SmartFusion2 MSS Timer Driver User's Guide

Version 2.2

Table of Contents

Introduction

The SmartFusion2™ microcontroller subsystem (MSS) includes a timer hardware block that can be used as two independent 32-bit timers or as a single 64-bit timer in periodic or one-shot mode. This software driver provides a set of functions for controlling the MSS Timer 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 Timer driver provides support for the following features:

- Configuring the timer's operating mode
- Setting the timer's immediate load value
- Setting the timer's background load value
- Reading the timer's current value
- Enabling, disabling and clearing MSS Timer interrupts

The MSS Timer driver is provided as C source code.

Supported Hardware IP

The SmartFusion2 MSS Timer 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 Timer 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)
- Release notes

Driver Source Code

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

mss_timer.h

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

This file should be included in any C source file that uses the MSS Timer 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 Timer 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.

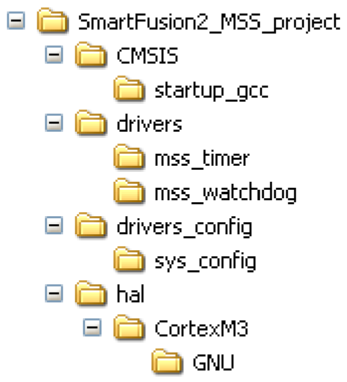

Figure 1 · SmartFusion2 MSS Project Example

Driver Configuration

The configuration of all features of the MSS Timer is covered by this driver. There are no dependencies on the hardware flow when configuring the MSS Timer. The counters in the MSS Timer are clocked by the PCLK (APB_0_CLK) on the SmartFusion2 MSS APB_0 bus.

The base address and register addresses and interrupt number assignment for the MSS Timer 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 Timer peripheral.

Theory of Operation

The MSS Timer can be used in one of two mutually exclusive modes; either as a single 64-bit timer or as two independent 32-bit timers. The MSS Timer can be used in either periodic mode or one-shot mode.

A timer configured for periodic mode operation will generate an interrupt and reload its down-counter when it reaches zero. The timer will then continue decrementing from its reload value without waiting for the interrupt to be cleared.

A timer configured for one-shot mode will only generate an interrupt once when its down-counter reaches zero. It must be explicitly reloaded to start decrementing again.

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

- Initialization and configuration
- Timer control
- Interrupt control

Initialization and Configuration

The MSS Timer driver provides three initialization functions:

- *MSS_TIM1_init()*
- *MSS_TIM2_init()*
- *MSS_TIM64_init()*

The MSS Timer driver is initialized through calls to these functions and at least one of them must be called before any other MSS Timer driver functions can be called.

Note: You should use only the *MSS_TIM1_init()* and *MSS_TIM2_init()* functions if you intend to use the MSS Timer in 32-bit mode. Use the *MSS_TIM64()* function if you intend to use the MSS Timer as a single 64-bit timer. The initialization functions take a single parameter specifying the operating mode of the timer being initialized.

Timer Control

Once initialized, a timer can be controlled using the following functions:

- *MSS_TIM1_load_immediate()*
- *MSS_TIM1_load_background()*
- *MSS_TIM1_get_current_value()*
- *MSS_TIM1_start()*
- *MSS_TIM1_stop()*
- *MSS_TIM2_load_immediate()*
- *MSS_TIM2_load_background()*
- *MSS_TIM2_get_current_value()*
- *MSS_TIM2_start()*
- *MSS_TIM2_stop()*
- *MSS_TIM64_load_immediate()*
- *MSS_TIM64_load_background()*

- *MSS_TIM64_get_current_value()*
- *MSS_TIM64_start()*
- *MSS_TIM64_stop()*

Interrupt Control

Timer interrupts are controlled using the following functions:

- *MSS_TIM1_enable_irq()*
- *MSS_TIM1_disable_irq()*
- *MSS_TIM1_clear_irq()*
- *MSS_TIM2_enable_irq()*
- *MSS_TIM2_disable_irq()*
- *MSS_TIM2_clear_irq()*
- *MSS_TIM64_enable_irq()*
- *MSS_TIM64_disable_irq()*
- *MSS_TIM64_clear_irq()*

The function prototypes for the timer interrupt handlers are as follows:

- *void Timer1_IRQHandler(void)*
- *void Timer2_IRQHandler(void)*

Entries for these interrupt handlers are provided in the SmartFusion2 CMSIS HAL vector table. To add a Timer 1 interrupt handler, you must implement a *Timer1_IRQHandler( )* function as part of your application code. To add a Timer 2 interrupt handler, you must implement a *Timer2_IRQHandler( )* function as part of your application code. When using the MSS Timer as a 64-bit timer, you must implement a *Timer1_IRQHandler( )* function as part of your application code. The Timer 2 interrupt is not used when the MSS Timer is configured as a 64-bit timer.

