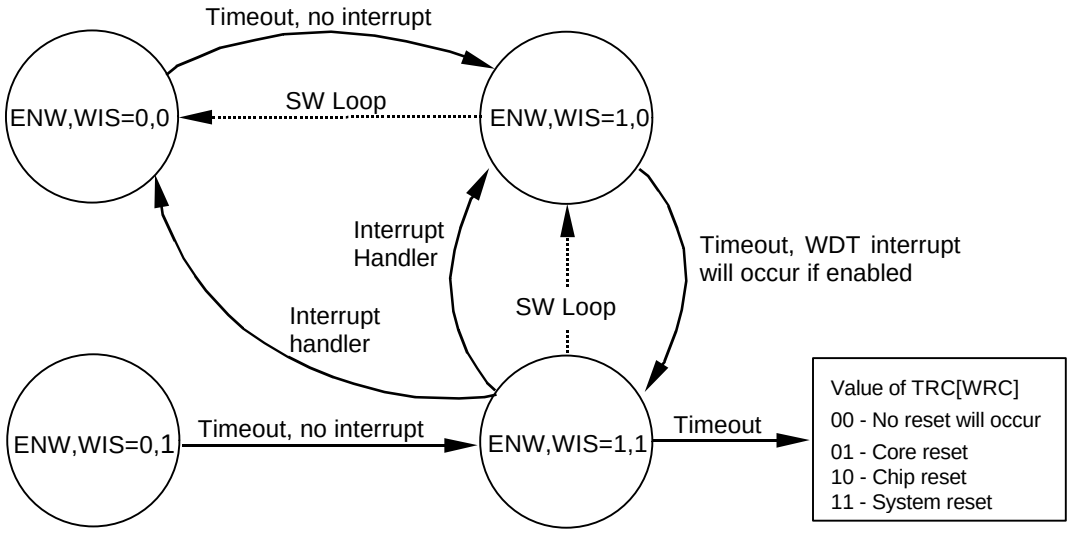
Figure 3 shows the bits of the Timer Status Register (TSR) that are used in combination with the TCR bits to control the WDT function.



0	ENW	Enable Next Watchdog 0 - Action on next Watchdog event TSR[0]=0 -> TSR[0]=1.* 1 - Action on next Watchdog event is governed by TSR[1]. *Note: to stay at a 0 the ENW bit must be reset to 0 by software after each WDT event
1	WIS	Watchdog Interrupt Status 0 - No Watchdog interrupt is pending. 1 - Watchdog interrupt is pending.
2:3	WRS	Watchdog Reset Status 00 - No Watchdog reset has occurred. 01 - Core reset has been forced by the Watchdog. 10 - Chip reset has been forced by the Watchdog. 11 - System reset has been forced by the Watchdog.

Figure 3. Timer Status Register (TSR)

The decision is what effect you want the WDT events to have on your system. A state diagram is included to help illustrate the function.



Note: If a reset occurs the TCR[WRC] bits will be copied to the TSR[WRS] bits and the TCR[WRC] bits will be set to 00.

FIGURE 4. Watchdog Timer State Diagram

The two bits that control how WDT events propagate to a WDT reset are TSR[ENW] and TSR[WIS]. ENW is an enable bit. When a WDT event occurs and the ENW bit is a 0 the WDT event has no effect on the WIS bit, but the ENW bit itself is set to a 1. If ENW is not reset back to a 0, the next WDT event will cause an action based on the state of the WIS bit. When ENW is a 1 and WIS is a 0 the WDT event will set WIS to 1 to indicate a pending WDT exception. The state of TCR[WIE] determines whether an

exception is generated when WIS is a 1. If WIE is a 0 there will be no exception. If WIE is a 1 a critical exception will be generated (assuming critical exceptions are enabled in MSR[CE], bit 14). When ENW and WIS are both 1 and a WDT event occurs, the action taken is based on the value of TCR[WRC]. The 400 family of processors and cores support two classes of exception, critical and non-critical. A full description of the exception classes can be found in the 40x user's manuals.

