Freescale Semiconductor
Application Note

AN2817
Rev. 1.1, 06/2005

MPC5500 Watchdog Timer

Configuration and Operation

by: Bill Terry
TECD MPC5500 Applications

Members of the MPC5500 family of devices that utilize the e200z6 core provide a watchdog timer function that is different from many previous Freescale microcontrollers. This application note describes the basic function of the watchdog and provides example code for typical configurations.

It is recommended that the user obtain the *"e200z6 PowerPC™ Core Reference Manual"*, Rev. 0, for complete details of the registers and timer features discussed in this document.

The following three methods for watchdog implementation are discussed.

- Periodic Service Routine
- Periodic Service Routine with Interrupt Handler
- Interrupt Driven Service Routine

Table of Contents

freescale™ semiconductor

1 Overview

A watchdog timer is a common feature on many MCUs. The purpose of the watchdog is to allow the system or application a means to recover in the case of errant code execution or other events that may cause uncontrolled operation of the MCU.

1.1 Basic Watchdog

Typically, a watchdog is a continuously running timer that may be configured by the application so that it expires or rolls over at a predetermined time interval. This interval is usually determined by the system clock frequency (as in the case of the MPC5500 devices) and a watchdog timeout value that is set by the application.

The application must perform some specific action before the timeout occurs, which causes a reset of the watchdog timer, and a restart of the timeout count. The required action may be writing a specific location in memory, setting or clearing a bit, or some other method. The application must service the watchdog periodically at intervals short enough to prevent the timeout.

If the watchdog interval expires before the watchdog is serviced by the application, the system or application is assumed to be in an unknown state and the hardware may generate an interrupt or reset the MCU. The hardware response to a watchdog timeout can vary, depending on the MCU.

The watchdog period may be set for short times in the case of critical, time-sensitive applications, at the expense of increased overhead to service the watchdog. Conversely, the watchdog may be set for longer periods, requiring less intervention from the application, at the expense of slower detection of potential software or system problems.

1.2 MPC5500 Watchdog

The design philosophy of the watchdog timer implemented in the e200z6 core differs somewhat from typical watchdog operation. The MPC5500 devices recognize both a first and second occurrence of a watchdog timeout event. A watchdog timeout event may optionally be configured to generate an interrupt or an MCU reset. Essentially, this two-event, programmable mechanism provides the application an opportunity to correct a problem before resorting to a full reset of the device.

The key registers associated with the control of the watch dog timer are the timer control register (TCR) and the timer status register (TSR). The machine state register (MSR) and time base (TB) are also used. Each of these registers is described in detail in the e200z6 Reference Manual.

The state transition of the watchdog timer is dependent on the enable next watchdog (TSR[ENW]) and watchdog interrupt status (TSR[WIS]) bits. The effect of these bits on the state transitions is shown in Table 1.

Table 1. Watchdog Timer Control

Current Bit States		Action When Timeout Occurs
TSR[ENW]	TSR[WIS]	
0	0	TSR[ENW] is set to 1.
0	1	TSR[ENW] is set to 1.
1	0	TSR[ENW] remains set to 1, TSR[WIS] is set to 1. If TCR[WIE] and MSR[CE] are enabled, generate an interrupt.
1	1	Do action as set by TCR[WRC], copy contents of TCR[WRC] to TSR[WRS].

2 Example Implementation Methods

Based on the watchdog state operation described in Table 1, there are three ways in which the watchdog may be implemented/serviced.

- Periodic service routine without interrupt handler
- Periodic service routine with interrupt handler
- Interrupt driven watchdog service routine

These are discussed in the following sections.

2.1 Method 1 - Periodic Service Routine without Interrupt Handler

The first method is very similar to the basic watchdog operation described in Section 1.1 and will likely be the method used by most customers. TSR[ENW] is set so that a watchdog timeout event causes TSR[WIS] to be set, but TCR[WIE] is cleared, thus preventing an interrupt with associated overhead. The application provides a periodic service routine that clears the TSR[WIS] bit at a period less than the programmed watchdog timeout, thus preventing a reset. Note that the TSR[ENW] bit can not be written directly by the application. An initial timeout must be allowed to set this bit before the service routine begins running. A flow diagram of this method is shown in Figure 1.

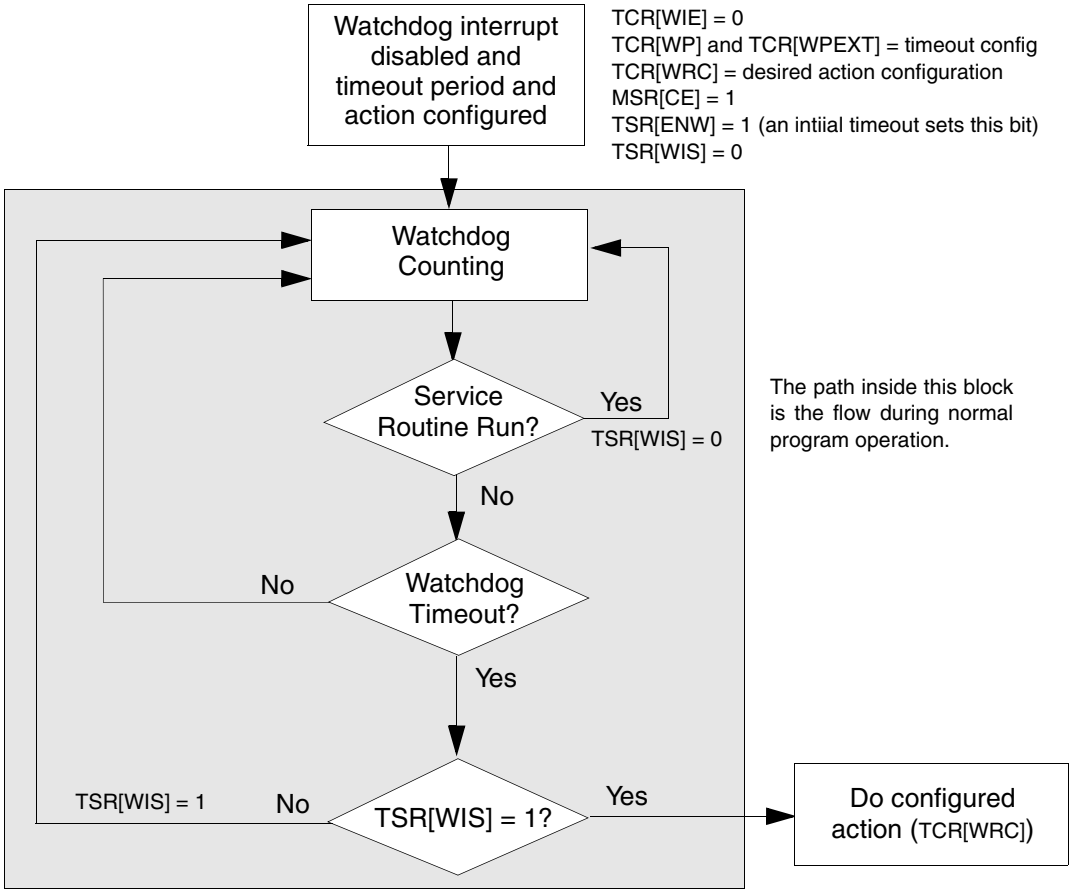

Figure 1. Flow Diagram - Method 1

2.1.1 Configuration - Method 1

Table 2. Watchdog Configuration Sequence (no Interrupt)

Step	Description	Bit Field	Pseudo Code
Configure watchdog	Set for reset on 2nd timeout, Set timeout for 3.3554432 seconds (at 80MHz system clock)	TCR[WRC] TCR[WP] TCR[WPEXT]	SPR TCR[WRC] = 0b01 SPR TCR[WP] = 0b00 SPR TCR[WPEXT] = 0b1001
	Enable time base	HID0[TBEN]	SPR HID0[TBEN] = 1

2.1.2 Periodic Service Routine - Method 1

Method 1 requires an independent periodic service routine that prevents the watchdog from resetting the part. The function of the service routine is outlined in Table 3.

