SmartDebug for Libero SoC v11.8

User 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.**

Microsemi.

Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email:
sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center 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; enterprise storage and communication solutions; 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, California, and has approximately 4,800 employees globally. Learn more at **www.microsemi.com.**

5-02-00638-2/01.17

Table of Contents

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 to go through SmartDebug project creation and import a Design Debug Data Container (DDC) file, exported from Libero, to access all 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 obtain 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*.

Supported Tools

The following table lists device family support for SmartDebug tools.

SmartDebug Support per Device Family	SmartFusion2	IGLOO2	RTG4	SmartFusion	Fusion
Live Probes	X	X	X		
Active Probes	X	X	X		
Memory Debug	X	X	X		
Probe Insertion (available only through Libero flow)	X	X	X		
View Flash Memory Content	X	X		X	X
Debug SERDES	X	X	X		
FPGA Hardware Breakpoint (Needs FHB Auto Instantiation)	X	X			
Event Counter (Needs FHB Auto Instantiation)	X	X			
Frequency Monitor (Needs FHB Auto Instantiation)	X	X			
FlashROM				X	X
Analog Block Configuration				X	X

Note: "X" indicates the tool is supported.

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.

See 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. You must have a FlashPro programmer connected to use SmartDebug.
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:

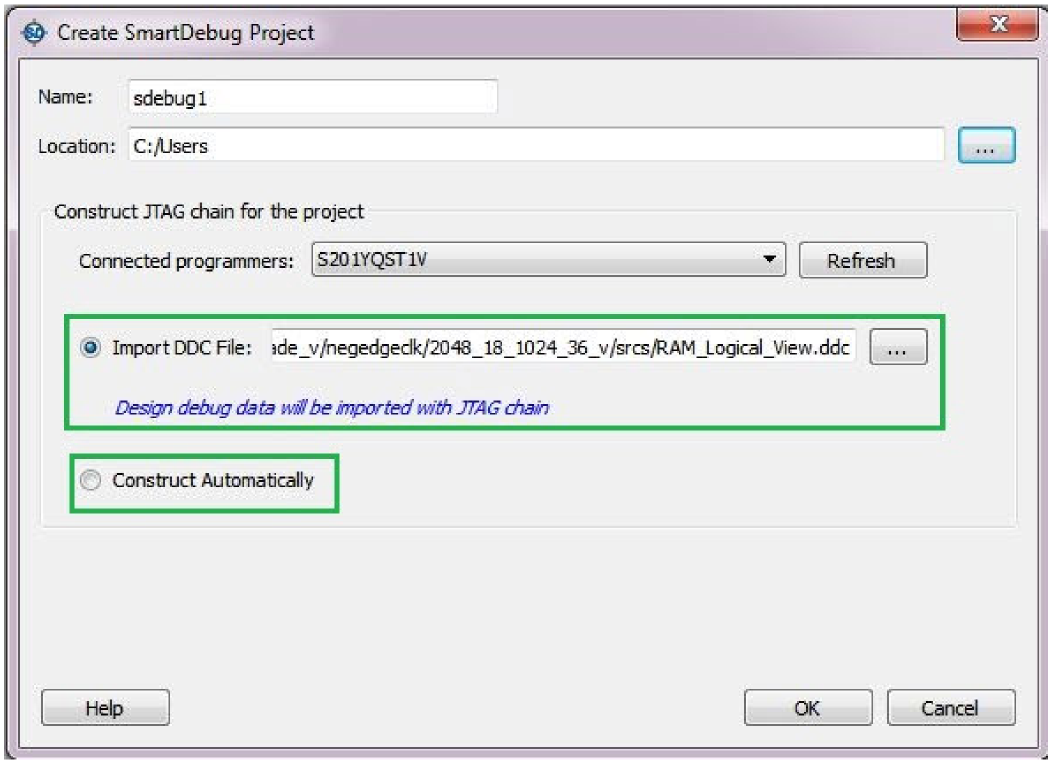
1. Create your design. You must have a FlashPro programmer connected to use SmartDebug.
2. Expand **Debug Design** and double-click **Smart Debug Design** in the Design Flow window. SmartDebug opens for your target device.
3. Click **View Device Status** to view the device status report and check for 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.


Create SmartDebug Project

Name: sdebug1

Location: C:/Users

Construct JTAG chain for the project

Connected programmers: S201YQST1V

(● Import DDC File: ade_v/negedgeclk/2048_18_1024_36_v/srcs/RAM_Logical_View.ddc

Design debug data will be imported with JTAG chain

○ Construct Automatically

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)

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

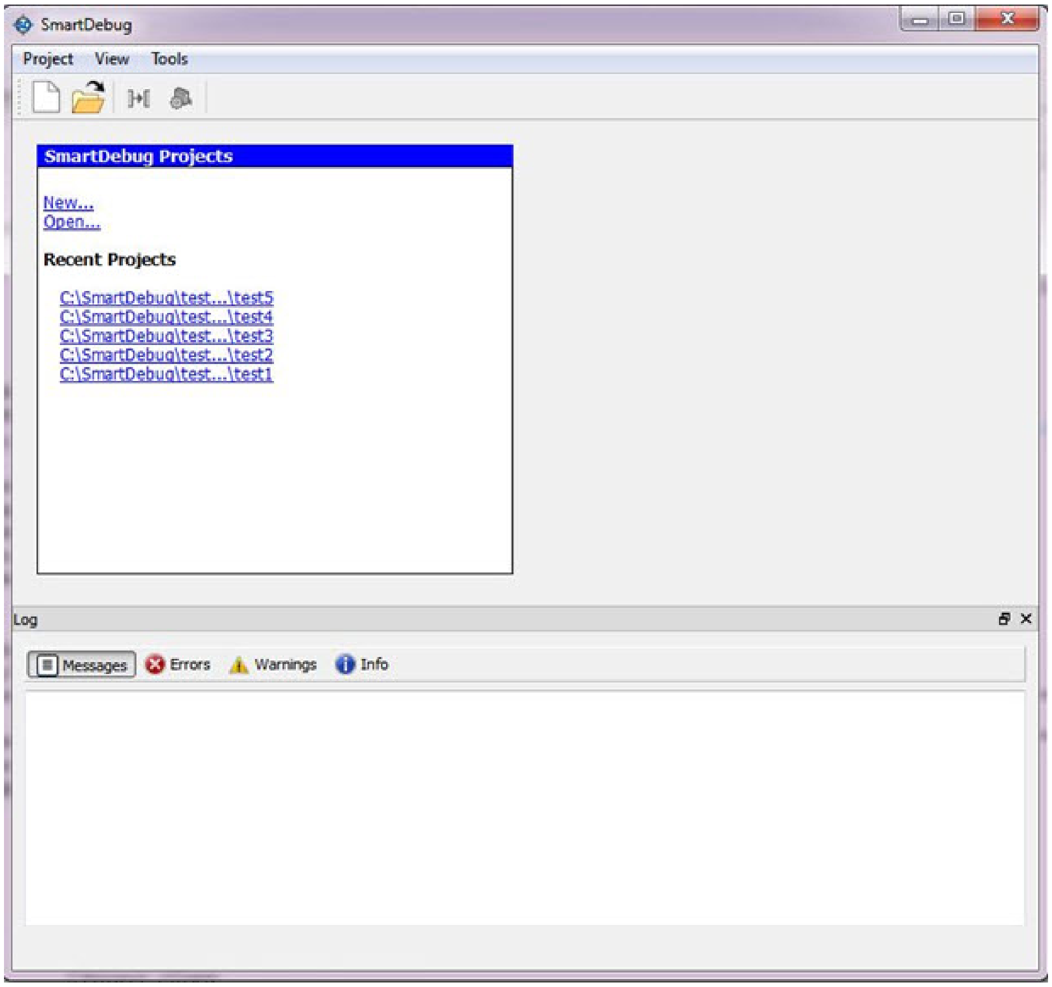

SmartDebug
Project
View
Tools
SmartDebug Projects
New...
Open...
Recent Projects
C:\SmartDebug\test...\test5
C:\SmartDebug\test...\test4
C:\SmartDebug\test...\test3
C:\SmartDebug\test...\test2
C:\SmartDebug\test...\test1
Log
Messages
Errors
Warnings
Info

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.

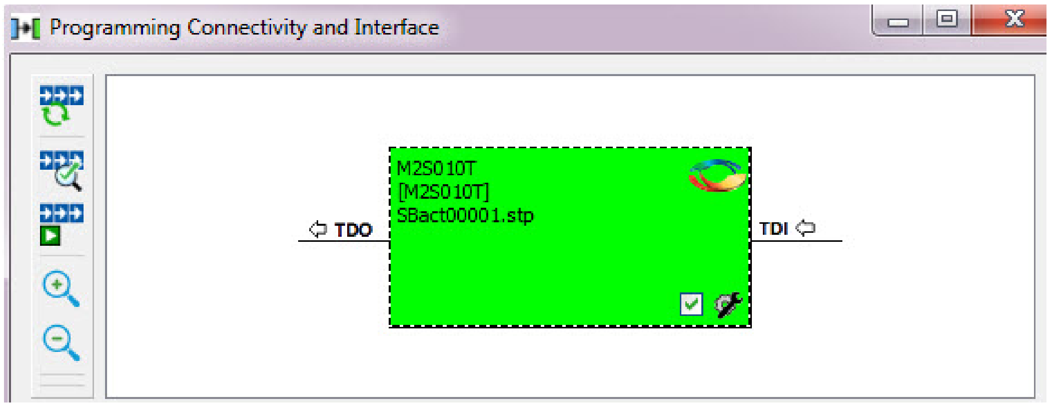
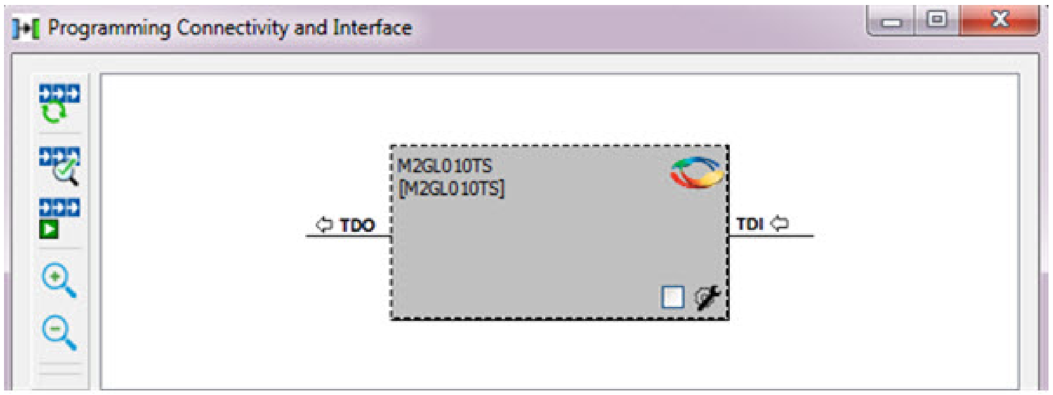


Figure 3 · Programming Connectivity and Interface Dialog Box – Project created using Import from DDC File

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.



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 Construct Chain Automatically in the 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 Chain 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.

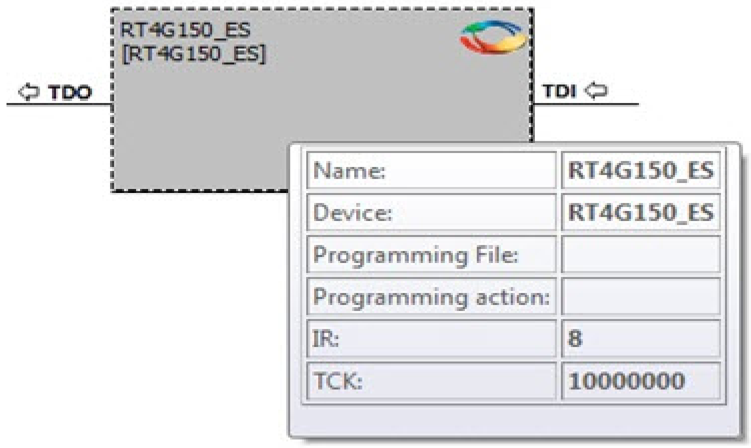
When two devices are connected in the chain, the programming actions are independent of the device. For example, if M2S090 and M2GL010 devices are connected in the chain, and the M2S090 device is to be programmed and the M2GL010 device is to be erased, both actions can be done at the same time using 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.



Name:	RT4G150_ES
Device:	RT4G150_ES
Programming File:	
Programming action:	
IR:	8
TCK:	10000000

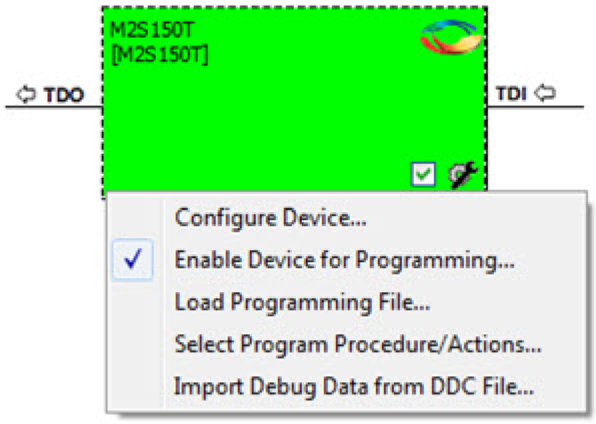
Device Chain Details

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

The following options are available when you right-click a device in the Programming Connectivity and Interface dialog box.


M2S150T
[M2S150T]
TDO
TDI
Configure Device...
✓ Enable Device for Programming...
Load Programming File...
Select Program Procedure/Actions...
Import Debug Data from DDC File...

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:

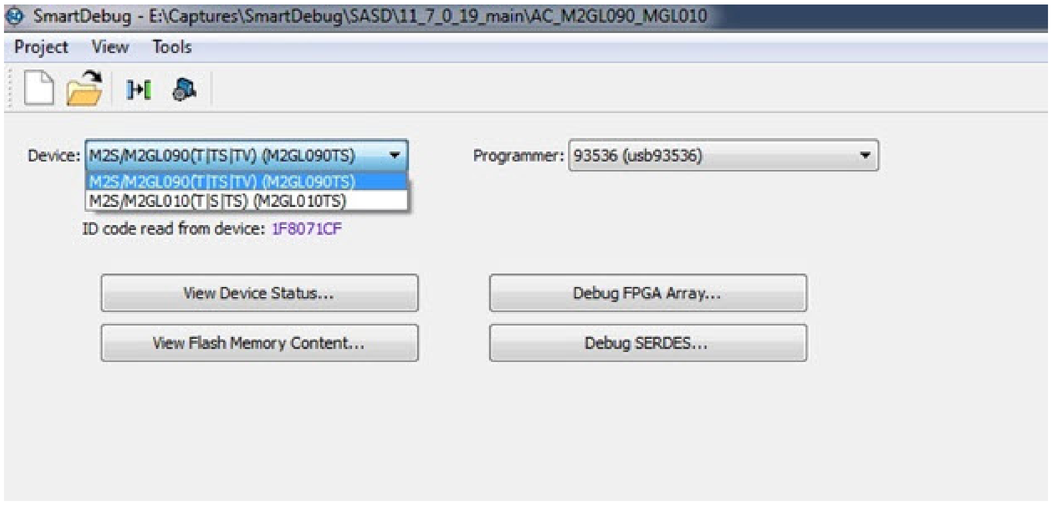
- Enable Device for Programming
- Select Programming Procedure/Actions
- Scan and Check Chain
- Run Programming Action

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

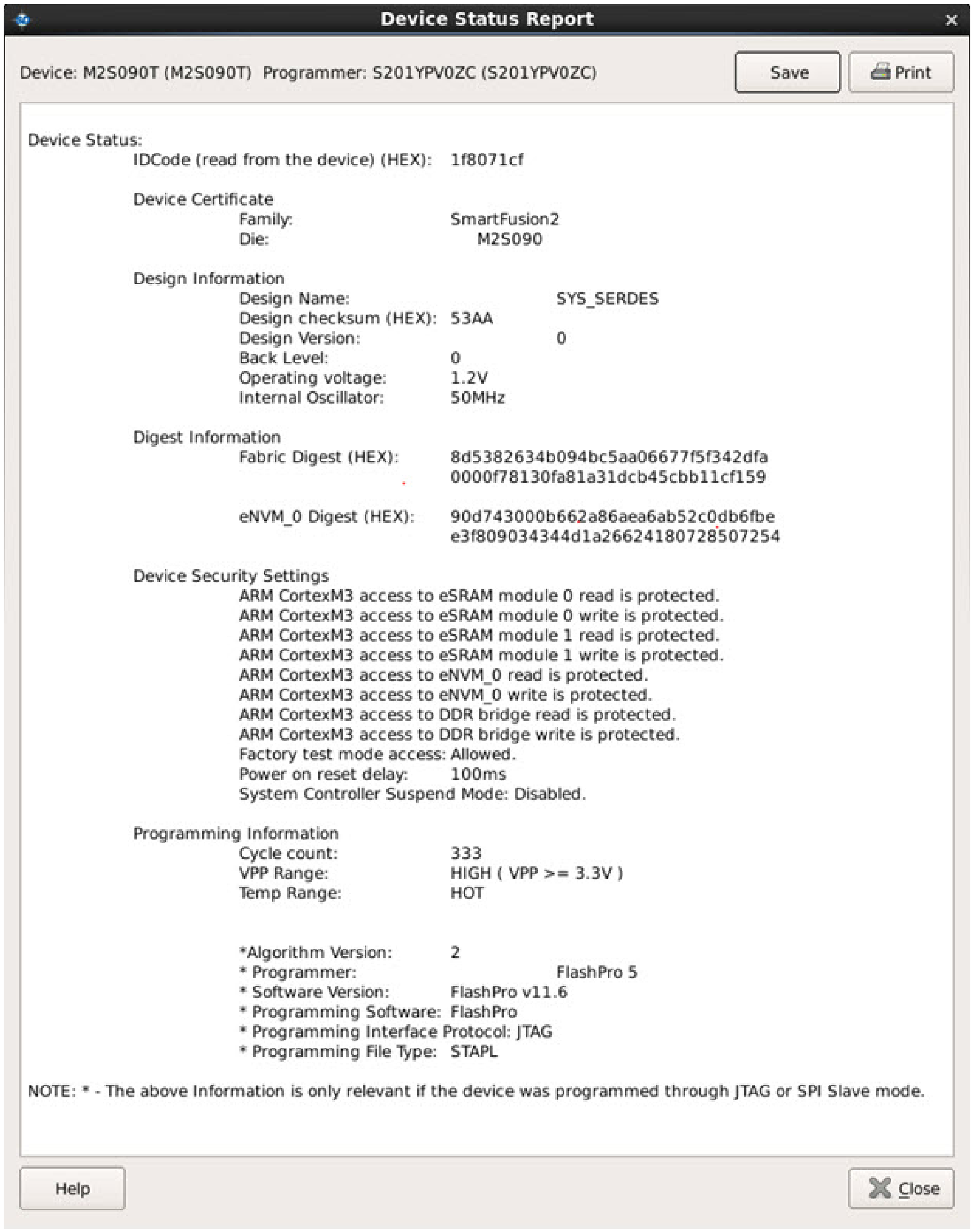
Standalone SmartDebug provides an option to select the devices connected in the JTAG chain for debug. The device debug context is not saved when another debug device is selected.


SmartDebug - E:\Captures\SmartDebug\SASD\11_7_0_19_main\AC_M2GL090_MGL010
Project
View
Tools
View Device Status...
View Flash Memory Content...
Debug FPGA Array...
Debug SERDES...

Device: M2S/M2GL090(T|TS|TV) (M2GL090TS)
M2S/M2GL090(T|TS|TV) (M2GL090TS)
M2S/M2GL010(T|S|TS) (M2GL010TS)
ID code read from device: 1F8071CF
Programmer: 93536 (usb93536)

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.



```
Device: M2S090T (M2S090T)  Programmer: S201YPV0ZC (S201YPV0ZC)

Device Status:
        IDCode (read from the device) (HEX):   1f8071cf

        Device Certificate
                Family:                         SmartFusion2
                Die:                                M2S090

        Design Information
                Design Name:                                SYS_SERDES
                Design checksum (HEX):  53AA
                Design Version:                             0
                Back Level:                     0
                Operating voltage:              1.2V
                Internal Oscillator:            50MHz

        Digest Information
                Fabric Digest (HEX):            8d5382634b094bc5aa06677f5f342dfa
                                                0000f78130fa81a31dcb45cbb11cf159

                eNVM_0 Digest (HEX):            90d743000b662a86aea6ab52c0db6fbe
                                                e3f809034344d1a26624180728507254

        Device Security Settings
                ARM CortexM3 access to eSRAM module 0 read is protected.
                ARM CortexM3 access to eSRAM module 0 write is protected.
                ARM CortexM3 access to eSRAM module 1 read is protected.
                ARM CortexM3 access to eSRAM module 1 write is protected.
                ARM CortexM3 access to eNVM_0 read is protected.
                ARM CortexM3 access to eNVM_0 write is protected.
                ARM CortexM3 access to DDR bridge read is protected.
                ARM CortexM3 access to DDR bridge write is protected.
                Factory test mode access: Allowed.
                Power on reset delay:           100ms
                System Controller Suspend Mode: Disabled.

        Programming Information
                Cycle count:                    333
                VPP Range:                      HIGH ( VPP >= 3.3V )
                Temp Range:                     HOT


                *Algorithm Version:             2
                * Programmer:                               FlashPro 5
                * Software Version:             FlashPro v11.6
                * Programming Software:  FlashPro
                * Programming Interface Protocol: JTAG
                * Programming File Type:   STAPL

NOTE: * - The above Information is only relevant if the device was programmed through JTAG or SPI Slave mode.
```

Figure 4 · 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 (SmartFusion2 and IGLOO2 only)
- Operating Voltage (SmartFusion2 and IGLOO2 only)
- Internal Oscillator (SmartFusion2 and IGLOO2 only)

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

Note: For RTG4 devices, only Lock Bit information is displayed.

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

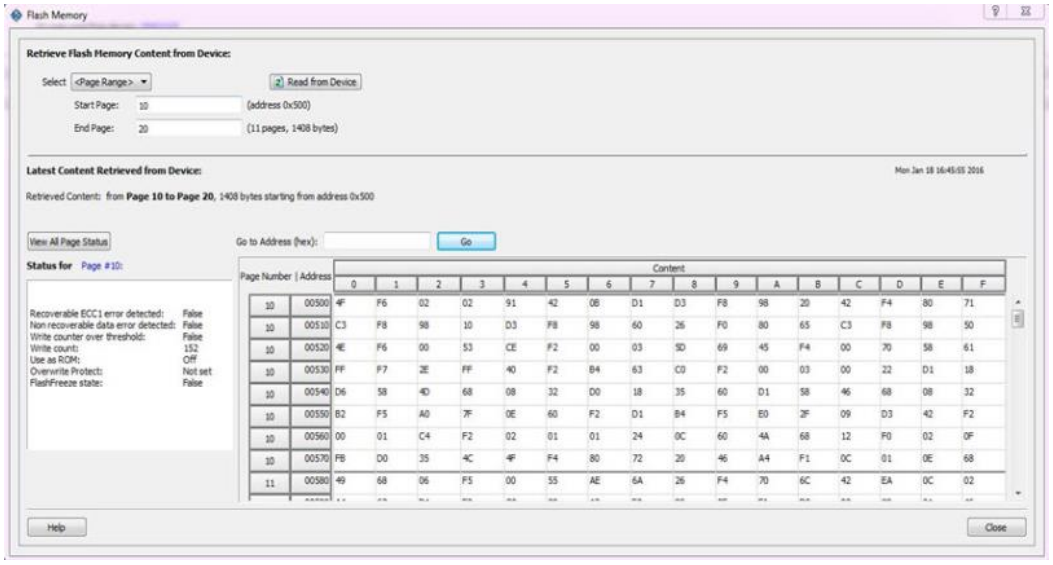
Choose the eNVM page contents to be viewed by specifying the page range (i.e., start page and the end page) and click **Read from Device** to view the values.

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)



Retrieve Flash Memory Content from Device:

Start Page: 10 (address 0x500)
End Page: 20 (11 pages, 1408 bytes)

Latest Content Retrieved from Device: Mon Jan 18 16:45:55 2016

Retrieved Content: from **Page 10 to Page 20**, 1408 bytes starting from address 0x500

Go to Address (hex):

Status for Page #10:

Recoverable ECC1 error detected:	False
Non recoverable data error detected:	False
Write counter over threshold:	False
Write count:	152
Use as ROM:	Off
Overwrite Protect:	Not set
FlashFreeze state:	False

Page Number	Address	Content															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	00500	4F	F6	02	02	91	42	0B	D1	D3	F8	98	20	42	F4	80	71
10	00510	C3	F8	98	10	D3	F8	98	60	26	F0	80	65	C3	F8	98	50
10	00520	4E	F6	00	53	CE	F2	00	03	5D	69	45	F4	00	70	58	61
10	00530	FF	F7	2E	FF	40	F2	B4	63	C0	F2	00	03	00	22	D1	18
10	00540	D6	58	40	68	08	32	D0	18	35	60	D1	58	46	68	08	32
10	00550	B2	F5	A0	7F	0E	60	F2	D1	B4	F5	E0	2F	09	D3	42	F2
10	00560	00	01	C4	F2	02	01	01	24	0C	60	4A	68	12	F0	02	0F
10	00570	FB	D0	35	4C	4F	F4	80	72	20	46	A4	F1	0C	01	0E	68
11	00580	49	68	06	F5	00	55	AE	6A	26	F4	70	6C	42	EA	0C	02

Figure 5 · Flash Memory Dialog Box for a SmartFusion2 Device (SmartDebug)

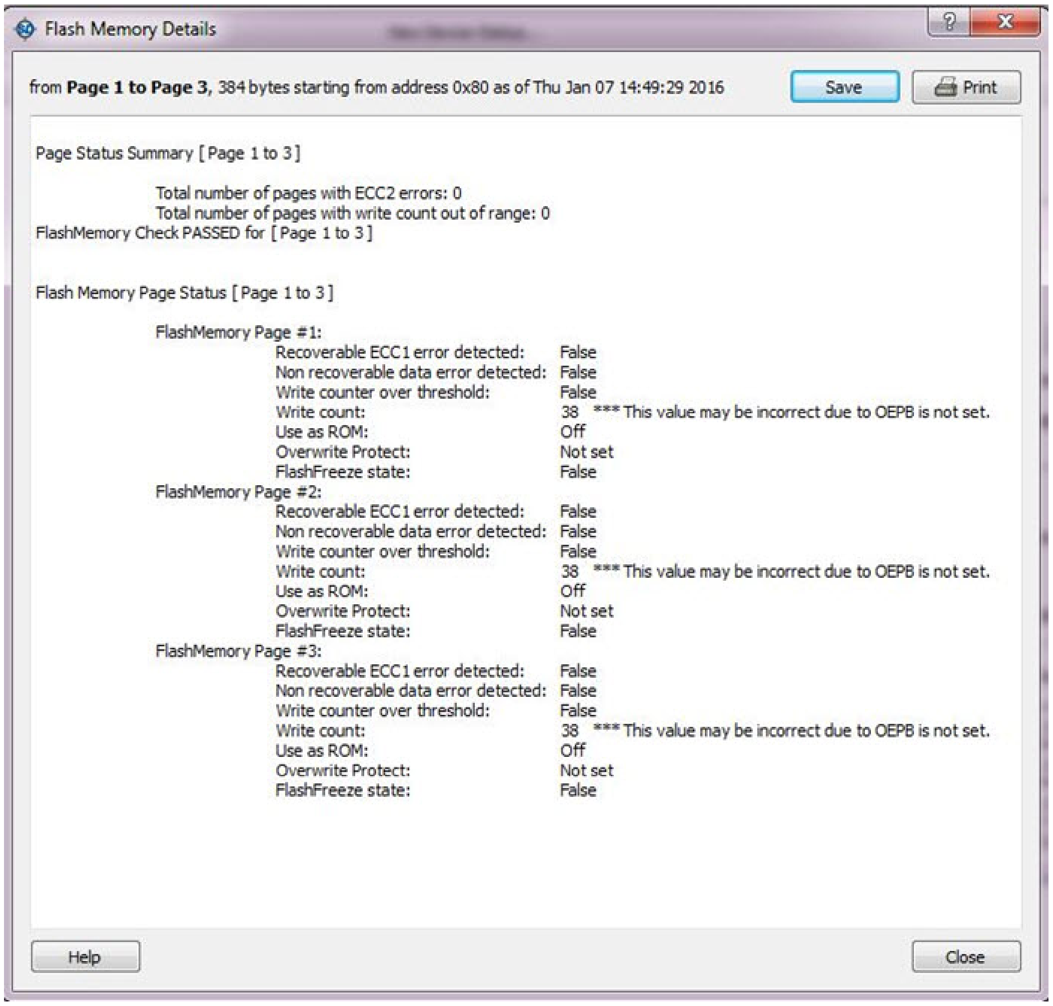
The page status gets updated when you:

- 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.



from **Page 1 to Page 3**, 384 bytes starting from address 0x80 as of Thu Jan 07 14:49:29 2016

```
Page Status Summary [ Page 1 to 3 ]

                    Total number of pages with ECC2 errors: 0
                    Total number of pages with write count out of range: 0
FlashMemory Check PASSED for [ Page 1 to 3 ]


Flash Memory Page Status [ Page 1 to 3 ]

                    FlashMemory Page #1:
                                        Recoverable ECC1 error detected:     False
                                        Non recoverable data error detected: False
                                        Write counter over threshold:        False
                                        Write count:                         38   *** This value may be incorrect due to OEPB is not set.
                                        Use as ROM:                          Off
                                        Overwrite Protect:                   Not set
                                        FlashFreeze state:                   False
                    FlashMemory Page #2:
                                        Recoverable ECC1 error detected:     False
                                        Non recoverable data error detected: False
                                        Write counter over threshold:        False
                                        Write count:                         38   *** This value may be incorrect due to OEPB is not set.
                                        Use as ROM:                          Off
                                        Overwrite Protect:                   Not set
                                        FlashFreeze state:                   False
                    FlashMemory Page #3:
                                        Recoverable ECC1 error detected:     False
                                        Non recoverable data error detected: False
                                        Write counter over threshold:        False
                                        Write count:                         38   *** This value may be incorrect due to OEPB is not set.
                                        Use as ROM:                          Off
                                        Overwrite Protect:                   Not set
                                        FlashFreeze state:                   False
```

Figure 6 · 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

It also includes the FPGA Hardware Breakpoint (FHB) controls, consisting of the following tabs:

- "Event Counter" **Error! Bookmark not defined.**
- "Frequency Monitor" **Error! Bookmark not defined.**
- "User Clock Frequencies" **Error! Bookmark not defined.**

Hierarchical View

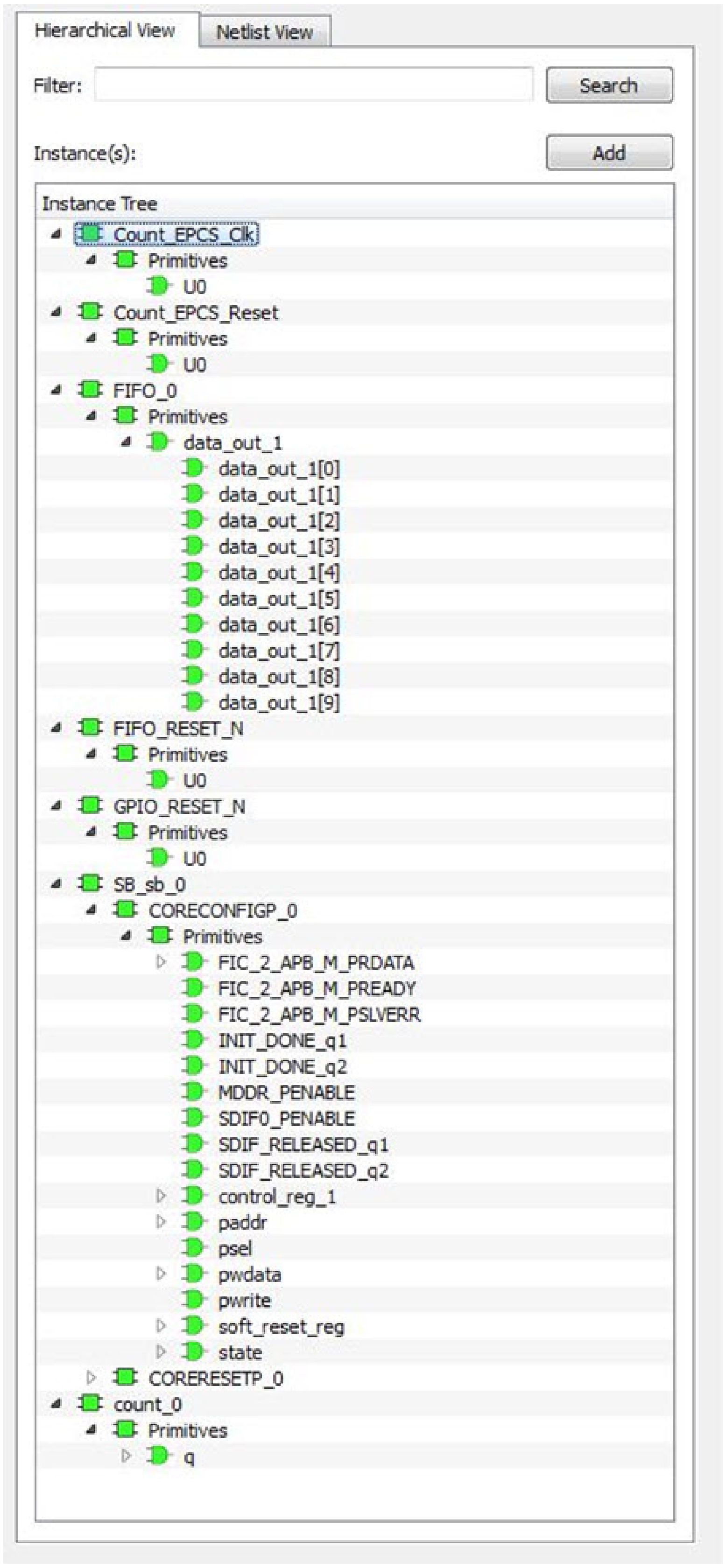
The Hierarchical View lets you view the instance level hierarchy of the design programmed on the device and select the signals to add to the Live Probes, Active Probes, and Probe Insertion tabs in the Debug FPGA Array dialog box. Logical and physical Memory Blocks can also be selected.