The typical use of the WDT events is to create a reset if a particular code routine is no longer running. The 40x allows the generated reset to be at the core, chip, or system level. The discussion of each of the resets can be found at the beginning of the chapter on initialization in the 40x user's manuals. Bits 2 and 3 of the TCR select WDT reset; but before these bits are set bits 0 and 1 of the Timer Status Register (TSR) should be reset to 0 by writing 0xC0000000 to the TSR. Note that writing a 1 to a bit in the TSR resets that bit. Once the TCR is set to generate a WDT reset a method is needed to prevent the WDT events from causing a reset during proper system operation.

The user's manuals suggest three methods of preventing ENW and WIS from both being 1 when a WDT event occurs. The presumption is that TCR[WRC] bits have been set to allow a WDT event to cause a reset. To provide some immunity from errant code disabling the WDT reset settings, TCR[WRC] can be set to a 1 but not a 0. Only a reset initialization can reset the bits to a 0.

Methods of Use

Method 1 Uses an exception handler to reset the WIS bit. To use this method TCR[WIE] must be set to a 1 to enable an exception. TSR[ENW] must be a 1 for the WDT event to cause the WIS bit to be set to a 1 and cause an exception. ENW should be set to a 1 by the initialization code, but it will also be set to a 1 at the first WDT event and stay a 1 unless explicitly reset. When a WDT event sets the WIS to a 1 and causes an exception, the exception handler must then reset the WIS bit in the TSR register by writing a 0x40000000. If the WIS bit is not reset to a 0 before the next WDT event a Watchdog reset will occur. A WDT state diagram for method 1 illustrates the concept.

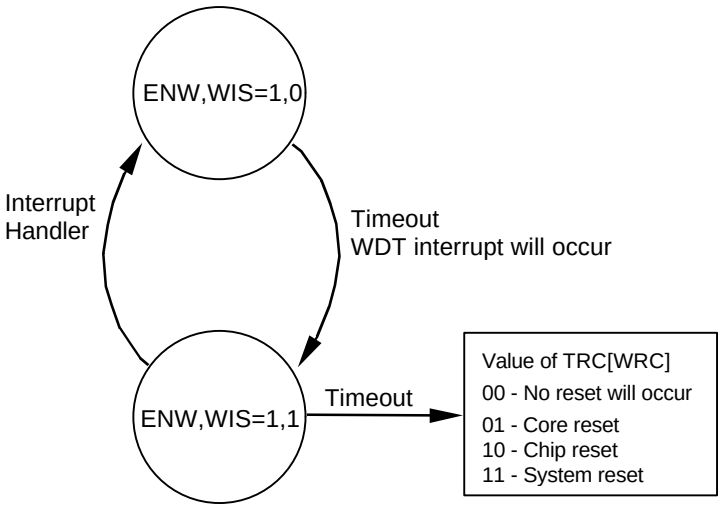


FIGURE 5. Watchdog Timer State Diagram for Method 1

For method 1, the minimum and maximum time from a fault occurring until a WDT reset occurs, based on programmed Watchdog Periods (WP), are 1 to 2 WPs; based on the point during the WP when the fault occurs. This is shown graphically in Figure 6.

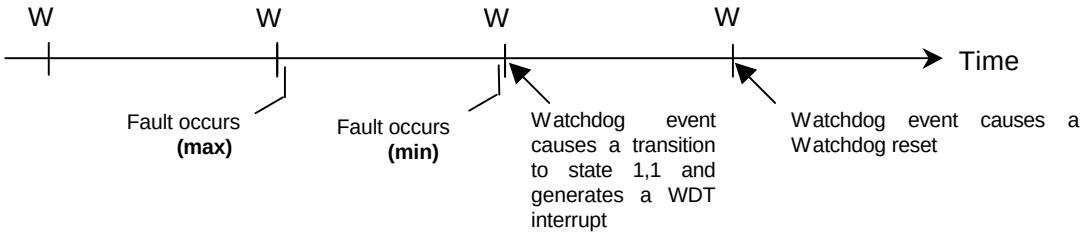


FIGURE 6. Time until a Watchdog Reset for Method 1

Method 1 will only cause a Watchdog reset when the fault is severe enough to prevent the processor from servicing a critical interrupt. It is possible for the software to be hung in a loop; service the critical interrupt and therefore not cause a Watchdog reset.

