

## Introduction

If you are designing with Altera® Stratix® II, Cyclone™ II, or Cyclone FPGAs, the active serial memory interface (ASMI) controller reference design is useful in accessing the unused portion of Altera EPCS1, EPCS4, EPCS16, and EPCS64 serial configuration devices. The Nios® II embedded processor also provides a complete solution for this capability by using the EPCS Controller module in SOPC Builder.

This reference design provides a top-level design to write to, read from, or erase data on the serial configuration device. By using this reference design, you do not need to be concerned with the protocol needed to read from or write to the memory in the serial configuration device. The ASMI controller reference design handles the protocol. You can use this reference design for all serial configuration devices (EPCS1, EPCS4, EPCS16, and EPCS64 devices). The MSEL pin settings must be set to Active Serial (AS) mode (Stratix II, Cyclone II, and Cyclone devices) or Fast Active Serial (FAS) mode (Stratix II and Cyclone II devices) so that the FPGA's ASMI is always enabled.



This reference design was tested and verified successfully on an Altera Cyclone Nios II board. You can modify and change this reference design to suit your system use.



For more information on the features of Altera serial configuration devices, refer to the *Serial Configuration Devices (EPCS1, EPCS4, EPCS16 & EPCS64)* chapter in *volume 2* of the *Configuration Handbook*.

## Port Names of ASMI\_CONTROLLER

Table 1 provides a brief description for the input and output ports found on the top level of this reference design, which is the ASMI\_CONTROLLER module.

Port Name	Direction	Description
clk <sub>in</sub>	Input	This input clock is fed to the <code>clk_divider</code> block, which is a counter inside the reference design. The <code>clk<sub>in</sub></code> signal can be divided by the <code>clk_divider</code> so that the interfacing signals sent to the serial configuration device can still meet the timing specifications.
reset	Input	This <code>reset</code> signal is used to reset the ASMI_CONTROLLER block in case it is stuck in some unknown state. When the memory inside the serial configuration device is protected, the erase and write process cannot be completed and continues until it is reset.
read <sub>sid</sub>	Input	This active high signal reads the silicon ID in the serial configuration device to verify its existence. When <code>read<sub>sid</sub></code> is high, the read silicon ID operation is repeated until a valid EPCS silicon ID is read. This input signal has the highest priority of other signals.
rden	Input	This active high signal reads the data stored in the serial configuration device memory. This input signal has the second highest priority of other active signals.
wren	Input	This active high signal writes the data into the serial configuration device memory and is based on the <code>data_write</code> and <code>addr</code> inputs. This input signal has the third highest priority compared with other active signals.
wrstatus	Input	This active high signal writes the status register to protect the selected sectors inside the serial configuration device memory. Once the selected sectors are protected, they continue to be protected until their block protect bits are cleared. The status register is a non-volatile register and thus its value is maintained even after power-off. This input signal has the fourth highest priority of other active signals.
erase <sub>sector</sub>	Input	This active high signal erases one sector in the serial configuration device memory and is based on the sector address shown by the <code>addr</code> inputs. This input signal has the fifth highest priority of other active signals.
erase <sub>bulk</sub>	Input	This active high signal erases one sector in the serial configuration device memory and is based on the sector address shown by the <code>addr</code> inputs. This input signal has the lowest priority of other active signals.

