SmartFusion2 MSS GPIO Driver User's Guide

Version 2.1

Microsemi

Table of Contents

Introduction

The SmartFusion2™ microcontroller subsystem (MSS) includes a block of 32 general purpose input/outputs (GPIO). This software driver provides a set of functions for controlling the MSS GPIO block 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 GPIO driver provides support for the following features:

- Configuring the operating modes of each individual GPIO port
- Setting and reading the state of the GPIO outputs
- Reading the state of the GPIO inputs
- Enabling, disabling and clearing GPIO interrupts

The MSS GPIO driver is provided as C source code.

Supported Hardware IP

The SmartFusion2 MSS GPIO 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 GPIO 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_gpio* subdirectory of the selected software project directory. The files making up the driver are detailed below.

mss_gpio.h

This header file contains the public application programming interface (API) of the MSS GPIO software driver. This file should be included in any C source file that uses the MSS GPIO software driver.

mss_gpio.c

This C source file contains the implementation of the MSS GPIO 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.

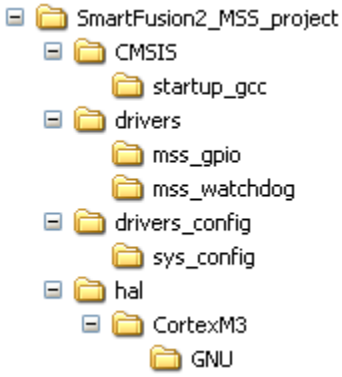

Figure 1 · SmartFusion2 MSS Project Example

Driver Configuration

The configuration of all features of the MSS GPIOs is covered by this driver with the exception of the SmartFusion2 IOMUX configuration. SmartFusion2 allows multiple non-concurrent uses of some external pins through IOMUX configuration. This feature allows optimization of external pin usage by assigning external pins for use by either the microcontroller subsystem or the FPGA fabric. The MSS GPIOs share SmartFusion2 device external pins with the FPGA fabric and with other MSS peripherals via an IOMUX. The MSS GPIO ports can alternatively be routed to the FPGA fabric through an IOMUX.

The IOMUXs are configured using the SmartFusion2 MSS configurator tool. You must ensure that the MSS GPIOs are enabled and configured in the SmartFusion2 MSS configurator if you wish to use them. For more information on IOMUXs, refer to the IOMUX section of the *SmartFusion2 Microcontroller Subsystem (MSS) User's Guide*.

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

Theory of Operation

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

- Initialization
- Configuration
- Reading and setting GPIO state
- Interrupt control

Initialization

The MSS GPIO driver is initialized through a call to the *MSS_GPIO_init()* function. The *MSS_GPIO_init()* function must be called before any other MSS GPIO driver functions can be called.

Configuration

Each GPIO port is individually configured through a call to the *MSS_GPIO_config()* function. Configuration includes deciding if a GPIO port will be used as an input, an output or both. GPIO ports configured as inputs can be further configured to generate interrupts based on the input's state. Interrupts can be level or edge sensitive.

Reading and Setting GPIO State

The state of the GPIO ports can be read and set using the following functions:

- *MSS_GPIO_get_inputs()*
- *MSS_GPIO_get_outputs()*
- *MSS_GPIO_set_outputs()*
- *MSS_GPIO_set_output()*
- *MSS_GPIO_drive_inout()*

Interrupt Control

Interrupts generated by GPIO ports configured as inputs are controlled using the following functions:

- *MSS_GPIO_enable_irq()*
- *MSS_GPIO_disable_irq()*
- *MSS_GPIO_clear_irq()*

Types

mss_gpio_id_t

Prototype

```
typedef enum __mss_gpio_id_t {
    MSS_GPIO_0 = 0,
    MSS_GPIO_1 = 1,
    MSS_GPIO_2 = 2,
    MSS_GPIO_3 = 3,
    MSS_GPIO_4 = 4,
    MSS_GPIO_5 = 5,
    MSS_GPIO_6 = 6,
    MSS_GPIO_7 = 7,
    MSS_GPIO_8 = 8,
    MSS_GPIO_9 = 9,
    MSS_GPIO_10 = 10,
    MSS_GPIO_11 = 11,
    MSS_GPIO_12 = 12,
    MSS_GPIO_13 = 13,
    MSS_GPIO_14 = 14,
    MSS_GPIO_15 = 15,
    MSS_GPIO_16 = 16,
    MSS_GPIO_17 = 17,
    MSS_GPIO_18 = 18,
    MSS_GPIO_19 = 19,
    MSS_GPIO_20 = 20,
    MSS_GPIO_21 = 21,
    MSS_GPIO_22 = 22,
    MSS_GPIO_23 = 23,
    MSS_GPIO_24 = 24,
    MSS_GPIO_25 = 25,
    MSS_GPIO_26 = 26,
    MSS_GPIO_27 = 27,
    MSS_GPIO_28 = 28,
    MSS_GPIO_29 = 29,
    MSS_GPIO_30 = 30,
    MSS_GPIO_31 = 31
} mss_gpio_id_t;
```

