# 16. In-System Updating of Memory and Constants

QII53012-9.0.0

## Introduction

The In-System Memory Content Editor allows you to view and update memories and constants using the JTAG port connection. This chapter explains how to use the Quartus II In-System Memory Content Editor as part of your FPGA design and verification flow.

FPGA designs are growing larger in density and are becoming more complex. Designers and verification engineers require more access to the design that is programmed in the device to quickly identify, test, and resolve issues. The in-system updating of memory and constants capability of the Quartus® II software provides read and write access to in-system FPGA memories and constants through the Joint Test Action Group (JTAG) interface, making it easier to test changes to memory contents in the FPGA while the FPGA is functioning in the end system.

This chapter contains the following sections:

## Overview

The ability to read and update memories and constants in a programmed device provides more insight into and control over your design. The Quartus II In-System Memory Content Editor gives you access to device memories and constants. When used in conjunction with the SignalTap® II Embedded Logic Analyzer, this feature provides the visibility required to debug your design in the hardware lab.

For more information about the SignalTap II Embedded Logic Analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

The ability to read data from memories and constants allows you to quickly identify the source of problems. In addition, the write capabilities allow you to bypass functional issues by writing expected data. For example, if a parity bit in your memory is incorrect, you can use the In-System Content Editor to write the correct parity bit values into your RAM, allowing your system to continue functioning. You can also intentionally write incorrect parity bit values into your RAM to check your design's error handling functionality.

The Quartus II software offers a portfolio of on-chip debugging tools. For an overview and comparison of all tools available in the Quartus II software on-chip debugging tool suite, refer to *Section V. In-System Debugging* in volume 3 of the *Quartus II Handbook*.

## Device Megafunction Support

The following tables list the devices and types of memories and constants that are currently supported by the Quartus II software. Table 16–1 lists the types of memory supported by the MegaWizard™ Plug-In Manager and the In-System Memory Content Editor.

**Table 16–1.** MegaWizard Plug-In Manager Support

| Installed Plug-Ins Category | Megafunction Name |
|---|---|
| Gates | LPM_CONSTANT |
| Memory Compiler | RAM: 1-PORT, ROM: 1-PORT |
| Storage | ALTSYNCRAM, LPM_RAM_DQ, LPM_ROM |

Table 16–2 lists support for in-system updating of memory and constants for the Stratix® series, Arria® GX, Cyclone® series, APEX™ II, and APEX 20K device families.

**Table 16–2.** Supported Megafunctions

| Megafunction | Arria GX / Stratix Series | | | Cyclone Series | APEX II | APEX 20K |
|---|---|---|---|---|---|---|
| | M512 Blocks | M4K Blocks | MegaRAM Blocks | | | |
| LPM_CONSTANT | Read/ Write | Read/ Write | Read/ Write | Read/ Write | Read/ Write | Read/ Write |
| LPM_ROM | Write | Read/ Write | N/A | Read/ Write | Read/ Write | Write |
| LPM_RAM_DQ | N/A | Read/ Write | Read/ Write | Read/ Write | Read/ Write | N/A *(1)* |
| ALTSYNCRAM (ROM) | Write | Read/ Write | N/A | Read/ Write | N/A | N/A |
| ALTSYNCRAM (Single-Port RAM Mode) | N/A | Read/ Write | Read/ Write | Read/ Write | N/A | N/A |

**Note to Table 16–2:**

(1) Only write-only mode is applicable for this single-port RAM. In read-only mode, use LPM_ROM instead of LPM_RAM_DQ.

## Using In-System Updating of Memory and Constants with Your Design

Using the In-System Updating of Memory and Constants feature requires the following steps:

1. Identify the memories and constants that you want to access.
2. Edit the memories and constants to be run-time modifiable.
3. Perform a full compilation.
4. Program your device.
5. Launch the In-System Memory Content Editor.

## Creating In-System Modifiable Memories and Constants

When you specify that a memory or constant is run-time modifiable, the Quartus II software changes the default implementation. A single-port RAM is converted to dual-port RAM, and a constant is implemented in registers instead of look-up tables (LUTs). These changes enable run-time modification without changing the functionality of your design. For a list of run-time modifiable megafunctions, refer to Table 16–1.

