SmartFusion2 MSS MMUART Driver User's Guide

Version 2.1

Microsemi

Table of Contents

Introduction

The SmartFusion2™ microcontroller subsystem (MSS) includes two multi-mode UART (MMUART) peripherals for serial communication. This driver provides a set of functions for controlling the MSS MMUARTs as part of a bare metal system where no operating system is available. These drivers 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.

Note: MSS UART is synonymous with MSS MMUART in this document.

Features

The SmartFusion2 MSS MMUART driver provides the following features:

- Support for configuring each MMUART block
- Support for polled transmit and receive
- Support for interrupt driven transmit and receive

The MSS MMUART driver is provided as C source code.

Supported Hardware IP

The MSS MMUART 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 UART 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_uart* subdirectory of the selected software project directory. The files making up the driver are detailed below.

mss_uart.h

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

mss_uart_regs.h

This header file contains the definitions for register bit offsets.

mss_uart.c

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

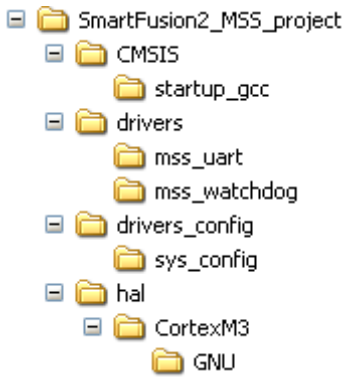

Figure 1 · SmartFusion2 MSS Project Example

Driver Configuration

The configuration of all features of the MSS MMUART peripherals 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 MMUART serial signals are routed through IOMUXs to the SmartFusion2 device external pins. The MSS MMUART serial signals may also be routed through IOMUXs to the SmartFusion2 FPGA fabric. For more information on IOMUX, refer to the IOMUX section of the *SmartFusion2 Microcontroller Subsystem (MSS) User's Guide*.

The IOMUXs are configured using the SmartFusion2 MSS configurator tool. You must ensure that the MSS MMUART peripherals 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 MMUART peripherals 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 MMUART peripheral.

Theory of Operation

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

- Initialization and configuration functions
- Polled transmit and receive functions
- Interrupt driven transmit and receive functions

Initialization and Configuration

The MSS MMUART supports the following four broad modes of operation:

- UART or USART mode
- LIN mode
- IrDA mode
- Smartcard or ISO 7816 mode

The MSS MMUART driver provides the *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* and *MSS_UART_smartcard_init()* functions to initialize the MSS MMUARTs for operation in one of these modes. One of these initialization functions must be called before any other MSS MMUART driver functions can be called. The MSS MMUART operating modes are mutually exclusive; therefore only one of the initialization functions must be called. The first parameter of the initialization functions is a pointer to one of two global data structures used to store state information for each MSS MMUART. A pointer to these data structures is also used as the first parameter to many of the driver functions to identify which MSS MMUART will be used by the called function. The names of these two data structures are *g_mss_uart0* and *g_mss_uart1*. Therefore, any call to an MSS MMUART function should be of the form *MSS_UART_function_name(&g_mss_uart0, ...)* or *MSS_UART_function_name(&g_mss_uart1, ...)*.

UART or USART Mode

For the UART or USART modes of operation, the MSS MMUART driver is initialized through a call to the *MSS_UART_init()* function. This function takes the UART's configuration as its parameters. The *MSS_UART_init()* function must be called before any other MSS MMUART driver functions can be called. The *MSS_UART_init()* function configures the baud rate based on the input baud rate parameter and if possible uses a fractional baud rate for greater precision. This function disables the LIN, IrDA and SmartCard modes.

LIN mode

For the LIN mode of operation, the MSS MMUART driver is initialized through a call to the *MSS_UART_lin_init()* function. This function takes the LIN node's configuration as its parameters. The *MSS_UART_lin_init()* function must be called before any other MSS MMUART driver functions can be called. The *MSS_UART_lin_init()* function configures the baud rate based on the input baud rate parameter and if possible uses a fractional baud rate for greater precision. This function disables the IrDA and SmartCard modes.

The driver also provides the following LIN mode configuration functions:

- *MSS_UART_set_break()*
- *MSS_UART_clear_break()*
- *MSS_UART_set_pidpei_handler()*

- *MSS_UART_set_linbreak_handler()*
- *MSS_UART_set_linsync_handler()*

Note: These LIN mode configuration functions can only be called after the *MSS_UART_lin_init()* function is called.

IrDA mode

For the IrDA mode of operation, the driver is initialized through a call to the *MSS_UART_irda_init()* function. This function takes the IrDA node's configuration as its parameters. The *MSS_UART_irda_init()* function must be called before any other MSS MMUART driver functions can be called. The *MSS_UART_irda_init()* function configures the baud rate based on the input baud rate parameter and if possible uses a fractional baud rate for greater precision. This function disables the LIN and SmartCard modes.

Smartcard or ISO 7816 mode

For the Smartcard or ISO 7816 mode of operation, the driver is initialized through a call to the MSS_UART_smartcard_init() function. This function takes the smartcard configuration as its parameters. The MSS_UART_smartcard_init() function must be called before any other MSS MMUART driver functions can be called. The *MSS_UART_smartcard_init()* function configures the baud rate based on the input baud rate parameter and if possible uses a fractional baud rate for greater precision. This function disables the LIN and IrDA modes.

The driver also provides the following Smartcard mode configuration functions:

- MSS_UART_enable_halfduplex()
- MSS_UART_disable_halfduplex()
- MSS_UART_set_nack_handler()

Note: These Smartcard mode configuration functions can only be called after the *MSS_UART_smartcard_init()* function is called.

Common Configuration Functions

The driver also provides the configuration functions that can be used with all MSS MMUART operating modes. These common configuration functions are as follows:

- *MSS_UART_set_rx_endian()*
- *MSS_UART_set_tx_endian()*
- *MSS_UART_enable_afclear()*
- *MSS_UART_disable_afclear()*
- *MSS_UART_enable_rx_timeout()*
- *MSS_UART_disable_rx_timeout()*
- *MSS_UART_enable_tx_time_guard()*
- *MSS_UART_disable_tx_time_guard()*
- *MSS_UART_set_address()*
- *MSS_UART_set_ready_mode()*
- *MSS_UART_set_usart_mode()*
- *MSS_UART_set_filter_length()*
- *MSS_UART_enable_afm()*
- *MSS_UART_disable_afm()*

Note: These configuration functions can only be called after one of the *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* or *MSS_UART_smartcard_init()* functions is called.

Polled Transmit and Receive Operations

The driver can be used to transmit and receive data once initialized.

Data is transmitted using the *MSS_UART_polled_tx()* function. This function is blocking, meaning that it will only return once the data passed to the function has been sent to the MSS MMUART hardware transmitter. Data received by the MSS MMUART hardware receiver can be read by the *MSS_UART_get_rx()* function.

The *MSS_UART_polled_tx_string()* function is provided to transmit a NULL ('\0') terminated string in polled mode. This function is blocking, meaning that it will only return once the data passed to the function has been sent to the MSS MMUART hardware transmitter.

The *MSS_UART_fill_tx_fifo()* function fills the MSS MMUART hardware transmit FIFO with data from a buffer passed as a parameter and returns the number of bytes transferred to the FIFO. If the transmit FIFO is not empty when the *MSS_UART_fill_tx_fifo()* function is called it returns immediately without transferring any data to the FIFO.

Interrupt Driven Operations

The driver can also transmit or receive data under interrupt control, freeing your application to perform other tasks until an interrupt occurs indicating that the driver's attention is required.

Interrupt Handlers

The MSS MMUART driver supports all types of interrupt triggered by the MSS MMUART. The driver's internal top level interrupt handler identifies the source of the MSS MMUART interrupt and calls the corresponding lower level handler function that you previously registered with the driver through calls to the *MSS_UART_set_rx_handler()*, *MSS_UART_set_tx_handler()*, *MSS_UART_set_rxstatus_handler()*, and *MSS_UART_set_modemstatus_handler()* functions. You are responsible for creating these lower level interrupt handlers as part of your application program and registering them with the driver.

Note: The SmartFusion2 CMSIS-PAL defines the *UART0_IRQHandler()* and *UART1_IRQHandler()* functions (with weak linkage) and assigns them as the interrupt service routines (ISR) for the MSS MMUART interrupt inputs to the Cortex-M3 NVIC. The MSS MMUART driver provides the implementation functions for both of these ISRs from which it calls its own internal top level, interrupt handler function.

The *MSS_UART_enable_irq()* and *MSS_UART_disable_irq()* functions are used to enable or disable the received line status, received data available/character timeout, transmit holding register empty and modem status interrupts at the MSS MMUART level. The *MSS_UART_enable_irq()* function also enables the MSS MMUART instance interrupt at the Cortex-M3 level.