Description

The *mss_gpio_id_t* enumeration is used to identify individual GPIO ports as an argument to functions:

- *MSS_GPIO_config()*
- *MSS_GPIO_set_output()* and *MSS_GPIO_drive_inout()*
- *MSS_GPIO_enable_irq()*, *MSS_GPIO_disable_irq()* and *MSS_GPIO_clear_irq()*

mss_gpio_inout_state_t

Prototype

```
typedef enum mss_gpio_inout_state {
```

```
    MSS_GPIO_DRIVE_LOW = 0,
    MSS_GPIO_DRIVE_HIGH,
    MSS_GPIO_HIGH_Z
} mss_gpio_inout_state_t;
```

Description

The *mss_gpio_inout_state_t* enumeration is used to specify the output state of an INOUT GPIO port as an argument to the *MSS_GPIO_drive_inout()* function.

Constant Values

GPIO Port Masks

These constant definitions are used as an argument to the *MSS_GPIO_set_outputs()* function to identify GPIO ports. A logical OR of these constants can be used to specify multiple GPIO ports.

These definitions can also be used to identify GPIO ports through logical operations on the return value of the *MSS_GPIO_get_inputs()* function.

Table 1 · GPIO Port Mask Constants

Constant	Description
MSS_GPIO_0_MASK	GPIO port 0-bit mask
MSS_GPIO_1_MASK	GPIO port 1-bit mask
MSS_GPIO_2_MASK	GPIO port 2-bit mask
MSS_GPIO_3_MASK	GPIO port 3-bit mask
MSS_GPIO_4_MASK	GPIO port 4-bit mask
MSS_GPIO_5_MASK	GPIO port 5-bit mask
MSS_GPIO_6_MASK	GPIO port 6-bit mask
MSS_GPIO_7_MASK	GPIO port 7-bit mask
MSS_GPIO_8_MASK	GPIO port 8-bit mask
MSS_GPIO_9_MASK	GPIO port 9-bit mask
MSS_GPIO_10_MASK	GPIO port 10-bit mask
MSS_GPIO_11_MASK	GPIO port 11-bit mask
MSS_GPIO_12_MASK	GPIO port 12-bit mask
MSS_GPIO_13_MASK	GPIO port 13-bit mask
MSS_GPIO_14_MASK	GPIO port 14-bit mask
MSS_GPIO_15_MASK	GPIO port 15-bit mask

Constant	Description
MSS_GPIO_16_MASK	GPIO port 16-bit mask
MSS_GPIO_17_MASK	GPIO port 17-bit mask
MSS_GPIO_18_MASK	GPIO port 18-bit mask
MSS_GPIO_19_MASK	GPIO port 19-bit mask
MSS_GPIO_20_MASK	GPIO port 20-bit mask
MSS_GPIO_21_MASK	GPIO port 21-bit mask
MSS_GPIO_22_MASK	GPIO port 22-bit mask
MSS_GPIO_23_MASK	GPIO port 23-bit mask
MSS_GPIO_24_MASK	GPIO port 24-bit mask
MSS_GPIO_25_MASK	GPIO port 25-bit mask
MSS_GPIO_26_MASK	GPIO port 26-bit mask
MSS_GPIO_27_MASK	GPIO port 27-bit mask
MSS_GPIO_28_MASK	GPIO port 28-bit mask
MSS_GPIO_29_MASK	GPIO port 29-bit mask
MSS_GPIO_30_MASK	GPIO port 30-bit mask
MSS_GPIO_31_MASK	GPIO port 31-bit mask

GPIO Port I/O Mode

These constant definitions are used as an argument to the *MSS_GPIO_config()* function to specify the I/O mode of each GPIO port.

Table 2 · GPIO Port I/O Mode

Constant	Description
MSS_GPIO_INPUT_MODE	Input port only
MSS_GPIO_OUTPUT_MODE	Output port only
MSS_GPIO_INOUT_MODE	Both input and output port

GPIO Interrupt Mode

These constant definitions are used as an argument to the *MSS_GPIO_config()* function to specify the interrupt mode of each GPIO port.

Table 3 · GPIO Interrupt Mode

Constant	Description
MSS_GPIO_IRQ_LEVEL_HIGH	Interrupt on GPIO input level High
MSS_GPIO_IRQ_LEVEL_LOW	Interrupt on GPIO input level Low
MSS_GPIO_IRQ_EDGE_POSITIVE	Interrupt on GPIO input positive edge
MSS_GPIO_IRQ_EDGE_NEGATIVE	Interrupt on GPIO input negative edge
MSS_GPIO_IRQ_EDGE_BOTH	Interrupt on GPIO input positive and negative edges

Data Structures

There are no MSS GPIO driver specific data structures.