To enable your memory or constant to be modifiable, perform the following steps:

1. On the Tools menu, click **MegaWizard Plug-In Manager**.
2. If you are creating a new megafunction, select **Create a new custom megafunction variation**. If you have an existing megafunction, select **Edit an existing custom megafunction variation**.
3. Make the necessary changes to the megafunction based on the characteristics required by your design, turn on **Allow In-System Memory Content Editor to capture and update content independently of the system clock**, and type a value in the **Instance ID** text box. These parameters can be found on the last page of the wizard for megafunctions that support in-system updating.

   ☞ The Instance ID is a four-character string used to distinguish the megafunction from other in-system memories and constants.

4. Click **Finish**.
5. On the Processing menu, click **Start Compilation**.

If you instantiate a memory or constant megafunction directly using ports and parameters in VHDL or Verilog HDL, add or modify the `lpm_hint` parameter as follows:

In VHDL code, add the following:

```
lpm_hint => "ENABLE_RUNTIME_MOD = YES,
  INSTANCE_NAME = <instantiation name>";
```

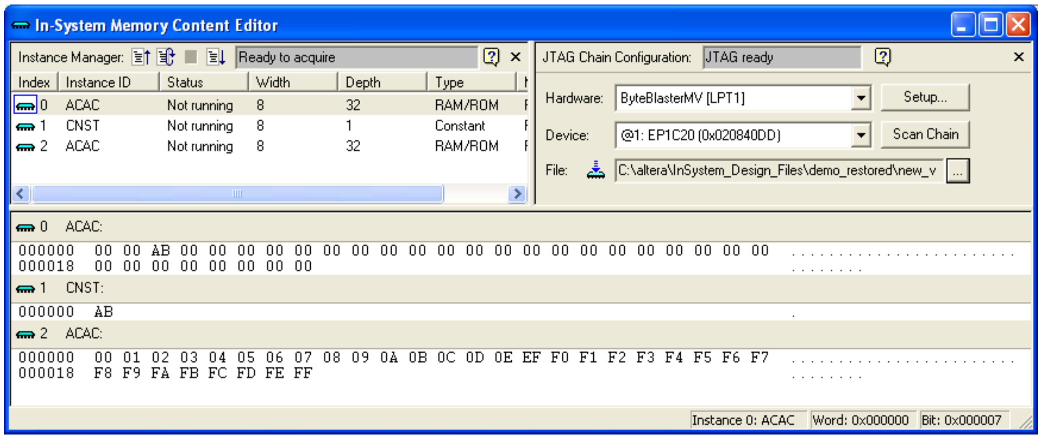
In Verilog HDL code, add the following:

```
defparam <megafunction instance name>.lpm_hint =
  "ENABLE_RUNTIME_MOD = YES,
  INSTANCE_NAME = <instantiation name>";
```

## Running the In-System Memory Content Editor

The In-System Memory Content Editor is separated into the Instance Manager, JTAG Chain Configuration, and the Hex Editor (Figure 16–1).

**Figure 16–1.** In-System Memory Content Editor


In-System Memory Content Editor

Instance Manager: Ready to acquire

| Index | Instance ID | Status | Width | Depth | Type |
|---|---|---|---|---|---|
| 0 | ACAC | Not running | 8 | 32 | RAM/ROM |
| 1 | CNST | Not running | 8 | 1 | Constant |
| 2 | ACAC | Not running | 8 | 32 | RAM/ROM |

JTAG Chain Configuration: JTAG ready

Hardware: ByteBlasterMV [LPT1]

Device: @1: EP1C20 (0x020840DD)

File: C:\altera\InSystem_Design_Files\demo_restored\new_v

```
0  ACAC:
000000  00 00 AB 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ........................
000018  00 00 00 00 00 00 00 00                                                   ........
1  CNST:
000000  AB                                                                        .
2  ACAC:
000000  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E EF F0 F1 F2 F3 F4 F5 F6 F7   ........................
000018  F8 F9 FA FB FC FD FE FF                                                   ........
```

Instance 0: ACAC  Word: 0x000000  Bit: 0x000007

The Instance Manager displays all available run-time modifiable memories and constants in your FPGA device. The JTAG Chain Configuration section allows you to program your FPGA and select the Altera® device in the chain to update.

Using the In-System Memory Content Editor does not require that you open a project. The In-System Memory Content Editor retrieves all instances of run-time configurable memories and constants by scanning the JTAG chain and sending a query to the specific device selected in the JTAG Chain Configuration section.