Transmitting Data

Interrupt-driven transmit is initiated by a call to *MSS_UART_irq_tx()*, specifying the block of data to transmit. Your application is then free to perform other tasks and inquire later whether transmit has completed by calling the *MSS_UART_tx_complete()* function. The *MSS_UART_irq_tx()* function enables the UART's transmit holding register empty (THRE) interrupt and then, when the interrupt goes active, the driver's default THRE interrupt handler transfers the data block to the UART until the entire block is transmitted.

Note: You can use the *MSS_UART_set_tx_handler()* function to assign an alternative handler to the THRE interrupt. In this case, you must not use the *MSS_UART_irq_tx()* function to initiate the transmit, as this will re-assign the driver's default THRE interrupt handler to the THRE interrupt. Instead, your alternative THRE interrupt handler must include a call to the *MSS_UART_fill_tx_fifo()* function to transfer the data to the UART.

Receiving Data

Interrupt-driven receive is performed by first calling *MSS_UART_set_rx_handler()* to register a receive handler function that will be called by the driver whenever receive data is available. You must provide this receive handler function which must include a call to the *MSS_UART_get_rx()* function to actually read the received data.

UART Status

The function *MSS_UART_get_rx_status()* is used to read the receiver error status. This function returns the overrun, parity, framing, break, and FIFO error status of the receiver.

The function *MSS_UART_get_tx_status()* is used to read the transmitter status. This function returns the transmit empty (TEMT) and transmit holding register empty (THRE) status of the transmitter.

The function *MSS_UART_get_modem_status()* is used to read the modem status flags. This function returns the current value of the modem status register.

Loopback

The *MSS_UART_set_loopback()* function can be used to locally loopback the Tx and Rx lines of a UART. This is not to be confused with the loopback of UART0 to UART1, which can be achieved through the microcontroller subsystem's system registers.

Types

mss_uart_rx_trig_level_t

Prototype

```
typedef enum mss_uart_rx_trig_level_t {
    MSS_UART_FIFO_SINGLE_BYTE    = 0x00,
    MSS_UART_FIFO_FOUR_BYTES     = 0x40,
    MSS_UART_FIFO_EIGHT_BYTES    = 0x80,
    MSS_UART_FIFO_FOURTEEN_BYTES = 0xC0,
    MSS_UART_FIFO_INVALID_TRIG_LEVEL
} mss_uart_rx_trig_level_t;
```

Description

This enumeration specifies the receiver FIFO trigger level. This is the number of bytes that must be received before the UART generates a receive data available interrupt. It provides the allowed values for the *MSS_UART_set_rx_handler()* function *trigger_level* parameter.

mss_uart_loopback_t

Prototype

```
typedef enum {
    MSS_UART_LOCAL_LOOPBACK_OFF,
    MSS_UART_LOCAL_LOOPBACK_ON,
    MSS_UART_REMOTE_LOOPBACK_OFF,
    MSS_UART_REMOTE_LOOPBACK_ON,
    MSS_UART_AUTO_ECHO_OFF,
    MSS_UART_AUTO_ECHO_ON,
    MSS_UART_INVALID_LOOPBACK
} mss_uart_loopback_t;
```

Description

This enumeration specifies the loopback configuration of the UART. It provides the allowed values for the *MSS_UART_set_loopback()* function's *loopback* parameter. Use MSS_UART_LOCAL_LOOPBACK_ON to set up the UART to locally loopback its Tx and Rx lines. Use MSS_UART_REMOTE_LOOPBACK_ON to set up the UART in remote loopback mode.

mss_uart_irq_handler_t

Prototype

```
typedef void(* mss_uart_irq_handler_t)(mss_uart_instance_t * this_uart);
```

Description

This typedef specifies the function prototype for MSS UART interrupt handlers. All interrupt handlers registered with the MSS UART driver must be of this type. The interrupt handlers are registered with the driver through the *MSS_UART_set_rx_handler()*, *MSS_UART_set_tx_handler()*, *MSS_UART_set_rxstatus_handler()*, and *MSS_UART_set_modemstatus_handler()* functions.

The *this_uart* parameter is a pointer to either *g_mss_uart0* or *g_mss_uart1* to identify the MSS UART to associate with the handler function.

mss_uart_irq_t

Prototype

```
typedef uint16_t mss_uart_irq_t;
```

Description

This typedef specifies the *irq_mask* parameter for the *MSS_UART_enable_irq()* and *MSS_UART_disable_irq()* functions. The driver defines a set of bit masks that are used to build the value of the *irq_mask* parameter. A bitwise OR of these bit masks is used to enable or disable multiple MSS MMUART interrupts.

mss_uart_rzi_polarity_t

Prototype

```
typedef enum {
    MSS_UART_ACTIVE_LOW = 0u,
    MSS_UART_ACTIVE_HIGH = 1u,
    MSS_UART_INVALID_POLARITY
}mss_uart_rzi_polarity_t;
```

Description

This enumeration specifies the RZI modem polarity for input and output signals. This is passed as parameters in *MSS_UART_irda_init()* function.

mss_uart_rzi_pulsewidth_t

Prototype

```
typedef enum {
    MSS_UART_3_BY_16 = 0u,
    MSS_UART_1_BY_4 = 1u,
    MSS_UART_INVALID_PW
} mss_uart_rzi_pulsewidth_t;
```

Description

This enumeration specifies the RZI modem pulse width for input and output signals. This is passed as parameters in *MSS_UART_irda_init()* function.

mss_uart_endian_t

Prototype

```
typedef enum {
    MSS_UART_LITTLEEND,
    MSS_UART_BIGEND,
    MSS_UART_INVALID_ENDIAN
}mss_uart_endian_t;
```

Description

This enumeration specifies the MSB first or LSB first for MSS UART transmitter and receiver. The parameter of this type shall be passed in *MSS_UART_set_rx_endian()and MSS_UART_set_tx_endian()* functions.

mss_uart_filter_length_t

Prototype

```
typedef enum {
    MSS_UART_LEN0,
    MSS_UART_LEN1,
    MSS_UART_LEN2,
    MSS_UART_LEN3,
    MSS_UART_LEN4,
    MSS_UART_LEN5,
    MSS_UART_LEN6,
    MSS_UART_LEN7,
    MSS_UART_INVALID_FILTER_LENGTH
}mss_uart_filter_length_t;
```

Description

This enumeration specifies the glitch filter length. The function *MSS_UART_set_filter_length()* accepts the parameter of this type.

mss_uart_ready_mode_t

Prototype

```
typedef enum {
    MSS_UART_READY_MODE0,
    MSS_UART_READY_MODE1,
    MSS_UART_INVALID_READY_MODE
}mss_uart_ready_mode_t;
```

Description

This enumeration specifies the TXRDY and RXRDY signal modes. The function MSS*_UART_set_ready_mode()* accepts the parameter of this type.

mss_uart_usart_mode_t

Prototype

```
typedef enum {
    MSS_UART_ASYNC_MODE,
    MSS_UART_SYNC_SLAVE_POS_EDGE_CLK,
    MSS_UART_SYNC_SLAVE_NEG_EDGE_CLK,
    MSS_UART_SYNC_MASTER_POS_EDGE_CLK,
    MSS_UART_SYNC_MASTER_NEG_EDGE_CLK,
    MSS_UART_INVALID_SYNC_MODE
}mss_uart_usart_mode_t;
```

Description

This enumeration specifies the mode of operation of MSS UART when operating as USART. The function MSS*_UART_set_usart_mode()* accepts the parameter of this type.

Constant Values

Baud Rates

The following definitions are used to specify standard baud rates as a parameter to the *MSS_UART_init(), MSS_UART_lin_init(), MSS_UART_irda_init() and MSS_UART_smartcard_init()* functions.

Table 1 · Standard Baud Rates

Constant	Description
MSS_UART_110_BAUD	110 baud rate
MSS_UART_300_BAUD	300 baud rate
MSS_UART_1200_BAUD	1200 baud rate
MSS_UART_2400_BAUD	2400 baud rate
MSS_UART_4800_BAUD	4800 baud rate
MSS_UART_9600_BAUD	9600 baud rate
MSS_UART_19200_BAUD	19200 baud rate
MSS_UART_38400_BAUD	38400 baud rate
MSS_UART_57600_BAUD	57600 baud rate
MSS_UART_115200_BAUD	115200 baud rate
MSS_UART_230400_BAUD	230400 baud rate
MSS_UART_460800_BAUD	460800 baud rate
MSS_UART_921600_BAUD	921600 baud rate

Data Bits Length

The following defines are used to build the value of the *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* and *MSS_UART_smartcard_init()* functions' *line_config* parameter.