**Table 1. Input & Output Port Descriptions (Part 2 of 3)**

Port Name	Direction	Description
data_write[7..0]	Input	This is the 8-bit data input bus that provides the data value to be written into the serial configuration device memory. When <code>wrstatus</code> is enabled, the EPCS status register is written with the <code>data_write</code> value.
addr[23..0]	Input	This is the 24-bit address input bus that provides the start address to write the data into the serial configuration device memory. When the <code>erase_sector</code> is enabled, the memory sector shown by the <code>addr</code> bus is erased accordingly, provided that the selected memory sector is not protected.
num_of_bytes[8..0]	Input	This 9-bit input specifies the number of data bytes to be written into the serial configuration device memory when <code>wren</code> is enabled. The maximum value for this 9-bit input is 256 or <code>0x100</code> , since the serial configuration device allows a maximum of 256 data bytes written in the one <code>write_bytes</code> operation.
data_read[7..0]	Output	This 8-bit data output bus shows the data read from the serial configuration device memory when <code>rden</code> is enabled. The most significant bit (MSB) of a data byte read is shown in <code>data_read[7]</code> and the least significant bit (LSB) of a data byte read is shown in <code>data_read[0]</code> .
read_addr[23..0]	Output	This 24-bit <code>read_addr</code> bus shows the address of the data read from the serial configuration device memory when <code>rden</code> is enabled.
write_in_progress	Output	This output signal is high when the <code>write_bytes</code> operation is in progress until it is complete. If the <code>write_bytes</code> operation is performed on the protected memory in the serial configuration device, then this signal is always high because the <code>write_bytes</code> operation can never be completed. In this case, you can toggle the <code>reset</code> input signal to stop the <code>write_bytes</code> operation and the <code>write_in_progress</code> operation will go low.
ws_in_progress	Output	This output signal is high when the <code>write_status</code> operation is in progress.
es_in_progress	Output	This output signal is high when the <code>erase_sector</code> operation is in progress until the operation is complete. If the <code>erase_sector</code> operation is performed on the protected memory in the serial configuration device, then this signal is always high because the <code>erase_sector</code> operation can never be completed. In this case, you can toggle the <code>reset</code> input signal to stop the <code>erase_sector</code> operation and the <code>es_in_progress</code> operation will go low.

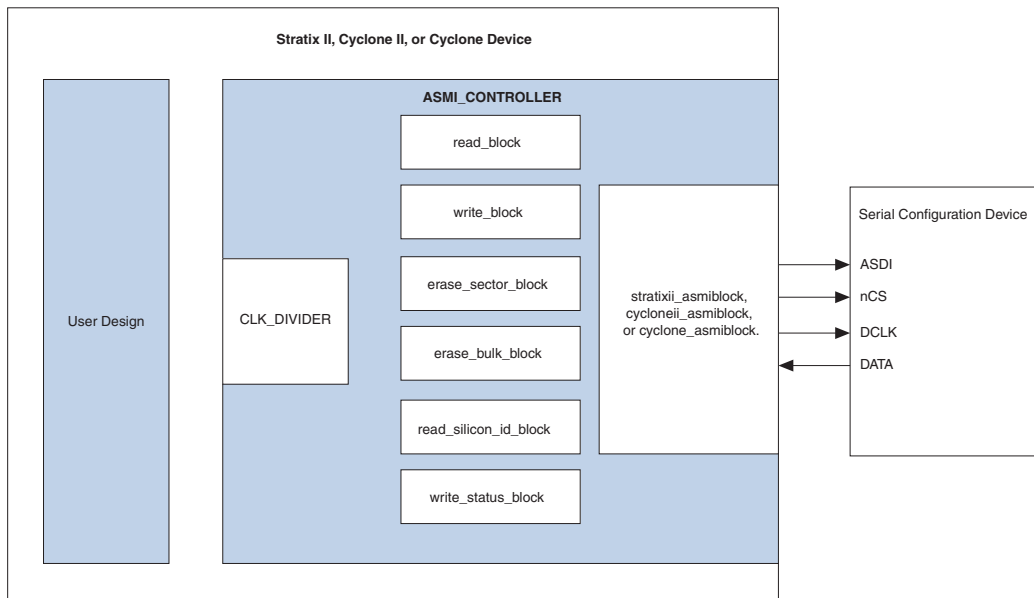
**Table 1. Input & Output Port Descriptions (Part 3 of 3)**

Port Name	Direction	Description
eb_in_progress	Output	This output signal is high when the <code>erase_bulk</code> operation is in progress until the operation is complete. If there is at least one protected sector memory in the serial configuration device, the <code>erase_bulk</code> operation cannot be completed and therefore, this signal is always high and the unprotected sector memory is not erased. In this case, you can toggle the <code>reset</code> input signal to stop the <code>erase_bulk</code> operation and the <code>eb_in_progress</code> operation will go low.
epcs_id[7..0]	Output	After <code>read_sid</code> is enabled, the read EPCS silicon ID is shown on this <code>epcs_id[7..0]</code> output bus. <code>epcs_id[7..0]</code> is reset to zero only when the <code>reset</code> signal is toggled. You can identify the memory capacity in the serial configuration device by reading the <code>epcs_id[7..0]</code> output bus.
write_clk	Output	You can use <code>write_clk</code> to synchronize your <code>data_write</code> input to the ASMI Controller. If you do not need this signal, you can leave it unconnected.

