
SmartDebug for Software v11.7

User's Guide

NOTE: PDF files are intended to be viewed on the printed page; links and cross-references in this PDF file may point to external files and generate an error when clicked. **View the online help included with software to enable all linked content.**



Table of Contents

Welcome to SmartDebug	5
Introduction to SmartDebug.....	6
Getting Started with SmartDebug.....	7
Using SmartDebug with SmartFusion and Fusion	7
Using SmartDebug with SmartFusion2, IGLOO2, and RTG4	7
Create Standalone SmartDebug Project.....	7
SmartDebug User Interface	10
Standalone SmartDebug User Interface	10
Programming Connectivity and Interface	11
View Device Status (SmartFusion2, IGLOO2, and RTG4 Only)	15
Embedded Flash Memory (NVM) Content Dialog Box (SmartFusion2 and IGLOO2 Only)....	18
Debugging	20
Debug FPGA Array (SmartFusion2, IGLOO2, and RTG4 Only)	20
Hierarchical View.....	20
Netlist View	22
Probe Grouping (Active Probes Only)	26
Memory Blocks (SmartFusion2, IGLOO2, and RTG4).....	30
Probe Insertion (Post-Layout) - SmartFusion2, IGLOO2, and RTG4.....	32
Debug SERDES – Loopback Test	36
Debug SERDES – PRBS Test	37
Debug SERDES – PHY Reset	39
Inspect Device Dialog Box (SmartFusion, Fusion, and ProASIC3 Only)	39
Device Status Report (SmartFusion and Fusion Only)	40
Embedded Flash Memory (NVM) Content Dialog Box (SmartFusion and Fusion Only)	41
Embedded Flash Memory: Browse Retrieved Data (SmartFusion and Fusion Only)	43
Embedded Flash Memory: Compare Memory Client (SmartFusion and Fusion Only).....	44
FlashROM Content Dialog Box (Fusion and SmartFusion Only)	45
Analog Block Configuration Dialog Box (SmartFusion and Fusion Only)	46
SmartDebug Tcl Commands	49
SmartDebug Tcl Support (SmartFusion2, IGLOO2, and RTG4).....	49
Device Debug / SmartDebug Tcl Commands (SmartFusion, IGLOO, ProASIC3, and Fusion Only)	51
add_to_probe_group (SmartFusion2, IGLOO2, and RTG4)	52
check_flash_memory.....	52
compare_analog_config	54
compare_flashrom_client	54
compare_memory_client	55
create_probe_group (SmartFusion2, IGLOO2, and RTG4)	56
delete_active_probe	56
ddr_read (SmartFusion2, IGLOO2, and RTG4)	56
ddr_write (SmartFusion2, IGLOO2, and RTG4).....	57

export_smart_debug_data (SmartFusion2, IGLOO2, and RTG4)	58
load_active_probe_list.....	59
loopback_test (SmartFusion2, IGLOO2, and RTG4)	59
move_to_probe_group (SmartFusion2, IGLOO2, and RTG4)	60
prbs_test (SmartFusion2, IGLOO2, and RTG4).....	60
read_active_probe (SmartFusion2, IGLOO2, and RTG4).....	61
read_analog_block_config.....	62
read_device_status	62
read_id_code	63
read_flashrom.....	63
read_flash_memory.....	64
read_lsrsm (SmartFusion2, IGLOO2, and RTG4)	65
read_usram (SmartFusion2 and IGLOO2)	66
recover_flash_memory.....	66
remove_from_probe_group (SmartFusion2, IGLOO2, and RTG4).....	67
sample_analog_channel.....	67
save_active_probe_list.....	69
select_active_probe (SmartFusion2, IGLOO2, and RTG4)	69
serdes_lane_reset	70
serdes_read_register (SmartFusion2, IGLOO2, and RTG4)	71
serdes_write_register (SmartFusion2, IGLOO2, and RTG4)	72
set_debug_device	72
set_debug_programmer	73
set_live_probe (SmartFusion2, IGLOO2, and RTG4)	73
ungroup (SmartFusion2, IGLOO2, and RTG4)	74
unset_live_probe	74
write_active_probe (SmartFusion2, IGLOO2, and RTG4)	75
write_lsrsm (SmartFusion2, IGLOO2, and RTG4)	75
write_usram (SmartFusion2, IGLOO2, and RTG4).....	76
Solutions to Common Issues Using SmartDebug	77
Embedded Flash Memory (NVM) - Failure when Programming/Verifying	77
Analog System Not Working as Expected.....	77
ADC Not Sampling the Correct Value	77
Frequently Asked Questions	79
How do I unlock the device security so I can debug?	79
How do I export a report?	79
How do I generate diagnostic reports for my target device?	79
Where can I find files to compare my contents/settings?.....	80
What is a UFC file? What is an EFC file?.....	80
Is my FPGA fabric enabled?.....	80
Is my Embedded Flash Memory (NVM) programmed?.....	81
How do I display Embedded Flash Memory (NVM) content in the Client partition?	81
How do I know if I have Embedded Flash Memory (NVM) corruption?	81
Why does Embedded Flash Memory (NVM) corruption happen?.....	81

How do I recover from Embedded Flash Memory corruption?.....	82
What is a JTAG IR-Capture value?	82
What does the ECC1/ECC2 error mean?	82
How can I tell if my FlashROM is programmed?	82
Can I compare serialization data?	82
Can I tell what security options are programmed in my device?	82
Is my analog system configured?	82
How do I interpret data in the Device Status report?.....	83
Device Status Report: User Info	83
Device Status Report: Device State	84
Device Status Report: Analog Block	85
Device Status Report: Factory Data	86
Device Status Report: Security.....	86
How do I interpret data in the Flash Memory (NVM) Status Report?	87
Product Support.....	89
Customer Service	89
Customer Technical Support Center	89
Technical Support.....	89
Website.....	89
Contacting the Customer Technical Support Center.....	89
ITAR Technical Support	90

Welcome to SmartDebug

Introduction to SmartDebug

Design debug is a critical phase of FPGA design flow. Microsemi's SmartDebug tool complements design simulation by allowing verification and troubleshooting at the hardware level. SmartDebug provides access to non-volatile memory (eNVM), SRAM, SERDES, DDR controller, and probe capabilities. Microsemi SmartFusion2 System-on-chip (SoC) field programmable gate array (FPGA), IGLOO2 FPGA, and RTG4 FPGA devices have built-in probe logic that greatly enhance the ability to debug logic elements within the device. SmartDebug accesses the built-in probe points through the Active Probe and Live Probe features, which enables designers to check the state of inputs and outputs in real-time without re-layout of the design.

Use Models

SmartDebug can be run in two modes:

- Integrated mode from the Libero Design Flow
- Standalone mode

Integrated Mode

When run in integrated mode from Libero, SmartDebug can access all design and programming hardware information. No extra setup step is required. In addition, the Probe Insertion feature is available in Debug FPGA Array.

To open SmartDebug in the Libero Design Flow window expand **Debug Design** and double-click **SmartDebug Design**.

Standalone Mode

SmartDebug can be installed separately in the setup containing FlashPro, FlashPro Express, and Job Manager. This provides a lean installation that includes all the programming and debug tools to be installed in a lab environment for debug. In this mode, SmartDebug is launched outside of the Libero Design Flow. When launched in standalone mode, you must go through SmartDebug project creation and import a Design Debug Data Container (DDC) file, exported from Libero, to access all the debug features in the supported devices.

Note: In standalone mode, the Probe Insertion feature is not available in FPGA Array Debug, as it requires incremental routing to connect the user net to the specified I/O.

Standalone Mode Use Model Overview

The main use model for standalone SmartDebug requires users to generate the DDC file from Libero and import it into a SmartDebug project to gain full access to the device debug features. Alternatively, SmartDebug can be used without a DDC file with a limited feature set.

Supported Families, Programmers, and Operating Systems

Programming and Debug: SmartFusion2, IGLOO2, and RTG4

Programming only: ProAsic3/E, IGLOO, Fusion, and SmartFusion

Programmers: FlashPRO3, FlashPRO4, and FlashPRO5

Operating Systems: Windows XP, Windows 7, and RHEL 6.x

Note: Debug for ProAsic3/E, IGLOO, Fusion, and SmartFusion devices are available via FlashPro. Also refer to "Inspect Device" in the "Device Debug User Interface" section of the *FlashPro User's Guide*.

Getting Started with SmartDebug

This topic introduces the basic elements and features of SmartDebug. If you are already familiar with the user interface, proceed to the Solutions to Common Issues Using SmartDebug or Frequently Asked Questions sections.

SmartDebug enables you to use JTAG to interrogate and view embedded silicon features and device status (FlashROM, Security Settings, Embedded Flash Memory (NVM) and Analog System). SmartDebug is available as a part of the FlashPro programming tool.

See [Using SmartDebug](#) and [Using SmartDebug with SmartFusion2, IGLOO2, and RTG4](#) for an overview of the use flow.

You can use the debugger to:

- [Get device status and view diagnostics](#)
- [Use the FlashROM debug GUI to read out and compare content](#)
- [Use the Embedded Flash Memory Debug GUI to read out and compare your content with your original files](#)
- [Use the Analog System Debug to read out and compare your analog block configuration with your original file](#)

Using SmartDebug with SmartFusion and Fusion

Note: SmartDebug is referred to as Device Debug in some older families.

The most common flow for SmartDebug is:

1. [Start FlashPro](#). If necessary, [create a new project](#).
2. Set up your FlashPro Project with or without a PDB file. If you are in single-device mode you will need a PDB file. You can create a PDB file in both Single Device and Chain mode.

With a PDB, you will get additional information such as FlashROM and Embedded Flash Memory partitions when debugging the silicon features. Best practice is to use a PDB with a valid-use design to start a debug session.

3. Select the target device from your chain and click **Inspect Device**.
4. Click **Device Status** to get device status and check for issues
5. Examine individual silicon features (FlashROM, Embedded Flash Memory Block and Analog System) on the device.

Using SmartDebug with SmartFusion2, IGLOO2, and RTG4

The most common flow for SmartDebug is:

1. [Create your design](#). You must have a FlashPro programmer connected to use SmartDebug.
2. Expand **Debug Design** and double-click **Smart Debug Design** in the Design Flow window. SmartDebug opens for your target device.
3. Click **View Device Status** to view the device status report and check for issues.
4. Examine individual silicon features, such as FPGA debug.

Create Standalone SmartDebug Project

A standalone SmartDebug project can be configured in two ways:

- Import DDC files exported from Libero
- Construct Automatically

From the SmartDebug main window, click **Project** and choose **New Project**. The Create SmartDebug Project dialog box opens.

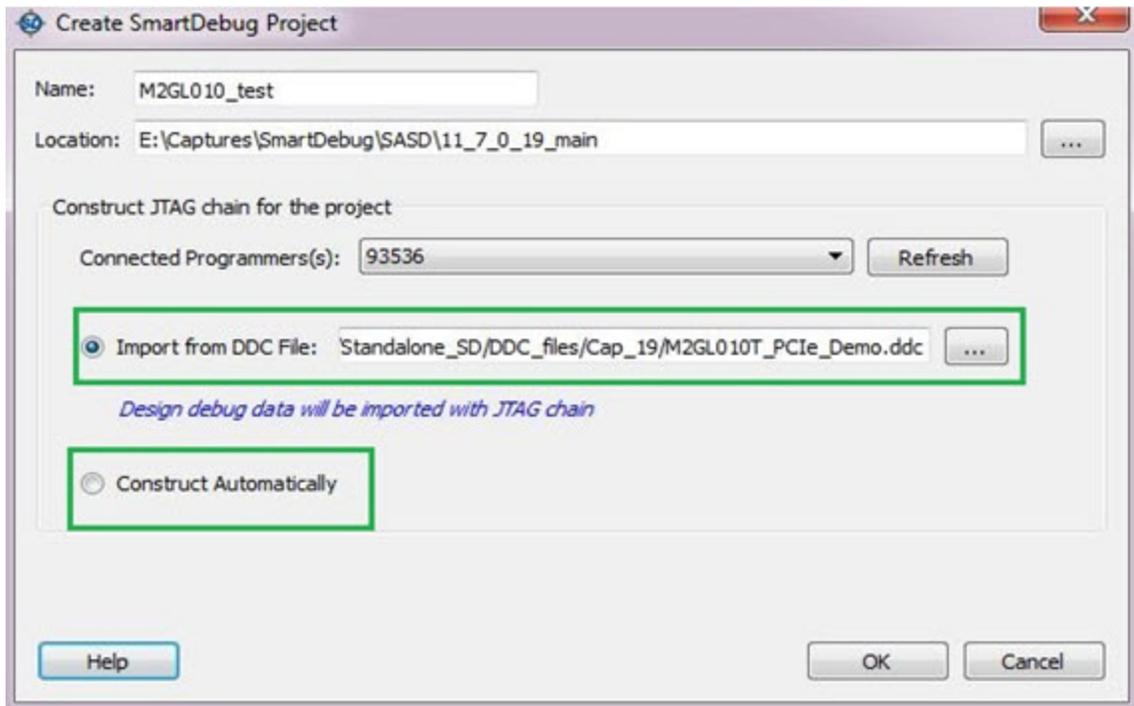


Figure 1 · Create SmartDebug Project Dialog Box

Import from DDC File (created from Libero)

When you select the **Import from DDC File** option in the Create SmartDebug Project dialog box, the Design Debug Data of the target device and all hardware and JTAG chain information present in the DDC file exported in Libero are automatically inherited by the SmartDebug project. The programming file information loaded onto other Microsemi devices in the chain, including ProAsic3/E, SmartFusion, and Fusion devices, are also transferred to the SmartDebug project.

Debug data is imported from the DDC file (created through Export SmartDebug Data in Libero) into the debug project, and the devices are configured using data from the DDC file.

Construct Automatically

When you select the **Construct Automatically** option, a debug project is created with all the devices connected in the chain for the selected programmer. This is equivalent to Construct Chain Automatically in FlashPRO.

Configuring a Generic Device

For Microsemi devices having the same JTAG IDCODE (i.e., multiple derivatives of the same Die—for example, M2S090T, M2S090TS, and so on), the device type must be configured for SmartDebug to enable relevant features for debug. The device can be configured by loading the programming file, by manually selecting the device using Configure Device, or by importing DDC files through Programming Connectivity and Interface. When the device is configured, all debug options are shown.

For debug projects created using Construct Automatically, you can use the following options to debug the devices:

- Load the programming file – Right-click the device in Programming Connectivity and Interface.
- Import Debug Data from DDC file – Right-click the device in Programming Connectivity and Interface.

The appropriate debug features of the targeted devices are enabled after the programming file or DDC file is imported.

Connected FlashPRO Programmers

The drop-down lists all FlashPro programmers connected to the device. Select the programmer connected to the chain with the debug device. At least one programmer must be connected to create a standalone SmartDebug project.

Before a debugging session or after a design change, program the device through Programming Connectivity and Interface.

See Also

[Programming Connectivity and Interface](#)

[View Device Status](#)

[Export SmartDebug Data \(from Libero\)](#)

Import from DDC File (created from Libero)

When you select the **Import from DDC File** option in the Create SmartDebug Project dialog box, the Design Debug Data of the target device and all hardware and JTAG chain information present in the DDC file exported in Libero are automatically inherited by the SmartDebug project. The programming file information loaded onto other Microsemi devices in the chain, including ProAsic3/E, SmartFusion, and Fusion devices, are also transferred to the SmartDebug project.

Debug data is imported from the DDC file (created through Export SmartDebug Data in Libero) into the debug project, and the devices are configured using data from the DDC file.

Construct Automatically

When you select the **Construct Automatically** option, a debug project is created with all the devices connected in the chain for the selected programmer. This is equivalent to Construct Chain Automatically in FlashPRO.

Configuring a Generic Device

For Microsemi devices having the same JTAG IDCODE (i.e., multiple derivatives of the same Die—for example, M2S090T, M2S090TS, and so on), the device type must be resolved for SmartDebug to enable relevant features for debug. The device can be resolved by loading the programming file, manually selecting the device using Configure Device, or by importing DDC files through Programming Connectivity and Interface. When the device is resolved, all debug options are shown.

For debug projects created using Construct Automatically, you can use the following options to debug the devices:

- Load the programming file – Right-click the device in Programming Connectivity and Interface.
- Import Debug Data from DDC file – Right-click the device in Programming Connectivity and Interface.

The appropriate debug features of the targeted devices are enabled after the programming file or DDC file is imported.

SmartDebug User Interface

Standalone SmartDebug User Interface

You can start standalone SmartDebug from the Libero installation folder or from the FlashPRO installation folder.

Windows:

<Libero Installation folder>/Designer/bin/sdebug.exe

<FlashPRO Installation folder>/bin/sdebug.exe

Linux :

<Libero Installation folder>/ bin/sdebug

<FlashPRO Installation folder>/bin/sdebug

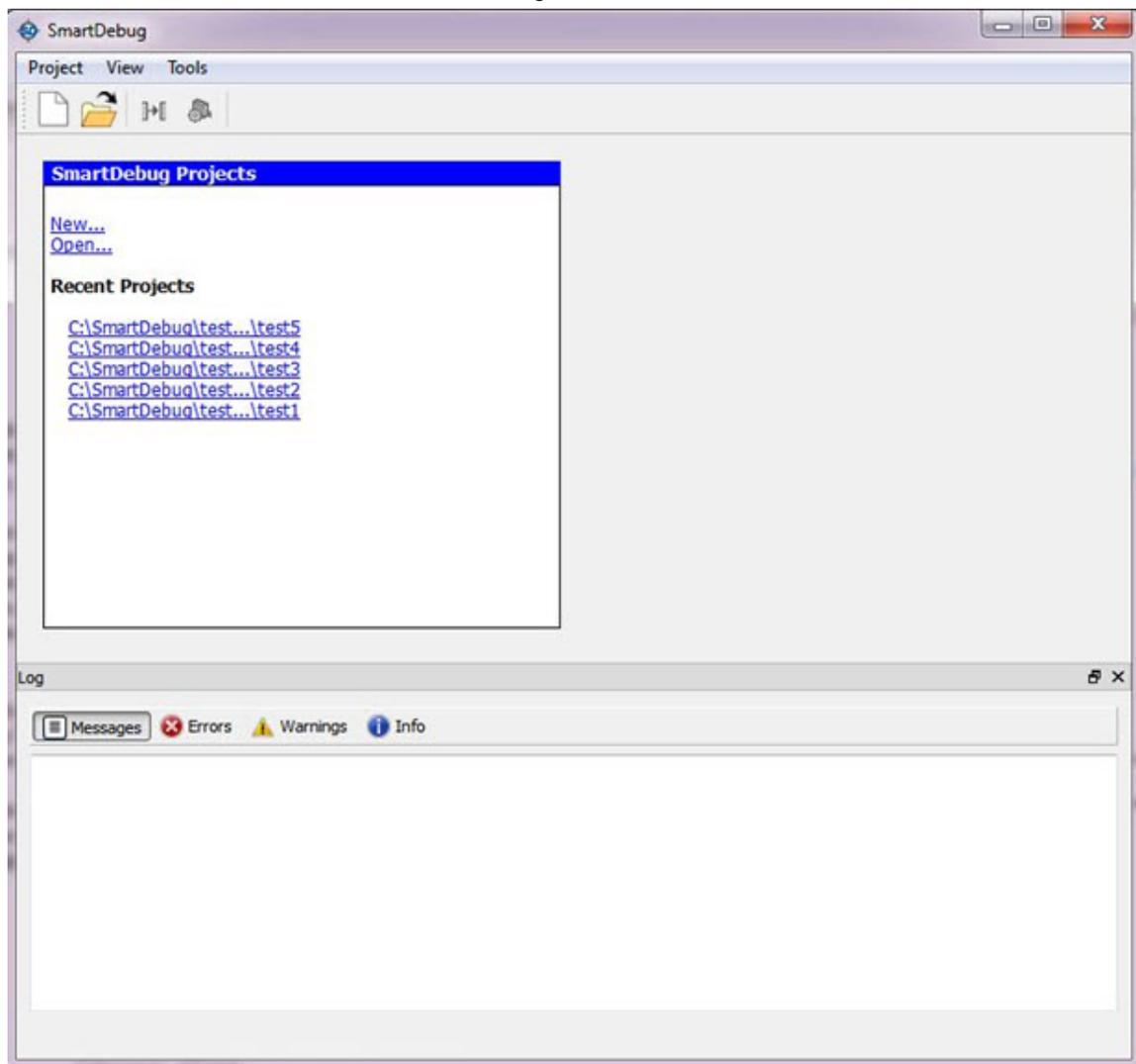


Figure 2 · Standalone SmartDebug Main Window

Project Menu

The Project menu allows you do the following:

- Create new SmartDebug projects (**Project > New Project**)
- Open existing debug projects (**Project > Open Project**)
- Execute SmartDebug-specific Tcl scripts (**Project > Execute Script**)
- Export SmartDebug-specific commands to a script file (**Project > Export Script File**)
- See a list of recent SmartDebug projects (**Project > Recent Projects**).

Log Window

SmartDebug displays the Log window by default when it is invoked. To suppress the Log window display, click the View menu and toggle **View Log**.

The Log window has four tabs:

- Messages** – displays standard output messages
- Errors** – displays error messages
- Warnings** – displays warning messages
- Info** – displays general information

Tools Menu

The Tools menu includes Programming Connectivity and Interface and Programmer Settings options, which are enabled after creating or opening a SmartDebug project.

Programming Connectivity and Interface

To open the Programming Connectivity and Interface dialog box, from the standalone SmartDebug Tools menu, choose **Programming Connectivity and Interface**. The Programming Connectivity and Interface dialog box displays the physical chain from TDI to TDO.

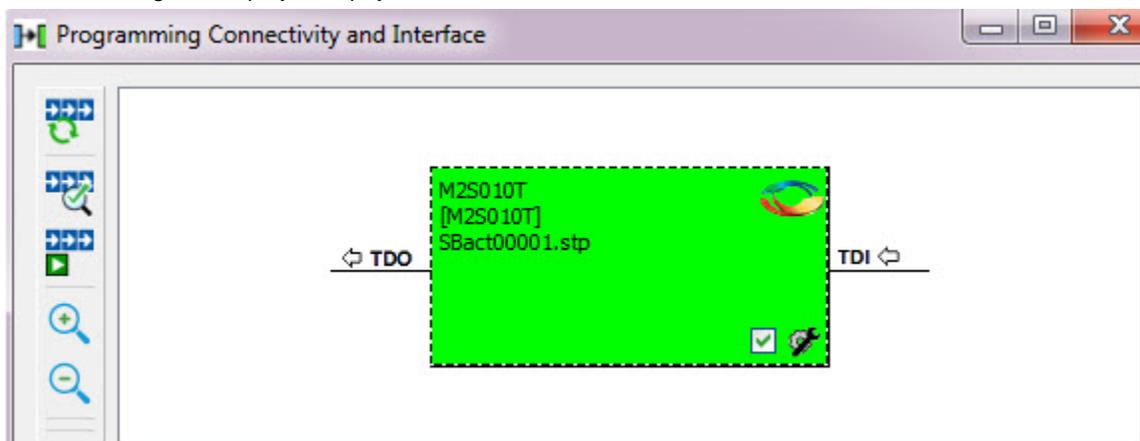


Figure 3 · Programming Connectivity and Interface Dialog Box – Project created using Import from DDC File

All devices in the chain are disabled by default when a standalone SmartDebug project is created using the **Construct Automatically** option in the Create SmartDebug Project dialog box.