A fragment of C code is provided as an example of the use of method 1.

```
/*****************************************************************/
/* Method 1 - Use Watchdog Timer exception handler to constantly */
/* turn off Watchdog Interrupt Status to avoid reset             */
/*****************************************************************/
init_code1()  /* initialization code - method 1*/
{
  ...
  wdt_enable_int(2,3,1); /* Set watchdog period TCR[WP]= 2**25     */
                         /* Set watchdog reset TCR[WRC]= Sys Reset */
                         /* Enable watchdog interrupts,TCR[WIE]=1  */

  ...
}

  wdt_interrupt_handler()   /* Watchdog Interrupt        */
                            /* Located at EVPR || 0x1020 */

{
  wdt_clear_intstat();

  /* Could add code here to save information about currently /*
  /* running task. E.g. for later debug */
}
```

An assembly language version of this routine can be found in the appendix

An assembly language version of this routine can be found in the appendix

Method 2 can avoid the overhead of taking an exception. The intent is that there is a background task that runs on a fixed interval that is shorter than the programmed time for the WDT event. The task simply resets the TSR ENW bit to 0. With the ENW bit kept at a 0 the WDT event cannot set the WIS bit. The WDT event will set ENW to a 1 but it will be reset to a 0 by the background task before the next WDT event. If the code hangs the task will not run and on the first WDT event ENW will be set to a 1, on the next WDT event WIS will be set to a 1, and on the third WDT event a Watchdog reset will occur. Method 2 provides an additional chance to avoid a Watchdog reset. When the WDT event sets the WIS bit and TCR[WIE] bit 4 is a 1 a critical exception will occur. The WDT exception handler can reset the ENW and WIS bits and attempt to recover. This presumes that the system is not totally trashed and that a critical exception will be serviced. A WDT state diagram for method 2 illustrates the concept.

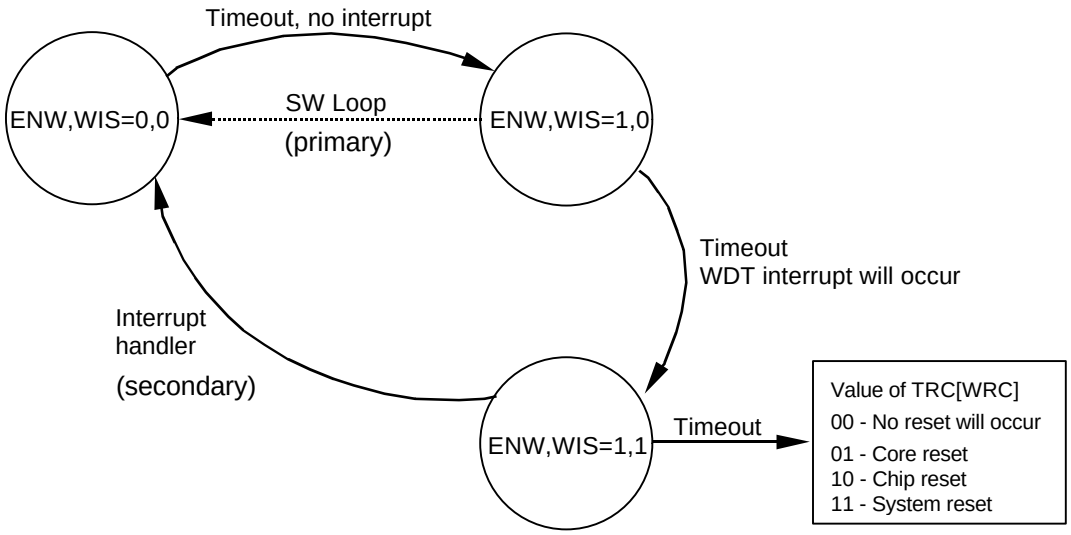


FIGURE 7. Watchdog Timer State Diagram for Method 2

For method 2 the minimum and maximum time from a fault occurring until a WDT reset occurs, based on programmed Watchdog Periods (WP), are approximately 1.5 to 3 WPs based on the point during the WP when the fault occurs. This time is also dependent on the choice of the Watchdog service timer interval. This is shown graphically in Figure 8. (Presumably the Watchdog service timer stops when a fault occurs, so the dashed service timer lines indicate where the service timer would have occurred)