Types

timer_mode_t

Prototype

```
typedef enum __mss_timer_mode_t
{
    MSS_TIMER_PERIODIC_MODE = 0,
    MSS_TIMER_ONE_SHOT_MODE = 1
} mss_timer_mode_t;
```

Description

This enumeration is used to select between the two possible timer modes of operation: periodic and one-shot mode. It is used as an argument to the *MSS_TIM1_init()*, *MSS_TIM2_init()* and *MSS_TIM64_init()* functions.

MSS_TIMER_PERIODIC_MODE: In periodic mode the timer generates interrupts at constant intervals. On reaching zero, the timer's counter is reloaded with a value held in a register and begins counting down again.

MSS_TIMER_ONE_SHOT_MODE: The timer generates a single interrupt in this mode. On reaching zero, the timer's counter halts until reprogrammed by the user.

Constant Values

There are no MSS Timer driver specific constant values.

Data Structures

There are no MSS Timer driver-specific data structures.

Functions

MSS_TIM1_init

Prototype

```
void
MSS_TIM1_init
(
    mss_timer_mode_t mode
);
```

Description

The *MSS_TIM1_init()* function initializes the MSS Timer block for use as a 32-bit timer and selects the operating mode for Timer 1. This function takes the MSS Timer block out of reset in case this has not been done already, stops Timer 1, disables its interrupt, and sets the Timer 1 operating mode.

Note: The MSS Timer block cannot be used both as a 64-bit and 32-bit timer. Calling *MSS_TIM1_init()* will overwrite any previous configuration of the MSS Timer as a 64-bit timer.

Parameters

mode

The *mode* parameter specifies whether the timer will operate in periodic or one-shot mode. Allowed values for this parameter are as follows:

- MSS_TIMER_PERIODIC_MODE
- MSS_TIMER_ONE_SHOT_MODE

Return Value

This function does not return a value.

MSS_TIM1_load_immediate

Prototype

```
void
MSS_TIM1_load_immediate
(
    uint32_t load_value
);
```

Description

The *MSS_TIM1_load_immediate()* function loads the value passed by the *load_value* parameter into the Timer 1 down-counter. The counter will decrement immediately from this value once Timer 1 is enabled. The MSS Timer will generate an interrupt when the counter reaches zero, if Timer 1 interrupts are enabled. This function is intended to be used when Timer 1 is configured for one-shot mode to time a single delay.

Note: The value passed by the *load_value* parameter is loaded immediately into the down-counter regardless of whether Timer 1 is operating in periodic or one-shot mode.

Parameters

load_value

The *load_value* parameter specifies the value from which the Timer 1 down-counter will start decrementing.

Return Value

This function does not return a value.

MSS_TIM1_load_background

Prototype

```
void
MSS_TIM1_load_background
(
    uint32_t load_value
);
```

Description

The *MSS_TIM1_load_background()* function is used to specify the value that will be reloaded into the Timer 1 down-counter the next time the counter reaches zero. This function is typically used when Timer 1 is configured for periodic mode operation to select or change the delay period between the interrupts generated by Timer 1.

Parameters

load_value

The *load_value* parameter specifies the value that will be loaded into the Timer 1 down-counter the next time the down-counter reaches zero. The Timer 1 down-counter will start decrementing from this value after the current count expires.

Return Value

This function does not return a value.

MSS_TIM1_get_current_value

Prototype

```
uint32_t
MSS_TIM1_get_current_value
(
    void
);
```

Description

The *MSS_TIM1_get_current_value()* returns the current value of the Timer 1 down-counter.

Return Value

This function returns the 32-bit current value of the Timer 1 down-counter.

