Actel SmartFusion® MSS Watchdog Driver User's Guide

Version 2.0

Actel POWER MATTERS

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200190-0

Release: December 2009

Trademarks

Table of Contents

Introduction

The SmartFusion® microcontroller subsystem (MSS) includes a watchdog timer used to detect system lockups. This software driver provides a set of functions for controlling the MSS watchdog as part of a bare metal system where no operating system is available. This driver can be adapted for use as part of an operating system but the implementation of the adaptation layer between this driver and the operating system's driver model is outside the scope of this driver.

Features

The MSS watchdog driver provides the following features:

- Support for initializing and configuring the MSS watchdog
- Reading the current value and status of the watchdog timer
- Refreshing the watchdog timer value
- Support for enabling, disabling and clearing time-out and wake-up interrupts.

The MSS watchdog driver is provided as C source code.

Supported Hardware IP

The MSS watchdog bare metal driver can be used with Actel's MSS_WATCHDOG IP version 0.1 or higher included in the SmartFusion MSS.

Files Provided

The files provided as part of the MSS watchdog driver fall into three main categories: documentation, driver source code, and example projects. The driver is distributed via the Actel Firmware Catalog, which provides access to the documentation for the driver, generates the driver's source files into an application project, and generates example projects that illustrate how to use the driver. The Actel Firmware Catalog is available from the Actel web site: www.actel.com/products/software/firmwarecat/default.aspx.

Documentation

The Actel Firmware Catalog provides access to these documents for the driver:

- User's guide (this document)
- A copy of the license agreement for the driver source code
- Release notes

Driver Source Code

The Actel Firmware Catalog generates the driver's source code into the *drivers\mss_watchdog* subdirectory of the selected software project directory. The files making up the driver are detailed below.

mss_watchdog.h

This header file contains the public application programming interface (API) of the MSS watchdog software driver.

This header file also contains the implementation of the MSS watchdog software driver.

This file should be included in any C source file that uses the MSS watchdog software driver.

Example Code

The Actel Firmware Catalog provides access to example projects illustrating the use of the driver. Each example project is self contained and is targeted at a specific processor and software toolchain combination. The example projects are targeted at the FPGA designs in the hardware development tutorials supplied with Actel's development boards. The tutorial designs may be found on the Actel Development Kit web page (www.actel.com/products/hardware).

Driver Deployment

This driver is intended to be deployed from the Actel Firmware Catalog into a software project by generating the driver's source files into the project directory. The driver uses the SmartFusion Cortex Microcontroller Software Interface Standard – Peripheral Access Layer (CMSIS-PAL) to access MSS hardware registers. You must ensure that the SmartFusion CMSIS-PAL is either included in the software tool chain used to build your project or is included in your project. The most up-to-date SmartFusion CMSIS-PAL files can be obtained using the Actel Firmware Catalog.

The following example shows the intended directory structure for a SoftConsole ARM® Cortex™-M3 project targeted at the SmartFusion MSS. This project uses the MSS GPIO and MSS watchdog drivers. Both of these drivers rely on SmartFusion CMSIS-PAL for accessing the hardware. The contents of the *drivers* directory result from generating the source files for each driver into the project. The contents of the *CMSIS* directory result from generating the source files for the SmartFusion CMSIS-PAL into the project.

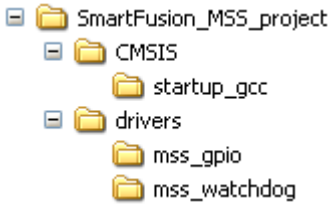

Figure 1 · SmartFusion MSS Project Example

Driver Configuration

The configuration of all features of the MSS watchdog is covered by this driver. There are no dependencies on the hardware flow when configuring the SmartFusion MSS watchdog timer.

Make sure that the MSS watchdog is not disabled elsewhere in your application program or by your software development toolchain, before using this driver. Once disabled, the MSS watchdog can only be re-enabled by a power-on reset.

The base address and register addresses and interrupt number assignment for the MSS watchdog block are defined as constants in the SmartFusion CMSIS-PAL You must ensure that the SmartFusion CMSIS-PAL is either included in the software tool chain used to build your project or is included in your project.

Application Programming Interface

This section describes the driver's API. The functions and related data structures described in this section are used by the application programmer to control the MSS watchdog peripheral from the user's application.