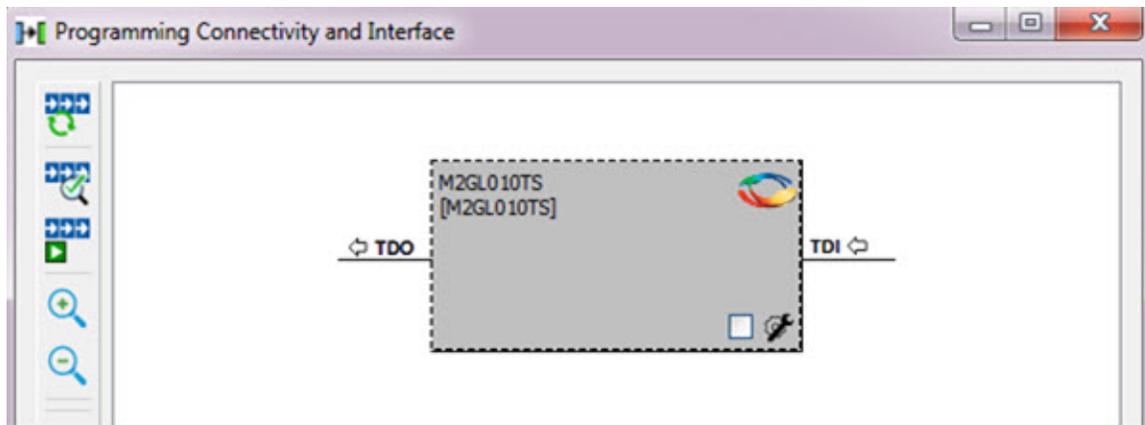


Figure 4 · Programming Connectivity and Interface window – Project created using Construct Automatically

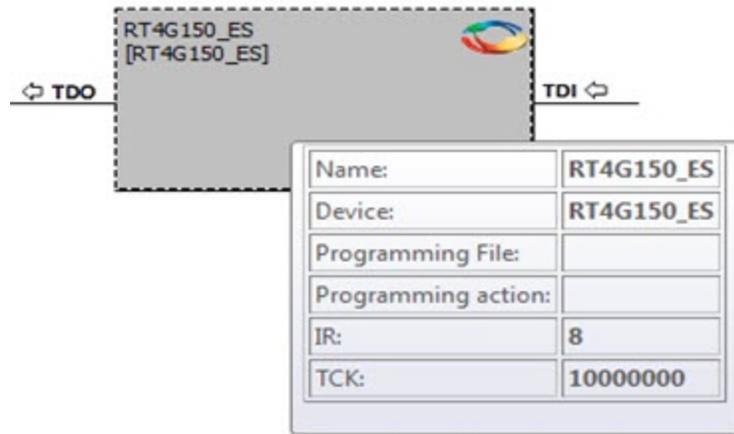
The Programming Connectivity and Interface dialog box includes the following actions:

- **Construct Chain Automatically** - Automatically construct the physical chain.
Running Auto-Construct in Programming Connectivity and Interface removes all existing debug/programming data included using DDC/programming files. The project is the same as a new project created using the Construct Automatically option.
- **Scan and Check Chain** – Scan the physical chain connected to the programmer and check if it matches the chain constructed in the scan chain block diagram.
- **Run Programming Action** – Option to program the device with the selected programming procedure.
When two devices are connected in the chain, the programming actions are independent of the device. For example, if M2S090 and M2GL010 devices are connected in the chain, and the M2S090 device is to be programmed and the M2GL010 device is to be erased, both actions can be done at the same time using the Run Programming Action option.
- **Zoom In** – Zoom into the scan chain block diagram.
- **Zoom Out** – Zoom out of the scan chain block diagram.

Hover Information

The device tooltip displays the following information if you hover your cursor over a device in the scan chain block diagram:

- **Name:** User-specified device name. This field indicates the unique name specified by the user in the Device Name field in Configure Device (right-click **Properties**).
- **Device:** Microsemi device name.
- **Programming File:** Programming file name.
- **Programming action:** The programming action selected for the device in the chain when a programming file is loaded.
- **IR:** Device instruction length.
- **TCK:** Maximum clock frequency in MHz to program a specific device; standalone SmartDebug uses this information to ensure that the programmer operates at a frequency lower than the slowest device in the chain.

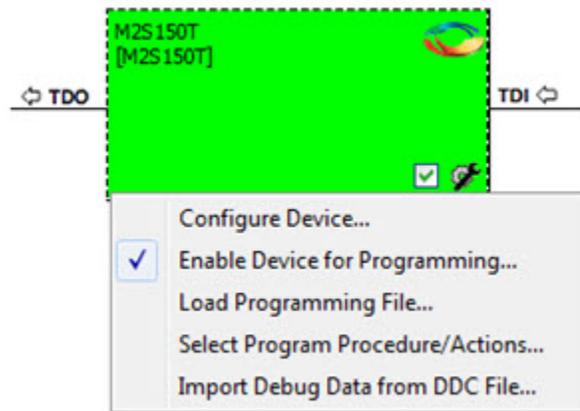


Device Chain Details

The device within the chain has the following details:

- User-specified device name
- Device name
- Programming file name
- Programming action – Select **Enable Device for Programming** to enable the device for programming. Enabled devices are green, and disabled devices are grayed out.

Right-click Properties



Configure Device - Ability to reconfigure the device.

- **Family and Die:** The device can be explicitly configured from the Family, Die drop-down.
- **Device Name:** Editable field for providing user-specified name for the device.

Enable Device for Programming - Select to enable the device for programming. Enabled devices are shown in green, and disabled devices are grayed out.

Load Programming File - Load the programming file for the selected device.

Select Programming Procedure/Actions- Option to select programming action/procedures for the devices connected in the chain.

- **Actions:** List of programming actions for your device.

- **Procedures:** Advanced option; enables you to customize the list of recommended and optional procedures for the selected action.

Import Debug Data from DDC File - Option to import debug data information from the DDC file.

The DDC file selected for import into device must be created for a compatible device. When the DDC file is imported successfully, all current device debug data is removed and replaced with debug data from the imported DDC file.

The JTAG Chain configuration from the imported DDC file is ignored in this option.

If a programming file is already loaded into the device prior to importing debug data from the DDC file, the programming file content is replaced with the content of the DDC file (if programming file information is included in the DDC file).

Debug Context Save

Debug context refers to the user selections in debug options such as Debug FPGA Array, Debug SERDES, and View Flash Memory Content. In standalone SmartDebug, the debug context of the current session is saved or reset depending on the user actions in Programming Connectivity and Interface.

The debug context of the current session is retained for the following actions in Programming Connectivity and Interface:

- Enable Device for Programming
- Select Programming Procedure/Actions
- Scan and Check Chain
- Run Programming Action

The debug context of the current session is reset for the following actions in Programming Connectivity and Interface:

- Auto Construct – Clears all the existing debug data. You need to reimport the debug data from DDC file.
- Import Debug Data from DDC file
- Configure Device – Renaming the device in the chain
- Configure Device – Family/Die change
- Load Programming File

Selecting Devices for Debug

Standalone SmartDebug provides an option to select the devices connected in the JTAG chain for debug. The device debug context is not saved when another debug device is selected.

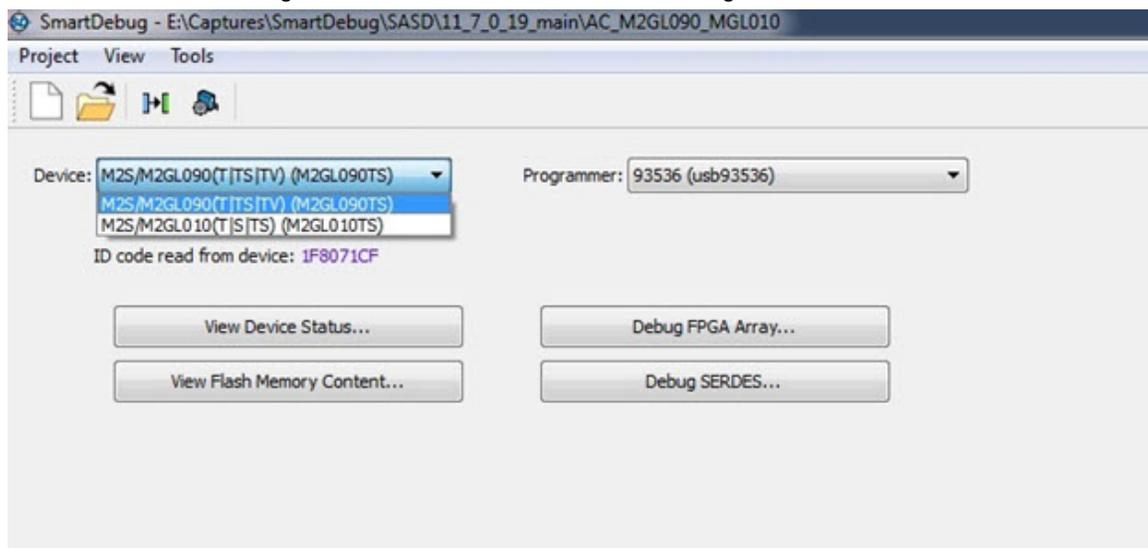


Figure 5 · Select Devices

View Device Status (SmartFusion2, IGLOO2, and RTG4 Only)

Click **View Device Status** in the standalone SmartDebug main window to display the Device Status Report. The Device Status Report is a complete summary of IDCode, device certificate, design information, programming information, digest, and device security information. Use this dialog box to save or print your information for future reference.

Note: This information is available for SmartFusion2 and IGLOO2 devices only. For RTG4 devices, View Device Status displays IDCode information only.

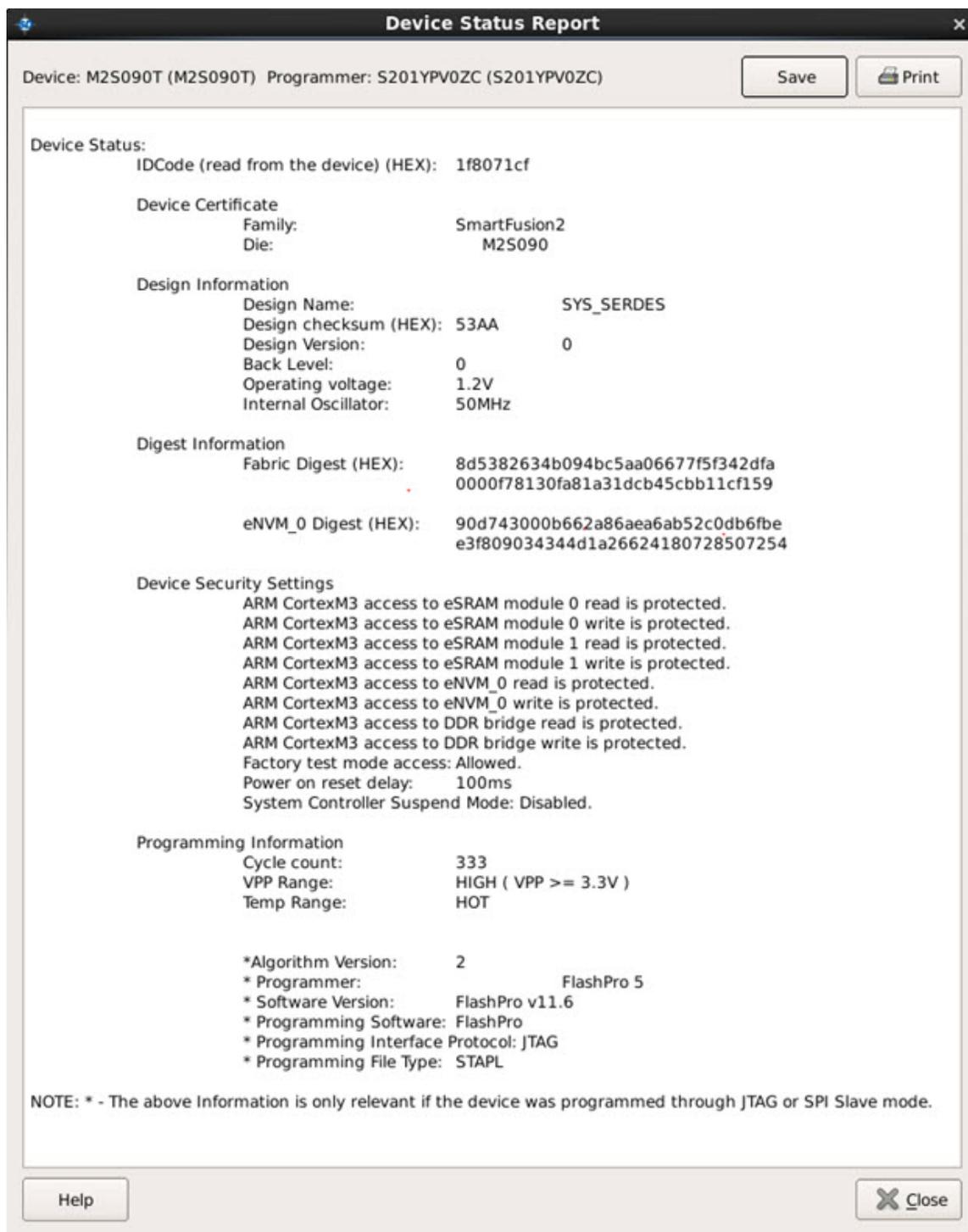


Figure 6 · Device Status Report

IdCode

IDCode read from the device under debug.

Device Certificate

Device certificate displays Family and Die information if device certificate is installed on the device.

If the device certificate is not installed on the device, a message indicating that the device certificate may not have been installed is shown.

Design Information

Design Information displays the following:

- Design Name
- Design Checksum
- Design Version
- Back Level
- Operating Voltage
- Internal Oscillator

Digest Information

Digest Information displays Fabric Digest, eNVM_0 Digest and eNVM_1 Digest (for M2S090 and M2S150 devices only) computed from the device during programming. eNVM digest is shown when eNVM is used in the design.

Device Security Settings

Device Security Settings indicate the following:

- Factory test mode access
- Power on reset delay
- System Controller Suspend Mode

In addition, if custom security options are used, Device Security Settings indicate:

- User Lock segment is protected
- User Pass Key 1/2 encrypted programming is enforced for the FPGA Array
- User Pass Key 1/2 encrypted programming is enforced for the eNVM_0 and eNVM_1
- SmartDebug write access to Active Probe and AHB mem space
- SmartDebug read access to Active Probe, Live Probe & AHB mem space
- UJTAG access to fabric

Programming Information

Programming Information displays the following:

- Cycle Count
- VPP Range
- Temp Range
- Algorithm Version
- Programmer
- Software Version
- Programming Software
- Programming Interface Protocol
- Programming File Type

Embedded Flash Memory (NVM) Content Dialog Box (SmartFusion2 and IGLOO2 Only)

The NVM content dialog box is divided into two sections:

- View content of Flash Memory pages (as shown in the figure below)
- Check page status and identify if a page is corrupted or if the write count limit has exceeded the 10-year retention threshold

Choose the eNVM page contents to be viewed by specifying the page range (i.e., start page and the end page) and click **Read from Device** to view the values.

You must click **Read from Device** each time you specify a new page range to update the view.

Specify a page range if you wish to examine a specific set of pages. In the Retrieved Data View, you can enter an Address value (such as 0010) in the Go to Address field and click the corresponding button to go directly to that address. Page Status information appears to the right.

Contents of Page Status

- ECC1 detected and corrected
- ECC2 detected
- Write count of the page
- If write count has exceeded the threshold
- If the page is used as ROM (first page lock)
- Overwrite protect (second page lock)
- Flash Freeze state (deep power down)

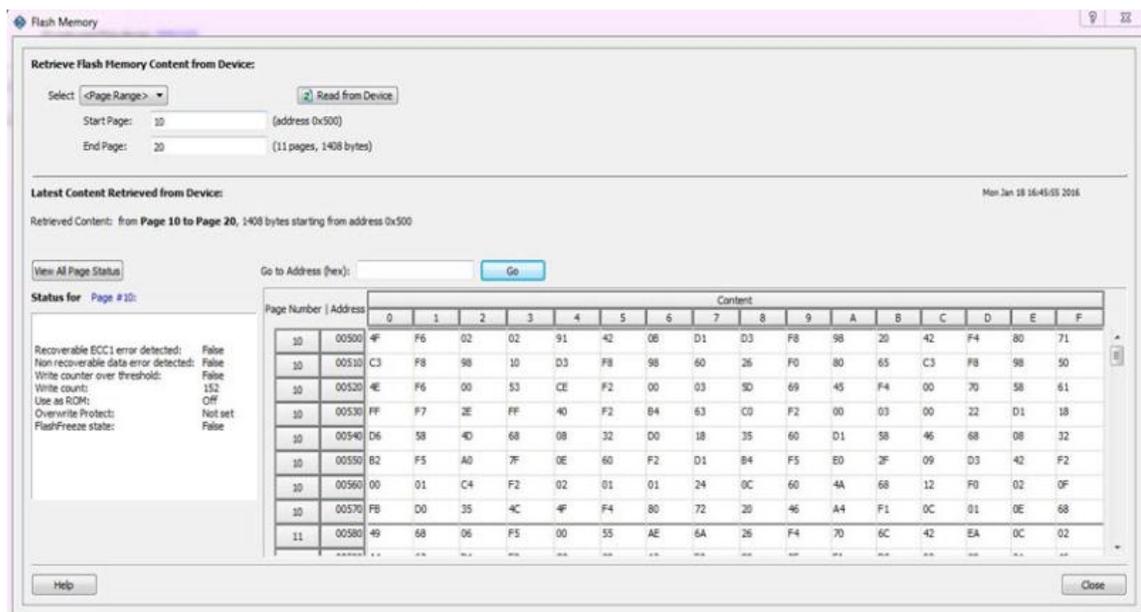


Figure 7 · Flash Memory Dialog Box for a SmartFusion2 Device (SmartDebug)

The page status gets updated when you:

- Click Page Range
- Click a particular cell in the retrieved eNVM content table
- Scroll pages from the keyboard using the Up and Down arrow keys
- Click Go to Address (hex)

The retrieved data table displays the content of the page range selection. If content cannot be read (for example, pages are read-protected, but security has been erased or access to eNVM private sectors), Read from Device reports an error.

Click **View Detailed Status** for a detailed report on the page range you have selected.

For example, if you want to view a report on pages 1-3, set the Start Page to 1, set the End Page to 3, and click **Read from Device**. Then click **View Detailed Status**. The figure below is an example of the data for a specific page range.

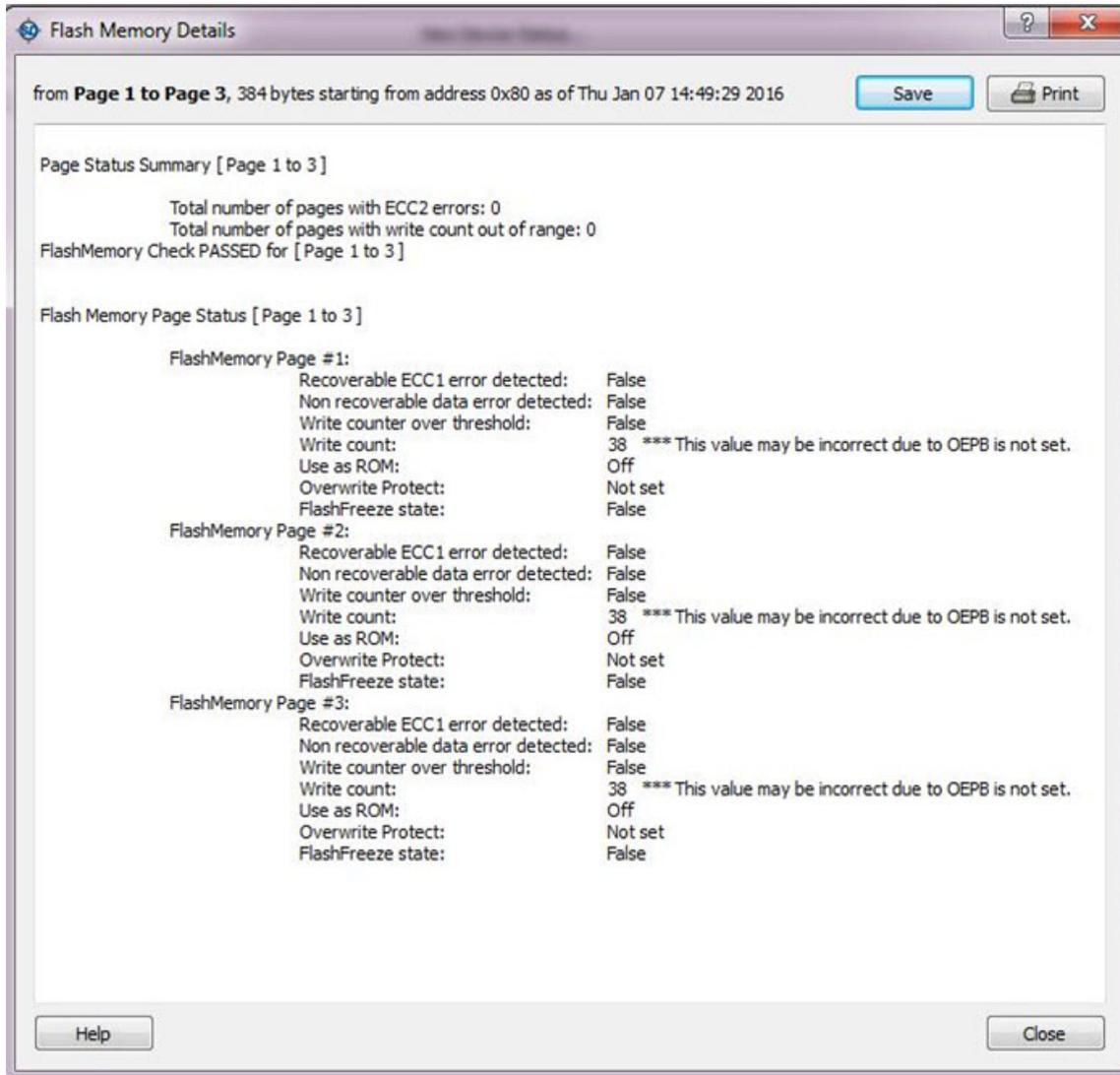


Figure 8 · Flash Memory Details Dialog Box (SmartDebug)

Debugging

Debug FPGA Array (SmartFusion2, IGLOO2, and RTG4 Only)

In the Debug FPGA Array dialog box, you can view your Live Probes, Active Probes, and Memory Blocks, and Insert Probes (Probe Insertion).

The Debug FPGA Array dialog box includes the following four tabs:

- [Live Probes](#)
- [Active Probes](#)
- [Memory Blocks](#)
- [Probe Insertion](#)

Hierarchical View

The Hierarchical View lets you view the instance level hierarchy of the design programmed on the device and select the signals to add to the Live Probes, Active Probes, and Probe Insertion tabs in the Debug FPGA Array dialog box.

- **Instance** – Displays the probe points available at the instance level.
- **Primitives** – Displays the lowest level of probeable points in the hierarchy for the corresponding component —i.e., leaf cells (hard macros on the device).

You can expand the hierarchy tree to see lower level logic.

Signals with the same name are grouped automatically into a bus that is presented at instance level in the instance tree.

The probe points can be added by selecting any instance or the leaf level instance in the Hierarchical View. Adding an instance adds all the probe able points available in the instance to Live Probes, Active Probes, and Probe Insertion.

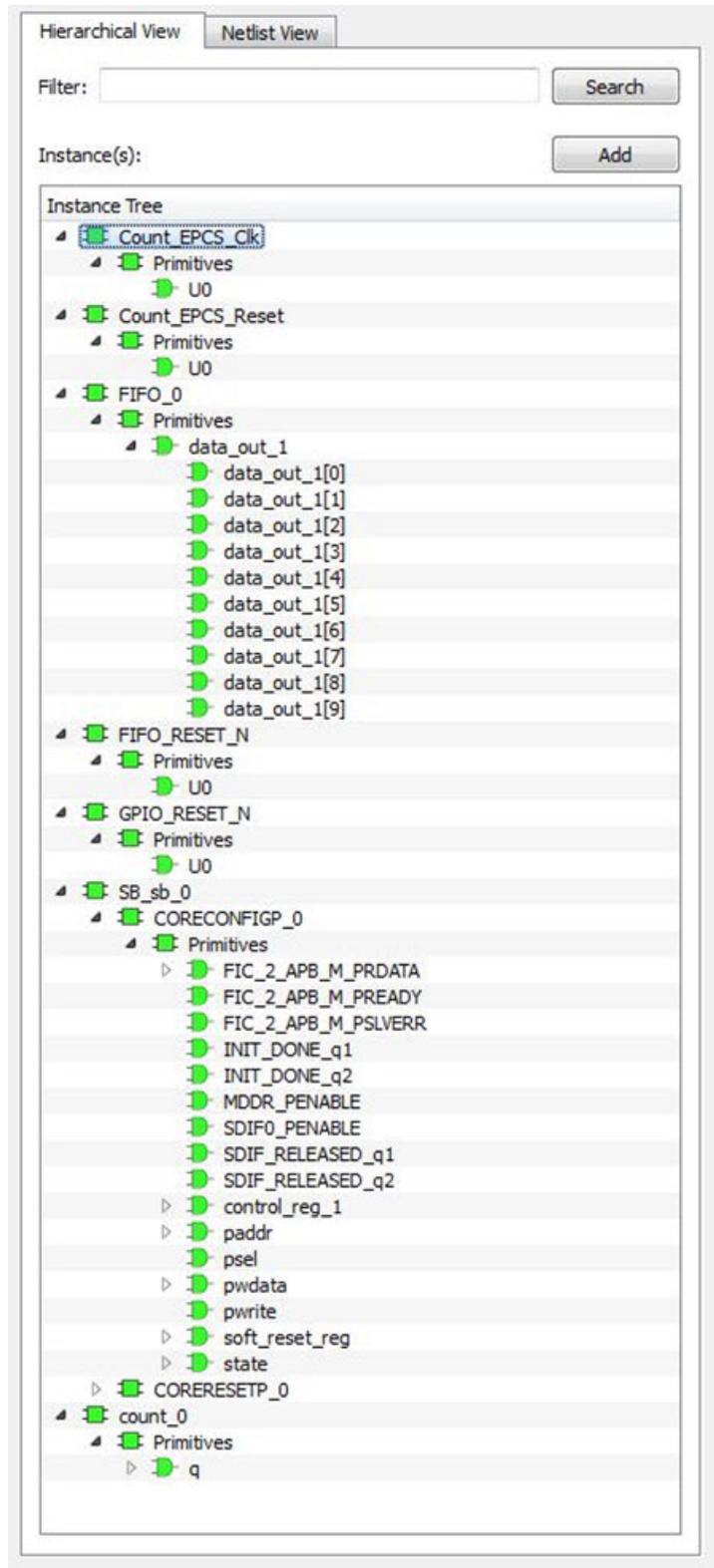


Figure 9 - Hierarchical View

Search

In Live Probes, Active Probes, and the Probe Insertion UI, a search option is available in the Hierarchical View. You can use wildcard characters such as * or ? in the search column for wildcard matching.