MSS_TIM1_start

Prototype

```
void
MSS_TIM1_start
(
    void
);
```

Description

The *MSS_TIM1_start()* function enables Timer 1 and starts its down-counter decrementing from the *load_value* specified in previous calls to the *MSS_TIM1_load_immediate()* or *MSS_TIM1_load_background()* functions.

Note: The *MSS_TIM1_start()* function is also used to resume the down-counter, if previously stopped using the *MSS_TIM1_stop()* function.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM1_stop

Prototype

```
void
MSS_TIM1_stop
(
    void
);
```

Description

The *MSS_TIM1_stop()* function disables Timer 1 and stops its down-counter decrementing.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM1_enable_irq

Prototype

```
void
MSS_TIM1_enable_irq
(
    void
);
```

Description

The *MSS_TIM1_enable_irq()* function is used to enable interrupt generation for Timer 1. This function also enables the interrupt in the Cortex-M3 interrupt controller. The *Timer1_IRQHandler()* function will be called when a Timer 1 interrupt occurs.

Note: A *Timer1_IRQHandler()* default implementation is defined, with weak linkage, in the SmartFusion2 CMSIS HAL. You must provide your own implementation of the *Timer1_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.

MSS_TIM1_disable_irq

Prototype

```
void
MSS_TIM1_disable_irq
(
    void
);
```

Description

The *MSS_TIM1_disable_irq()* function is used to disable interrupt generation for Timer 1. This function also disables the interrupt in the Cortex-M3 interrupt controller.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM1_clear_irq

Prototype

```
void
MSS_TIM1_clear_irq
(
    void
);
```

Description

The *MSS_TIM1_clear_irq()* function is used to clear a pending interrupt from Timer 1. This function also clears the interrupt in the Cortex-M3 interrupt controller.

Note: You must call the *MSS_TIM1_clear_irq()* function as part of your implementation of the *Timer1_IRQHandler()* Timer 1 interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the ISR.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM2_init

Prototype

```
void
MSS_TIM2_init
(
    mss_timer_mode_t mode
);
```

Description

The *MSS_TIM2_init()* function initializes the MSS Timer block for use as a 32-bit timer and selects the operating mode for Timer 2. This function takes the MSS Timer block out of reset in case this has not been done already, stops Timer 2, disables its interrupt and sets the Timer 2 operating mode.

Note: The MSS Timer block cannot be used both as a 64-bit and 32-bit timer. Calling *MSS_TIM2_init()* will overwrite any previous configuration of the MSS Timer as a 64-bit timer.

Parameters

mode

The *mode* parameter specifies whether the timer will operate in periodic or one-shot mode. Allowed values for this parameter are as follows:

- MSS_TIMER_PERIODIC_MODE
- MSS_TIMER_ONE_SHOT_MODE

Return Value

This function does not return a value.

MSS_TIM2_load_immediate

Prototype

```
void
MSS_TIM2_load_immediate
(
    uint32_t load_value
);
```

Description

The *MSS_TIM2_load_immediate()* function loads the value passed by the *load_value* parameter into the Timer 2 down-counter. The counter will decrement immediately from this value once Timer 2 is enabled. The MSS Timer will generate an interrupt when the counter reaches zero, if Timer 2 interrupts are enabled. This function is intended to be used when Timer 2 is configured for one-shot mode to time a single delay.

Note: The value passed by the *load_value* parameter is loaded immediately into the down-counter regardless of whether Timer 2 is operating in periodic or one-shot mode.

Parameters

load_value

The *load_value* parameter specifies the value from which the Timer 2 down-counter will start decrementing.

Return Value

This function does not return a value.

MSS_TIM2_load_background

Prototype

```
void
MSS_TIM2_load_background
(
    uint32_t load_value
);
```

Description

The *MSS_TIM2_load_background()* function is used to specify the value that will be reloaded into the Timer 2 down-counter the next time the counter reaches zero. This function is typically used when Timer 2 is configured for periodic mode operation to select or change the delay period between the interrupts generated by Timer 2.

Parameters

load_value

The *load_value* parameter specifies the value that will be loaded into the Timer 2 down-counter the next time the down-counter reaches zero. The Timer 2 down-counter will start decrementing from this value after the current count expires.

Return Value

This function does not return a value.

