# Actel SmartFusion™ MSS UART Driver User's Guide

*Version 2.0*

Actel
POWER MATTERS

**Actel Corporation, Mountain View, CA 94043**



Printed in the United States of America

Part Number: 50200195-1

Release: February 2010



## Trademarks

# Table of Contents

# Introduction

The SmartFusion™ microcontroller subsystem (MSS) includes two UART peripherals for serial communication. This driver provides a set of functions for controlling the MSS UARTs 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.

## Features

The MSS UART driver provides the following features:

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

The MSS UART driver is provided as C source code.

## Supported Hardware IP

The MSS UART bare metal driver can be used with Actel's MSS_UART IP version 0.2 or higher, included in the SmartFusion MSS.

# 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 Actel Firmware Catalog, which provides access to the documentation for the driver, generates the driver's source files into an application project, and generates example projects that illustrate how to use the driver. The Actel Firmware Catalog is available from the Actel website: www.actel.com/products/software/firmwarecat/default.aspx.

## Documentation

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

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

## Driver Source Code

The Actel Firmware Catalog generates the driver's source code into the *drivers\mss_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 UART software driver. This file should be included in any C source file that uses the MSS UART software driver.

### mss_uart.c

This C source file contains the implementation of the MSS UART software driver.

## Example Code

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

# Driver Deployment

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

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

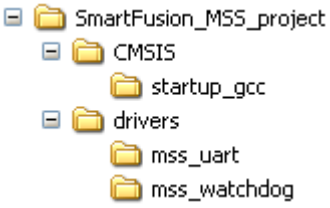


Figure 1 · SmartFusion MSS Project Example

# Driver Configuration

The configuration of all features of the MSS UARTs is covered by this driver with the exception of the SmartFusion IOMUX configuration. SmartFusion 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 UARTs serial signals are routed through IOMUXes to the SmartFusion device external pins. These IOMUXes are configured automatically by the MSS configurator tool in the hardware flow correctly when the MSS UARTs are enabled in that tool. You must ensure that the MSS UARTs are enabled by the MSS configurator tool in the hardware flow; otherwise the serial inputs and outputs will not be connected to the chip's external pins. For more information on IOMUX, refer to the IOMUX section of the SmartFusion Datasheet.

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

# Application Programming Interface

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

## Theory of Operation

The MSS UART 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 UART driver is initialized through a call to the *MSS_UART_init()* function. This function takes the UART's configuration as parameters. The *MSS_UART_init()* function must be called before any other UART driver functions can be called. The first parameter of the *MSS_UART_init()* function is a pointer to one of two global data structures used to store state information for each UART driver. A pointer to these data structures is also used as first parameter to any of the driver functions to identify which UART 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 UART function should be of the form MSS_UART_function_name( &g_mss_uart0, ... ) or MSS_UART_function_name( &g_mss_uart1, ... ).

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.

### Polled Operations

Polled operations where the processor constantly polls the state of the UART registers in order to control data transmit or data receive is performed using these functions:

- *MSS_UART_polled_tx()*
- *MSS_UART_polled_tx_string*
- *MSS_UART_get_rx()*

### Interrupt-Driven Operations

Interrupt-driven operations where the processor sets up transmit or receive and then returns to performing some other operation until an interrupts occurs, indicating that its attention is required, are performed using these functions:

- *MSS_UART_irq_tx()*
- *MSS_UART_tx_complete()*
- *MSS_UART_set_rx_handler()*
- *MSS_UART_get_rx()*

Interrupt-driven transmit is initiated by a call to *MSS_UART_irq_tx()*, specifying the block of data to transmit. The processor can then perform some other operation and later inquire whether transmit has completed by calling the *MSS_UART_tx_complete()* function. Interrupt-driven receive is performed by first registering a receive handler function that will be called by the driver whenever receive data is available. This receive handler function in turns calls the *MSS_UART_get_rx()* function to actually read the received data.