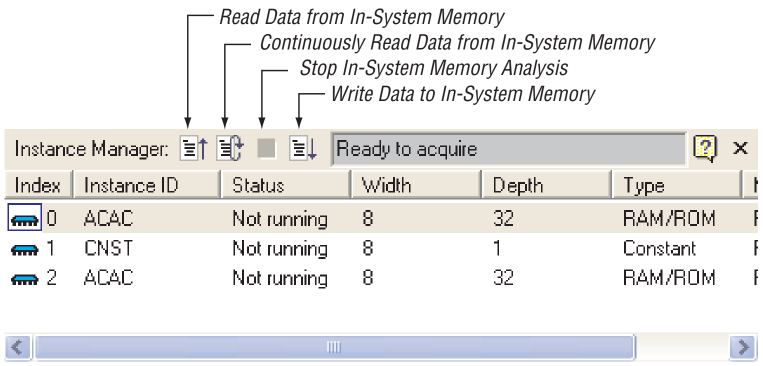
Each In-System Memory Content Editor can access the in-system memories and constants in a single device. If you have more than one device containing in-system configurable memories or constants in a JTAG chain, you can launch multiple In-System Memory Content Editors within the Quartus II software to access the memories and constants in each of the devices.

## Instance Manager

Scan the JTAG chain to update the Instance Manager with a list of all run-time modifiable memories and constants in the design. The Instance Manager displays the Index, Instance, Status, Width, Depth, Type, and Mode of each element in the list.

You can read and write to in-system memory using the Instance Manager, as shown in Figure 16–2.

**Figure 16–2.** Instance Manager Controls


Read Data from In-System Memory
Continuously Read Data from In-System Memory
Stop In-System Memory Analysis
Write Data to In-System Memory

Instance Manager: Ready to acquire

| Index | Instance ID | Status | Width | Depth | Type |
|---|---|---|---|---|---|
| 0 | ACAC | Not running | 8 | 32 | RAM/ROM |
| 1 | CNST | Not running | 8 | 1 | Constant |
| 2 | ACAC | Not running | 8 | 32 | RAM/ROM |

The following buttons are provided in the Instance Manager:

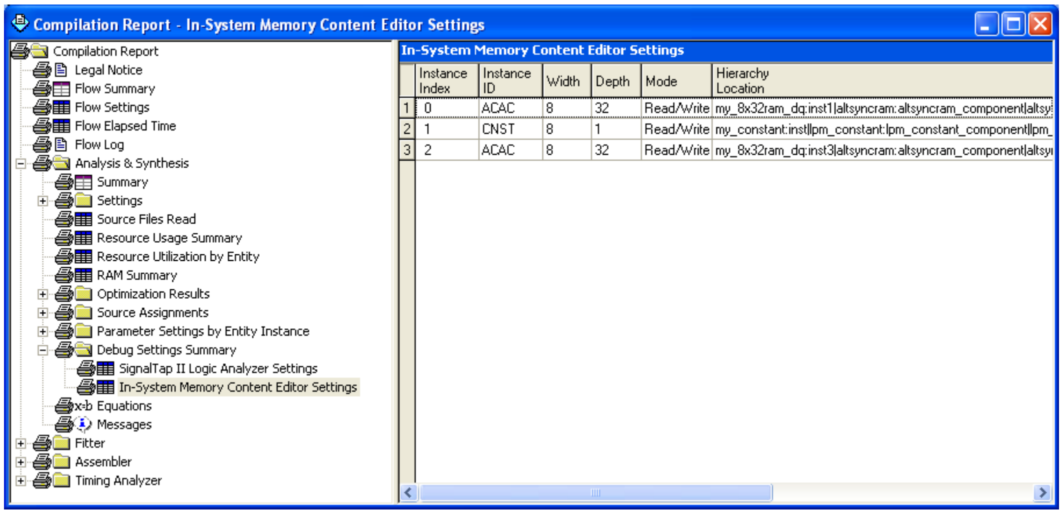
- **Read data from In-System Memory**—Reads the data from the device independently of the system clock and displays it in the Hex Editor
- **Continuously Read Data from In-System Memory**—Continuously reads the data asynchronously from the device and displays it in the Hex Editor
- **Stop In-System Memory Analysis**—Stops the current read or write operation
- **Write Data to In-System Memory**—Asynchronously writes data present in the Hex Editor to the device

In addition to the buttons available in the Instance Manager, you can also read and write data by selecting the command from the Processing menu, or the right button pop-up menu in the Instance Manager or Hex Editor.