Table 3. Watchdog Service Routine (Method 3)

Step	Description	Bit Field	Pseudo Code
Service watchdog	Prevent a watchdog timeout reset by clearing the watchdog interrupt status bit.	TSR[WIS]	SPR TSR[WIS] = 1

2.1.3 Timing Considerations - Method 1

This method of operation does not incur the overhead of interrupt handling, however it requires a periodic service routine. The TSR[ENW] bit is always set and the service routine must clear the TSR[WIS] often enough to prevent a reset from occurring.

The TSR[WIS] bit must be cleared between watchdog timeouts as shown in Figure 2. This implies that to keep the timeout from generating a reset, the service routine must run at least once each watchdog period. Therefore the service routine should execute at a period less than the watchdog period. As with the service routine discussed in method 2 (See Section 2.2), it is advisable to make the service routine period enough shorter than the watchdog period to insure it runs at least once at some time not near the beginning or end of the watchdog period.

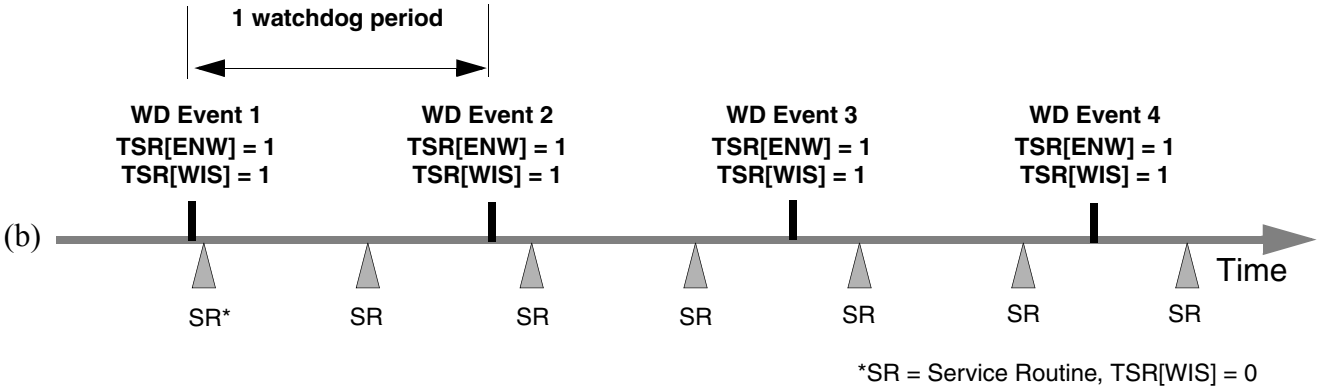

Figure 2. Watchdog Service Routine TIming - Method 1

The effective watchdog reset timeout is determined by the period of the service routine and at what time within a watchdog timeout the problem occurs. See Figure 3 for an illustration.

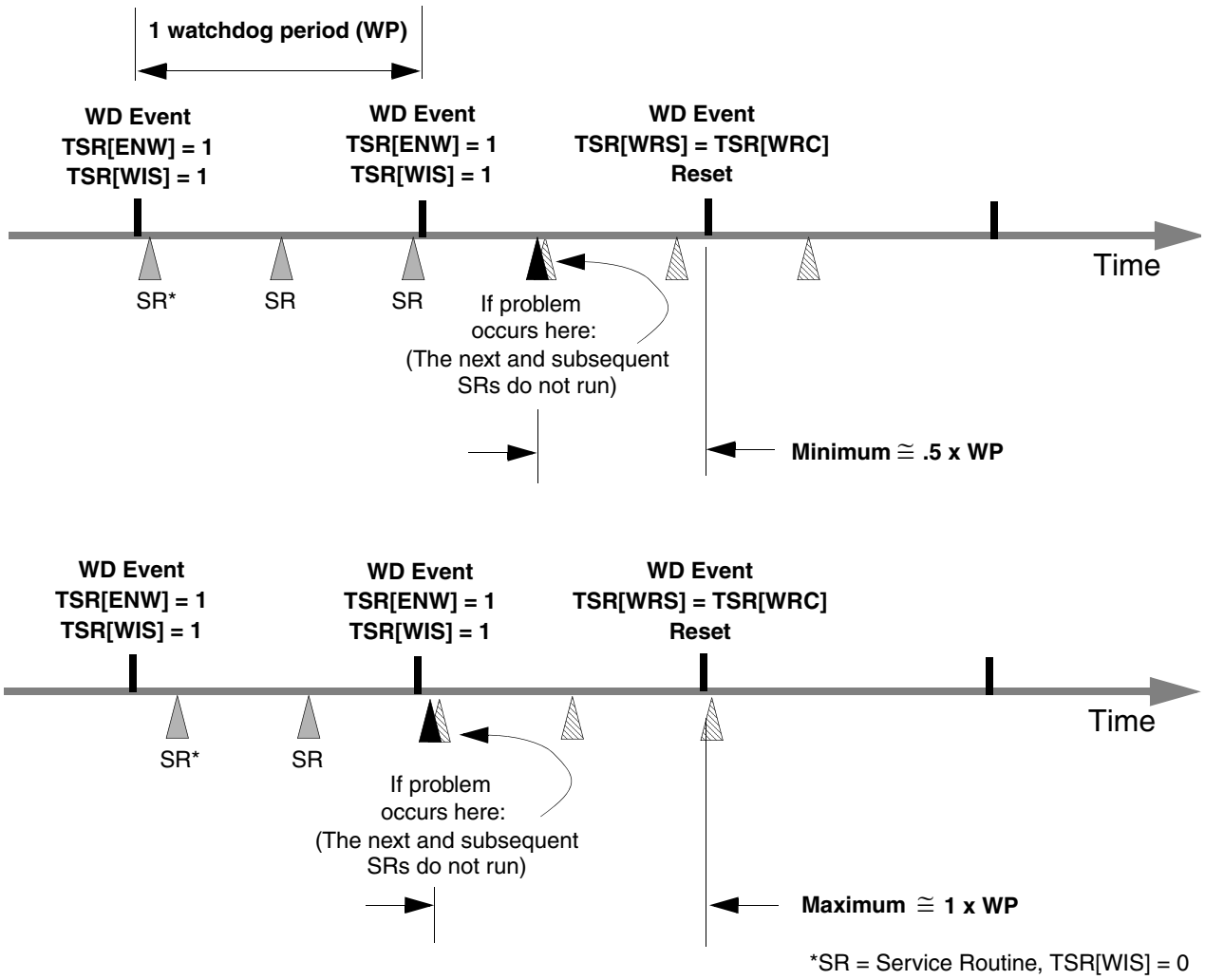

Figure 3. Effective Watchdog Reset Timeout - Method 1

2.2 Method 2 - Periodic Service Routine with Interrupt Handler

The second method uses a periodic service routine. This routine runs periodically to repeatedly clear the TSR[ENW] bit so that a first timeout event is avoided and no timeout exception occurs. Depending on the count of the watchdog timer when the TSR[ENW] bit is cleared, the software has between one and two full timeout periods before an exception can occur and be indicated by TSR[WIS]. If this happens before the application clears TSR[ENW] again, an interrupt is generated and the interrupt handler runs.

When the handler runs, it is assumed that the software is not operating normally and the handler may either try to store relevant debug information before the impending reset on the next timeout, or clear both TSR[ENW] and TSR[WIS] in an attempt to avoid another watchdog interrupt.