# Types

## mss_uart_loopback_t

### Prototype

```
typedef enum __mss_uart_loopback_t {
    MSS_UART_LOOPBACK_OFF   = 0,
    MSS_UART_LOOPBACK_ON    = 1
} mss_uart_loopback_t;
```

### Description

This enumeration is used as parameter to function *MSS_UART_set_loopback()*. It specifies the loopback configuration of the UARTs. Using MSS_UART_LOOPBACK_ON as parameter to function *MSS_UART_set_loopback()* will set up the UART to locally loopback its Tx and Rx lines.

## 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_rx_trig_level_t;
```

### Description

This enumeration specifies the number of bytes that must be received before a receive interrupt is generated. This enumeration provides the allowed values for the *MSS_UART_set_rx_handler()* function *trigger_level* parameter.

## mss_uart_rx_handler_t

### Prototype

```
typedef void(* mss_uart_rx_handler_t)(void);
```

### Description

This typedef specifies the prototype of functions that can be registered with this driver as receive handler functions.

# Constant Values

## Baud Rates

The following definitions are used to specify standard baud rates as a parameter to the *MSS_UART_init()* function.

| 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 |

Table 1 · Standard Baud Rates

## Data Bits Length

The following defines are used to build the value of the *MSS_UART_init()* function *line_config* parameter.

| 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 |

Table 2 • Data Bits Length Configuration Options

## Parity

The following defines are used to build the value of the *MSS_UART_init()* function *line_config* parameter.

| 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 |

Table 3 • Parity Configuration Options

## Number of Stop Bits

The following defines are used to build the value of the *MSS_UART_init()* function *line_config* parameter.

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

Table 4 • Stop Bit Length Configuration Options

# 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 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. 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. 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 SmartFusion 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 logical OR of one of the following in each of three groups:

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_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.

### 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.

## 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 zero-terminated string. It transfers the text found 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.

### 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 zero-terminated string to be transmitted.

### Return Value

This function does not return a value.

## 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 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 to a memory buffer containing the data to transmit as a parameter. The memory buffer specified through this pointer should remain allocated and contain the data to transmit until the transmit completion has been detected through calls to function *MSS_UART_tx_complete()*.

Note: The *MSS_UART_irq_tx()* function also enables the Transmitter Holding Register Empty (THRE) interrupt and the UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

### 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_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 is used to read the content of a UAR's receive FIFO. It 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. This function is non-blocking and will return 0 immediately if no data has been 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 will be 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( void )
{
    uint8_t rx_buff[RX_BUFF_SIZE];
    uint32_t rx_idx  = 0;
    rx_size = MSS_UART_get_rx( &g_mss_uart1, rx_buff, sizeof(rx_buff) );
    process_rx_data( rx_buff, rx_size );
}
```

## 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:

- LOOPBACK_ON
- LOOPBACK_OFF

### Return Value

This function does not return a value.

## MSS_UART_set_rx_handler

### Prototype

```
void MSS_UART_set_rx_handler
(
        mss_uart_instance_t * this_uart,
        mss_uart_rx_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 which will be called by the driver when a UART Received Data Available (RDA) interrupt occurs. You must create and register the handler function to suit your application. The *MSS_UART_set_rx_handler()* function also enables the UART Received Data Available interrupt and the UART instance interrupt in the Cortex-M3 NVIC as part of its implementation.

## 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 handler function provided by your application which will be called as a result of a UART Received Data Available interrupt.

### 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 a Received Data Available 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( void )
{
    MSS_UART_get_rx( &g_mss_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_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 an 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.

### 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.

# Product Support

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

## Customer Service

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

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

## Actel Customer Technical Support Center

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

## Actel Technical Support

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

## Website

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

## Contacting the Customer Technical Support Center

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

### Email

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

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

## Phone

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

**650.318.4460**
**800.262.1060**

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


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

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

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

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

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

50200195-1/2.10