- **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.



Hierarchical View | Netlist View

Filter: Search

Instance(s): Add

```
Instance Tree
◢ Count_EPCS_Clk
    ◢ Primitives
        U0
◢ Count_EPCS_Reset
    ◢ Primitives
        U0
◢ FIFO_0
    ◢ Primitives
        ◢ data_out_1
            data_out_1[0]
            data_out_1[1]
            data_out_1[2]
            data_out_1[3]
            data_out_1[4]
            data_out_1[5]
            data_out_1[6]
            data_out_1[7]
            data_out_1[8]
            data_out_1[9]
◢ FIFO_RESET_N
    ◢ Primitives
        U0
◢ GPIO_RESET_N
    ◢ Primitives
        U0
◢ SB_sb_0
    ◢ CORECONFIGP_0
        ◢ Primitives
            ▷ FIC_2_APB_M_PRDATA
              FIC_2_APB_M_PREADY
              FIC_2_APB_M_PSLVERR
              INIT_DONE_q1
              INIT_DONE_q2
              MDDR_PENABLE
              SDIF0_PENABLE
              SDIF_RELEASED_q1
              SDIF_RELEASED_q2
            ▷ control_reg_1
            ▷ paddr
              psel
            ▷ pwdata
              pwrite
            ▷ soft_reset_reg
            ▷ state
    ▷ CORERESETP_0
◢ count_0
    ◢ Primitives
        ▷ q
```

Figure 7 · Hierarchical View

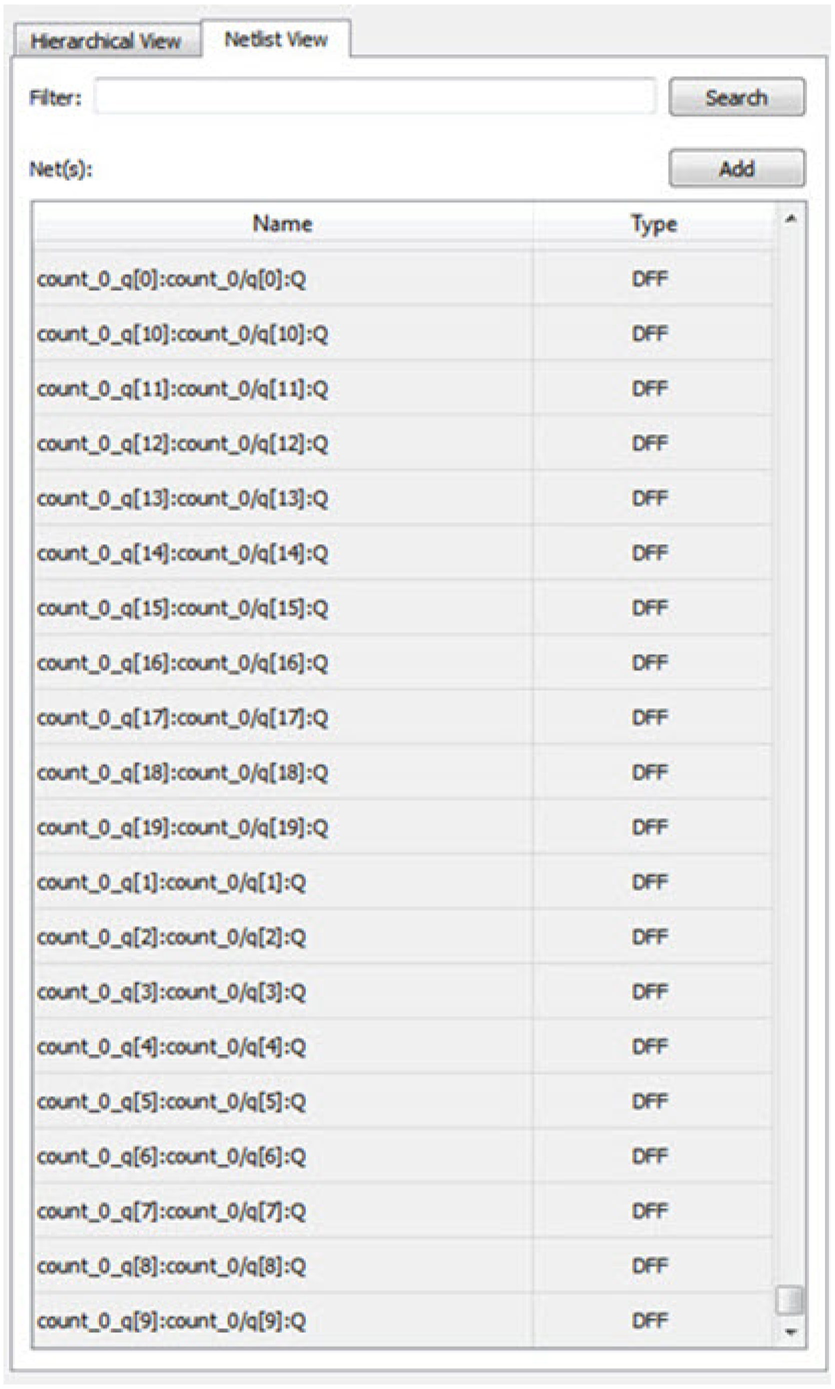
Search

In Live Probes, Active Probes, Memory Blocks, 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.



Name	Type
count_0_q[0]:count_0/q[0]:Q	DFF
count_0_q[10]:count_0/q[10]:Q	DFF
count_0_q[11]:count_0/q[11]:Q	DFF
count_0_q[12]:count_0/q[12]:Q	DFF
count_0_q[13]:count_0/q[13]:Q	DFF
count_0_q[14]:count_0/q[14]:Q	DFF
count_0_q[15]:count_0/q[15]:Q	DFF
count_0_q[16]:count_0/q[16]:Q	DFF
count_0_q[17]:count_0/q[17]:Q	DFF
count_0_q[18]:count_0/q[18]:Q	DFF
count_0_q[19]:count_0/q[19]:Q	DFF
count_0_q[1]:count_0/q[1]:Q	DFF
count_0_q[2]:count_0/q[2]:Q	DFF
count_0_q[3]:count_0/q[3]:Q	DFF
count_0_q[4]:count_0/q[4]:Q	DFF
count_0_q[5]:count_0/q[5]:Q	DFF
count_0_q[6]:count_0/q[6]:Q	DFF
count_0_q[7]:count_0/q[7]:Q	DFF
count_0_q[8]:count_0/q[8]:Q	DFF
count_0_q[9]:count_0/q[9]:Q	DFF

Figure 8 · 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)

Live Probe is a design debug option that uses non-intrusive real time scoping of up to two probe points with no design changes.

The Live Probes tab in the Debug FPGA Array dialog box displays a table with the probe names and pin types.

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).

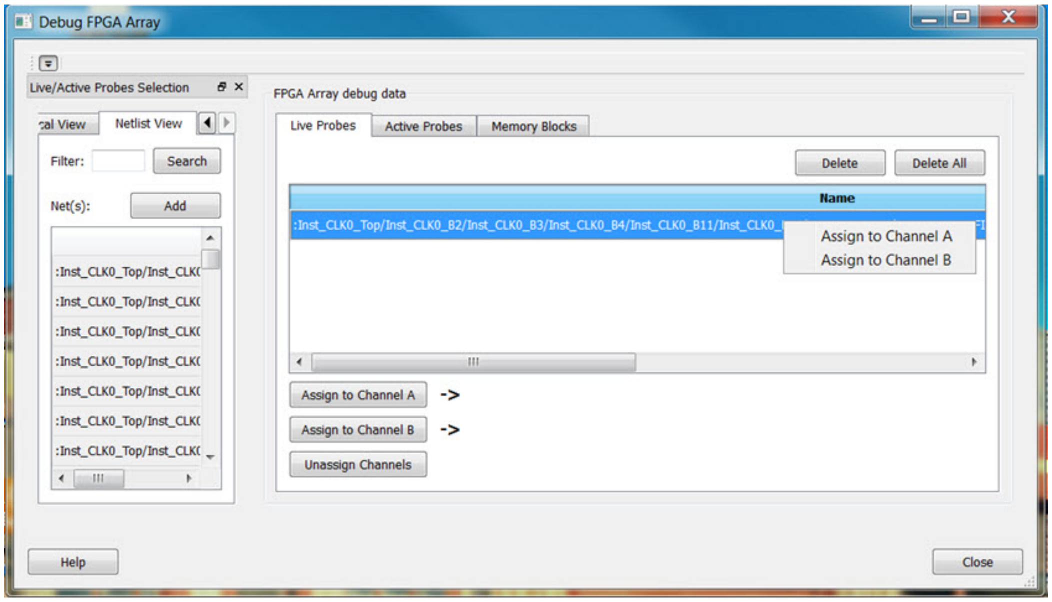


Figure 9 · Live Probes Tab (SmartFusion2 and IGLOO2) in SmartDebug FPGA Array Dialog Box

RTG4

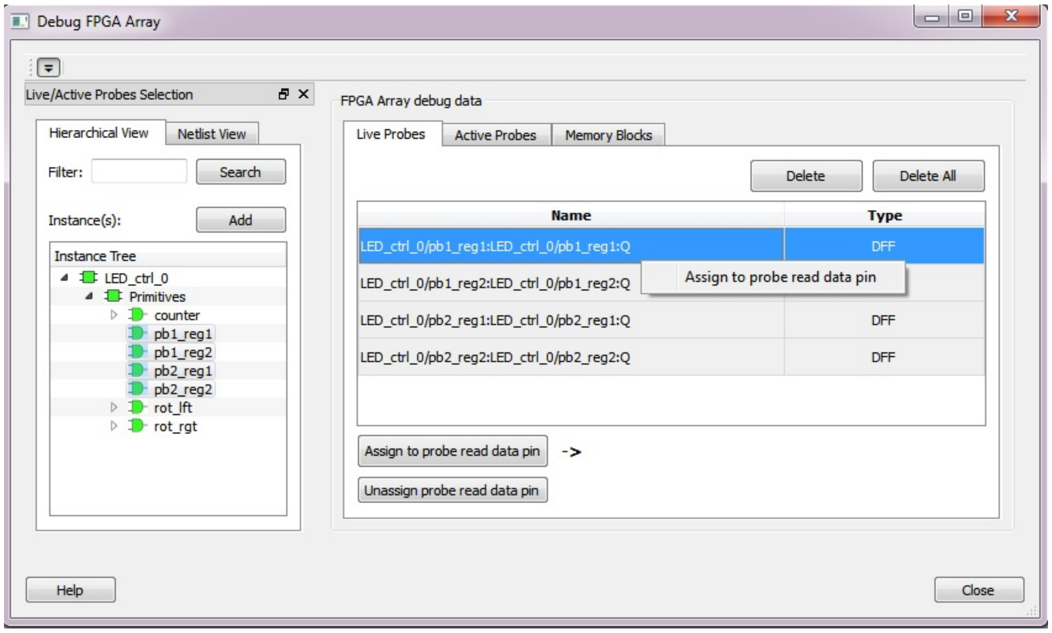
One probe channel (Probe Read Data Pin) is available for RTG4 for debug. When a probe name is selected, it can be assigned to the Probe Channel (Probe Read Data Pin).

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).



Name	Type
LED_ctrl_0/pb1_reg1:LED_ctrl_0/pb1_reg1:Q	DFF
LED_ctrl_0/pb1_reg2:LED_ctrl_0/pb1_reg2:Q	
LED_ctrl_0/pb2_reg1:LED_ctrl_0/pb2_reg1:Q	DFF
LED_ctrl_0/pb2_reg2:LED_ctrl_0/pb2_reg2:Q	DFF

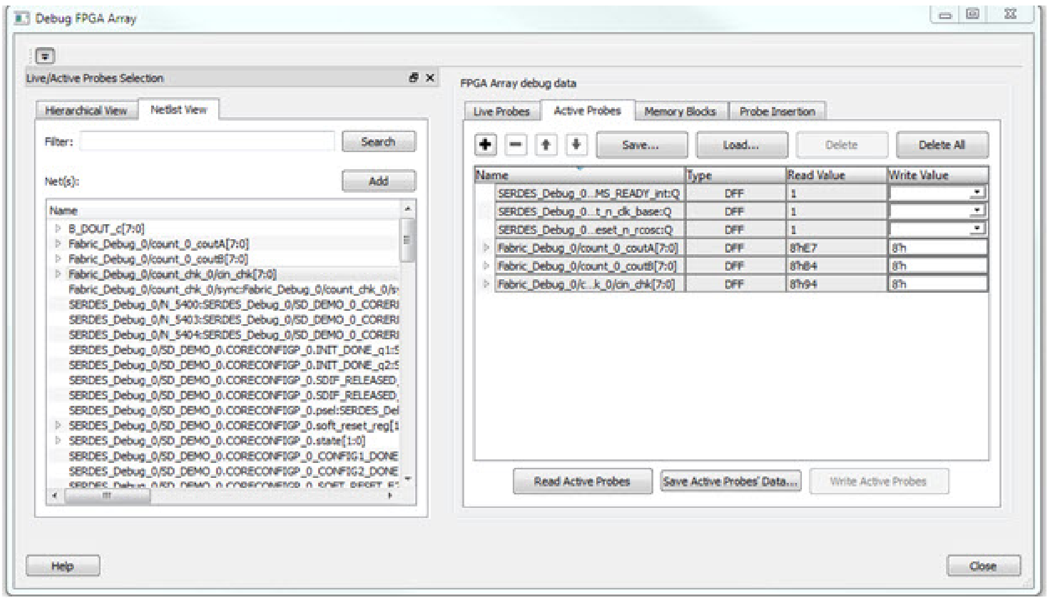
Figure 10 · Live Probes Tab (RTG4) in SmartDebug FPGA Array Dialog Box

Active Probes (SmartFusion2, IGLOO2, and RTG4)

Active Probe is a design debug option to read and write to one or many probe points in the design through JTAG.

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.



Debug FPGA Array

Live/Active Probes Selection

Hierarchical View | Netlist View

Filter: Search

Net(s): Add

Name
B_DOUT_c[7:0]
Fabric_Debug_0/count_0_coutA[7:0]
Fabric_Debug_0/count_0_coutB[7:0]
Fabric_Debug_0/count_chk_0/cin_chk[7:0]
Fabric_Debug_0/count_chk_0/sync:Fabric_Debug_0/count_chk_0/s
SERDES_Debug_0/N_5400:SERDES_Debug_0/SD_DEMO_0_CORERI
SERDES_Debug_0/N_5403:SERDES_Debug_0/SD_DEMO_0_CORERI
SERDES_Debug_0/N_5404:SERDES_Debug_0/SD_DEMO_0_CORERI
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.INIT_DONE_q1:S
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.INIT_DONE_q2:S
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.SDIF_RELEASED_
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.SDIF_RELEASED_
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.psel:SERDES_Del
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.soft_reset_reg[1
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0.state[1:0]
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0_CONFIG1_DONE
SERDES_Debug_0/SD_DEMO_0.CORECONFIGP_0_CONFIG2_DONE

Help

FPGA Array debug data

Live Probes | Active Probes | Memory Blocks | Probe Insertion

Save... Load... Delete Delete All

Name	Type	Read Value	Write Value
SERDES_Debug_0...MS_READY_int:Q	DFF	1	
SERDES_Debug_0...t_n_clk_base:Q	DFF	1	
SERDES_Debug_0...eset_n_rcosc:Q	DFF	1	
Fabric_Debug_0/count_0_coutA[7:0]	DFF	8'hE7	8'h
Fabric_Debug_0/count_0_coutB[7:0]	DFF	8'hB4	8'h
Fabric_Debug_0/c...k_0/cin_chk[7:0]	DFF	8'h94	8'h

Read Active Probes | Save Active Probes' Data... | Write Active Probes

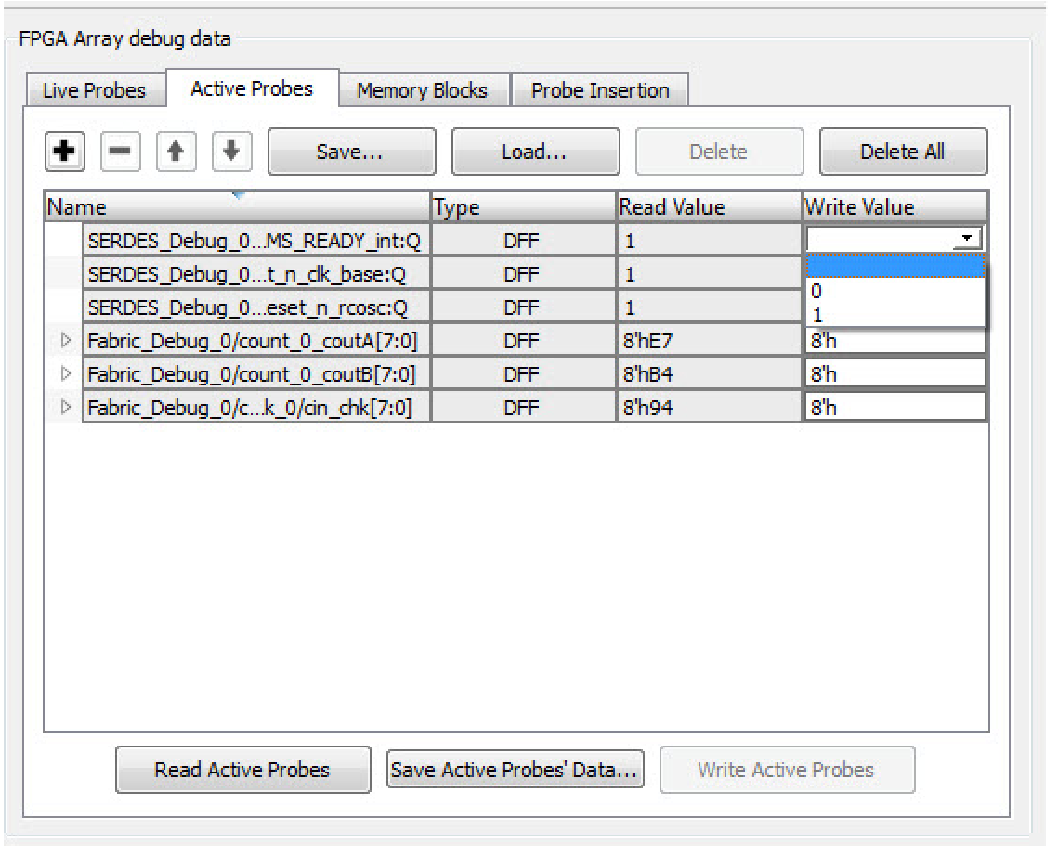
Close

Figure 11 · 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



FPGA Array debug data

Live Probes | Active Probes | Memory Blocks | Probe Insertion

Save... Load... Delete Delete All

Name	Type	Read Value	Write Value
SERDES_Debug_0...MS_READY_int:Q	DFF	1	0 / 1
SERDES_Debug_0...t_n_clk_base:Q	DFF	1	
SERDES_Debug_0...eset_n_rcosc:Q	DFF	1	
Fabric_Debug_0/count_0_coutA[7:0]	DFF	8'hE7	8'h
Fabric_Debug_0/count_0_coutB[7:0]	DFF	8'hB4	8'h
Fabric_Debug_0/c...k_0/cin_chk[7:0]	DFF	8'h94	8'h

Read Active Probes | Save Active Probes' Data... | Write Active Probes

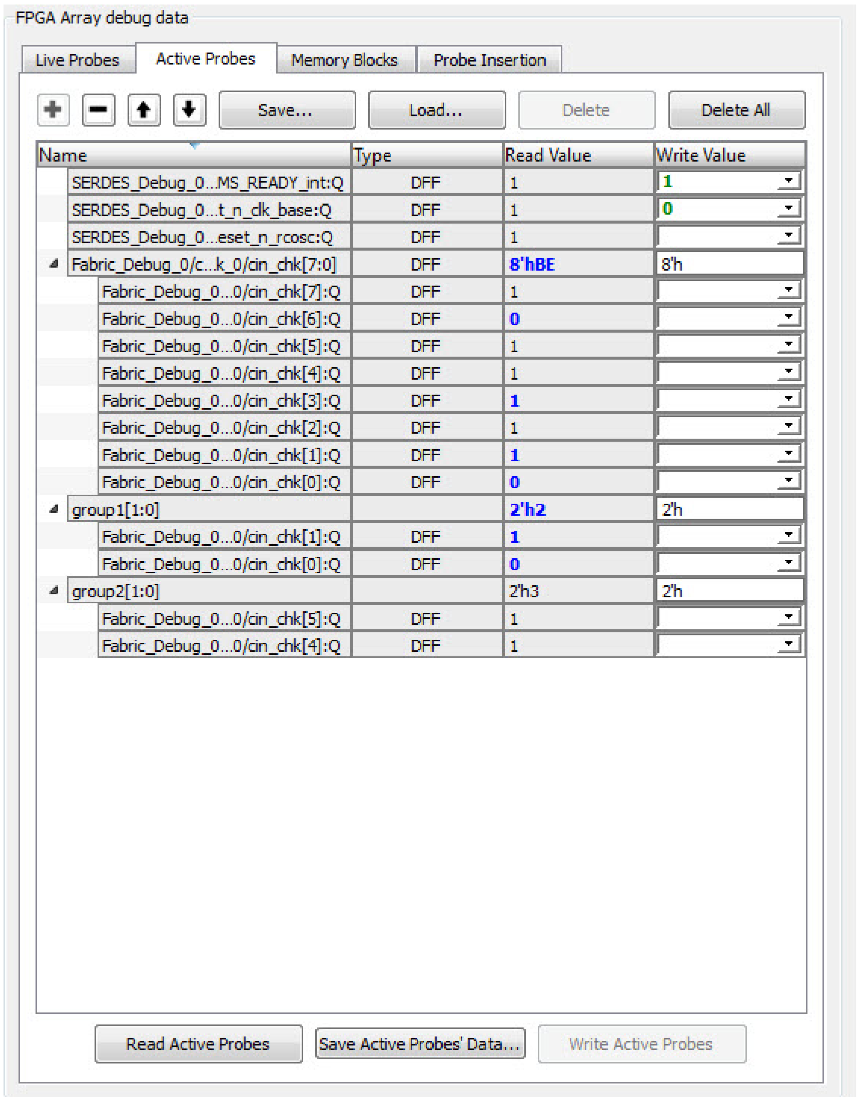
Figure 12 · Active Probes Tab - Write Value Column Options

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

Name	Type	Read Value	Write Value
SERDES_Debug_0...MS_READY_int:Q	DFF	1	1
SERDES_Debug_0...t_n_clk_base:Q	DFF	1	0
SERDES_Debug_0...eset_n_rcosc:Q	DFF	1	
Fabric_Debug_0/c...k_0/cin_chk[7:0]	DFF	8'hBE	8'h
Fabric_Debug_0...0/cin_chk[7]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[6]:Q	DFF	0	
Fabric_Debug_0...0/cin_chk[5]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[4]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[3]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[2]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[1]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[0]:Q	DFF	0	
group1[1:0]		2'h2	2'h
Fabric_Debug_0...0/cin_chk[1]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[0]:Q	DFF	0	
group2[1:0]		2'h3	2'h
Fabric_Debug_0...0/cin_chk[5]:Q	DFF	1	
Fabric_Debug_0...0/cin_chk[4]:Q	DFF	1	

Figure 13 · Active Probes Tab

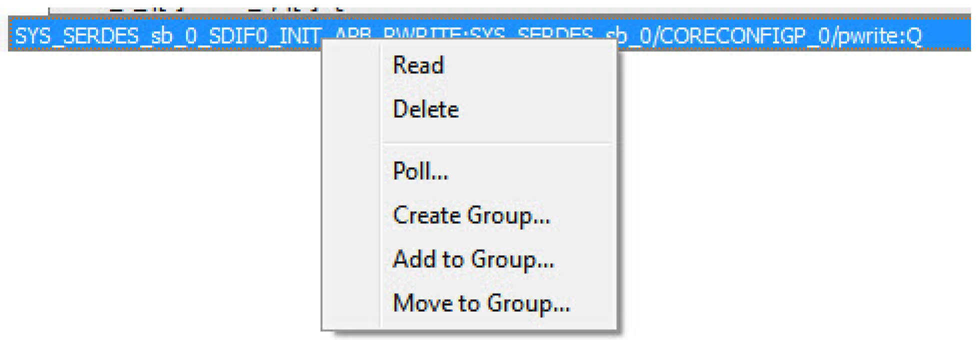
- Green entries in the “Write Value” column indicate that the operation was successful.
- Blue entries in the “Read Value” column show indicate values that have changed since the last read.

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:

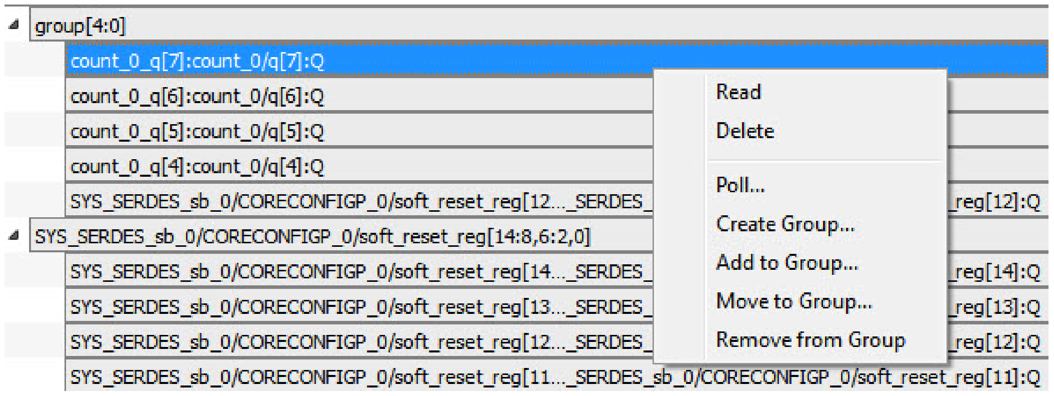
For individual signals that are not part of a probe group or bus:

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


SYS_SERDES_sb_0_SDIF0_INIT_APB_PWRITE:SYS_SERDES_sb_0/CORECONFIGP_0/pwrite:Q

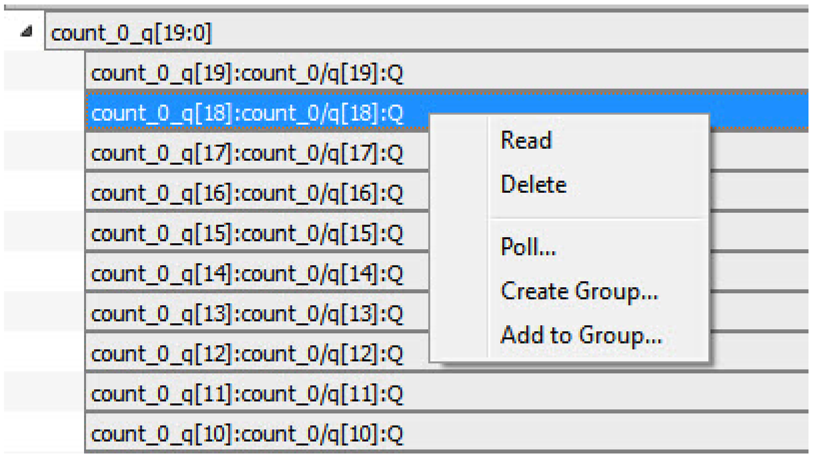
For individual signals in a probe group:

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


group[4:0]
count_0_q[7]:count_0/q[7]:Q
count_0_q[6]:count_0/q[6]:Q
count_0_q[5]:count_0/q[5]:Q
count_0_q[4]:count_0/q[4]:Q
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[12..._SERDES_ reg[12]:Q
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[14:8,6:2,0]
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[14..._SERDES_ reg[14]:Q
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[13..._SERDES_ reg[13]:Q
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[12..._SERDES_ reg[12]:Q
SYS_SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[11..._SERDES_sb_0/CORECONFIGP_0/soft_reset_reg[11]:Q

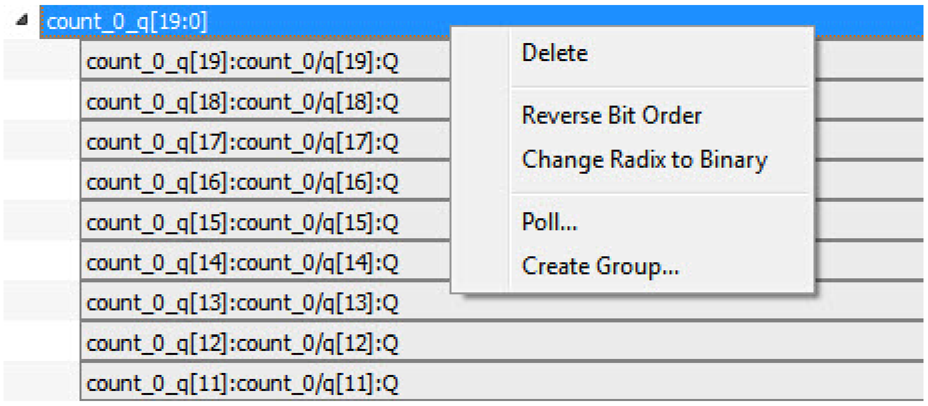
For individual signals in a bus:

- Read
- Write
- Create Group
- Add to Group


count_0_q[19:0]
count_0_q[19]:count_0/q[19]:Q
count_0_q[18]:count_0/q[18]:Q
count_0_q[17]:count_0/q[17]:Q
count_0_q[16]:count_0/q[16]:Q
count_0_q[15]:count_0/q[15]:Q
count_0_q[14]:count_0/q[14]:Q
count_0_q[13]:count_0/q[13]:Q
count_0_q[12]:count_0/q[12]:Q
count_0_q[11]:count_0/q[11]:Q
count_0_q[10]:count_0/q[10]:Q

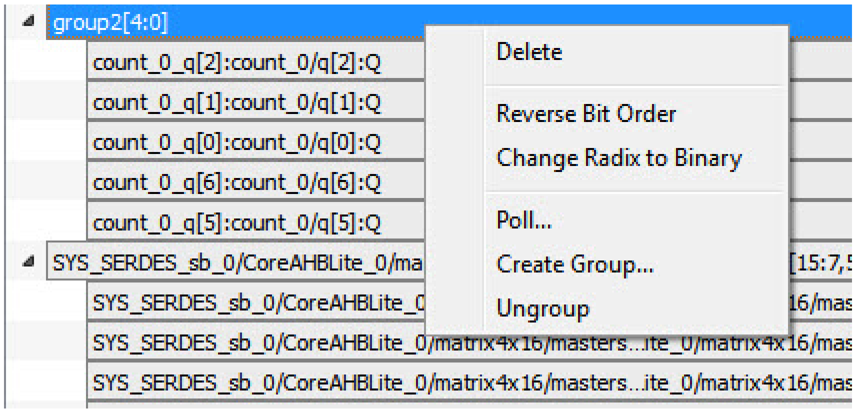
For a bus:

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


count_0_q[19:0]
count_0_q[19]:count_0/q[19]:Q
count_0_q[18]:count_0/q[18]:Q
count_0_q[17]:count_0/q[17]:Q
count_0_q[16]:count_0/q[16]:Q
count_0_q[15]:count_0/q[15]:Q
count_0_q[14]:count_0/q[14]:Q
count_0_q[13]:count_0/q[13]:Q
count_0_q[12]:count_0/q[12]:Q
count_0_q[11]:count_0/q[11]:Q
Delete
Reverse Bit Order
Change Radix to Binary
Poll...
Create Group...