Theory of Operation

The MSS watchdog driver functions are grouped into the following categories:

- Initialization and configuration
- Reading the current value and status of the watchdog timer
- Refreshing the watchdog timer value
- Support for enabling, disabling and clearing time-out and wake-up interrupts.

Initialization and configuration

The watchdog driver is initialized and configured through a call to the *MSS_WD_init()* function. The *MSS_WD_init()* function must be called before any other watchdog driver functions can be called with the exception of the *MSS_WD_disable()* function which may be called at any time.

The parameters passed to the *MSS_WD_init()* function specify the watchdog timer configuration. The configuration parameters include the value that will be reloaded into the watchdog timer down counter every time the watchdog is refreshed. Also included as part of the configuration parameters is the value for the permitted refresh window. When the current value of the watchdog timer is greater than the permitted refresh window value, refreshing the watchdog is forbidden. Attempting to refresh the watchdog timer in this circumstance will cause a reset or interrupt depending on the watchdog configuration. The permitted refresh window can be disabled by specifying a permitted refresh window value equal to or higher than the watchdog reload value. Finally, the configuration parameters specify the watchdog's operating mode in the event that the watchdog timer expires.

The watchdog timer can be disabled using the *MSS_WD_disable()* function. Once disabled, the watchdog timer can only be re-enabled by a power-on reset.

Reading the watchdog timer value and status

The current value of the watchdog timer can be read using the *MSS_WD_current_value()* function. The watchdog status can be read using the *MSS_WD_status()* function. These functions are typically required when using the watchdog configured with an permitted refresh window to check if a watchdog reload is currently allowed.

Refreshing the watchdog timer value

The watchdog timer value is refreshed using the *MSS_WD_reload() function*. The value reloaded into the watchdog timer down counter is the value specified earlier as a parameter to the *MSS_WD_init()* function.

Interrupt control

The watchdog timer can generate interrupts instead of resetting the system when its down counter timer expires. The MSS watchdog driver provides the following functions to control time-out interrupts:

- *MSS_WD_enable_timeout_irq*
- *MSS_WD_disable_timeout_irq*
- *MSS_WD_clear_timeout_irq*

The watchdog timer is external to the Cortex-M3 processor core and operates even when the Cortex-M3 is in sleep mode. A wakeup interrupt can be generated by the watchdog timer to wake up the Cortext-M3 when the watchdog timer value reaches the permitted refresh window while the Cortex-M3 is in sleep mode. The MSS watchdog driver provides the following functions to control wakeup interrupts:

- *MSS_WD_enable_wakeup_irq*
- *MSS_WD_disable_wakeup_irq*
- *MSS_WD_clear_wakeup_irq*

Types

There are no MSS watchdog driver specific types.

Constant Values

The driver defines the following macros for use as function arguments.

MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE

The MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE macro is one of the allowed values for the mode parameter of function the *WD_init()* function. It is used to specify that the MSS watchdog will generate a timeout interrupt when the watchdog's down counter expires.

MSS_WDOG_RESET_ON_TIMEOUT_MODE

The MSS_WDOG_RESET_ON_TIMEOUT_MODE macro is one of the allowed values for the *mode* parameter of the *WD_init()* function. It is used to specify that the MSS watchdog will generate a reset when the watchdog's down counter expires.

MSS_WDOG_NO_WINDOW

The MSS_WDOG_NO_WINDOW macro can be used as the value for the *reload_window* parameter of the *WD_init()* function. It is used to specify that no forbidden window will exist for the reload of the watchdog's down counter.

Data Structures

There are no MSS watchdog driver specific data structures.

Functions

MSS_WD_init

Prototype

```
void
MSS_WD_init
(
        uint32_t load_value,
        uint32_t reload_window,
        uint32_t mode
);
```

Description

The *MSS_WD_init()* function initializes the SmartFusion MSS watchdog. It disables interrupt generation and clears any pending interrupts. It sets up the watchdog timer reload value, the permitted reload window value and the watchdog's operating mode. It also causes the watchdog to reload its down counter timer with the new reload value and begin counting down from this value.