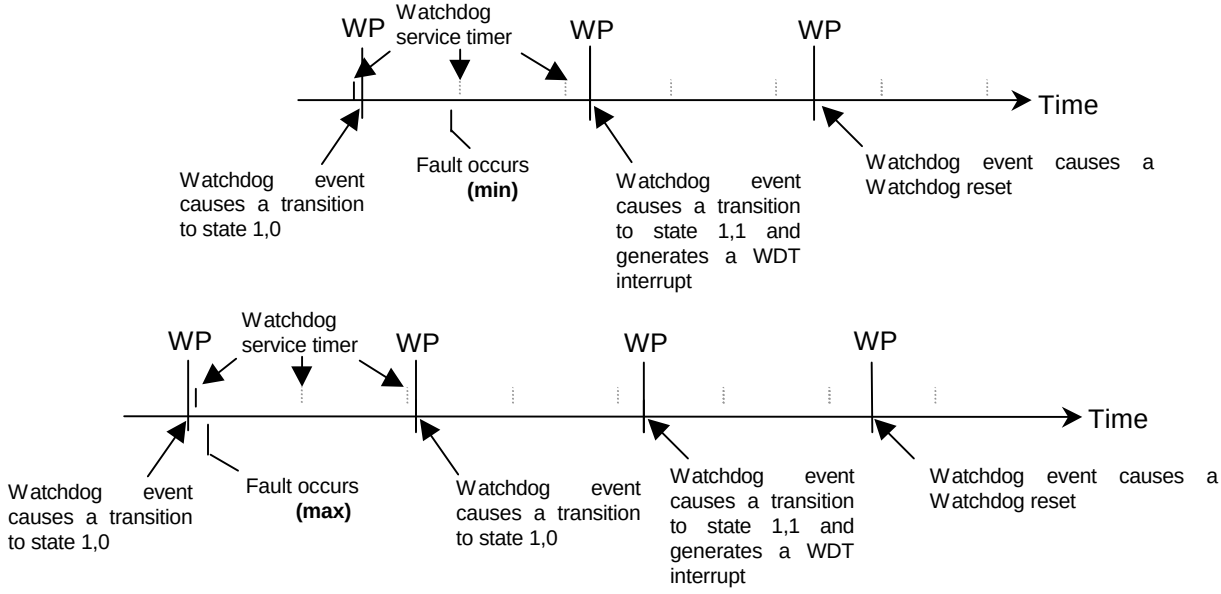


FIGURE 8. Time until a Watchdog Reset for Method 2

The Watchdog service timer must be set with a shorter period than the WP. The question then becomes how much shorter. The Watchdog service routine should have sufficient time to be taken and complete before the WP expires. Figure 8 shows a Watchdog service timer period of just under ½ the WP. The

WDT service routine may run multiple times during the WP; but it does guarantee that at least one Watchdog service routine will run a minimum of ½ the Watchdog period before the next WP.

When the Watchdog service routine fails to restore the state to 0,0 the Watchdog will transition to state 1,1 and cause a WDT interrupt. This means that the software is not operating normally. There are a couple of choices that can be made. If the WDT interrupt service routine simply resets the ENW and WIS bits to either state 0,0 or 0,1, and the original fault is not cleared, the WDT will loop back to the 1,1 state and generate another WDT interrupt. One strategy is to recognize that by the time you are in 1,1 state a fault has occurred and use the time before the next WDT event (and WDT reset) to store as much program state information as possible to help debug. Another strategy is to try to do a soft restart and thereby clear the fault before resetting the WDT state. To avoid getting into a perpetual loop the WDT interrupt service routine could set a location in memory to indicate whether this was the first time through the routine. If it was not the first time through, then the routine could store as much program state information as possible to help debug, and allow the WDT reset to occur.

