Freescale Semiconductor
Application Note

AN2804
Rev. 0, 12/2004

Watchdog Timer for e500

1 Introduction

Embedded software should be self-reliant and it should not have to rely on human intervention to start the system over when the erroneous software causes a fatal error. The watchdog timer (WDT) is used to detect defective software and allows the system to reset itself when a fatal software error occurs, thereby avoiding the need for an operator to manually reset the system. For example, the ‘infinite for’ loop in a code can cause an erroneous behavior of the software because the system may get stuck in the infinite loop. This is not a desirable situation since all other software modules can no longer use the CPU. Therefore, the critical code of an embedded software should be designed in such a way that it would start the watchdog just prior to entering that code region. If the code doesn’t complete within a calculated amount of time, the watchdog reset will occur. On the other hand, if a code under investigation completes within a defined amount of time, then it would be able to restart the watchdog timer and the timeout will not occur, which will prevent the system from resetting. However, the software designer needs to define the timeout for the watchdog very carefully.

In this application note, the on-chip watchdog timer of the MPC8555 silicon is discussed. The WDT is a part of the e500 core complex module of the MPC8555. This documentation also discusses the software accompanied with this application note.

Contents

2 Overview

The watchdog timer is part of the e500 core complex. The state machine for this watchdog is shown in Figure 1. Note that programming the WDT only requires a few special purpose registers of the e500. These are discussed in detail in the following sections.