Table 2 · Data Bits Length Configuration Options

Constant	Description
MSS_UART_DATA_5_BITS	5 bits of data transmitted
MSS_UART_DATA_6_BITS	6 bits of data transmitted
MSS_UART_DATA_7_BITS	7 bits of data transmitted
MSS_UART_DATA_8_BITS	8 bits of data transmitted

Parity

The following defines are used to build the value of the *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* and *MSS_UART_smartcard_init()* functions' *line_config* parameter.

Table 3 · Parity Configuration Options

Constant	Description
MSS_UART_NO_PARITY	No parity
MSS_UART_ODD_PARITY	Odd Parity
MSS_UART_EVEN_PARITY	Even parity
MSS_UART_STICK_PARITY_0	Stick parity bit to zero
MSS_UART_STICK_PARITY_1	Stick parity bit to one

Number of Stop Bits

The following defines are used to build the value of the *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* and *MSS_UART_smartcard_init()* functions' *line_config* parameter.

Table 4 · Stop Bit Length Configuration Options

Constant	Description
MSS_UART_ONE_STOP_BIT	One stop bit
MSS_UART_ONEHALF_STOP_BIT	One and a half stop bits
MSS_UART_TWO_STOP_BITS	Two stop bits

Receiver Error Status

The following defines are used to determine the UART receiver error type. These bit mask constants are used with the return value of the *MSS_UART_get_rx_status()* function to find out if any errors occurred while receiving data.

Table 5 · Receiver Error Status Mask Constants

Constant	Description
MSS_UART_NO_ERROR	No error bit mask (0x00)
MSS_UART_OVERUN_ERROR	Overrun error bit mask (0x02)
MSS_UART_PARITY_ERROR	Parity error bit mask (0x04)
MSS_UART_FRAMING_ERROR	Framing error bit mask (0x08)
MSS_UART_BREAK_ERROR	Break error bit mask (0x10)
MSS_UART_FIFO_ERROR	FIFO error bit mask (0x80)
MSS_UART_INVALID_PARAM	Invalid function parameter bit mask (0xFF)

Transmitter Status

The following definitions are used to determine the UART transmitter status. These bit mask constants are used with the return value of the *MSS_UART_get_tx_status()* function to find out the status of the transmitter.

Table 6 · Transmitter Status Mask Constants

Constant	Description
MSS_UART_TX_BUSY	Transmitter busy (0x00)
MSS_UART_TEMT	Transmitter empty bit mask (0x40)
MSS_UART_THRE	Transmitter holding register empty bit mask (0x20)

Modem Status

The following defines are used to determine the modem status. These bit mask constants are used with the return value of the *MSS_UART_get_modem_status()* function to find out the modem status of the UART.

Table 7 · Modem Status Mask Constants

Constant	Description
MSS_UART_DCTS	Delta clear to send bit mask (0x01)
MSS_UART_DDSR	Delta data set ready bit mask (0x02)
MSS_UART_TERI	Trailing edge of ring indicator bit mask (0x04)
MSS_UART_DDCD	Delta data carrier detect bit mask (0x08)
MSS_UART_CTS	Clear to send bit mask (0x10)
MSS_UART_DSR	Data set ready bit mask (0x20)
MSS_UART_RI	Ring indicator bit mask (0x40)
MSS_UART_DCD	Data carrier detect bit mask (0x80)

MSS MMUART Interrupts

The following defines specify the interrupt masks to enable and disable MSS MMUART interrupts. They are used to build the value of the *irq_mask* parameter for the *MSS_UART_enable_irq()* and *MSS_UART_disable_irq()* functions. A bitwise OR of these constants is used to enable or disable multiple interrupts.

Table 8 · MSS MMUART interrupt bit masks

Constant	Description
MSS_UART_RBF_IRQ	Receive Data Available Interrupt bit mask (0x001)
MSS_UART_TBE_IRQ	Transmitter Holding Register Empty interrupt bit mask (0x002)
MSS_UART_LS_IRQ	Receiver Line Status interrupt bit mask (0x004)
MSS_UART_MS_IRQ	Modem Status interrupt bit mask (0x008)
MSS_UART_RTO_IRQ	Receiver time-out interrupt bit mask (0x010)
MSS_UART_NACK_IRQ	NACK / ERR signal interrupt bit mask (0x020)
MSS_UART_PIDPE_IRQ	PID parity error interrupt bit mask (0x040)
MSS_UART_LINB_IRQ	LIN break detection interrupt bit mask (0x080)
MSS_UART_LINS_IRQ	LIN Sync detection interrupt bit mask (0x100)

Data Structures

mss_uart_instance_t

There is one instance of this structure for each instance of the microcontroller subsystem's UARTs. Instances of this structure are used to identify a specific UART. A pointer to an initialized instance of the *mss_uart_instance_t* structure is passed as the first parameter to MSS UART driver functions to identify which UART should perform the requested operation.

Global Variables

g_mss_uart0

Prototype

```
mss_uart_instance_t g_mss_uart0;
```

Description

This instance of *mss_uart_instance_t* holds all data related to the operations performed by UART0. Depending on mode operation, one of the functions *MSS_UART_init(), MSS_UART_lin_init() MSS_UART_irda_init() or MSS_UART_smartcard_init()* initializes this structure. A pointer to *g_mss_uart0* is passed as the first parameter to MSS UART driver functions to indicate that UART0 should perform the requested operation.

g_mss_uart1

Prototype

```
mss_uart_instance_t g_mss_uart1;
```

Description

This instance of *mss_uart_instance_t* holds all data related to the operations performed by UART1. Depending on mode operation, one of the functions *MSS_UART_init()*, *MSS_UART_lin_init()*, *MSS_UART_irda_init()* or *MSS_UART_smartcard_init()* initializes this structure. A pointer to *g_mss_uart1* is passed as the first parameter to MSS UART driver functions to indicate that UART1 should perform the requested operation.

Functions

MSS_UART_init

Prototype

```
void MSS_UART_init
(
    mss_uart_instance_t * this_uart,
    uint32_t baud_rate,
    uint8_t line_config
);
```

Description

The *MSS_UART_init()* function initializes and configures one of the SmartFusion2 MSS UARTs with the configuration passed as a parameter. The configuration parameters are the *baud_rate* which is used to generate the baud value and the *line_config* which is used to specify the line configuration (bit length, stop bits and parity).

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block to be initialized. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1 respectively. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

baud_rate

The *baud_rate* parameter specifies the baud rate. It can be specified for common baud rates' using the following defines:

- MSS_UART_110_BAUD
- MSS_UART_300_BAUD
- MSS_UART_1200_BAUD
- MSS_UART_2400_BAUD
- MSS_UART_4800_BAUD
- MSS_UART_9600_BAUD
- MSS_UART_19200_BAUD
- MSS_UART_38400_BAUD
- MSS_UART_57600_BAUD
- MSS_UART_115200_BAUD
- MSS_UART_230400_BAUD
- MSS_UART_460800_BAUD
- MSS_UART_921600_BAUD

Alternatively, any nonstandard baud rate can be specified by simply passing the actual required baud rate as the value for this parameter.

line_config

The *line_config* parameter is the line configuration specifying the bit length, number of stop bits and parity settings. This is a bitwise OR of one value from each of the following groups of allowed values:

- One of the following to specify the transmit/receive data bit length:

  MSS_UART_DATA_5_BITS

  MSS_UART_DATA_6_BITS,

MSS_UART_DATA_7_BITS
MSS_UART_DATA_8_BITS

- One of the following to specify the parity setting:
  MSS_UART_NO_PARITY
  MSS_UART_EVEN_PARITY
  MSS_UART_ODD_PARITY
  MSS_UART_STICK_PARITY_0
  MSS_UART_STICK_PARITY_1
- One of the following to specify the number of stop bits:
  MSS_UART_ONE_STOP_BIT
  MSS_UART_ONEHALF_STOP_BIT
  MSS_UART_TWO_STOP_BITS

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"
int main(void)
{
    MSS_UART_init
    (
        &g_mss_uart0,
        MSS_UART_57600_BAUD,
        MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
    );
    return(0);
}
```

MSS_UART_lin_init

Prototype