## Block Diagram

Figure 1 shows a simplified block diagram for the ASMI\_CONTROLLER module. You can remove any sub-block inside the ASMI\_CONTROLLER module to reduce the number of logic elements (LEs) used, or add your own code to the module.

`<device>_asmiblock` is a WYSIWYG component used to enable the ASMI connection to the serial configuration device. The `<device>_asmiblock` primitive atom contains the ASMI ports (`DCLK`, `DATA`, `nCSO`, and `ASDO`) and must be included in the ASMI\_CONTROLLER module.

**Figure 1. ASMI\_CONTROLLER Block Diagram**

## Included Files

Table 2 provides the directory structure of the deliverables in this reference design release:

Directory	Filename	Description
\Doc	asmi_controller_reference_design.pdf	This file contains documentation on the ASMI_CONTROLLER reference design.
\Source	asmi_controller.v	This is the top-level design file of the ASMI_CONTROLLER design. The sub-blocks are instantiated in this file. There is an <code>lpm_counter</code> instantiated with the name <code>clk_divider</code> in this top-level design. The input to the <code>clk_divider</code> is your system clock. Use the <code>clk_divider</code> module to divide your system clock to a maximum of 40 MHz for EPCS1 and EPCS4 devices, or 80 MHz for EPCS16 and EPCS64 devices. If you can directly feed the ASMI_CONTROLLER module with the desired clock frequency, you can remove the <code>clk_divider</code> module.
	read_silicon_id_block.v	This module performs <code>read_silicon_id</code> operation.
	read_block.v	This module performs <code>read_bytes</code> operation.
	write_block.v	This module performs <code>write_bytes</code> operation.
	write_status_block.v	This module performs <code>write_status</code> operation.
	erase_sector_block.v	This module performs <code>erase_sector</code> operation.
	erase_bulk_block.v	This module performs <code>erase_bulk</code> operation.

## Write to Serial Configuration Device

While writing data to the serial configuration device memory, the serial configuration device only allows you to change the bits in the memory from 1 to 0 since it is serial flash memory. For example, if the initial data in the address 0x070000 is 0xEC and you try to write a data byte 0x79 in the same address using the `write_bytes` operation, the data in that address becomes 0x68 since the `bit0` and `bit4` of the initial data byte 0xEC cannot be changed from 0 to 1.

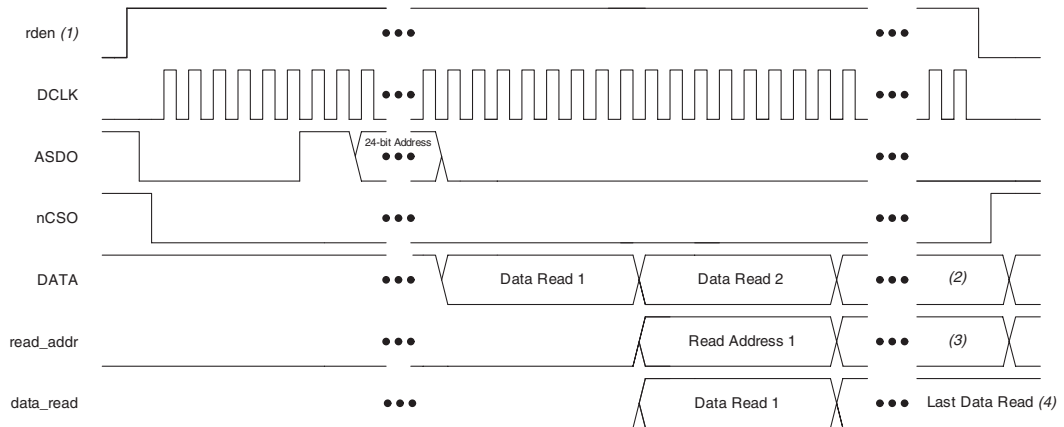
To change bits from 0 to 1, you have to erase the targeted sector memory or erase all memory in the serial configuration device. Therefore, it is important to carefully plan when writing data bytes to the serial configuration device memory.