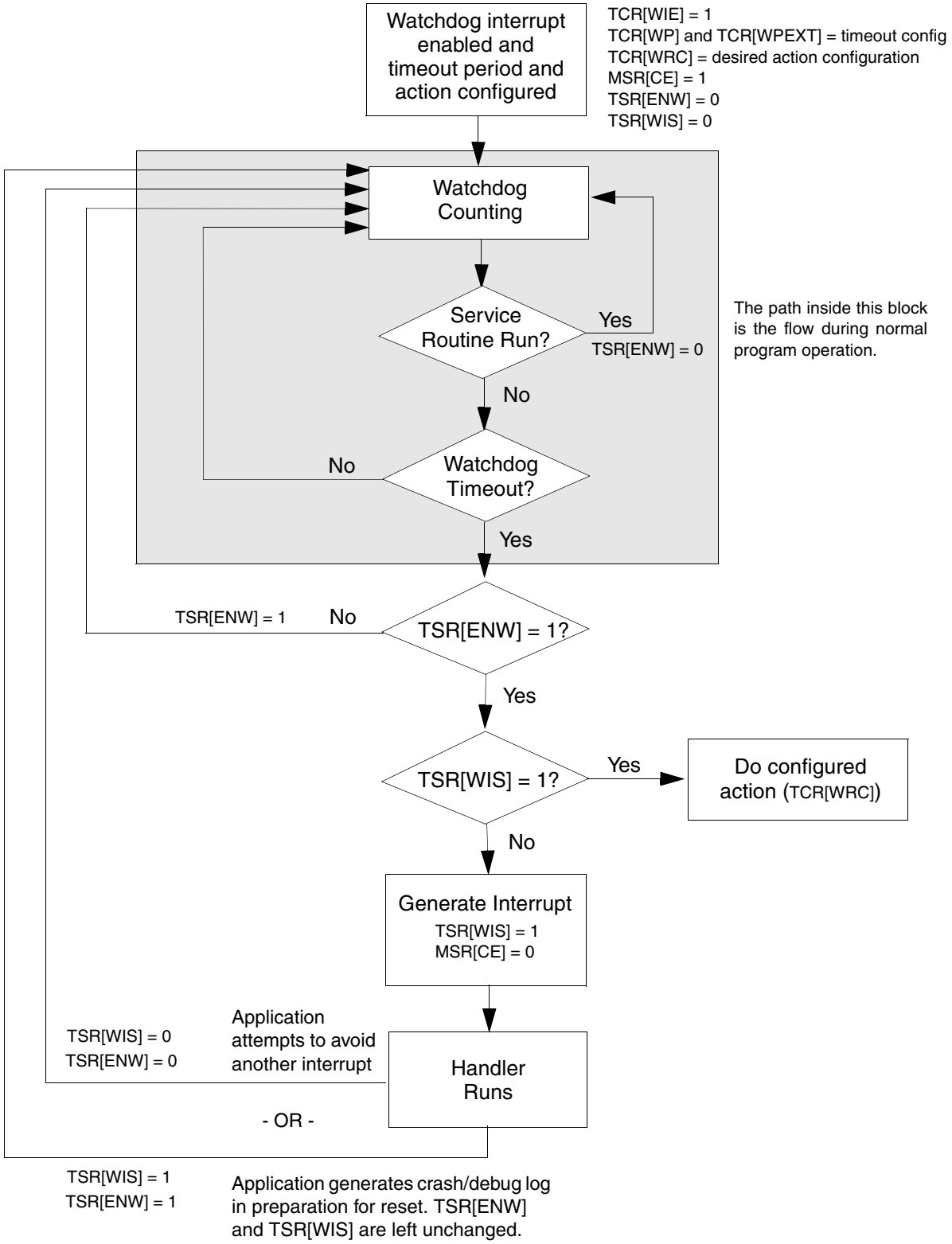

Figure 4. Flow Diagram - Method 2

2.2.1 Configuration - Method 2

Table 4 lists the actions required to configure the watchdog for this method.

Table 4. Watchdog Configuration Sequence (with Interrupt)

Step	Description	Bit Field	Pseudo Code
Configure watchdog	Enable watchdog interrupt Set for reset on 3nd timeout, Set timeout for 4.194304 seconds (at 80MHz system clock)	TCR[WIE] TCR[WRC] TCR[WP] TCR[WPEXT]	SPR TCR[WIE] = 0b1 SPR TCR[WRC] = 0b01 SPR TCR[WP] = 0b00 SPR TCR[WPEXT] = 0b1001
	Enable critical interrupts	MSR[CE]	MSR[CE] = 1
	Enable time base	HID0[TBEN]	SPR HID0[TBEN] = 1

2.2.2 Interrupt Vector Setup - Method 2

The e200z6 core provides a set of registers that allow the interrupt vector addresses to be programmed. The watchdog timer uses interrupt vector 12. The address of the handler for this interrupt is set by writing the upper 16 bits of the handler address to the interrupt vector prefix register (IVPR) and the lower 16 bits of the handler address to interrupt vector offset register 12 (IVOR12). When interrupted, the MCU determines the interrupt source and concatenates the IVPR and relevant IVOR to create the actual handler address. The e200z6 Reference Manual contains a complete description of the IVPR and IVOR register functions. Table 5 lists the actions required to setup the watchdog interrupt vector.

Table 5. Initializing the Watchdog Interrupt Vector

Step	Description	Bit Field	Pseudo Code
Initialize watchdog vector	Load IVPR with upper 16-bits of WD handler address. Load IVOR12 with lower 16-bits of WD handler address	IVPR IVOR12	SPR IVPR = handler_ address[0:15] SPR IVOR12 = handler_address[16:31]

2.2.3 Interrupt Handler - Method 2

This section provides information for implementing the required watchdog interrupt handler.

Context Switching and Alignment

The critical save and restore registers 0 and 1 (CSRR0 and CSRR1) must be preserved across the interrupt. The MPC5500 devices automatically save the MSR and program resume address information in these two registers when a critical interrupt occurs. This particular example also saves one general purpose register on the stack so that it may be used for temporary storage in the handler.

Table 6. Interrupt Handler (Method 2)

Step	Description	Bit Field	Pseudo Code
Save context	Allocate 16 bytes of stack space Save a working register. Save CSSR*n* registers	stack pointer (sp) r6 CSSR0 CSSR1	sp = sp - 16 addr (sp + 4) = r6 addr (sp + 8) = SPR CSRR0 addr (sp + 12) = SPR CSRR1
Option 1			
Create a crash record	Recognize that the application is unstable and save state information to assist in debug. Next timeout will cause a reset.	User defined	User defined
Option 2			
Service watchdog timeout and return	Clear the watchdog interrupt status and enable next watchdog bits, by writing them with 1. Assume the system will correct the problem and avoid further timeouts.	TSR[WIS] TSR[ENW]	SPR TSR[WIS] = 1 SPR TSR[ENW] = 1
Restore context	Restore CSSRn registers Restore r6 Restore stack pointer	CSSR1 CSSR0 r6 sp	SPR CSSR1 = addr (sp + 12) SPR CSSR0 = addr (sp + 8) r6 = addr (sp + 4) sp = sp + 16
Return	Return from critical interrupt	–	rfci (restores machine state and reenables critical interrupts)

2.2.4 Periodic Service Routine - Method 2

Method 2 requires an independent periodic service routine that prevents the watchdog from resetting the part. The function of the service routine is outlined in Table 7.

Table 7. Watchdog Service Routine (Method 2)

Step	Description	Bit Field	Pseudo Code
Service watchdog	Prevent an initial watchdog timeout by clearing the next watchdog event bit.	TSR[ENW]	SPR TSR[ENW] = 1

2.2.5 Timing Considerations - Method 2

In this method, a periodic watchdog service routine resets the TSR[ENW] bit often enough to prevent an interrupt from being generated. The sequence of updates to the TSR[ENW] and TSR[WIS] bits must occur as shown in Figure 5. This implies that to keep the TSR[WIS] bit from being set and an interrupt being generated, the service routine must run at least once each watchdog period. Therefore the service routine should execute at a period less than the watchdog period. It is advisable to make the service routine period enough shorter than the watchdog period to insure it runs at least once at some time not near the beginning or end of the watchdog period, (as is shown in Figure 5a, at WD Event 3 and WD Event 4).