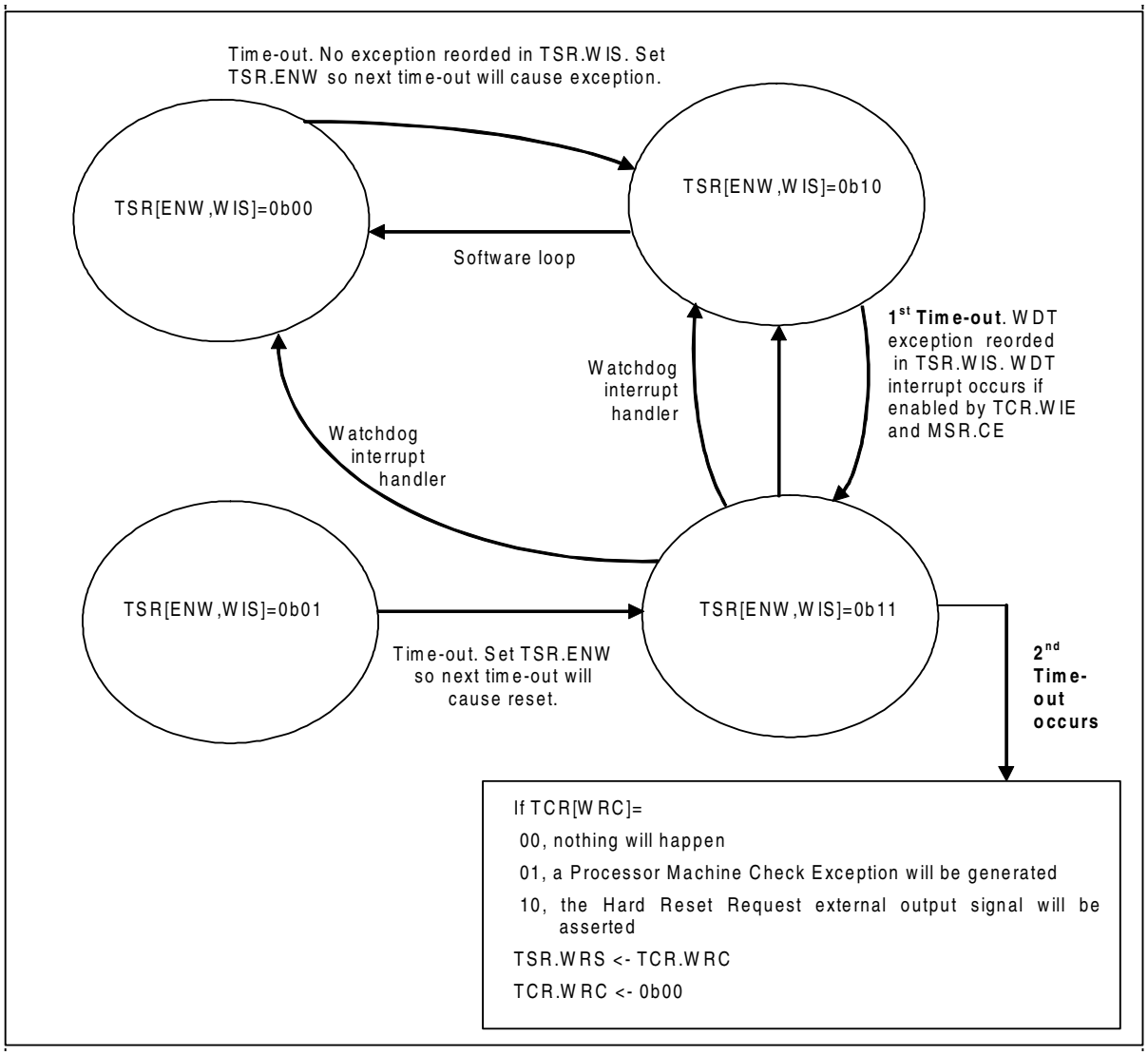

Figure 1. Watchdog State Machine

The state machine is defined by the status of the two bit Timer Status Register (TSR), which is described below. For a detailed discussion on the state machine of the WDT, please refer to Section 3.5 of the *PowerPC e500 Core Complex Reference Manual*.

The state of the watchdog timer state machine is stored in the Timer Status Register (TSR - SPR336). This register is always set by hardware on successive time-outs of the watchdog reset timer. The TSR can only be reset by software to force the state machine into a previous state. The two bits that hold the status are TSR[ENW] (Enable Next Watchdog timer timeout - bit 32) and TSR[WIS] (Watchdog Interrupt Timer Status - bit 33) as shown in Figure 1. Both TSR[ENW] and TSR[WIS] are set to 0 at reset. In the following discussion, the coordinate symbol (x,y) is used to show the value of ENW and WIS; x being ENW and y being WIS.

Note that the transition of the state machine from (0,0) to (1,0) is not known as first timeout. Rather, the first timeout is defined as the timeout when the state machine is in (1,0) state and a timeout occurs. This first timeout moves the state of the state machine from (1,0) to (1,1).

On the first watchdog timer timeout, a watchdog timer exception is generated and logged by setting TSR[WIS] = 1. If watchdog timer interrupts are enabled (TCR[WIE] = 1 and MSR[CE] = 1), a watchdog timer interrupt will be taken. The interrupt handler would clear the WIS bit of TSR register by writing a 1 to this location. Additionally, the handler can bring the state machine to (0,0) or (1,0). In the former case, 1 will be written to the ENW bit of the TSR, bringing the state machine to the initial state of (0,0). In the latter case, the ENW bit is not cleared, leaving the state machine in this state (1,0).

When TSR[ENW] = 1 and TSR[WIS] = 1 (either the watchdog timer interrupt is not taken or the handler does not reset the state machine), the next watchdog timer timeout is effectively the second watchdog timeout. Note that on a second timeout, the state still remains at (1,1) and no state transition takes place. In this case, the behavior is determined by TCR[WRC]. In the event that either a machine check exception is generated (TCR[WRC] = 01) or hardware reset request is generated (TCR[WRC] = 10), the value of TCR[WRC] is copied into TSR[WRS] (Watchdog Timer Reset Status - bits 34:35) and the value of TCR[WRC] is reset back to 00.