Additionally, you must be careful not to overwrite the configuration data in the serial configuration device. The FPGA configuration eventually fails if the next configuration cycle initiated is in AS mode. If you

overwrite the configuration data in the EPCS device during user mode, you must replace it with valid configuration data for the selected FPGA before the next configuration cycle is initiated.

Figures 2 through 7 show the timing diagrams for the ASMI Controller.

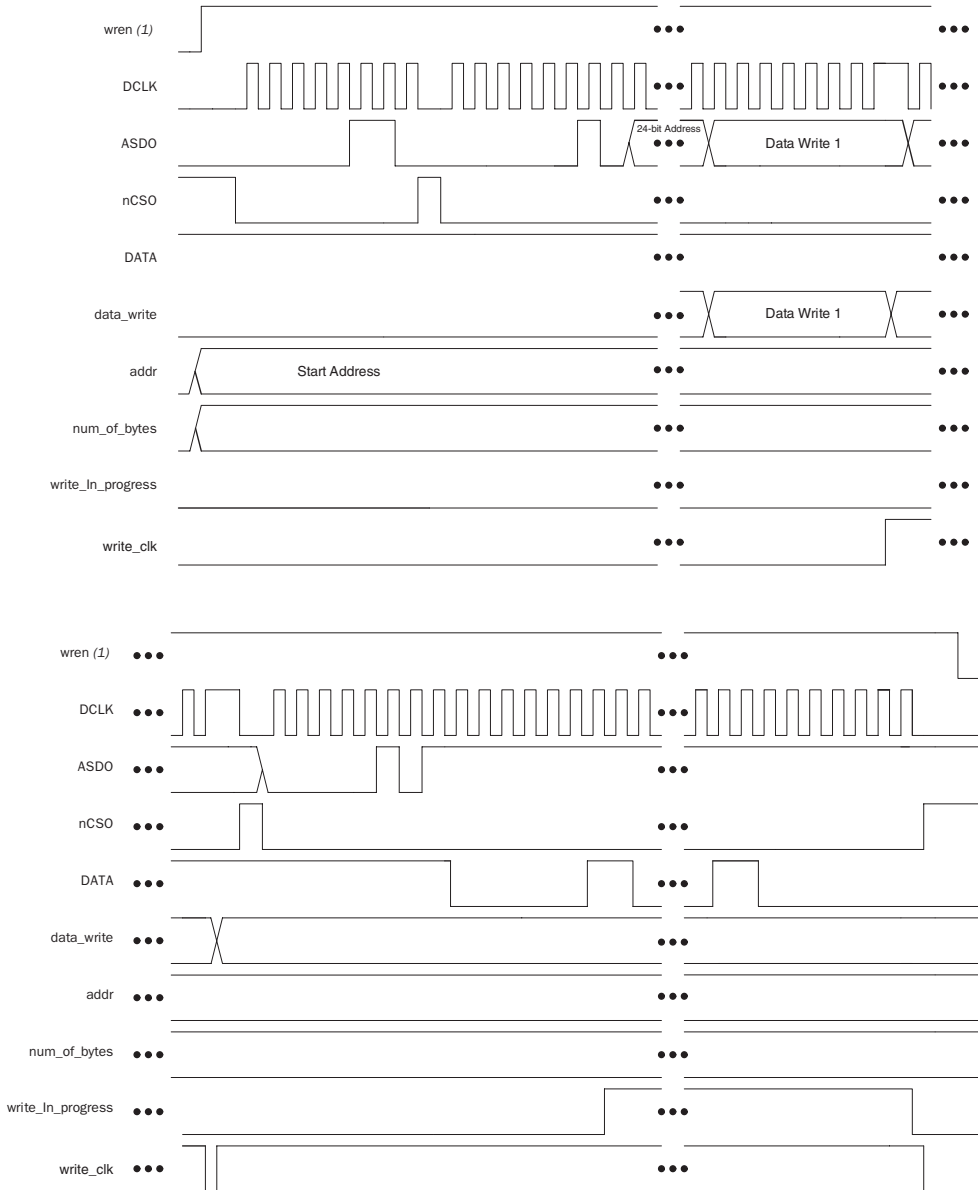
**Figure 2. read\_bytes Timing Diagram**



**Notes to Figure 2:**

- (1) The rden signal is asynchronous with DCLK. When rden is de-asserted, the read\_bytes operation stops immediately.
- (2) The DATA signal is truncated after the rden signal is de-asserted.
- (3) The last read\_addr signal is valid before the rden signal is de-asserted. The read\_addr signal is then reset after the rden signal is de-asserted.
- (4) The last data read is valid before the rden signal is de-asserted. The data\_read value is not reset after the rden signal is de-asserted.

