

Introduction

Altera® Cyclone™ devices can be configured in active serial configuration mode. This mode reads a configuration bitstream from an external serial configuration device. The memory within the serial configuration device is divided into two sections:

- Configuration memory — contains the bitstream
- General purpose memory — used for any application-specific storage



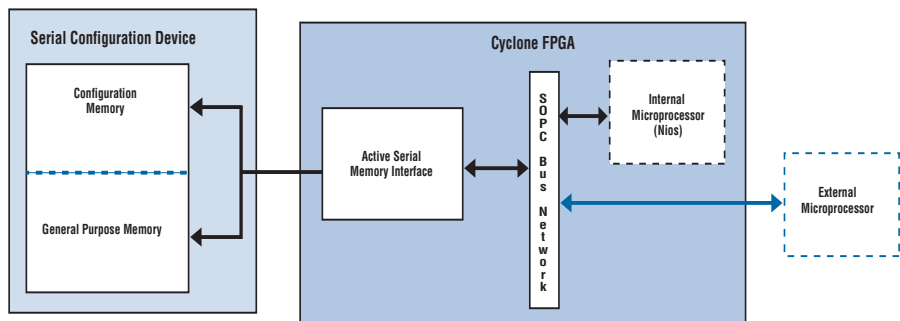
See the *Cyclone FPGA Family Data Sheet* for more information about active serial configuration mode.

The active serial memory interface (ASMI) is an intellectual property (IP) block that can be included in your Cyclone design. Application logic can access both the configuration memory and general purpose memory sections of the serial configuration device through the ASMI. See **Figure 1**. The ASMI is an SOPC Builder component that is especially suitable for allowing microprocessor systems access to serial configuration memory.



The active serial memory interface can only be used within Cyclone devices.

Figure 1. Active Serial Memory Interface & Serial Configuration Device



Functional Description

Through SOPC Builder, the ASMI component can be used by an internal microprocessor on the FPGA (e.g. Nios[®] embedded processor) or by an external processor. Software running on a microprocessor accesses the ASMI component through a standard set of C library functions.



See the [SOPC Builder Data Sheet](#) for more information about connecting external processors to SOPC Builder systems.

The ASMI block allows software running on a microprocessor to read and write data to the non-volatile serial configuration device. This software can use memory within the serial configuration device for two distinct purposes:

- General-purpose non-volatile storage
- Software access to device-configuration data



In order to access the serial configuration device, the Cyclone device must be configured in active-serial mode or via JTAG. The ASMI component is disabled when configuration is done in passive-serial mode.



See the [Serial Configuration Devices \(EPCS1 & EPCS4\) Data Sheet](#) for further information.

Using the ASMI to Access General-Purpose Nonvolatile Storage

Software subroutines can read and write data to the general-purpose memory portion of the serial configuration device (i.e., the portion of the device which does not contain configuration data). The general-purpose memory can be used, through the provided C subroutines, like a mass-storage read/write device. Software applications can use this memory for any purpose, including main program storage, persistent settings or data between system power-down events, and accessing device-configuration data.

Main Program Storage

Embedded systems can use the general-purpose memory to store a main program. An auxiliary boot-loader program (residing, for example, in on-chip memory) can use the ASMI to copy a main program out of general-purpose memory into RAM (main memory) and then execute the main program. In this example, the boot-loader program could call the `nr_asmi_read_buffer` subroutine. See "[nr_asmi_read_buffer](#)" on [page 7](#) to copy the program into main memory.

Persistent Settings or Data between System Power-Down Events

Some embedded systems need access to small amounts of non-volatile memory to store settings, logs, or other persistent data. The ASMI's C subroutines allow a system to read and write individual non-volatile storage locations.

Using the ASMI to Access Device-Configuration Data

Some embedded systems may include the ability to modify their own configuration data. For example, a system could receive new configuration data through a network and then update its own hardware configuration by writing this data into the configuration memory portion of the serial configuration device.

The ASMI's C library includes a subroutine `nr_asmi_past_config` (see "[nr_asmi_past_config](#)" on page 8) for finding the boundary between the configuration memory and general-purpose memory portions of the serial configuration device.