Parameters

load_value

The *load_value* parameter specifies the value that will be loaded into the watchdog's down counter when the reload command is issued through a call to *MSS_WD_reload()*. The *load_value* can be any 32 bit unsigned integer.

reload_window

The *reload_window* parameter specifies the time window during which a reload of the watchdog's down counter is permitted. A reload of the watchdog's down counter should only be performed when the down counter value is below the *reload_window* value. Reloading the watchdog's down counter value before it has reached the reload window will result in an interrupt or reset depending on the watchdog's mode. The *reload_window* can be any 32 bit unsigned integer.

Note: The permitted reload window can be disabled by using WDOG_NO_WINDOW for this parameter.

mode

The *mode* parameter specifies the watchdog's operating mode. There are 2 options for the *mode* parameter.

A reset will occur if the watchdog timer expires and the *mode* is parameter specified as:

- MSS_WDOG_RESET_ON_TIMEOUT_MODE

An NMI interrupt will occur if the watchdog timer expires and the *mode* parameter is specified as:

- MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE

Return Value

This function does not return a value.

Example

This following call will initialize the reload value of the MSS watchdog to 0x10000000, disable the permitted reload window and configure the watchdog to generate an NMI interrupt will occur if the watchdog timer expires.

```
MSS_WD_init( 0x10000000, MSS_WDOG_NO_WINDOW, MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE );
```

MSS_WD_reload

Prototype

```
void
MSS_WD_reload
(
        void
);
```

Description

The *MSS_WD_reload()* function causes the watchdog to reload its down counter timer with the load value configured through the call to *WD_init()*. This function must be called regularly to avoid a system reset or a watchdog interrupt.

Return Value

This function does not return a value.

Example

This following call causes the watchdog to reload its down counter timer with 0x10000000 and then resume counting down from that value.

```
MSS_WD_init( 0x10000000, MSS_WDOG_NO_WINDOW, MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE );
MSS_WD_reload();
```

MSS_WD_disable

Prototype

```
void
MSS_WD_disable
(
        void
);
```

Description

The *MSS_WD_disable()* function disables the watchdog.

Note: Please note that the watchdog can only be re-enabled as a result of a power-on reset.

Return Value

This function does not return a value.

Example

This following call disables the watchdog.

```
MSS_WD_disable();
```

MSS_WD_current_value

Prototype

```
uint32_t
MSS_WD_current_value
(
        void
);
```

Description

The *MSS_WD_current_value()* function returns the current value of the watchdog's down counter.

Return Value

This function returns the current value of the watchdog's down counter as a 32 bit unsigned integer.

Example

Read and assign current value of the watchdog's down counter to a variable.

```
uint32_t wd_current_count;
wd_current_count = MSS_WD_current_value();
```

MSS_WD_status

Prototype

```
uint32_t
MSS_WD_status
(
        void
);
```

Description

The *MSS_WD_status()* function returns the status of the watchdog.

Return Value

This function returns the status of the watchdog. A value of 0 indicates that watchdog's down counter is within the forbidden window and that a reload should not be done. A value of 1 indicates that the watchdog's down counter is within the permitted window and that a reload is allowed.

Example

Read and assign current value of the watchdog's down counter to a variable. Reload the counter if the counter is not within the forbidden window.

```
#define PERMITTED_WINDOW (uint32_t)0x00000001
uint32_t wd_status;
wd_status = MSS_WD_status();
if ( 0 == ( wd_status & PERMITTED_WINDOW )
{
MSS_WD_reload();
}
```

MSS_WD_enable_timeout_irq

Prototype

```
void
MSS_WD_enable_timeout_irq
(
       void
);
```

Description

The *MSS_WD_enable_timeout_irq()* function enables the watchdog's timeout interrupt which is connected to the Cortex-M3 NMI interrupt. The *NMI_Handler()* function will be called when a watchdog timeout occurs.

Note: An *NMI_handler()* default implementation is weakly defined in the SmartFusion CMSIS-PAL. You must provide your own implementation of the *NMI_Handler()* function, that will override the default implementation, to suit your application.

Return Value

This function does not return a value.