The status of each instance is also displayed beside each entry in the Instance Manager. The status indicates if the instance is **Not running**, **Offloading data**, or **Updating data**. The health monitor provides useful information about the status of the editor.

The Quartus II software assigns a different index number to each in-system memory and constant to distinguish between multiple instances of the same memory or constant function. View the **In-System Memory Content Editor Settings** section of the compilation report to match an index with the corresponding instance ID (Figure 16–3).

**Figure 16–3.** Compilation Report In-System Memory Content Editor Settings Section



In-System Memory Content Editor Settings

| | Instance Index | Instance ID | Width | Depth | Mode | Hierarchy Location |
|---|---|---|---|---|---|---|
| 1 | 0 | ACAC | 8 | 32 | Read/Write | my_8x32ram_dq:inst1\|altsyncram:altsyncram_component\|altsy |
| 2 | 1 | CNST | 8 | 1 | Read/Write | my_constant:inst\|lpm_constant:lpm_constant_component\|lpm_ |
| 3 | 2 | ACAC | 8 | 32 | Read/Write | my_8x32ram_dq:inst3\|altsyncram:altsyncram_component\|altsy |

## Editing Data Displayed in the Hex Editor

You can edit the data read from your in-system memories and constants displayed in the Hex Editor by typing values directly into the editor or by importing memory files.

To modify the data displayed in the Hex Editor, click a location in the editor and type or paste in the new data. The new data appears in blue, indicating modified data that has not been written into the FPGA. On the Edit menu, choose **Value**, and click **Fill with 0's**, **Fill with 1's**, **Fill with Random Values**, or **Custom Fills to** update a block of data by selecting a block of data.

## Importing and Exporting Memory Files

The In-System Memory Content editor allows you specify a file to import and export data values to and from memories that have the In-System Updating feature enabled. Importing from a data files enables you to quickly load an entire memory image. Exporting to a data file enables you to save the contents of the memory for future use and analysis.

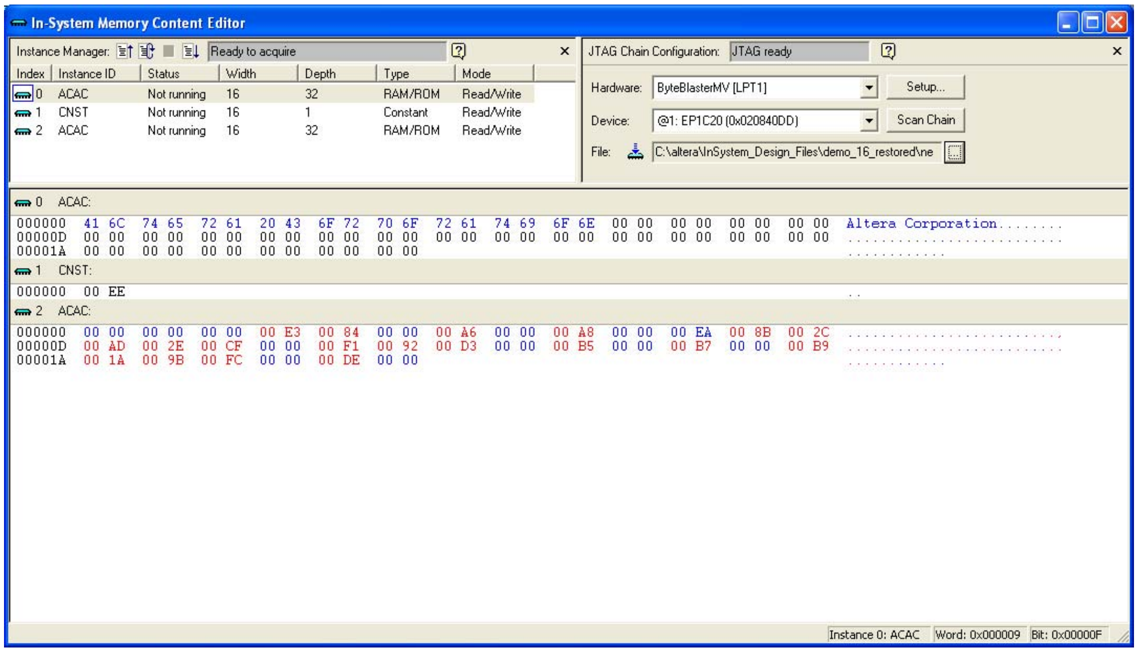
To import a file to memory using the In-System Memory Content Editor, select the memory or constant that you want to target from the instance manager. From the Edit menu, click **Import Data from File**, and specify the data file that you want to load to the targeted memory or constant. You can only import a memory file that is in either a Hexadecimal (Intel-Format) file (**.hex**) or memory initialization file (**.mif**) format.