Figure 5b illustrates a better method of service routine timing, with a service routine period of about 1/2 of the watchdog period.

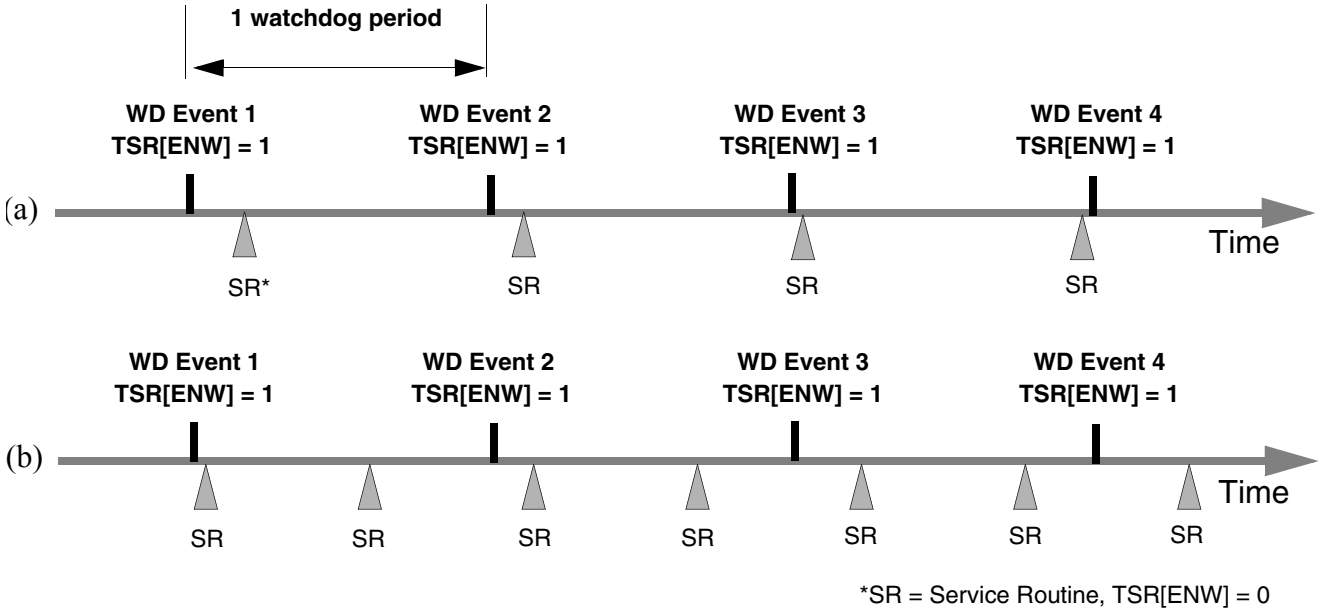

Figure 5. Watchdog Service Routine Timing - Method 2

When method 2 is implemented, the minimum and maximum effective timeouts before a reset can occur are determined by at what point during a watchdog timeout the problem occurs, and by how often the service routine runs. Using a service routine of approximately 1/2 of a watchdog period gives a minimum effective reset time of about 1.5 watchdog periods, and a maximum effective reset time of about 3 watchdog periods. Figure 6 provides an illustration of the timing for both cases.

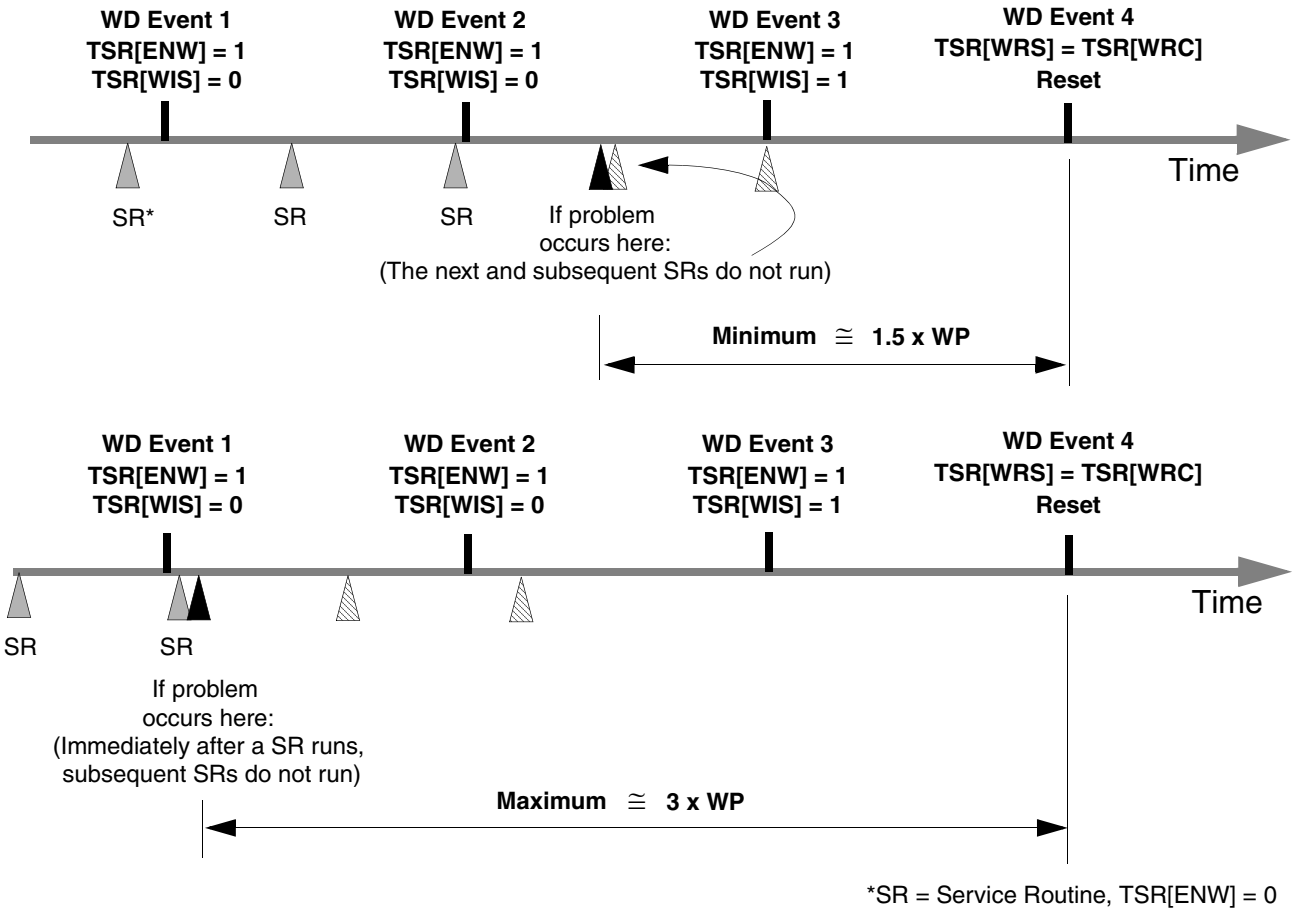

Figure 6. Effective Watchdog Reset Timeout - Method 2

2.3 Method 3 - Interrupt Driven Watchdog Service Routine

This section describes a method for an interrupt driven watchdog implementation/service. Detailed information about the registers and bits that are referenced in this section is found in Section 3.