Probe points of leaf level instances resulting from a search pattern can only be added to Live Probes, Active Probes, and the Probe Insertion UI. You cannot add instances of search results in the Hierarchical View.

Netlist View

The Netlist View displays a flattened net view of all the probe able points present in the design, along with the associated cell type.

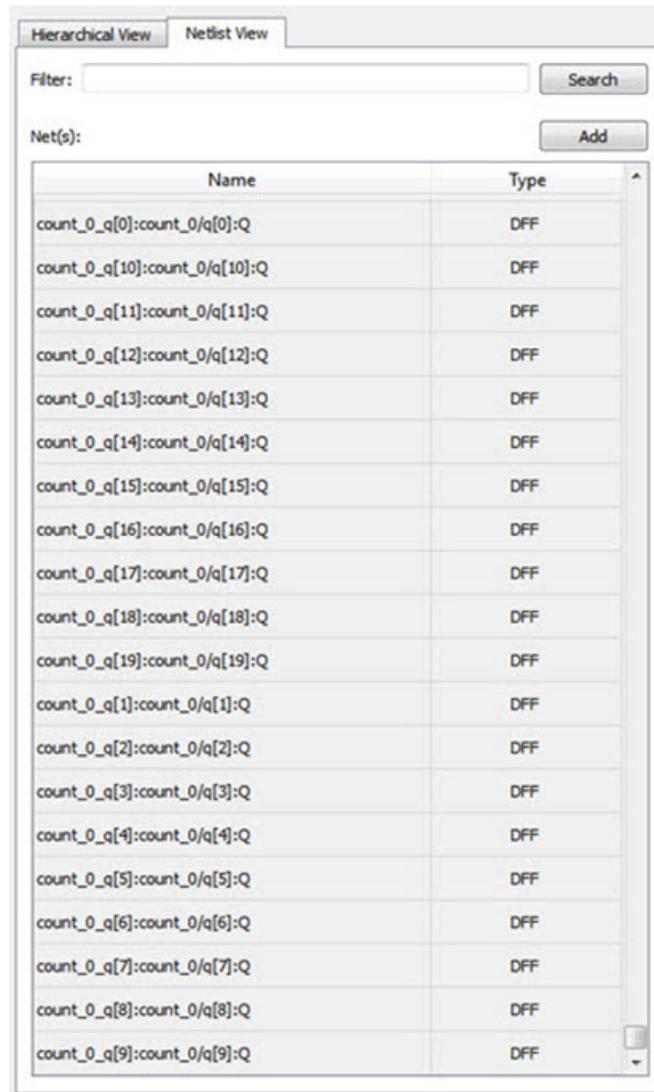


Figure 10 · Netlist View

Search

A search option is available in the Netlist View for Live Probes, Active Probes, and Probe Insertion. You can use wildcard characters such as * or ? in the search column for wildcard matching.

Live Probes (SmartFusion2, IGLOO2, and RTG4)

The Live Probes tab displays a table with the probe name and pin type.

Note: SmartFusion2 and IGLOO2 support two probe channels, and RTG4 supports one probe channel.

SmartFusion2 and IGLOO2

Two probe channels (ChannelA and ChannelB) are available. When a probe name is selected, it can be assigned to either ChannelA or ChannelB.

You can assign a probe to a channel by doing either of the following:

- Right-click a probe in the table and choose **Assign to Channel A** or **Assign to Channel B**.
- Click the **Assign to Channel A** or **Assign to Channel B** button to assign the probe selected in the table to the channel. The buttons are located below the table.

When the assignment is complete, the probe name appears to the right of the button for that channel, and SmartDebug configures the ChannelA and ChannelB I/Os to monitor the desired probe points. Because there are only two channels, a maximum of two internal signals can be probed simultaneously.

Click the **Unassign Channels** button to clear the live probe names to the right of the channel buttons and discontinue the live probe function during debug.

Note: At least one channel must be set; if you want to use both probes, they must be set at the same time.

The Active Probes READ/WRITE overwrites the settings of Live Probe channels (if any).

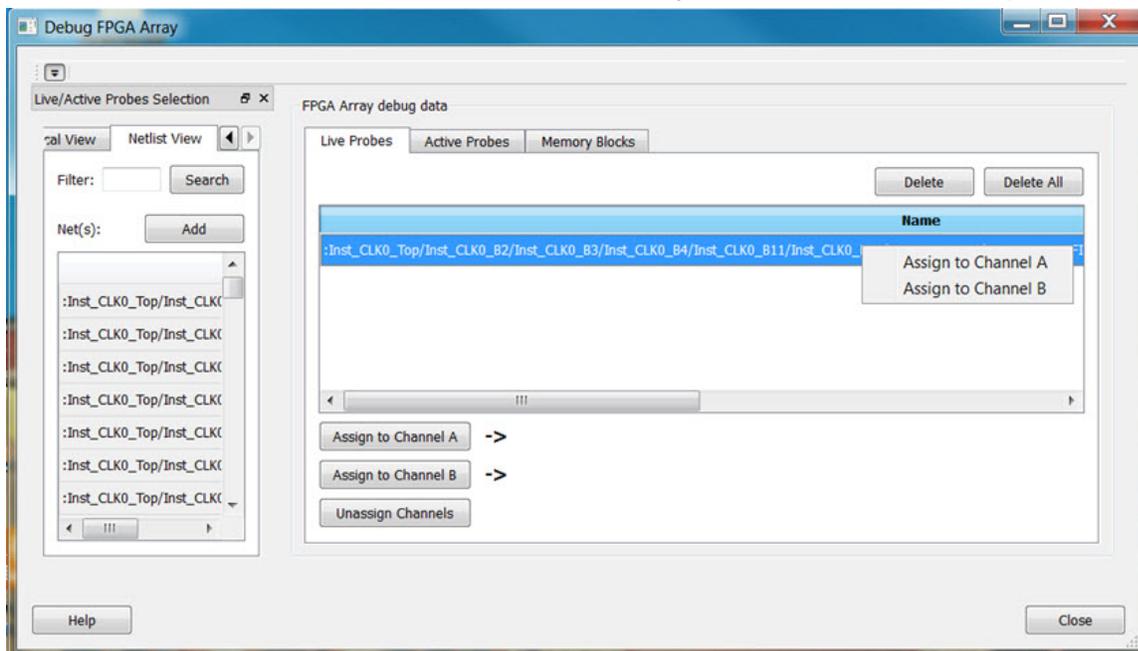


Figure 11 · Live Probes Tab (SmartFusion2 and IGLOO2) in SmartDebug FPGA Array Dialog Box

RTG4

One probe channel (Probe Read Data Pin) is available for RTG4 for debug. When a probe name is selected, it can be assigned to the Probe Channel (Probe Read Data Pin).

You can assign a probe to a channel by doing either of the following:

- Right-click a probe in the table and choose **Assign to Probe Read Data Pin**.
- Click the **Assign to Probe Read Data Pin** button to assign the probe selected in the table to the channel. The button is located below the table.

Click the **Unassign probe read data pin** button to clear the live probe name to the right of the channel button and discontinue the live probe function during debug.

The Active Probes READ/WRITE overwrites the settings of Live Probe channels (if any).

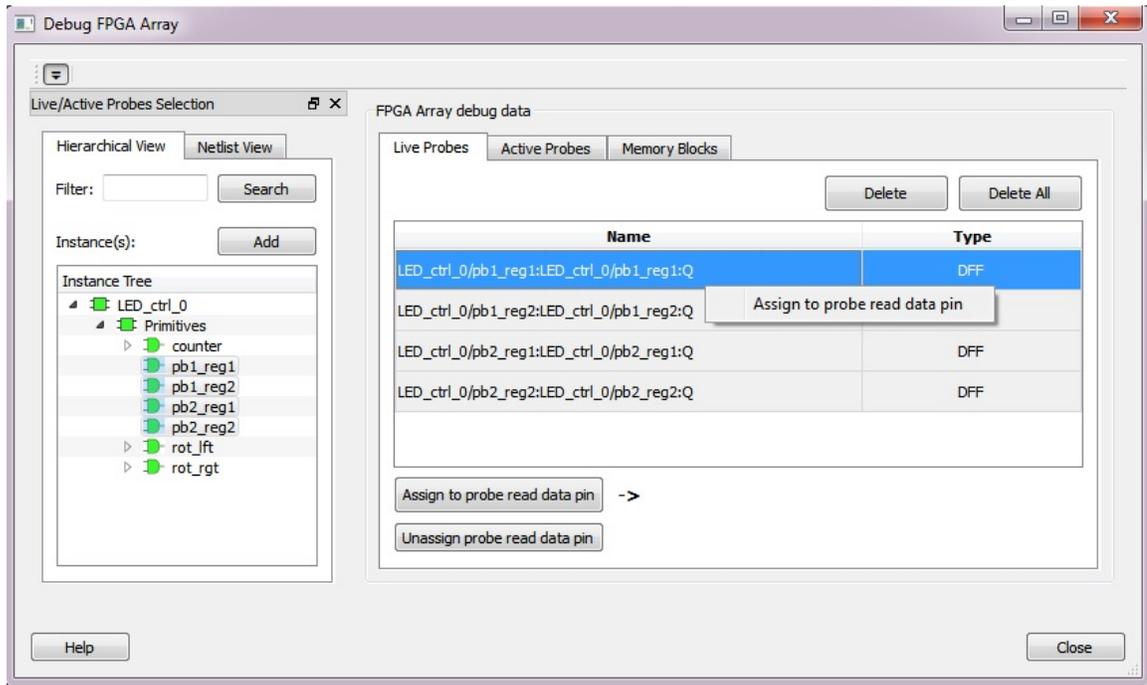


Figure 12 · Live Probes Tab (RTG4) in SmartDebug FPGA Array Dialog Box

Active Probes (SmartFusion2, IGLOO2, and RTG4)

In the left pane of the Active Probes tab, all available Probe Points are listed in instance level hierarchy in the Hierarchical View. All Probe Names are listed with the Name and Type (which is the physical location of the flip-flop) in the Netlist View.

Select probe points from the Hierarchical View or Netlist View, right-click and choose **Add** to add them to the Active Probes UI. You can also add the selected probe points by clicking the **Add** button. The probes list can be filtered by using the Filter box.

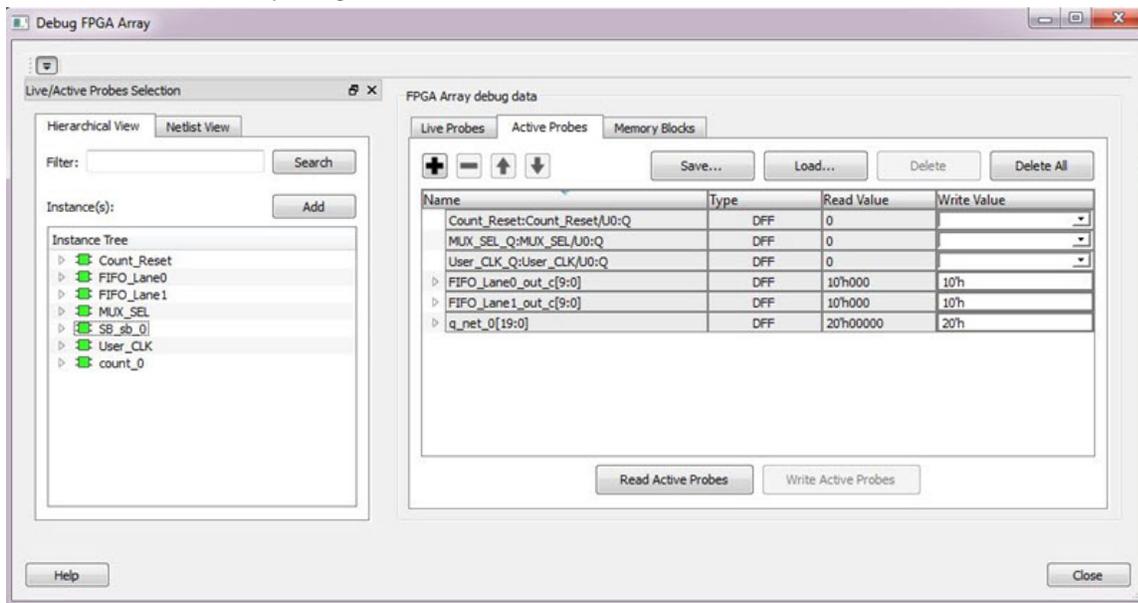


Figure 13 · Active Probes Tab in SmartDebug FPGA Array Dialog Box

When you have selected the desired probe, points appear in the Active Probe Data chart and you can read and write multiple probes (as shown in the figure below).

You can use the following options in the Write Value column to modify the probe signal added to the UI:

- Drop-down menu with values '0' and '1' for individual probe signals
- Editable field to enter data in hex or binary for a probe group or a bus

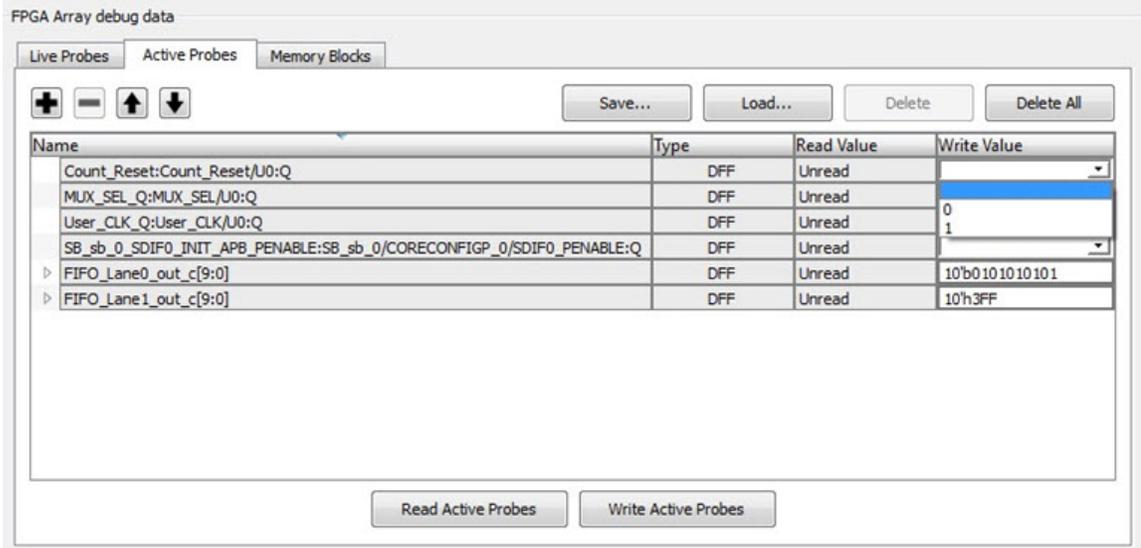


Figure 14 · Active Probes Tab - Write Value Column Options

Probe Grouping (Active Probes Only)

During the debug cycle of the design, designers often want to examine the different signals. In large designs, there can be many signals to manage. The Probe Grouping feature assists in comprehending multiple signals as a single entity. This feature is applicable to Active Probes only. Probe nets with the same name are automatically grouped in a bus when they are added to the Active Probes tab. Custom probe groups can also be created by manually selecting probe nets of a different name and adding them into the group.

The Active Probes tab provides the following options for probe points that are added from the Hierarchical View/Netlist View:

- Display of bus name. An automatically generated bus name cannot be modified. Only custom bus names can be modified.
- Expand/collapse of bus or probe group
- Move Up/Down the signal or bus or probe group
- Save (Active Probes list)
- Load (already saved Active Probes list)
- Delete (applicable to a single probe point added to the Active Probes tab)
- Delete All (deletes all probe points added to the Active Probes tab)
- In addition, the context (right-click) menu provides the following operations:
 - Create Group, Add/Move signals to Group, Remove signals from Group,
 - Ungroup
 - Reverse bit order, Change Radix for a bus or probe group
 - Read, Write, or Delete the signal or bus or probe group

FPGA Array debug data

Live Probes Active Probes Memory Blocks

Name	Type	Read Value	Write Value
Count_EPCS_Clk_Q_0:Count_EPCS_Clk/U0:Q	DFF	1	1
Count_EPCS_Reset:Count_EPCS_Reset/U0:Q	DFF	0	0
FIFO_RESET_N_Q:FIFO_RESET_N/U0:Q	DFF	1	
GPIO_RESET_N_Q:GPIO_RESET_N/U0:Q	DFF	1	
q_c[19:0]	DFF	20'hF0F0F	20'hF0F0F
q_c[19]:count_0/q[19]:Q	DFF	1	
q_c[18]:count_0/q[18]:Q	DFF	1	
q_c[17]:count_0/q[17]:Q	DFF	1	
q_c[16]:count_0/q[16]:Q	DFF	1	
q_c[15]:count_0/q[15]:Q	DFF	0	
q_c[14]:count_0/q[14]:Q	DFF	0	
q_c[13]:count_0/q[13]:Q	DFF	0	
q_c[12]:count_0/q[12]:Q	DFF	0	
q_c[11]:count_0/q[11]:Q	DFF	1	
q_c[10]:count_0/q[10]:Q	DFF	1	
q_c[9]:count_0/q[9]:Q	DFF	1	
q_c[8]:count_0/q[8]:Q	DFF	1	
q_c[7]:count_0/q[7]:Q	DFF	0	
q_c[6]:count_0/q[6]:Q	DFF	0	
q_c[5]:count_0/q[5]:Q	DFF	0	
q_c[4]:count_0/q[4]:Q	DFF	0	
q_c[3]:count_0/q[3]:Q	DFF	1	
q_c[2]:count_0/q[2]:Q	DFF	1	
q_c[1]:count_0/q[1]:Q	DFF	1	
q_c[0]:count_0/q[0]:Q	DFF	1	
group1[1:0]		2'h0	2'h
q_c[14]:count_0/q[14]:Q	DFF	0	1
q_c[13]:count_0/q[13]:Q	DFF	0	1
group2[1:0]		2'h1	2'h
Count_EPCS_Reset:Count_EPCS_Reset/U0:Q	DFF	0	0
FIFO_RESET_N_Q:FIFO_RESET_N/U0:Q	DFF	1	

Figure 15 · Active Probes Tab

- Green entries in the “Write Value” column indicate that the operation was successful.
- Blue entries in the “Read Value” column show indicate values that have changed since the last read.

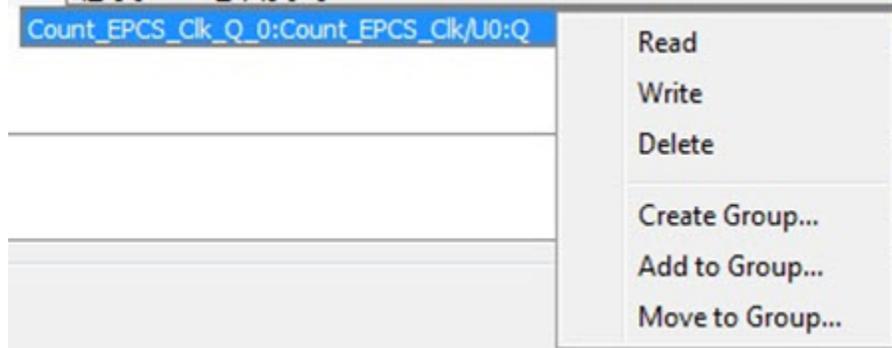
Context Menu of Probe Points Added to the Active Probes UI

When you right-click a signal or bus, you will see the following menu options:

For individual signals that are not part of a probe group or bus:

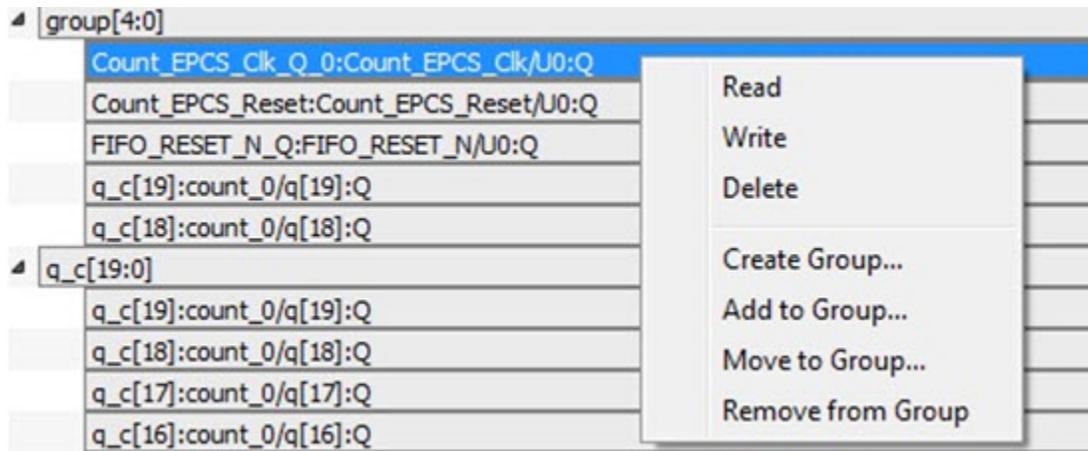
- Read
- Write

- Delete
- Create Group
- Add to Group
- Move to Group



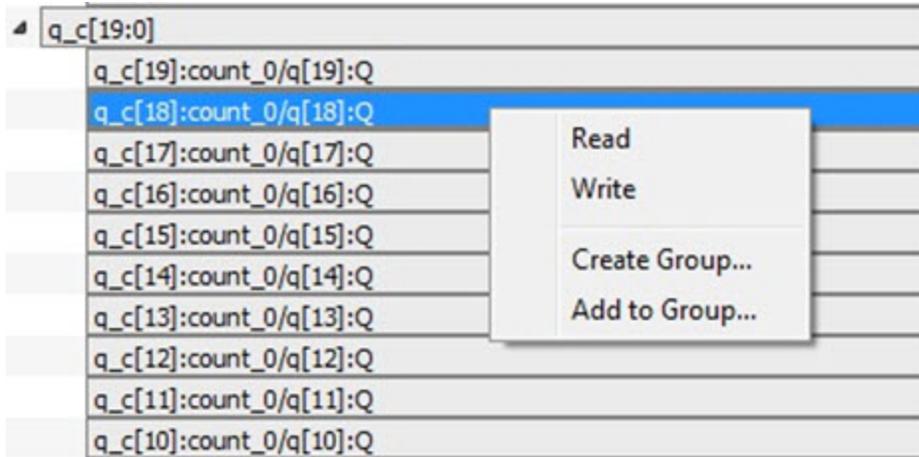
For individual signals in a probe group:

- Read
- Write
- Delete
- Create Group
- Add to Group
- Move to Group
- Remove from Group



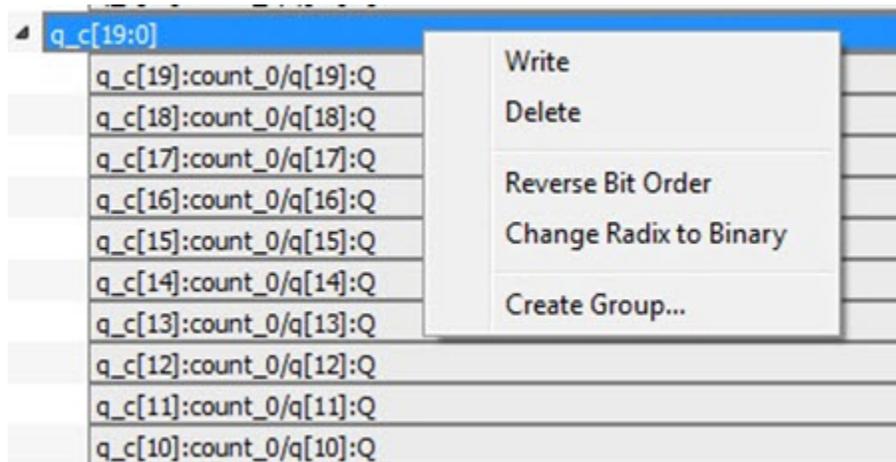
For individual signals in a bus:

- Read
- Write
- Create Group
- Add to Group



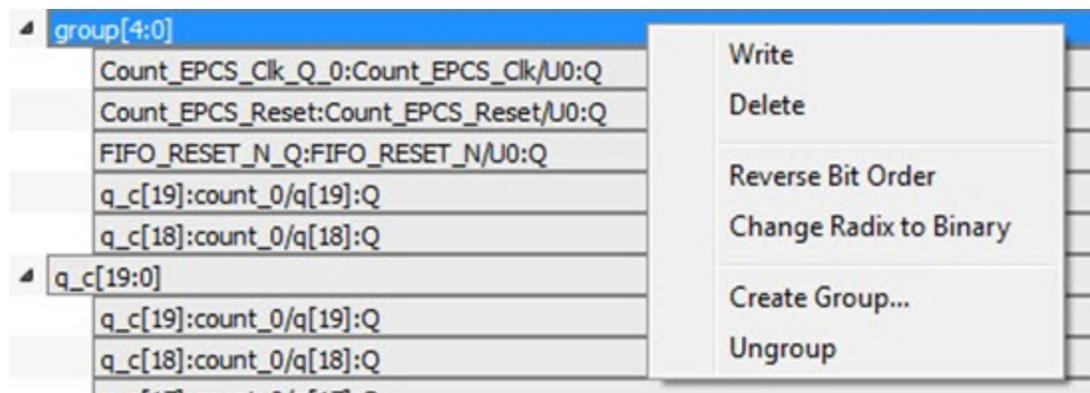
For a bus:

- Read
- Write
- Delete
- Reverse Bit Order
- Change Radix to Hex/Binary
- Create Group



For a probe group:

- Write
- Delete
- Reverse Bit Order
- Change Radix to Hex/Binary
- Create Group
- Ungroup



Differences Between a Bus and a Probe Group

A bus is created automatically by grouping selected probe nets with the same name into a bus. A bus *cannot* be ungrouped.