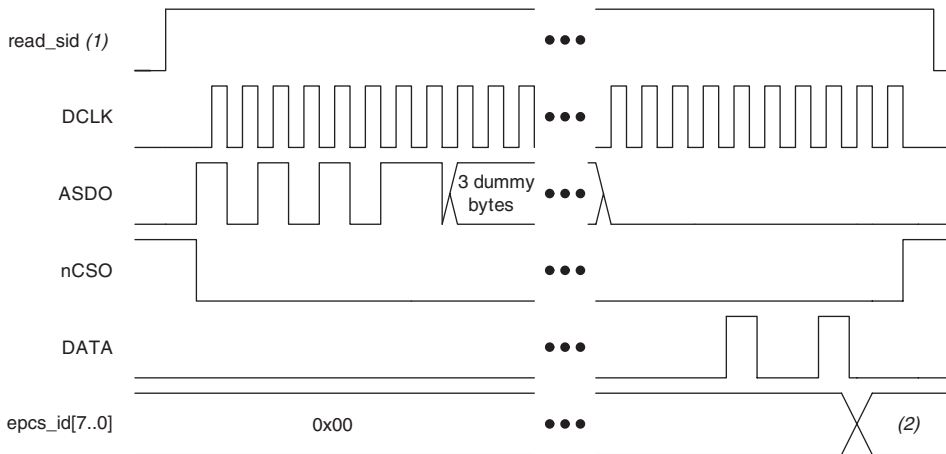
**Figure 3. write\_bytes Timing Diagram**



**Note to Figure 3:**

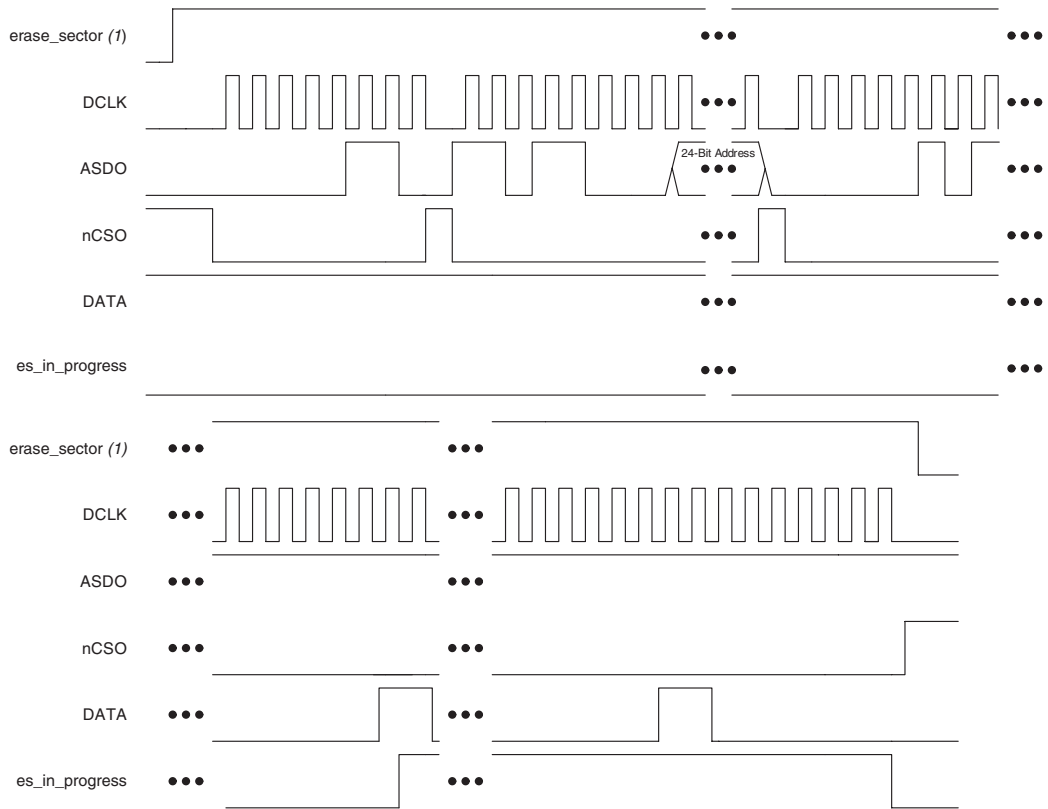
(1) Assert the `wren` signal until the `write_in_progress` signal goes low.