For a probe group:

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


group2[4:0]
count_0_q[2]:count_0/q[2]:Q
count_0_q[1]:count_0/q[1]:Q
count_0_q[0]:count_0/q[0]:Q
count_0_q[6]:count_0/q[6]:Q
count_0_q[5]:count_0/q[5]:Q
SYS_SERDES_sb_0/CoreAHBLite_0/ma
SYS_SERDES_sb_0/CoreAHBLite_0/matrix4x16/masters...ite_0/matrix4x16/mas
Delete
Reverse Bit Order
Change Radix to Binary
Poll...
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 Memory Blocks tab in the Debug FPGA Array dialog box shows the hierarchical view of all memory blocks in the design. The depth and width of blocks shown in the logical view are determined by the user in SmartDesign, RTL, or IP cores using memory blocks.

The example figure that follows shows the hierarchical view of the Memory Blocks tab. You can view logical blocks and physical blocks. Logical blocks are shown with an **L** (L), and physical blocks are shown with a **P** (P).

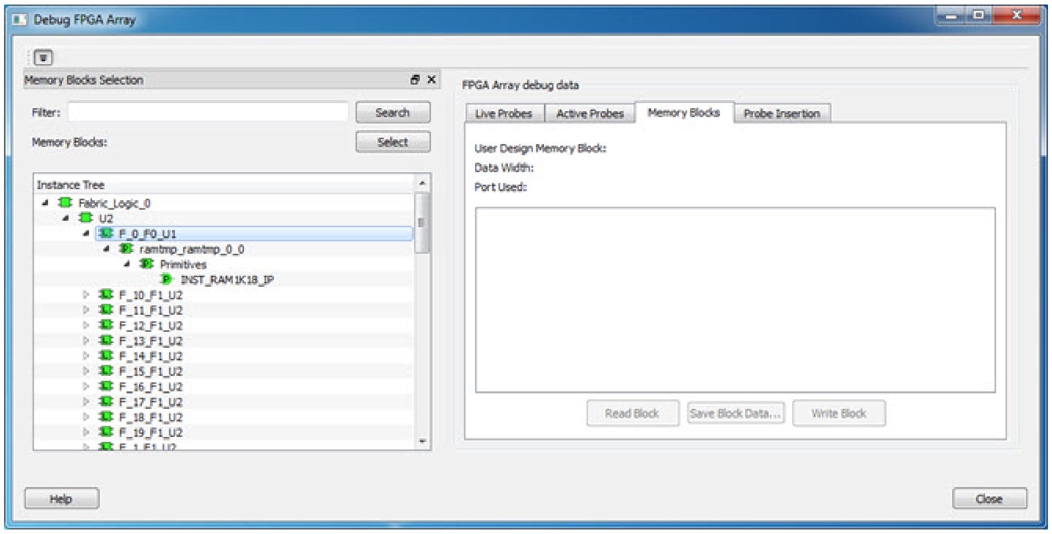


Figure 14 · Memory Blocks Tab - Hierarchical View

You can only select one block at a time. You can select and add blocks in the following ways:

- Right-click the name of a memory block and click **Add** as shown in the following figure.

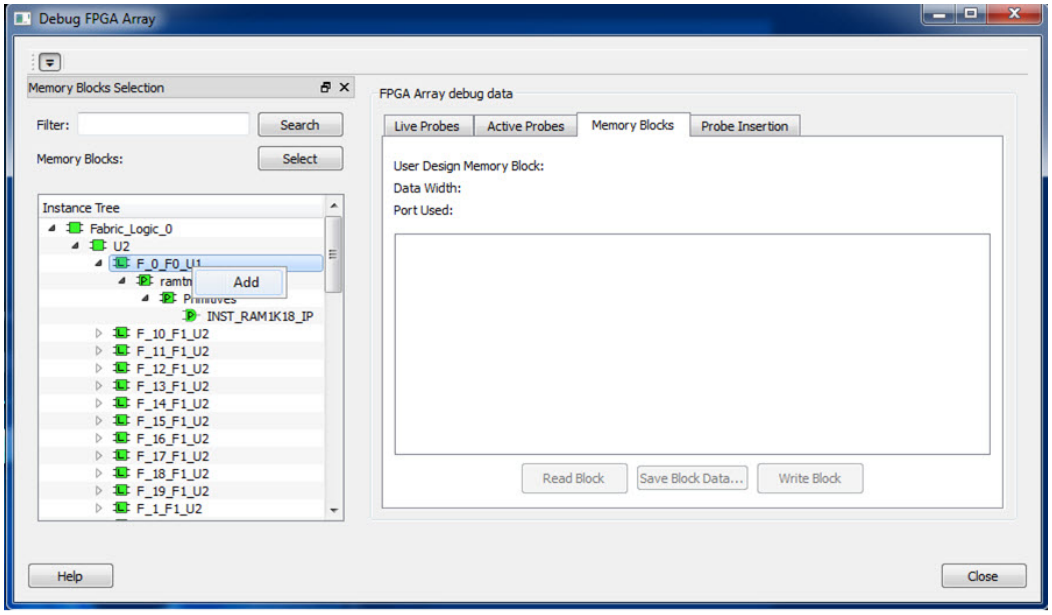


Figure 15 · Add Memory Block

- Click on a name in the list and then click **Select** .
- Select a name, drag it to the right, and drop it into the Memory Blocks tab.
- Enter a memory block name in the Filter box and click **Search** or press **Enter** . Wildcard search is supported.

Note: Only memory blocks with an **L** or **P** icon can be selected in the hierarchical view.

Memory Block Fields

The following memory block fields appear in the Memory Blocks tab.

User Design Memory Block

The selected block name appears on the right side. If the block selected is logical, the name from top of the block is shown.

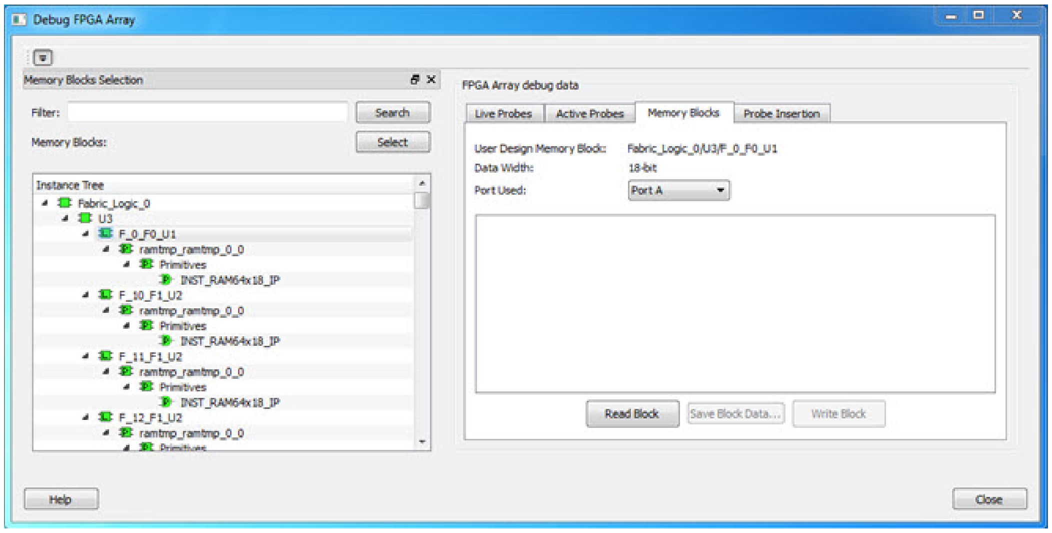
Data Width

If a block is logical, the width from each physical block is retrieved, consolidated, and displayed. If the block is physical, the width is 9-bits, and the depth is 128 for uSRAM blocks and 2048 for LSRAM blocks.

Port Used

This field is displayed only in the logical block view. Because configurators can have asymmetric ports, memory location can have different widths. The port shown can either be Port A or Port B. For TPSRAM, where both ports are used for reading, Port A is used. This field is hidden for physical blocks, as the values shown will be irrespective of read ports.

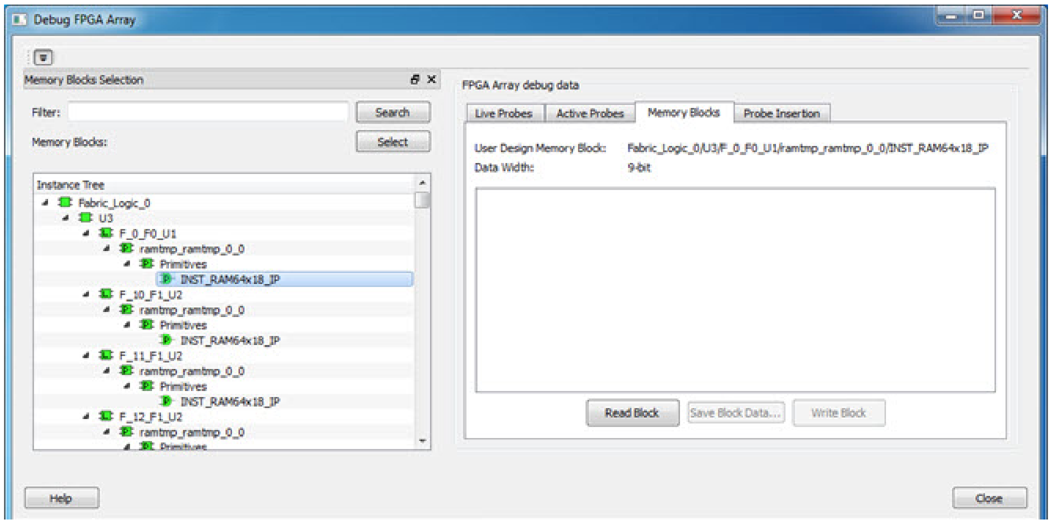
The following figure shows the Memory Blocks tab fields for a logical block view.



User Design Memory Block: Fabric_Logic_0/U3/F_0_F0_U1
Data Width: 18-bit
Port Used: Port A

Figure 16 · Memory Blocks Tab Fields for Logical Block View

The following figure shows the Memory Blocks tab fields for a physical block view.



User Design Memory Block: Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x18_IP
Data Width: 9-bit

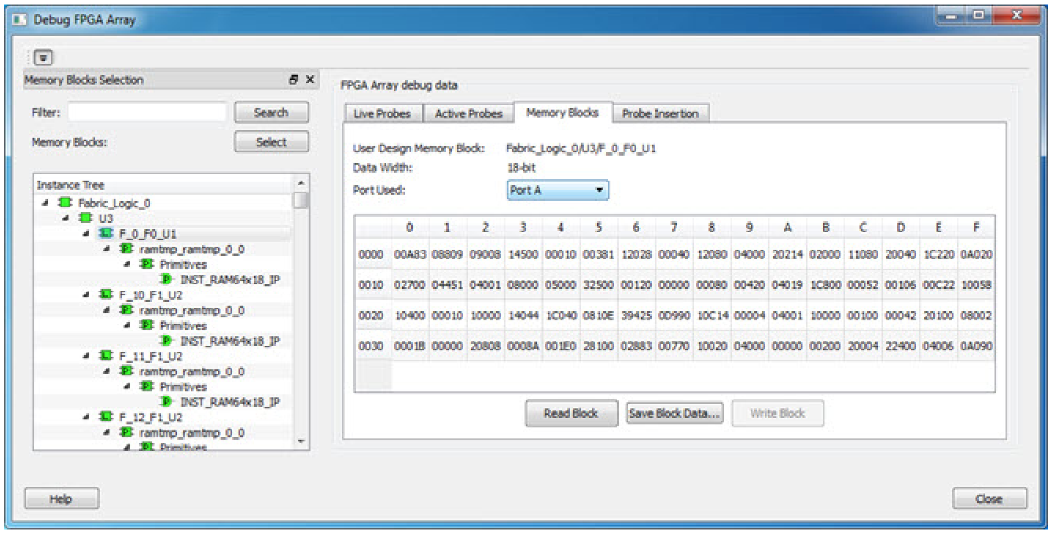
Figure 17 · Memory Blocks Tab Fields for Physical Block View

Read Block

Memory blocks can be read once they are selected. If the block name appears on the right-hand side, the Read Block button is enabled. Click **Read Block** to read the memory block.

Logical Block Read

A logical block shows three fields. User Design Memory Block and Data Width are read only fields, and Port Used has options. If the design uses both ports, Port A and Port B are shown under options. If only one port is used, only that port is shown.



User Design Memory Block: Fabric_Logic_0/U3/F_0_F0_U1
Data Width: 18-bit
Port Used: Port A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	00A83	08809	09008	14500	00010	00381	12028	00040	12080	04000	20214	02000	11080	20040	1C220	0A020
0010	02700	04451	04001	08000	05000	32500	00120	00000	00080	00420	04019	1C800	00052	00106	00C22	10058
0020	10400	00010	10000	14044	1C040	0810E	39425	0D990	10C14	00004	04001	10000	00100	00042	20100	08002
0030	0001B	00000	20808	0008A	001E0	28100	02883	00770	10020	04000	00000	00200	20004	22400	04006	0A090

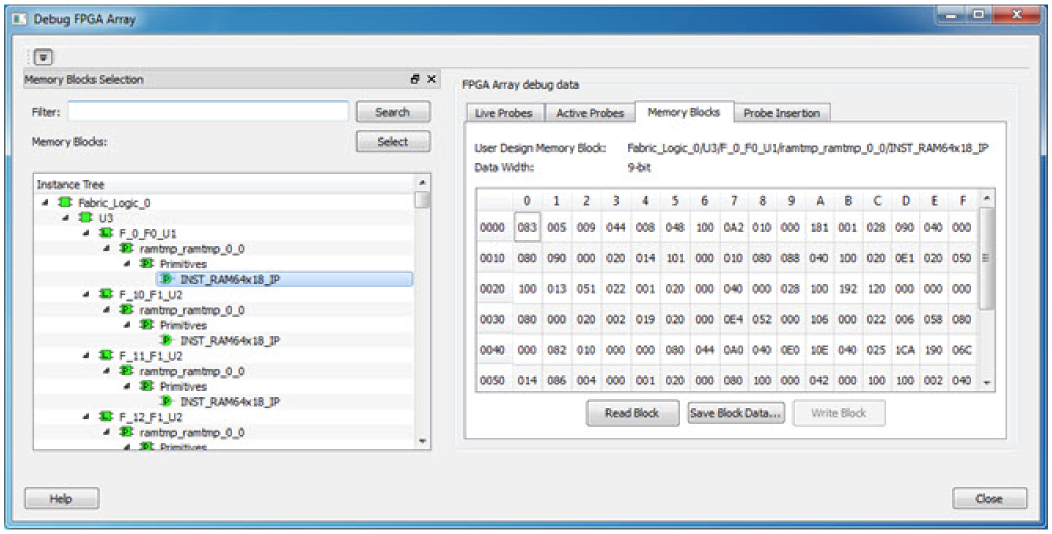
Figure 18 · Logical Block Read

The data shown is in Hexadecimal format. In the example figure above, data width is 18. Since each hexadecimal character has 4 bits of information, we can see 5 characters corresponding to 18 bits. Each row has 16 locations (shown in the column headers) which are numbered in hexadecimal from 0 to F.

Note: For all logical blocks that cannot be inferred from physical blocks, the corresponding icon does not contain a letter.

Physical Block Read

When a Physical block is selected, only the User Design Memory Block and Data Width fields are shown.



User Design Memory Block: Fabric_Logic_0/U3/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM64x18_IP
Data Width: 9-bit

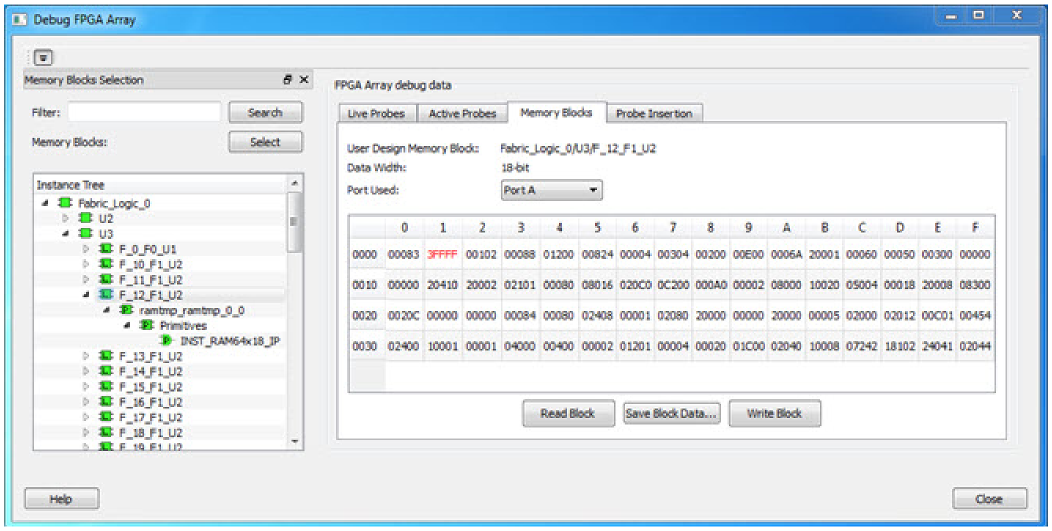
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	083	005	009	044	008	048	100	0A2	010	000	181	001	028	090	040	000
0010	080	090	000	020	014	101	000	010	080	088	040	100	020	0E1	020	050
0020	100	013	051	022	001	020	000	040	000	028	100	192	120	000	000	000
0030	080	000	020	002	019	020	000	0E4	052	000	106	000	022	006	058	080
0040	000	082	010	000	000	080	044	0A0	040	0E0	10E	040	025	1CA	190	06C
0050	014	086	004	000	001	020	000	080	100	000	042	000	100	100	002	040

Figure 19 · Physical Block Read

Write Block

Logical Block write

Memory block write can be done on each location individually. Logical block has each location of width that is displayed. Written format is hexadecimal numbers from 0 to F. Width is shown in bits, and values are shown in hexadecimal format. If an entered value exceeds the limit, SmartDebug displays a popup message showing the range of values that can be entered.



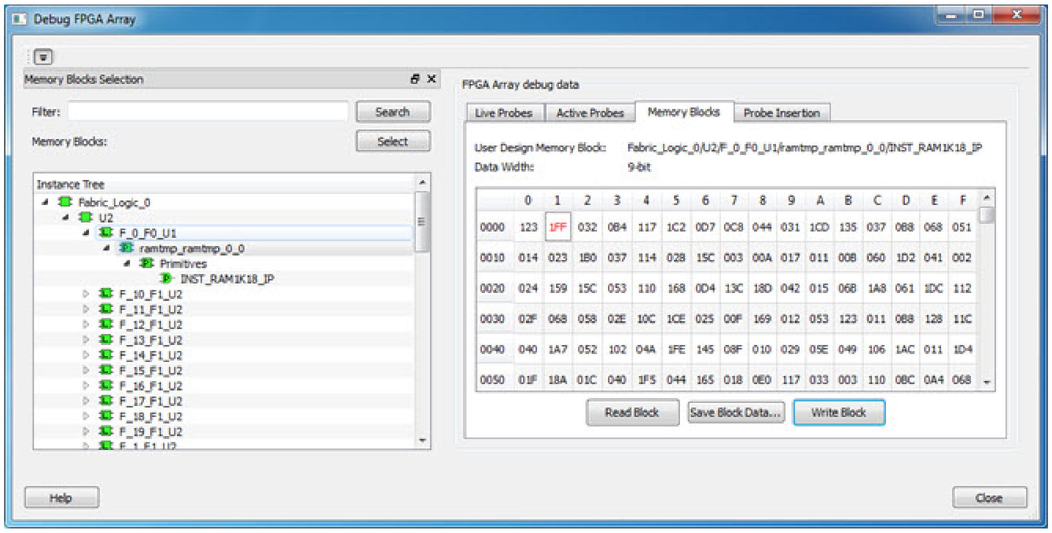
User Design Memory Block: Fabric_Logic_0/U3/F_12_F1_U2
Data Width: 18-bit
Port Used: Port A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	00083	3FFFF	00102	00088	01200	00824	00004	00304	00200	00E00	0006A	20001	00060	00050	00300	00000
0010	00000	20410	20002	02101	00080	08016	020C0	0C200	000A0	00002	08000	10020	05004	00018	20008	08300
0020	0020C	00000	00000	00084	00080	02408	00001	02080	20000	00000	20000	00005	02000	02012	00C01	00454
0030	02400	10001	00001	04000	00400	00002	01201	00004	00020	01C00	02040	10008	07242	18102	24041	02044

Figure 20 · Logical Block Write

Physical Block Write

Physical blocks have a fixed width of 9 bits. The maximum value that can be written in hexadecimal format is 1FF. If an entered value exceeds the limit, SmartDebug displays a popup message showing the range of values that can be entered.



User Design Memory Block: Fabric_Logic_0/U2/F_0_F0_U1/ramtmp_ramtmp_0_0/INST_RAM1K18_IP

Data Width: 9-bit

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	123	1FF	032	0B4	117	1C2	0D7	0C8	044	031	1CD	135	037	088	068	051
0010	014	023	1B0	037	114	028	15C	003	00A	017	011	008	060	1D2	041	002
0020	024	159	15C	053	110	168	0D4	13C	18D	042	015	068	1A8	061	1DC	112
0030	02F	068	058	02E	10C	1CE	025	00F	169	012	053	123	011	088	128	11C
0040	040	1A7	052	102	04A	1FE	145	08F	010	029	05E	049	106	1AC	011	1D4
0050	01F	18A	01C	040	1F5	044	165	018	0E0	117	033	003	110	0BC	0A4	068

Figure 21 · Physical Block Write

Unsupported Memory Blocks

If RTL is used to configure memory blocks, it is recommended that you follow RAM block inference guidelines provided by Microsemi. See Inferring Microsemi SmartFusion2 RAM Blocks for more information.

SmartDebug may or may not be able to support logical view for memory blocks that are inferred using RTL coding not specified in the above document.

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.

Note: This feature is not available in standalone mode because of the need to run incremental routing.

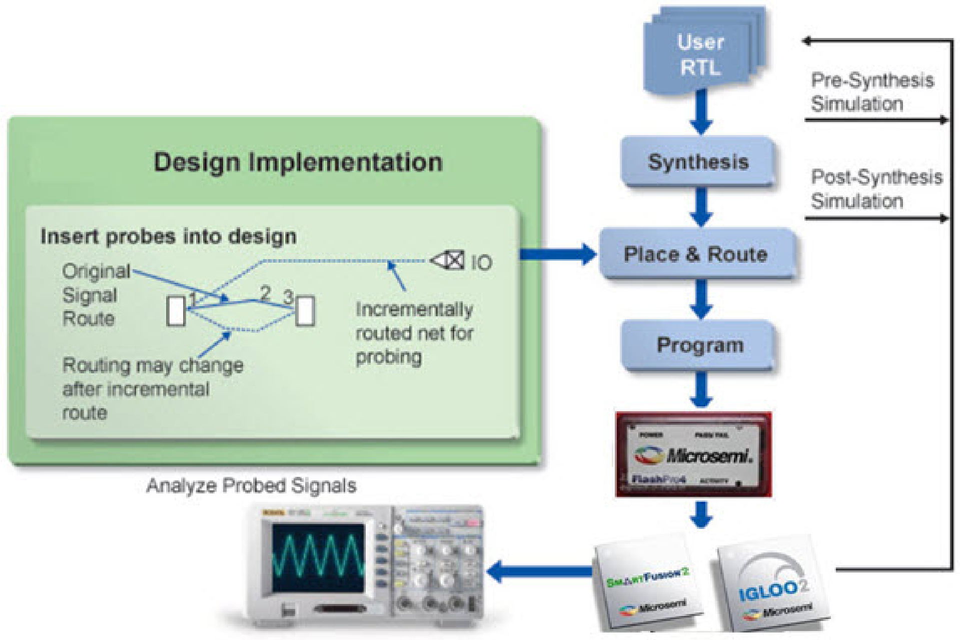


Figure 22 · Probe Insertion in the Design Process

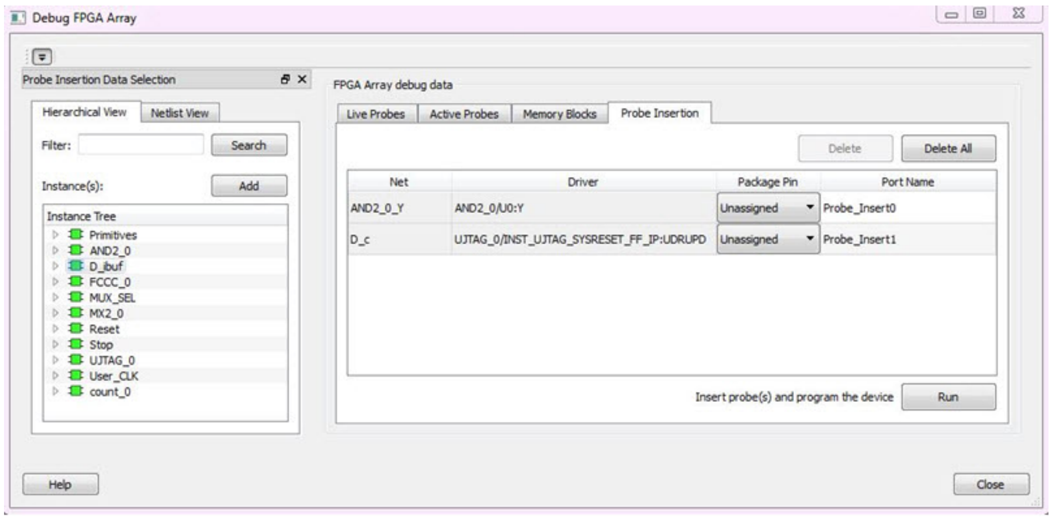
The Probe Insertion debug feature is complementary to Live Probes and Active Probes. Live Probes and Active Probes use a special dedicated probe circuitry.

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.



Net	Driver	Package Pin	Port Name
AND2_0_Y	AND2_0/U0:Y	Unassigned	Probe_Insert0
D_c	UJTAG_0/INST_UJTAG_SYSRESET_FF_IP:UDRUPD	Unassigned	Probe_Insert1

Figure 23 · Probe Insertion Tab

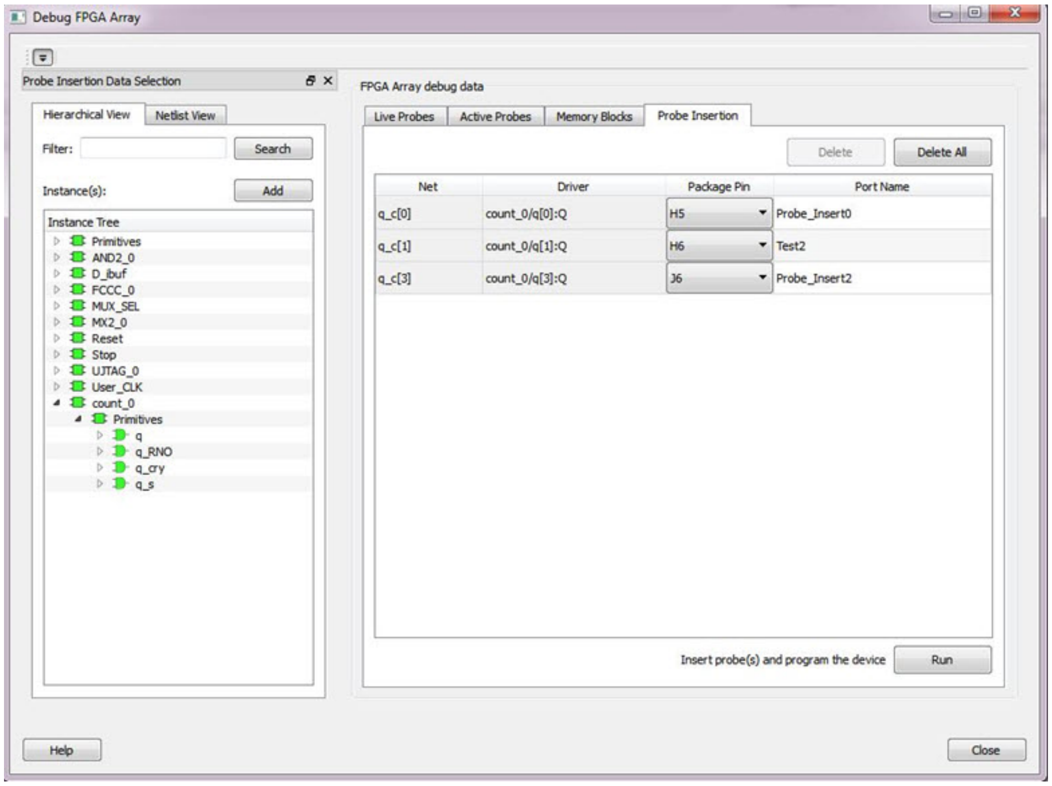
In the left pane of the Probe Insertion tab, all available Probe Points are listed in instance level hierarchy in the Hierarchical View. All Probe Names are shown with the Name and Type in the Netlist View.

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).



Net	Driver	Package Pin	Port Name
q_c[0]	count_0/q[0]:Q	H5	Probe_Insert0
q_c[1]	count_0/q[1]:Q	H6	Test2
q_c[3]	count_0/q[3]:Q	J6	Probe_Insert2

Figure 24 · 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.

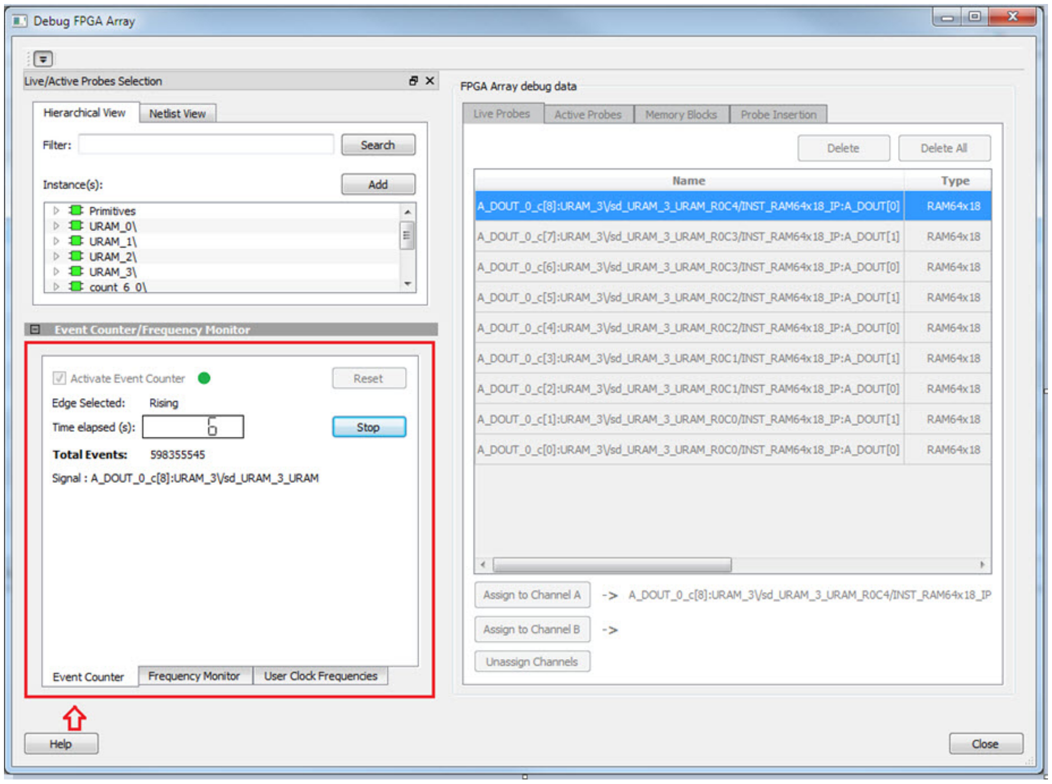
Reverting to the Original Design

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.