3 Configuration of the Watchdog Timer

3.1 Defining the Timeout

The watchdog timer is based on the Time Base Registers (Time Base Lower TBL - Read = SPR268, Write = SPR284 and Time Base Upper TBU - Read = SPR269, Write = SPR285). The watchdog timer will timeout on a low to high transition of one of the 64 bits of TBU/TBL. This bit is set by configuring with the Watchdog Timer Period in the Timer Control Register (TCR - SPR 340). The Watchdog Timer period is a 6-bit concatenation of {TCR[WPEXT], TCR[WP]} representing bits TCR[43:46, 32:33]. When {TCR[WPEXT], TCR[WP]} = 0b11_1111, the LSB of the Time Base (TBL[63]) is selected as the watchdog timer period, resulting in the shortest possible timeout period. Decrementing {TCR[WPEXT], TCR[WP]} increases the timer period by one bit of the Time Base until {TCR[WPEXT], TCR[WP]} = 0b00_0000, the MSB of the Time Base (TBU[32]) is selected resulting in the longest possible timeout. See Figure 2 below.

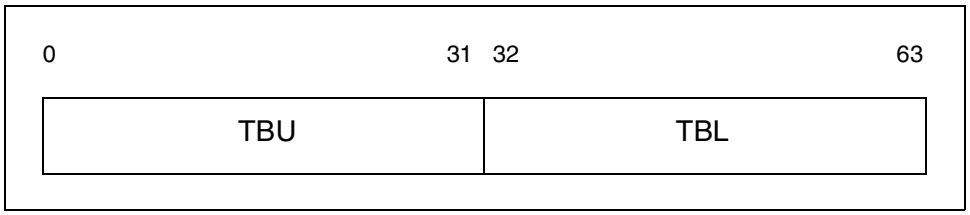

Figure 2. Bit in Timebase to be Toggled

3.1.1 An Example

The example below explains how to calculate the period of the watchdog timer and which bit needs to be toggled in the Timebase register.

Period

When WPEXT[0:3]=0b1001 and WP[0:1]=0b00, the **period** is 36. Period 36 is counted from the left in Figure 1. The bit position that must be toggled is counted from the right in Figure 2. In this case, the bit position of the Timebase which needs to be toggled to get a timeout is 63 – 36 + 1 = 28.

1st and 2nd Timeout

Once the Timebase is enabled on every 8 Platform/CCB (Core Complex Bus) cycles, the value of the Timebase is incremented once. Position 28 will toggle in the following fashion:

- In (0.5 x timeout) time, this position will be toggled once, which will move the state machine of the watchdog from (00) to (10). Note that, by definition, this is not considered to be the first timeout.
- In the next (1 x timeout) time, this position will be toggled again (first timeout) and the state machine would change from (10) to (11).
- In another (1 x timeout) time, this position will be toggled once again (second timeout) and the state machine would change from (10) to (11).

Note that in the following calculations, CCB clock rate is assumed to be 266MHz.

The total time needed for the first timeout is calculated by adding the time in the first two bullets shown above:

$1.5 \times (2\text{^}(63 - \text{period} + 1)) / (\text{CCBclock rate} \div 8) = 12.11 \text{ sec}$

The total time needed for the second timeout is calculated by adding the time in the above three bullets:

$2.5 \times (2\text{^}(63 - \text{period} + 1)) / (\text{CCB clock rate} \div 8) = 20.18 \text{ sec}$

3.2 Timebase Register

In order for watchdog timer timeouts to occur, a 0-1 transition on the appropriate bit of the Time Base Registers must occur. This can be done in the following way:

Enable the time base and supply it a clock using the HID0 (SPR1008 - Hardware Independent Register 0). Setting HID0[TBEN} (bit 49 of HID0) will cause the Time Base Register to be incremented at each cycle of the timebase clock. The Time Base clock is controlled by HID0[SEL_TBCLK] (bit 50 of HID0). When HID0[SEL_TBCLK] = 0, the platform clock/CCB clock is used; otherwise, if HID0[SEL_TBCLK] = 1, an external input TBCLK is used.