Similarly, to export the contents of memory to a file using the In-System Memory Content Editor, select the memory or constant that you want to target from the instance manager. From the Edit menu, click **Export Data from File**, and specify the file name to which you want to save the data. You can export data to a **.hex**, **.mif**, Verilog Value Change Dump file (**.vcd**), or RAM Initialization file (**.rif**) format.

## Viewing Memories and Constants in the Hex Editor

For each instance of an in-system memory or constant, the Hex Editor displays data in hexadecimal representation and ASCII characters (if the word size is a multiple of 8 bits). The arrangement of the hexadecimal numbers depends on the dimensions of the memory. For example, if the word width is 16 bits, the Hex Editor displays data in columns of words that contain columns of bytes (Figure 16–4).

**Figure 16–4.** Editing 16-Bit Memory Words Using the Hex Editor


In-System Memory Content Editor

Instance Manager: Ready to acquire

| Index | Instance ID | Status | Width | Depth | Type | Mode |
|---|---|---|---|---|---|---|
| 0 | ACAC | Not running | 16 | 32 | RAM/ROM | Read/Write |
| 1 | CNST | Not running | 16 | 1 | Constant | Read/Write |
| 2 | ACAC | Not running | 16 | 32 | RAM/ROM | Read/Write |

JTAG Chain Configuration: JTAG ready

Hardware: ByteBlasterMV [LPT1]

Device: @1: EP1C20 (0x020840DD)

File: C:\altera\InSystem_Design_Files\demo_16_restored\ne

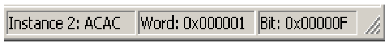
```
0  ACAC:
000000  41 6C  74 65  72 61  20 43  6F 72  70 6F  72 61  74 69  6F 6E  00 00  00 00  00 00  00 00  Altera Corporation........
00000D  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  00 00  ..........................
00001A  00 00  00 00  00 00  00 00  00 00  00 00                                                      ............
1  CNST:
000000  00 EE                                                                                          ..
2  ACAC:
000000  00 00  00 00  00 00  00 E3  00 84  00 00  00 A6  00 00  00 A8  00 00  00 EA  00 8B  00 2C  .........................,
00000D  00 AD  00 2E  00 CF  00 00  00 F1  00 92  00 D3  00 00  00 B5  00 00  00 B7  00 00  00 B9  ..........................
00001A  00 1A  00 9B  00 FC  00 00  00 DE  00 00                                                      ............
```

Instance 0: ACAC | Word: 0x000009 | Bit: 0x00000F

Unprintable ASCII characters are represented by a period (`.`). The color of the data changes as you perform reads and writes. Data displayed in black indicates the data in the Hex Editor was the same as the data read from the device. If the data in the Hex Editor changes color to red, the data previously shown in the Hex Editor was different from the data read from the device.

As you analyze the data, you can use the cursor and the status bar to quickly identify the exact location in memory. The status bar is located at the bottom of the In-System Memory Content Editor and displays the selected instance name, word position, and bit offset (Figure 16–5).

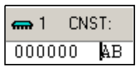
**Figure 16–5.** Status Bar in the In-System Memory Content Editor


Instance 2: ACAC | Word: 0x000001 | Bit: 0x00000F

The bit offset is the bit position of the cursor within the word. In the following example, a word is set to be 8-bits wide.

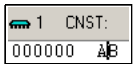
With the cursor in the position shown in Figure 16–6, the word location is 0x0000 and the bit position is 0x0007.

**Figure 16–6.** Hex Editor Cursor Positioned at Bit 0×0007


```
1  CNST:
000000  AB
```

With the cursor in the position shown in Figure 16–7, the word location remains 0x0000 but the bit position is 0x0003.

**Figure 16–7.** Hex Editor Cursor Positioned at Bit 0×0003



## Scripting Support

The In-System Memory Content Editor supports reading and writing of memory contents via a Tcl script or Tcl commands entered at a command prompt. For detailed information about scripting command options, refer to the Quartus II command-line and Tcl API Help browser.