**Figure 4. read\_sid Timing Diagram****Notes to Figure 4:**

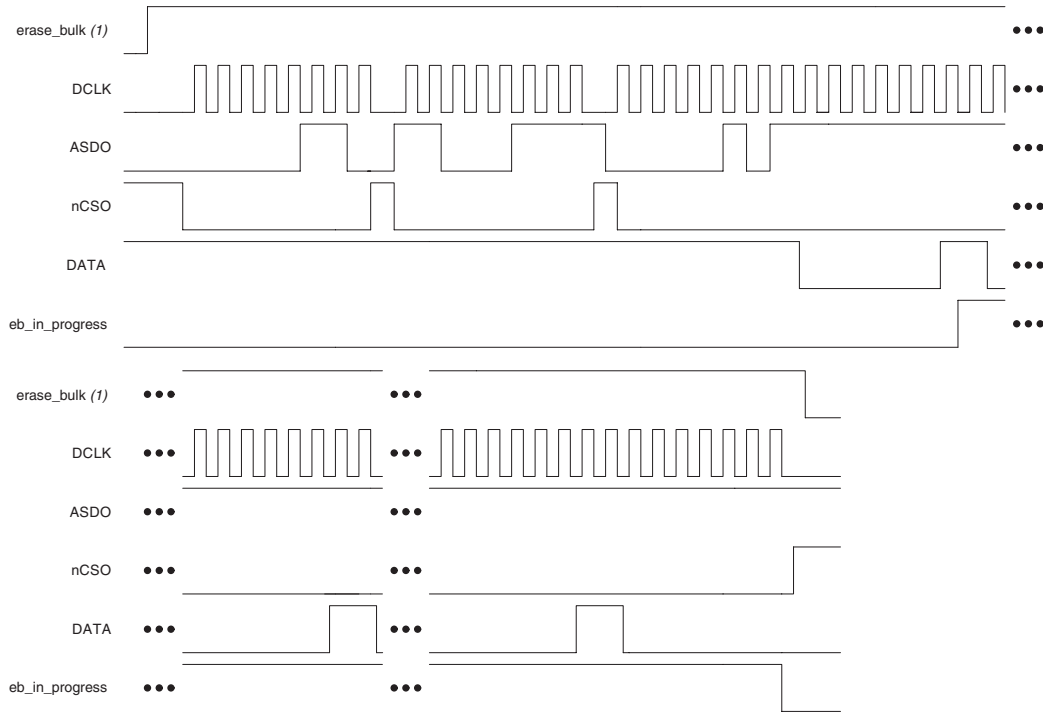
- (1) Assert the read\_sid signal until a valid epcs\_id[7..0] value is obtained.
- (2) The valid epcs\_id[7..0] bus value is 0x10 (EPCS1), 0x12 (EPCS4), 0x14 (EPCS16), or 0x16 (EPCS64).