3.3 Action on aTimeout

The Timer Control Register (TCR - SPR340) is used to control the actions that occur on a timeout using the fields TCR[WIE] (bit 36 - Watchdog Timer Interrupt Enable) and TCR[WRC] (bits 34:35 - Watchdog Timer Reset Control). The number of timeouts that occur is monitored using the Timer Status Register (TSR). If TCR[WIE] = 1, a watchdog timer interrupt will be generated on the first timeout of the watchdog timer. The general purpose of this interrupt handler is to reset the state machine using the TSR such that no action is taken on the second timeout of the watchdog timer. In the event that either TCR[WIE] = 0 or the Watchdog Timer Interrupt handler does not reset the state machine, one of the following will occur on the second watchdog timer timeout based on the value of TCR[WRC]:

- If TCR[WRC] = 00, nothing will happen
- If TCR[WRC] = 01, a Processor Machine Check Exception will be generated

- If TCR[WRC] = 10, the Hard Reset Request external output signal will be asserted
- TCR[WRC] = 11 is reserved

After a machine check exception is taken, logic resets the TCR[WRC] field.

3.3.1 Taking the Watchdog Timer Interrupt on a First Timeout

When a watchdog timer interrupt event is generated, the interrupt will be taken only if watchdog timer interrupts are enabled (TCR[WIE] = 1 and MSR[CE] = 1). In this case, the routine at IVOR12 (Watchdog timer interrupt offset register - SPR412) will be executed.

3.3.2 Taking the Machine Check Exception on a Second Timeout

When a watchdog timer machine check condition is generated, it will be indicated in MPCSUMR[WRS] (Machine Check Summary Register, offset 0xE_0090 - bit 29). When MPCSUMR[WRS] = 1, the system has generated a watchdog timer machine check event. The core will only respond to the machine check event if watchdog timer (and external input) machine checks are enabled. This is enabled by setting HID0[EMCP] = 1 (Hardware Implementation Dependent Register 0, bit 32 - Enable Machine Check Pin). When HID0[EMCP] = 1 and if MSR[ME] = 1 (Machine State Register - Machine check Enable, but 51), the machine check exception will be taken by the core, in which case the routine at IVOR1 (Machine Check Interrupt Offset register) will be invoked.

4 Description of the Software and Hardware

The software tests the functionality of the WDT and runs on the CDS platform. It was also tested in the MPC8560ADS Pilot board. For the purpose of discussion we restrict the platform to be CDS. The platform is described in the following sections. Also, the files in the software project and the important routines are described. In order to get a detailed understanding of the software, please browse through the source files. The main.c file describes what are the various platforms on which this software is tested. Note that whichever platform you pick among the ones listed in the main.c file, the platform/Core Complex Bus frequency has to be set to 266MHz. Because the mathematical values used in the project assumes the platform frequency to be 266MHz.

The software was tested both in Debug mode and in Flash mode. By Flash mode we mean programming the ROM image into the onboard Flash of the platform and then run it in a standalone fashion. If you are interested in the ROM version then select the ROM Version from the drop down list of the ‘Debug Version Setting’ of the CodeWarrior project, compile it and then with the use of the ‘Flash Programmer’ of the CodeWarrior IDE program the image into the FLASH.

4.1 Platform

Target System : CDS board

Debugger: MetroWorks CodeWarrior for PowerQUICC III, Rev 1.1

Debugger Probe: PowerTAP Pro

4.2 Files

Table 1. Files

File Name	File Description
main.c	This file contains all the WDT APIs.
interrupt.c	Contains the exeption vector table, including the machine check handler and watchdog timer interrupt
tuning.c	Contains definitions used in main.c and interrupt.c
time.c	It provides the utility routines to perform the time measurements. In the WDT project the timeout period is programmed and then using the utility it is measured.
UART2_MOT_8540_ADS.UC.a	Serial driver is a part of the standard CodeWarrior source tree. Provided in the package for convenience. Typically this serial driver component is picked up from the Codewarrior's source installation tree.
__ppc_eabi_init.c	Iniatialization related file which is part of the standard CodeWarrior source tree but provided with the package for convenience.
8555cds_init.c	Used with the ROM version only. This file contains all the low level intializations.
eppc_exception.c	Contains the low level layout of all the e500 related exceptions. This is part of the standard CodeWarrior project.
reset.c	Part of this package and also part of standard CodeWarrior project. This file is used when you select the ROM target which you want to program into the Flash of your platform.
MSL_C.PPCEABI.bare.E.UC.a	Standard CodeWarrior library which includes complete C and C++ library collection.
Runtime.PPCEABI.E.UC.a	Standard Codewarrior runtime libray.
flash_prog_bank_0.xml	This is the *.xml file which holds all the configuration parameters needed to program the flash. For your use you have to make sure that the paths provided for certains files are accurate.