A fragment of C code is provided as an example of the use of method 2.

```
/*************************************************************************/
/* Method 2 - Use background task on predefined fixed interval to turn off */
/* Enable Next Watchdog, TSR[ENW].  This can avoid taking an exception     */
/*************************************************************************/
init_code2()  /* initialization code for method 2*/
{
  ...
  wdt_enable_int(0,1,1);  /* Set watchdog period TCR[WP]=2**17      */
                          /* Set watchdog reset TCR[WRC]= Core Reset*/
                          /* Enable watchdog interrupts TCR[WIE]=1  */

  ...
}
```

An assembly language version of this routine can be found in the appendix

```
/* The following background task MUST be executed in a fixed */
/*  interval shorter than the wdt period (2**17 for this example) */
wdt_background_task()    /* Watchdog background task */
{
  ...
  wdt_disable_next_wdevent(); /* Disable next wd event TSR[ENW]=(0) */

  ...
}
```

An assembly language version of this routine can be found in the appendix

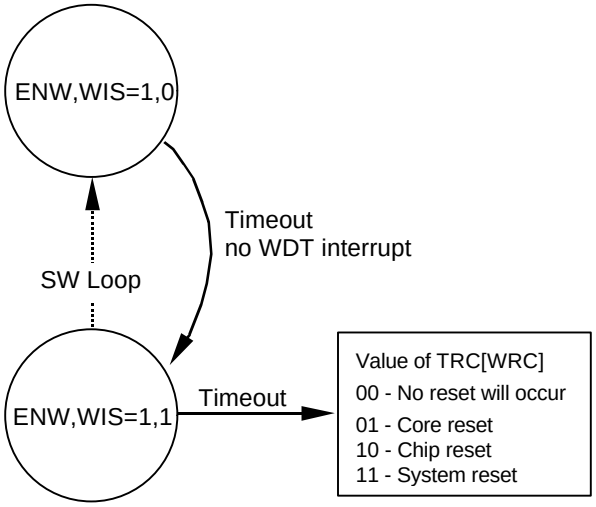
```
wdt_interrupt_handler()    /* Watchdog Interrupt */
                                        /* Located at EVPR || 0x1020 */
{
  /* This will be called if background task did not get to run */

  /* If attempting to recover to normal processing, we could */
  /* issue wdt_disable_next_wdevent() and                    */
  /* wdt_clear_intstat() to clear out current wdt state      */

  /* Could add code here to save information about currently  */
```

```
  /* running task.  E.g. for later debug                    */
}
```

Method 3 avoids the overhead of taking exceptions. TCR[WIE] bit 4 is reset to 0 to disable the watchdog exception. TSR[ENW] must be a 1 for the WDT event to cause the WIS bit to be set to a 1 and cause a state transition. ENW should be set to a 1 by the initialization code, but it will also be set to a 1 at the first WDT event and stay a 1 unless explicitly reset. The intent is that there is a background task that runs on a fixed interval that is shorter than the programmed time for the WDT event. The task simply resets the WIS bit to 0. With WIS kept at 0 the WDT event cannot cause a reset. The WDT event will set the WIS to a 1 but it will be reset to a 0 by the background task before the next WDT event. If the code hangs the task will not run and on the first WDT event WIS will be set to a 1, and on the next WDT event a Watchdog reset will occur. A WDT state diagram for method 3 illustrates the concept.



Note: if a reset occurs the WRC bits will be copied to the WRS bits and the WRC bits will be set to 00.

FIGURE 9. Watchdog Timer State Diagram for Method 3

For method 3 the minimum and maximum time from a fault occurring until a WDT reset occurs, based on programmed Watchdog Periods (WP), are ½ to 1 WPs based on the point during the WP when the fault occurs. . This time is also dependent on the choice of the Watchdog service timer interval. This is shown graphically in Figure 10.