To run the Help browser, type the following command at the command prompt:

```
quartus_sh --qhelp ↵
```

The *Quartus II Scripting Reference Manual* includes the same information.

For more information about Tcl scripting, refer to the *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*. For more information about command-line scripting, refer to the *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*.

The commonly used commands for the In-System Memory Content Editor are as follows:

- Reading from memory:

```
read_content_from_memory
[-content_in_hex]
-instance_index <instance index>
-start_address <starting address>
-word_count <word count>
```

- Writing to memory:

```
write_content_to_memory
```

- Save memory contents to file:

```
save_content_from_memory_to_file
```

- Update memory contents from File:

```
update_content_to_memory_from_file
```

For descriptions about the command options and scripting examples, refer to the Tcl API Help Browser and the *Quartus II Scripting Reference Manual*.

## Programming the Device Using the In-System Memory Content Editor

If you make changes to your design, you can program the device from within the In-System Memory Content Editor. To program the device, follow these steps:

1. On the Tools menu, click **In-System Memory Content Editor.**
2. In the **JTAG Chain Configuration** panel of the In-System Memory Content Editor, select the SRAM object file (**.sof**) that includes the modifiable memories and constants.

3. Click **Scan Chain**.
4. In the **Device** list, select the device you want to program.
5. Click **Program Device**.

### Example: Using the In-System Memory Content Editor with the SignalTap II Embedded Logic Analyzer

The following scenario describes how you can use the In-System Updating of Memory and Constants feature with the SignalTap II Embedded Logic Analyzer to efficiently debug your design in-system. Although both the In-System Content Editor and the SignalTap II Embedded Logic Analyzer use the JTAG communication interface, you are able to use them simultaneously.

After completing your FPGA design, you find that the characteristics of your FIR filter design are not as expected.

1. To locate the source of the problem, change all your FIR filter coefficients to be in-system modifiable and instantiate the SignalTap II Embedded Logic Analyzer.
2. Using the SignalTap II Embedded Logic Analyzer to tap and trigger on internal design nodes, you find the FIR filter to be functioning outside of the expected cut-off frequency.
3. Using the **In-System Memory Content Editor**, you check the correctness of the FIR filter coefficients. Upon reading each coefficient, you discover that one of the coefficients is incorrect.
4. Because your coefficients are in-system modifiable, you update the coefficients with the correct data using the **In-System Memory Content Editor**.

In this scenario, you are able to quickly locate the source of the problem using both the In-System Memory Content Editor and the SignalTap II Embedded Logic Analyzer. You are also able to verify the functionality of your device by changing the coefficient values before modifying the design source files.

An extension to this example would be to modify the coefficients with the In-System Memory Content Editor to vary the characteristics of the FIR filter (for example, filter attenuation, transition bandwidth, cut-off frequency, and windowing function).

## Conclusion

The In-System Updating of Memory and Constants feature provides access into a device for efficient debug in a hardware lab. You can use In-System Updating of Memory and Constants with the SignalTap II Embedded Logic Analyzer to maximize the visibility into an Altera FPGA. By increasing visibility and access to internal logic of the device, you can identify and resolve problems with your design more easily.

## Referenced Documents

This chapter references the following documents:

- *Command-Line Scripting* chapter in volume 2 of the *Quartus II Handbook*
- *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*

- *Quartus II Scripting Reference Manual*
- *Tcl Scripting* chapter in volume 2 of the *Quartus II Handbook*

## Document Revision History

Table 16–3 shows the revision history of this chapter.

**Table 16–3.** Document Revision History

| Date and Version | Changes Made | Summary of Changes |
|---|---|---|
| March 2009 v9.0.0 | No change to content. | Updated for the Quartus II software version 9.0 release. |
| November 2008 v8.1.0 | Changed to 8-1/2 x 11 page size. No change to content. | Updated for the Quartus II software version 8.1 release. |
| May 2008 v8.0.0 | ■ Added reference to Section V. In-System Debugging in volume 3 of the Quartus II Handbook on page 16-1. ■ Removed references to the Mercury device, as it is now considered to be a “Mature” device ■ Added links to referenced documents throughout document ■ Minor editorial updates | Updated for the Quartus II software version 8.0 release. |

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.