SmartFusion2 MSS Watchdog Driver User's Guide

Version 2.1

Table of Contents

Introduction

The SmartFusion2™ 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. The driver can be adapted for use as part of an operating system, but the implementation of the adaptation layer between the driver and the operating system's driver model is outside the scope of the driver.

Features

The MSS watchdog driver provides support for the following features:

- Initialization of the MSS watchdog
- Reading the current value and status of the watchdog timer
- Refreshing the watchdog timer value
- Enabling, disabling and clearing time-out and wake-up interrupts.

The MSS watchdog driver is provided as C source code.

Supported Hardware IP

The SmartFusion2 MSS watchdog bare metal driver can be used with the SmartFusion2 MSS version 0.0.500 or higher.

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 Microsemi SoC Products Group's 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 Firmware Catalog is available from: www.microsemi.com/soc/products/software/firmwarecat/default.aspx.

Documentation

The 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 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 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 the SoC Products Group's development boards. The tutorial designs can be found on the Microsemi SoC Development Kit web page.

Driver Deployment

This driver is deployed from the Firmware Catalog into a software project by generating the driver's source files into the project directory. The driver uses the SmartFusion2 Cortex Microcontroller Software Interface Standard Hardware Abstraction Layer (CMSIS HAL) to access MSS hardware registers. You must ensure that the SmartFusion2 CMSIS HAL is included in the project settings of the software toolchain used to build your project and that it is generated into your project. The most up-to-date SmartFusion2 CMSIS HAL files can be obtained using the Firmware Catalog.

The following example shows the intended directory structure for a SoftConsole ARM® Cortex™-M3 project targeted at the SmartFusion2 MSS. This project uses the MSS GPIO and MSS watchdog drivers. Both of these drivers rely on the SmartFusion2 CMSIS HAL for accessing the hardware. The contents of the *drivers* directory result from generating the source files for the driver into the project. The contents of the *CMSIS* and *hal* directories result from generating the source files for the SmartFusion2 CMSIS HAL into the project. The contents of the *drivers_config* directory are generated by the Libero project and must be copied into the into the software project.

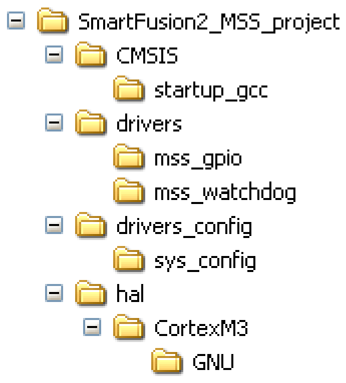

Figure 1 · SmartFusion2 MSS Project Example

Driver Configuration

The MSS watchdog must be configured by the SmartFusion2 MSS configurator in the Libero hardware flow before using this driver in a firmware project. The MSS configurator is responsible for the configuration of the MSS watchdog's mode of operation, reload value and refresh window. The MSS configurator sets the power-on reset state of the system registers that control the configuration of the MSS watchdog.

The MSS watchdog configuration options are as follows:

- Enable or disable the watchdog after a power-on or system reset
- Generate a system reset or an interrupt if a watchdog counter timeout occurs
- Set the reload value. This is the value that is loaded into the watchdog counter when it is refreshed
- Set the counter threshold value. This sets up permitted and forbidden refresh windows for the watchdog counter
- Lock the system register controlling watchdog enable/disable and timeout mode to the selected configuration (read-only) or allow it to be modified by firmware

This driver assumes that the Libero hardware flow enables the MSS watchdog, selects its timeout mode and locks the system register that controls the configuration of the MSS watchdog. If you want to have firmware control of the MSS watchdog enable/disable and its timeout mode, you must ensure that the MSS configurator allows read-write access to the system register that controls the configuration of the MSS watchdog. The driver does not provide functions to write to this system register.

The base address, register addresses and interrupt number assignment for the MSS watchdog block are defined as constants in the SmartFusion2 CMSIS HAL. You must ensure that the latest SmartFusion2 CMSIS HAL is included in the project settings of the software tool chain used to build your project and that it is generated into 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 MSS configurator in the Libero hardware flow is responsible for the configuration of the MSS watchdog's mode of operation, reload value and refresh window. It configures the watchdog to generate an interrupt or a system reset when its counter times out. It also configures the MSS watchdog to be enabled or disabled when power-on reset is deasserted.

Note: The MSS watchdog cannot be enabled or disabled by the driver.