```
void
MSS_UART_lin_init
(
    mss_uart_instance_t* this_uart,
    uint32_t baud_rate,
    uint8_t line_config
);
```

Description

The *MSS_UART_lin_init()* function is used to initialize the MSS UART for LIN mode of operation. The configuration parameters are the *baud_rate* which is used to generate the baud value and the *line_config* which is used to specify the line configuration (bit length, stop bits and parity).

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

baud_rate

The *baud_rate* parameter specifies the baud rate. It can be specified for common baud rates' using the following defines:

- MSS_UART_110_BAUD
- MSS_UART_300_BAUD
- MSS_UART_1200_BAUD
- MSS_UART_2400_BAUD
- MSS_UART_4800_BAUD
- MSS_UART_9600_BAUD
- MSS_UART_19200_BAUD
- MSS_UART_38400_BAUD
- MSS_UART_57600_BAUD
- MSS_UART_115200_BAUD
- MSS_UART_230400_BAUD
- MSS_UART_460800_BAUD
- MSS_UART_921600_BAUD

Alternatively, any nonstandard baud rate can be specified by simply passing the actual required baud rate as the value for this parameter.

line_config

The *line_config* parameter is the line configuration specifying the bit length, number of stop bits and parity settings. This is a bitwise OR of one value from each of the following groups of allowed values:

- One of the following to specify the transmit/receive data bit length:
  MSS_UART_DATA_5_BITS
  MSS_UART_DATA_6_BITS,
  MSS_UART_DATA_7_BITS
  MSS_UART_DATA_8_BITS
- One of the following to specify the parity setting:
  MSS_UART_NO_PARITY
  MSS_UART_EVEN_PARITY
  MSS_UART_ODD_PARITY
  MSS_UART_STICK_PARITY_0
  MSS_UART_STICK_PARITY_1
- One of the following to specify the number of stop bits:
  MSS_UART_ONE_STOP_BIT
  MSS_UART_ONEHALF_STOP_BIT

MSS_UART_TWO_STOP_BITS

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

  int main(void)
  {
     MSS_UART_lin_init
     (
         &g_mss_uart0,
```

```
        MSS_UART_57600_BAUD,
        MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
    );
    return(0);
}
```

MSS_UART_irda_init

Prototype

```
void
MSS_UART_irda_init
(
    mss_uart_instance_t* this_uart,
    uint32_t baud_rate,
    uint8_t line_config,
    mss_uart_rzi_polarity_t rxpol,
    mss_uart_rzi_polarity_t txpol,
    mss_uart_rzi_pulsewidth_t pw
);
```

Description

The *MSS_UART_irda_init()* function is used to initialize the MSS UART instance referenced by the parameter *this_uart* for IrDA mode of operation. This function must be called before calling any other IrDA functionality specific functions.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

baud_rate

The *baud_rate* parameter specifies the baud rate. It can be specified for common baud rates' using the following defines:

- MSS_UART_110_BAUD
- MSS_UART_300_BAUD
- MSS_UART_1200_BAUD
- MSS_UART_2400_BAUD
- MSS_UART_4800_BAUD
- MSS_UART_9600_BAUD
- MSS_UART_19200_BAUD
- MSS_UART_38400_BAUD
- MSS_UART_57600_BAUD
- MSS_UART_115200_BAUD
- MSS_UART_230400_BAUD
- MSS_UART_460800_BAUD
- MSS_UART_921600_BAUD

Alternatively, any nonstandard baud rate can be specified by simply passing the actual required baud rate as the value for this parameter.

line_config

The *line_config* parameter is the line configuration specifying the bit length, number of stop bits and parity settings. This is a bitwise OR of one value from each of the following groups of allowed values:

- One of the following to specify the transmit/receive data bit length:
  MSS_UART_DATA_5_BITS
  MSS_UART_DATA_6_BITS,
  MSS_UART_DATA_7_BITS
  MSS_UART_DATA_8_BITS
- One of the following to specify the parity setting:
  MSS_UART_NO_PARITY
  MSS_UART_EVEN_PARITY
  MSS_UART_ODD_PARITY
  MSS_UART_STICK_PARITY_0
  MSS_UART_STICK_PARITY_1
- One of the following to specify the number of stop bits:
  MSS_UART_ONE_STOP_BIT
  MSS_UART_ONEHALF_STOP_BIT
  MSS_UART_TWO_STOP_BITS

rxpol

The *rxpol* parameter is of type *mss_uart_rzi_polarity_t*. This parameter is used to configure the receiver polarity for the IrDA modem.

txpol

The *txpol* parameter is of type *mss_uart_rzi_polarity_t*. This parameter is used to configure the transmitter polarity for the IrDA modem.

pw

The *pw* parameter is of type *mss_uart_rzi_pulsewidth_t*. This parameter is used to configure the IrDA modem pulse width.

Return Value

This function does not return a value.

Example

```
MSS_UART_irda_init( &g_mss_uart0,
                    MSS_UART_57600_BAUD,
                    MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT,
                    MSS_UART_ACTIVE_LOW,
                    MSS_UART_ACTIVE_LOW,
                    MSS_UART_3_BY_16);
```

MSS_UART_smartcard_init

Prototype

```
void
MSS_UART_smartcard_init
(
    mss_uart_instance_t* this_uart,
    uint32_t baud_rate,
    uint8_t line_config
```

```
);
```

Description

The *MSS_UART_smartcard_init()* function is used to initialize the MSS UART for ISO 7816 (smartcard) mode of operation. The configuration parameters are the *baud_rate* which is used to generate the baud value and the *line_config* which is used to specify the line configuration (bit length, stop bits and parity). This function disables all other modes of the MSS UART instance pointed by the parameter *this_uart*.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

baud_rate

The *baud_rate* parameter specifies the baud rate. It can be specified for common baud rates' using the following defines:

- MSS_UART_110_BAUD
- MSS_UART_300_BAUD
- MSS_UART_1200_BAUD
- MSS_UART_2400_BAUD
- MSS_UART_4800_BAUD
- MSS_UART_9600_BAUD
- MSS_UART_19200_BAUD
- MSS_UART_38400_BAUD
- MSS_UART_57600_BAUD
- MSS_UART_115200_BAUD
- MSS_UART_230400_BAUD
- MSS_UART_460800_BAUD
- MSS_UART_921600_BAUD

Alternatively, any nonstandard baud rate can be specified by simply passing the actual required baud rate as the value for this parameter.

line_config

The *line_config* parameter is the line configuration specifying the bit length, number of stop bits and parity settings. This is a bitwise OR of one value from each of the following groups of allowed values:

- One of the following to specify the transmit/receive data bit length:
  MSS_UART_DATA_5_BITS
  MSS_UART_DATA_6_BITS,
  MSS_UART_DATA_7_BITS
  MSS_UART_DATA_8_BITS
- One of the following to specify the parity setting:
  MSS_UART_NO_PARITY
  MSS_UART_EVEN_PARITY
  MSS_UART_ODD_PARITY
  MSS_UART_STICK_PARITY_0
  MSS_UART_STICK_PARITY_1
- One of the following to specify the number of stop bits:
  MSS_UART_ONE_STOP_BIT

MSS_UART_ONEHALF_STOP_BIT
MSS_UART_TWO_STOP_BITS

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

  int main(void)
  {
     MSS_UART_smartcard_init
     (
         &g_mss_uart0,
         MSS_UART_57600_BAUD,
         MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
     );
     return(0);
  }
```

MSS_UART_polled_tx

Prototype

```
void MSS_UART_polled_tx
(
    mss_uart_instance_t * this_uart,
    const uint8_t * pbuff,
    uint32_t tx_size
);
```

Description

The function *MSS_UART_polled_tx()* is used to transmit data. It transfers the contents of the transmitter data buffer, passed as a function parameter, into the UART's hardware transmitter FIFO. It returns when the full content of the transmit data buffer has been transferred to the UART's transmit FIFO. It is safe to release or reuse the memory used as the transmitter data buffer once this function returns.

Note: This function reads the UART's line status register (LSR) to poll for the active state of the transmitter holding register empty (THRE) bit before transferring data from the data buffer to the transmitter FIFO. It transfers data to the transmitter FIFO in blocks of 16 bytes or less and allows the FIFO to empty before transferring the next block of data.

Note: The actual transmission over the serial connection will still be in progress when this function returns. Use the *MSS_UART_get_tx_status()* function if you need to know when the transmitter is empty.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

pbuff

The *pbuff* parameter is a pointer to a buffer containing the data to be transmitted.

tx_size

The *tx_size* parameter specifies the size, in bytes, of the data to be transmitted.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"
int main(void)
{
    uint8_t message[12] = "Hello World";

    MSS_UART_init( &g_mss_uart0, MSS_UART_57600_BAUD,
                   MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT );
    MSS_UART_polled_tx( &g_mss_uart0, message, sizeof(message) );
    return(0);
}
```

MSS_UART_polled_tx_string

Prototype

```
void MSS_UART_polled_tx_string
(
    mss_uart_instance_t * this_uart,
    const uint8_t * p_sz_string
);
```

Description

The function *MSS_UART_polled_tx_string()* is used to transmit a NULL ('\0') terminated string. It transfers the text string, from the buffer starting at the address pointed to by *p_sz_string* into the UART's hardware transmitter FIFO. It returns when the complete string has been transferred to the UART's transmit FIFO. It is safe to release or reuse the memory used as the string buffer once this function returns.

Note: This function reads the UART's line status register (LSR) to poll for the active state of the transmitter holding register empty (THRE) bit before transferring data from the data buffer to the transmitter FIFO. It transfers data to the transmitter FIFO in blocks of 16 bytes or less and allows the FIFO to empty before transferring the next block of data.