In this method, the TCR[WIE] bit (watchdog interrupt enable), the MSR[CE] bit (critical interrupt enable), and the TSR[ENW] bit (enable next watchdog) are all set, so that every watchdog event sets the TSR[WIS] and consequently generates an interrupt. Note that the TSR[ENW] bit can not be written directly by the application. An initial timeout must be allowed to set this bit. The application services each watchdog timer interrupt when pending, and never attempts to prevent its occurrence. The handler clears the TSR[WIS] status bit, thus clearing the interrupt, but the TSR[ENW] remains set so that each subsequent watchdog event will trigger a new interrupt.

The state flow of this method is shown in Figure 7.

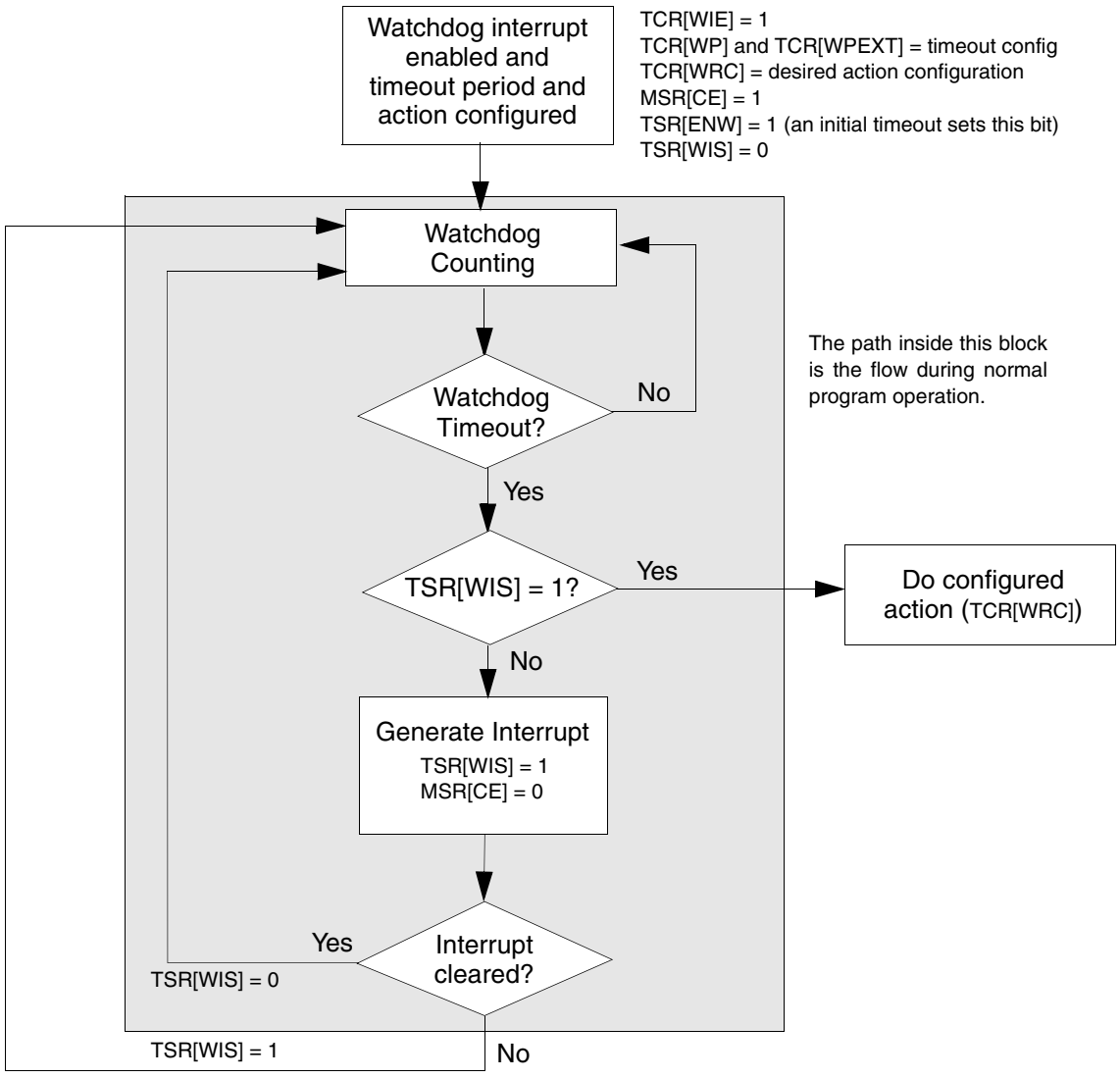

Figure 7. Flow Diagram - Method 3

2.3.1 Configuration - Method 3

Table 8 lists the actions required to configure the watchdog for this method.

Table 8. Watchdog Configuration Sequence (with Interrupt)

Step	Description	Bit Field	Pseudo Code
Configure watchdog	Enable watchdog interrupt Set for reset on 2nd timeout, Set timeout for 3.3554432 seconds (at 80MHz system clock)	TCR[WIE] TCR[WRC] TCR[WP] TCR[WPEXT]	SPR TCR[WIE] = 0b1 SPR TCR[WRC] = 0b01 SPR TCR[WP] = 0b00 SPR TCR[WPEXT] = 0b1001
	Enable critical interrupts	MSR[CE]	MSR[CE] = 1
	Enable time base	HID0[TBEN]	SPR HID0[TBEN] = 1

2.3.2 Interrupt Vector Setup - Method 3

The interrupt vector for this method is setup in the same manner as for Method 2, described in Section 2.2.2.

2.3.3 Interrupt Handler - Method 3

This section provides information for implementing the required watchdog interrupt handler.

Context Switching

The context switching and alignment requirements for this method are the same as described in Section 2.2.3.

Alignment

The least significant 4 bits of the interrupt handler address must always be 0, that is, the handler starting address must be on a quad-word (16-byte) boundary. See the e200z6 Reference Manual for more information on this requirement. Various methods may be used to insure this alignment, depending on the toolset being used.

Table 9 lists the actions that must be performed by the interrupt handler.

Table 9. Interrupt Handler (Method 1)

Step	Description	Bit Field	Pseudo Code
Save context	Allocate 16 bytes of stack space Save working register. Save CSSR*n* registers	stack pointer (sp) r6 CSSR0 CSSR1	sp = sp - 16 addr (sp + 4) = r6 addr (sp + 8) = SPR CSRR0 addr (sp + 12) = SPR CSRR1
Service watchdog timeout	Clear the watchdog interrupt status bit, by writing it with 1.	TSR[WIS]	SPR TSR[WIS] = 1
Restore context	Restore CSSRn registers	CSSR1 CSSR0 r6 sp	SPR CSSR1 = addr (sp + 12) SPR CSSR0 = addr (sp + 8) r6 = addr (sp + 4) sp = sp + 16
Return	Return from critical interrupt	–	rfci (restores machine state and reenables critical interrupts)

2.3.4 Timing Considerations - Method 3

In this mode of operation, the watchdog generates an interrupt periodically, and the interrupt handler resets the TSR[WIS] bit to avoid a reset (or whatever response is configured by the TCR[WRC] bits). See Figure 8 for an illustration of the timing sequence.

The first watchdog timeout sets the TSR[WIS] bit and generates the interrupt that clears the TSR[WIS] bit. If the interrupt does not run, a second timeout with TSR[WIS] set will trigger a reset or the response determined by TCR[WRC]. Therefore, the maximum effective timeout before a reset is approximately 2 times the configured timeout value, and the minimum effective timeout before a reset is approximately 1 times the configured timeout value. See Figure 9.

Since there is no periodic service routine, the only software timing constraint is that the interrupt handler complete execution before the end of a watchdog period. Unless the watchdog is configured for an extremely short timeout period, this is should not be a problem.

NOTE

It is possible that the watchdog interrupt could occur during a critical external input interrupt, causing the service routine to miss its deadline due to the occurrence of an event with a lower priority. This possibility should be taken into consideration when determining the watchdog timeout period.