A Probe Group is a custom group created by adding a group of signals in the Active Probe tab into the group. The members of a Probe Group are not associated by their names. A Probe Group *can* be ungrouped.

In addition, the certain operations are also restricted to the member of a bus, whereas they are allowed in a probe group.

The following operations are not allowed in a bus:

- **Delete:** Deleting an individual signal in a bus
- **Move to Group:** Moving a signal to a probe group
- **Remove from Group:** Removing a signal from a probe group

Memory Blocks (SmartFusion2, IGLOO2, and RTG4)

The left pane in the Memory Blocks tab shows the list of defined memory blocks that are specified in your design. Select a memory block and add it to the UI. After you select a memory block, you can click **Read Block** to show the current content of the memory or write to individual memory locations. Each field is editable, and multiple memory locations can be written at the same time.

Each field is a 9-bit memory word (for SmartFusion2 and IGLOO2 devices) and 9-bit or 12-bit memory word (for RTG4 devices, depending on the memory configuration). Valid inputs are hexadecimal values between 0x0 and 0x1FF (for SmartFusion2, IGLOO2, and RTG4 devices in 9-bit mode) and 0x0 and 0xFFFF (for RTG4 devices only in 12-bit mode), as shown in the figure below.

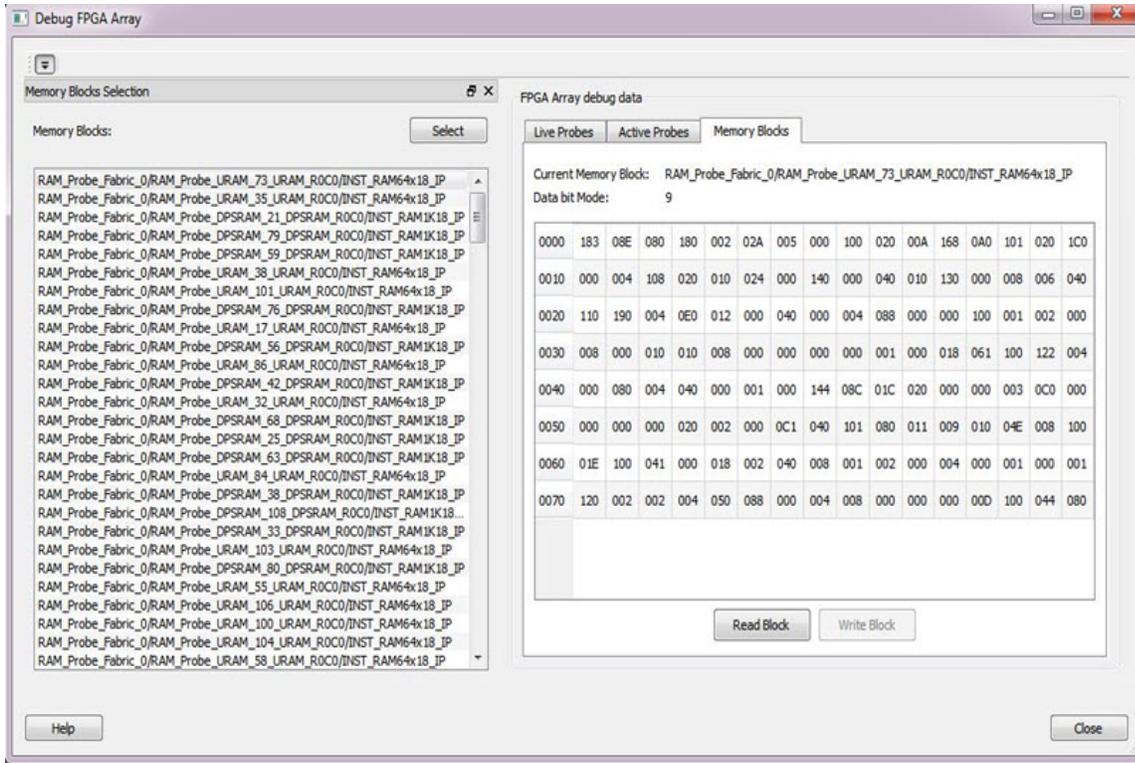


Figure 16 · Debug FPGA Array - Memory Blocks Tab

Fields that have changed but have not yet been written appear in red text until you click **Write Block** to initiate a memory write (as shown in the figure below).

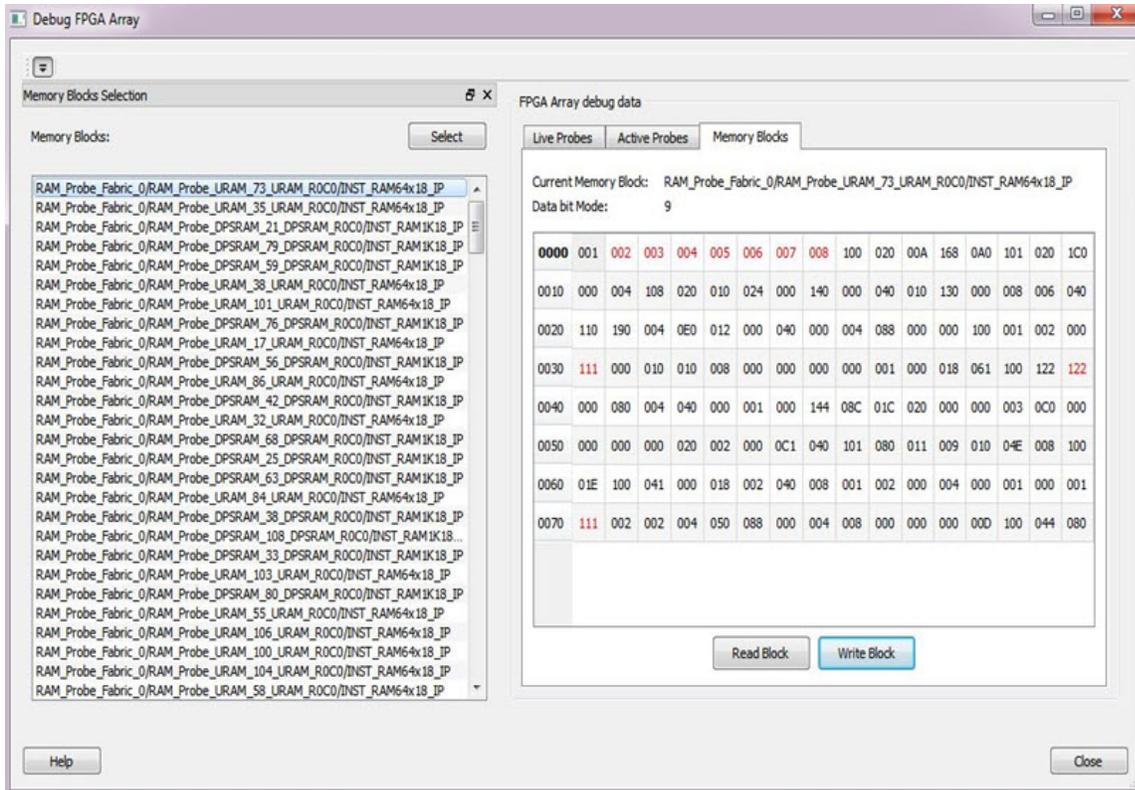


Figure 17 · Memory Blocks with Updated Values

Probe Insertion (Post-Layout) - SmartFusion2, IGLOO2, and RTG4

Introduction

Probe insertion is a post-layout debug process that enables internal nets in the FPGA design to be routed to unused I/Os. Nets are selected and assigned to probes using the Probe Insertion window in SmartDebug. The rerouted design can then be programmed into the FPGA, where an external logic analyzer or oscilloscope can be used to view the activity of the probed signal.

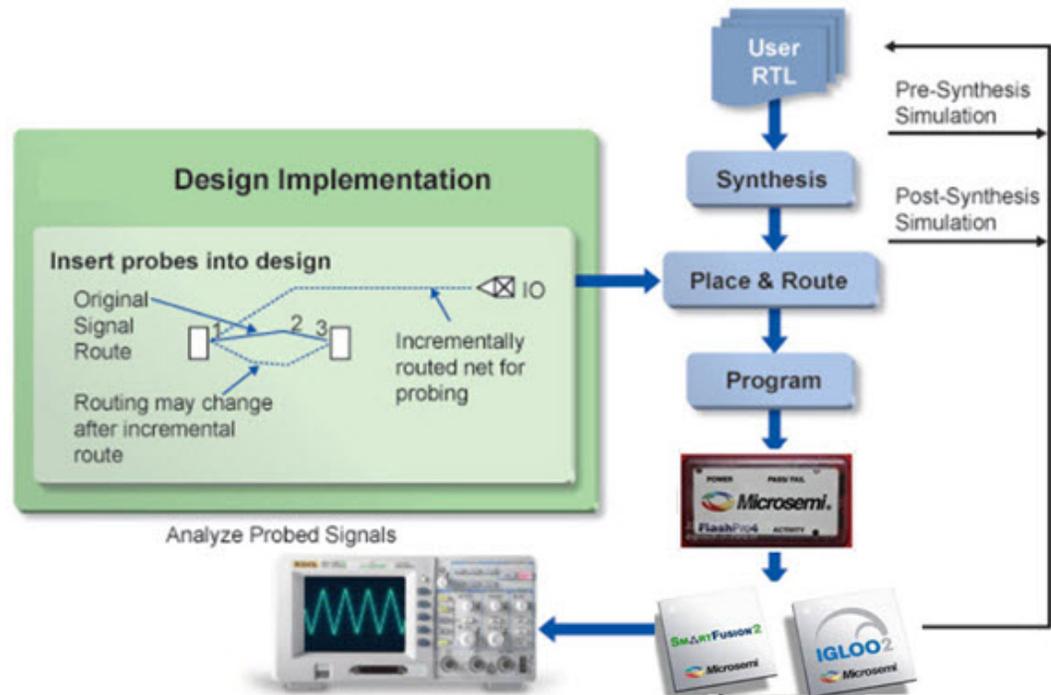


Figure 18 · Probe Insertion in the Design Process

The Probe Insertion debug feature is complementary to Live Probes and Active Probes. Live Probes and Active Probes use a special dedicated probe circuitry.

Probe Insertion

1. Double-click **SmartDebug Design** in the Design Flow window to open the SmartDebug main window.
Note: FlashPro Programmer must be connected for SmartDebug.
2. Select **Debug FPGA Array** and then select the Probe Insertion tab.

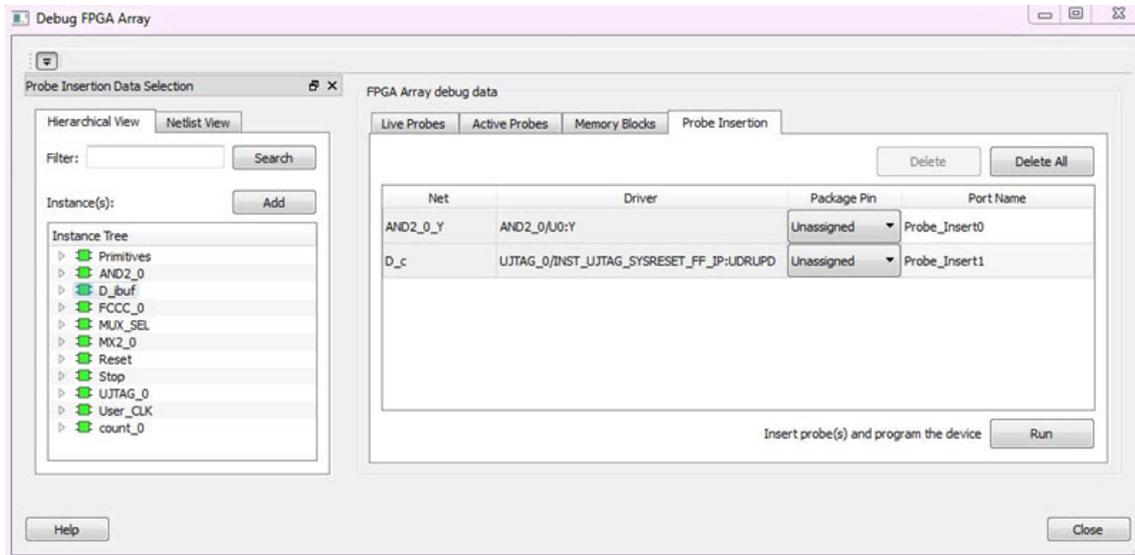


Figure 19 · Probe Insertion Tab

In the left pane of the Probe Insertion tab, all available Probe Points are listed in instance level hierarchy in the Hierarchical View. All Probe Names are shown with the Name and Type in the Netlist View.

3. Select probe points from the Hierarchical View or Netlist View, right-click and choose **Add** to add them to the Active Probes UI. You can also add the selected probe points by clicking the **Add** button. The probes list can be filtered by using the Filter box.

Each entry has a Net and Driver name which identifies that probe point.

The selected net(s) appear in the Probes table in the Probe Insertion tab, as shown in the figure below. SmartDebug automatically generates the Port Name for the probe. You can change the Port Name from the default if desired.

4. Assign a package pin to the probe using the drop-down list in the Package Pin column. You can assign the probe to any unused package pin (spare I/O).

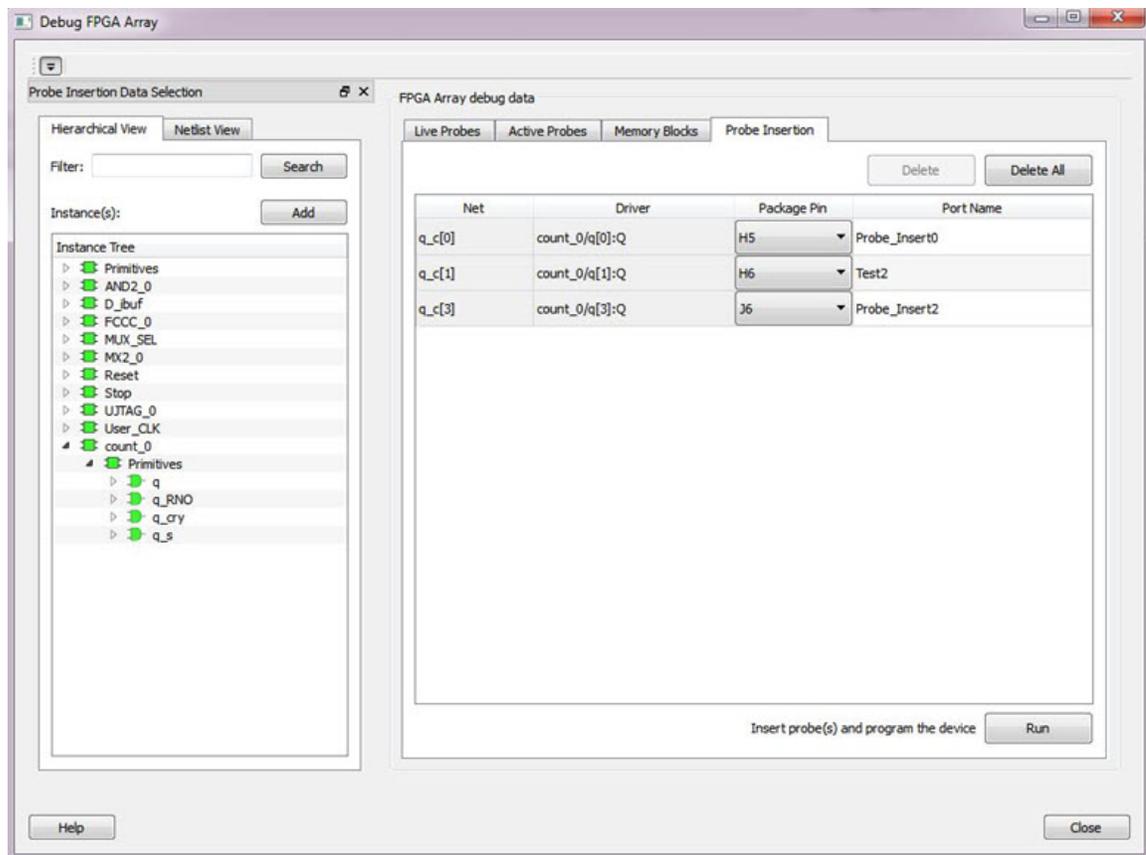


Figure 20 · Debug FPGA Array > Probe Insertion > Add Probe

5. Click **Run**.

This triggers Place and Route in incremental mode, and the selected probe nets are routed to the selected package pin. After incremental Place and Route, Libero automatically reprograms the device with the added probes.

The log window shows the status of the Probe Insertion run.

Probe Deletion

To delete a probe, select the probe and click **Delete**. To delete all the probes, click **Delete All**.

Note: Deleting probes from the probes list without clicking **Run** does not automatically remove the probes from the design.

Reverting to the Original Design

To revert to the original design after you have finished debugging:

1. In SmartDebug, click **Delete All** to delete all probes.
2. Click **Run**.
3. Wait until the action has completed by monitoring the activity indicator (spinning blue circle). Action is completed when the activity indicator disappears.
4. Close SmartDebug.

Debug SERDES (SmartFusion2, IGLOO2, and RTG4)

You can examine and debug the SERDES blocks in your design in the Debug SERDES dialog box (shown in the figure below).

To Debug SERDES, expand **SmartDebug** in the Design Flow window and double-click **Debug SERDES**.

Debug SERDES Configuration is explained below. See the [PRBS Test](#) and [Loopback Test](#) topics for information specific to those procedures.

SERDES Block identifies which SERDES block you are configuring. Use the drop-down menu to select from the list of SERDES blocks in your design.

Debug SERDES - Configuration

Configuration Report

The Configuration Report output depends on the options you select in your [PRBS Test](#) and [Loopback Tests](#). The default report lists the following for each Lane in your SERDES block:

Lane mode - Indicates the programmed mode on a SERDES lane as defined by the SERDES system register.

PMA Ready - Indicates whether PMA has completed its internal calibration sequence for the specific lane and whether the PMA is operational. See the [SmartFusion2](#) or [IGLOO2](#) High Speed Serial Interfaces User Guide on the Microsemi website for details.

TxPLL status - Indicates the loss-of-lock status for the TXPLL is asserted and remains asserted until the PLL reacquires lock.

RxPLL status - Indicates the CDR PLL frequency is not grossly out of range of with incoming data stream.

Click **Refresh Report** to update the contents of your SERDES Configuration Report. Changes to the specified SERDES register programming can be read back to the report.

SERDES Register Read or Write

Script - Runs Read/Write commands to access the SERDES control/status register map using a script. Enter the full pathname for the script location or click the browse button to navigate to your script file. Click **Execute** to run the script.

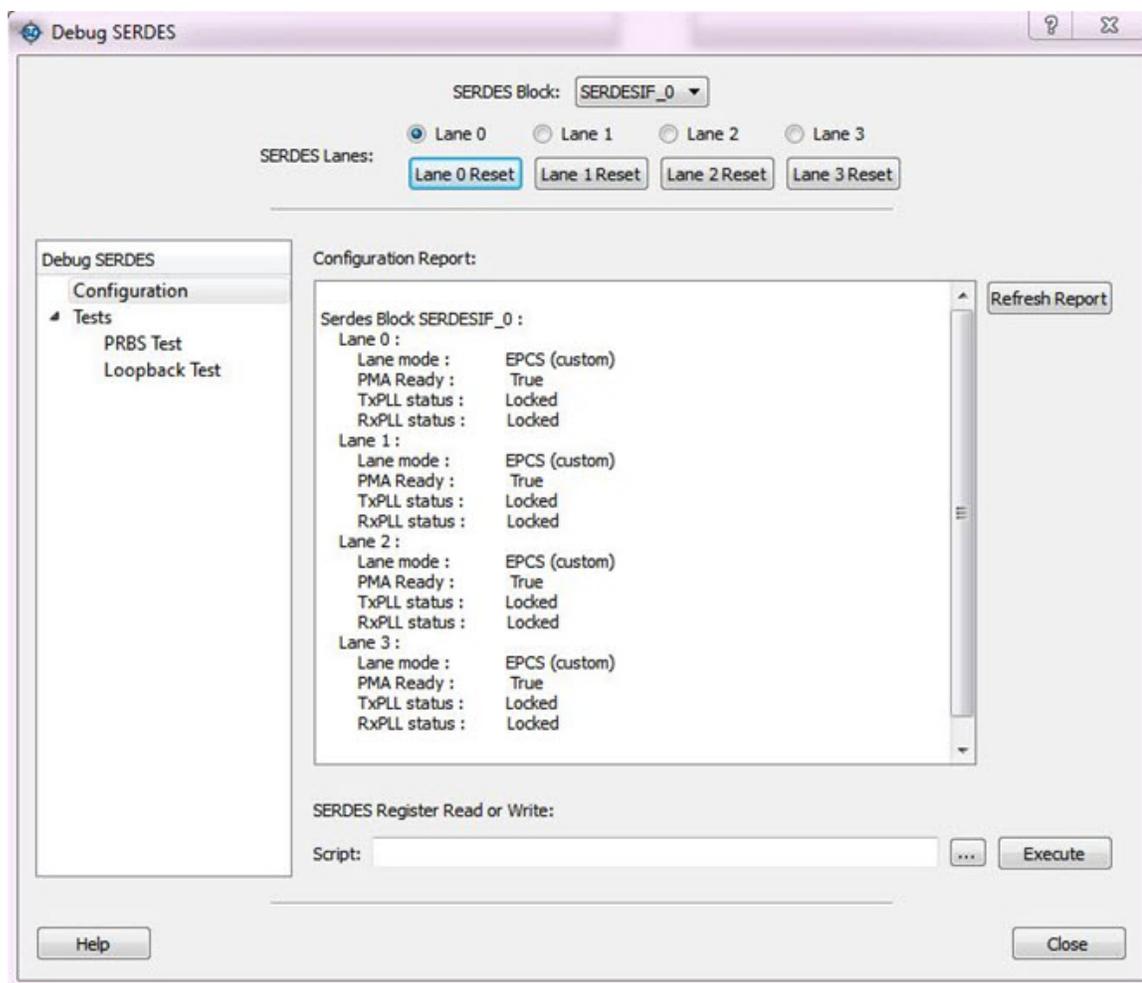


Figure 21 · Debug SERDES - Configuration

Note: The PCIe and XAUI protocols only support PRBS7. The EPCS protocol supports PRBS7/11/23/31.

Debug SERDES – Loopback Test

Loopback data stream patterns are generated and checked by the internal SERDES block. These are used to self-test signal integrity of the device. You can switch the device through predefined tests.

See the [PRBS Test topic](#) for more information about the PRBS test options.

SERDES Block identifies which SERDES block you are configuring. Use the drop-down menu to select from the list of SERDES blocks in your design.

SERDES Lanes

Select the **Lane** and **Lane Status** on which to run the Loopback test. Lane mode indicates the programmed mode on a SERDES lane as defined by the SERDES system register.

Test Type

PCS Far End PMA RX to TX Loopback- This loopback brings data into the device and deserializes and serializes the data before sending it off-chip. This loopback requires 0PPM clock variation between the TX and RX SERDES clocks.

See the [SmartFusion2](#) or [IGLOO2](#) High Speed Serial Interfaces User's Guide on the Microsemi website for details.

Near End Loopback (On Die) - To enable, select the Near End Loopback (On Die) option and click **Start**. Click **Stop** to disable. Using this option allows you to send and receive user data without sending traffic off-chip. You can test design functionality without introducing other issues on the PCB.

See the [SmartFusion2](#) or [IGLOO2](#) High Speed Serial Interfaces User's Guide on the Microsemi website for details.