When Quartus® II software programs the serial configuration device, all previous content is erased.

SOPC Builder Library Component

SOPC Builder creates bus-based systems (typically microprocessor-based) in Altera FPGAs. The SOPC Builder tool assembles library components like processors, memories, interfaces and I/O peripherals.

The ASMI block appears within SOPC Builder as a library component with one Avalon slave port and no (zero) external I/O pins. The ASMI accesses the serial configuration device through dedicated pins on the Cyclone device — not through general-purpose I/O pins. Because the ASMI does not use general-purpose I/O pins, the external connections are not exposed as interface pins on the design. Systems which include an ASMI component will not have any top-level I/O pins associated with this block. Quartus II software handles the connections between the ASMI and the dedicated Cyclone device pins.



See the [SOPC Builder Data Sheet](#) for more information about the SOPC Builder tool.

Software Subroutines

The ASMI component is packaged with C subroutines accessible through SOPC Builder-generated software development kit (SDK) libraries.



The provided C subroutines are the only supported interface to the ASMI block.

Table 1 lists the ASMI software subroutines available in the Nios library (**lib** directory). These functions, types, and definitions are located in **excalibur.h**.

| <i>Table 1. ASMI Software Subroutines</i> | |
|-----------------------------------------------|---------------------------------------------------------------------------------------|
| Subroutine | Description |
| <code>nr_asmi_protect_region</code> | Selects the portion of serial memory that is write protected |
| <code>nr_asmi_lowest_protected_address</code> | Returns the lowest protected address in serial memory |
| <code>nr_asmi_read_byte</code> | Reads one byte from serial memory at the specified address |
| <code>nr_asmi_write_byte</code> | Writes one byte of data to the specified address |
| <code>nr_asmi_erase_sector</code> | Erases the sector of serial memory containing the specified address |
| <code>nr_asmi_erase_bulk</code> | Erases the entire serial memory |
| <code>nr_asmi_read_buffer</code> | Reads a consecutive sequence of bytes from the serial memory into the provided buffer |
| <code>nr_asmi_write_buffer</code> | Writes a consecutive sequence of bytes to the serial configuration memory |
| <code>nr_asmi_past_config</code> | Returns the first available address past the hardware configuration |

nr_asmi_protect_region

This subroutine selects the portion of serial memory that is write protected. Write protected means the memory cannot be overwritten or erased. This region of serial memory is always some fraction of the total memory at the top of the address range. The available set of fractions that can be protected depends on the size of the memory device. See [Table 2 on page 5](#).

Access to a protected region of memory by the `nr_asmi_write_byte`, `nr_asmi_erase_sector`, `nr_asmi_erase_bulk`, or `nr_asmi_write_buffer` subroutine will return the `na_asmi_err_write_failed` code. See [Table 3 on page 9](#).

Syntax

```
int nr_asmi_protect_region (int bpcode);
```

Parameters

| <i>Table 2. bpcode Parameter</i> | |
|------------------------------------|------------------------------------------------------------------------------------------------|
| Description | |
| Code specifying region to protect. | |
| Value ⁽¹⁾ | Value Description |
| na_asmi_protect_none | Makes the entire memory available for write and erase |
| na_asmi_protect_top_eighth | Protects the highest (top) eighth of the memory address range – ONLY valid for a 4 Mbit device |
| na_asmi_protect_top_quarter | Protects the highest (top) quarter of the memory address range |
| na_asmi_protect_top_half | Protects the highest (top) half of the memory address range |
| na_asmi_protect_all | Protects the entire memory address range |

Note to Table 2

(1) These values are declared in `excalibur.h`.

nr_asmi_lowest_protected_address

This subroutine returns the address at the boundary between protected and unprotected serial memory. See “[nr_asmi_protect_region](#)” on page 4. The address returned is the first (lowest) address within the region that has been protected. If the entire memory is protected, this subroutine returns zero. If none of the memory is protected, it returns the size of the memory device.