Event Counter

The Event Counter counts the signals that are assigned to Channel A through the Live Probe feature. This feature can track events from the MSS or the board. When the Event Counter is activated, and a signal is assigned to Channel A, the counter starts counting the rising edge transitions. The counter must be stopped to get the final signal transition count. During the count, you cannot assign another signal to Channel A/Channel B or go to any other tab on the window.



Activate Event Counter ●

Edge Selected: Rising

Time elapsed (s): 6

Total Events: 598355545

Signal : A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP

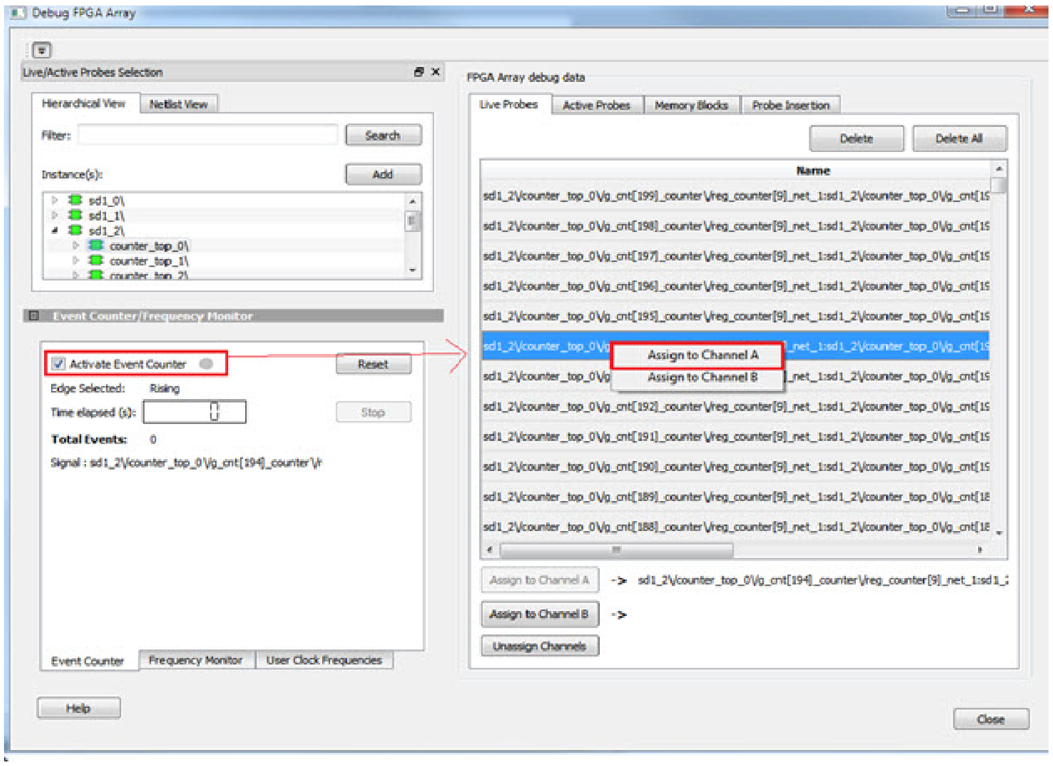
Assign to Channel B ->

Figure 25 · Event Counter Tab/UI

Activating the Event Counter

You can activate the Event Counter in either of the following two ways:

- Click **Activate Event Counter** and then assign a signal to Live Probe Channel A.



Filter:

Instance(s):
- sd1_0\
- sd1_1\
- sd1_2\
  - counter_top_0\
  - counter_top_1\
  - counter_top_2\

Activate Event Counter [✓]

Edge Selected: Rising

Time elapsed (s): 0

Total Events: 0

Signal : sd1_2\/counter_top_0\/g_cnt[194]_counter\/r

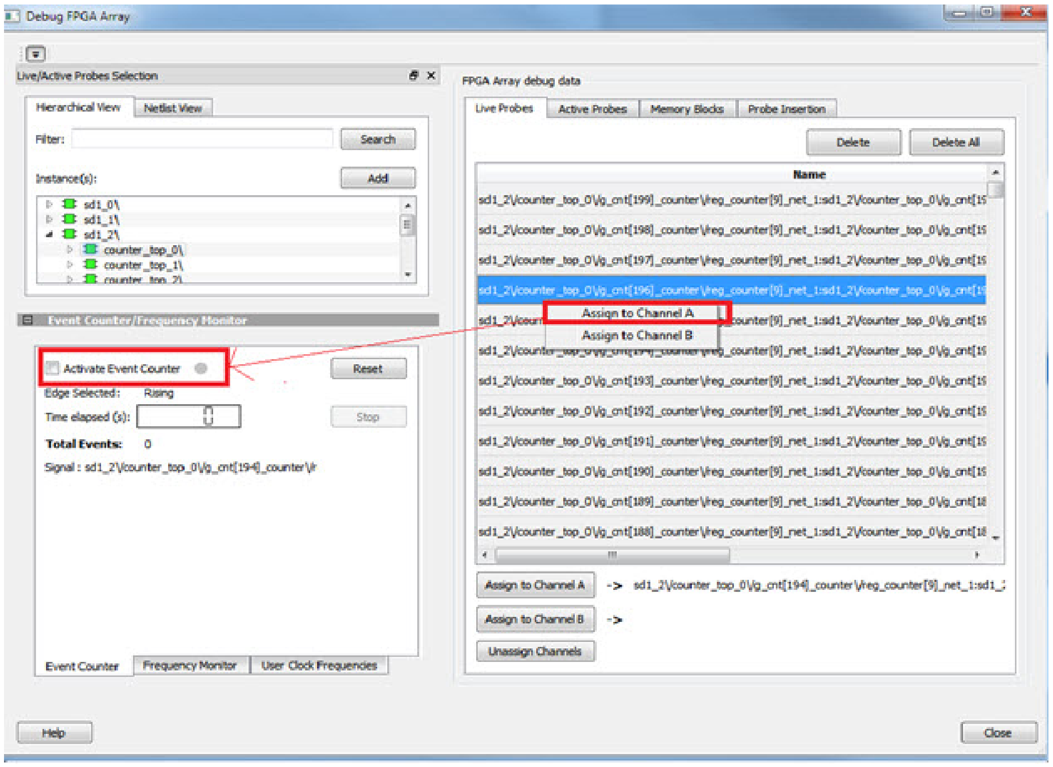
Name
sd1_2\/counter_top_0\/g_cnt[199]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[198]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[197]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[196]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[195]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g ... _net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g ... _net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[191]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[190]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[189]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18
sd1_2\/counter_top_0\/g_cnt[188]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18

Assign to Channel A -> sd1_2\/counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_2

Assign to Channel B ->

Figure 26 · Activating the Event Counter

- Assign a signal to Probe Channel A and then click **Activate Event Counter**.



Filter:

Instance(s):
- sd1_0\
- sd1_1\
- sd1_2\
  - counter_top_0\
  - counter_top_1\
  - counter_top_2\

Activate Event Counter [ ]

Edge Selected: Rising

Time elapsed (s): 0

Total Events: 0

Signal : sd1_2\/counter_top_0\/g_cnt[194]_counter\/r

Name
sd1_2\/counter_top_0\/g_cnt[199]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[198]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[197]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[196]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/coun ... counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter ... counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[193]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[191]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[190]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[189]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18
sd1_2\/counter_top_0\/g_cnt[188]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18

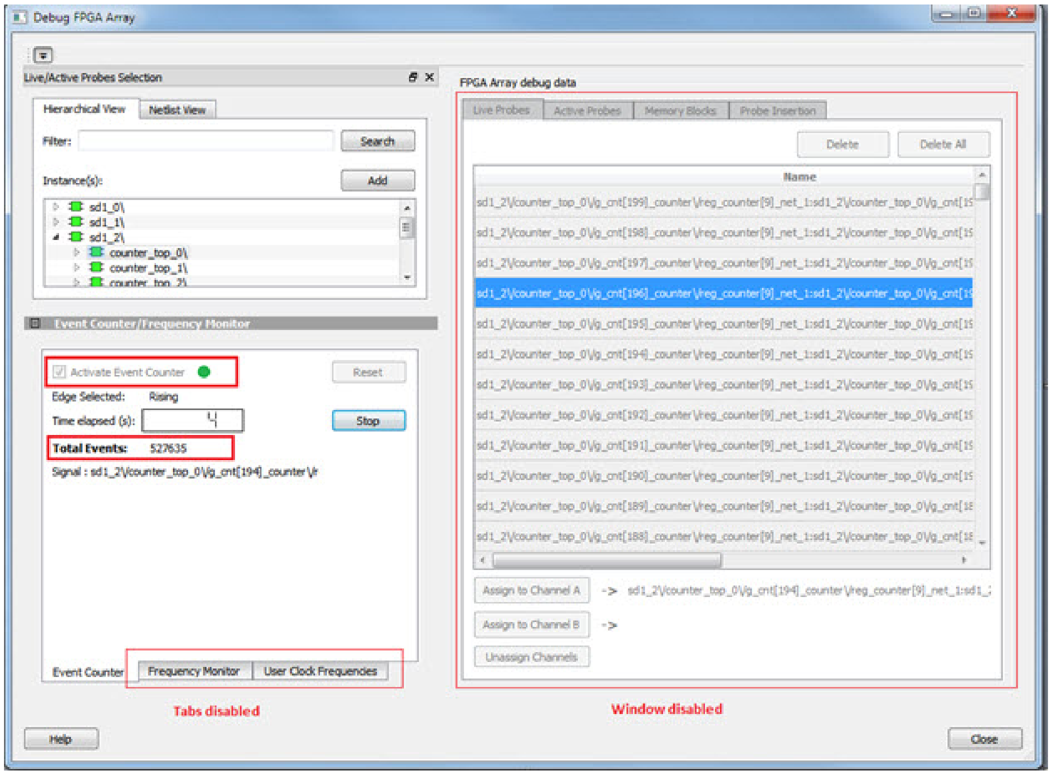
Assign to Channel A -> sd1_2\/counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_2

Assign to Channel B ->

Figure 27 · Activating the Event Counter - Assign Probe Channel

Running the Event Counter

Event Counter automatically runs the counter, which is indicated by a green LED. The counts are updated every second, and are shown next to Total Events. FPGA Array debug data and the control tabs in the Event Counter panel are disabled while Event Counter is running. When a signal is assigned, the signal name appears next to Signal.



Edge Selected: Rising

Time elapsed (s): 4

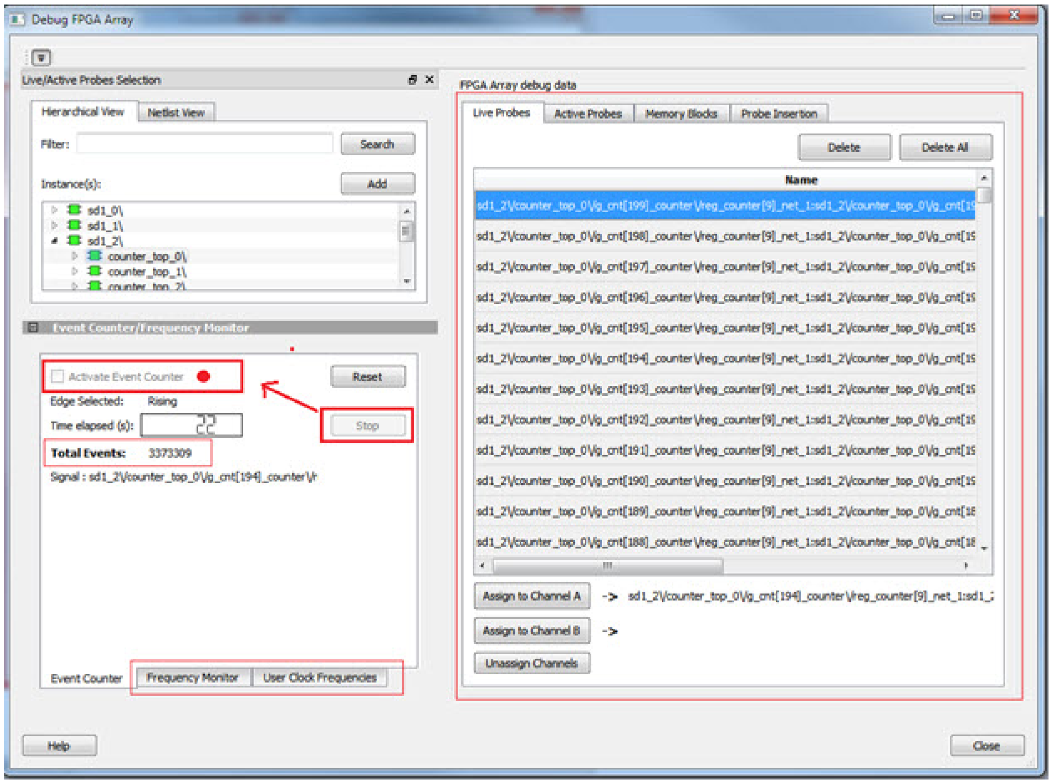
Total Events: 527635

Signal : sd1_2\/counter_top_0\/g_cnt[194]_counter\/r

Figure 28 · Running the Event Counter

Stopping the Event Counter

The only button enabled when Event Counter is running is the “Stop” button. Click button to stop counting. A red LED is shown to indicates the Event Counter has stopped. FPGA Array debug data and the control tabs in the Event Counter panel are enabled when Event Counter is not running.



Edge Selected: Rising

Time elapsed (s): 22

Total Events: 3373309

Signal : sd1_2\/counter_top_0\/g_cnt[194]_counter\/r

sd1_2\/counter_top_0\/g_cnt[199]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[198]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[197]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[196]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[195]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[193]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[191]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[190]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19

sd1_2\/counter_top_0\/g_cnt[189]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18

sd1_2\/counter_top_0\/g_cnt[188]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18

-> sd1_2\/counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_

->

Figure 29 · Stopping the Event Counter

Note: When a DC signal (signal tied to logic ‘0’) is assigned to Live Probe Channel A, or if there are no transitions on the signal assigned to Live Probe Channel A with initial state ‘0’, the Event Counter value is updated as ‘1’ when the counter is stopped. This is a limitation of the FHB IP, and will be fixed in upcoming releases.

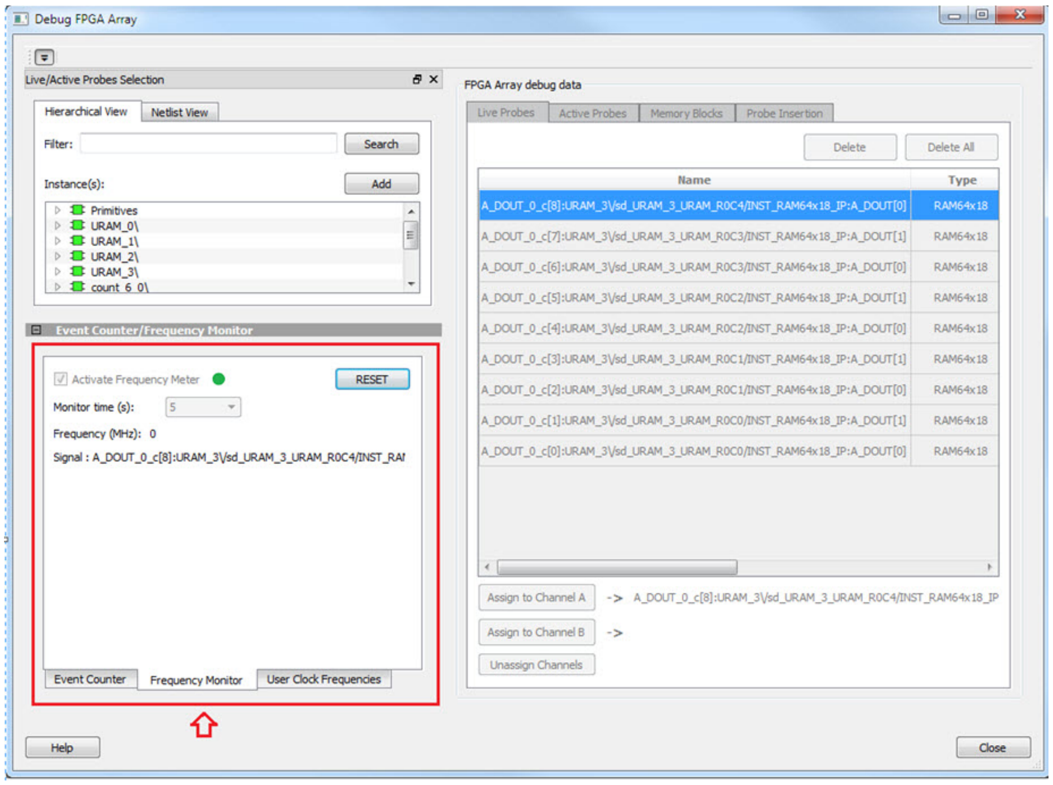
See Also

"Frequency Monitor" **Error! Bookmark not defined.**

"User Clock Frequencies" **Error! Bookmark not defined.**

Frequency Monitor

The Frequency Monitor calculates the frequency of any signal in the design that can be assigned to Live Probe channel A. The Frequency Monitor must be activated before or after the signal is assigned to Live Probe Channel A. You can enter the time to monitor the signal. The accuracy of results increases as the monitor time increases. The unit of measurement is displayed in Megahertz (MHz). During the run, progress is displayed in the pane.



Monitor time (s): 5

Frequency (MHz): 0

Signal : A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAI

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP

Assign to Channel B ->

Figure 30 · Frequency Monitor Tab/UI

In the Frequency Monitor tab, you can activate the Frequency Monitor, change the monitor time (delay to calculate frequency), reset the monitor, and set the frequency in megahertz (MHz). Click the drop-down list to select monitor time value. During the frequency calculation, all tabs on the right side of the window are disabled, as well as the tabs in the FPGA Hardware Breakpoint (FHB) pane.

Activating the Frequency Monitor

You can activate the Frequency Monitor in either of the following two ways:

- Click **Activate Frequency Monitor**, and then click the **Live Probe** tab and assign a signal to Channel A (Channel B is not configured for spatial debug operations).

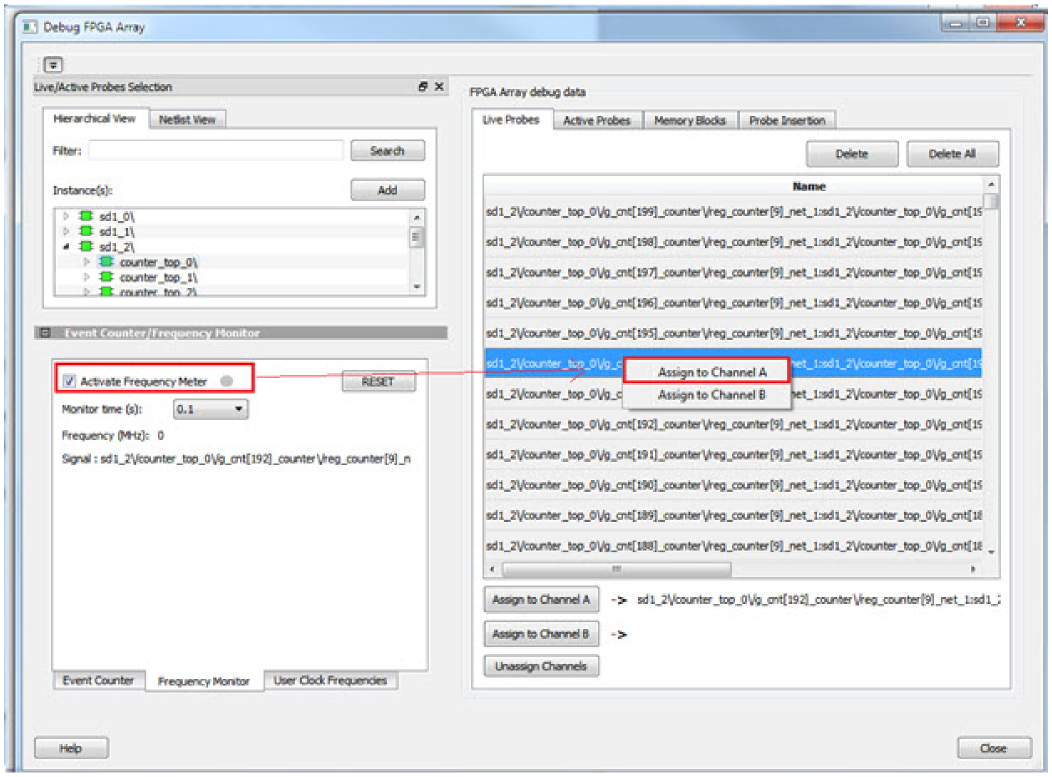


Figure 31 · Activating the Frequency Monitor - Assign a Signal

- Click the **Live Probe** tab and assign a signal to Channel A, and then click the **Frequency Monitor** tab and check the Activate Frequency Monitor checkbox.

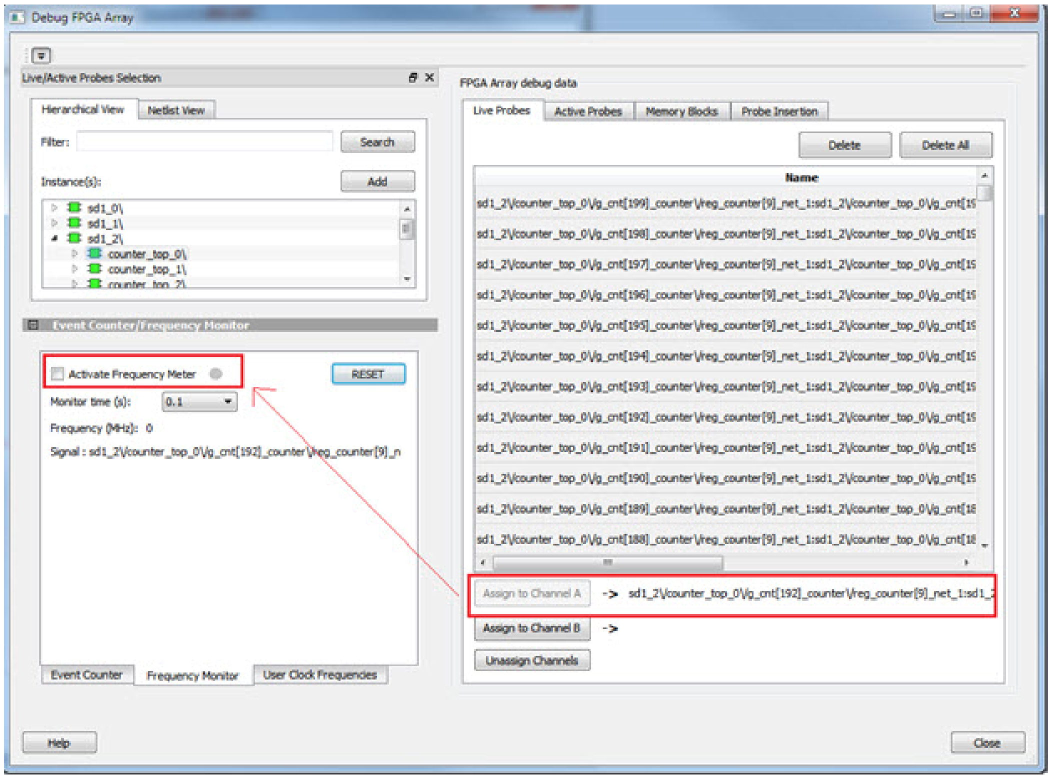
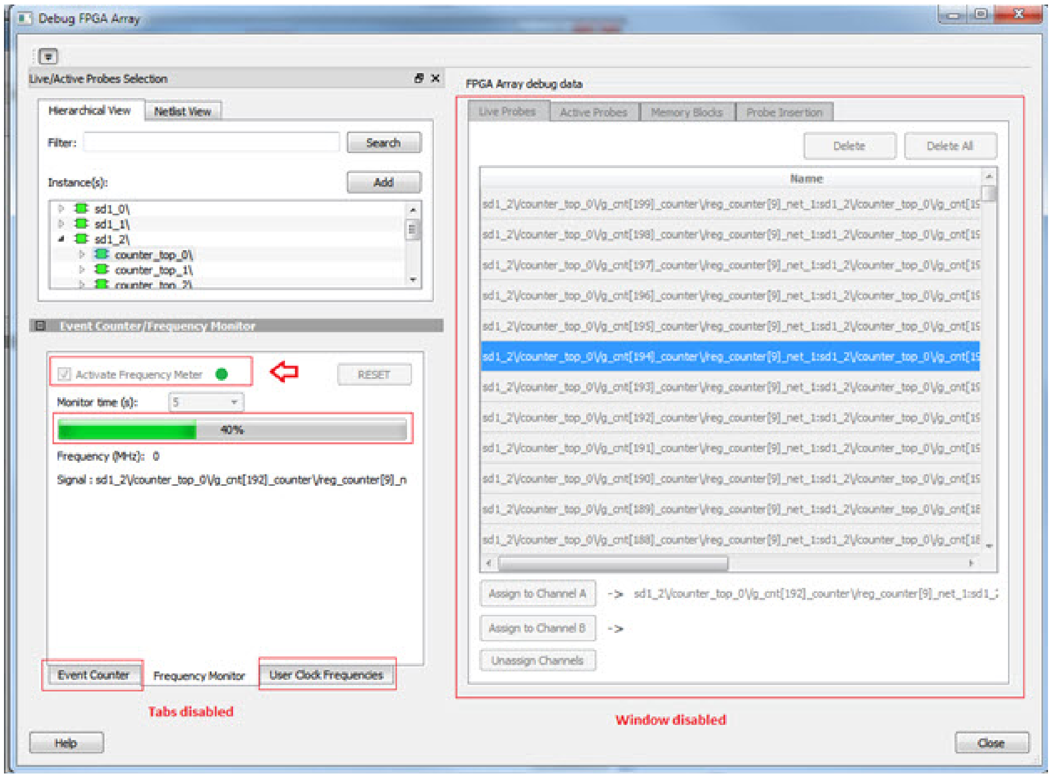


Figure 32 · Activating the Frequency Monitor

Running the Frequency Monitor

The Frequency Monitor runs automatically, and is indicated by a green LED. While it is running, FPGA Array debug data and the control tabs in the panel are disabled. A progress bar shows the monitor time progress when it is 1 second and above (as shown in the following figure). The Reset button is also disabled during the run. When a signal is assigned, the signal name appears next to Signal.



Monitor time (s): 5

40%

Frequency (MHz): 0

Signal : sd1_2\counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_n

```
sd1_2\counter_top_0\/g_cnt[199]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[198]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[197]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[196]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[195]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[193]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[191]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[190]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[189]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
sd1_2\counter_top_0\/g_cnt[188]_counter\/reg_counter[9]_net_1:sd1_2\counter_top_0\/g_cnt[1
```

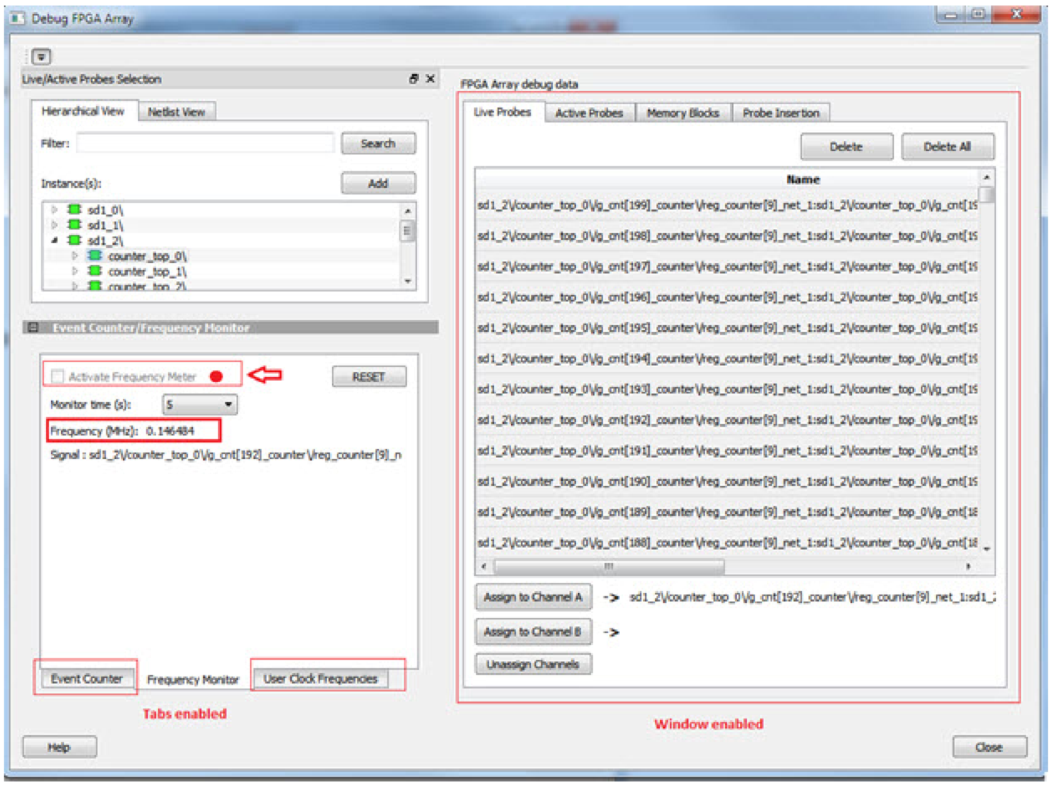
Assign to Channel A -> sd1_2\counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_

Assign to Channel B ->

Figure 33 · Running the Frequency Monitor

Stopping the Frequency Monitor

The Frequency Monitor stops when the specified monitor time has elapsed. This is indicated by a red LED. The result appears next to Frequency. The window and the tabs on the control panel are enabled. The Reset button is also enabled to reset the Frequency to 0 to start over the next iteration. The progress bar is hidden when the Frequency Monitor stops.