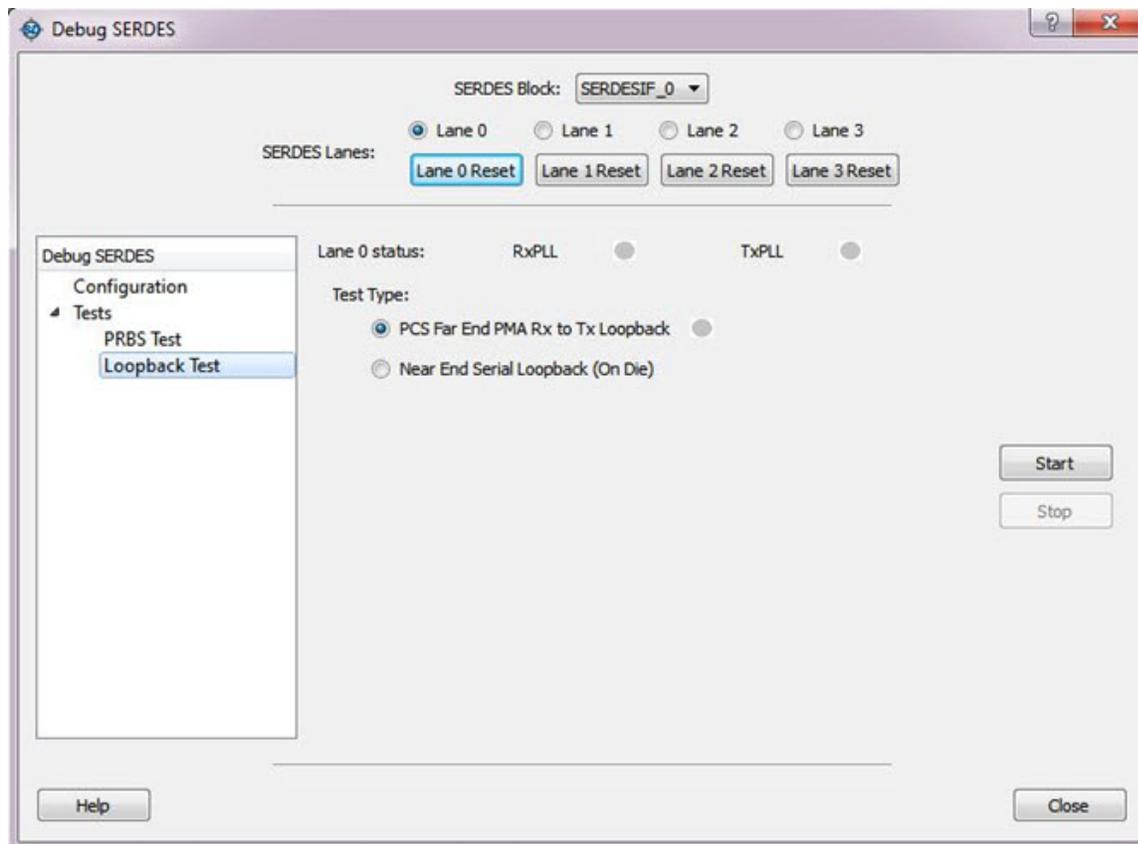


Figure 22 · Debug SERDES - Loopback Test

Debug SERDES – PRBS Test

PRBS data stream patterns are generated and checked by the internal SERDES block. These are used to self-test signal integrity of the device. You can switch the device through several predefined patterns.

View Loopback Test settings in the [Debug SERDES - Loopback Test topic](#).

SERDES Block identifies which SERDES block you are configuring. Use the drop-down menu to select from the list of SERDES blocks in your design.

SERDES Lanes

Select the **Lane** and **Lane Status** on which to run the PRBS test. Lane mode indicates the programmed mode on a SERDES lane as defined by the SERDES system register.

Test Type

Near End Serial Loopback (On-Die) enables a self test of the device. The serial data stream is sent from the SERDES TX output and folded back onto the SERDES RX input.

Serial Data (Off-Die) is the normal system operation where the data stream is sent off chip from the TX output and must be connected to the RX input via a cable or other type of electrical interconnection.

Pattern

The SERDESIF includes an embedded test pattern generator and checker used to perform serial diagnostics on the serial channel, as shown in the table below.

Pattern	Type
PRBS7	Pseudo-Random data stream of 2 ⁷ polynomial sequences
PRBS11	Pseudo-Random data stream of 2 ¹¹ polynomial sequences
PRBS23	Pseudo-Random data stream of 2 ²³ polynomial sequences
PRBS31	Pseudo-Random data stream of 2 ³¹ polynomial sequences

Cumulative Error Count

Lists the number of cumulative errors after running your PRBS test. Click **Reset** to reset to zero. 1. By default, Cumulative Error Count = 0, the Data Rate text box is blank, and Bit Error Rate = NA.

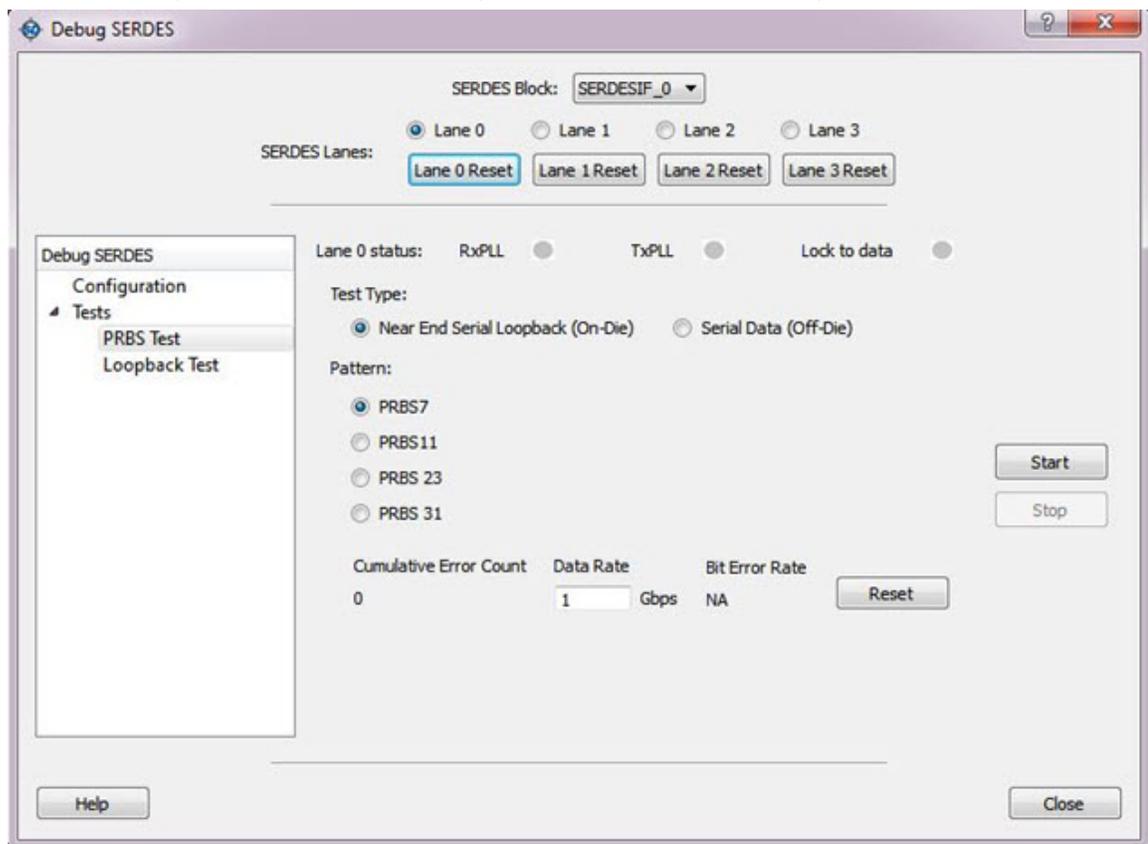


Figure 23 · Debug SERDES - PRBS Test

Note: If the design uses SERDES PCIe, PRBS7 will be the only option shown for PRBS tests..

Bit Error Rate

Displays the Bit Error Rate (BER) for the PRBS test in progress.

The formula for calculating the BER is as follows:

$$\text{BER} = (\text{\#bit errors} + 1) / \text{\#bits sent}$$

$$\text{\#bits sent} = \text{Elapsed time} / \text{bit period}$$

When clicked on Start:

- The BER is updated every second for the entered data rate and errors observed.
- If no data rate is entered by the user, the BER is set to the default.

When clicked on Stop:

- The BER resets to default.

When clicked on Reset:

- The BER is reset to default.
- If no test is in progress, the BER remains in the default value.
- If the PRBS test is in progress, the BER calculation restarts.

Debug SERDES – PHY Reset

SERDES PMA registers (for example, TX_AMP_RATIO) modified using a TCL script from the Configuration tab require a soft reset for the new values to be updated. Lane Reset for individual lanes achieves this functionality. Depending on the SERDES lanes used in the design, the corresponding Lane Reset buttons are enabled.

Lane Reset Behavior for SERDES Protocols Used in the Design

- EPCS: Reset is independent for individual lanes. Reset to Lane X (where X = 0,1,2,3) resets the Xth lane.
- PCIe: Reset to Lane X (where X = 0,1,2,3) resets all lanes present in the PCIe link and PCIe controller.

For more information about soft reset, refer to the [SmartFusion2 and IGL002 High Speed Serial Interfaces User Guide](#).

Inspect Device Dialog Box (SmartFusion, Fusion, and ProASIC3 Only)

Inspect Device is available as a part of the FlashPro programming tool. Refer to [Using SmartDebug](#) for information about how to configure FlashPro to access this feature.

In the Inspect Device dialog box, you can access all SmartDebug features, such as the FlashROM, Embedded Flash Memory (NVM), and Analog Block. If you have multiple devices and programmers connected, choose your target device/programmer from the drop-down menu, and use the ID code to verify that you are inspecting the correct device.

View Device Status - Displays the [Device Status Report](#). The Device Status Report is a summary of your device state, analog block test values, user information, factory data, and security information. You can save or print your information for future reference.

View Analog Block Configuration - Opens the [Analog Block Configuration dialog box](#). You can view the channel configuration for your analog block and compare the channel configuration with any other analog block file.

View Flash Memory Content - Opens the [Flash Memory dialog box](#). You can view the details for each flash memory block in your device.

View FlashROM Content - Opens the [FlashROM data dialog box](#). You can view a list of the physical blocks in your FlashROM and the client partitions in FlashROM configuration files.

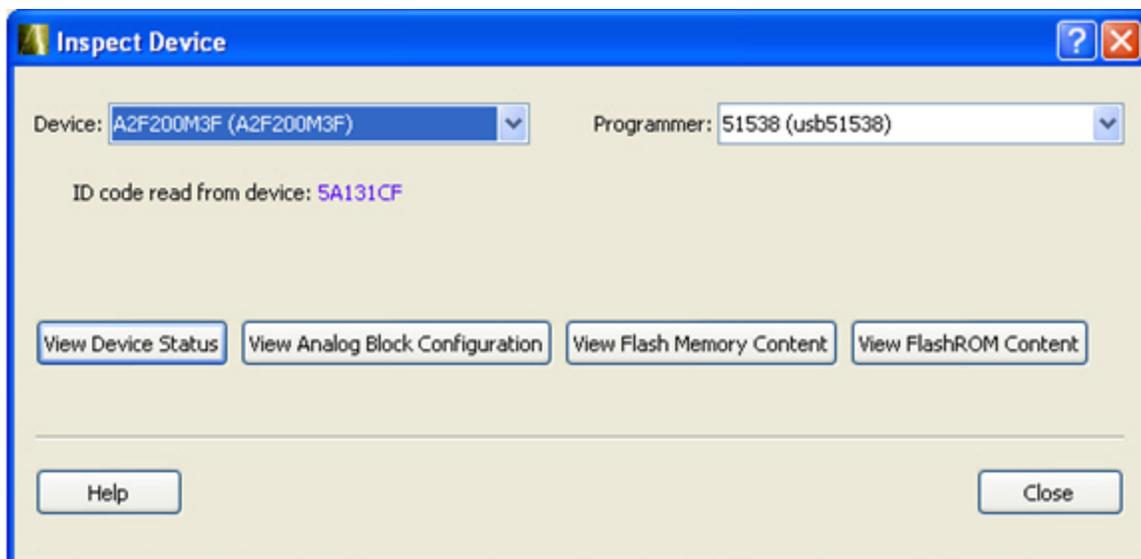


Figure 24 · Inspect Device Dialog Box

Device Status Report (SmartFusion and Fusion Only)

This dialog box displays the Device Information report. The Device Information report is a complete summary of your device state, analog block test values, user information, factory serial number, and security information. Use this dialog box to save or print your information for future reference.

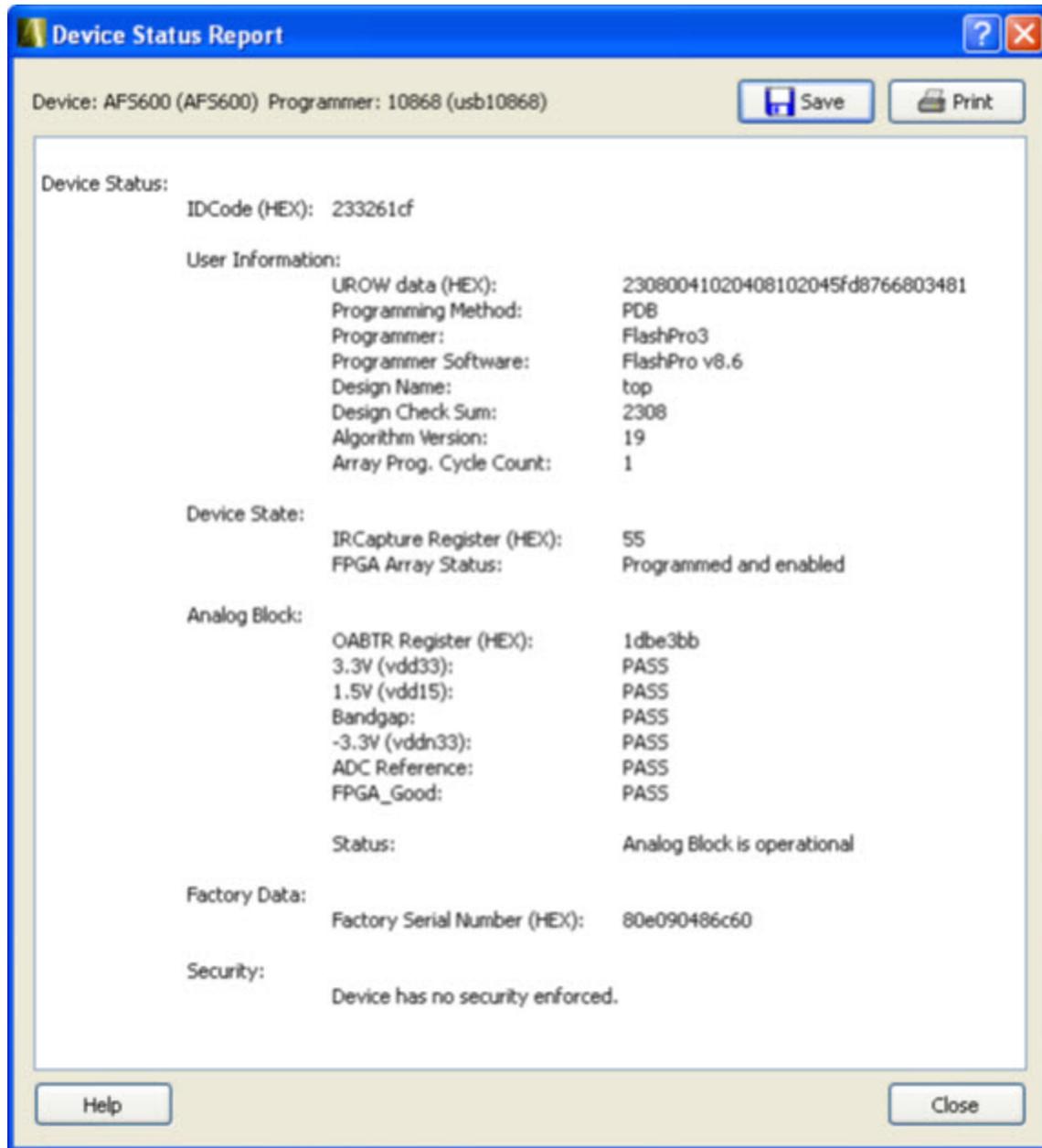


Figure 25 · Device Status Report

Embedded Flash Memory (NVM) Content Dialog Box (SmartFusion and Fusion Only)

You can do the following in the NVM content dialog box:

- View content of Flash Memory pages (as shown in the figure below)
- Compare device content with original design content (requires a PDB that contains your EFC data)

- Check page status and identify if a page is corrupted or if the write count limit has exceeded the 10-year retention threshold

Fusion Devices: Choose your block from the **From block** drop-down list. This action populates the Select drop-down list with the names of the clients in the selected block that is configured in the Flash Memory System Builder.

SmartFusion Devices: Block selection is unused and unavailable.

Choose a client name from the Select drop-down list and click **Read from Device** to view the values. You can also view a specific page range by selecting the <Page Range> option in the Select drop-down list and then specifying the start page and the end page.

You must click **Read from Device** each time you specify a new page range to update the view.

If you do not have your original design programming database (PDB) file, you can examine and retrieve a range of pages. Specify a page range if you wish to examine a specific set of pages. Page Status information appears to the right.

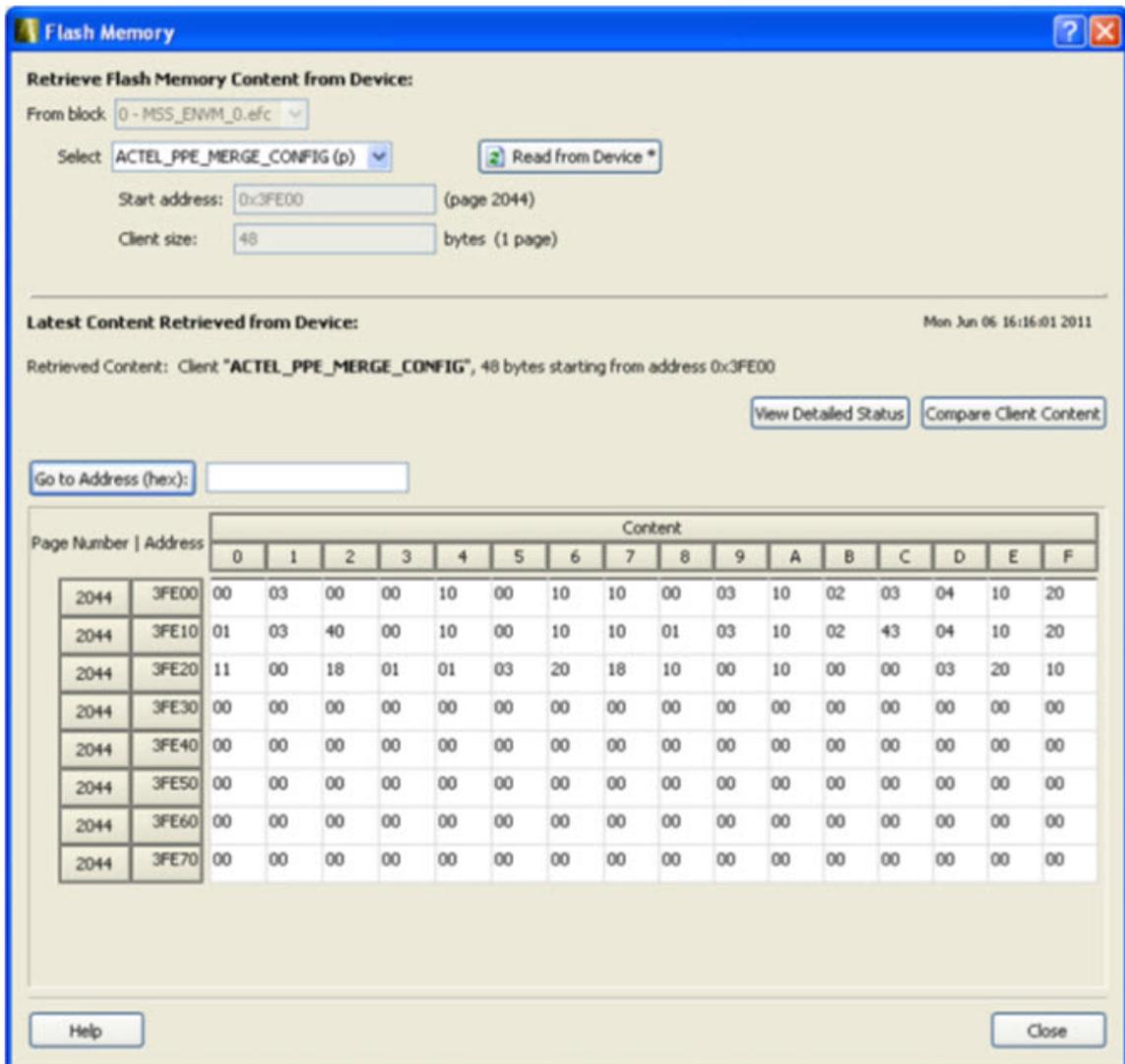


Figure 26 · Flash Memory Content Dialog Box for a SmartFusion Device (SmartDebug)

Embedded Flash Memory: Browse Retrieved Data (SmartFusion and Fusion Only)

The retrieved data table displays the content of the selected client or the page range selection. Corrupted page content is displayed in red. Read-only page content, corresponding to clients defined with the Prevent read option in Flash Memory System Builder, is displayed with a gray background. If content cannot be read (for example, pages are read-protected, but security has been erased), the content is displayed as XX. The mouse tooltip summarizes abnormal content status (as shown in the figure below).

The corresponding page number and address (relative to the current block) are displayed in the left column. The client size specified in the Flash Memory System Builder is shown at the top of the content table.

In the Retrieved Data View, you can enter an Address value (such as 0010) in the Go to Address field and click the corresponding button to go directly to that address.

Click **View Detailed Status** for a detailed report on the page range you have selected.

For example, if you want to view a report on pages 1-3, set the **Start Page** to 1, set the **End Page** to 3, and click **Read from Device**. Then click **View Detailed Status**. The figure below is an example of the data for a specific page range.

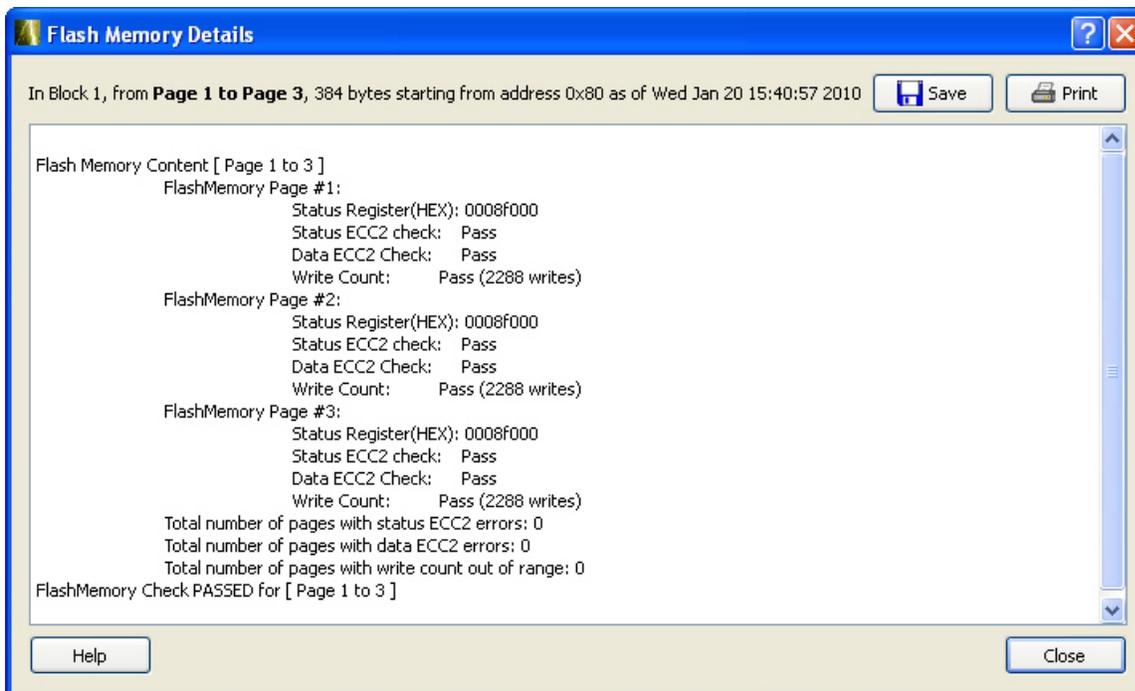


Figure 27 · Flash Memory Details Dialog Box (SmartDebug)

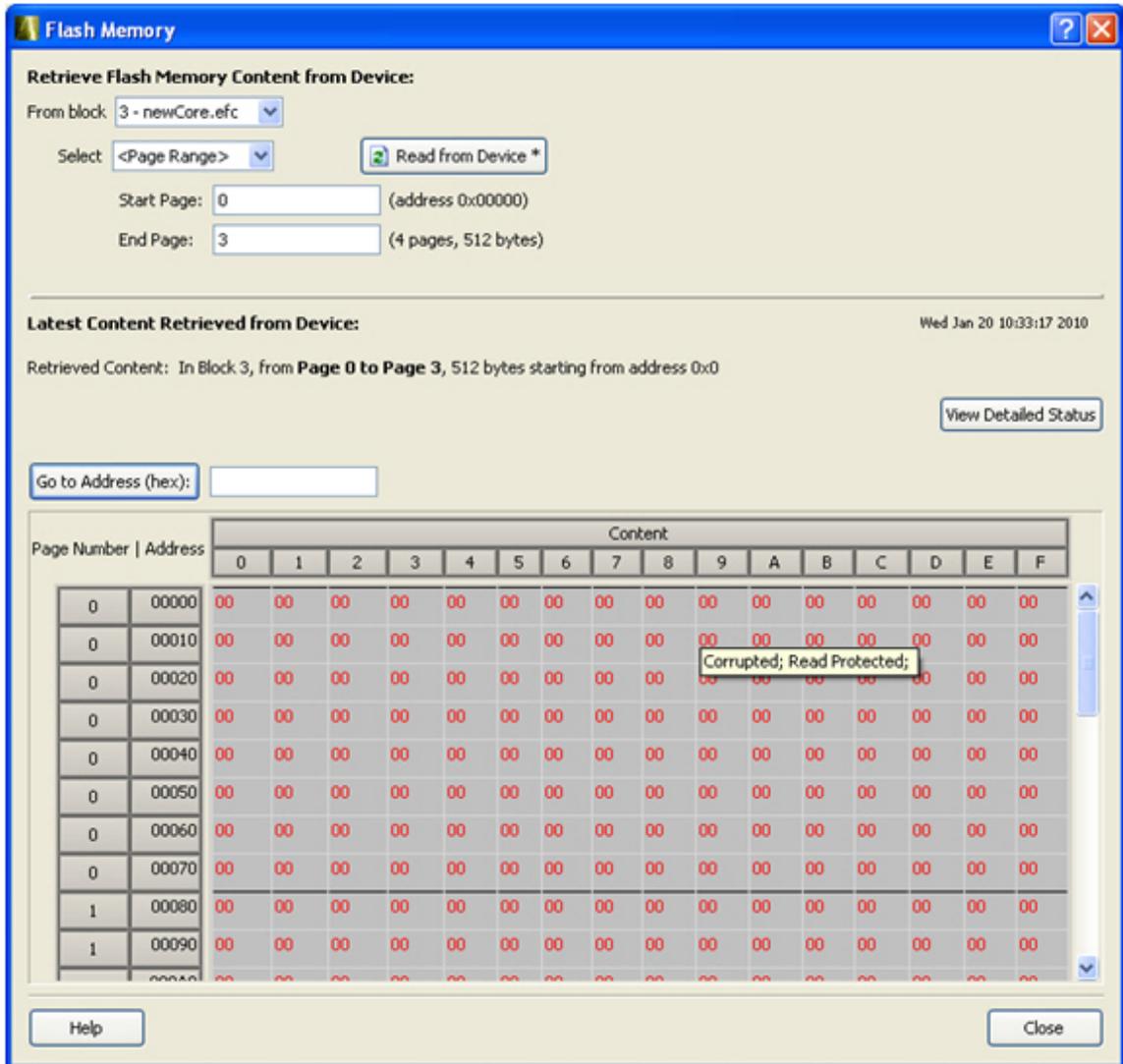


Figure 28 · Flash Memory Browse Retrieved Data

Embedded Flash Memory: Compare Memory Client (SmartFusion and Fusion Only)

After you retrieve the data from the device, the Compare Client Content button lets you compare the content of the selected client from the device with the original programming database (PDB) file. The differences are shown in the Compare Memory Client dialog box (as shown in the figure below).