Functions

MSS_GPIO_init

Prototype

```
void
MSS_GPIO_init
(
    void
);
```

Description

The *MSS_GPIO_init()* function initializes the SmartFusion2 MSS GPIO block. It resets the MSS GPIO hardware block and it also clears any pending MSS GPIO interrupts in the ARM® Cortex™-M3 interrupt controller. When the function exits, it takes the MSS GPIO block out of reset.

Parameters

This function has no parameters.

Return Value

This function does not return a value.

Example

```
MSS_GPIO_init();
```

MSS_GPIO_config

Prototype

```
void
MSS_GPIO_config
(
    mss_gpio_id_t port_id,
    uint32_t config
);
```

Description

The *MSS_GPIO_config()* function is used to configure an individual GPIO port.

Parameters

port_id

The *port_id* parameter identifies the GPIO port to be configured. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

config

The *config* parameter specifies the configuration to be applied to the GPIO port identified by the *port_id* parameter. It is a logical OR of the required I/O mode and the required interrupt mode. The interrupt mode is not relevant if the GPIO is configured as an output only.

These I/O mode constants are allowed:

- MSS_GPIO_INPUT_MODE
- MSS_GPIO_OUTPUT_MODE
- MSS_GPIO_INOUT_MODE

These interrupt mode constants are allowed:

- MSS_GPIO_IRQ_LEVEL_HIGH
- MSS_GPIO_IRQ_LEVEL_LOW
- MSS_GPIO_IRQ_EDGE_POSITIVE
- MSS_GPIO_IRQ_EDGE_NEGATIVE
- MSS_GPIO_IRQ_EDGE_BOTH

Return Value

This function does not return a value.

Example

The following call will configure GPIO 4 as an input generating interrupts on a Low to High transition of the input:

```
MSS_GPIO_config( MSS_GPIO_4, MSS_GPIO_INPUT_MODE | MSS_GPIO_IRQ_EDGE_POSITIVE );
```

MSS_GPIO_set_outputs

Prototype

```
void
MSS_GPIO_set_outputs
(
    uint32_t value
);
```

Description

The *MSS_GPIO_set_outputs()* function is used to set the state of all GPIO ports configured as outputs.

Parameters

value

The value parameter specifies the state of the GPIO ports configured as outputs. It is a bit mask of the form (MSS_GPIO_n_MASK | MSS_GPIO_m_MASK) where *n* and *m* are numbers identifying GPIOs. For example, (MSS_GPIO_0_MASK | MSS_GPIO_1_MASK | MSS_GPIO_2_MASK) specifies that the first, second and third GPIO outputs must be set High and all other GPIO outputs set Low. The driver provides 32 mask constants, MSS_GPIO_0_MASK to MSS_GPIO_31_MASK inclusive, for this purpose.

Return Value

This function does not return a value.

Example

Example 1: Set GPIO outputs 0 and 8 High and all other GPIO outputs Low.

```
MSS_GPIO_set_outputs( MSS_GPIO_0_MASK | MSS_GPIO_8_MASK );
```

Example 2: Set GPIO outputs 2 and 4 Low without affecting other GPIO outputs.

```
uint32_t gpio_outputs;
gpio_outputs = MSS_GPIO_get_outputs();
gpio_outputs &= ~( MSS_GPIO_2_MASK | MSS_GPIO_4_MASK );
MSS_GPIO_set_outputs( gpio_outputs );
```

MSS_GPIO_set_output

Prototype

```
void
MSS_GPIO_set_output
(
    mss_gpio_id_t port_id,
    uint8_t value
);
```

Description

The *MSS_GPIO_set_output()* function is used to set the state of a single GPIO port configured as an output.

Note: Using bit-band writes might be a better option than this function for performance critical applications where the application code is not intended to be ported to a processor other than the ARM® Cortex™-M3 in SmartFusion2. The bit-band write equivalent to this function would be:
GPIO_BITBAND->GPIO_OUT[port_id] = (uint32_t)value;

Parameters

port_id

The *port_id* parameter identifies the GPIO port that is to have its output set. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

value

The value parameter specifies the desired state for the GPIO output. A value of 0 will set the output Low and a value of 1 will set the output High.

Return Value

This function does not return a value.

Example

The following call will set GPIO output 12 High, leaving all other GPIO outputs unaffected:

```
MSS_GPIO_set_output( MSS_GPIO_12, 1 );
```

MSS_GPIO_get_outputs

Prototype

```
uint32_t
MSS_GPIO_get_outputs
(
    void

);
```

Description

The *MSS_GPIO_get_outputs()* function is used to read the current state all GPIO ports configured as outputs.

Parameters

This function has no parameters.