Monitor time (s): 5

Frequency (MHz): 0.146484

Signal : sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_n

```
sd1_2\/counter_top_0\/g_cnt[199]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[198]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[197]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[196]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[195]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[194]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[193]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[191]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[190]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[19
sd1_2\/counter_top_0\/g_cnt[189]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18
sd1_2\/counter_top_0\/g_cnt[188]_counter\/reg_counter[9]_net_1:sd1_2\/counter_top_0\/g_cnt[18
```

-> sd1_2\/counter_top_0\/g_cnt[192]_counter\/reg_counter[9]_net_1:sd1_2

->

Figure 34 · Stopping the Frequency Monitor

See Also

"Event Counter" **Error! Bookmark not defined.**

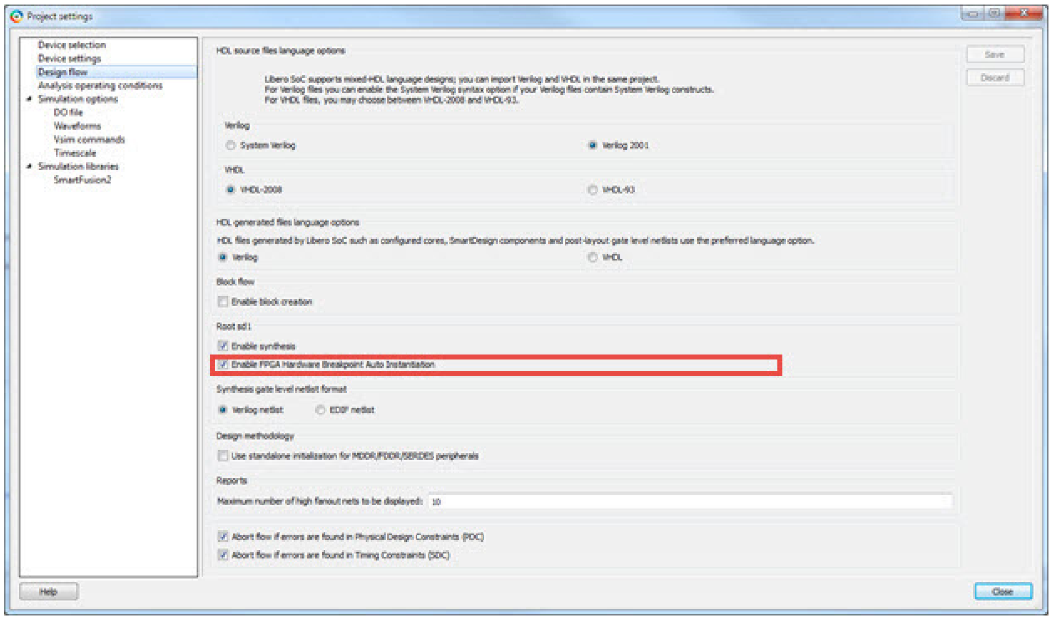
"User Clock Frequencies" **Error! Bookmark not defined.**

FPGA Hardware Breakpoint Auto Instantiation

The FPGA Hardware Breakpoint (FHB) Auto Instantiation feature automatically instantiates an FHB instance per clock domain that is using gated clocks (GL0/GL1/GL2/GL3) from an FCCC instance. The FHB instances gate the clock domain they are instantiated on. These instances can be used to force halt the design or halt the design through a live probe signal. Once a selected clock domain or all clock domains are halted, you can play or step on the clock domains, either selectively or all at once. The FPGA Hardware Breakpoint controls in the Smart Debug UI allow you to control the debugging cycle.

Note: This option is enabled in the Enhanced Constraint Flow when you select Verilog netlist as the Synthesis gate level netlist format in Project Settings (**Project Settings > Design flow**). It is only available for SmartFusion2 and IGLOO2 devices.

To enable this option, select the Enable FHB Auto Instantiation check box in the Design flow tab of the Project Settings dialog box (**Libero > File > Project Settings**). See the example figure that follows.



HDL source files language options

Libero SoC supports mixed-HDL language designs; you can import Verilog and VHDL in the same project.
For Verilog files you can enable the System Verilog syntax option if your Verilog files contain System Verilog constructs.
For VHDL files, you may choose between VHDL-2008 and VHDL-93.

Verilog: System Verilog [✓ Verilog 2001]

VHDL: [✓ VHDL-2008] VHDL-93

HDL generated files language options

HDL files generated by Libero SoC such as configured cores, SmartDesign components and post-layout gate level netlists use the preferred language option.

[✓ Verilog] VHDL

Block flow

☐ Enable block creation

Root sd1

☑ Enable synthesis

☑ Enable FPGA Hardware Breakpoint Auto Instantiation

Synthesis gate level netlist format

[✓ Verilog netlist] EDIF netlist

Design methodology

☐ Use standalone initialization for MDDR/FDDR/SERDES peripherals

Reports

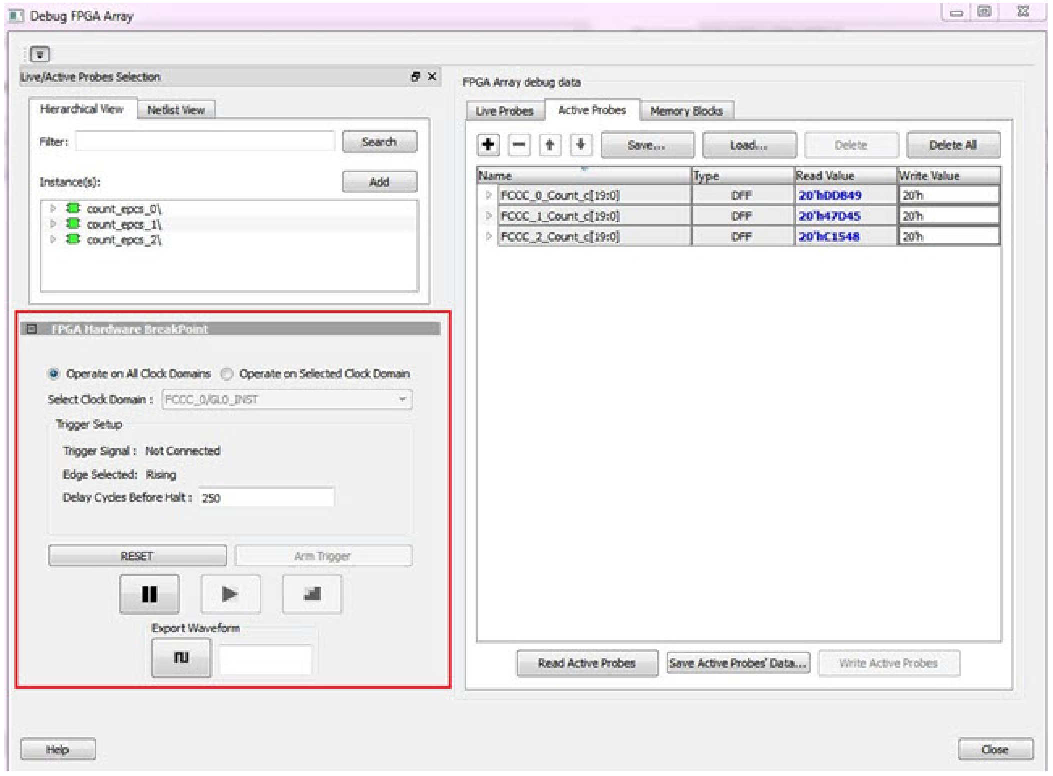
Maximum number of high fanout nets to be displayed: 10

☑ Abort flow if errors are found in Physical Design Constraints (PDC)

☑ Abort flow if errors are found in Timing Constraints (SDC)

Figure 35 · Enable FHB Auto Instantiation in Project Settings: Design flow Window

FPGA Hardware Breakpoint (FHB) controls appear in the Debug FPGA Array dialog box when there is an auto-instantiated FHB instance in the design. See the example figure that follows.



count_epcs_0\
count_epcs_1\
count_epcs_2\

[✓ Operate on All Clock Domains] Operate on Selected Clock Domain

Select Clock Domain : FCCC_0/GL0_INST

Trigger Setup

Trigger Signal : Not Connected

Edge Selected: Rising

Delay Cycles Before Halt : 250

Name	Type	Read Value	Write Value
FCCC_0_Count_c[19:0]	DFF	20'hDD849	20'h
FCCC_1_Count_c[19:0]	DFF	20'h47D45	20'h
FCCC_2_Count_c[19:0]	DFF	20'hC1548	20'h

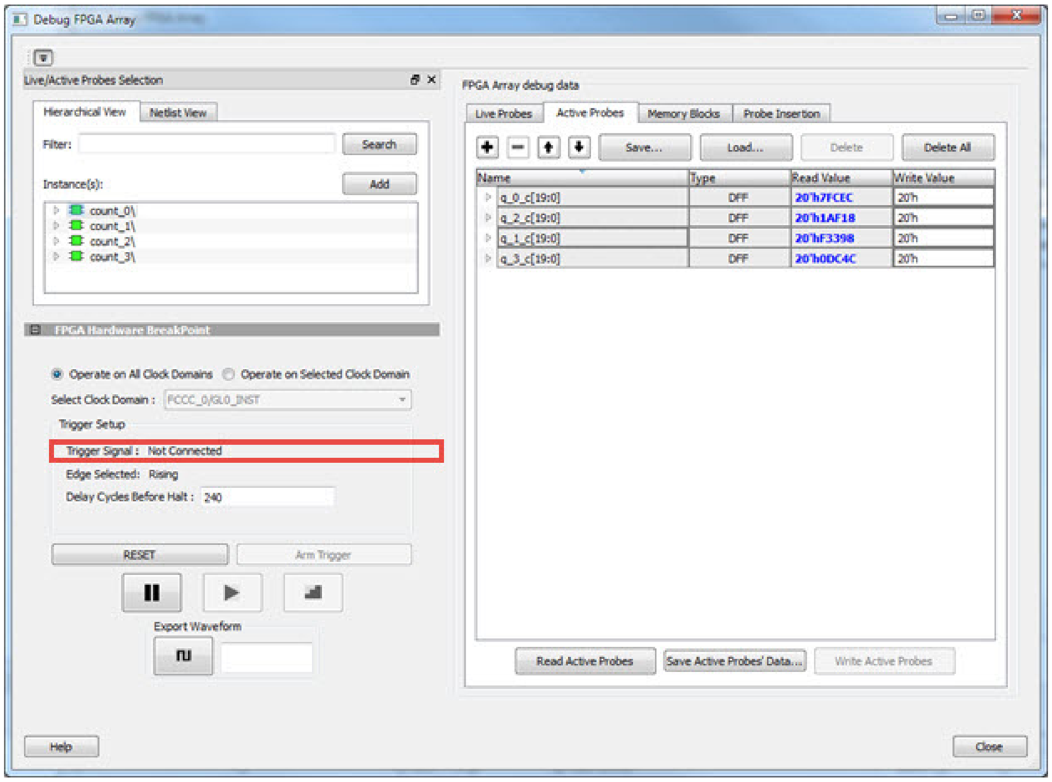
Figure 36 · FPGA Hardware Breakpoint (FHB) Controls

You can choose **Operate on All Clock Domains** or **Operate on Selected Clock Domain** by selecting the appropriate radio button. Selecting either of these modes sets the FHB instances to the respective mode. Once you assign the Live Probe PROBE_A connection and click **Arm Trigger**, the DUT halts on the next positive edge that occurs on the signal connected to Live Probe PROBE_A.

When you choose **Operate on Selected Clock Domain** mode, the Select Clock Domain combo box is enabled, and all available clock domains are listed. The Halt (Pause), Play, and Step buttons are associated for that clock domain. If you switch between clock domains in this mode, previous clock domain settings are not retained.

When you choose **Operate on All Clock Domains** mode, the Select Clock Domain combo box is disabled. The Halt, Play, and Step buttons are associated for all clock domains.

The Trigger Signal is shown as Not Connected until a live probe is assigned. See the example figure that follows.



[✓ Operate on All Clock Domains] Operate on Selected Clock Domain

Select Clock Domain : FCCC_0/GL0_INST

Trigger Setup

Trigger Signal : Not Connected

Edge Selected: Rising

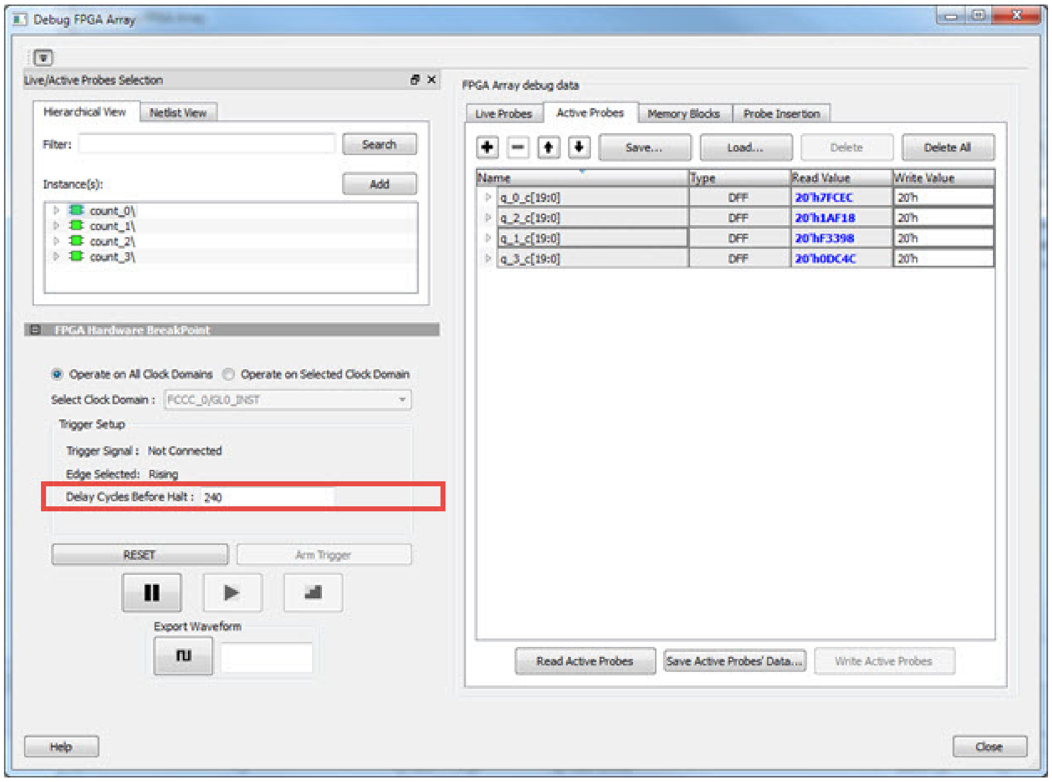
Delay Cycles Before Halt : 240

Name	Type	Read Value	Write Value
q_0_c[19:0]	DFF	20'h7FCEC	20'h
q_2_c[19:0]	DFF	20'h1AF18	20'h
q_1_c[19:0]	DFF	20'hF3398	20'h
q_3_c[19:0]	DFF	20'h0DC4C	20'h

When a probe is assigned to Live Probe PROBE_A, the Trigger Signal updates.

If you require a certain number of clock cycles before halting the clock domain after triggering, a value between 0 and 255 must be entered for Delay Cycles Before Halt before you click **Arm Trigger**. This sets the FHBs to trigger after the specified delay from the rising edge trigger.

Delay is not applied to a forced Halt. See the example figure that follows.



Filter:

Instance(s):
- count_0\
- count_1\
- count_2\
- count_3\

(●) Operate on All Clock Domains ( ) Operate on Selected Clock Domain

Select Clock Domain : FCCC_0/GL0_INST

Trigger Setup

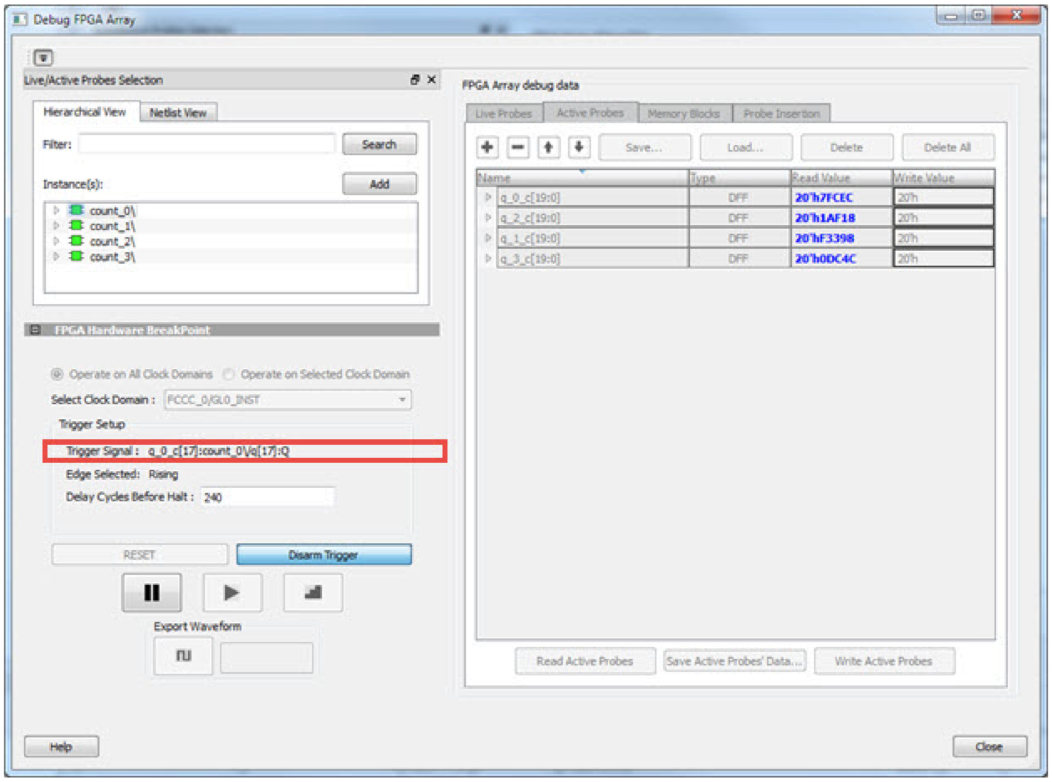
Trigger Signal : Not Connected

Edge Selected: Rising

Delay Cycles Before Halt : 240

Name	Type	Read Value	Write Value
q_0_c[19:0]	DFF	20'h7FCEC	20'h
q_2_c[19:0]	DFF	20'h1AF18	20'h
q_1_c[19:0]	DFF	20'hF3398	20'h
q_3_c[19:0]	DFF	20'h0DC4C	20'h

When a live probe connection is made, you click **Arm Trigger**, FPGA Hardware Breakpoint functionality is disabled until the trigger is disarmed automatically or the design is force halted.



Filter:

Instance(s):
- count_0\
- count_1\
- count_2\
- count_3\

(●) Operate on All Clock Domains ( ) Operate on Selected Clock Domain

Select Clock Domain : FCCC_0/GL0_INST

Trigger Setup

Trigger Signal : q_0_c[17]:count_0\/q[17]:Q

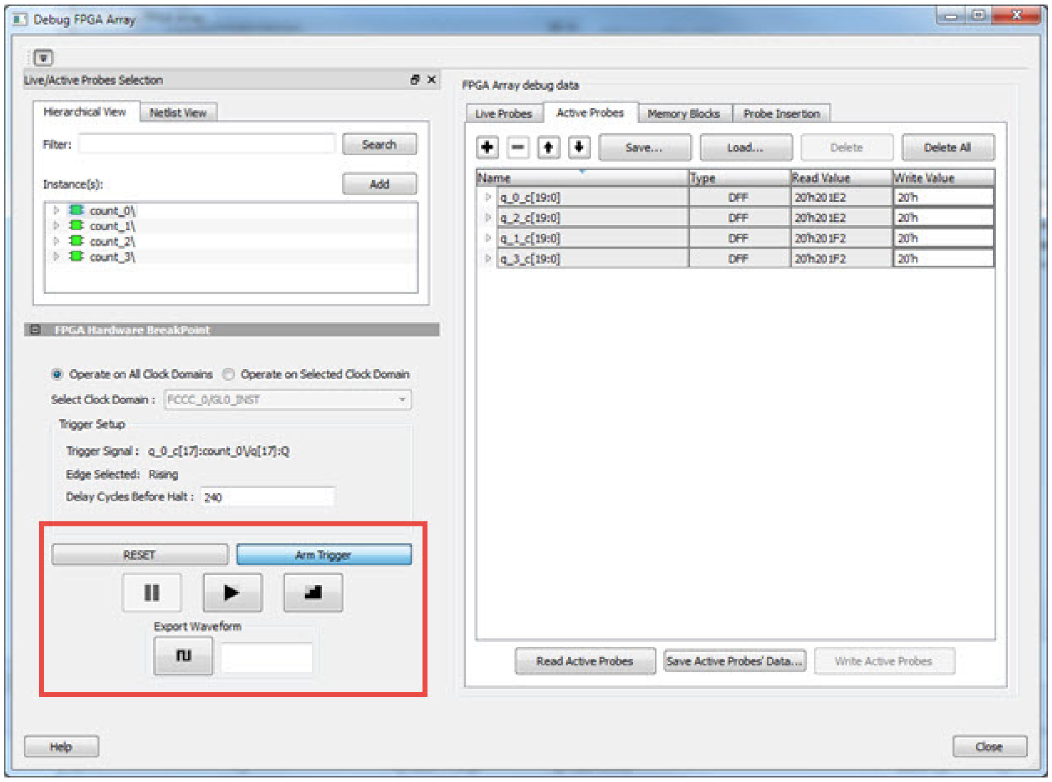
Edge Selected: Rising

Delay Cycles Before Halt : 240

Name	Type	Read Value	Write Value
q_0_c[19:0]	DFF	20'h7FCEC	20'h
q_2_c[19:0]	DFF	20'h1AF18	20'h
q_1_c[19:0]	DFF	20'hF3398	20'h
q_3_c[19:0]	DFF	20'h0DC4C	20'h

Force Halt/Play/Step is done using the FPGA Hardware Breakpoint controls (see the example figure that follows). Once the clock domain is halted, you can either force Play the clock domain or Step the clock domain by 1 clock cycle.

You can save the waveform view of the selected active probes using Export Waveform by specifying the number of clock cycles to capture. The waveform is saved to a .vcd file.



[✓ Operate on All Clock Domains] Operate on Selected Clock Domain

Select Clock Domain : FCCC_0/GL0_INST

Trigger Setup

Trigger Signal : q_0_c[17]:count_0\/q[17]:Q

Edge Selected: Rising

Delay Cycles Before Halt : 240

Name	Type	Read Value	Write Value
q_0_c[19:0]	DFF	20'h201E2	20'h
q_2_c[19:0]	DFF	20'h201E2	20'h
q_1_c[19:0]	DFF	20'h201F2	20'h
q_3_c[19:0]	DFF	20'h201F2	20'h

FPGA Hardware Breakpoint Operations

Live Probe Halt

You can halt a selected clock domain or all clock domains in Live Probe Halt mode based on the mode selection (**Operate on All Clock Domains** or **Operate on Selected Clock Domain**).

Assign a signal to Live Probe PROBE_A in the **Live Probes** tab of the UI, and then click the **Active Probe** tab to see the FPGA Hardware Breakpoint controls.

Click **Arm Trigger** to arm the FHBs to look for a trigger on the signal connected to Live Probe PROBE_A.

Once the trigger occurs, the clock domains are halted.

Note: If only one clock domain is halted, other clock domains continue to run, and you should anticipate results accordingly.

See Assumptions and Limitations for more information.

Force Halt

You can force halt a selected clock domain or all clock domains based on mode selection without having to wait for a trigger from a live probe signal. Click the **Halt** button in the FPGA Hardware Breakpoint (FHB) controls.

In **Operate on Selected Clock Domain** mode, the state of the Halt button is updated based on the state of the clock domain selected.

In **Operate on all Clock Domains** mode, the Halt button is disabled only when all clock domains are halted. Each clock domain is halted sequentially in the order shown in the Select Clock Domain combo box.

Note: If only one clock domain is halted, other clock domains continue to run, and you should anticipate results accordingly.

Play

Once the clock domain is in a halted state (live probe halt or force halt), you can click **Play** in the FPGA Hardware Breakpoint controls. This resumes the clock domain from the halted state.

In **Operate on all Clock Domains** mode, each clock domain runs sequentially in the order shown in the Select Clock Domain combo box.

Step

Once the clock domain is in a halted state (live probe halt or force halt), you can click the **Step** button in the FPGA Hardware Breakpoint controls. This advances the clock domain by one clock cycle and holds the state of the clock domain.

In **Operate on All Clock Domains** mode, each clock domain steps sequentially in the order shown in the Select Clock Domain combo box.

Waveform Capture

You can save the waveform view of the selected active probes using Export Waveform by specifying the number of clock cycles to capture in text box and then clicking **Capture Waveform** . The waveform is saved to a .vcd file.

You can view the waveforms by importing the .vcd file. The waveform file can be viewed in any waveform viewer that supports vcd format.

Reset

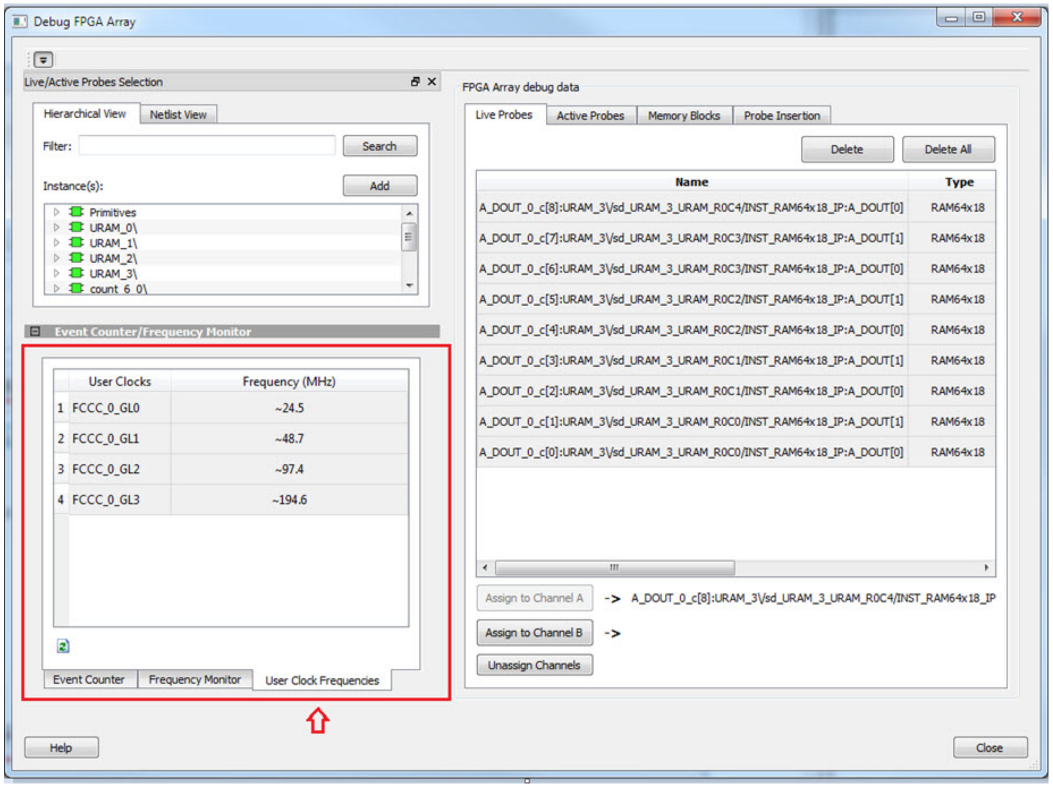
You can reset a selected clock domain or all clock domains (based on the mode selection) by clicking **RESET** at any time. This resets the FHBs on clock domains and instructs FHB muxes not to look for a trigger.

Assumptions and Limitations

- If you select the auto instantiation option in Libero, you need to rerun Synthesis (if already run) to get the FHB related functionality.
- The auto instantiation flow is supported for SmartFusion2 and IGLOO2 only.
- Available for Enhanced Constraint Flow only.
- Supported for FCC driven gated clocks (GL0/GL1/GL2/GL3) only.
- CLKINT_PRESERVE – FHB is not auto-instantiated if the user design contains this macro.
- Designs that have Encrypted IPs are not supported.
- EDIF using constraints flow is not supported.
- Live Probe triggering occurs on the Positive Edge only.
- For imported Verilog netlist files (.vm files), you must rerun synthesis to get FHB-related functionality. If synthesis is disabled and the netlist is compiled directly, FHB functionality is not inferred.
- If only one clock domain is halted during operations, other clock domains continue to run, and you should anticipate results accordingly.
- FHB performance can only be characterized against the clock which it is running at (i.e. 50MHz).
  - If the DUT clock is running at or less than 50MHz, the DUT clock will halt within one clock cycle (1 or less).
  - For frequencies higher than 50MHz, the point at which the DUT halts cannot be guaranteed.

User Clock Frequencies

The User Clock Frequencies tab shows the frequencies that have been configured from the FCCC block. If assigned, live probe channels are temporarily unassigned, and reassigned after user clock frequencies have been calculated. The Refresh button recalculates frequencies if clocks have been changed.



	User Clocks	Frequency (MHz)
1	FCCC_0_GL0	~24.5
2	FCCC_0_GL1	~48.7
3	FCCC_0_GL2	~97.4
4	FCCC_0_GL3	~194.6

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP
Assign to Channel B ->

Figure 37 · User Clock Frequencies Tab/UI

See Also

"Event Counter" **Error! Bookmark not defined.**

"Frequency Monitor" **Error! Bookmark not defined.**

UG0449- SmartFusion2 and IGLOO2 Clocking Resources User Guide

UG0586- RTG4 FPGA Clocking Resources User Guide

Pseudo Static Signal Polling

With Active Probes you can check the current state of any probe in the design. However, in most cases, you will not able to time the active probe read to capture its intended value. For these cases, you can use Pseudo Static Signal Polling, in which the SmartDebug software polls the signal at intervals of one second to check if the probe has the intended value. This feature is useful in probing signals which reach the intended state and stay in that state.

From the Active Probes tab in the Debug FPGA Array dialog box, right-click a signal, bus, or group and choose **Poll...**. See the example figure that follows.

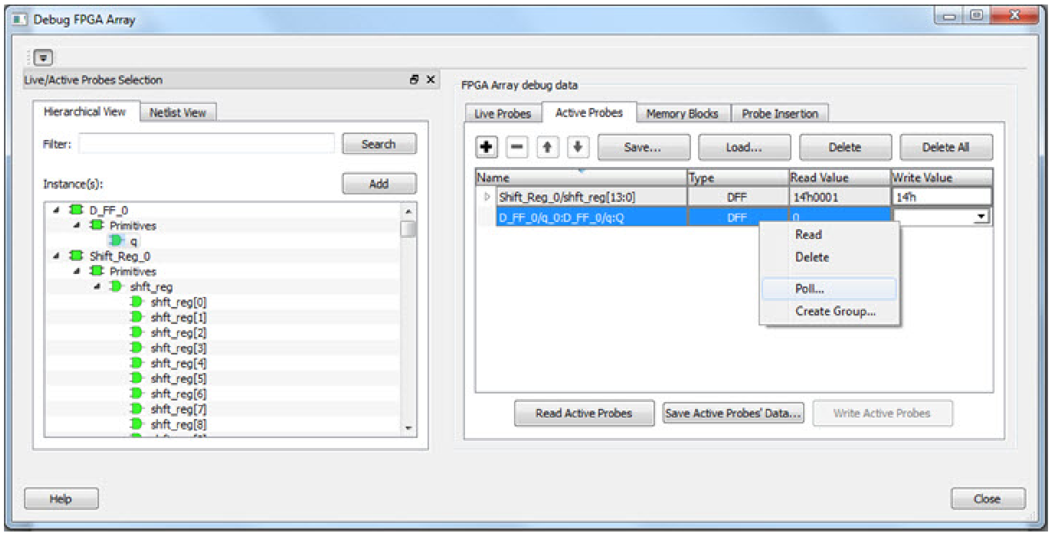


Figure 38 · Debug FPGA Array Dialog Box - Poll Option

The Pseudo-static signal polling dialog box opens.

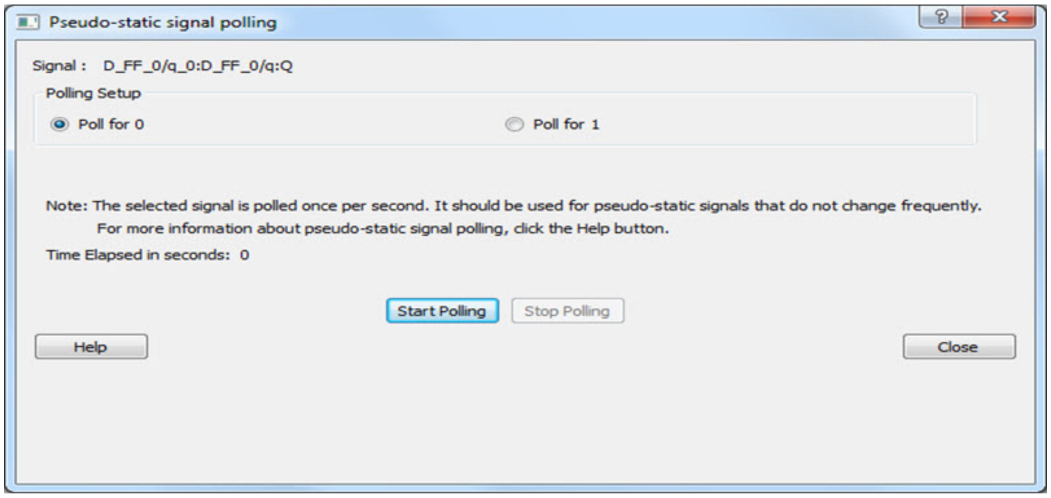
Scalar Signal Polling

Polling Setup

To poll scalar signals, select **Poll for 0** or **Poll for 1**.

The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently. The elapsed time is shown next to **Time Elapsed in seconds**.