MSS_TIM2_get_current_value

Prototype

```
uint32_t
MSS_TIM2_get_current_value
(
    void
);
```

Description

The *MSS_TIM2_get_current_value()* returns the current value of the Timer 2 down-counter.

Return Value

This function returns the 32-bit current value of the Timer 2 down-counter.

MSS_TIM2_start

Prototype

```
void
MSS_TIM2_start
(
    void
);
```

Description

The *MSS_TIM2_start()* function enables Timer 2 and starts its down-counter decrementing from the *load_value* specified in previous calls to the *MSS_TIM2_load_immediate()* or *MSS_TIM2_load_background()* functions.

Note: The *MSS_TIM2_start()* function is also used to resume the down-counter, if previously stopped using the *MSS_TIM2_stop()* function.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM2_stop

Prototype

```
void
MSS_TIM2_stop
(
    void
);
```

Description

The *MSS_TIM2_stop()* function disables Timer 2 and stops its down-counter decrementing.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM2_enable_irq

Prototype

```
void
MSS_TIM2_enable_irq
(
    void
);
```

Description

The *MSS_TIM2_enable_irq()* function is used to enable interrupt generation for Timer 2. This function also enables the interrupt in the Cortex-M3 interrupt controller. The *Timer2_IRQHandler()* function will be called when a Timer 2 interrupt occurs.

Note: A *Timer2_IRQHandler()* default implementation is defined, with weak linkage, in the SmartFusion2 CMSIS HAL. You must provide your own implementation of the *Timer2_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.

MSS_TIM2_disable_irq

Prototype

```
void
MSS_TIM2_disable_irq
(
    void
);
```

Description

The *MSS_TIM2_disable_irq()* function is used to disable interrupt generation for Timer 2. This function also disables the interrupt in the Cortex-M3 interrupt controller.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM2_clear_irq

Prototype

```
void
MSS_TIM2_clear_irq
(
    void
);
```

Description

The *MSS_TIM2_clear_irq()* function is used to clear a pending interrupt from Timer 2. This function also clears the interrupt in the Cortex-M3 interrupt controller.

Note: You must call the *MSS_TIM2_clear_irq()* function as part of your implementation of the *Timer2_IRQHandler()* Timer 2 interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the ISR.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM64_init

Prototype

```
void
MSS_TIM64_init
(
    mss_timer_mode_t mode
);
```

Description

The *MSS_TIM64_init()* function initializes the MSS Timer block for use as a single 64-bit timer and selects the operating mode of the timer. This function takes the MSS Timer block out of reset in case this has not been done already, stops the timer, disables its interrupts, and sets the timer's operating mode.

Note: The MSS Timer block cannot be used both as a 64-bit and 32-bit timer. Calling *MSS_TIM64_init()* will overwrite any previous configuration of the MSS Timer as a 32-bit timer.

Parameters

mode

The *mode* parameter specifies whether the timer will operate in periodic or one-shot mode. Allowed values for this parameter are as follows:

- MSS_TIMER_PERIODIC_MODE
- MSS_TIMER_ONE_SHOT_MODE

Return Value

This function does not return a value.

MSS_TIM64_load_immediate

Prototype

```
void
MSS_TIM64_load_immediate
(
    uint32_t load_value_u,
    uint32_t load_value_l
);
```

Description

The *MSS_TIM64_load_immediate()* function loads the values passed by the *load_value_u* and *load_value_l* parameters into the 64-bit timer down-counter. The counter will decrement immediately from the concatenated 64-bit value once the 64-bit timer is enabled. The MSS Timer will generate an interrupt when the counter reaches zero, if 64-bit timer interrupts are enabled. This function is intended to be used when the 64-bit timer is configured for one-shot mode to time a single delay.

Note: The value passed by the *load_value* parameter is loaded immediately into the down-counter regardless of whether the 64-bit timer is operating in periodic or one-shot mode.

Parameters

load_value_u

The *load_value_u* parameter specifies the upper 32 bits of the 64-bit timer load value from which the 64-bit timer down-counter will start decrementing.

load_value_l

The *load_value_l* parameter specifies the lower 32 bits of the 64-bit timer load value from which the 64-bit timer down-counter will start decrementing.

Return Value

This function does not return a value.