Syntax

```
unsigned long nr_asmi_lowest_protected_address();
```

nr_asmi_read_byte

This subroutine reads one byte from the serial memory at the specified address.

Syntax

```
int nr_asmi_read_byte (unsigned long address,
                      unsigned char *data);
```

Parameters

| Parameter Name | Description |
|----------------|-------------------------------------|
| address | Address in serial memory to be read |
| data | Pointer to retrieved data value |

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_write_byte

This subroutine writes one byte of data to the specified address. Data cannot be written to memory unless the data stored at the address has been previously erased. If the address is within a protected region, this subroutine will return the `na_asmi_err_write_failed` code. See [Table 3 on page 9](#).

Syntax

```
int nr_asmi_write_byte (unsigned long address, unsigned
                        char data);
```

Parameters

| Parameter Name | Description |
|----------------|-----------------------|
| address | Address to be written |
| data | Data to be written |

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_erase_sector

This subroutine erases the sector of serial memory that contains the specified address. A sector is the smallest segment of serial memory that can be erased. The sector size depends on the size of the memory device.

- 1 Mbit of memory is divided into 4 equal sectors of 32 Kbytes
- 4 Mbits of memory are divided into 8 equal sectors of 64 Kbytes

If the sector is within a protected region, this subroutine will return the `na_asmi_err_write_failed` code. See [Table 3 on page 9](#).

Syntax

```
int nr_asmi_erase_sector (unsigned long address);
```

Parameters

| Parameter Name | Description |
|----------------|---------------------------------------|
| address | An address in the sector to be erased |

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_erase_bulk

This subroutine erases the entire memory. If any portion of the memory is protected, this subroutine returns the `na_asmi_err_write_failed` code. See [Table 3 on page 9](#).

Syntax

```
int nr_asmi_erase_bulk();
```

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_read_buffer

This subroutine reads a consecutive sequence of bytes from the serial memory into the provided buffer.

Syntax

```
int nr_asmi_read_buffer (unsigned long address, int
                        length, unsigned char *data);
```

Parameters

| Parameter Name | Description |
|----------------|--------------------------------|
| address | The first address to be read |
| length | The number of bytes to be read |
| data | Pointer to destination buffer |

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_write_buffer

This subroutine writes a consecutive sequence of bytes to the serial configuration memory. Data cannot be written to serial memory unless the memory content at the specific address of the buffer has been previously erased. If any portion of the input buffer attempts to write to a protected region of serial memory, this subroutine returns the `na_asmi_err_write_failed` code. See [Table 3 on page 9](#).



If the destination address range overlaps a protected region, the entire write operation will fail and none of the data will be written.

Syntax

```
int nr_asmi_write_buffer (unsigned long address, int
                          length, unsigned char *data);
```

Parameters

| Parameter Name | Description |
|----------------|-----------------------------------|
| address | The first address to be written |
| length | The number of bytes to be written |
| data | Pointer to source buffer |

Return value — Return code (see [Table 3 on page 9](#)).

nr_asmi_past_config

The serial memory is typically used to store hardware configuration information. Configuration information is stored starting at address 0. This subroutine returns the first address past the end of the configuration information that is available for use.

Syntax

```
int nr_asmi_past_config (unsigned long *addr);
```

Parameters

| Parameter Name | Description |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| addr | This parameter is a pointer to an integer that will contain the first available address past the configuration data. |

Return Codes

Table 3 describes the return codes that can occur when using the ASMI subroutines which are found in Table 1 on page 4.

| Table 3. ASMI Subroutine Return Codes | |
|----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Return Code | Description |
| na_asmi_err_device_not_present | Not able to communicate with configuration memory |
| na_asmi_err_device_not_ready | Able to communicate with configuration memory but could not perform the operation |
| na_asmi_success | Subroutine was completed successfully |
| na_asmi_err_timedout | The device timed out while memory was writing received data |
| na_asmi_err_write_failed | (1) Data was written to memory but during validation, an error occurred (2) Attempts to write data to a protected region or erase data from a protected region failed |



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com

Copyright © 2003 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