To begin polling, click **Start Polling**. See the following example figure.



Signal : D_FF_0/q_0:D_FF_0/q:Q

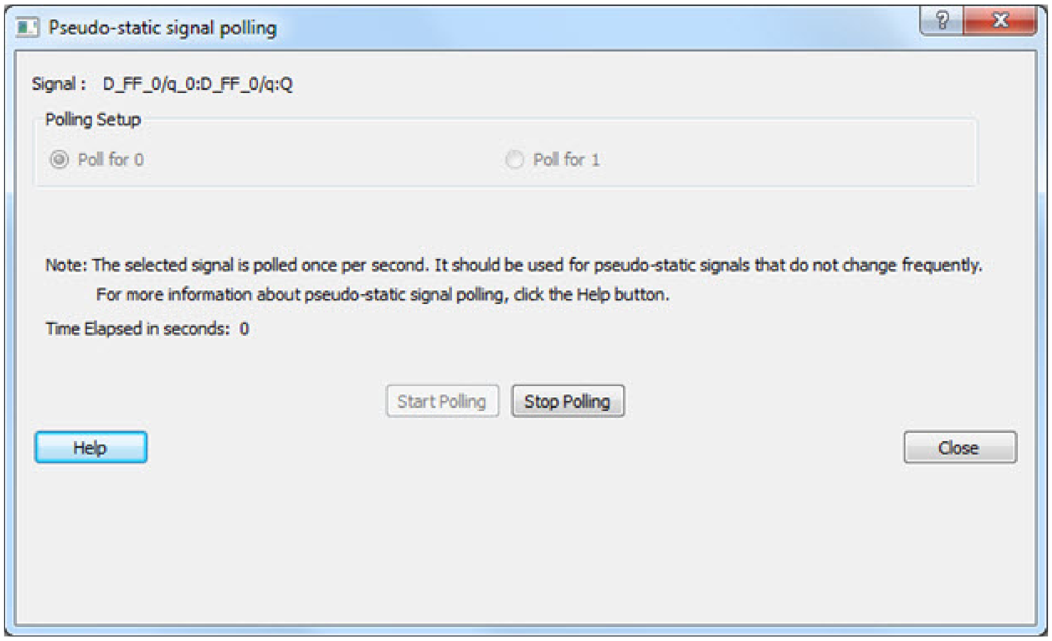
[✓ Poll for 0] Poll for 1

Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

Time Elapsed in seconds: 0

Figure 39 · Pseudo-static signal polling Dialog Box (Scalar Signal Polling) - Start Polling

To end polling, click **Stop Polling**. See the following example figure.


Pseudo-static signal polling

Signal : D_FF_0/q_0:D_FF_0/q:Q

Polling Setup

[✓ Poll for 0] Poll for 1

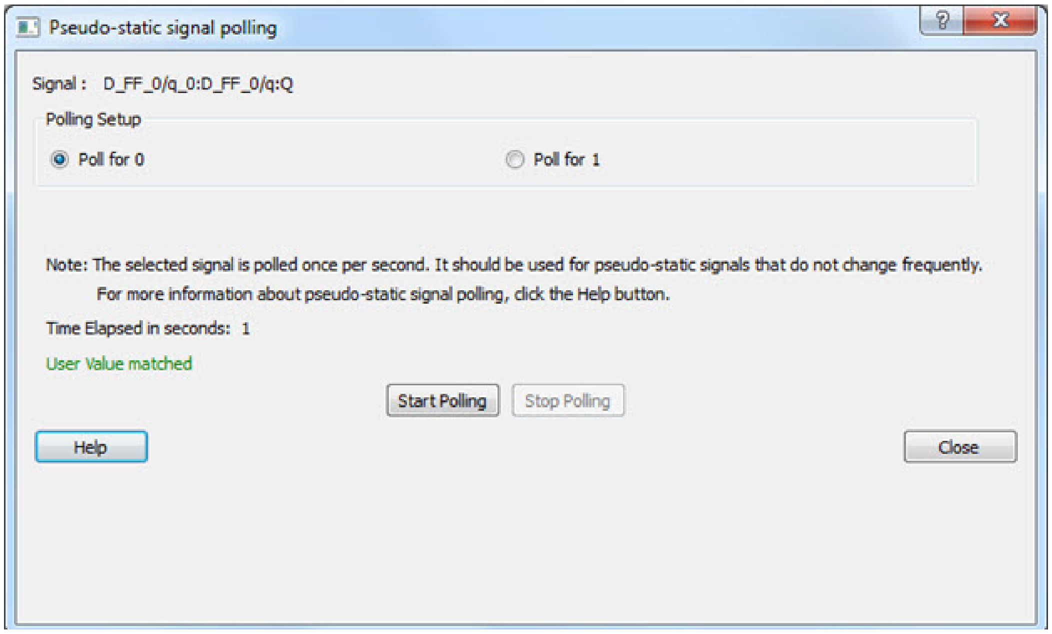
Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

Time Elapsed in seconds: 0

Figure 40 · Pseudo-static signal polling Dialog Box (Scalar Signal Polling) - Stop Polling

Note: You cannot change the poll value or close the polling dialog box while polling is in progress.

The elapsed time is updated in seconds until the polled value is found. When the polled value is found, **User value matched** is displayed in green in the dialog box. See the following example figure.


Pseudo-static signal polling

Signal : D_FF_0/q_0:D_FF_0/q:Q

Polling Setup

[✓ Poll for 0] Poll for 1

Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

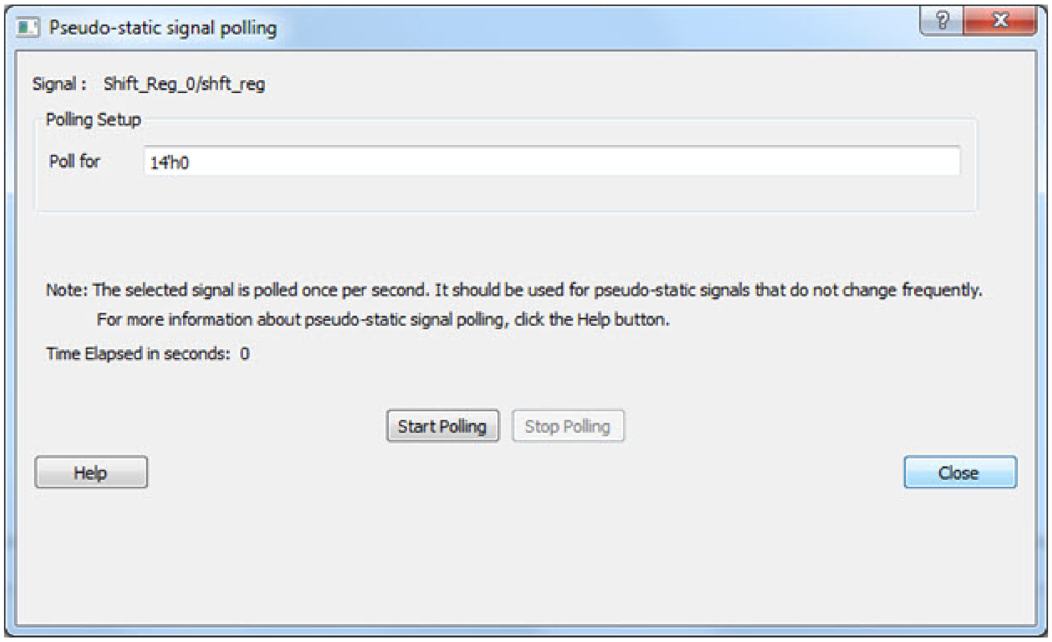
Time Elapsed in seconds: 1

User Value matched

Figure 41 · Pseudo-static signal polling Dialog Box (Scalar Signal Polling) - User Value matched

Vector Signal Polling

To poll vector signals, enter a value in the text box. The entered value is checked and validated. If an invalid value is entered, start polling is disabled, and an example displays showing the required format. See the following example figures.


Pseudo-static signal polling

Signal : Shift_Reg_0/shft_reg

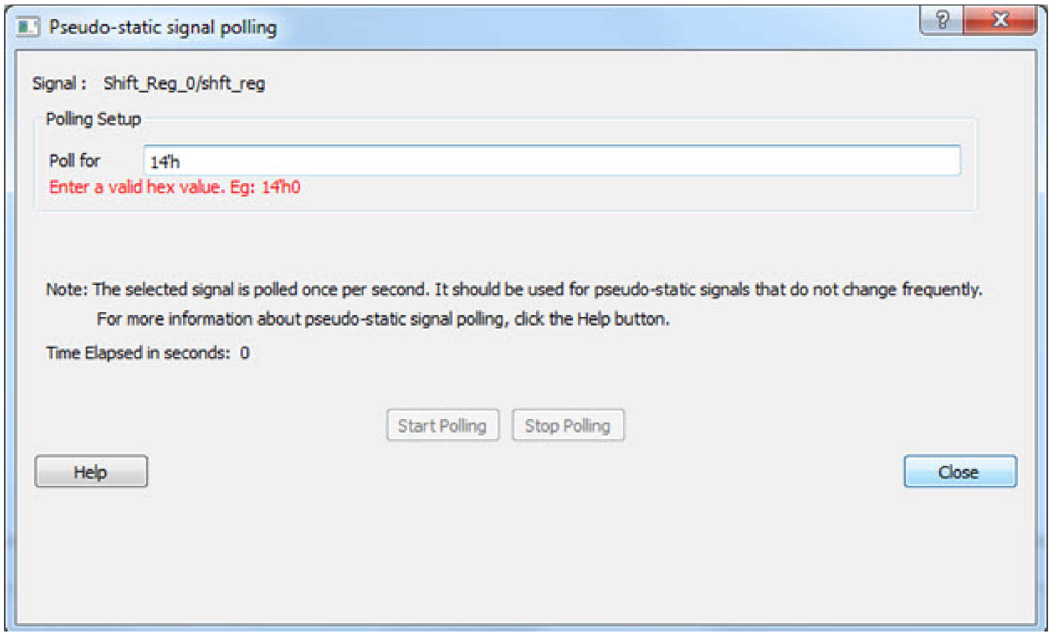
Polling Setup

Poll for 14'h0

Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

Time Elapsed in seconds: 0

Figure 42 · Pseudo-static signal polling Dialog Box (Vector Signal Polling)


Pseudo-static signal polling

Signal : Shift_Reg_0/shft_reg

Polling Setup

Poll for 14'h

Enter a valid hex value. Eg: 14'h0

Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

Time Elapsed in seconds: 0

Figure 43 · Pseudo-static signal polling Dialog Box (Vector Signal Polling) -- After Validation

When you enter a valid value and click **Start Polling** is clicked, polling begins.

To end polling, click **Stop Polling**.

Note: You cannot change the poll value or close the polling dialog box while polling is in progress.

The elapsed time is updated in seconds until the polled value is found. When the polled value is found, **User value matched** is displayed in green in the dialog box.

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.

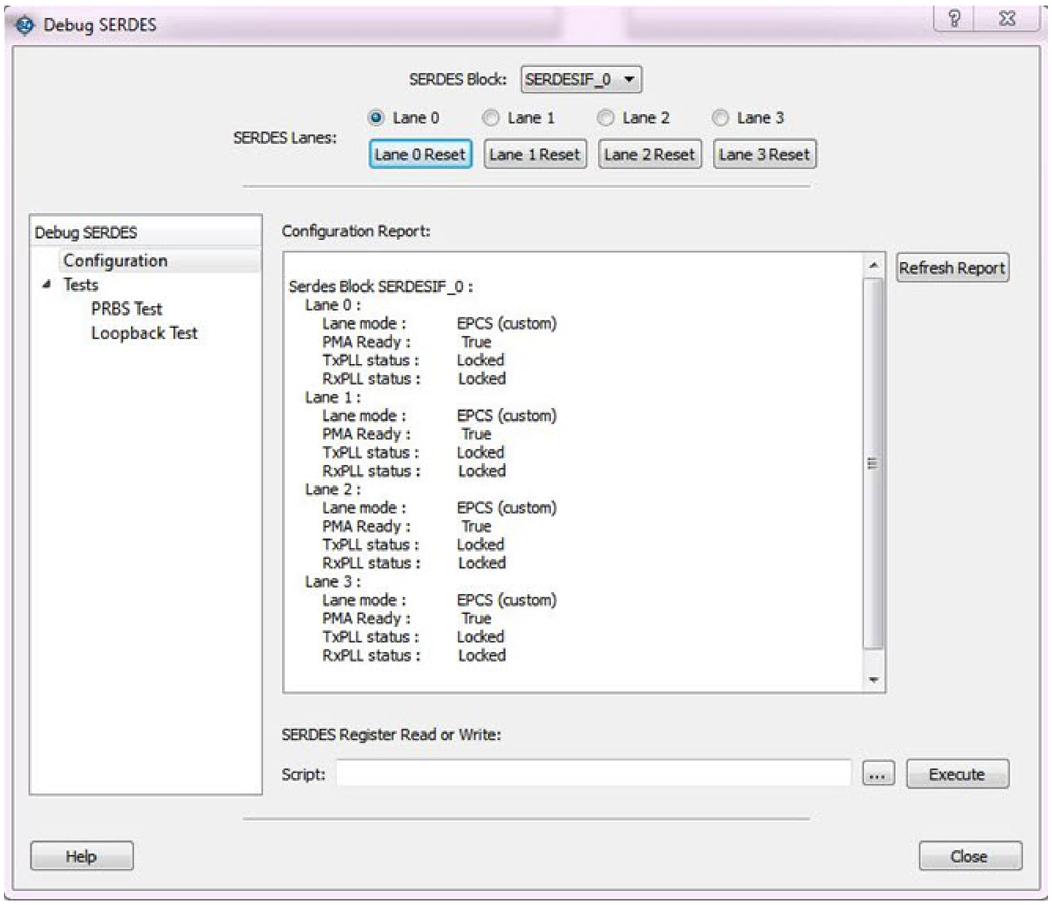
TxPll status - Indicates the loss-of-lock status for the TXPLL is asserted and remains asserted until the PLL reacquires lock.

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.



```
Serdes Block SERDESIF_0 :
  Lane 0 :
     Lane mode :       EPCS (custom)
     PMA Ready :        True
     TxPLL status :    Locked
     RxPLL status :    Locked
  Lane 1 :
     Lane mode :       EPCS (custom)
     PMA Ready :        True
     TxPLL status :    Locked
     RxPLL status :    Locked
  Lane 2 :
     Lane mode :       EPCS (custom)
     PMA Ready :        True
     TxPLL status :    Locked
     RxPLL status :    Locked
  Lane 3 :
     Lane mode :       EPCS (custom)
     PMA Ready :        True
     TxPLL status :    Locked
     RxPLL status :    Locked
```

Figure 44 · 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.

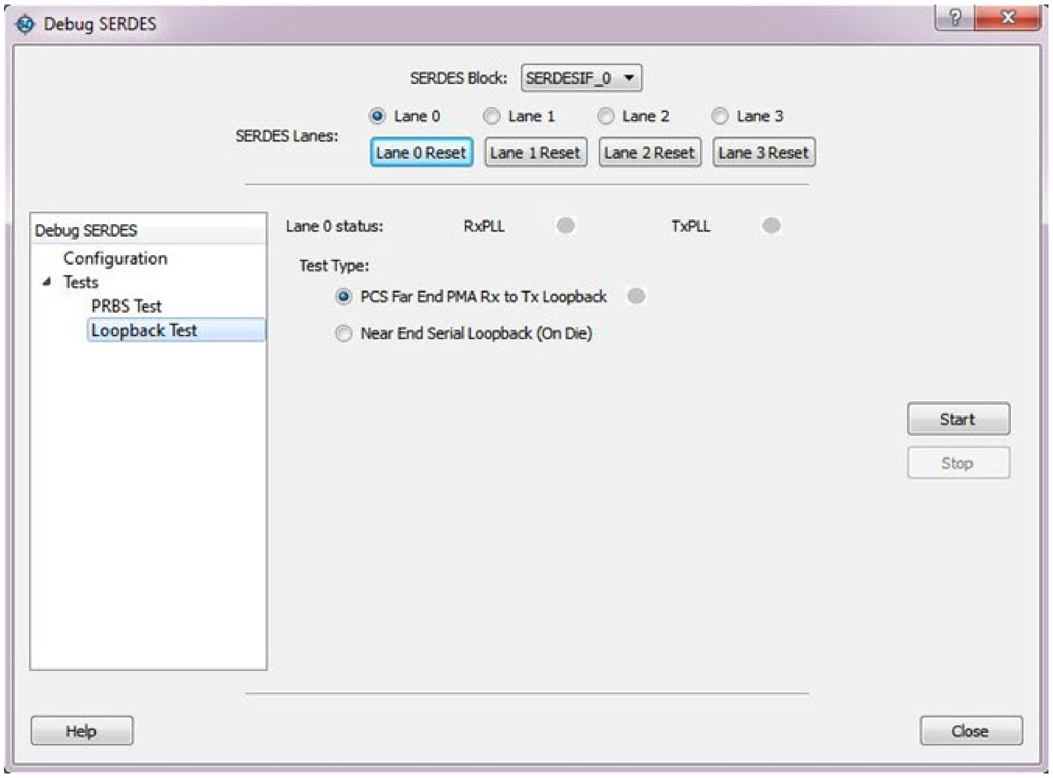


Figure 45 · Debug SERDES - Loopback Test

Debug SERDES – PRBS Test

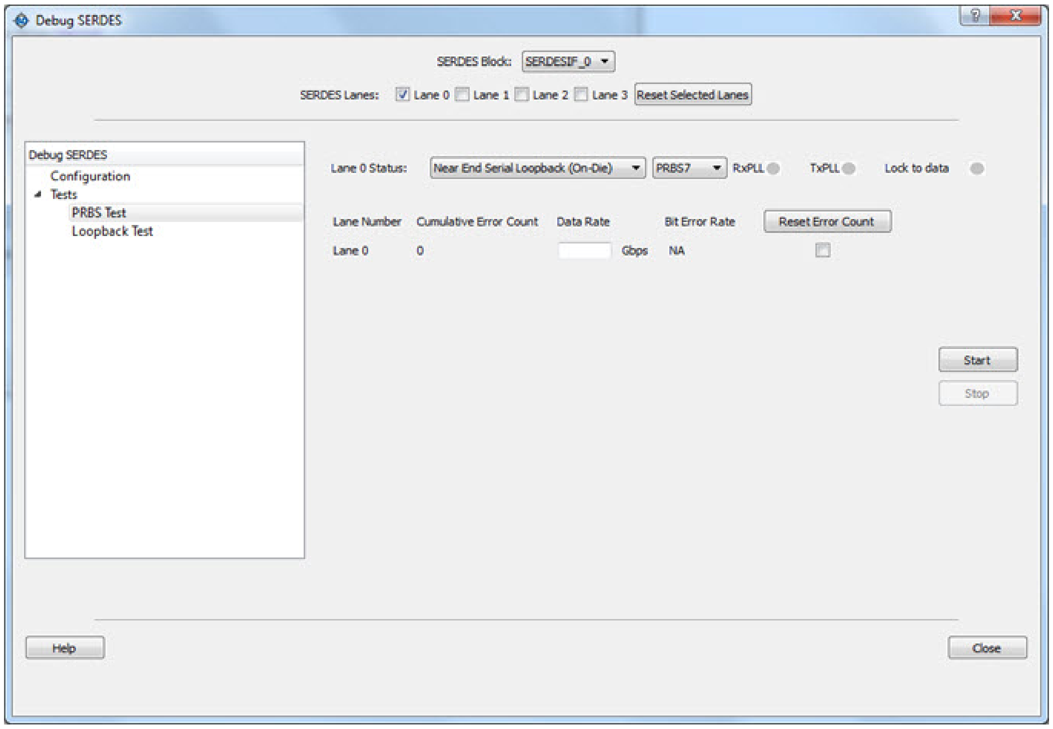
PRBS data stream patterns are generated and checked by the internal SERDES block. These are used to self-test signal integrity of the device. You can switch the device through several predefined patterns.

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

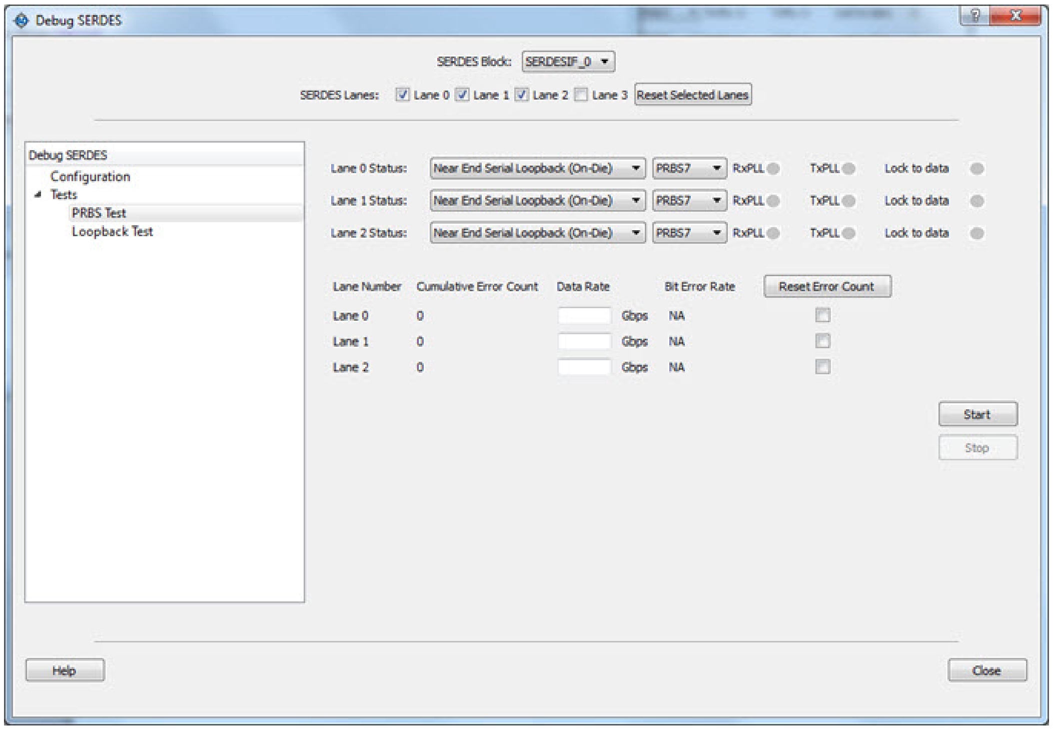
Check the box or boxes to select the lane(s) on which to run the PRBS test. Then select the Lane Status, test type, and pattern for each lane you have selected. Lane mode indicates the programmed mode on a SERDES lane as defined by the SERDES system register. See the examples below.



Lane 0 Status: Near End Serial Loopback (On-Die) PRBS7 RxPLL TxPLL Lock to data

Lane Number	Cumulative Error Count	Data Rate		Bit Error Rate	Reset Error Count
Lane 0	0		Gbps	NA	

Figure 46 · SERDES Lanes - Single Lane Selected



Lane 0 Status: Near End Serial Loopback (On-Die) PRBS7 RxPLL TxPLL Lock to data
Lane 1 Status: Near End Serial Loopback (On-Die) PRBS7 RxPLL TxPLL Lock to data
Lane 2 Status: Near End Serial Loopback (On-Die) PRBS7 RxPLL TxPLL Lock to data

Lane Number	Cumulative Error Count	Data Rate		Bit Error Rate	Reset Error Count
Lane 0	0		Gbps	NA	
Lane 1	0		Gbps	NA	
Lane 2	0		Gbps	NA	

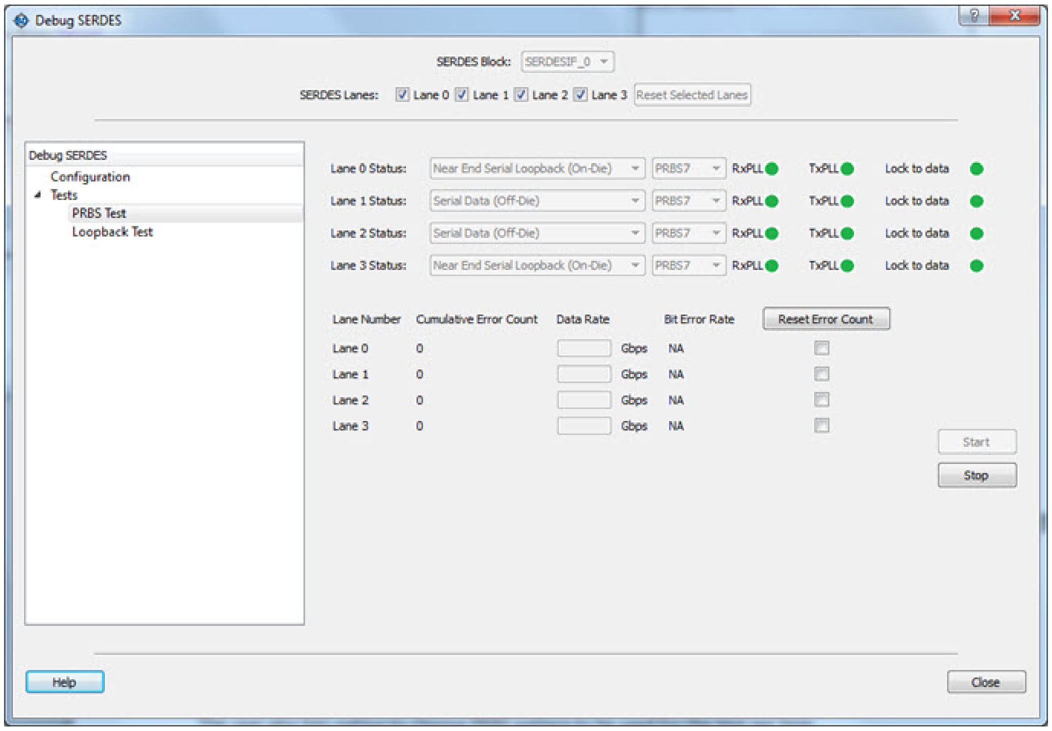
Figure 47 · SERDES Lanes - Multiple Lanes Selected

Test Type

Near End Serial Loopback (On-Die) enables a self-test of the device. The serial data stream is sent internally 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.

If more than one SERDES Lane has been selected, the test type can be selected per lane. In the following example, Near End Serial Loopback (On-Die) has been selected for Lane 0 and Lane 3, and Serial Data (Off-Die) has been selected for Lane 1 and Lane 2.



SERDES Block: SERDESIF_0

SERDES Lanes: ☑ Lane 0 ☑ Lane 1 ☑ Lane 2 ☑ Lane 3

Lane 0 Status:	Near End Serial Loopback (On-Die)	PRBS7	RxPLL ●	TxPLL ●	Lock to data ●
Lane 1 Status:	Serial Data (Off-Die)	PRBS7	RxPLL ●	TxPLL ●	Lock to data ●
Lane 2 Status:	Serial Data (Off-Die)	PRBS7	RxPLL ●	TxPLL ●	Lock to data ●
Lane 3 Status:	Near End Serial Loopback (On-Die)	PRBS7	RxPLL ●	TxPLL ●	Lock to data ●

Lane Number	Cumulative Error Count	Data Rate		Bit Error Rate	
Lane 0	0		Gbps	NA	☐
Lane 1	0		Gbps	NA	☐
Lane 2	0		Gbps	NA	☐
Lane 3	0		Gbps	NA	☐

Figure 48 · Test Type Example

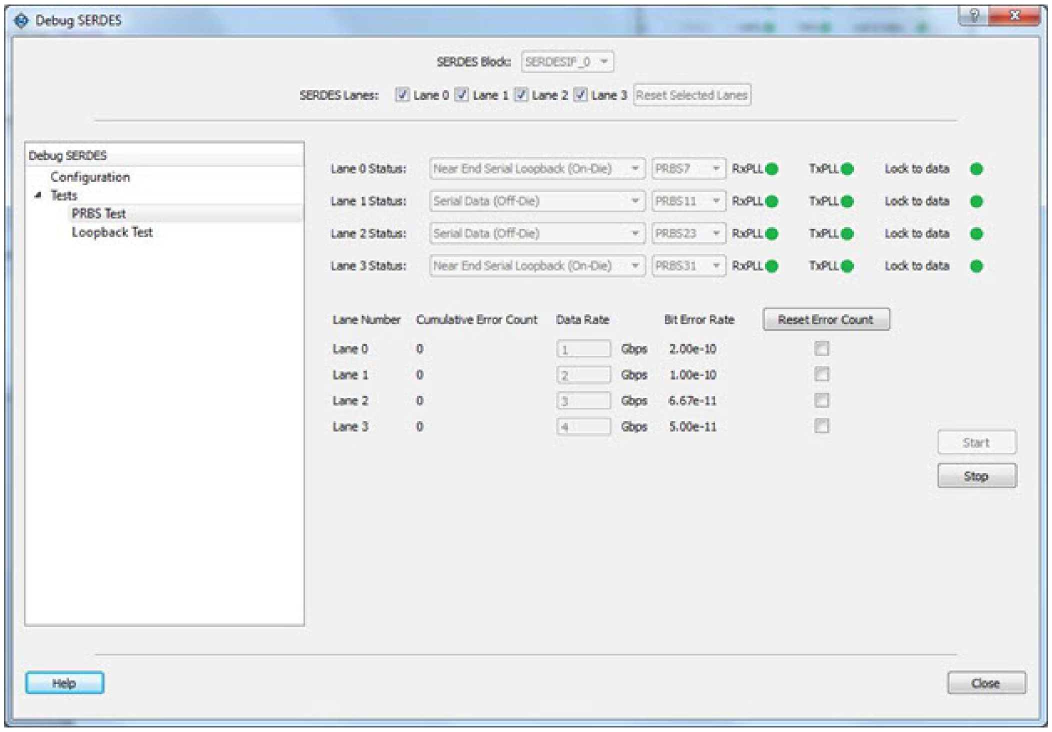
Pattern

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. If more than one lane is selected, the PRBS pattern can be selected per lane.

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. To reset the error count to zero, select the lane(s) and click **Reset**. By default, Cumulative Error Count = 0, the Data Rate text box is blank, and Bit Error Rate = NA.



SERDES Block: SERDESIF_0

SERDES Lanes: [✓ Lane 0] [✓ Lane 1] [✓ Lane 2] [✓ Lane 3]

Lane 0 Status:	Near End Serial Loopback (On-Die)	PRBS7	RxPLL ●	TxPLL ●	Lock to data ●
Lane 1 Status:	Serial Data (Off-Die)	PRBS11	RxPLL ●	TxPLL ●	Lock to data ●
Lane 2 Status:	Serial Data (Off-Die)	PRBS23	RxPLL ●	TxPLL ●	Lock to data ●
Lane 3 Status:	Near End Serial Loopback (On-Die)	PRBS31	RxPLL ●	TxPLL ●	Lock to data ●

Lane Number	Cumulative Error Count	Data Rate	Bit Error Rate	Reset Error Count
Lane 0	0	1 Gbps	2.00e-10	☐
Lane 1	0	2 Gbps	1.00e-10	☐
Lane 2	0	3 Gbps	6.67e-11	☐
Lane 3	0	4 Gbps	5.00e-11	☐

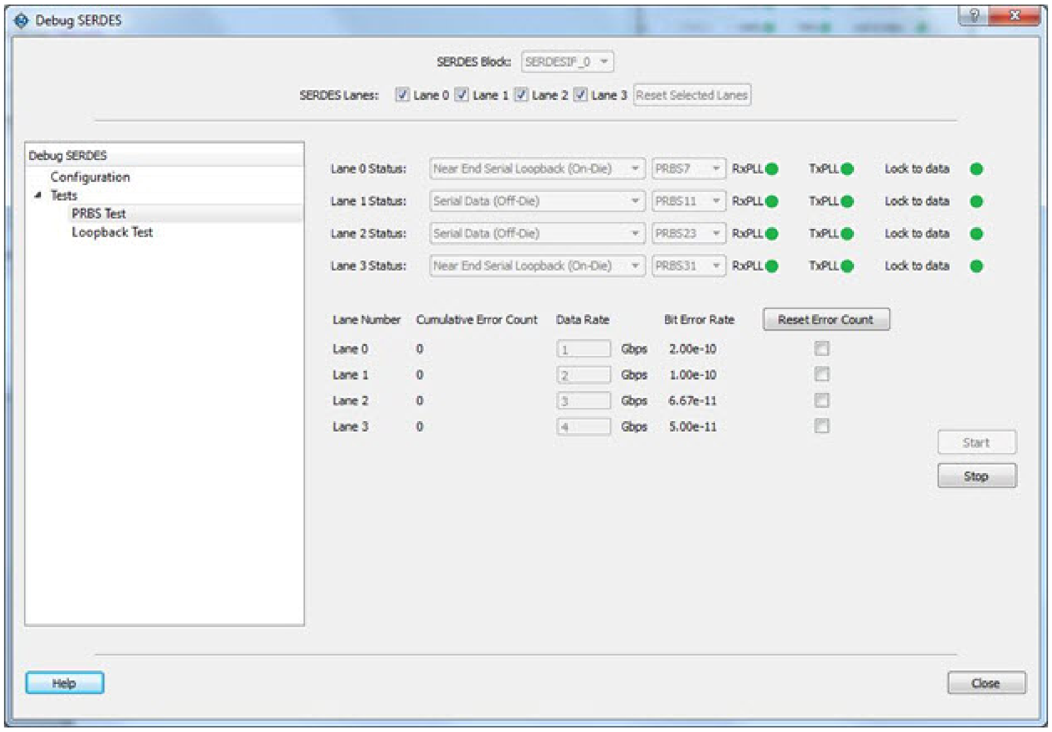
Figure 49 · Debug SERDES - PRBS Test

Note: If the design uses SERDES PCIe, PRBS7 is the only available option for PRBS tests.