Note: This option is not available when you select to retrieve the data based on a page range.

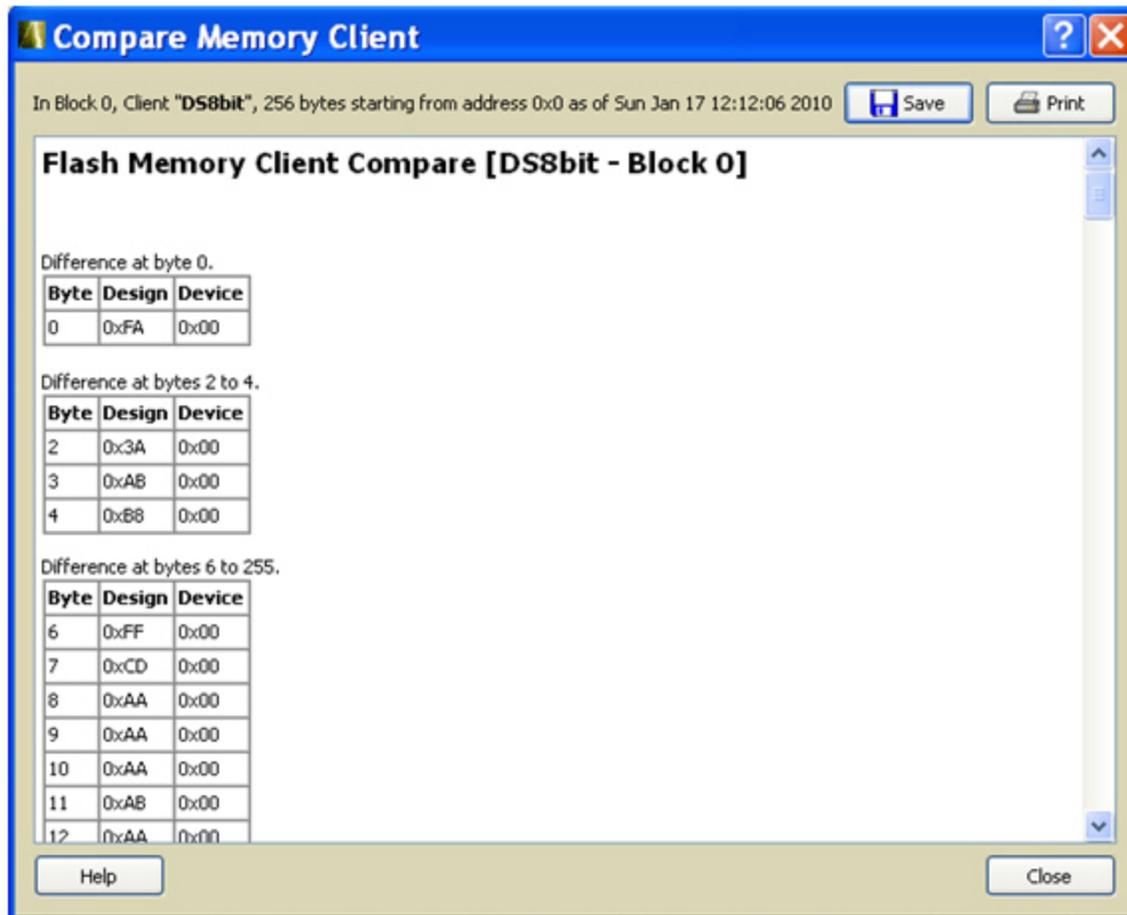


Figure 29 · Compare Memory Client Dialog Box

FlashROM Content Dialog Box (Fusion and SmartFusion Only)

In the FlashROM Content dialog box, you can view the physical blocks in your FlashROM and the client partitions specified in the original design content (requires a PDB that contains your UFC data). If the project's PDB does not contain UFC data, only the physical blocks are displayed.

Scroll through the table to view the Words and Pages for your physical blocks.

The Client Partitions section lists the names and configuration details of the clients set up in the FlashROM Builder. It automatically finds all mismatched client regions. To view the differences between a client and the device content, select a region row in the Client Partitions table. This action highlights the corresponding device content in the Physical Blocks table. The mismatch details are displayed below the Client Partitions table.

To copy the content of the Physical Blocks table to clipboard, select one or more cells in the table and type Ctrl+C.

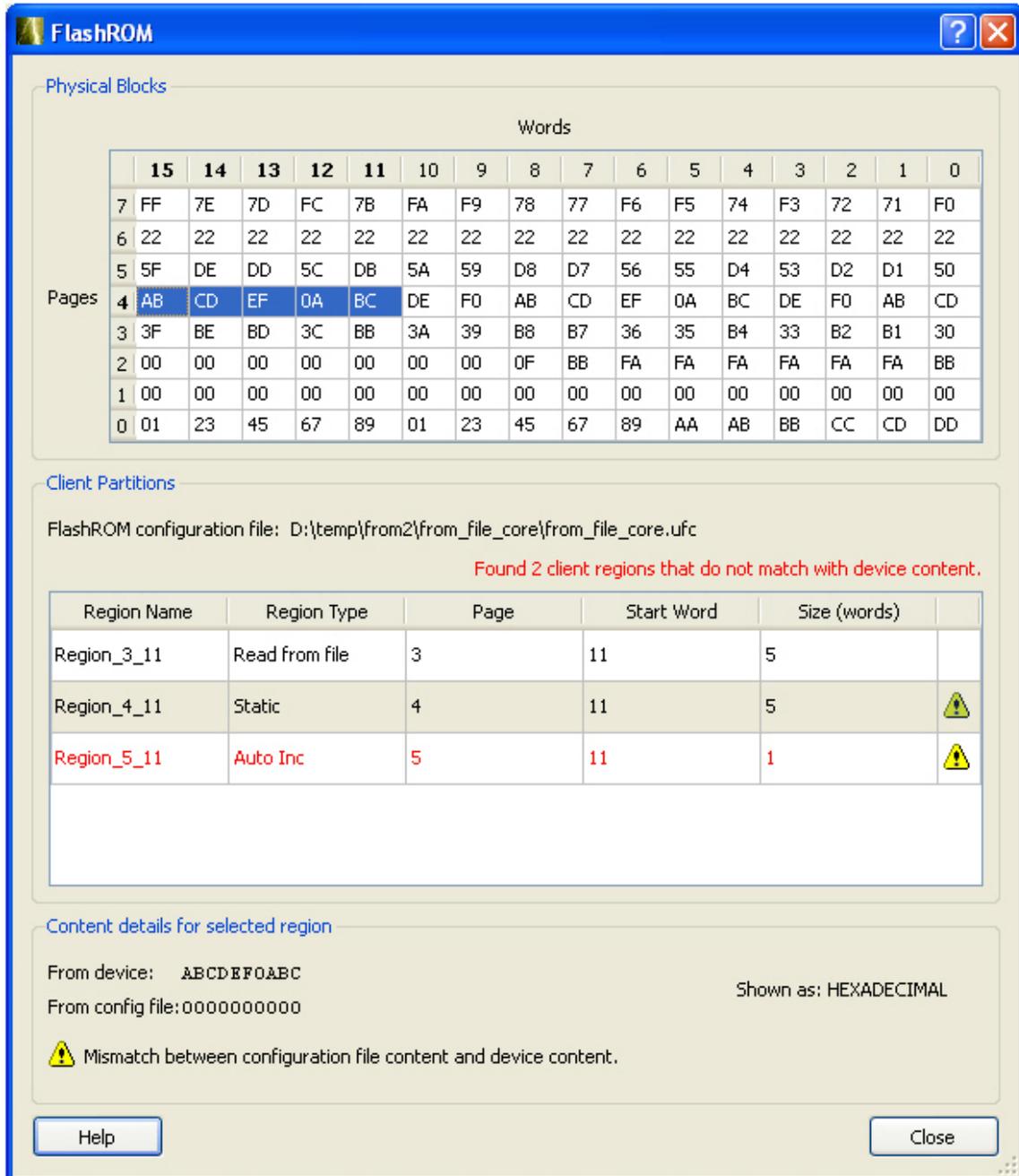


Figure 30 · FlashROM Content Dialog Box

Analog Block Configuration Dialog Box (SmartFusion and Fusion Only)

In the Analog Block Configuration dialog box, you can:

- View the channel configuration on your analog system and identify if/how the channels are configured.
- Compare with the design configuration from the Analog System Builder for Fusion and SmartDesign MSS Configurator for SmartFusion.

The values displayed for each channel vary depending on the device family and channel you select; the Channel configuration register read from the ACM is shown for each analog channel. Individual, decoded bit

fields of the register are listed immediately beneath (as described in the Fusion and SmartFusion handbook). The dialog box may display the following values:

Fusion Device:

- Analog MUX select
- Internal chip T monitor
- Scaling factor control
- Current monitor switch
- Current monitor drive control
- Direct analog input switch
- Pad polarity - G, T, V, C pad polarity, positive or negative
- Select low/high drive
- Prescaler op amp mode

SmartFusion Device:

- Gain select
- Channel state
- Direct Input state
- Current Monitor state
- Current monitor strobe state
- Comparator state
- Hysteresis select
- Analog MUX select
- DAC input select
- Temperature monitor state
- Temperature monitor strobe state
- Vref switch state

To use the compare feature, select the **Compare with** checkbox. If the loaded PDB file contains Analog Block configuration information, the comparison appears automatically.

To use a specific Project File, click **Browse** and navigate to the Analog System Builder directory for Fusion or SmartDesign for SmartFusion. In a typical IDE project, this directory is located at:

- Fusion - <project_root>/smartgen/<analog_block_core_name>
- SmartFusion - <project root>/component/work/<SmartDesign project>/MSS_ACE_0

After specifying the compare directory, the differences (if any) are indicated in red on a channel by channel basis, as shown in the figure below.

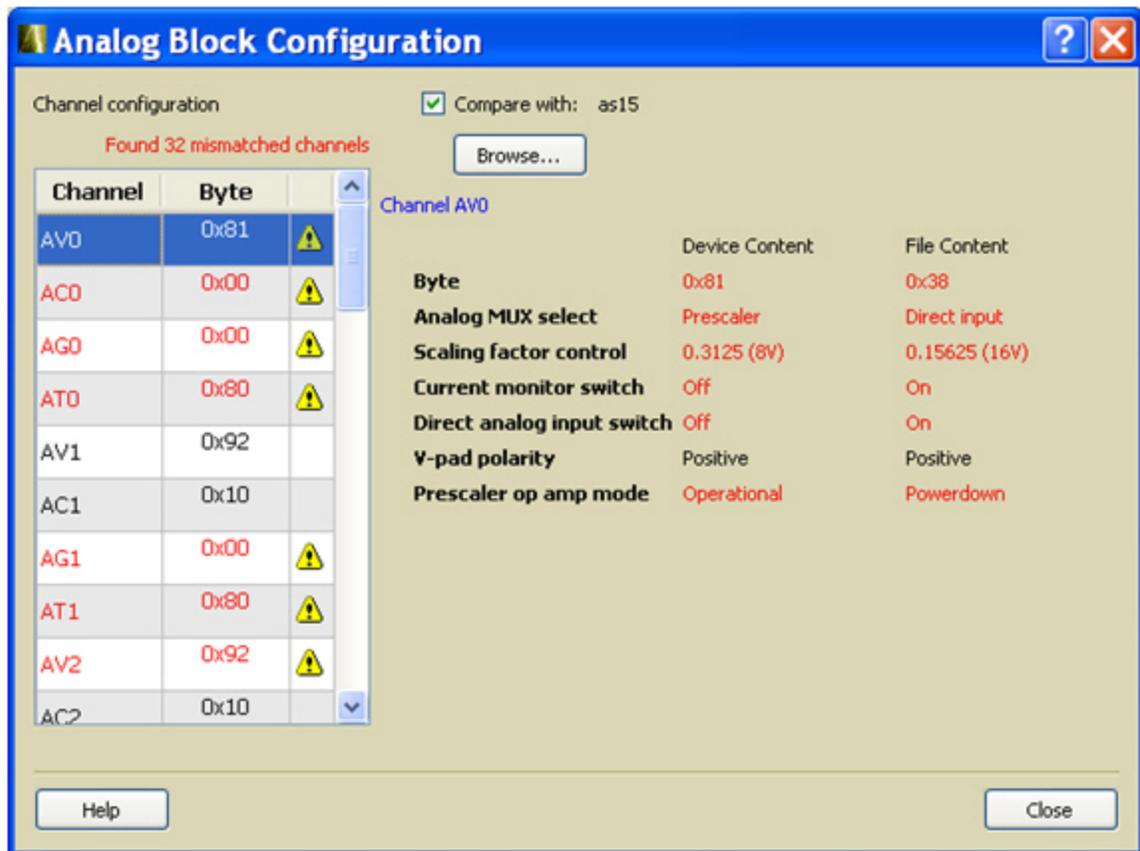


Figure 31 · Analog Block Configuration Dialog Box for a Fusion Device (Differences in Red)

SmartDebug Tcl Commands

SmartDebug Tcl Support (SmartFusion2, IGLOO2, and RTG4)

The following table lists the Tcl commands related to SmartDebug for SmartFusion2, IGLOO2, and RTG4. Click the command to view more information.

Table 1 · SmartDebug Tcl Commands

Command	Action
DDR/MDDR	
ddr_read	Reads the value of specified configuration registers pertaining to the DDR memory controller (MDDR/FDDR).
ddr_write	Writes the value of specified configuration registers pertaining to the DDR memory controller (MDDR/FDDR).
Probe	
add_to_probe_group	Adds the specified probe points to the specified probe group.
create_probe_group	Creates a new probe group.
delete_active_probe	Deletes either all or the selected active probes.
load_active_probe_list	Loads the list of probes from the file.
move_to_probe_group	Moves the specified probe points to the specified probe group.
set_live_probe	Set Live probe channels A and/or B to the specified probe point (or points).
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
read_active_probe	Reads active probe values from the device.
remove_from_probe_group	Move out the specified probe points from the group.
save_active_probe_list	Saves the list of active probes to a file.
select_active_probe	Manages the current selection of active probe points to be used by active probe READ operations.
ungroup	Disassociates the probes as group.
unset_live_probe	Discontinues the debug function and clears live probe channels.

Command	Action
DDR/MDDR	
write_active_probe	Sets the target probe point on the device to the specified value.
LSRAM	
read_lsram	Reads a specified block of large SRAM from the device.
write_lsram	Writes a seven bit word into the specified large SRAM location.
uSRAM	
read_usram	Reads a uSRAM block from the device.
write_usram	Writes a seven bit word into the specified uSRAM location.
SERDES	
prbs_test	Starts, stops, resets the error counter and reads the error counter value in PRBS tests.
loopback_test	Starts and stops the loopback tests.
serdes_lane_reset	In EPCS mode, this command resets the lane. In PCI mode, this command resets the lane, all other lanes in the link, and the corresponding PCIe controller.
serdes_read_register	Reads the SERDES register value and displays the result in the log window/console.
serdes_write_register	Writes the value to the SERDES register.
Additional Commands	
export_smart_debug_data	Exports debug data for the SmartDebug application.

Device Debug / SmartDebug Tcl Commands (SmartFusion, IGLOO, ProASIC3, and Fusion Only)

Note: Tcl commands in this section may not be supported by all device families listed above. See the individual commands for specific device support.

The following table lists the Tcl commands related to Device Debug / SmartDebug for SmartFusion and Fusion). Click the command to view more information.

Table 2 · Device Debug / SmartDebug Tcl Commands

Command	Action	Type
check_flash_memory	Performs diagnostics of the page status and data information.	Embedded Flash Memory (NVM)
compare_analog_config	Compares the content of the analog block configurations in your design against the actual values in the device.	Analog Block
compare_flashrom_client	Compares the content of the FlashROM configurations in your design against the actual values in the selected device.	FlashROM
compare_memory_client	Compares the memory client in a specific device and block.	Embedded Flash Memory (NVM)
read_analog_block_config	Reads each channel configuration on your analog system, enabling you to identify if/how each channel is configured.	Analog Block
read_device_status	Displays a summary of the selected device.	
read_flashrom	Reads the content of the FlashROM from the selected device.	FlashROM
read_flash_memory	Reads information from the NVM modules (page status and page data).	Embedded Flash Memory (NVM)
read_id_code	Reads IDCode from the device without masking any IDCode fields.	
recover_flash_memory	Removes ECC2 errors due to memory corruption by reprogramming specified flash memory (NVM) pages and initializing all pages to zeros.	Embedded Flash Memory (NVM)
sample_analog_channel	Samples analog channel; enables you to debug ADC conversion of the preconfigured analog channel (you must provide ADC conversion parameters).	
set_debug_device	Identifies the device you intend to debug.	
set_debug_programmer	Identifies the programmer you want to use	

Command	Action	Type
	for debugging (if you have more than one).	

add_to_probe_group (SmartFusion2, IGLOO2, and RTG4)

Tcl command; adds the specified probe points to the specified probe group.

```
add_to_probe_group -name probe_name -group group_name
```

Arguments

- name *probe_name*
Specifies one or more probes to add.
- group *group_name*
Specifies name of the probe group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
add_to_probe_group -name out[5]:out[5]:Q \  
                  -name grp1.out[3]:out[3]:Q \  
                  -name out.out[1].out[1]:Q \  
                  -group my_new_grp
```

check_flash_memory

The command performs diagnostics of the page status and data information as follows:

- Page Status – includes ECC2 check of the page status information, write count
- Page Data - ECC2 check

```
check_flash_memory  
[-name {device_name}]  
[-block {integer_value}]  
[-client {client_name}]  
[-startpage {integer_value}]  
[-endpage {integer_value}]  
[-access {all | status | data}]  
[-show {summary | pages}]  
[-file {filename}]
```

At a minimum you must specify -client <*name*> OR

-startpage <*page_number*> -endpage <*page_number*> -block <*number*>

Arguments

- name {*device_name*}
Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.
- block {*integer_value*}

(Optional argument; you must set -client or -startpage, -endpage and -block before use.) Specifies location of block for memory check.

-client {*client_name*}

Name of client for memory check.

-startpage {*integer_value*}

Startpage for page range; value must be an integer. You must specify a -endpage and -block along with this argument.

-endpage {*integer_value*}

Endpage for page range; value must be an integer. You must specify a -startpage and -block along with this argument.

-access {*all* / *status* / *data*}

(Optional argument; you must set -client or -startpage, -endpage and -block before use.) Specifies what NVM information to check: page status, data or both.

Value	Description
all	Shows the number of pages with corruption status, data corruption and out-of-range write count (default)
status	Shows the number of pages with corruption status and the number of pages with out-of-range write count
data	Shows only the number of pages with data corruption

-show {*summary* / *pages*}

(Optional argument; you must set -client or -startpage, -endpage and -block before use.) Specifies output level, as explained in the table below.

Value	Description
summary	Displays the summary for all checked pages (default)
pages	Displays the check results for each checked page

-file {*filename*}

(Optional argument; you must set -client or -startpage, -endpage and -block before use.) Name of output file for memory check.

Supported Families

SmartFusion, Fusion

Exceptions

None

Example

The following command checks the page status for block 0 from startpage 0 to endpage 2:

```
check_flash_memory -startpage 0 -endpage 2 -block 0
```

The following command checks the memory status for the client 'DS8bit' and saves it to the file 'checkFlashMemory.log':

```
check_flash_memory -client {DS8bit} -file {checkFlashMemory.log}
```

compare_analog_config

Compares the content of the analog block configurations in your design against the actual values in the device. In a typical SoC project, this directory is located at `<project_root>/smartgen/<analog_block_core_name>`.

```
compare_analog_config  
[-name "device_name"] -mem_file_dir "mem_file_directory"  
[-file "filename"]
```

Arguments

-name {*device_name*}

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-mem_file_dir {*mem_file_directory*}

Location of memory file.

-file {*filename*}

Output filename.

Supported Families

Fusion

Exceptions

None

Example

The following command reads the analog block configuration in the directory `F:/tmp/Analog_Block` and saves the data in the logfile `compare_analogReport.log`:

```
compare_analog_config -mem_file_dir {F:/tmp/Analog_Block} -file  
{compare_analogReport.log}
```

The following command reads the analog block configuration information in the device 'AFS600' in the directory `F:/tmp/Analog_Block` and saves the data in the log file `compare_analogReport.log`:

```
compare_analog_config -name {AFS600} -mem_file_dir {F:/tmp/Analog_Block} -file  
{compare_analogReport.log}
```

compare_flashrom_client

Compares the content of the FlashROM configurations in your design against the actual values in the selected device.

```
compare_flashrom_client [-name {device_name}] [-file {filename}]
```

Arguments

-name {*device_name*}

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-file {*filename*}

Optional file name for FlashROM compare log.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following command saves the FlashROM data to the file 'FlashRomCompReport.log':

```
compare_flashrom_client -file {FlashRomCompReport.log}
```

The following command compares the data in the device 'A3P250' and saves the data in the logfile 'FlashRomCompReport.log':

```
compare_flashrom_client -name {A3P250} -file {FlashRomCompReport.log}
```

compare_memory_client

Compares the memory client in a specific device and block.

```
compare_memory_client [-name {device_name}] [-block integer_value] -client {client_name} [-file {filename}]
```

Arguments

-name { *device_name* }

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set debug device](#) command.

-block {*integer_value*}

(Optional argument; you must set -client.) Specifies location of block for memory compare.

-client {*client_name*}

Name of client for memory compare.

-file {*filename*}

Optional file name.

Supported Families

SmartFusion and Fusion

Exceptions

None

Example

The following command compares the memory in the client 'DS32' on the device 'AFS600':

```
compare_memory_client -client DS32 -name AFS600
```

The following command compares the data at block '0' to the client 'DS8bit':

```
compare_memory_client -block 0 -client {DS8bit}
```

The following command compares the memory in the device 'AFS600' at block '0' to the memory client 'DS8bit':

```
compare_memory_client -name {AFS600} -block 0 -client {DS8bit}
```

The following command compares the memory at block '1' to the memory client 'DS8bit' and saves the information in a log file to F:/tmp/NVMCompReport.log:

```
compare_memory_client -block 1 -client {DS8bit} -file {F:/tmp/NVMCompReport.log}
```

create_probe_group (SmartFusion2, IGLOO2, and RTG4)

Tcl command; creates a new probe group.