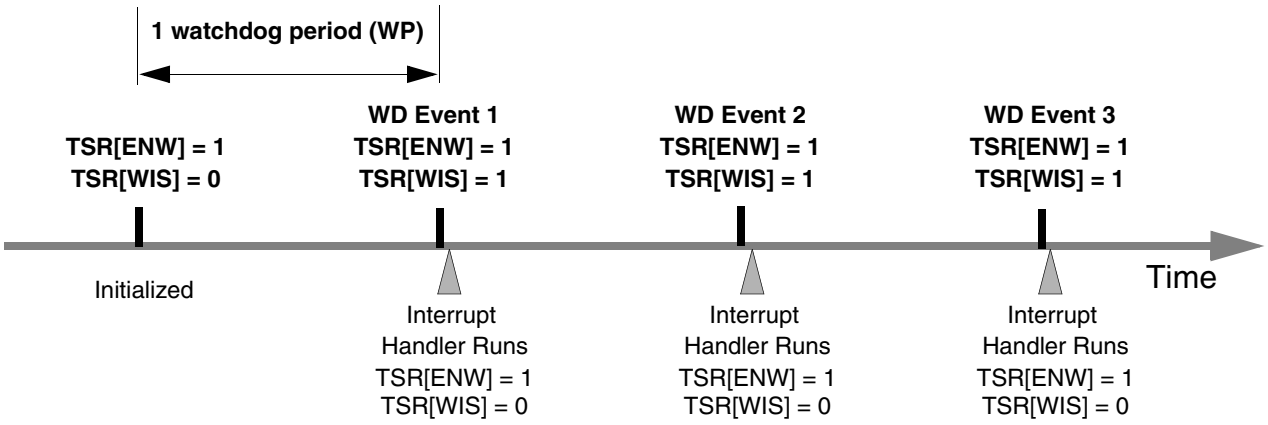

Figure 8. Interrupt Handler Sequence - Method 3

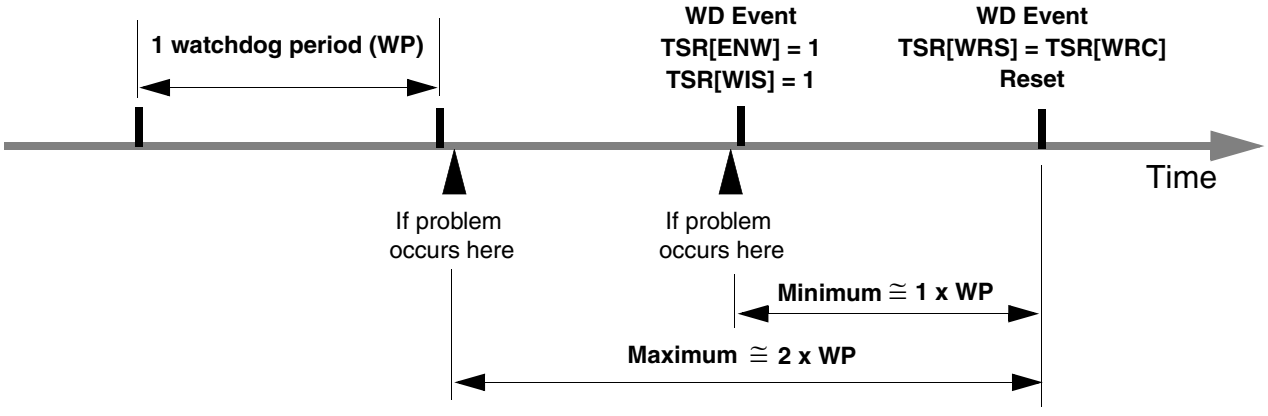

Figure 9. Effective Watchdog Reset Timeout - Method 3

3 MPC5500 Watchdog Registers

This section describes the various registers and actions required for configuring and controlling the MPC5500 watchdog.

3.1 Setting the Watchdog Reset Control

The MPC5500 device may be configured to do one of three things upon detection of the second[1] timeout event; 1) nothing, 2) force a processor checkstop, or 3) force a processor reset. The watchdog timeout action is set by the TCR[WRC] bits.

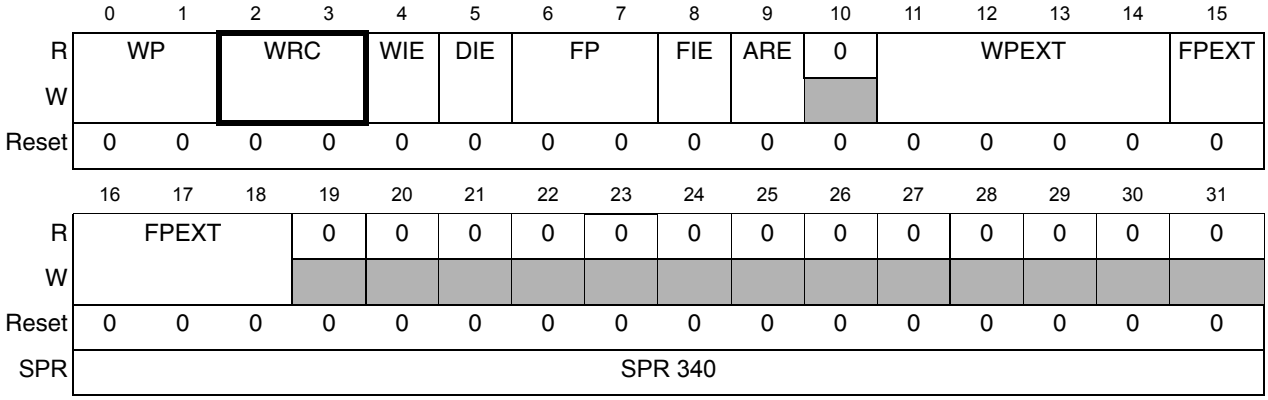

Figure 10. Timer Control Register - Watchdog Reset Control Bits (TCR[WRC])

The TCR[WRC] bits function is defined as follows:

WRC[0]	WRC[1]	Function
0	0	No action
0	1	Force a processor checkstop.
1	0	Force a processor reset.
1	1	Reserved.

The TCR[WRC] bits may be written only once by software after reset. Further writes to these bits will have no effect until they are cleared by a reset.

3.2 Setting the Watchdog Timeout Value

The watchdog timeout period is configured using the TCR register, shown in Figure 11. The relevant bits are highlighted.

1. Depending on the method of implementation, this could be the third timeout event instead.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WP		WRC		WIE	DIE	FP		FIE	ARE	0	WPEXT				FPEXT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FPEXT			0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPR	SPR 340															

Figure 11. Timer Control Register - Watchdog Period Bits (TCR[WP], TCR[WPEXT])

MPC5500 Time Base

The MPC5500 incorporates a 64-bit time base consisting of two 32-bit registers, time base upper (TBU) and (TBL), where TBU represents the most significant 32 bits of the 64-bit counter. The time base is incremented at the system clock frequency. The time base registers are illustrated below.

Register Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Counter Bit	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33

TBU

Register Bit	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Counter Bit	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

TBL

The watchdog timeout period is controlled by the system clock frequency and the values programmed in the TCR[WP] and TCR[WPEXT] bits. These two bit fields are concatenated to form a 6-bit binary value that can represent a value from 0 to 63. This concatenated value represents and selects the corresponding bit in the time base. When the selected bit in the time base transitions from 0 to 1, a watchdog timer exception is signaled.

Example

In this example TCR[WP] = 0b00 and TCR[WPEXT] = 0b1001.

These two bit fields are concantenated (WPEXT represents the most significant bits) to form 0b100100, or decimal 36.

This is TBL[36], or bit 28 of the counter (see TBL diagram).