Note: The actual transmission over the serial connection will still be in progress when this function returns. Use the *MSS_UART_get_tx_status()* function if you need to know when the transmitter is empty.

Parameters

this_uart

The this_uart parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

p_sz_string

The *p_sz_string* parameter is a pointer to a buffer containing the NULL ('\0') terminated string to be transmitted.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"
int main(void)
{
    uint8_t message[12] = "Hello World";

    MSS_UART_init( &g_mss_uart0, MSS_UART_57600_BAUD,
                   MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT );
    MSS_UART_polled_tx_string( &g_mss_uart0, message );
    return(0);
}
```

MSS_UART_fill_tx_fifo

Prototype

```
size_t
MSS_UART_fill_tx_fifo
(
    mss_uart_instance_t * this_uart,
    const uint8_t * tx_buffer,
    size_t tx_size
);
```

Description

The *MSS_UART_fill_tx_fifo()* function fills the UART's hardware transmitter FIFO with the data found in the transmitter buffer that is passed via the *tx_buffer* function parameter. If the transmitter FIFO is not empty when the function is called, the function returns immediately without transferring any data to the FIFO; otherwise, the function transfers data from the transmitter buffer to the FIFO until it is full or until the complete contents of the transmitter buffer have been copied into the FIFO. The function returns the number of bytes copied into the UART's transmitter FIFO.

Note: This function reads the UART's line status register (LSR) to check for the active state of the transmitter holding register empty (THRE) bit before transferring data from the data buffer to the transmitter FIFO. If THRE is 0, the function returns immediately, without transferring any data to the FIFO. If THRE is 1, the function transfers up to 16 bytes of data to the FIFO and then returns.

Note: The actual transmission over the serial connection will still be in progress when this function returns. Use the *MSS_UART_get_tx_status()* function if you need to know when the transmitter is empty.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

tx_buffer

The *tx_buffer* parameter is a pointer to a buffer containing the data to be transmitted.

tx_size

The *tx_size* parameter is the size in bytes, of the data to be transmitted.

Return Value

This function returns the number of bytes copied into the UART's transmitter FIFO.

Example

```
void send_using_interrupt( uint8_t * pbuff, size_t tx_size )
{
    size_t size_in_fifo;
    size_in_fifo = MSS_UART_fill_tx_fifo( &g_mss_uart0, pbuff, tx_size );
}
```

MSS_UART_get_rx

Prototype

```
size_t MSS_UART_get_rx
(
    mss_uart_instance_t * this_uart,
    uint8_t * rx_buff,
    size_t buff_size
);
```

Description

The *MSS_UART_get_rx()* function reads the content of the UART receiver's FIFO and stores it in the receive buffer that is passed via the *rx_buff* function parameter. It copies either the full contents of the FIFO into the receive buffer, or just enough data from the FIFO to fill the receive buffer, dependent upon the size of the receive buffer passed by the *buff_size* parameter. The *MSS_UART_get_rx()* function returns the number of bytes copied into the receive buffer .This function is non-blocking and will return 0 immediately if no data has been received.

Note: The *MSS_UART_get_rx()* function reads and accumulates the receiver status of the MSS UART instance before reading each byte from the receiver's data register/FIFO. This allows the driver to maintain a sticky record of any receiver errors that occur as the UART receives each data byte; receiver errors would otherwise be lost after each read from the receiver's data register. A call to the *MSS_UART_get_rx_status()* function returns any receiver errors accumulated during the execution of the *MSS_UART_get_rx()* function.

Note: If you need to read the error status for each byte received, set the *buff_size* to 1 and read the receive line error status for each byte using the *MSS_UART_get_rx_status()* function.

The *MSS_UART_get_rx()* function can be used in polled mode, where it is called at regular intervals to find out if any data has been received, or in interrupt driven-mode, where it is called as part of a receive handler that is called by the driver as a result of data being received.

Note: In interrupt driven mode you should call the *MSS_UART_get_rx()* function as part of the receive handler function that you register with the MSS UART driver through a call to *MSS_UART_set_rx_handler()*.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure, identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

rx_buff

The *rx_buff* parameter is a pointer to a buffer where the received data is copied.

buff_size

The *buff_size* parameter specifies the size of the receive buffer in bytes.

Return Value

This function returns the number of bytes that were copied into the *rx_buff* buffer. It returns 0 if no data has been received.

Example1: Polled Mode Example

```
int main( void )
{
    uint8_t rx_buff[RX_BUFF_SIZE];
    uint32_t rx_idx  = 0;
```

```
    MSS_UART_init
         (
            &g_mss_uart0,
            MSS_UART_57600_BAUD,
            MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
        );
    while( 1 )
    {
        rx_size = MSS_UART_get_rx( &g_mss_uart0, rx_buff, sizeof(rx_buff) );
        if (rx_size > 0)
        {
            process_rx_data( rx_buff, rx_size );
        }
        task_a();
        task_b();
    }
    return 0;
}
```

Example2: Interrupt-Driven Example

```
int main( void )
{
    MSS_UART_init
        (
            &g_mss_uart1,
            MSS_UART_57600_BAUD,
            MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
        );
    MSS_UART_set_rx_handler( &g_mss_uart1, uart1_rx_handler,
                                MSS_UART_FIFO_SINGLE_BYTE );
    while( 1 )
    {
        task_a();
        task_b();
    }
    return 0;
}

void uart1_rx_handler( mss_uart_instance_t * this_uart )
{
    uint8_t rx_buff[RX_BUFF_SIZE];
    uint32_t rx_idx  = 0;
    rx_size = MSS_UART_get_rx( this_uart, rx_buff, sizeof(rx_buff) );
    process_rx_data( rx_buff, rx_size );
}
```

MSS_UART_irq_tx

Prototype

```
void MSS_UART_irq_tx
(
    mss_uart_instance_t * this_uart,
    const uint8_t * pbuff,
    uint32_t tx_size
);
```

Description

The function *MSS_UART_irq_tx()* is used to initiate an interrupt-driven transmit. It returns immediately after making a note of the transmit buffer location and enabling transmit interrupts both at the UART and Cortex-M3 NVIC level. This function takes a pointer via the *pbuff* parameter to a memory buffer containing the data to transmit. The memory buffer specified through this pointer must remain allocated and contain the data to transmit until the transmit completion has been detected through calls to function *MSS_UART_tx_complete()*. The actual transmission over the serial connection is still in progress until calls to the *MSS_UART_tx_complete()* function indicate transmit completion.

Note: The *MSS_UART_irq_tx()* function enables both the transmit holding register empty (THRE) interrupt in the UART and the MSS UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

Note: The *MSS_UART_irq_tx()* function assigns an internal default transmit interrupt handler function to the UART's THRE interrupt. This interrupt handler overrides any custom interrupt handler that you may have previously registered using the *MSS_UART_set_tx_handler()* function.

Note: The *MSS_UART_irq_tx()* function's default transmit interrupt handler disables the UART's THRE interrupt when all of the data has been transferred to the UART's transmit FIFO.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

pbuff

The *pbuff* parameter is a pointer to a buffer containing the data to be transmitted.

tx_size

The *tx_size* parameter specifies the size, in bytes, of the data to be transmitted.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"
int main(void)
{
    uint8_t tx_buff[10] = "abcdefghi";
    MSS_UART_init
    (
        &g_mss_uart0,
        MSS_UART_57600_BAUD,
        MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
    );
```

```
    MSS_UART_irq_tx( &g_mss_uart0, tx_buff, sizeof(tx_buff));
    while ( 0 == MSS_UART_tx_complete( &g_mss_uart0 ) )
    {
        ;
    }
    return(0);
}
```

MSS_UART_tx_complete

Prototype

```
int8_t MSS_UART_tx_complete
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_tx_complete()* function is used to find out if the interrupt-driven transmit previously initiated through a call to MSS_*UART_irq_tx()* is complete. This is typically used to find out when it is safe to reuse or release the memory buffer holding transmit data.

Note: The transfer of all of the data from the memory buffer to the UART's transmit FIFO and the actual transmission over the serial connection are both complete when a call to the *MSS_UART_tx_complete()* function indicates transmit completion.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function return a non-zero value if transmit has completed, otherwise it returns zero.

Example

See the *MSS_UART_irq_tx()* function for an example that uses the *MSS_UART_tx_complete()* function.

MSS_UART_set_tx_handler

Prototype

```
void MSS_UART_set_tx_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_tx_handler()* function is used to register a transmit handler function that is called by the driver when a UART transmit holding register empty (THRE) interrupt occurs. You must create and register the transmit handler function to suit your application. You can use the *MSS_UART_fill_tx_fifo()* function in your transmit handler function to write data to the transmitter.