Bit Error Rate

The Bit Error Rate is displayed per lane. If you did not specify a Data Rate, the Bit Error Rate displays the default NA. When the PRBS test is started, the Cumulative Error Count and Bit Error Rate are updated every second. You can select specific lanes and click **Reset Error Count** to clear the Cumulative Error Count and Bit Error Rate fields of the selected lanes.

In the example below, the Bit Error Rate is displayed for all lanes.

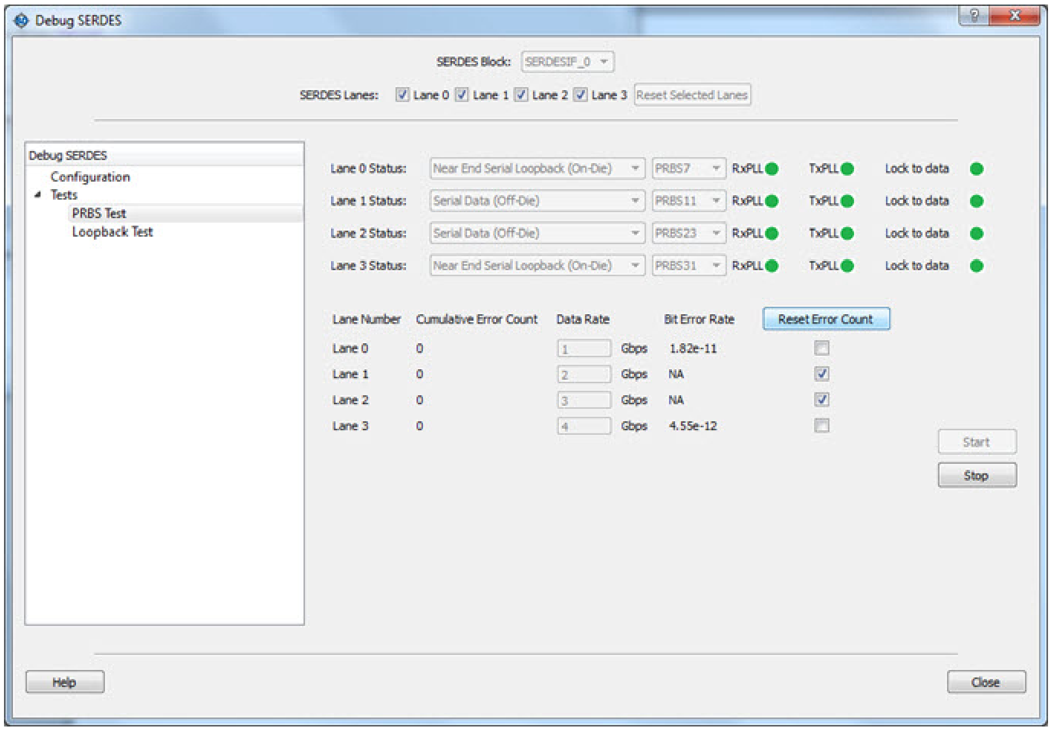


			RxPLL	TxPLL	Lock to data
Lane 0 Status:	Near End Serial Loopback (On-Die)	PRBS7	●	●	●
Lane 1 Status:	Serial Data (Off-Die)	PRBS11	●	●	●
Lane 2 Status:	Serial Data (Off-Die)	PRBS23	●	●	●
Lane 3 Status:	Near End Serial Loopback (On-Die)	PRBS31	●	●	●

Lane Number	Cumulative Error Count	Data Rate		Bit Error Rate	Reset Error Count
Lane 0	0	1	Gbps	2.00e-10	☐
Lane 1	0	2	Gbps	1.00e-10	☐
Lane 2	0	3	Gbps	6.67e-11	☐
Lane 3	0	4	Gbps	5.00e-11	☐

Figure 50 · Bit Error Rate Example - All Lanes

In the example below, Lane 1 and Lane 2 are selected and **Reset Error Count** is clicked.



			RxPLL	TxPLL	Lock to data
Lane 0 Status:	Near End Serial Loopback (On-Die)	PRBS7	●	●	●
Lane 1 Status:	Serial Data (Off-Die)	PRBS11	●	●	●
Lane 2 Status:	Serial Data (Off-Die)	PRBS23	●	●	●
Lane 3 Status:	Near End Serial Loopback (On-Die)	PRBS31	●	●	●

Lane Number	Cumulative Error Count	Data Rate		Bit Error Rate	Reset Error Count
Lane 0	0	1	Gbps	1.82e-11	☐
Lane 1	0	2	Gbps	NA	✓
Lane 2	0	3	Gbps	NA	✓
Lane 3	0	4	Gbps	4.55e-12	☐

Figure 51 · Reset Error Count Example

Notes:

The formula for calculating the BER is as follows:

BER = (#bit errors+1)/#bits sent

#bits sent = Elapsed time/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 NA.

When clicked on Stop:

- The BER resets to default.

When clicked on Reset:

- The BER resets 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 IGLOO2 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 device 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.

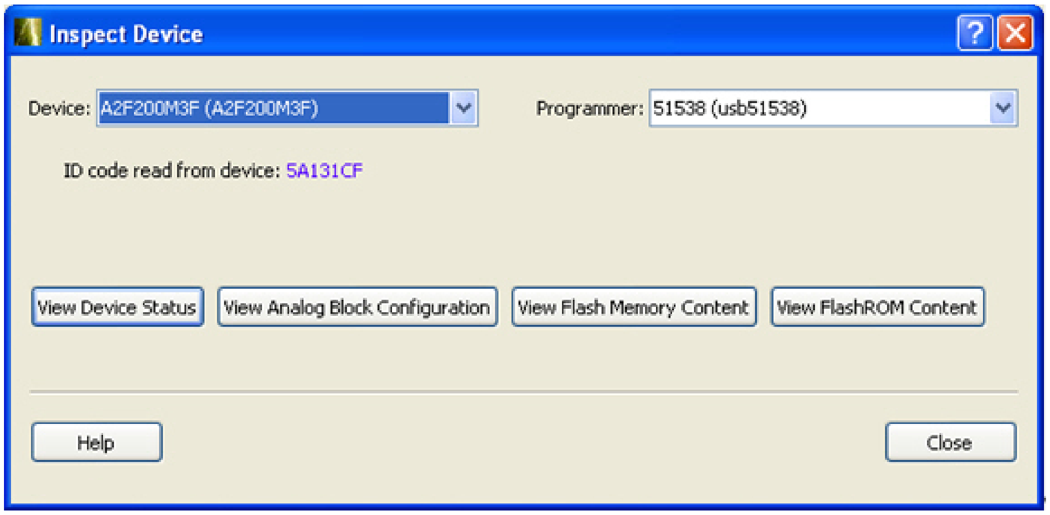
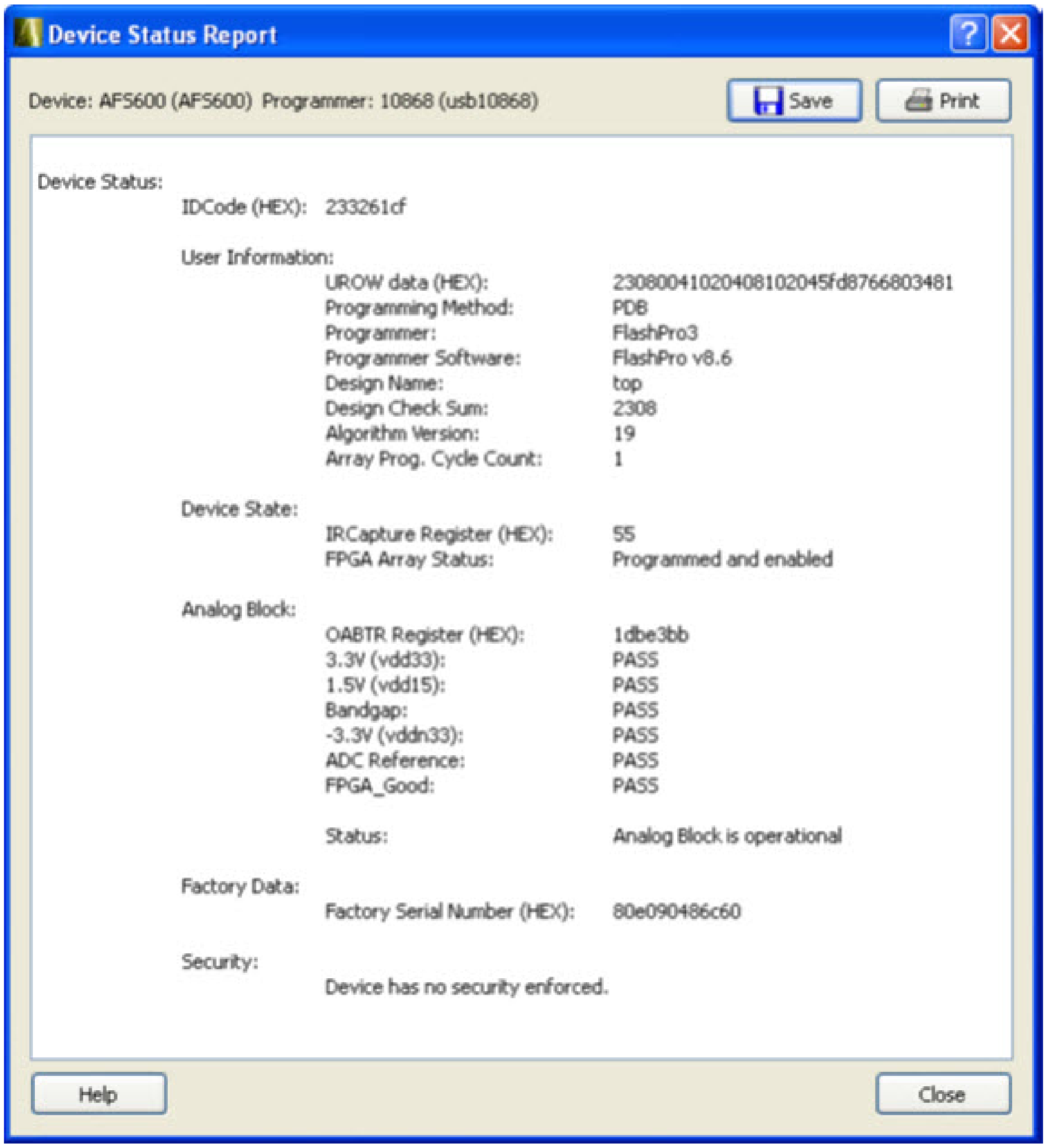

Inspect Device
Device: A2F200M3F (A2F200M3F)
Programmer: 51538 (usb51538)
ID code read from device: 5A131CF
View Device Status
View Analog Block Configuration
View Flash Memory Content
View FlashROM Content
Help
Close

Figure 52 · 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.



Device: AFS600 (AFS600) Programmer: 10868 (usb10868)

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

          User Information:
                    UROW data (HEX):               230800410204081020 45fd8766803481
                    Programming Method:            PDB
                    Programmer:                    FlashPro3
                    Programmer Software:           FlashPro v8.6
                    Design Name:                   top
                    Design Check Sum:              2308
                    Algorithm Version:             19
                    Array Prog. Cycle Count:       1

          Device State:
                    IRCapture Register (HEX):      55
                    FPGA Array Status:             Programmed and enabled

          Analog Block:
                    OABTR Register (HEX):          1dbe3bb
                    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

          Factory Data:
                    Factory Serial Number (HEX):   80e090486c60

          Security:
                    Device has no security enforced.
```

Figure 53 · 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

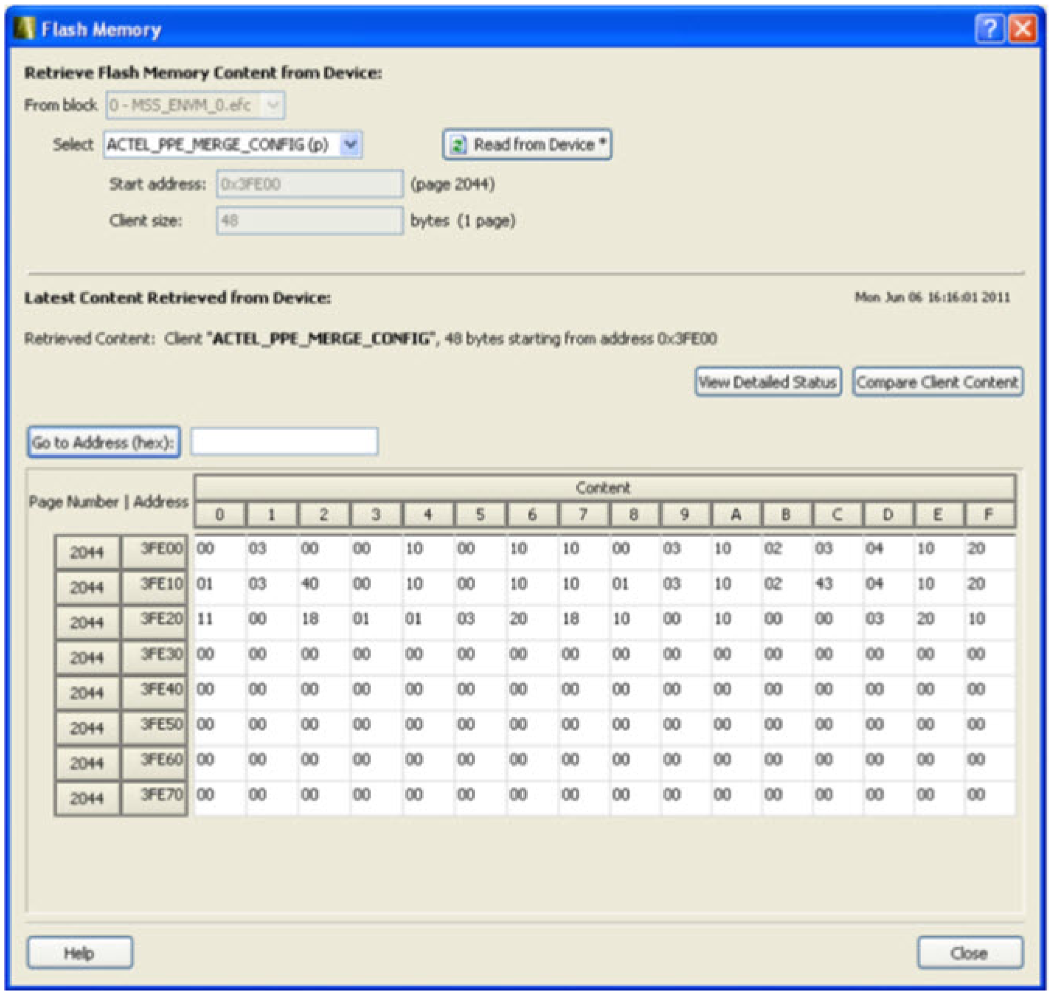
Fusion Devices: Choose your block from the **From block** drop-down list This action populates the Select drop-down list with the names of the clients in the selected block that is configured in the Flash Memory System Builder.

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.



Retrieve Flash Memory Content from Device:

From block: 0 - MSS_ENVM_0.efc

Select: ACTEL_PPE_MERGE_CONFIG (p)

Start address: 0x3FE00 (page 2044)

Client size: 48 bytes (1 page)

Latest Content Retrieved from Device: Mon Jun 06 16:16:01 2011

Retrieved Content: Client "ACTEL_PPE_MERGE_CONFIG", 48 bytes starting from address 0x3FE00

Page Number | Address		Content															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2044	3FE00	00	03	00	00	10	00	10	10	00	03	10	02	03	04	10	20
2044	3FE10	01	03	40	00	10	00	10	10	01	03	10	02	43	04	10	20
2044	3FE20	11	00	18	01	01	03	20	18	10	00	10	00	00	03	20	10
2044	3FE30	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2044	3FE40	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2044	3FE50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2044	3FE60	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
2044	3FE70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Figure 54 · Flash Memory Content Dialog Box for a SmartFusion Device (SmartDebug)

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

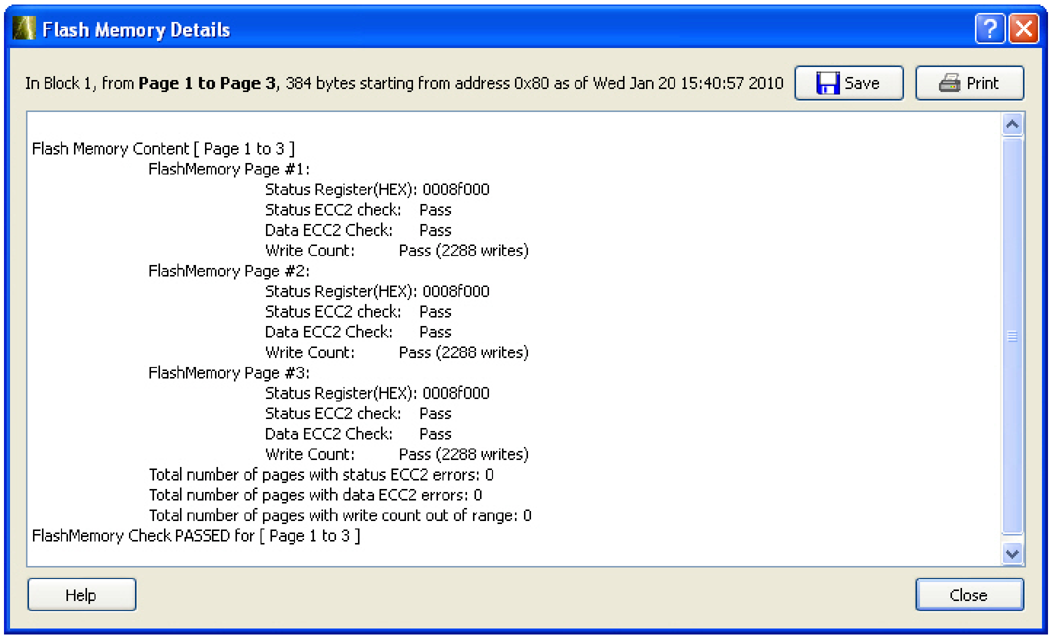
The retrieved data table displays the content of the selected client or the page range selection. Corrupted page content is displayed in red. Read-only page content, corresponding to clients defined with the Prevent read option in Flash Memory System Builder, is displayed with a gray background. If content 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.

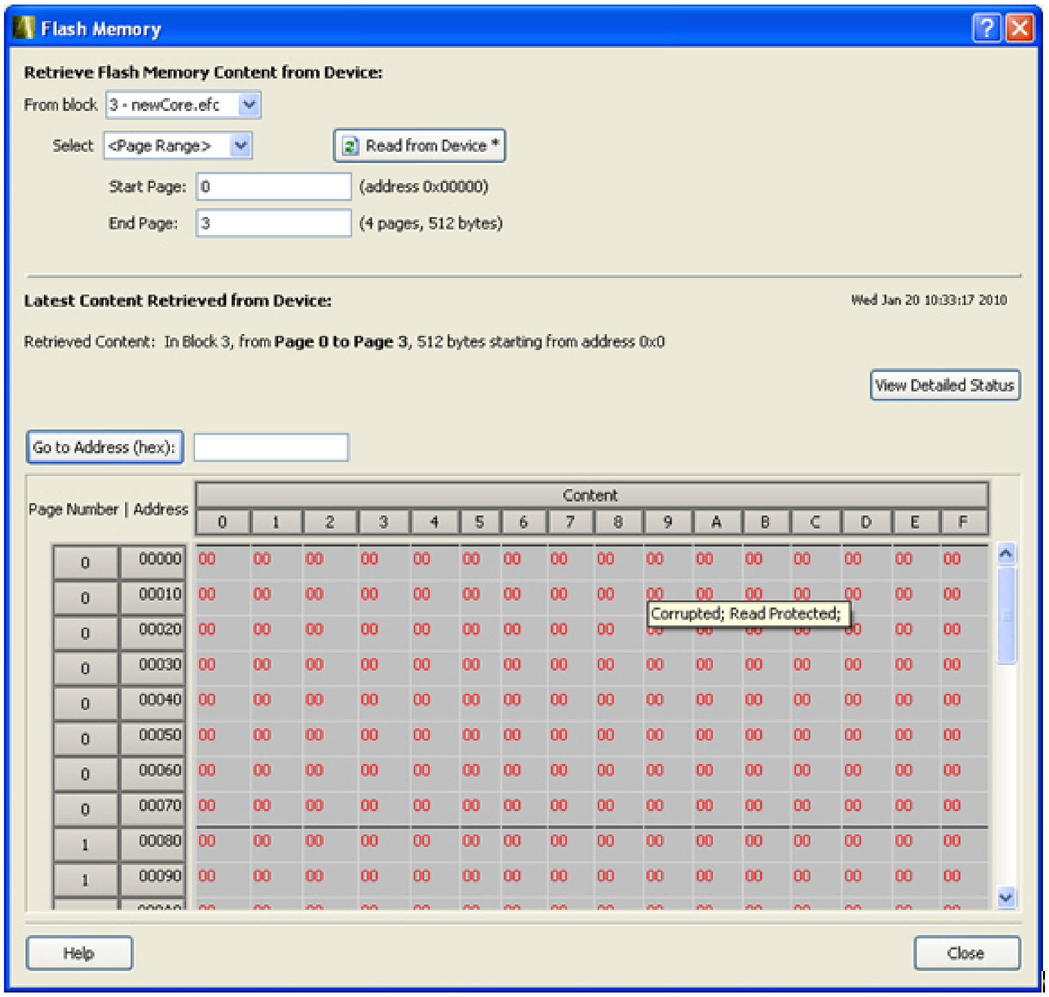
For example, if you want to view a report on pages 1-3, set the **Start Page** to 1, set the **End Page** to 3, and click **Read from Device**. Then click **View Detailed Status** The figure below is an example of the data for a specific page range.



In Block 1, from **Page 1 to Page 3**, 384 bytes starting from address 0x80 as of Wed Jan 20 15:40:57 2010

```
Flash Memory Content [ Page 1 to 3 ]
                FlashMemory Page #1:
                                Status Register(HEX): 0008f000
                                Status ECC2 check:    Pass
                                Data ECC2 Check:      Pass
                                Write Count:          Pass (2288 writes)
                FlashMemory Page #2:
                                Status Register(HEX): 0008f000
                                Status ECC2 check:    Pass
                                Data ECC2 Check:      Pass
                                Write Count:          Pass (2288 writes)
                FlashMemory Page #3:
                                Status Register(HEX): 0008f000
                                Status ECC2 check:    Pass
                                Data ECC2 Check:      Pass
                                Write Count:          Pass (2288 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 1 to 3 ]
```

Figure 55 · Flash Memory Details Dialog Box (SmartDebug)


Flash Memory

Retrieve Flash Memory Content from Device:

From block: 3 - newCore.efc

Select: <Page Range> Read from Device *

Start Page: 0 (address 0x00000)

End Page: 3 (4 pages, 512 bytes)

Latest Content Retrieved from Device: Wed Jan 20 10:33:17 2010

Retrieved Content: In Block 3, from **Page 0 to Page 3**, 512 bytes starting from address 0x0

View Detailed Status

Go to Address (hex):

Page Number	Address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	00070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Corrupted; Read Protected;

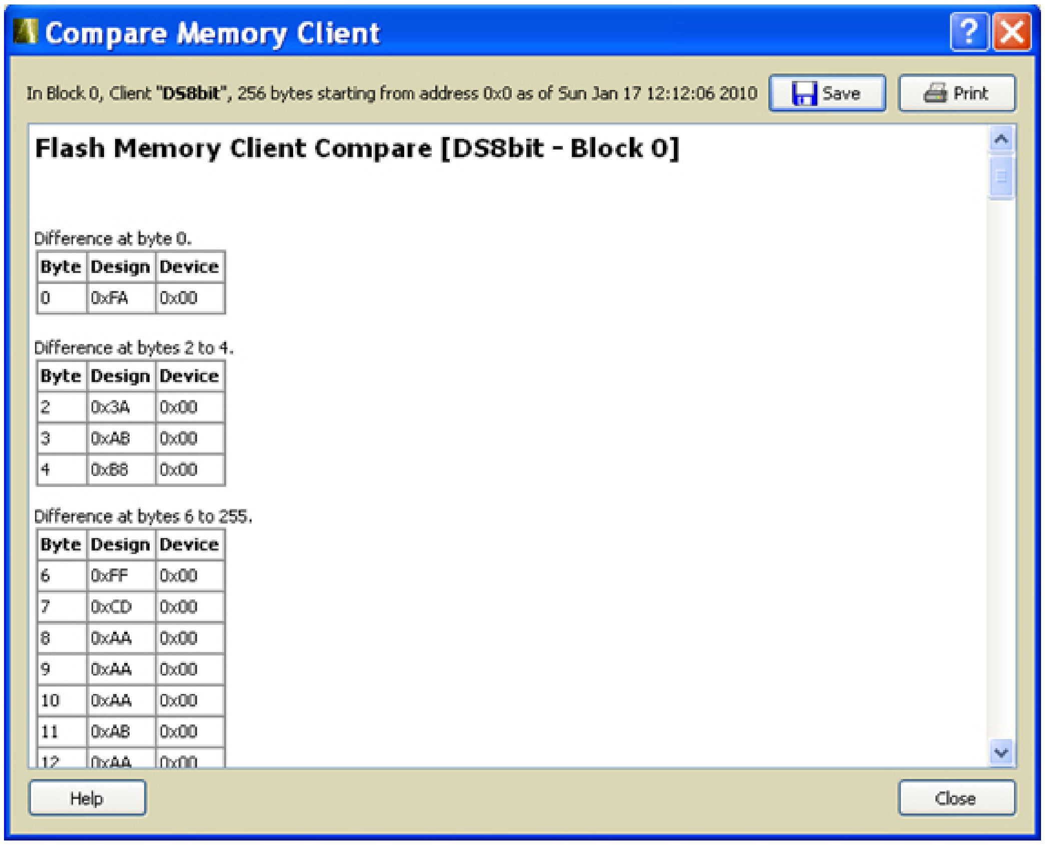
Help Close

Figure 56 · 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.



In Block 0, Client "DS8bit", 256 bytes starting from address 0x0 as of Sun Jan 17 12:12:06 2010

Flash Memory Client Compare [DS8bit - Block 0]

Difference at byte 0.

Byte	Design	Device
0	0xFA	0x00

Difference at bytes 2 to 4.

Byte	Design	Device
2	0x3A	0x00
3	0xAB	0x00
4	0xB8	0x00

Difference at bytes 6 to 255.

Byte	Design	Device
6	0xFF	0x00
7	0xCD	0x00
8	0xAA	0x00
9	0xAA	0x00
10	0xAA	0x00
11	0xAB	0x00
12	0xAA	0x00

Figure 57 · Compare Memory Client Dialog Box

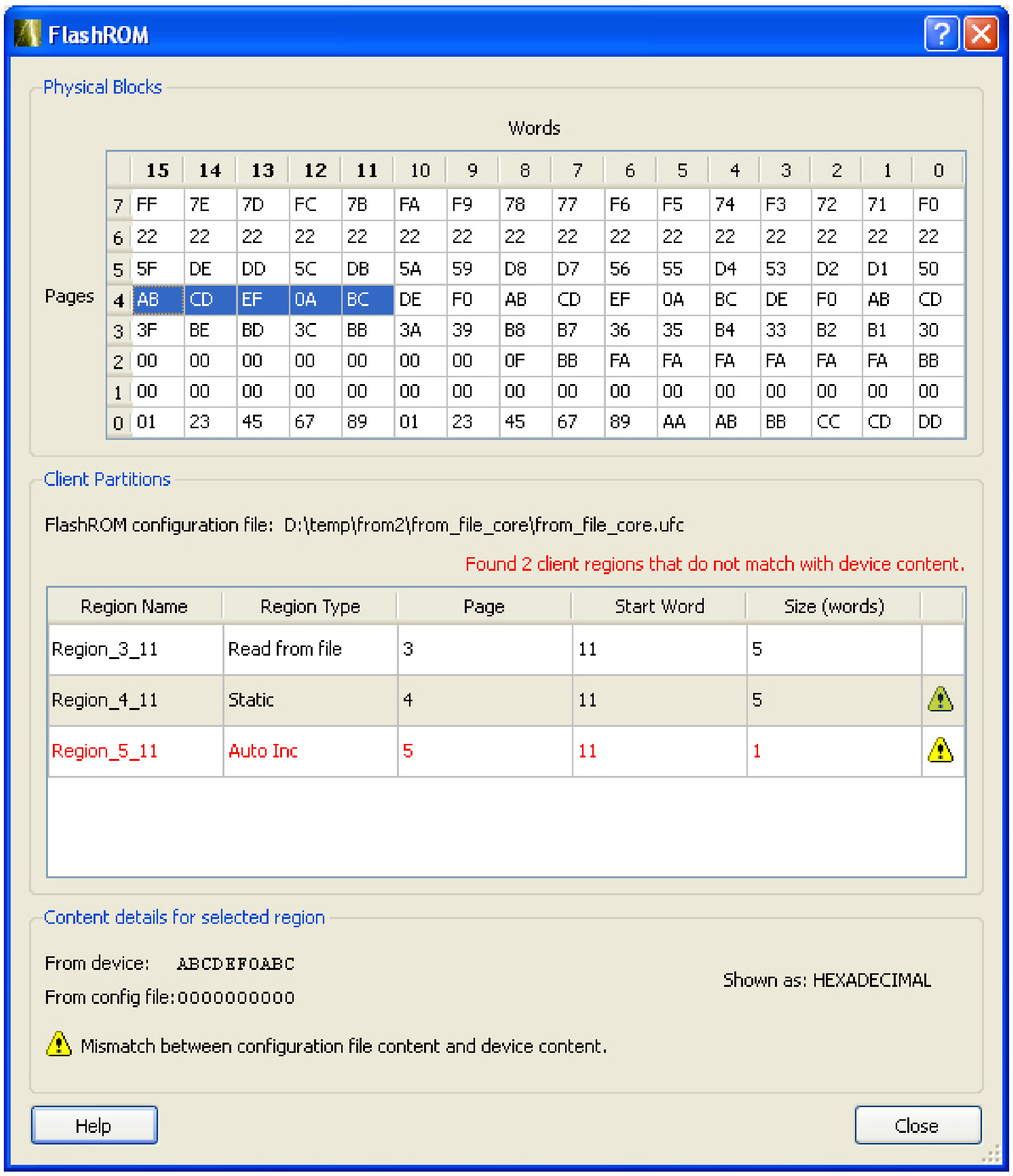
FlashROM Content Dialog Box (Fusion and SmartFusion Only)

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.



FlashROM

Physical Blocks

Words

Pages

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	FF	7E	7D	FC	7B	FA	F9	78	77	F6	F5	74	F3	72	71	F0
6	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22	22
5	5F	DE	DD	5C	DB	5A	59	D8	D7	56	55	D4	53	D2	D1	50
4	AB	CD	EF	0A	BC	DE	F0	AB	CD	EF	0A	BC	DE	F0	AB	CD
3	3F	BE	BD	3C	BB	3A	39	B8	B7	36	35	B4	33	B2	B1	30
2	00	00	00	00	00	00	00	0F	BB	FA	FA	FA	FA	FA	FA	BB
1	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0	01	23	45	67	89	01	23	45	67	89	AA	AB	BB	CC	CD	DD

Client Partitions

FlashROM configuration file: D:\temp\from2\from_file_core\from_file_core.ufc

Found 2 client regions that do not match with device content.

Region Name	Region Type	Page	Start Word	Size (words)	
Region_3_11	Read from file	3	11	5	
Region_4_11	Static	4	11	5	⚠
Region_5_11	Auto Inc	5	11	1	⚠

Content details for selected region

From device: ABCDEF0ABC

From config file: 0000000000

Shown as: HEXADECIMAL

⚠ Mismatch between configuration file content and device content.

Figure 58 · 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:

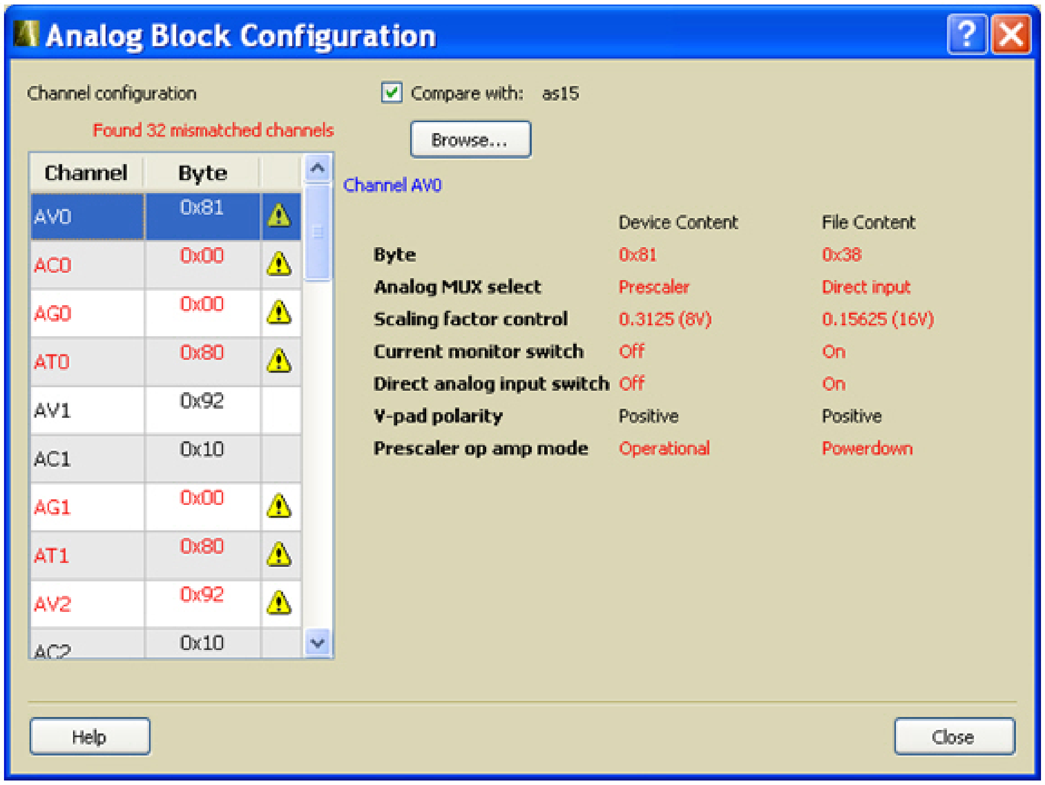
- Gain select
- Channel state
- Direct Input state
- Current Monitor state
- Current monitor strobe state
- Comparator state
- Hysteresis select
- Analog MUX select
- DAC input select
- Temperature monitor state
- Temperature monitor strobe state
- Vref switch state

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.



Channel	Byte	
AV0	0x81	⚠
AC0	0x00	⚠
AG0	0x00	⚠
AT0	0x80	⚠
AV1	0x92	
AC1	0x10	
AG1	0x00	⚠
AT1	0x80	⚠
AV2	0x92	⚠
AC2	0x10	

Channel AV0

	Device Content	File Content
Byte	0x81	0x38
Analog MUX select	Prescaler	Direct input
Scaling factor control	0.3125 (8V)	0.15625 (16V)
Current monitor switch	Off	On
Direct analog input switch	Off	On
V-pad polarity	Positive	Positive
Prescaler op amp mode	Operational	Powerdown

Figure 59 · 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_probe_insertion_point	Adds probe points to be connected to user-specified I/Os for probe insertion flow.
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.
program_probe_insertion	Runs the probe insertion flow on the selected nets.
remove_probe_insertion_point	Deletes an added probe from the probe insertion UI.
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

Command	Action
DDR/MDDR	
	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.
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	
event_counter	Runs on signals that are assigned to channel A on the live probe, and displays the total events.
export_smart_debug_data	Exports debug data for the SmartDebug application.
fhb_control	Provides FPGA Hardware Breakpoint (FHB) feature capability for SmartDebug.
frequency_monitor	Calculates the frequency of a signal that is assigned to live probe A.

Command	Action
DDR/MDDR	
get_programmer_info	Lists the IDs of all FlashPRO programmers connected to the machine.

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

Note: 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	Embedded Flash Memory (NVM)

Command	Action	Type
	all pages to zeros.	
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 for debugging (if you have more than one).	

add_probe_insertion_point

This Tcl command adds probe points to be connected to user-specified I/Os for probe insertion flow.