```
create_probe_group -name group_name
```

Arguments

-name *group_name*

Specifies the name of the new probe group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
create_probe_group -name my_new_grp
```

delete_active_probe

Tcl command; deletes either all or the selected active probes.

Note: You cannot delete an individual probe from the Probe Bus.

```
Delete_active_probe -all | -name probe_name
```

Arguments

-all

Deletes all active probe names.

-name *probe_name*

Deletes the selected probe names.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
Delete -all          <- deletes all active probe names
Delete -name out[5]:out[5]:Q \
      -name my_grp1.out[1]:out[1]:Q          <- deletes the selected probe names
Delete -name my_grp1 \
      -name my_bus          <- deletes the group, bus and their members.
```

ddr_read (SmartFusion2, IGLOO2, and RTG4)

Tcl command; reads the value of specified configuration registers pertaining to the DDR memory controller (MDDR/FDDR).

```
ddr_read -block ddr_name -name reg_name
```

Arguments

-block <fddr || mddr || east_fddr || west_fddr>

- Specifies which DDR configurator is used in the Libero design.
- SmartFusion2 and IGLOO2 - fddr and mddr

- RTG4 - east_fddr and west_fddr
- name *register_name*
- Specifies which configuration registers need to be read.
 - A complete list of registers is available in the DDR Interfaces User Guides for the respective families.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

Read DDR Controller register DDRC_DYN_REFRESH_1_CR for a configured FDDR block on a SmartFusion2 or IGLOO2 device:

```
ddr_read -block fddr -name DDRC_DYN_REFRESH_1_CR
```

Returns

Returns 16-bit hexadecimal value.

The result of the command in the example above will be:

```
Register Name: DDRC_DYN_REFRESH_1_CR Value: 0x1234  
"ddr_read" command succeeded.
```

ddr_write (SmartFusion2, IGLOO2, and RTG4)

Tcl command; writes the value of specified configuration registers pertaining to the DDR memory controller (MDDR/FDDR).

```
ddr_write -block ddr_name -name reg_name -value hex_value
```

Arguments

-block <fddr || mddr || east_fddr || west_fddr>

- Specifies which DDR configurator is used in the Libero design.
- SmartFusion2 and IGLOO2 - fddr and mddr
- RTG4 - east_fddr and west_fddr

-name *register_name*

- Specifies which configuration registers need to be read.
- A complete list of registers is available in the DDR Interfaces User Guides for the respective families.

-value *hex_value*

- Specifies the value to be written into the specified register of a given block.
- Hex_value in the form of "0x12FA".

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

Write a 16-bit value DDR Controller register DDRC_DYN_REFRESH_1_CR for a configured FDDR block on a SmartFusion2 or IGLOO2 device:

```
ddr_write -block fddr -name DDRC_DYN_REFRESH_1_CR -value 0x123f
```

Returns

Returns if the command succeeded or failed to execute.

"ddr_write" command succeeded

export_smart_debug_data (SmartFusion2, IGLOO2, and RTG4)

Tcl command; exports debug data for the SmartDebug application.

```
export_smart_debug_data [device_components] [bitstream_components] [-file_name {file} [-  
export_dir {dir}]
```

The command corresponds to the Export SmartDebug Data tool in Libero. The command creates a file with the extension "ddc" that contains data based on selected options. This file is used by SmartDebug to create a new SmartDebug project, or it can be imported into a device in SmartDebug.

- If you not specify any design components, all components available in the design will be included by default.
- The generate_bitstream parameter is required if you want to generate bitstream file and include it in the exported file.
 - o You must specify the bitstream components you want to include in the generated bitstream file or all available components will be included.
 - o If you choose to include bitstream, and the design has custom security, the custom security bitstream component must be included.

Arguments

device_components

The following device components can be selected. Specify "1" to include the component, and "0" if you do not want to include the component.

```
-probes <1|0>  
-package_pins <1|0>  
-memory_blocks <1|0>  
-envm_data <1|0>  
-security_data <1|0>  
-chain <1|0>  
-programmer_settings <1|0>  
-io_states <1|0>
```

bitstream_components

The following bitstream components can be selected. Specify "1" to include the component, and "0" if you do not want to include the component.

```
-generate_bitstream <1|0>  
-bitstream_security <1|0>  
-bitstream_fabric <1|0>  
-bitstream_envm <1|0>  
-file_name file
```

Name of exported file with extension "ddc".

```
-export_dir dir
```

Location where DDC file will be exported. If omitted, design export folder will be used.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

The following example shows the export_smart_debug_data command with all parameters:

```
export_smart_debug_data \  
-file_name {sd1} \  
-export_dir {d:\sd_prj\test3T\designer\sd1\export} \  
-probes 1 \  
-package_pins 0 \  
-memory_blocks 1 \  
-envm_data 0 \  
-security_data 1 \  
-chain 1 \  
-programmer_settings 1 \  
-ios_states 1 \  
-generate_bitstream 0 \  
-bitstream_security 0 \  
-bitstream_fabric 0 \  
-bitstream_envm 0
```

The following example shows the command with no parameters:

```
export_smart_debug_data
```

load_active_probe_list

Tcl command; loads the list of probes from the file.

```
load_active_probe_list -file file_path
```

Arguments

-file *file_path*

The input file location.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
load_active_probe_list -file "./my_probes.txt"
```

loopback_test (SmartFusion2, IGLOO2, and RTG4)

Tcl command; used to start and stop the loopback tests.

```
loopback_test [-deviceName device_name] -start -serdes num -lane num -type LoopbackType  
loopback_test [-deviceName device_name] -stop -serdes num -lane num
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see the SmartDebug User's Guide for details).

-start

Starts the loopback test.

-stop

Stops the loopback test.

-serdes *num*

Serdes block number. Must be between 0 and 4 and varies between dies.

-lane *num*

Serdes lane number. Must be between 0 and 4

-type *LoopbackType*

Specifies the loopback test type. Must be *meso* (PCS Far End PMA RX to TX Loopback)

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
loopback_test -start -serdes 1 -lane 1 -type meso
loopback_test -start -serdes 0 -lane 0 -type plesio
loopback_test -start -serdes 1 -lane 2 -type parallel
loopback_test -stop -serdes 1 -lane 2
```

move_to_probe_group (SmartFusion2, IGLOO2, and RTG4)

Tcl command; moves the specified probe points to the specified probe group.

Note: Probe points related to a bus cannot be moved to another group.

```
move_to_probe_group -name probe_name -group group_name
```

Arguments

-name *probe_name*

Specifies one or more probes to move.

-group *group_name*

Specifies name of the probe group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
move_to_probe_group -name out[5]:out[5]:Q \  
                    -name grp1.out[3]:out[3]:Q \  
                    -group my_grp2
```

prbs_test (SmartFusion2, IGLOO2, and RTG4)

Tcl command; used in PRBS test to start, stop, reset the error counter and read the error counter value.

```
prbs_test [-deviceName device_name] -start -serdes num -lane num [-near] -pattern PatternType
prbs_test [-deviceName device_name] -stop -serdes num -lane num
prbs_test [-deviceName device_name] -reset_counter -serdes num -lane num
prbs_test [-deviceName device_name] -read_counter -serdes num -lane num
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see the SmartDebug User's Guide for details).

-start

Starts the prbs test.

-stop

Stops the prbs test.

`-reset_counter`

Resets the prbs error count value to 0.

`-read_counter`

Reads and prints the error count value.

`-serdes num`

Serdes block number. Must be between 0 and 4 and varies between dies.

`-lane num`

Serdes lane number. Must be between 0 and 4.

`-near`

Corresponds to near-end (on-die) option for prbs test. Not specifying implies off-die.

`-pattern PatternType`

The pattern sequence to use for PRBS test. It can be one of the following:

prbs7, *prbs11*, *prbs23*, or *prbs31*

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
prbs_test -start -serdes 1 -lane 0 -near -pattern prbs11
prbs_test -start -serdes 2 -lane 2 -pattern custom -value all_zeros
prbs_test -start -serdes 0 -lane 1 -near -pattern user -value 0x0123456789ABCDEF0123
```

read_active_probe (SmartFusion2, IGLOO2, and RTG4)

Tcl command; reads active probe values from the device. The target probe points are selected by the [select_active_probe](#) command.

```
read_active_probe [-deviceName device_name] [-name probe_name] [-file file_path]
```

Arguments

`-deviceName device_name`

Parameter is optional if only one device is available in the current configuration.

`-name probe_name`

Instead of all probes, read only the probes specified. The probe name should be prefixed with bus or group name if the probe is in the bus or group.

`-file file_path`

Optional. If specified, redirects output with probe point values read from the device to the specified file.

Note: When the user tries to read at least one signal from the bus/group, the complete bus or group is read. The user is presented with the latest value for all the signals in the bus/group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
Read_active_probe -file output.txt
Read_active_probe -name grp1.out[3]:out[3]:Q -file output.txt
```

read_analog_block_config

Reads each channel configuration on your analog system, enabling you to identify if/how each channel is configured.

```
read_analog_block_config [-name {device_name}] [-file {filename}]
```

Arguments

-name {*device_name*}

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-file {*filename*}

(Optional) Identifies the name of the file to which read results will be saved.

Supported Families

Fusion

Exceptions

None

Example

The following command reads the analog block configuration information in the device 'AFS600':

```
read_analog_block_config -name {AFS600}
```

read_device_status

Displays the Device Information report; the Device Information report is a complete summary of your device state, analog block test values, user information, factory serial number and security information..

```
read_device_status [-name {device_name}] [-file {filename}]
```

Arguments

-name *device_name*

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-file {*filename*}

(Optional) Identifies the name of the file to which read results will be saved.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following reads device info from the 'AFS600' device.

```
read_device_status -name AFS600
```

read_id_code

The command reads IDCode from the device without masking any IDCode fields. This is the raw IDCode from the silicon.

Note: Being able to read the IDCode is an indication that the JTAG interface is working correctly.

```
read_id_code [-name {device_name}]
```

Arguments

-name *device_name*

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following command reads the IDCODE from the device 'AFS600':

```
read_id_code -name {AFS600}
```

read_flashrom

Reads the content of the FlashROM from the selected device.

```
read_flashrom [-name {device_name}] [-mapping {logical | physical}] [-file {filename}]
```

Arguments

-name *device_name*

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-mapping {*logical* | *physical*}

(Optional) Specifies how the data read from the UFROM is mapped. Values are explained in the table below.

Value	Description
logical	Logical mapping (default)
physical	Physical mapping

-file {*filename*}

(Optional) Identifies the name of the file to which read results will be saved.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following reads the FROM content on the device 'AFS600' and sets to physical mapping:

```
read_flashrom -name {AFS600} -mapping {physical}
```

read_flash_memory

The command reads information from the NVM modules. There are two types of information that can be read:

- Page Status – includes ECC2 status, write count, access protection
- Page Data

```
read_flash_memory
[-name {device_name}]
[-block {integer_value}]
[-client {client_name}]
[-startpage {integer_value}]
[-endpage {integer_value}]
[-access {all | status | data}]
[-file {filename}]
```

At a minimum you must specify `-client <name>` OR

```
-startpage <page_number> -endpage <page_number> -block <number>
```

Arguments

`-name {device_name}`

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

`-block {integer_value}`

(Optional argument; you must set `-client` or `-startpage` and `-endpage` before use.) Specifies location of block for memory read.

`-client {client_name}`

Name of client for memory read.

`-startpage {integer_value}`

Startpage for page range; value must be an integer. You must specify a `-endpage` and `-block` along with this argument.

`-endpage {integer_value}`

Endpage for page range; value must be an integer. You must specify a `-startpage` and `-block` along with this argument.

`-access {all | status | data}`

(Optional argument; you must set `-client` or `-startpage`, `-endpage` and `-block` before use.) Specifies what eNVM information to check: page status, data or both.

Value	Description
all	Shows the number of pages with corruption status, data corruption and out-of-range write count (default)
status	Shows the number of pages with corruption status and the number of pages with out-of-range write count
data	Shows only the number of pages with data corruption

-file {*filename*}

(Optional argument; you must set -client or -startpage, -endpage and -block before use.) Name of output file for memory read.

Supported Families

SmartFusion, Fusion

Exceptions

None

Example

The following command reads the flash memory for the client 'DS8bit' and reports the data in a logfile 'readFlashMemoryReport.log':

```
read_flash_memory -client {DS8bit} -file {readFlashMemoryReport.log}
read_flash_memory -startpage 0 -endpage 2 -block 0 -access {data}
```

read_lsram (SmartFusion2, IGLOO2, and RTG4)

Tcl command; reads a specified block of large SRAM from the device.

```
read_lsram [-deviceName device_name] -block block_name [-file filename]
```

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

-block *block_name*

Specifies the name for the target block.

-file *filename*

Optional; specifies the output file name for the data read from the device.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- You must set a debug file
- Array must be programmed and active
- Security locks may disable this function

Example

Reads the SRAM Block sram_block1 from the sf2 device and writes it to the file sram_block_output.

```
read_lsram [-deviceName sf2] -block sram_block1 [-file sram_block_output]
```

read_usram (SmartFusion2 and IGLOO2)

Tcl command; reads a uSRAM block from the device.

```
read_lsram [-deviceName device_name] -block block_name [-file filename]
```

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

-block *block_name*

Specifies the name for the target block.

-file *filename*

Optional; specifies the output file name for the data read from the device.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- You must set a debug file
- Array must be programmed and active
- Security locks may disable this function

Example

Reads the uSRAM Block `usram_block2` from the `sf2` device and writes it to the file `sram_block_output`.

```
read_usram [-deviceName sf2] -block usram_block2 [-file sram_block_output]
```

recover_flash_memory

The command removes ECC2 errors due to memory corruption by reprogramming specified flash memory (NVM) pages and initializing all pages to zeros. The recovery affects data blocks and auxiliary blocks.

The write counters of the corrupted pages might not be accurate due to corruption. The recovery operation will not change state of the page write counters.

Use the `check_flash_memory` command to detect flash memory errors.

```
recover_flash_memory  
[-name {device_name}]  
[-block {integer_value}]  
[-client {client_name}]  
[-startpage {integer_value}]  
[-endpage {integer_value}]
```

At a minimum you must specify `-client <name>` OR

```
-startpage <page_number> -endpage <page_number> -block <number>
```

Arguments

-name {*device_name*}

Optional user-defined device name. The device name is not required if there is only one device in the current configuration, or a device has already been selected using the [set_debug_device](#) command.

-block {*integer_value*}

(Optional argument; you must set `-client` or `-startpage` and `-endpage` before use.) Specifies location of block for memory recovery.

-client {*client_name*}

Name of client for memory recovery.

`-startpage {integer_value}`

Startpage for page range; value must be an integer. You must specify a `-endpage` and `-block` along with this argument.

`-endpage {integer_value}`

Endpage for page range; value must be an integer. You must specify a `-startpage` and `-block` along with this argument.

Supported Families

SmartFusion, Fusion

Exceptions

None

Example

The following command recovers flash memory data in the client 'DS8bit':

```
recover_flash_memory -client {DS8bit}
```

The following command recovers flash memory from block 0, startpage 0, and endpage 3:

```
recover_flash_memory -block 0 -startpage 0 -endpage 3
```

remove_from_probe_group (SmartFusion2, IGLOO2, and RTG4)

Tcl command; moves out the specified probe points from the group. That is, the moved out probe points won't be associated with any probe group.

Note: Probes cannot be moved out from the bus.

```
remove_from_probe_group -name probe_name
```

Arguments

`-name probe_name`

Specifies one or more probe points to move out.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

The following command moves out two probes from `my_grp2`.

```
Move_out_of_probe_group -name my_grp2.out[3]:out[3]:Q \  
-name my_grp2.out[3]:out[3]:Q
```

sample_analog_channel

Performs analog-to-digital conversion of a selected analog channel. This command is used when debugging the Analog Subsystem and is performed on the pre-configured analog channel with user-supplied ADC conversion parameters. The command also performs digital filtering using a single-pole low-pass filter if you opt to use it.

```
sample_analog_channel [(-name {name})*]  
[-resolution {8 | 10 | 12}]  
[-clock_periods {int_value}]  
[-clock_divider {int_value}]  
[-num_samples { int_value}]
```

```
[-filtering_factor {real_value}]
[-initial_value {int_value}]
[-show_details {yes | no}]
[-file {filename}]
```

Arguments

-name { *name* }

Specifies the analog channel to be sampled. Channel name is a combination of the channel type followed by the channel index. Valid channel names are listed in the table below.

Family	Valid Channel Name
Fusion	AV<n>, AT<n>, AC<n>
SmartFusion	AV<n>, AT<n>, AC<n>, ADC<n>

The maximum number of channels depends on particular device type; refer to the Analog Block specification in the device handbook.

-resolution {8 | 10 | 12}

ADC conversion resolution. Specifies bit size of the conversion results. Selection of certain resolutions may affect timing parameter valid ranges. See your device handbook for details.

-clock_periods {int_value }

Parameter specifying sampling time: Sampling_time = clock_periods * adc_clock_period.

-clock_divider {int_value }

Specifies clock prescaling factor.

-num_samples { int_value }

Optional argument that specifies the number of samples to be performed by the ADC. Default number of samples is 1. Selecting multiple vs single sample will change appearance of the generated report. For the single sample a single result is shown and if "show_details" is set to "yes" then detailed status of the ADC register is also shown.

If multiple samples are requested then the results are printed in a table. If the digital filtering is enabled the table also includes filtered results.

-filtering_factor {real_value}

Optional argument that specifies the filtering factor if multiple samples requested. The default value of 1.0 disables digital filtering.

-initial_value {int_value}

Optional argument that specifies the initial value for the digital averaging filter. The value is specified in ADC register counts. Default value is set to 0. Specifying this parameter improves filtering process during initial samples.

-show_details {yes | no}

Optional argument that specifies the level of the report output. Detailed output includes initial user-supplied conversion parameters. For the single-sampling case final output also includes detailed content of ADC register after sampling.

-file {filename}

Optional argument. Specifies name of output file for conversion results.

Supported Families

SmartFusion and Fusion

Exceptions

None

Example

The following example performs single sample analog-to-digital conversion for channel AV0:

```
sample_analog_channel -channel AV0 -resolution 8 -clock_periods 4 -clock_divider 4
```

Example with multiple sampling and digital signal filtering for AV0:

```
sample_analog_channel -channel AV0 -resolution 10 -clock_periods 4 -clock_divider 4 -  
num_samples 10 -filtering_factor 2.5
```

save_active_probe_list

Tcl command; saves the list of active probes to a file.

```
save_active_probe_list -file file_path
```

Arguments

-file *file_path*

The output file location.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
save_active_probe_list -file "./my_probes.txt"
```

select_active_probe (SmartFusion2, IGLOO2, and RTG4)

Tcl command; manages the current selection of active probe points to be used by active probe READ operations. This command extends or replaces your current selection with the probe points found using the search pattern.

```
select_active_probe [-deviceName device_name] [-name probe_name_pattern] [-reset true/false]
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration..

-name *probe_name_pattern*

Specifies the name of the probe. Optionally, search pattern string can specify one or multiple probe points. The pattern search characters "*" and "?" also can be specified to filter out the probe names.

-reset *true / false*

Optional parameter; resets all previously selected probe points. If name is not specified, empties out current selection.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

The following command selects three probes. In the below example, "grp1" is a group and "out" is a bus..

```
Select_active_probe -name out[5]:out[5]:Q  
Select_active_probe -name out.out[1]:out[1]:Q \  
-name out.out[3]:out[3]:Q \  
-name out.out[5]:out[5]:Q
```

serdes_lane_reset

Tcl command. In EPCS mode, this command resets the lane. In PCI mode, this command resets the lane, all other lanes in the link, and the corresponding PCIe controller. The result is shown in the log window/console.

```
serdes_lane_reset -serdes num -lane num
```

Arguments

-serdes *num*

The SERDES block number. It must be between 0 and varies between dies. It must be one of the SERDES blocks used in the design.

lane *num*

The SERDES lane number. It must be between 0 and 3. It must be one of the lanes enabled for the block in the design.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
serdes_lane_reset -serdes 0 -lane 0
```

In EPCS mode, resets Lane 0, for block 0. In PCI mode, resets Lane 0 for block 0, all other lanes in the same link for block 0

```
serdes_lane_reset -serdes 5 -lane 3
```

Errors

The following errors result in the failure of the Tcl command and the corresponding message on the smart debug log window:

When the “-serdes” parameter is not specified:

```
Error: Required parameter 'serdes' is missing.
```

```
Error: Failure when executing Tcl script. [Line 26: Error in command serdes_lane_reset]
```

```
Error: The Execute Script command failed.
```

When the “-lane” parameter is not specified:

```
Error: Required parameter 'lane' is missing.
```

```
Error: Failure when executing Tcl script. [Line 26: Error in command serdes_lane_reset]
```

```
Error: The Execute Script command failed.
```

When “block number” is not specified:

```
Error: Parameter 'serdes' has illegal value.
```

```
Error: Failure when executing Tcl script. [Line 26: Error in command serdes_lane_reset]
```

```
Error: The Execute Script command failed.
```

When “lane number” is not specified:

```
Error: Required parameter 'lane' is missing.
```

```
Error: Failure when executing Tcl script. [Line 26: Error in command serdes_lane_reset]
```

```
Error: The Execute Script command failed.
```

When “block number” is invalid:

```
Error: Phy Reset: Serdes block number should be one of the following: 0
```

```
Error: The command 'serdes_lane_reset' failed.
```

```
Error: Failure when executing Tcl script. [Line 26]
```

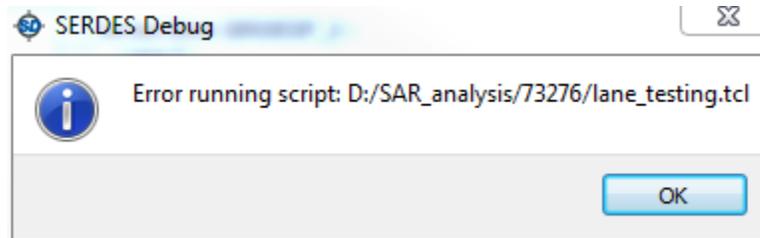
```
Error: The Execute Script command failed.
```

Note: Only the SERDES blocks used the design will be mentioned in the above list.

When “lane number” is invalid:

Error: Phy Reset: Serdes lane number should be between 0 and 3.
Error: The command 'serdes_lane_reset' failed.
Error: Failure when executing Tcl script. [Line 26]
Error: The Execute Script command failed.

For all the above scenarios, the following message appears:



serdes_read_register (SmartFusion2, IGLOO2, and RTG4)

Tcl command; reads the SERDES register value and displays the result in the log window/console.

```
serdes_read_register -serdes num [ -lane num ] -name REGISTER_NAME
```

Arguments

-serdes *num*

SERDES block number. Must be between 0 and varies between dies.

-lane *num*

SERDES lane number. Must be between 0 and 3.

The lane number must be specified when the lane register is used. Otherwise, the command will fail.

When the lane number is specified along with the SYSTEM or PCIe register, the command will fail with an error message, as the lane is not applicable to them.

-name *REGISTER_NAME*

Name of the SERDES register.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
serdes_read_register -serdes 0 -name SYSTEM_SER_PLL_CONFIG_HIGH
```

```
serdes_read_register -serdes 0 -lane 0 -name CR0
```

[serdes_write_register](#)

[UG0567: RTG4 High-Speed Serial Interfaces User Guide](#) (includes all SERDES register names)

[UG0447: SmartFusion2 and IGLOO2 FPGA High-Speed Serial Interfaces User Guide](#)

serdes_write_register (SmartFusion2, IGLOO2, and RTG4)

Tcl command; writes the value to the SERDES register. Displays the result in the log window/console.

```
serdes_write_register -serdes num [-lane num ] -name REGISTER_NAME -value 0x1234
```

Arguments

-serdes *num*

SERDES block number. Must be between 0 and 5 and varies between dies.

-lane *num*

SERDES lane number. Must be between 0 and 3.

The lane number should be specified when the lane register is used. Otherwise, the command will fail.

When the lane number is specified along with the SYSTEM or PCIe register, the command will fail with an error message, as the lane is not applicable to them.

-name *REGISTER_NAME*

Name of the SERDES register.

-value

Specify the value in hexadecimal format.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
serdes_write_register -serdes 0 -name SYSTEM_SER_PLL_CONFIG_HIGH -value 0x5533
```

See Also

[serdes_read_register.htm](#)

[UG0567: RTG4 High-Speed Serial Interfaces User Guide](#) (includes all SERDES register names)

[UG0447: SmartFusion2 and IGLOO2 FPGA High-Speed Serial Interfaces User Guide](#)

set_debug_device

Identifies the device you intend to debug.

```
set_debug_device -name {device_name}
```

Arguments

name {*device_name*}

Device name. The device name is not required if there is only one device in the current configuration.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following example identifies the device 'A3P250' for debugging:

```
set_debug_device -name {A3P250}
```

set_debug_programmer

Identifies the programmer you want to use for debugging (if you have more than one). The name of the programmer is the serial number on the bar code label on the FlashPro programmer.