Note: The *MSS_UART_set_tx_handler()* function enables both the THRE interrupt in the UART and the MSS UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

Note: You can disable the THRE interrupt when required by calling the *MSS_UART_disable_irq()* function. This is your choice and is dependent upon your application.

Note: The *MSS_UART_irq_tx()* function does not use the transmit handler function that you register with the *MSS_UART_set_tx_handler()* function. It uses its own internal THRE interrupt handler function that overrides any custom interrupt handler that you register using the *MSS_UART_set_tx_handler()* function.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

handler

The *handler* parameter is a pointer to a transmit interrupt handler function provided by your application that will be called as a result of a UART THRE interrupt. This handler function must be of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

uint8_t * g_tx_buffer;
size_t g_tx_size = 0;

    void uart_tx_handler( mss_uart_instance_t * this_uart )
    {
        size_t size_in_fifo;
        size_in_fifo = MSS_UART_fill_tx_fifo( this_uart,
                                              (const uint8_t *)g_tx_buffer,
                                              g_tx_size );
        if ( size_in_fifo  ==  g_tx_size )
        {
            g_tx_size = 0;
```

```
        MSS_UART_disable_irq( this_uart, MSS_UART_TBE_IRQ );
    }
    else
    {
        g_tx_buffer = &g_tx_buffer[size_in_fifo];
        g_tx_size = g_tx_size - size_in_fifo;
    }
}

int main(void)
{
    uint8_t message[12] = "Hello world";

    MSS_UART_init( &g_mss_uart0,
                   MSS_UART_57600_BAUD,
                   MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY |
                                          MSS_UART_ONE_STOP_BIT );

    g_tx_buffer = message;
    g_tx_size = sizeof(message);

    MSS_UART_set_tx_handler( &g_mss_uart0, uart_tx_handler);

    while ( 1 )
    {
        ;
    }
    return(0);
}
```

MSS_UART_set_rx_handler

Prototype

```
void MSS_UART_set_rx_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler,
    mss_uart_rx_trig_level_t trigger_level
);
```

Description

The *MSS_UART_set_rx_handler()* function is used to register a receive handler function that is called by the driver when a UART receive data available (RDA) interrupt occurs. You must create and register the receive handler function to suit your application and it must include a call to the *MSS_UART_get_rx()* function to actually read the received data.

Note: The *MSS_UART_set_rx_handler()* function enables both the RDA interrupt in the UART and the MSS UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

Note: You can disable the RDA interrupt once the data is received by calling the *MSS_UART_disable_irq()* function. This is your choice and is dependent upon your application.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

handler

The *handler* parameter is a pointer to a receive interrupt handler function provided by your application that will be called as a result of a UART RDA interrupt. This handler function must be of type *mss_uart_irq_handler_t*.

trigger_level

The *trigger_level* parameter is the receive FIFO trigger level. This specifies the number of bytes that must be received before the UART triggers an RDA interrupt.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

#define RX_BUFF_SIZE    64

uint8_t g_rx_buff[RX_BUFF_SIZE];

void uart0_rx_handler( mss_uart_instance_t * this_uart )
{
    MSS_UART_get_rx( this_uart, &g_rx_buff[g_rx_idx], sizeof(g_rx_buff) );
}

int main(void)
{
```

```
    MSS_UART_init
        (
            &g_mss_uart0,
            MSS_UART_57600_BAUD,
            MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT
        );
    MSS_UART_set_rx_handler( &g_mss_uart0,
                             uart0_rx_handler,
                             MSS_UART_FIFO_SINGLE_BYTE );

    while ( 1 )
    {
       ;
    }
    return(0);
}
```

MSS_UART_set_rxstatus_handler

Prototype

```
void MSS_UART_set_rxstatus_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_rxstatus_handler()* function is used to register a receiver status handler function that is called by the driver when a UART receiver line status (RLS) interrupt occurs. You must create and register the handler function to suit your application.

Note: The *MSS_UART_set_rxstatus_handler()* function enables both the RLS interrupt in the UART and the MSS UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

Note: You can disable the RLS interrupt when required by calling the *MSS_UART_disable_irq()* function. This is your choice and is dependent upon your application.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

handler

The *handler* parameter is a pointer to a receiver line status interrupt handler function provided by your application that will be called as a result of a UART RLS interrupt. This handler function must be of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

void uart_rxsts_handler( mss_uart_instance_t * this_uart )
```

```
{
    uint8_t status;
    status = MSS_UART_get_rx_status( this_uart );
    if( status & MSS_UART_OVERUN_ERROR )
    {
        discard_rx_data();
    }
}

int main(void)
{
    MSS_UART_init( &g_mss_uart0,
                   MSS_UART_57600_BAUD,
                   MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY |
                                          MSS_UART_ONE_STOP_BIT );
    MSS_UART_set_rxstatus_handler( &g_mss_uart0, uart_rxsts_handler);

    while ( 1 )
    {
        ;
    }
    return(0);
}
```

MSS_UART_set_modemstatus_handler

Prototype

```
void MSS_UART_set_modemstatus_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_modemstatus_handler()* function is used to register a modem status handler function that is called by the driver when a UART modem status (MS) interrupt occurs. You must create and register the handler function to suit your application.

Note: The *MSS_UART_set_modemstatus_handler()* function enables both the MS interrupt in the UART and the MSS UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

Note: You can disable the MS interrupt when required by calling the *MSS_UART_disable_irq()* function. This is your choice and is dependent upon your application.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

handler

The *handler* parameter is a pointer to a modem status interrupt handler function provided by your application that will be called as a result of a UART MS interrupt. This handler function must be of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
#include "mss_uart.h"

void uart_modem_handler( mss_uart_instance_t * this_uart )
{
    uint8_t status;
    status = MSS_UART_get_modem_status( this_uart );
    if( status & MSS_UART_CTS )
    {
        uart_cts_handler();
    }
}

int main(void)
{
    MSS_UART_init( &g_mss_uart0,
                   MSS_UART_57600_BAUD,
                   MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY |
                                          MSS_UART_ONE_STOP_BIT );
    MSS_UART_set_modemstatus_handler( &g_mss_uart0, uart_modem_handler);

    while ( 1 )
    {
        ;
    }
    return(0);
}
```

MSS_UART_enable_irq

Prototype

```
void MSS_UART_enable_irq
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_t irq_mask
);
```

Description

The *MSS_UART_enable_irq()* function enables the MSS UART interrupts specified by the *irq_mask* parameter. The *irq_mask* parameter identifies the MSS UART interrupts by bit position, as defined in the interrupt enable register (IER) of MSS UART. The MSS UART interrupts and their identifying *irq_mask* bit positions are as follows:

When an *irq_mask* bit position is set to 1, this function enables the corresponding MSS UART interrupt in the IER register. When an *irq_mask* bit position is set to 0, the corresponding interrupt's state remains unchanged in the IER register.

Note: The *MSS_UART_enable_irq()* function also enables the MSS UART instance interrupt in the Cortex-M3 NVIC.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

irq_mask

The *irq_mask* parameter is used to select which of the MSS UART's interrupts you want to enable. The allowed value for the *irq_mask* parameter is one of the following constants or a bitwise OR of more than one:

- MSS_UART_RBF_IRQ (bit mask = 0x001)
- MSS_UART_TBE_IRQ (bit mask = 0x002)
- MSS_UART_LS_IRQ (bit mask = 0x004)
- MSS_UART_MS_IRQ (bit mask = 0x008)
- MSS_UART_RTO_IRQ (bit mask = 0x010)
- MSS_UART_NACK_IRQ (bit mask = 0x020)
- MSS_UART_PIDPE_IRQ (bit mask = 0x040)
- MSS_UART_LINB_IRQ (bit mask = 0x080)
- MSS_UART_LINS_IRQ (bit mask = 0x100)

Return Value

This function does not return a value.

Example

```
MSS_UART_enable_irq( &g_mss_uart0,( MSS_UART_RBF_IRQ | MSS_UART_TBE_IRQ ) );
```

MSS_UART_disable_irq

Prototype

```
void MSS_UART_disable_irq
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_t irq_mask
);
```

Description

The *MSS_UART_disable_irq()* function disables the MSS UART interrupts specified by the *irq_mask* parameter. The *irq_mask* parameter identifies the MSS UART interrupts by bit position, as defined in the interrupt enable register (IER) of MSS UART. The MSS UART interrupts and their identifying bit positions are as follows:

When an *irq_mask* bit position is set to 1, this function disables the corresponding MSS UART interrupt in the IER register. When an *irq_mask* bit position is set to 0, the corresponding interrupt's state remains unchanged in the IER register.