**Figure 5. erase\_sector Timing Diagram**



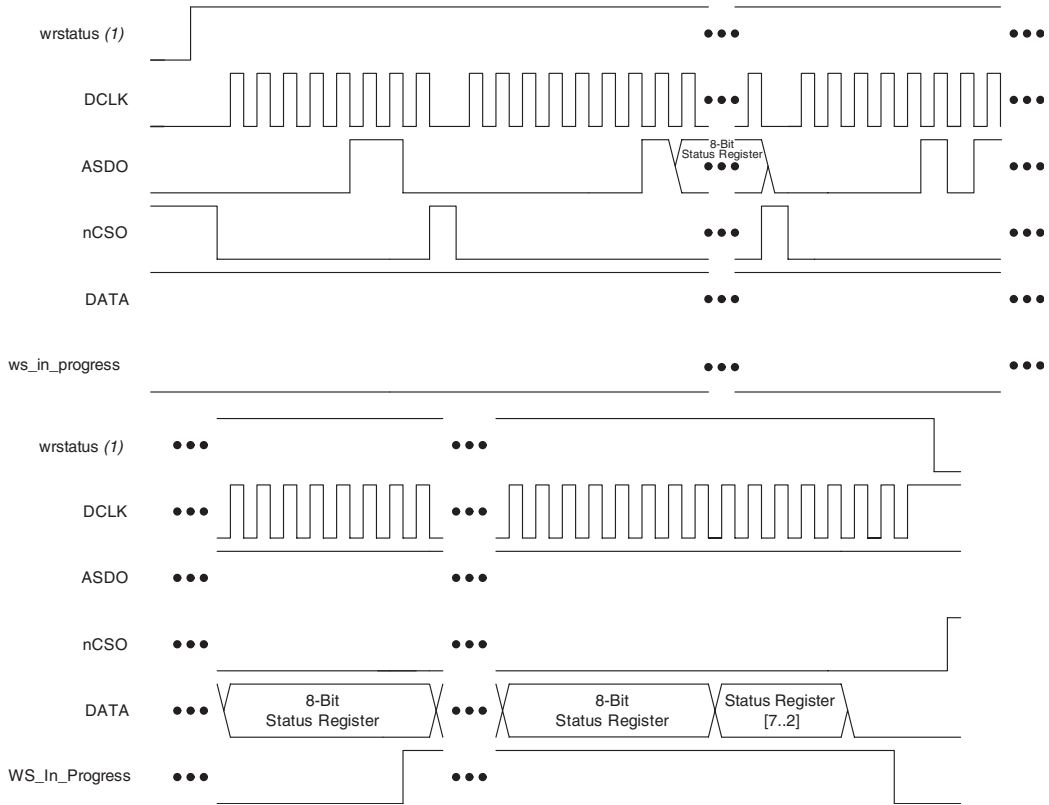
**Note to Figure 5:**

- (1) The rising edge of the `erase_sector` signal triggers the `erase_sector` operation. The `erase_sector` signal is not level-sensitive.

**Figure 6. erase\_bulk Timing Diagram****Note to Figure 6:**

- (1) The rising edge of the erase\_bulk signal triggers the erase\_bulk operation. The erase\_bulk signal is not level-sensitive.

**Figure 7. wrstatus Timing Diagram**



**Note to Figure 7:**

(1) Assert the `wren` signal until the `ws_in_progress` signal goes low.

## How to Read Back All Serial Configuration Data Using the Quartus II Software

You can use the Quartus® II Programmer during the prototyping stage to read back all the data in your serial configuration device, verifying that your design is working.

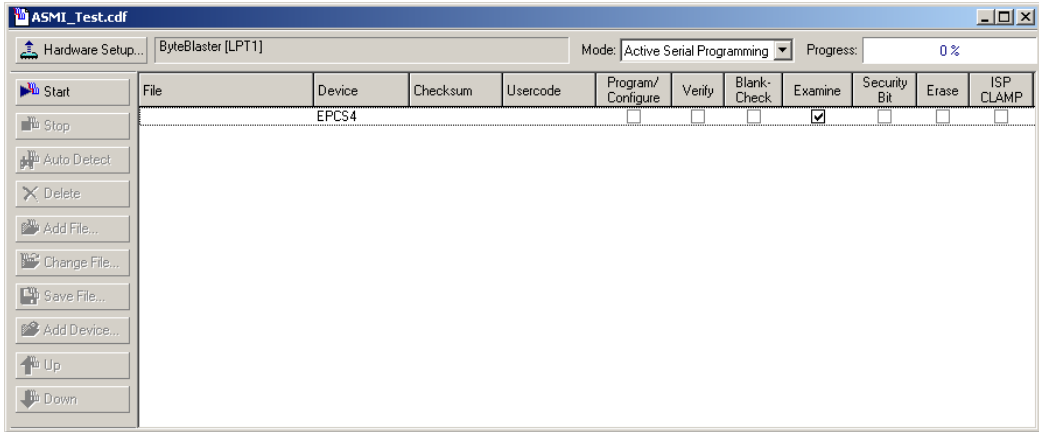
### Step 1: Read Back the Data in Programmer Object File Format

In Step 1, perform the following steps:

1. Open the Quartus II Programmer and connect the download cable (ByteBlaster™ II or USB Blaster™ cable) to the AS programming interface.