```
set_debug_programmer -name {programmer_name}
```

Arguments

-name {*programmer_name*}

Programmer name is the serial number on the bar code label of the FlashPro programmer.

Supported Families

SmartFusion, IGLOO, ProASIC3 and Fusion

Exceptions

None

Example

The following example selects the programmer 10841

```
set_debug_programmer -name {10841}
```

set_live_probe (SmartFusion2, IGLOO2, and RTG4)

Tcl command; set_live_probe channels A and/or B to the specified probe point(s). At least one probe point must be specified. Only exact probe name is allowed (i.e. no search pattern that may return multiple points).

```
set_live_probe [-deviceName device_name] [-probeA probe_name] [-probeB probe_name]
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug user guide for details).

-probeA *probe_name*

Specifies target probe point for the probe channel A.

-probeB *probe_name*

Specifies target probe point for the probe channel B.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- The array must be programmed and active
- Active probe read or write operation will affect current settings of Live probe since they use same probe circuitry inside the device
- Setting only one Live probe channel affects the other one, so if both channels need to be set, they must be set from the same call to set_live_probe
- Security locks may disable this function

- In order to be available for Live probe, ProbeA and ProbeB I/O's must be reserved for Live probe respectively

Example

Sets the Live probe channel A to the probe point A12 on device sf2.

```
set_live_probe [-deviceName sf2] [-probeA A12]
```

ungroup (SmartFusion2, IGLOO2, and RTG4)

Tcl command; disassociates the probes as a group.

```
nngroup -name group_name
```

Arguments

-name *group_name*

Name of the group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
ungroup -name my_grp4
```

unset_live_probe

Tcl command; discontinues the debug function and clears both live probe channels (Channel A and Channel B). An all zeros value is shown for both channels in the oscilloscope.

Note: For RTG4, only one probe channel (Probe Read Data Pin) is available.

```
unset_live_probe [-deviceName device_name]
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see the SmartDebug User's Guide for details).

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- The array must be programmed and active.
- Active probe read or write operation affects current of Live Probe settings, because they use the same probe circuitry inside the device.
- Security locks may disable this function.

Example

The following example unsets both live probe channels (Channel A and Channel B) from the device sf2.

```
unset_live_probes [-deviceName sf2]
```

write_active_probe (SmartFusion2, IGLOO2, and RTG4)

Tcl command; sets the target probe point on the device to the specified value. The target probe point name must be specified.

```
write_active_probe [-deviceName device_name] -name probe_name -value true/false  
-group_name group_bus_name -group_value "hex-value" | "binary-value"
```

Arguments

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration.

-name *probe_name*

Specifies the name for the target probe point. Cannot be a search pattern.

-value *true* | *false* *hex-value* | *binary-value*

Specifies values to be written.

True = High

False = Low

-group_name *group_bus_name*

Specify the group or bus name to write to complete group or bus.

-group_value "*hex-value*" | "*binary-value*"

Specify the value for the complete group or bus.

Hex-value format: "<size>'h<value>"

Binary-value format: "<size>'b<value>"

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

```
write_active_probe -name out[5]:out[5]:Q -value true <-- write to a single probe  
write_active_probe -name grp1.out[3]:out[3]:Q -value low <-- write to a probe in the group  
write_active_probe -group_name grp1 -group_value "8'hF0" <-- write the value to complete group  
write_active_probe -group_name out -group_value "8'b11110000" \  
-name out[2]:out[2]:Q -value true <-- write multiple probes at the same time.
```

write_lsrām (SmartFusion2, IGLOO2, and RTG4)

Tcl command; writes a seven bit word into the specified large SRAM location.

```
write_lsrām [-deviceName device_name] -block block_name] -offset offset_value -value value
```

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

-block *block_name*

Specifies the name for the target block.

-offset *offset_value*

Offset (address) of the target word within the memory block.

-value *value*

Value to be written to the target location.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- You must set a debug file
- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x1A to the device sf2 in the block sram_block1 with an offset of 16.

```
write_lsrsm [-deviceName sf2] -block sram_block1 -offset 16 -value 0x1A
```

write_usram (SmartFusion2, IGLOO2, and RTG4)

Tcl command; writes a seven bit word into the specified uSRAM location.

```
write_usram [-deviceName device_name] -block block_name] -offset offset_value -value value
```

-deviceName *device_name*

Parameter is optional if only one device is available in the current configuration or set for debug (see SmartDebug help for details).

-block *block_name*

Specifies the name for the target block.

-offset *offset_value*

Offset (address) of the target word within the memory block.

-value *value*

Seven-bit value to be written.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Exceptions

- You must set a debug file
- Array must be programmed and active
- The maximum value that can be written is 0x1FF
- Security locks may disable this function

Example

Writes a value of 0x1A to the device sf2 in the block usram_block2 with an offset of 16.

```
write_usram [-deviceName sf2] -block usram_block2 -offset 16 -value 0x1A
```

Solutions to Common Issues Using SmartDebug

Embedded Flash Memory (NVM) - Failure when Programming/Verifying

If the Embedded Flash Memory failed verification when executing the PROGRAM_NVM, VERIFY_NVM or PROGRAM_NVM_ACTIVE_ARRAY action, the failing page may be [corrupted](#). To confirm and address this issue:

1. In the **Inspect Device** window click **View Flash Memory Content**.
2. Select the Flash Memory block and client (or page range) to retrieve from the device.
3. Click **Read from Device**; the retrieved data appears in the lower part of the window.
4. Click **View Detailed Status** to [check the NVM Status](#).

Note: You can use the `check_flash_memory` and `read_flash_memory` Tcl commands to perform diagnostics similar to the commands outlined above.

5. If the NVM is [corrupted](#) you must reset the affected NVM pages.

To reset the affected NVM pages, either re-program the pages with your original data or 'zero-out' the pages by using the Tcl command [recover_flash_memory](#).

If the Embedded Flash Memory failed verification when executing a VERIFY_NVM or VERIFY_NVM_ACTIVE_ARRAY action, the failure may be due to the change of content in your design. To confirm this, repeat steps 1-3 above.

Note: NVM corruption is still possible when writing from user design. Check NVM status for confirmation.

Analog System Not Working as Expected

If the Analog System is not working correctly, it may be due the following:

1. System supply issue. To troubleshoot:
 - Physically verify that all the supplies are properly connected to the device and they are at the proper level. Then confirm by running the Device Status.
 - Physically verify that the relevant channels are correctly connected to the device.
2. Analog system is not properly configured. You can confirm this by [examining the Analog System](#).

ADC Not Sampling the Correct Value

If the ADC is sampling all zero values then the wrong analog pin may be connected to the system, or the analog pin is disconnected. If that is not the case and the ADC is not sampling the correct value, it may be due to the following:

1. System supply issues - Run the device status to confirm.
2. Analog system is not configured at all - To confirm, [read out the ACM configuration](#) and verify if the ACM content is all zero.
3. Analog system is not configured correctly - To confirm, [read out the ACM configuration](#) and verify that the configuration is as expected .

4. Once analog block configuration has been confirmed, you can use the [sample_analog_channel](#) Tcl command for debug sampling of the analog channel with user-supplied sampling parameters.

If you have access to your Analog System Builder settings project (<Libero IDE project>/Smartgen/AnalogBlock), you may use the [compare function provided by the tool](#).

Frequently Asked Questions

How do I unlock the device security so I can debug?

You must provide the PDB file with a User Pass Key in order to unlock the device and continue debugging. If you do not have a PDB with User Pass Key, you can [create a PDB file in FlashPro](#) (if you know the Pass Key value).

How do I export a report?

You can export three reports from the SmartDebug GUI: Device Status, Client Detailed Status from the NVM, or the Compare Client Content report from the NVM. Each of those reports can be saved and printed.

If using a Tcl command, you can use the `-file <filename>` option for the following commands:

[read_flash_memory](#)
[check_flash_memory](#)
[compare_memory_client](#)
[read_device_status](#)
[read_flashrom](#)
[read_analog_block_config](#)
[sample_analog_channel](#)
[compare_flashrom_client](#)
[compare_analog_config](#)

For example, you can use the following command to export the content of the client 'datastore1' in NVM block 0 to the report file datastore1_content.txt:

```
read_flash_memory -client "datastore1" -file {C:\temp\datastore1_content.txt}
```

For more information about Tcl commands supported by SmartDebug, see [SmartDebug Tcl Commands](#).

How do I generate diagnostic reports for my target device?

A set of diagnostic reports can be generated for your target device depending on which silicon feature you are debugging. A set of Tcl commands are available to export those reports. The following is a summary of those Tcl commands based on the silicon features.

When using the `-file` parameter, ensure that you use a different file name for each command so you do not overwrite the report content. If you do not specify the `-file` option in the Tcl, the output results will be directed to the FlashPro log window.

For the overall device:

[read_device_status](#)
[read_id_code](#)

For FlashROM:

[compare_flashrom_client](#)
[read_flashrom](#)

For Embedded Flash Memory (NVM):

[compare_memory_client](#)
[check_flash_memory](#)
[read_flash_memory](#)

For Analog Block:

[read_analog_block_config](#)
[compare_analog_config](#)
[sample_analog_channel](#)

To execute the Tcl command, from the **File** menu choose **Run Script**.

Where can I find files to compare my contents/settings?

FlashROM

You can compare the FlashROM content in the device with the data in the PDB file. You can find the PDB in the <Liberio IDE project>/Designer/Impl directory.

Embedded Flash Memory (NVM)

You can compare the Embedded Flash Memory content in the device with the data in the PDB file. You can find the PDB in the <Liberio IDE project>/Designer/Impl directory.

Analog System

You can compare the Analog System configuration in the device with the data in the loaded PDB file or in the Analog System folder. Go to:

- Fusion devices - <Liberio IDE project>/Smartgen/AnalogBlock
- SmartFusion devices - <Liberio IDE Project>/component/<SmartDesign Project>/MSS_ACE_0

The tool automatically identifies the necessary files in the selected folder for comparison.

What is a UFC file? What is an EFC file?

UFC is the User FlashROM Configuration file, generated by the FlashROM configurator; it contains the partition information set by the user. It also contains the user-selected data for region types with static data.

However, for AUTO_INC and READ_FROM_FILE, regions the UFC file contains only:

- Start value, end value, and step size for AUTO_INC regions, and
- File directory for READ_FROM_FILE regions

EFC is the Embedded Flash Configuration file, generated by the Flash Memory Builder in the Project Manager [Catalog](#); it contains the partition information and data set by the user.

Both UFC and EFC information is embedded in the PDB when you generate the PDB file.

Is my FPGA fabric enabled?

When your FPGA fabric is programmed, you will see the following statement under Device State in the Device Status report:

```
FPGA Array Status: Programmed and Enabled
```

If the FPGA fabric is not programmed, the Device State shows:

```
FPGA Array Status: Not Enabled
```

Embedded Flash Memory (NVM) Frequently Asked Questions

Is my Embedded Flash Memory (NVM) programmed?

To figure out if your NVM is programmed, read out and view the NVM content or perform verification with the PDB file.

To examine the NVM content, see the [FlashROM Memory Content Dialog Box](#).

To verify the NVM with the PDB select the [VERIFY](#) or [VERIFY NVM](#) action in FlashPro.

How do I display Embedded Flash Memory (NVM) content in the Client partition?

You must load your PDB into your FlashPro project in order to view the Embedded Flash Memory content in the Client partition. To view NVM content in the client partition:

1. Load your PDB into your FlashPro project.
2. Click **Inspect Device**.
3. Click **View Flash Memory Content**.
4. Choose a block from the drop-down menu.
5. Select a client.
6. Click **Read from Device**. The Embedded Flash Memory content from the device appears in the Flash Memory dialog box.

See the [Flash Memory Dialog Box](#) topic for more description on viewing the NVM content.

How do I know if I have Embedded Flash Memory (NVM) corruption?

When Embedded Flash Memory is [corrupted](#), [checking Embedded Flash Memory](#) may return with any or all of the following page status:

- ECC1/ECC2 failure
- Page write count exceeds the 10-year retention threshold
- Page write count is invalid
- Page protection is set illegally (set when it should not be)

See the [How do I interpret data in the Flash Memory \(NVM\) Status Report?](#) topic for details.

If your Embedded Flash Memory is corrupted, you can recover by reprogramming with original design data. Alternatively, you can 'zero-out' the pages by using the Tcl command [recover_flash_memory](#).

Why does Embedded Flash Memory (NVM) corruption happen?

Embedded Flash Memory corruption occurs when Embedded Flash Memory programming is interrupted due to:

- Supply brownout; monitor power supplies for brownout conditions. For SmartFusion monitor the VCC_ENVM/VCC_ROSC voltage levels; for Fusion, monitor VCC_NVM/VCC_OSC.
- Reset signal is not properly tied off in your design. Check the Embedded Memory reset signal.

How do I recover from Embedded Flash Memory corruption?

Reprogram with original design data or 'zero-out' the pages by using the Tcl command [recover_flash_memory](#).

What is a JTAG IR-Capture value?

JTAG IR-Capture value contains private and public device status values. The public status value in the value read is ISC_DONE, which indicates if the FPGA Array is programmed and enabled.

The ISC_DONE signal is implemented as part of IEEE 1532 specification.

What does the ECC1/ECC2 error mean?

ECC is the Error Correction Code embedded in each Flash Memory page.

ECC1 – One bit error and correctable.

ECC2 – Two or more errors found, and not correctable.

How can I tell if my FlashROM is programmed?

To verify that your FlashROM is programmed, [read out and view the FlashROM content](#) or perform verification with the PDB file by selecting the [VERIFY](#) or [VERIFY_FROM](#) action in FlashPro.

Can I compare serialization data?

To compare the serialization data, you can read out the FlashROM content and visually check data in the serialization region. Note that a serialization region can be an AUTO_INC or READ_FROM_FILE region.

For serialization data in the AUTO_INC region, check to make sure that the data is within the specified range for that region.

For READ_FROM_FILE region, you can search for a match in the source data file.

Can I tell what security options are programmed in my device?

To determine the programmed security settings, run the Device Status option from the Inspect Device dialog and examine the Security Section in the report.

This section lists the security status of the FlashROM, FPGA Array and Flash Memory blocks.

Is my analog system configured?

To determine if the analog block is configured, run the Device Status option from the Inspect Device dialog and examine the Analog Block Section in the report. For example, the excerpt from the Device Status report below shows that the analog block status is operational:

```
Analog Block:  
  OABTR Register (HEX): 0dbe37b  
  3.3V (vdd33): PASS  
  1.5V (vdd15): PASS  
  Bandgap: PASS  
  -3.3V (vddn33): PASS  
  ADC Reference: PASS  
  FPGA_Good: PASS  
  Status: Analog Block is operational
```

If you read out an all zero value when [examining the Analog System Configuration](#), it is possible that the Analog System is not configured.

You need to compare your analog system configuration with the design configuration from the Analog System Builder.

The -3.3V (vddn33) voltage is optional.

How do I interpret data in the Device Status report?

The Device Status Report generated from the FlashPro SmartDebug Feature contains the following sections:

- IDCode (see below)
- [User Information](#)
- [Device State](#)
- [Analog Block](#) (SmartFusion and Fusion only)
- [Factory Data](#)
- [Security Settings](#)

IDCode

The IDCode section shows the raw IDCode read from the device. For example, in the Device Status report for an AFS600 device, you will find the following statement:

```
IDCode (HEX): 233261cf
```

The IDCode is compliant to IEEE 1149.1. The following table lists the IDCode bit assignments:

Table 3 · IDCode Bit Assignments

Bit Field (little endian)	Example Bit Value for AFS600 (HEX)	Description
Bit [31-28] (4 bits)	2	Silicon Revision
Bit [27-12] (16 bits)	3326	Device ID
Bit [11-0] (12 bits)	1cf	IEEE 1149.1 Manufacturer ID for Microsemi

Device Status Report: User Info

The User Information section reports the information read from the User ROW (UROW) of IGLOO, ProASIC3, SmartFusion and Fusion devices. The User Row includes user design information as well as troubleshooting information, including:

- Design name (10 characters max)
- Design check sum (16-bit CRC)
- Last programming setup used to program/erase any of the silicon features.
- FPGA Array / Fabric programming cycle count

For example:

```
User Information:
```

```
UROW data (HEX): 603a04e0a1c2860e59384af926fe389f
```

```
Programming Method: STAPL
```

```
Programmer: FlashPro3
```

```
Programmer Software: FlashPro vX.X
```

```
Design Name: ABCBASICTO
```

```
Design Check Sum: 603A
```

Algorithm Version: 19
 Array Prog. Cycle Count: 19

Table 4 · Device Status Report User Info Description

Category	Field	Description
User Row Data	(Example) UROW data (HEX): 603a04e0a1c2860e59384af926fe389f	Raw data from User Row (UROW)
Programming Troubleshooting Info	(Example) Programming Method: STAPL Programmer: FlashPro3 Programmer Software: FlashPro v8.6 Algorithm Version: 19	Known programming setup used. This includes: Programming method/file, programmer and software. It also includes programming Algorithm version used.
Design Info	(Example) Design Name: ABCASICTO Design Check Sum: 603A	Design name (limited to 10 characters) and check sum. Design check sum is a 16-bit CRC calculated from the fabric (FPGA Array) datastream generated for programming. If encrypted datastream is generated selected, the encrypted datastream is used for calculating the check sum.

Device Status Report: Device State

The device state section contains:

- IR-Capture register value, and
- The FPGA status

The IR-Capture is the value captured by the IEEE1149.1 instruction register when going through the IR-Capture state of the IEEE 1149.1 state machine. It contains information reflecting some of the states of the devices that is useful for troubleshooting.

One of the bits in the value captured is the ISC_DONE value, specified by IEEE 1532 standard. When the value is '1' it means that the FPGA array/fabric is programmed and enabled. This is available for IGLOO, ProASIC3, SmartFusion and Fusion devices.

For example:

Device State:

IRCapture Register (HEX): 55

FPGA Array Status: Programmed and enabled

For a blank device:

Device State:

IRCapture Register (HEX): 51

FPGA Array Status: Not enabled

Device Status Report: Analog Block

The Analog block of the SmartFusion and Fusion devices monitors some of the key power supplies needed by the device to function. These power supply status is captured in the OABTR test register in the Analog block.

For example, if you run Device Status when the Fabric and Analog configuration is programmed and powered up successfully the report indicates:

```

Analog Block:
OABTR Register (HEX): 0dbe3bb
3.3V (vdd33): PASS
1.5V (vdd15): PASS
Bandgap: PASS
-3.3V (vddn33): PASS
ADC Reference: PASS
FPGA_Good: PASS
Status: Analog Block is operational
  
```

Table 5 · Device Status Report - Analog Block Description

Analog Block Status	Description
OABTR Register	RAW data captured from the device
3.3V (vdd33)	Vcc33a supply status
1.5V (vdd15)	Vccnmv supply status
Bandgap	Internal bandgap supply status
ADC Reference	ADC reference voltage status
-3.3V (vddn33)	Vddn33 supply status (optional voltage)
FPGA Good	FPGA array or Fabric status

If the Fusion device is erased, the report indicates:

```

Analog Block:
OABTR Register (HEX): 188e3ba
3.3V (vdd33): PASS
1.5V (vdd15): PASS
Bandgap: PASS
-3.3V (vddn33): FAIL
ADC Reference: FAIL
FPGA_Good: FAIL
Status: Analog Block is non-operational
Analog Block is not programmed
  
```

Device Status Report: Factory Data

The Factory Data section lists the Factory Serial Number (FSN).

Each of the IGLOO, ProASIC3, SmartFusion and Fusion devices has a unique 48-bit FSN.

Device Status Report: Security

The security section shows the security options for the FPGA Array, FlashROM and Flash Memory (NVM) block that you programmed into the device.

For example, using a Fusion AFS600 device:

```

Security:
Security Register (HEX): 0000000088c01b
FlashROM
Write/Erase protection: Off
Read protection: Off
Encrypted programming: Off
FPGA Array
Write/Erase protection: Off
Verify protection: Off
Encrypted programming: Off
FlashMemory Block 0
Write protection: On
Read protection: On
Encrypted programming: Off
FlashMemory Block 1
Write protection: On
Read protection: On
Encrypted programming: Off
  
```

Table 6 · Device Status Report - Security Description

Security Status Info	Description
Security Register (HEX)	Raw data captured from the device's security status register
Write/Erase Protection	Write protection is applicable to FlashROM, FPGA Array (Fabric) and Flash Memory (NVM) blocks. When On, the Silicon feature is write/erase protected by user passkey.
Read Protection	Read protection is applicable to FlashROM and Flash Memory (NVM) blocks. When On, the Silicon feature is read protected by user passkey.
Verify Protection	Verify Protection is only applicable to FPGA Array (Fabric) only. When On, the FPGA Array require user passkey for verification. Reading back from the FPGA Array (Fabric) is not supported. Verification is accomplished by sending in the expected data for verification.
Encrypted Programming	Encrypted Programming is supported for FlashROM, FPGA Array (Fabric) and Flash Memory (NVM) blocks. When On, the silicon feature is enable for encrypted programmed. This allows field design

Security Status Info	Description
	update with encrypted datastream so the user design is protected.

Encrypted Programming

To allow encrypted programming of the features, the target feature cannot be Write/Erase protected by user passkey.

The security settings of each silicon feature when they are enabled for encrypted programming are listed below.

FPGA Array (Fabric)

Write/Erase protection: Off
 Verify protection: Off
 Encrypted programming: On

Set automatically by Designer or FlashPro when you select to enable encrypted programming of the FPGA Array (Fabric). This setting allows the FPGA Array (Fabric) to be programmed and verified with an encrypted datastream.

FlashROM

Write/Erase protection: Off
 Read protection: On
 Encrypted programming: On

Set automatically by Designer or FlashPro when you select to enable encrypted programming of the FlashROM. This setting allows the FlashROM to be programmed and verified with an encrypted datastream.

FlashROM always allows verification. If encrypted programming is set, verification has to be performed with encrypted datastream.

Designer and FlashPro automatically set the FlashROM to be read protected by user passkey when encrypted programming is enabled. This protects the content from being read out of the JTAG port after encrypted programming.

Flash Memory (NVM) Block

Write/Erase protection: Off
 Read protection: On
 Encrypted programming: On

The above setting is set automatically set by Designer or FlashPro when you select to enable encrypted programming of the Flash Memory (NVM) block. This setting allows the Flash Memory (NVM) block to be programmed with an encrypted datastream.

The Flash Memory (NVM) block does not support verification with encrypted datastream.

Designer and FlashPro automatically set the Flash Memory (NVM) block to be read protected by user passkey when encrypted programming is enabled. This protects the content from being read out of the JTAG port after encrypted programming.

How do I interpret data in the Flash Memory (NVM) Status Report?

The Embedded Flash Memory (NVM) Status Report generated from the FlashPro SmartDebug feature consists of the page status of each NVM page. For example:

```
Flash Memory Content [ Page 34 to 34 ]
FlashMemory Page #34:
Status Register(HEX): 00090000
Status ECC2 check: Pass
```

```

Data ECC2 Check: Pass
Write Count: Pass (2304 writes)
Total number of pages with status ECC2 errors: 0
Total number of pages with data ECC2 errors: 0
Total number of pages with write count out of range: 0
FlashMemory Check PASSED for [ Page 34 to 34 ]
The 'check_flash_memory' command succeeded.
The Execute Script command succeeded.
  
```

Table 7 - Embedded Flash Memory Status Report Description

Flash Memory Status Info	Description
Status Register (HEX)	Raw page status register captured from device
Status ECC2 Check	Check for ECC2 issue in the page status
Data ECC2 Check	Check for ECC2 issue in the page data
Write Count	<p>Check if the page-write count is within the expected range.</p> <p>The expected write count is greater than or equal to:</p> <p>6,384 - SmartFusion devices 2,288 - Fusion devices</p> <p>Note: Write count, if corrupted, cannot be reset to a valid value within the customer flow; invalid write count will not prevent device from being programmed with the FlashPro tool.</p> <p>The write count on all good eNVM pages is set to be 2288 instead of 0 in the manufacturing flow. The starting count of the eNVM is 2288. Each time the page is programmed or erased the count increments by one. There is a Threshold that is set to 12288, which equals to 3 * 4096.</p> <p>Since the threshold can only be set in multiples of 4096 (2¹²), to set a 10,000 limit, the Threshold is set to 12288 and the start count is set to 2288; and thus the eNVM has a 10k write cycle limit. After the write count exceeds the threshold, the STATUS bit goes to 11 when attempting to erase/program the page.</p>

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 **650. 318.8044**

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

For Microsemi SoC Products Support, visit <http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support>.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group [home page](http://www.microsemi.com/soc/), 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. Visit [About Us](#) for [sales office listings](#) and [corporate contacts](#).

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@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 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

© 2016 Microsemi Corporation. All rights reserved. Microsemi and the Microsemi logo are trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace, and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs, and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,600 employees globally. Learn more at www.microsemi.com.

Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer's responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided "as is, where is" and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.