The MSS watchdog driver is initialized 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.

The occurrence of a time out event before the last system reset can be detected through the use of the *MSS_WD_timeout_occured()* function. This function would be typically used at the start of the application to detect whether the application is starting as a result of a power-on reset or a watchdog reset. The time out event must be cleared through a call to function *MSS_WD_clear_timeout_event()* in order to allow the detection of subsequent time out events or differentiating between a Cortex-M3 initiated system reset and watchdog 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 a permitted refresh window to check whether a watchdog reload is currently allowed. 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 the forbidden window will cause a reset or interrupt, depending on the watchdog configuration. The forbidden refresh window can be disabled by the MSS configurator in the hardware flow by specifying a permitted refresh window value equal to or higher than the watchdog reload value.

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 specified in the Libero hardware flow using the MSS configurator.

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 timeout 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 Cortex-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

There are no MSS watchdog driver specific constants.

Data Structures

There are no MSS watchdog driver specific data structures.

Functions

MSS_WD_init

Prototype

```
void
MSS_WD_init
(
    void
);
```

Description

The *MSS_WD_init()* function initializes the MSS watchdog timer. It disables interrupt generation and clears any pending watchdog interrupts. It also refreshes the watchdog counter if the current count is within the permitted refresh window.

Note: This function does not enable or disable the watchdog; that is the responsibility of the hardware flow.

Note: Calling this function does not affect the hardware reported time out event indicator reported by a call to function *MSS_WD_timeout_occured()*. It is therefore allowed to call the *MSS_WD_init()* function before calling *MSS_WD_timeout_occured()*.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
  MSS_WD_init();
```

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 the MSS configurator in the hardware flow. This function must be called regularly to avoid a system reset or a watchdog interrupt.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

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.

Parameters

This function has no parameters.

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 refresh status of the watchdog.

Parameters

This function has no parameters.

Return Value

This function returns the refresh 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_timeout_occured

Prototype

```
uint32_t
MSS_WD_timeout_occured
(
    void
);
```

Description

The *MSS_WD_timeout_occured()* function reports the occurance of a timeout event. It can be used to detect if the last Cortex-M3 reset was due to a watchdog timeout or a power-on reset.

Parameters

This function has no parameters.

Return Value

0: No watchdog timeout event has occurred.

1: A watchdog timeout event has occurred.

Example

```
  #include "mss_watchdog.h"
  int main( void )
  {
      uint32_t wdg_reset;
      MSS_WD_init();
      wdg_reset = MSS_WD_timeout_occured();
      if(wdg_reset)
      {
          log_watchdog_event();
          MSS_WD_clear_timeout_event();
      }

      for(;;)
      {
          main_task();
      }
  }
```

MSS_WD_clear_timeout_event

Prototype

```
void
MSS_WD_clear_timeout_event
(
    void
);
```

Description

The *MSS_WD_clear_timeout_event()* function clears the hardware's report of a watchdog timeout event. This function would typically be called after a call to *MSS_WD_timeout_occured()* to clear the hardware's report of a timeout event and therefore enable detection of subsequent timeout events.

Parameters

This function has no parameters.

Return Value

0: No watchdog timeout event has occurred.

1: A watchdog timeout event has occurred.

Example

```
#include "mss_watchdog.h"
int main( void )
{
    uint32_t wdg_reset;
    MSS_WD_init();
    wdg_reset = MSS_WD_clear_timeout_event();
    if(wdg_reset)
    {
        log_watchdog_event();
        MSS_WD_clear_timeout_event();
    }

    for(;;)
    {
        main_task();
    }
}
```

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 SmartFusion2 CMSIS HAL. You must provide your own implementation of the *NMI_Handler()* function, that will override the default implementation, to suit your application.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
#include "mss_watchdog.h"
int main( void )
{
    MSS_WD_init();
    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.

Parameters

This function has no parameters.

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.

Parameters

This function has no parameters.

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 SmartFusion2 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 SmartFusion2 CMSIS HAL. You must provide your own implementation of the *WdogWakeup_IRQHandler()* function, which will override the default implementation, to suit your application.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

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

  void WdogWakeup_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 SmartFusion2 wakeup interrupt.

Parameters

This function has no parameters.

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.

Parameters

This function has no parameters.

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

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group 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 North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world **408.643.6913**

Customer Technical Support Center

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

Technical Support

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

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at http://www.microsemi.com/soc/.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

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 soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

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

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

50200420-1/09.15