Example

```
#include "mss_watchdog.h"
int main( void )
{
    MSS_WD_init( 0x10000000, MSS_WDOG_NO_WINDOW, MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE );
    MSS_WD_enable_timeout_irq();
    for (;;)
    {
        main_task();
    }
}

void NMI_Handler( void )
{
    process_timeout();
    MSS_WD_clear_timeout_irq();
}
```

MSS_WD_disable_timeout_irq

Prototype

```
void
MSS_WD_disable_timeout_irq
(
        void
);
```

Description

The *WD_disable_timeout_irq()* function disables the generation of the NMI interrupt when the watchdog times out.

Return Value

This function does not return a value.

Example

This following disables the generation of the NMI interrupt.

```
MSS_WD_disable_timeout_irq();
```

MSS_WD_clear_timeout_irq

Prototype

```
void
MSS_WD_clear_timeout_irq
(
        void
);
```

Description

The *MSS_WD_clear_timeout_irq()* function clears the watchdog's timeout interrupt which is connected to the Cortex-M3 NMI interrupt. Calling *MSS_WD_clear_timeout_irq()* results in clearing the Cortex-M3 NMI interrupt.

Note: You must call the *MSS_WD_clear_timeout_irq()* function as part of your implementation of the *NMI_Handler()* timeout interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the timeout ISR.

Return Value

This function does not return a value.

Example

The example below demonstrates the use of the *MSS_WD_clear_ timeout_irq()* function as part of the NMI interrupt service routine.

```
void NMI_Handler( void )
{
    process_timeout();
    MSS_WD_clear_timeout_irq();
}
```

MSS_WD_enable_wakeup_irq

Prototype

```
void
MSS_WD_enable_wakeup_irq
(
       void
);
```

Description

The *MSS_WD_enable_wakeup_irq()* function enables the SmartFusion wakeup interrupt. The *WdogWakeup_IRQHandler()* function will be called when a wakeup interrupt occurs.

Note: A *WdogWakeup_IRQHandler()* default implementation is weakly defined in the SmartFusion CMSIS-PAL. You must provide your own implementation of the *WdogWakeup_IRQHandler()* function, that will override the default implementation, to suit your application.

Return Value

This function does not return a value.

Example

```
#include "mss_watchdog.h"
int main( void )
{
    MSS_WD_init( 0x10000000, MSS_WDOG_NO_WINDOW, MSS_WDOG_INTERRUPT_ON_TIMEOUT_MODE );
    MSS_WD_enable_wakeup_irq();
    for (;;)
    {
        main_task();
        cortex_sleep();
    }
}

void WDOG_WAKEUP_IRQHandler( void )
{
    process_wakeup();
    MSS_WD_clear_wakeup_irq();
}
```

MSS_WD_disable_wakeup_irq

Prototype

```
void
MSS_WD_disable_wakeup_irq
(
        void
);
```

Description

The *MSS_WD_disable_wakeup_irq()* function disables the generation of the SmartFusion wakeup interrupt.

Return Value

This function does not return a value.

Example

This following call disables the generation of the wakeup interrupt.

```
MSS_WD_disable_wakeup_irq();
```

MSS_WD_clear_wakeup_irq

Prototype

```
void
MSS_WD_clear_wakeup_irq
(
        void
);
```

Description

The *MSS_WD_clear_wakeup_irq()* function clears the wakeup interrupt. This function also clears the interrupt in the Cortex-M3 interrupt controller through a call to *NVIC_ClearPendingIRQ()*.

Note: You must call the *MSS_WD_clear_wakeup_irq()* function as part of your implementation of the *WdogWakeup_IRQHandler()* wakeup interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the wakeup ISR.

Return Value

This function does not return a value.

Example

The example below demonstrates the use of the *MSS_WD_clear_wakeup_irq()* function as part of the wakeup interrupt service routine.

```
void WdogWakeup_IRQHandler( void )
{
    do_interrupt_processing();
    MSS_WD_clear_wakeup_irq();
}
```

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650. 318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (http://www.actel.com/support/search/default.aspx) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at http://www.actel.com/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/company/contact/default.aspx.

Actel is the leader in low-power and mixed-signal FPGAs and offers the most comprehensive portfolio of system and power management solutions. Power Matters. Learn more at http://www.actel.com .

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 • USA*
Phone 650.318.4200 • Fax 650.318.4600 • Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd. *• River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Building 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • http://jp.actel.com

Actel Hong Kong *• Room 2107, China Resources Building • 26 Harbour Road • Wanchai • Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200190-0/12.09