Assume a system clock frequency of 80 MHz. The timeout value is represented in seconds as:

$$\frac{1}{80,000,000} \times 2^{28} = 3.3554432 \quad \text{Seconds}$$

NOTE

This timeout value is the period for one watchdog event. Depending on the method being used to control the watchdog, the effective timeout before a reset may vary significantly. See "Timing Considerations", Section 2.1.3, Section 2.2.5, and Section 2.3.4 for further information.

3.3 Enabling Interrupts

The watchdog timer may be configured to generate an interrupt by setting the watchdog interrupt enable bit (TCR[WIE]). The TCR register is shown in Figure 12.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WP		WRC		WIE	DIE	FP		FIE	ARE	0	WPEXT				FPEXT
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FPEXT			0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPR	SPR 340															

Figure 12. Timer Control Register - Watchdog Interrupt Enable (TCR[WIE])

The TCR[WIE] bit function is defined as follows:

TCR[WIE]	Function
0	Watchdog interrupt is disabled.
1	Watchdog interrupt is enabled.

The critical interrupt is enabled by setting the MSR[CE] bit. The MSR register is show in Figure 13. A complete description of the MSR register is found in the e200z6 reference manual.

	0				4	5	6	7						13	14	15
R	0	0	0	0	0	UCLE	SPE	0	0	0	0	0	0	0	CE	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28			31
R	EE	PR	FP	ME	FE0	0	DE	FE1	0	0	IS	DS	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13. Machine State Register - Critical Interrupt Enable Bit (MSR[CE])

The MSR[CE] bit function is defined as follows:

MSR[CE]	Function
0	Critical input and watchdog timer interrupts are disabled.
1	Critical input and watchdog timer interrupts are enabled.

3.4 Enabling the Time Base

The MPC5500 time base is the clock source for the watchdog and must be enabled by setting the HID0[TBEN] bit. If the time base is disabled at some later time, the watchdog timer will also stop. The time base must run continuously to assure watchdog timeouts.

	0	1				5	6	7	8	9	10	11		13	14	15
R	0	0	0	0	0	0	BPRED		0	0	0	0	0	0	ICR	NHR
W	0	0	0	0	0	0			0	0	0	0	0	0		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24							31
R	0	TBEN	0	DCLREE	DCLRCE	CICLRDE	MCCLRDE	DAPUEN	0	0	0	0	0	0	0	0
W	0		0						0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SPR	SPR 1008															

Figure 14. Hardware Implementation-Dependent Register 0 - Time Base Enable Bit (HID0[TBEN])

The HID0[TBEN] bit function is defined as follows:

HIDO[TBEN]	Function
0	Time base is disabled.
1	Time base is enabled.

3.5 Servicing the Watchdog

Once enabled, the watchdog must be controlled by one of the methods described in Section 2. The TSR register shown in Figure 15 is used for this function.

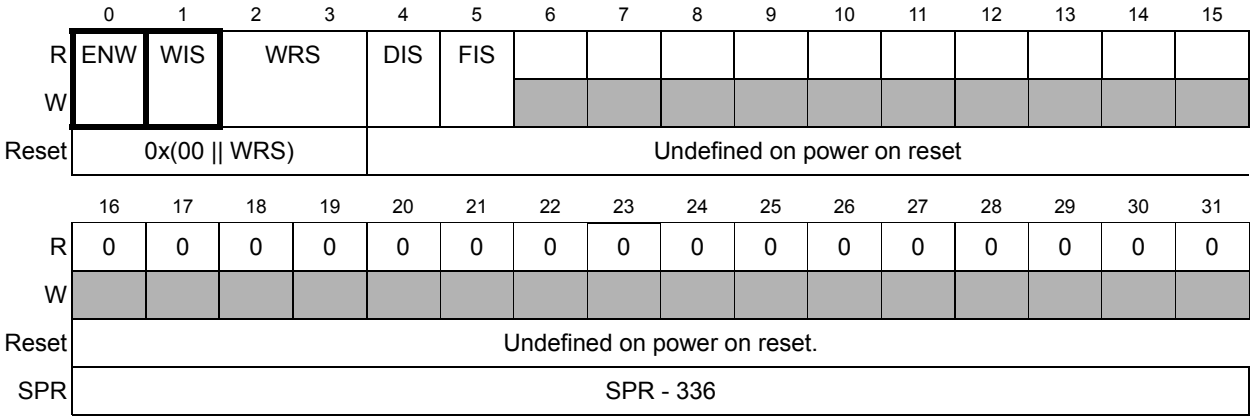

Figure 15. Timer Status Register (TSR)

The enable next watchdog bit (TSR[ENW]) controls the action that occurs on the next timeout. This bit is automatically manipulated by the watchdog logic and application intervention is not required. However, the TSR[ENW] may be cleared by the application (as in Method 2). The TSR[ENW] bit is defined as follows:

TSR[ENW]	Function
0	Action on next watchdog timer timeout is to set TSR[ENW].
1	Action on next watchdog timer timeout is governed by TSR[WIS].

The watchdog interrupt status bit (TSR[WIS]) bit is defined as follows:

TSR[WIS]	Function
0	A watchdog timer event has not occurred.
1	A watchdog timer event occurred. When MSR[CE] = 1, TCR[WIE] = 1, and TSR[ENW] = 1, a watchdog timer interrupt is taken.

4 Enabling the Watchdog Timer Using the Boot Assist Module (BAM)

The MPC5500 family provides a boot assist module (BAM). This is software that resides on the device in non-volatile memory. The BAM executes at reset and allows certain configuration options to be selected, based on the state of the BOOTCFG[0:1] and $\overline{\text{RSTCFG}}$ pins and/or the value in a specific reset configuration halfword (RCHW, provided by the user). The RCHW is located in either internal or external flash, depending on the boot mode selected. This user provided half-word may be used to enable the watchdog at boot time.

The RCHW is shown in Figure 16.

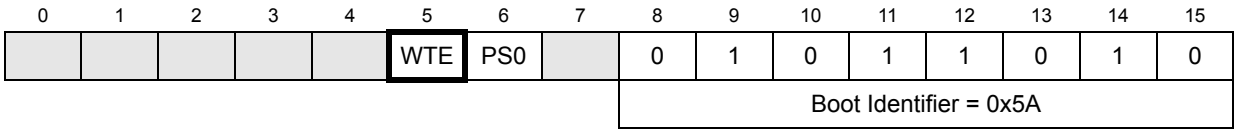

Figure 16. Reset Configuration Half Word (RCHW) Definition

The WTE bits of the RCHW determine whether the BAM enables the watchdog timer.

WTE	Description
0	BAM does not write the e200z6 timebase registers (TBU and TBL) nor enable the e200z6 core Watchdog Timer.
1	BAM writes the e200z6 timebase registers (TBU and TBL) to 0x0000_0000_0000_0000 and enables the e200z6 core Watchdog timer with a time-out period of 3×2^{17} system clock cycles. (Example: For 8 MHz crystal –> 12MHz system clock–> 32.7mS time-out. For 20 MHz crystal –> 30 MHz system clock –> 13.1mS time-out)

5 Enabling the Watchdog in Software

If serial boot mode is selected via the BOOTCFG[0:1] pins, or if the BAM fails to find a valid RCHW, the watchdog timer is enabled by default. See the application note “MPC5500 Boot Assist Module” and the BAM section of the MPC5554 Reference Manual and for more information about the watchdog service requirements in this mode.

If the RCHW is valid and configured to disable the watchdog (WRC left cleared as at reset), the user software can enable it by writing to the WRC bit. The WRC bit can only be written once by software, including a write of the bit by the BAM. Once enabled the watchdog cannot be disabled.

A Example Code

This appendix contains example code for each of the watchdog implementation and service methods described in this document.