2. Add the targeted serial configuration device or add any targeted serial configuration device Programmer Object File (.pof) in the Quartus II Programmer and check the **Examine** box, as shown in [Figure 8](#).
3. Click **Start** to begin the read back process.

**Figure 8. Examine the Data in EPCS Device using the Quartus II Programmer**

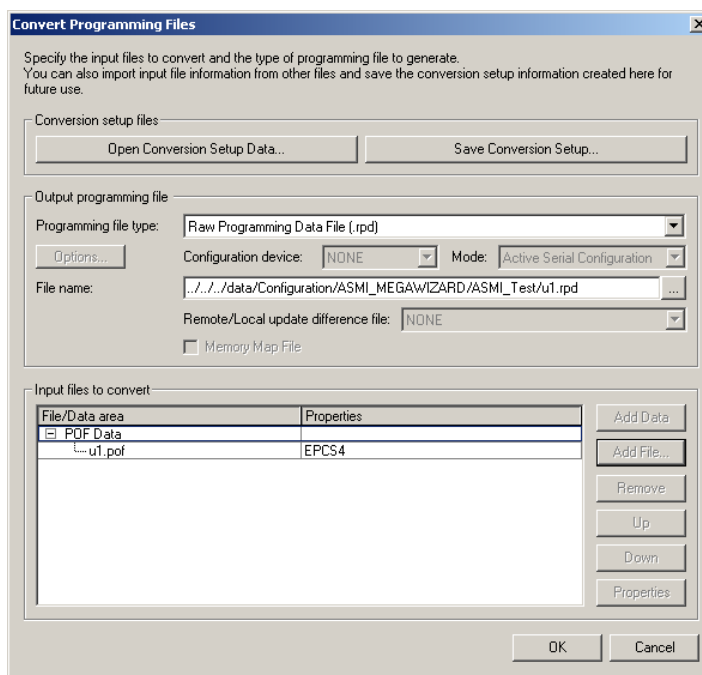


## Step 2: Convert the Read Back POF to a RPD File

In Step 2, perform the following steps:

1. Highlight the read back POF in the Quartus II Programmer and click **Save File...** to save the untitled read back POF in your selected location.
2. Convert the POF to a Raw Programming Data (.rpd) file by choosing **Convert Programming Files** (File menu), as shown in [Figure 9](#). The RPD file contains all the binary data in an serial configuration device POF. You can use a binary editor tool to read the RPD file.

**Figure 9. Converting a Serial Configuration Device POF to a RPD File**



## Quartus II Fitter

The Quartus II Fitter, which is also known as the PowerFit™ Fitter, performs place and route, which is also referred to as “fitting” in the Quartus II software. Using the database created by Quartus II Analysis & Synthesis, the Fitter matches the logic and timing requirements of the

project with the available resources of a device. It assigns each logic function to the best logic cell location for routing and timing, and selects appropriate interconnection paths and pin assignments.



Since the Quartus II Fitter cannot fit more than one asmiblock, do not insert more than one asmiblock in your design during compilation. If you insert more than one asmiblock in your design, a fitter error will occur during compilation in the Quartus II software. Therefore, if you already use a Nios II processor and include the EPCS Controller module in your design, you should not include the `ASMI_CONTROLLER` reference design in your design, or vice versa.

## Conclusion

When using Altera FPGAs, the ASMI reference design allows you to interface on-the-fly with serial configuration devices. Since serial configuration devices have non-volatile memory, you are able to customize this reference design to store your data.

You can use the Quartus II Programmer during the prototyping stage to read back all the data in your serial configuration device and convert the read back POF to a RPD file.



You can also use the SignalTap® II logic analyzer in the Quartus II software to understand additional details about the operations inside the ASMI Controller during the prototyping stage.

## Support

If you have additional questions with the reference design provided, contact Altera Applications at [www.altera.com/mysupport](http://www.altera.com/mysupport).

Copyright © 2005 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, maskwork 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.



I.S. EN ISO 9001