MSS_TIM64_load_background

Prototype

```
void
MSS_TIM64_load_background
(
    uint32_t load_value_u,
    uint32_t load_value_l
);
```

Description

The *MSS_TIM64_load_background()* function is used to specify the 64-bit value that will be reloaded into the 64-bit timer down-counter the next time the counter reaches zero. This function is typically used when the 64-bit timer is configured for periodic mode operation to select or change the delay period between the interrupts generated by the 64-bit timer.

Parameters

load_value_u

The *load_value_u* parameter specifies the upper 32 bits of the 64-bit timer load value. The concatenated 64-bit value formed from *load_value_u and load_value_l* will be loaded into the 64-bit timer down-counter the next time the down-counter reaches zero. The 64-bit timer down-counter will start decrementing from the concatenated 64-bit value after the current count expires.

load_value_l

The *load_value_l* parameter specifies the lower 32 bits of the 64-bit timer load value. The concatenated 64-bit value formed from *load_value_u and load_value_l* will be loaded into the 64-bit timer down-counter the next time the down-counter reaches zero. The 64-bit timer down-counter will start decrementing from the concatenated 64-bit value after the current count expires.

Return Value

This function does not return a value.

MSS_TIM64_get_current_value

Prototype

```
void
MSS_TIM64_get_current_value
(
    uint32_t * load_value_u,
    uint32_t * load_value_l
);
```

Description

The *MSS_TIM64_get_current_value()* is used to read the current value of the 64-bit timer down-counter.

Parameters

load_value_u

The *load_value_u* parameter is a pointer to a 32-bit variable where the upper 32 bits of the current value of the 64-bit timer down-counter will be copied.

load_value_l

The *load_value_l* parameter is a pointer to a 32-bit variable where the lower 32 bits of the current value of the 64-bit timer down-counter will be copied.

Return Value

This function does not return a value.

Example

```
  uint32_t current_value_u = 0;
  uint32_t current_value_l = 0;
  MSS_TIM64_get_current_value( &current_value_u, &current_value_l );
```

MSS_TIM64_start

Prototype

```
void
MSS_TIM64_start
(
    void
);
```

Description

The *MSS_TIM64_start()* function enables the 64-bit timer and starts its down-counter decrementing from the *load_value* specified in previous calls to the *MSS_TIM64_load_immediate()* or *MSS_TIM64_load_background()* functions.

Note: The *MSS_TIM64_start()* function is also used to resume the down-counter, if previously stopped using the *MSS_TIM64_stop()* function.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM64_stop

Prototype

```
void
MSS_TIM64_stop
(
    void
);
```

Description

The *MSS_TIM64_stop()* function disables the 64-bit timer and stops its down-counter decrementing.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM64_enable_irq

Prototype

```
void
MSS_TIM64_enable_irq
(
    void
);
```

Description

The *MSS_TIM64_enable_irq()* function is used to enable interrupt generation for the 64-bit timer. This function also enables the interrupt in the Cortex-M3 interrupt controller. The *Timer1_IRQHandler()* function will be called when a 64-bit timer interrupt occurs.

Note: A *Timer1_IRQHandler()* default implementation is defined, with weak linkage, in the SmartFusion2 CMSIS HAL. You must provide your own implementation of the *Timer1_IRQHandler()* function, which will override the default implementation, to suit your application.

Note: The *MSS_TIM64_enable_irq()* function enables and uses Timer 1 interrupts for the 64-bit timer. Timer 2 interrupts remain disabled.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM64_disable_irq

Prototype

```
void
MSS_TIM64_disable_irq
(
    void
);
```

Description

The *MSS_TIM64_disable_irq()* function is used to disable interrupt generation for the 64-bit timer. This function also disables the interrupt in the Cortex-M3 interrupt controller.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

MSS_TIM64_clear_irq

Prototype

```
void
MSS_TIM64_clear_irq
(
    void
);
```

Description

The *MSS_TIM64_clear_irq()* function is used to clear a pending interrupt from the 64-bit timer. This function also clears the interrupt in the Cortex-M3 interrupt controller.

Note: You must call the *MSS_TIM64_clear_irq()* function as part of your implementation of the *Timer1_IRQHandler()* 64-bit timer interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the ISR.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

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.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

50200418-2/09.15