A.1 Configuration

A.1.1 Method 1

```
#*****************************************************************************
# init_watchdog_no_irq()
#
# This function sets up the watchdog timeout and sets the action on a
# second watchdog event to be a device reset. The watchdog interrupt is
# not enabled, so no interrupt handler is required.
#
# Call with:    Nothing
# Returns:      Nothing
#*****************************************************************************

init_watchdog_no_irq:
# set for reset on 2nd timeout, set timeout for 3.3554432 seconds
      lis       r6, 0x2012      # load r6
                                # WPEXT = 0b1001, WP = 0b00
                                # WRC = 0b01, reset
                                # WIE = 0
      ori       r6, r6, 0@l     # clear lower half-word
      mtspr     TCR, r6         # move r6 to TCR

# enable time base
      mfspr     r6, HID0        # get HID0
      ori       r6, r6, 0x4000  # OR in the TBEN bit (bit 17)
      mtspr     HID0, r6        # move r6 to HID0

# return
      blr
```

A.1.2 Method 2 and 3

```
#*****************************************************************************
# init_watchdog_with_irq()
#
# This function sets up the watchdog timeout, enables the watchdog interrupt,
# and sets the action on a second watchdog event to be a device reset. This
# configuration requires an interrupt handler to service the watchdog.
#
# Call with:    Nothing
# Returns:      Nothing
#*****************************************************************************

init_watchdog_with_irq:
# enable watchdog interrupt, set for reset on 2nd timeout, set timeout for 3.3554432
# seconds
      lis       r6, 0x2812      # load r6
                                # WPEXT = 0b1001, WP = 0b00
                                # WRC = 0b01, reset
                                # WIE = 1
      ori       r6, r6, 0@l     # clear lower half-word
      mtspr     TCR, r6         # move r6 to TCR

# enable critical irqs
      mfmsr     r6              # get MSR val
      oris      r6, r6, 0x0002  # OR in the CE bit (bit 14)
      mtmsr     r6              # store val to MSR

# enable time base
      mfspr     r6, HID0        # get HID0
      ori       r6, r6, 0x4000  # OR in the TBEN bit (bit 17)
      mtspr     HID0, r6        # move r6 to HID0

# return
      blr
```

A.2 Interrupt Vector Setup

A.2.1 Methods 2 and 3

```
#*************************************************************************
# init_wd_vector()
#
# This function sets up the watchdog interrupt vector that is required
# for any implementation method requiring an interrupt service routine.
#
# Call with:    Nothing
# Returns:      Nothing
#*************************************************************************

.extern init_wd_vector

init_wd_vector:
      lis       r3, irq_handler12@h     # load r3 with upper 16-bits of handler
                                        # address
      mtspr     IVPR, r3                # move r3 to IVPR
      li r3,    irq_handler12@l         # load r3 with lower 16-bits of handler
                                        # address
      mtspr     IVOR12, r3              # move r3 to IVOR12

      blr                               # return
```

A.3 Interrupt Handler

A.3.1 Method 2

```
#*************************************************************************
# irq_handler12_2()
#
# This code is an interrupt handler that will only run when a periodic task
# fails to service the watchdog in time to prevent an interrupt. The user
# may choose to either save off debug information in preparation for a reset,
# or just clear the interrupt and hope the application recovers.
#
#*************************************************************************

.extern irq_handler12_2         # make this handler visible
```

```
# IVOR12 - watchdog interrupt

.align 4                        # align on quad-word boundary (Green Hills)
irq_handler12_2:
      # save context (might be meaningless here)
      stwu      sp, -16(sp)     # allocate 16 bytes on stack
      stw       r6, 4(sp)       # save r6 on stack so it can be used in the handler
      mfcsrr0   r6              # get CSRR0
      stw       r6, 8(sp)       # save it on the stack
      mfcsrr1   r6              # get CSRR1
      stw       r6, 12(sp)      # save it on the stack

      # option 1 -
      # recognize that the app is in an uncontrolled state and save off
      # debug information in whatever manner is appropriate prior to
      # next timeout, which will cause a reset.

      # user debug/crash code goes here

      # option 2 -
      # service the WD timeout by clearing TSR[WIS] and TSR[ENW] then
      # hope for the best.
      lis r6,   0xc000          # load r6 with TSR[ENW] and TSR[WIS] (bits 0,1)
      mtspr     TSR, r6         # move the val back to TSR

      # restore context
      lwz       r6, 12(sp)      # get CSRR1 off stack
      mtcsrr1   r6              # restore it
      lwz       r6, 8(sp)       # get CSRR0 off stack
      mtcsrr0   r6              # restore it
      lwz       r6, 4(sp)       # get r6 off stack
      addi      sp, sp, 16      # restore stack pointer

      # return from critical interrupt -
      rfci                      # restores machine state, including reenabling
                                # critical interrupts MSR[CE].
```

A.3.2 Method 3

```
#*****************************************************************************
# irq_handler12_3()
#
# This code is a simple interrupt handler that just clears the TSR[WIS] bit
# so that the interrupt is cleared. TSR[ENW] remains unchanged. This handler
# is intended to run once at each watchdog timeout. No periodic watchdog
# service is required.
#
#*****************************************************************************

.extern irq_handler12_3         # make this handler visible

# IVOR12 - watchdog interrupt

.align 4                        # align on quad-word boundary (Green Hills)
irq_handler12_3:
      # save context
      stwu      sp, -16(sp)     # allocate 16 bytes on stack
      stw       r6, 4(sp)       # save r6 on stack so it can be used in the handler
      mfcsrr0   r6              # get CSRR0
      stw       r6, 8(sp)       # save it on the stack
      mfcsrr1   r6              # get CSRR1
      stw       r6, 12(sp)      # save it on the stack

      # service the WD timeout by writing TSR[WIS] with 1
      lis r6,   0x4000          # load r6 with TSR[WIS] bit (bit 1)
      mtspr     TSR, r6         # move the val back to TSR

      # restore context
      lwz       r6, 12(sp)      # get CSRR1 off stack
      mtcsrr1   r6              # restore it
      lwz       r6, 8(sp)       # get CSRR0 off stack
      mtcsrr0   r6              # restore it
      lwz       r6, 4(sp)       # get r6 off stack
      addi      sp, sp, 16      # restore stack pointer

      # return from critical interrupt -
      rfci                      # restores machine state, including reenabling
                                # critical interrupts MSR[CE].
```

A.4 Periodic Watchdog Service Routine

A.4.1 Method 2

```
#*****************************************************************************
# clr_wd_next()
#
# This function clears the TSR[ENW] bit by writing 1 to the bit location.
#
# Call with:    Nothing
# Returns:      Nothing
#*****************************************************************************

.extern clr_wd_next

clr_wd_next:
      # prevent an initial watchdog timeout by writing TSR[ENW] with 1.
      lis r6,   0x8000          # load r6 with TSR[ENW] bit (bit 0)
      mtspr     TSR, r6         # move the val back to TSR
      blr                       # return
```

A.4.2 Method 3

```
#*****************************************************************************
# clr_wd_status()
#
# This function clears the TSR[WIS] bit by writing 1 to the bit location.
#
# Call with:    Nothing
# Returns:      Nothing
#*****************************************************************************

.extern clr_wd_status

clr_wd_status:
      # prevent a watchdog timeout reset by writing TSR[WIS] with 1 (clears the bit)
      lis r6,   0x4000          # load r6 with TSR[WIS] bit (bit 1)
      mtspr     TSR, r6         # move the val back to TSR
      blr                       # return
```

THIS PAGE INTENTIONALLY LEFT BLANK

HOW TO REACH US:

USA/Europe/Locations not listed:
Freescale Semiconductor Literature Distribution
P.O. Box 5405, Denver, Colorado 80217
1-800-521-6274 or 480-768-2130

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-8573, Japan
81-3-3440-3569

Asia/Pacific:
Freescale Semiconductor H.K. Ltd.
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
852-26668334

Learn More:
For more information about Freescale Semiconductor products, please visit
http://www.freescale.com

AN2817
Rev. 1.1, 06/2005

freescale™
semiconductor