Note: If you disable all four of the UART's interrupts, the *MSS_UART_disable_irq()* function also disables the MSS UART instance interrupt in the Cortex-M3 NVIC.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

irq_mask

The *irq_mask* parameter is used to select which of the MSS UART's interrupts you want to disable. The allowed value for the *irq_mask* parameter is one of the following constants or a bitwise OR of more than one:

- MSS_UART_RBF_IRQ (bit mask = 0x001)
- MSS_UART_TBE_IRQ (bit mask = 0x002)
- MSS_UART_LS_IRQ (bit mask = 0x004)
- MSS_UART_MS_IRQ (bit mask = 0x008)
- MSS_UART_RTO_IRQ (bit mask = 0x010)
- MSS_UART_NACK_IRQ (bit mask = 0x020)
- MSS_UART_PIDPE_IRQ (bit mask = 0x040)
- MSS_UART_LINB_IRQ (bit mask = 0x080)
- MSS_UART_LINS_IRQ (bit mask = 0x100)

Return Value

This function does not return a value.

Example

```
MSS_UART_disable_irq( &g_mss_uart0, ( MSS_UART_RBF_IRQ | MSS_UART_TBE_IRQ ) );
```

MSS_UART_set_pidpei_handler

Prototype

```
void
MSS_UART_set_pidpei_handler
```

```
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_pidpei_handler()* function is used assign a custom interrupt handler for the PIDPEI (PID parity error interrupt) when the MSS UART is operating in LIN mode.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Handler

The *handler* parameter is the pointer to the custom handler function. This parameter is of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_set_pidpei_handler ( &g_mss_uart0, my_pidpei_handler );
```

MSS_UART_set_linbreak_handler

Prototype

```
void
MSS_UART_set_linbreak_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_linbreak_handler()* function is used assign a custom interrupt handler for the LIN Break detection interrupt when the MSS UART is operating in LIN mode.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Handler

The *handler* parameter is the pointer to the custom handler function. This parameter is of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_set_linbreak_handler( &g_mss_uart0, my_break_handler );
```

MSS_UART_set_linsync_handler

Prototype

```
void
MSS_UART_set_linsync_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_linsync_handler()* function is used assign a custom interrupt handler for the LIN Sync character detection interrupt when the MSS UART is operating in LIN mode.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Handler

The *handler* parameter is the pointer to the custom handler function. This parameter is of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_set_linsync_handler ( &g_mss_uart0, my_linsync_handler );
```

MSS_UART_set_nack_handler

Prototype

```
void
MSS_UART_set_nack_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_nack_handler()* function is used assign a custom interrupt handler for the NACK character detection interrupt when the MSS UART is operating in Smartcard mode.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0*

and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Handler

The *handler* parameter is the pointer to the custom handler function. This parameter is of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_set_nack_handler ( &g_mss_uart0, my_nack_handler );
```

MSS_UART_set_rx_timeout_handler

Prototype

```
void
MSS_UART_set_rx_timeout_handler
(
    mss_uart_instance_t * this_uart,
    mss_uart_irq_handler_t handler
);
```

Description

The *MSS_UART_set_rx_timeout_handler()* function is used assign a custom interrupt handler for the receiver timeout interrupt when the MSS UART is operating in mode. It finds application in IrDA mode of operation.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Handler

The *handler* parameter is the pointer to the custom handler function. This parameter is of type *mss_uart_irq_handler_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_set_rx_timeout_handler ( &g_mss_uart0, my_rxtimeout_handler );
```

MSS_UART_get_rx_status

Prototype

```
uint8_t MSS_UART_get_rx_status
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_get_rx_status()* function returns the receiver error status of the MSS UART instance. It reads both the current error status of the receiver from the UART's line status register (LSR) and the accumulated error status from preceding calls to the *MSS_UART_get_rx()* function, and it combines them using a bitwise OR. It returns the cumulative overrun, parity, framing, break and FIFO error status of the receiver, since the previous call to *MSS_UART_get_rx_status()*, as an 8-bit encoded value.

Note: The *MSS_UART_get_rx()* function reads and accumulates the receiver status of the MSS UART instance before reading each byte from the receiver's data register/FIFO. The driver maintains a sticky record of the cumulative receiver error status, which persists after the *MSS_UART_get_rx()* function returns. The *MSS_UART_get_rx_status()* function clears the driver's sticky receiver error record before returning.

Note: The driver's transmit functions also read the line status register (LSR) as part of their implementation. When the driver reads the LSR, the UART clears any active receiver error bits in the LSR. This could result in the driver losing receiver errors. To avoid any loss of receiver errors, the transmit functions also update the driver's sticky record of the cumulative receiver error status whenever they read the LSR.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function returns the UART's receiver error status as an 8-bit unsigned integer. The returned value is 0 if no receiver errors occurred. The driver provides a set of bit mask constants that should be compared with and/or used to mask the returned value to determine the receiver error status.

When the return value is compared to the following bit masks, a non-zero result indicates that the corresponding error occurred:

- MSS_UART_OVERRUN_ERROR (bit mask = 0x02)
- MSS_UART_PARITY_ERROR (bit mask = 0x04)
- MSS_UART_FRAMING_ERROR (bit mask = 0x08)
- MSS_UART_BREAK_ERROR (bit mask = 0x10)
- MSS_UART_FIFO_ERROR (bit mask = 0x80)

When the return value is compared to the following bit mask, a non-zero result indicates that no error occurred:

- MSS_UART_NO_ERROR (bit mask = 0x00)

Upon unsuccessful execution, this function returns:

- MSS_UART_INVALID_PARAM (bit mask = 0xFF)

Example

```
uint8_t rx_data[MAX_RX_DATA_SIZE];
uint8_t err_status;
err_status = MSS_UART_get_rx_status(&g_mss_uart0);
if(MSS_UART_NO_ERROR == err_status )
{
     rx_size = MSS_UART_get_rx( &g_mss_uart0, rx_data, MAX_RX_DATA_SIZE );
}
```

MSS_UART_get_tx_status

Prototype

```
uint8_t MSS_UART_get_tx_status
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_get_tx_status()* function returns the transmitter status of the MSS UART instance. It reads both the UART's line status register (LSR) and returns the status of the transmit holding register empty (THRE) and transmitter empty (TEMT) bits.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function returns the UART's transmitter status as an 8-bit unsigned integer. The returned value is 0 if the transmitter status bits are not set or the function execution failed. The driver provides a set of bit mask constants that should be compared with and/or used to mask the returned value to determine the transmitter status.

When the return value is compared to the following bitmasks, a non-zero result indicates that the corresponding transmitter status bit is set:

- MSS_UART_THRE (bit mask = 0x20)
- MSS_UART_TEMT (bit mask = 0x40)

When the return value is compared to the following bit mask, a non-zero result indicates that the transmitter is busy or the function execution failed.

- MSS_UART_TX_BUSY (bit mask = 0x00)

Example

```
uint8_t tx_buff[10] = "abcdefghi";
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_polled_tx( &g_mss_uart0, tx_buff, sizeof(tx_buff));
while ( ! (MSS_UART_TEMT & MSS_UART_get_tx_status( &g_mss_uart0 ) ) )
{ ; }
```

MSS_UART_get_modem_status

Prototype

```
uint8_t
MSS_UART_get_modem_status
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_get_modem_status()* function returns the modem status of the MSS UART instance. It reads the modem status register (MSR) and returns the 8 bit value. The bit encoding of the returned value is exactly the same as the definition of the bits in the MSR.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function returns current state of the UART's MSR as an 8 bit unsigned integer. The driver provides the following set of bit mask constants that should be compared with and/or used to mask the returned value to determine the modem status:

- MSS_UART_DCTS (bit mask = 0x01)
- MSS_UART_DDSR (bit mask = 0x02)
- MSS_UART_TERI (bit mask = 0x04)
- MSS_UART_DDCD (bit mask = 0x08)
- MSS_UART_CTS (bit mask = 0x10)
- MSS_UART_DSR (bit mask = 0x20)
- MSS_UART_RI (bit mask = 0x40)
- MSS_UART_DCD (bit mask = 0x80)

Example

```
void uart_modem_status_isr( mss_uart_instance_t * this_uart )
{
    uint8_t status;
    status = MSS_UART_get_modem_status( this_uart );
    if( status & MSS_UART_DCTS ) {
        uart_dcts_handler();
    }
    if( status & MSS_UART_CTS ) {
        uart_cts_handler();
    }
}
```

MSS_UART_set_loopback

Prototype

```
void MSS_UART_set_loopback
(
    mss_uart_instance_t * this_uart,
    mss_uart_loopback_t loopback
);
```

Description

The *MSS_UART_set_loopback()* function is used to locally loopback the Tx and Rx lines of a UART. This is not to be confused with the loopback of UART0 to UART1, which can be achieved through the microcontroller subsystem's system registers.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

loopback

The *loopback* parameter indicates whether or not the UART's transmit and receive lines should be looped back. Allowed values are as follows:

- MSS_UART_LOCAL_LOOPBACK_ON
- MSS_UART_LOCAL_LOOPBACK_OFF
- MSS_UART_REMOTE_LOOPBACK_ON
- MSS_UART_REMOTE_LOOPBACK_OFF
- MSS_UART_AUTO_ECHO_ON
- MSS_UART_AUTO_ECHO_OFF

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_loopback( &g_mss_uart0, MSS_UART_LOCAL_LOOPBACK_OFF );
```

MSS_UART_set_break

Prototype

```
void
MSS_UART_set_break
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_set_break()* function is used to send the break (9 zeros after stop bit) signal on the TX line. This function can be used only when the MSS UART is initialized in LIN mode by using *MSS_UART_lin_init().*

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_break( &g_mss_uart0 );
```

MSS_UART_clear_break

Prototype

```
void
MSS_UART_clear_break
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_clear_break()* function is used to remove the break signal on the TX line. This function can be used only when the MSS UART is initialized in LIN mode by using *MSS_UART_lin_init().*

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_clear_break( &g_mss_uart0 );
```

MSS_UART_enable_half_duplex

Prototype

```
void
MSS_UART_enable_half_duplex
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_enable_half_duplex()* function is used to enable the half-duplex (single wire) mode for the MSS UART. Though it finds application in Smartcard mode, half-duplex mode can be used in other modes as well.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_enable_half_duplex( &g_mss_uart0 );
```

MSS_UART_disable_half_duplex

Prototype

```
void
MSS_UART_disable_half_duplex
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_disable_half_duplex()* function is used to disable the half-duplex (single wire) mode for the MSS UART. Though it finds application in Smartcard mode, half-duplex mode can be used in other modes as well.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_disable_half_duplex( &g_mss_uart0 );
```

MSS_UART_set_rx_endian

Prototype

```
void
MSS_UART_set_rx_endian
(
    mss_uart_instance_t * this_uart,
    mss_uart_endian_t endian
);
```

Description

The *MSS_UART_set_rx_endian()* function is used to configure the LSB first or MSB first setting for MSS UART receiver.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

endian

The *endian* parameter tells the LSB first or MSB first configuration. This parameter is of type *mss_uart_endian_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_rx_endian( &g_mss_uart0, MSS_UART_LITTLEEND );
```

MSS_UART_set_tx_endian

Prototype

```
void
MSS_UART_set_tx_endian
(
    mss_uart_instance_t * this_uart,
    mss_uart_endian_t endian
);
```

Description

The *MSS_UART_set_tx_endian()* function is used to configure the LSB first or MSB first setting for MSS UART transmitter.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

endian

The *endian* parameter tells the LSB first or MSB first configuration. This parameter is of type *mss_uart_endian_t*.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_tx_endian( &g_mss_uart0, MSS_UART_LITTLEEND );
```

MSS_UART_set_filter_length

Prototype

```
void
MSS_UART_set_filter_length
(
    mss_uart_instance_t * this_uart,
    mss_uart_filter_length_t length
);
```

Description

The *MSS_UART_set_filter_length()* function is used to configure the glitch filter length of the MSS UART. This should be configured in accordance with the chosen baud rate.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

length

The length parameter is of mss_uart_filter_length_t type that determines the length of the glitch filter.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
```

```
                MSS_UART_57600_BAUD,
                MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_filter_length( &g_mss_uart0, MSS_UART_LEN2 );
```

MSS_UART_enable_afm

Prototype

```
void
MSS_UART_enable_afm
(
     mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_enable_afm()* function is used to enable address flag detection mode of the MSS UART.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
                MSS_UART_57600_BAUD,
                MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_enable_afm( &g_mss_uart0 );
```

MSS_UART_disable_afm

Prototype

```
void
MSS_UART_disable_afm
(
     mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_disable_afm()* function is used to disable address flag detection mode of the MSS UART.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_disable_afm( &g_mss_uart0 );
```

MSS_UART_enable_afclear

Prototype

```
void
MSS_UART_enable_afclear
(
     mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_enable_afclear()* function is used to enable address flag clear of the MSS UART. This should be used in conjunction with address flag detection mode (AFM).

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_enable_afclear( &g_mss_uart0 );
```

MSS_UART_disable_afclear

Prototype

```
void
MSS_UART_disable_afclear
(
     mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_disable_afclear ()* function is used to disable address flag clear of the MSS UART. This should be used in conjunction with address flag detection mode (AFM).

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_disable_afclear( &g_mss_uart0 );
```

MSS_UART_enable_rx_timeout

Prototype

```
void
MSS_UART_enable_rx_timeout
(
    mss_uart_instance_t * this_uart,
    uint8_t timeout
);
```

Description

The *MSS_UART_enable_rx_timeout()* function is used to enable and configure the receiver timeout functionality of MSS UART. This function accepts the timeout parameter and applies the timeout based up on the baud rate as per the formula 4 x timeout x bit time.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

timeout

The timeout parameter specifies the receiver timeout multiple. It should be configured according to the baud rate in use.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_enable_rx_timeout( &g_mss_uart0 , 24 );
```

MSS_UART_disable_rx_timeout

Prototype

```
void
MSS_UART_disable_rx_timeout
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_disable_rx_timeout()* function is used to disable the receiver timeout functionality of MSS UART.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_disable_rx_timeout( &g_mss_uart0 );
```

MSS_UART_enable_tx_time_guard

Prototype

```
void
MSS_UART_enable_tx_time_guard
(
    mss_uart_instance_t * this_uart,
    uint8_t timeguard
);
```

Description

The *MSS_UART_enable_tx_time_guard()* function is used to enable and configure the transmitter time guard functionality of MSS UART. This function accepts the *timeguard* parameter and applies the *timeguard* based up on the baud rate as per the formula timeguard x bit time.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

timeguard

The timeguard parameter specifies the transmitter time guard multiple. It should be configured according to the baud rate in use.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_enable_tx_time_guard( &g_mss_uart0 , 24 );
```

MSS_UART_disable_tx_time_guard

Prototype

```
void
MSS_UART_disable_tx_time_guard
(
    mss_uart_instance_t * this_uart
);
```

Description

The *MSS_UART_disable_tx_time_guard()* function is used to disable the transmitter time guard functionality of MSS UART.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_disable_tx_time_guard( &g_mss_uart0 );
```

MSS_UART_set_address

Prototype

```
void
MSS_UART_set_address
(
    mss_uart_instance_t * this_uart,
    uint8_t address
);
```

Description

The *MSS_UART_set_address()* function is used to set the 8-bit address for the MSS UART referenced by *this_uart* parameter.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

address

The *address* parameter is the 8-bit address which is to be configured to the MSS UART referenced by *this_uart* parameter.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_address( &g_mss_uart0, 0xAA );
```

MSS_UART_set_ready_mode

Prototype

```
void
MSS_UART_set_ready_mode
(
    mss_uart_instance_t * this_uart,
    mss_uart_ready_mode_t mode
);
```

Description

The *MSS_UART_set_ready_mode()* function is used to configure the MODE0 or MODE1 to the TXRDY and RXRDY signals of the MSS UART referenced by *this_uart* parameter. The *mode* parameter is used to provide the mode to be configured.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

mode

The *mode* parameter is the *mss_uart_ready_mode_t* type which is used to configure the TXRDY and RXRDY signal modes.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_ready_mode( &g_mss_uart0, MSS_UART_READY_MODE0 );
```

MSS_UART_set_usart_mode

Prototype

```
void
MSS_UART_set_usart_mode
(
    mss_uart_instance_t * this_uart,
    mss_uart_usart_mode_t mode
);
```

Description

The *MSS_UART_set_usart_mode()* function is used to configure the MSS UART referenced by the parameter *this_uart* in USART mode. Various USART modes are supported which can be configured by the parameter *mode* of type *mss_uart_usart_mode_t*.

Parameters

this_uart

The *this_uart* parameter is a pointer to an *mss_uart_instance_t* structure identifying the MSS UART hardware block that will perform the requested function. There are two such data structures, *g_mss_uart0* and *g_mss_uart1*, associated with MSS UART0 and MSS UART1. This parameter must point to either the *g_mss_uart0* or *g_mss_uart1* global data structure defined within the UART driver.

mode

The *mode* parameter is the USART mode to be configured. This parameter is of type mss_uart_usart_mode_t.

Return Value

This function does not return a value.

Example

```
MSS_UART_init(&g_mss_uart0,
              MSS_UART_57600_BAUD,
              MSS_UART_DATA_8_BITS | MSS_UART_NO_PARITY | MSS_UART_ONE_STOP_BIT);

MSS_UART_set_usart_mode( &g_mss_uart0, MSS_UART_SYNC_SLAVE_POS_EDGE_CLK  );
```

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**.

50200413-1/10.15