Return Value

This function returns a 32-bit unsigned integer where each bit represents the state of a GPIO output. The least significant bit represents the state of GPIO output 0 and the most significant bit the state of GPIO output 31.

Example

Read and assign the current state of the GPIO outputs to a variable.

```
uint32_t gpio_outputs;
gpio_outputs = MSS_GPIO_get_outputs();
```

MSS_GPIO_get_inputs

Prototype

```
uint32_t
MSS_GPIO_get_inputs
(
    void

);
```

Description

The *MSS_GPIO_get_inputs()* function is used to read the current state all GPIO ports configured as inputs.

Parameters

This function has no parameters.

Return Value

This function returns a 32-bit unsigned integer where each bit represents the state of a GPIO input. The least significant bit represents the state of GPIO input 0 and the most significant bit the state of GPIO input 31.

Example

Read and assign the current state of the GPIO inputs to a variable.

```
uint32_t gpio_inputs;
gpio_inputs = MSS_GPIO_get_inputs();
```

MSS_GPIO_drive_inout

Prototype

```
void
MSS_GPIO_drive_inout
(
    mss_gpio_id_t port_id,
    mss_gpio_inout_state_t inout_state
);
```

Description

The *MSS_GPIO_drive_inout()* function is used to set the output state of a single GPIO port configured as an INOUT. An INOUT GPIO can be in one of three states:

- High
- Low
- High impedance

An INOUT output would typically be used where several devices can drive the state of a shared signal line. The High and Low states are equivalent to the High and Low states of a GPIO configured as an output. The High impedance state is used to prevent the GPIO from driving its output state onto the signal line, while at the same time allowing the input state of the GPIO to be read.

Parameters

port_id

The *port_id* parameter identifies the GPIO port for which you want to change the output state. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

inout_state

The *inout_state* parameter specifies the state of the GPIO port identified by the *port_id* parameter. Allowed values of type *mss_gpio_inout_state_t* are as follows:

- MSS_GPIO_DRIVE_HIGH
- MSS_GPIO_DRIVE_LOW
- MSS_GPIO_HIGH_Z (High impedance)

Return Value

This function does not return a value.

Example

The call to *MSS_GPIO_drive_inout()* below will set the GPIO 7 output to the High impedance state.

```
MSS_GPIO_drive_inout( MSS_GPIO_7, MSS_GPIO_HIGH_Z );
```

MSS_GPIO_enable_irq

Prototype

```
void
MSS_GPIO_enable_irq
(
    mss_gpio_id_t port_id

);
```

Description

The *MSS_GPIO_enable_irq()* function is used to enable interrupt generation for the specified GPIO input. Interrupts are generated based on the state of the GPIO input and the interrupt mode configured for it by *MSS_GPIO_config()*.

Parameters

port_id

The *port_id* parameter identifies the GPIO port for which you want to enable interrupt generation. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

Return Value

This function does not return a value.

Example

The call to *MSS_GPIO_enable_irq()* below will allow GPIO 8 to generate interrupts.

```
MSS_GPIO_enable_irq( MSS_GPIO_8 );
```

MSS_GPIO_disable_irq

Prototype

```
void
MSS_GPIO_disable_irq
(
    mss_gpio_id_t port_id

);
```

Description

The *MSS_GPIO_disable_irq()* function is used to disable interrupt generation for the specified GPIO input.

Parameters

port_id

The *port_id* parameter identifies the GPIO port for which you want to disable interrupt generation. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

Return Value

This function does not return a value.

Example

The call to *MSS_GPIO_disable_irq()* below will prevent GPIO 8 from generating interrupts.

```
MSS_GPIO_disable_irq( MSS_GPIO_8 );
```

MSS_GPIO_clear_irq

Prototype

```
void
MSS_GPIO_clear_irq
(
    mss_gpio_id_t port_id

);
```

Description

The *MSS_GPIO_clear_irq()* function is used to clear a pending interrupt from the specified GPIO input.

Note: The *MSS_GPIO_clear_irq()* function must be called as part of any GPIO interrupt service routine (ISR) in order to prevent the same interrupt event retriggering a call to the GPIO ISR.

Parameters

port_id

The *port_id* parameter identifies the GPIO port for which you want to clear the interrupt. An enumeration item of the form MSS_GPIO_n, where *n* is the number of the GPIO port, is used to identify the GPIO port. For example, MSS_GPIO_0 identifies the first GPIO port and MSS_GPIO_31 is the last one.

Return Value

This function does not return a value.

Example

The example below demonstrates the use of the *MSS_GPIO_clear_irq()* function as part of the GPIO 9 interrupt service routine.

```
void GPIO9_IRQHandler( void )
{
    do_interrupt_processing();
    MSS_GPIO_clear_irq( MSS_GPIO_9 );
}
```

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

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.

50200410-1/10.15