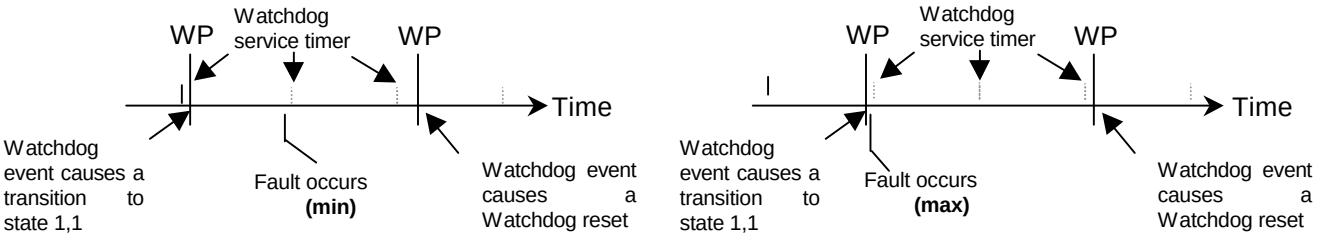


FIGURE 10. Time until a Watchdog Reset for Method 3

The Watchdog service timer must be set with a shorter period than the WP. The question then becomes how much shorter. The Watchdog service routine should have sufficient time to be taken and complete before the WP expires. The example shows a Watchdog service timer period of just under ½ the WP. The WDT service routine may run multiple times during the WP; but it does guarantee that at least one Watchdog service routine will run a minimum of ½ the Watchdog period before the next WP. Method 3 has the lowest latency between a fault and a Watchdog reset. Unfortunately since no interrupts are generated it also provides the least opportunity to do any soft error recovery.

A fragment of C code is provided as an example of the use of method 3.

```
/*****************************************************************/
/* Method 3- Use background task on predefined fixed interval to */
/* turn off Watchdog Interrupt Status to avoid wdt exception.    */
/*****************************************************************/
init_code3()  /* initialization code for method 3*/
{
  ...
  wdt_enable_int(1,2,0);  /* Set watchdog period TCR[WP]= 2**21 */
                          /* Set watchdog reset TCR[WRC]= Chip reset */
                          /* Disable watchdog interrupts, TCR[WIE]=0 */
  ...
}

/* The following background task MUST be executed in a fixed interval */
/* shorter than the wdt period (2**21 for this example)*/
wdt_background_task()    /* Watchdog background task */
{
  ...
  wdt_clear_intstat();   /* Clear Watchdog Int. Status - TSR[WIS] */
  ...
}
```

An assembly language version of this routine can be found in the appendix

An assembly language version of this routine can be found in the appendix

The state diagram in Figure 4 shows the all the possible states but only the transitions that support the three methods described above. A variation on Method 3 is available by using state ENW,WIS=0,1. A software loop could reset ENW from state 1,1 instead of resetting WIS. The distinction is that state 0,1 is only stable if the WDT interrupt is disabled, WIE is 0. In state 0,1 an exception would be taken immediately if WIE were set to 1. The inability to have an interrupt generated on a timeout renders this potential method either less useful than the other three methods or simply redundant.

Power On Reset

Occasionally the question arises, is there a way to distinguish between a Watchdog reset and a POR (Power On Reset). Since the values in the TSR are indeterminate at power up, there is no reliable way to distinguish a POR from a WDT reset. A common method to distinguish between the two resets is to have a small circuit on the board that is cleared during POR and set by the processor.