4.3 Routines

The important routines are defined in main.c. These are described in the following sections.

4.3.1 WatchDogCreate( ) - create a watchdog timer

void WatchDogCreate (int delay, int FirstTimeout, int SecondTimeout)

DESCRIPTION

This routine configures the watchdog timer of the e500 core complex. By the argument delay, the user provides the timeout in milliseconds. This timeout is for the WDT's second timeout. Also, FirstTimeout argument indicates whether the user wants to take an interrupt upon hitting the first timeout of the WDT. This argument can have a value of 0 or 1. If the value is 1, that indicates that the user wants to interrupt on the first timeout; otherwise, if the value is 0, no interrupt is taken on the first timeout. Similarly, SecondTimeout argument specifies whether the user wants to take a second timeout and get a machine check exception. Note that the acceptable values for FirstTimeout and SecondTimeout respectively are 1,0 and 0,1.

The following steps are performed in this routine:

- User's specified delay is converted to period.
- WP and WPEXT fields are programmed based upon the calculated period.
- By setting the TBCLK bit of the HID0 register, the platform/CCB clock is routed to WDT. Note that the TBEN bit is not set at this point, so the Timebase will not start counting.
- Finally, depending upon whether the user configured to take an interrupt on the first or second timeout, the TCR register is programmed. If the first timeout is chosen, then the WIE bit in TCR is set, as is the CE bit in the MSR. For the second timeout, WRC bit field is set to 0b01, ME bit in the MSR is set, and EMCP bit in the HID0 is set, which ensures that on a second timeout, machine check exception is generated and the corresponding handler is invoked.

RETURNS

None

4.3.2 WatchDogStart( ) - starts the watchdog timer

void WatchDogStart (void)

DESCRIPTION

This routine starts the WDT, assuming that the WDT has been created using WatchDogCreate() prior to calling this routine. The following steps are performed in the WatchDogStart routine:

- Reset Timebase.
- Enable Timebase by setting the TBEN bit of the HID0 register. At this point, the Timebase will start counting. When counting reaches the period of WDT, the timeout happens.

RETURNS

None

4.3.3 WatchDogCancel ( ) – stops the watchdog timer

void WatchDogCancel (void)

DESCRIPTION

This routine would stop the watchdog timer so that the timeouts do not take place.

The following actions are performed in this routine:

- ENW,WIS and WRS bits are cleared to bring the watchdog timer to its initial state, which is TSR[ENW,WIS]=0b00.
- WP and WPEXT fields of the TCR register are cleared.
- Timebase registers are cleared.

RETURNS

None

4.4 Interrupts and Exceptions

The project eppc_exception.asm file contains the exception vector table. In this file, IVOR1 is for machine check exception and IVOR12 is for Watchdog timer Interrupt. On a first timeout, IVOR12 is taken, and on a second timeout, IVOR1 is taken. Once the code jumps to either of these locations, it calls the InterruptHandler routine located in the interrupt.c file. The switch statements in this routine differentiate among various interrupts: ‘case 0x200’ is for machine check and ‘case 0xB00’ is for watchdog timer interrupt.

4.5 Example Run and Results

The main routine calls the WatchDogCreate, followed by WatchDogStart, which kicks off the Timebase. The watchdog timer’s state machine begins watching for the timeout that is going to occur. The program then comes is a tight loop keeping the CPU busy. Once the first/second timeout happens, the CPU jumps to the corresponding handler. Inside the handler, the timebase is stopped so that it does not roll over or continue counting. Then, the status in the TSR register is cleared. If it is a second timeout, then the status in the MCPSUMR register is also cleared; particularly the WRS bit in the MCPSUMR register is cleared. Then, the TCR register and timebase is programmed again so that timeout can keep happening. The following two cases were tested out in the lab:

CASE 1: Second timeout, 20sec

WatchDogCreate(20000, 0, 1); //20,000msec

WatchDogStart ();

This case is used to test out the 2nd timeout with a timeout of 20 seconds. It was observed that software was printing the following things (shown in bold below) at the console (57600 baud rate) after almost every 20 seconds. The console dump is made from the machine check handler. Software prints out the state of WDT at the time when the machine check handler is invoked. Then, it also prints the state after the status is cleared in the TSR register. Notice that once you clear the status of the TSR register, the WDT will come back to its initial state.

2nd tm

Watchdog state before...
End State : TSR[ENW,WIS] = 0b11
Watchdog state after...
End State : TSR[ENW,WIS] = 0b00

CASE 2 : First timeout, 3sec

WatchDogCreate(5046, 1, 0);//5046msec

WatchDogStart ();

This case is used to test out the 1st timeout with a timeout of ((1/2.5) × (1.5) × 5046 msec) = 3 seconds. Note that in WatchDogCreate, the user is asked to provide a timeout value for the 2nd timeout. The first timeout value is then calculated in software.

In this experiment, it was observed that software was printing the following things (shown in bold below) at the console after almost every 3 seconds. The console dump is made from the watchdog timer handler. Software prints out the state of the WDT at the time when the watchdog timer handler is invoked. Then, it also prints the state after the status is cleared in the TSR register. Notice that once you clear the status of the TSR register, the WDT will come back to its initial state.

1st tm

Watchdog state before...
TSR[ENW,WIS] = 0b11
Watchdog state after...
TSR[ENW,WIS] = 0b00

4.6 Example Usage of the Watchdog Timer APIs Provided

Consider an embedded system where an I/O pin is driven low from high after every 50 msec. The embedded software reads this port and, if the logic value of this pin is zero, it writes to some other I/O port. Consider the embedded software to be something like what is shown in Listing 1 below.

Suppose that the loop (in Listing 1 below) must execute at least once every fifty milliseconds. If the watchdog timer's counter is initialized to a value that corresponds to fifty milliseconds of elapsed time, and the software has no bugs, the watchdog timer will never expire; the software will always restart the counter before it reaches zero.

Listing 1: Restarting the Watchdog

```
main(void)
{
    int value;
    for (;;)
    {
        /* kicking/restarting the dog */
        WatchDogCreate(50, 0, 1);//50msec
        WatchDogStart ();
```

```
        /* code that should finish in 50msec */
        while (!(value =  read_io_port()));
        write_io_port();
        print_status();

        /* stopping the WDT */
        WatchDogCancel ()
    }
}
```

Suppose that the software gets stuck in the ‘while’ loop. This loop should break when the value read from the I/O becomes 0. If the I/O is not driven to 0 from 1, this means that the software is stuck in this ‘while’ loop. Meanwhile, the WDT states rolled over and 2nd timeout has happened. This would allow the software to break the ‘while’ loop because now the machine check handler will be taken and the CPU can then reset the entire system.

5 Conclusion

This application note discussed the watchdog timer of e500 core complex, as well as the accompanying software and the test results of the software. The software code is compiled and tested using the Codewarrior debugger on CDS platform. This software can be easily ported to some other platform as well. An example usage of the APIs developed is provided in this application note.

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Technical Information Center
3-20-1, Minami-Azabu, Minato-ku
Tokyo 106-0047 Japan
0120 191014
+81 3 3440 3569
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@
hibbertgroup.com

AN2804
Rev. 0
12/2004

freescale™ semiconductor