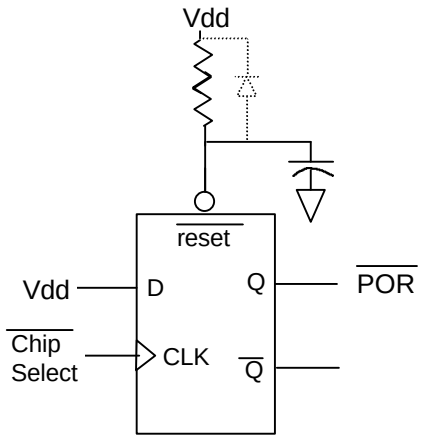


Figure 11 Power On Reset, POR, Detection Circuit

During power up the flip-flop is cleared by the capacitor holding the reset input low. The signal POR is low indicating a power on reset. This bit can be tested during initialization to see if the reset was caused by a power on reset or a system reset. Following the check an access to the flip-flop as a memory mapped I/O will set the POR signal to a 1. This circuit can be set by software but not cleared. To detect power glitches that may trigger the power on reset circuit, that drives the board reset, a diode can be added to quickly discharge the capacitor. The circuit is only for illustration since it is unlikely that a Chip Select would be dedicated. Some other points to remember are that the RC must be selected such that the reset input is still below 0.8 volts until Vdd is stable. If it takes 10ms for Vdd to become stable it would take a 100K ohm resistor and a 0.47uF capacitor to hold reset low for at least 13ms (depending on the slew rate of the supply), and it would be at least 44ms before the reset input reached 2v.

Appendix

Assembly Code Routines

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Function:   wdt_enable_int()
! Purpose:    Enables 4xx Watchdog, sets up passed in values for TCR[WP],
!             TCR[WRC], and TCR[WIE].
! Parameters: 1. r3 - Input Watchdog Period - TCR[WP] (in core clocks)
!                  0 = 2**17 clocks, 1 = 2**21 clocks
!                  2 = 2**25 clocks, 3 = 2**29 clocks
!             2. r4 - Watchdog reset control - TCR[WRC]
!                  0 = No reset
!                  1 = PPC Core reset only
!                  2 = PPC Chip reset
!                  3 = System reset
!             3. r5 - Interrupt Enable (1=enable)
! Returns:   Current value of TCR (in R3)
! Notes:      1. TSR[ENW] (Watchdog enable) is self-enabling.  If it has been
!                cleared by software, it will reenable on the next watchdog.
!                So no need to set it  (you cannot set it)
!             2. Use read/modify write of TCR to avoid changing other PPC !
!                timers
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.text
.align  2
.extern wdt_enable_int
.globl  wdt_enable_int

wdt_enable_int:
slwi   r3, r3, 30              ! move TCR[WP] to bits 0-1 in r3
slwi   r4, r4, 28              ! move TCR[WRC] to bits 2-3 in r4
slwi   r5, r5, 27              ! move TCR[WIE] to bits 4 in r5
or     r6, r3, r4              ! OR Watchdog period and reset controls into r6
mfspr  r3, tcr                 ! Read TCR into R3
or     r3, r6, r5              ! OR Watchdog Interrupt Enable
mtspr  tcr, r3                 ! Write new value of TCR
blr

.type wdt_enable_int,@function
.size wdt_enable_int,.-wdt_enable_int
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Function:   wdt_clear_intstat()
! Purpose:    Clears 4xx Watchdog interrupt Status, TSR[WIS]=0
! Parameters: None
! Returns:    None
! Notes:      The Timer Status Register is a read/clear register. To clear a
!             bit, you write a 0x1 to it.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.extern wdt_clear_intstat
.globl  wdt_clear_intstat

wdt_clear_intstat:
addis    r3, r0, 0x4000  ! Clear TSR[WIS] by setting the bit
mtspr    tsr, r3
blr

.type wdt_clear_intstat,@function
.size wdt_clear_intstat,.-wdt_clear_intstat


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! Function:   wdt_disable_next_wdevent()
! Purpose:  Disable the next Watchdog so that the Interrupt Status will not
!           get set, TSR[ENW]=0
! Parameters: None
! Returns:    None
! Notes:      1. The watchdog is auto-enabling, so disabling the next watchdog
!                event only works for 1 event.
!             2. The Timer Status Register is a read/clear register.  To clear
!                a bit, you write a 0x1 to it.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.extern wdt_disable_next_wdevent
.globl  wdt_disable_next_wdevent

wdt_disable_next_wdevent:
addis    r3, r0, 0x8000  ! Clear TSR[ENW] by setting the bit
mtspr    tsr, r3
blr

.type wdt_disable_next_wdevent,@function
.size wdt_disable_next_wdevent,.-wdt_disable_next_wdevent
```