```
add_probe_insertion_point –net net_name -driver driver -pin package_pin_name -port port name
```

Arguments

`-net net_name`

Name of the net used for probe insertion.

`-driver driver`

Driver of the net.

`-pin package_pin_name`

Package pin name (i.e. I/O to which the net will be routed during probe insertion).

`-port port_name`

User-specified name for the probe insertion point.

Supported Families

SmartFusion2, IGLOO2, RTG4

Example

```
add_probe_insertion_point -net {count_out_c[0]} -driver {Counter_8bit_0_count_out[0]:Q} -
pin {H5} -port {Probe_Insert0}
```

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

Value	Description
	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 starpage 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}
```

Note: If an absolute path is not entered, the log file is saved in the directory in which the Tcl script was executed in SmartDebug.

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}
```

Note: If an absolute path is not entered, the log file is saved in the directory in which the Tcl script was executed in SmartDebug.

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
```

event_counter

The event_counter Tcl command runs on signals that are assigned to channel A on the live probe, and displays the total events. It can be run before or after setting the live probe signal to channel A. The user specifies the duration to run the event_counter command.

```
event_counter -run -stop –after duration_in_seconds
```

Arguments

`-run`

Run event_counter.

`-stop`

Stop event_counter.

`-after` *`duration_in_seconds`*

Duration to stop event_counter. Specified by the user. This argument is required when `–stop` is specified.

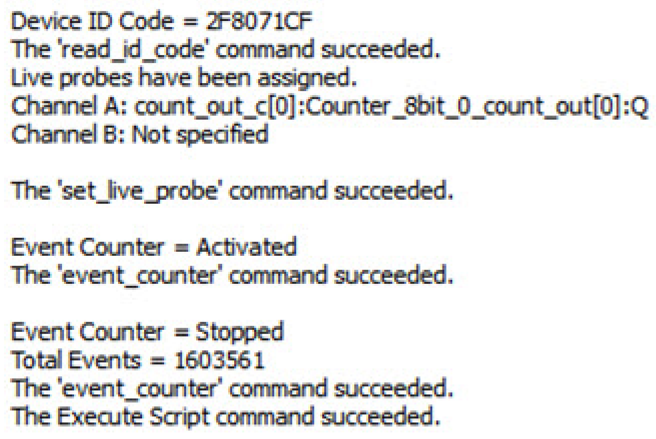
Supported Families

SmartFusion2, IGLOO2

Example

```
set_live_probe -probeA {count_out_c[0]:Counter_8bit_0_count_out[0]:Q} -probeB {}
event_counter -run
event_counter –stop –after 10
```

Output



```
Device ID Code = 2F8071CF
The 'read_id_code' command succeeded.
Live probes have been assigned.
Channel A: count_out_c[0]:Counter_8bit_0_count_out[0]:Q
Channel B: Not specified

The 'set_live_probe' command succeeded.

Event Counter = Activated
The 'event_counter' command succeeded.

Event Counter = Stopped
Total Events = 1603561
The 'event_counter' command succeeded.
The Execute Script 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.
  - You must specify the bitstream components you want to include in the generated bitstream file or all available components will be included.
  - 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
```

fhb_control

This Tcl command provides FPGA Hardware Breakpoint (FHB) feature capability for SmartDebug.

```
fhb_control
   -halt -clock_domain clkDomName(s)/all
   -run -clock_domain clkDomName(s)
   -step number_of_steps -clock_domain clkDomName(s)
   -reset -clock_domain clkDomName(s)
   -arm_trigger –trigger_signal liveProbePoint –trigger_edge_select rising -delay value -
clock_domain clkDomName(s)
   -disarm_trigger -clock_domain clkDomName(s)/all
   –capture_waveform number_of_steps -vcd_file target_file_name
   –clock_domain_status -clock_domain clkDomName(s)/all
```

Arguments

`-halt`

Specifies to halt the clock.

`-clock_domain` *`clkDomName(s)/all`*

Specifies clock domain names to halt. Can be single or multiple clock domains, halted in order specified by user.

`-run`

Specifies to run the clock.

`-clock_domain` *`clkDomName(s)`*

Specifies clock domain names to run. Can be single or multiple clock domains, releasing the user clock based on order specified.

`-step` *`number_of_steps`*

Specifies to step the clock “number_of_steps” times. Minimum value is 1.

`-clock_domain` *`clkDomName(s)`*

Specifies clock domain names to step. Can be single or multiple clock domains.

`-reset`

Specifies to reset FHB configuration for the specified clock domain.

`-clock_domain` *`clkDomName(s)`*

Specifies clock domain names to reset. Can be single or multiple clock domains.

`-arm_trigger`

Specifies to arm FHB configuration for the specified clock domain.

`-trigger_signal` *`liveProbePoint`*

Set the trigger signal to arm the FHBs.

`-trigger_edge_select` *`rising`*

Specifies the trigger signal edge to arm the FHBs. FHBs will be armed on rising edge of trigger signal.

`-delay` *`value`*

`-clock_domain` *`clkDomName(s)`*

Specifies clock domain names to be armed by the trigger signal. Can be single or multiple clock domains.

`-disarm_trigger`

Specifies to disarm FHB configuration for the specified clock domain.

`-clock_domain` *`clkDomName(s)`*

Specifies clock domain names to be reset by the trigger signal. Can be single or multiple clock domains.

`-capture_waveform` *`number_of_steps`*

Specifies to capture waveform of all the added signals to active probes in the specified clock domain for number_of_steps.

`- vcd_file` *`target_file_name`*

Target file to save the data and see the waveform.

`-clock_domain_status` *`clkDomName(s)/all`*

Specifies to read and display status of specified clock domain(s). Can be single or multiple clock domains.

Supported Families

SmartFusion2, IGLOO2

Examples

```
fhb_control –halt –clock_domain {“FCCC_0/GL0_INST “ “FCCC_0/GL1_INST” }
fhb_control –run –clock_domain {“FCCC_0/GL0_INST “ “FCCC_0/GL1_INST” }
fhb_control –step –clock_domain {“FCCC_0/GL0_INST “ “FCCC_0/GL1_INST” }
fhb_control –reset –clock_domain {“FCCC_0/GL0_INST “ “FCCC_0/GL1_INST” }
fhb_control –arm_trigger –trigger_signal {q_0_c[14]:count_1_q[14]:Q}
–trigger_edge_select {rising} – delay 0 – clock_domain {"FCCC_0/GL0_INST"}
fhb_control –disarm_trigger –trigger_signal {q_0_c[14]:count_1_q[14]:Q}
–trigger_edge_select {rising} – delay 0 – clock_domain {"FCCC_0/GL0_INST"}
fhb_control –capture_waveform {10} –vcd_file {D:/wvf_location/waveform.vcd}
fhb_control – clock_domain_status – clock_domain { "FCCC_0/GL0_INST" "FCCC_0/GL1_INST"
"FCCC_0/GL2_INST" }
```

frequency_monitor

The frequency_monitor Tcl command calculates the frequency of a signal that is assigned to live probe A.

```
run_frequency_monitor -signal signal_name –time duration
```

Arguments

`-signal` *`signal_name`*

Specifies the signal name.

`–time` *`duration`*

Specifies the duration to run the command. The value can be 0.1, 1, 5, 8, or 10.

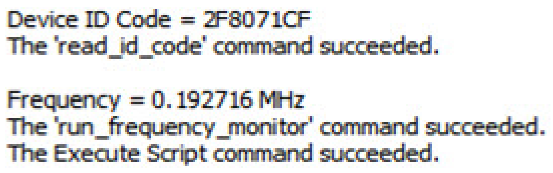
Supported Families

SmartFusion2, IGLOO2

Example

```
run_frequency_monitor -signal {count_out_c[7]:Counter_8bit_0_count_out[7]:Q} -time {5}
```

Output



```
Device ID Code = 2F8071CF
The 'read_id_code' command succeeded.

Frequency = 0.192716 MHz
The 'run_frequency_monitor' command succeeded.
The Execute Script command succeeded.
```

get_programmer_info

This Tcl command lists the IDs of all FlashPRO programmers connected to the machine.

```
get_programmer_info
```

This command takes no arguments.

Supported Families

SmartFusion2, IGLOO2, RTG4

Example

```
set a [get_programmer_info]
```

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, 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, 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, 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, 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
```

program_probe_insertion

This Tcl command runs the probe insertion flow on the selected nets.

```
program_probe_insertion
```

Supported Families

SmartFusion2, IGLOO2, RTG4

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] [-group_name
bus_name|group_name] [-value_type b|h] [-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.

`-group_name bus_name | group_name`

Instead of all probes, reads only the specified buses or groups specified here.

`-value_type b | h`

Optional parameter, used when the read value is stored into a variable as a string.

b = binary

h = hex

`-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 -group_name {bus1}
read_active_probe -group_name {group1}
```

To save into variable:

`set a [read_active_probe -group_name {bus_name} -value_type h]` *(save read data in hex string If read values are stored into a variable without specifying value_type parameter, it saves values as a binary string by default)*

Example

`set a [read_active_probe ]` *(sets variable a as binary string of read values after read_active_probe command)*

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)

Value	Description
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.

Physical block

```
read_lsram [-deviceName device_name] -name block_name [–file filename]
```

Arguments

`-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).

`-name block_name`

Specifies the name for the target block.

`-file filename`

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

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] –name sram_block1 [–file sram_block_output]
```

Logical block

```
read_lsram -logicalBlockName block_name -port port_name [–file filename]
```

Arguments

`-logicalBlockName block_name`

Specifies the name for the user defined memory block.

`-port port_name`

Specifies the port for the memory block selected. Can be either Port A or Port B.

`-file filename`

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

Example

```
read_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A}
```

read_usram (SmartFusion2, IGLOO2, and RTG4)

Tcl command; reads a uSRAM block from the device.

Physical block

```
read_usram [-deviceName device_name] –name block_name [–file filename]
```

Arguments

`-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).

`-name block_name`

Specifies the name for the target block.

`-file filename`

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

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] –name usram_block2 [–file sram_block_output]
```

Logical block

```
read_usram -logicalBlockName block_name -port port_name [–file filename]
```

Arguments

`-logicalBlockName block_name`

Specifies the name for the user defined memory block.

`-port port_name`

Specifies the port for the memory block selected. Can be either Port A or Port B.

`-file filename`

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

Example

```
read_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A}
```

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; removes the specified probe points from the group. That is, the removed probe points won’t be associated with any probe group.

Note: Probes cannot be removed from the bus.

```
remove_from_probe_group -name probe_name
```

Arguments

`-name probe_name`

Specifies one or more probe points to remove from the probe group.

Supported Families

SmartFusion2, IGLOO2, and RTG4

Example

The following command removes 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
```

remove_probe_insertion_point

This Tcl command deletes an added probe from the probe insertion UI.

```
remove_probe_insertion_point –net net_name -driver driver
```

Arguments

`-net net_name`

Name of the existing net which is added using the `add_probe_insertion_point` command.

`-driver driver`

Driver of the net.

Supported Families

SmartFusion2, IGLOO2, RTG4

Example

```
remove_probe_insertion_point -net {count_out_c[0]} -driver
{Counter_8bit_0_count_out[0]: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:

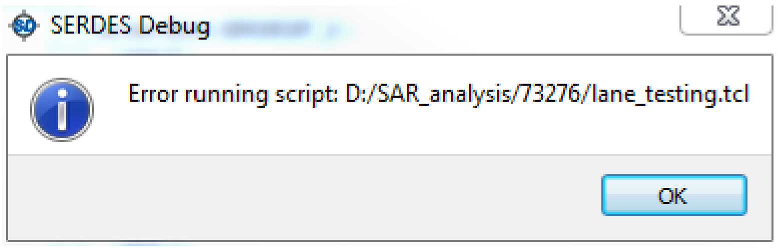
```
Error: Phy Reset: Serdes block number should be one of the following: 0
Error: The command 'serdes_lane_reset' failed.
Error: Failure when executing Tcl script. [Line 26]
Error: The Execute Script command failed.
```

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:



Error running script: D:/SAR_analysis/73276/lane_testing.tcl

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 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, 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, 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_lsram (SmartFusion2, IGLOO2, and RTG4)

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

Physical block

```
write_lsram [-deviceName device_name] –name block_name] –offset offset_value –value value
```

Arguments

`-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).

`-name block_name`

Specifies the name for the target block.

`-offset offset_value`

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

`-value value`

Nine-bit value to be written to the target location.

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_lsram [-deviceName sf2] –name sram_block1 -offset 16 -value 0x1A
```

Logical block

```
write_lsram -logicalBlockName block_name -port port_name -offset offset_value -logicalValue
hexadecimal_value
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name for the user defined memory block.

`-port` *`port_name`*

Specifies the port for the memory block selected. Can be either Port A or Port B.

`-offset` *`offset_value`*

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

`-logicalValue` *`hexadecimal_value`*

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_lsram -logicalBlockName {Fabric_Logic_0/U2/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

write_usram (SmartFusion2, IGLOO2, and RTG4)

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

Physical block

```
write_usram [-deviceName device_name] –name block_name] –offset offset_value –value value
```

Arguments

`-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).

`-name` *`block_name`*

Specifies the name for the target block.

`-offset` *`offset_value`*

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

`-value` *`value`*

Nine-bit value to be written.

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] –name usram_block2 -offset 16 -value 0x1A
```

Logical block

```
write_usram -logicalBlockName block_name -port port_name -offset offset_value -logicalValue
hexadecimal_value
```

Arguments

`-logicalBlockName` *`block_name`*

Specifies the name for the user defined memory block.

`-port` *`port_name`*

Specifies the port for the memory block selected. Can be either Port A or Port B.

`-offset` *`offset_value`*

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

`-logicalValue` *`hexadecimal_value`*

Specifies the hexadecimal value to be written to the memory block. Size of the value is equal to the width of the output port selected.

Example

```
write_usram -logicalBlockName {Fabric_Logic_0/U3/F_0_F0_U1} -port {Port A} -offset 1 -
logicalValue {00FFF}
```

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 corrupted 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: 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 .

   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`

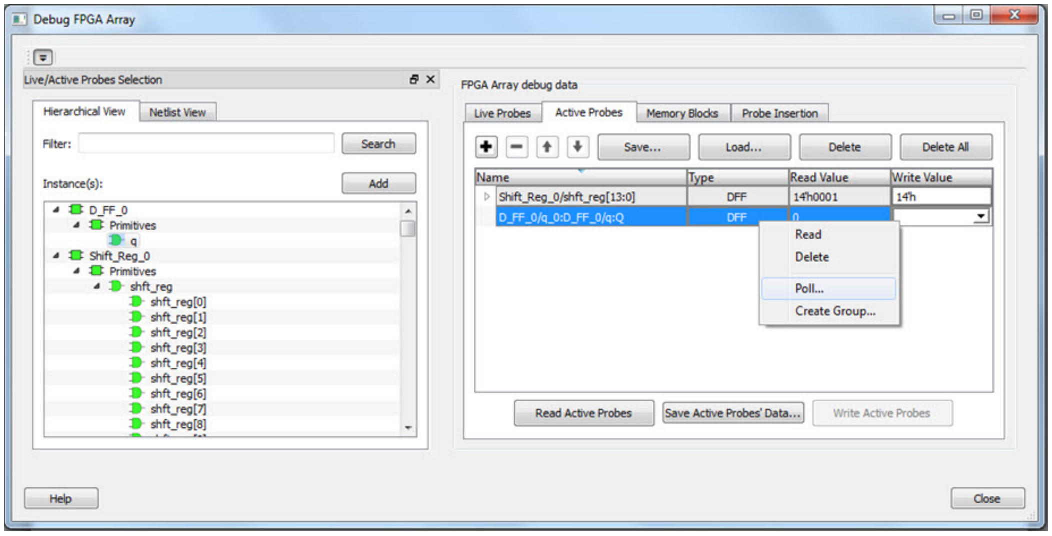
sample_analog_channel

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

How do I monitor a static or pseudo-static signal?

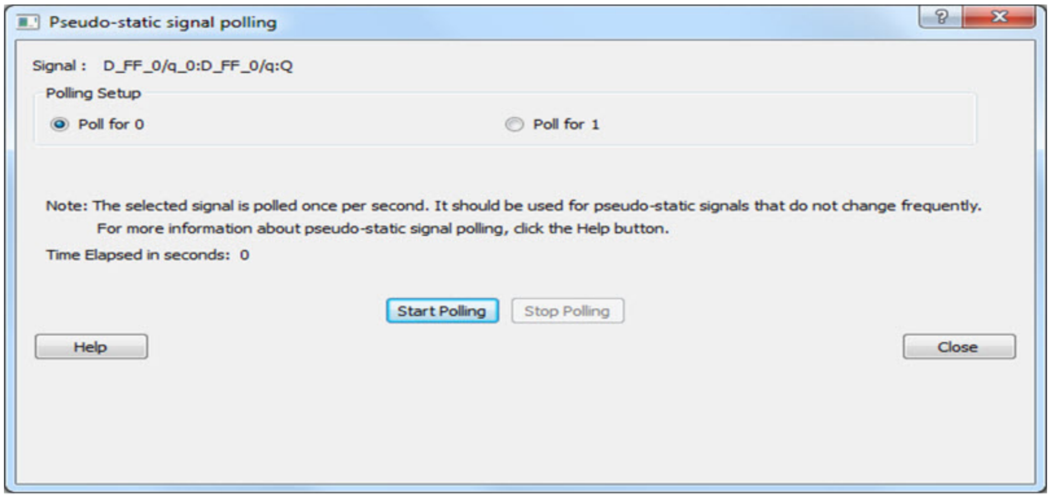
To monitor a static or pseudo-static signal:

1. Add the signal to the **Active Probes** tab.
2. Select the signal in the **Active Probes** tab, right-click, and choose **Poll...**.



Name	Type	Read Value	Write Value
Shift_Reg_0/shft_reg[13:0]	DFF	14'h0001	14'h
D_FF_0/q_0:D_FF_0/q:Q	DFF	0	

3. In the Pseudo-static Signal Polling dialog box, choose a value in Polling Setup and click **Start Polling**.



Signal : D_FF_0/q_0:D_FF_0/q:Q

Polling Setup: [✓ Poll for 0] Poll for 1

Note: The selected signal is polled once per second. It should be used for pseudo-static signals that do not change frequently.
For more information about pseudo-static signal polling, click the Help button.

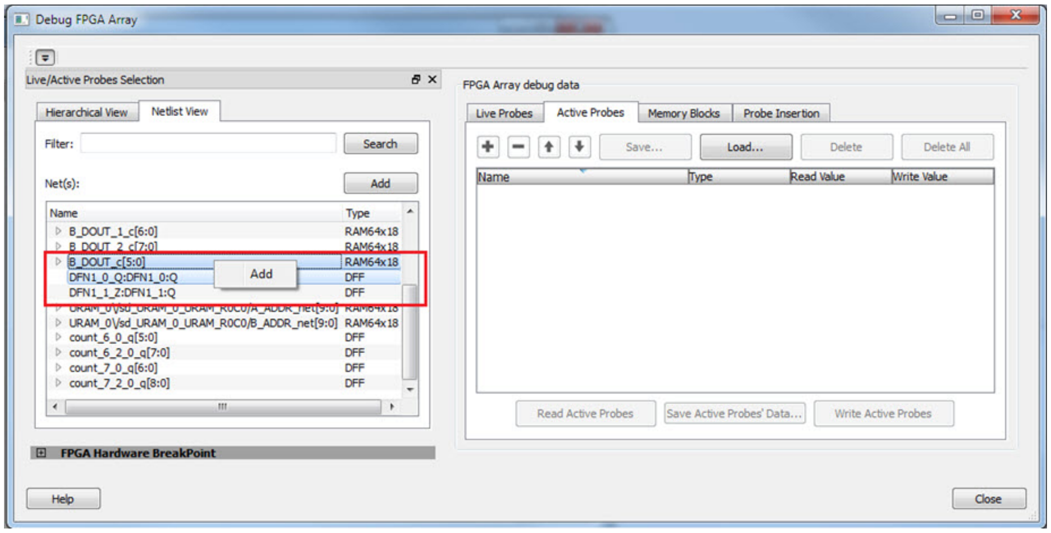
Time Elapsed in seconds: 0

For more information, refer to the SmartDebug for Libero SoC v11.8 User Guide.

How do I force a signal to a new value?

To force a signal to a new value:

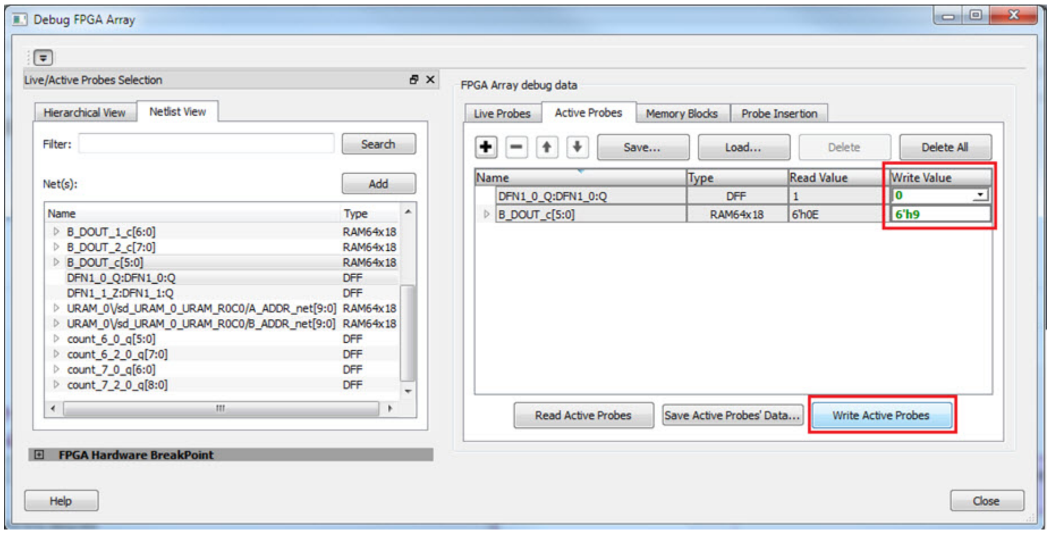
1. In the SmartDebug window, click **Debug FPGA Array**.
2. Click the **Active Probes** tab.
3. Select the signal from the selection panel and add it to Active Probes tab.



Name	Type
B_DOUT_1_c[6:0]	RAM64x18
B_DOUT_2_c[7:0]	RAM64x18
B_DOUT_c[5:0]	RAM64x18
DFN1_0_Q:DFN1_0:Q	DFF
DFN1_1_Z:DFN1_1:Q	DFF
URAM_0\/sd_URAM_0_URAM_R0C0/A_ADDR_net[9:0]	RAM64x18
URAM_0\/sd_URAM_0_URAM_R0C0/B_ADDR_net[9:0]	RAM64x18
count_6_0_q[5:0]	DFF
count_6_2_0_q[7:0]	DFF
count_7_0_q[6:0]	DFF
count_7_2_0_q[8:0]	DFF

Name	Type	Read Value	Write Value

1. Click **Read Active Probe** to read the value.
2. In the Write Value column, enter the value to write to the signal and then click **Write Active Probes**.



Name	Type
B_DOUT_1_c[6:0]	RAM64x18
B_DOUT_2_c[7:0]	RAM64x18
B_DOUT_c[5:0]	RAM64x18
DFN1_0_Q:DFN1_0:Q	DFF
DFN1_1_Z:DFN1_1:Q	DFF
URAM_0\/sd_URAM_0_URAM_R0C0/A_ADDR_net[9:0]	RAM64x18
URAM_0\/sd_URAM_0_URAM_R0C0/B_ADDR_net[9:0]	RAM64x18
count_6_0_q[5:0]	DFF
count_6_2_0_q[7:0]	DFF
count_7_0_q[6:0]	DFF
count_7_2_0_q[8:0]	DFF

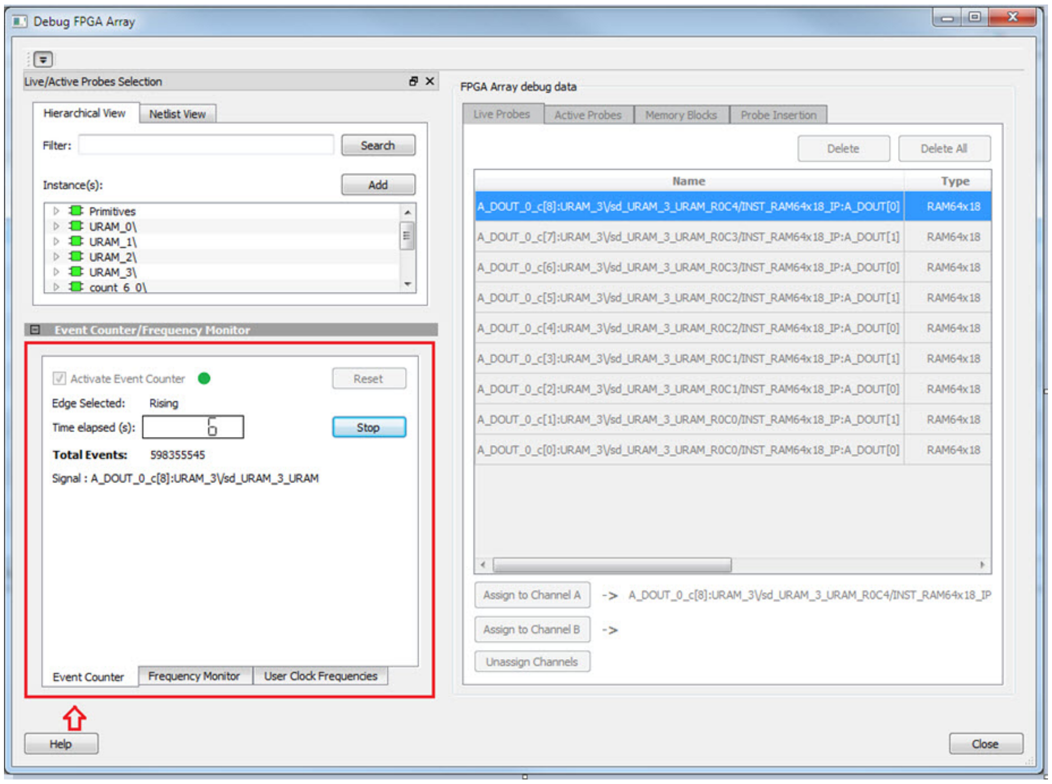
Name	Type	Read Value	Write Value
DFN1_0_Q:DFN1_0:Q	DFF	1	0
B_DOUT_c[5:0]	RAM64x18	6'h0E	6'h9

How do I count the transitions on a signal?

If FHB IP is auto-instantiated in the design, you can use the Event Counter in the **Live Probes** tab to count the transitions on a signal.

To count the transitions on a signal:

1. Assign the desired signal to Live Probe Channel A.
2. Click the **Event Counter** tab and check the Activate Event Counter checkbox.



Activate Event Counter ●

Edge Selected: Rising

Time elapsed (s): 6

Total Events: 598355545

Signal : A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP

Assign to Channel B ->

See Also

"Event Counter" **Error! Bookmark not defined.**

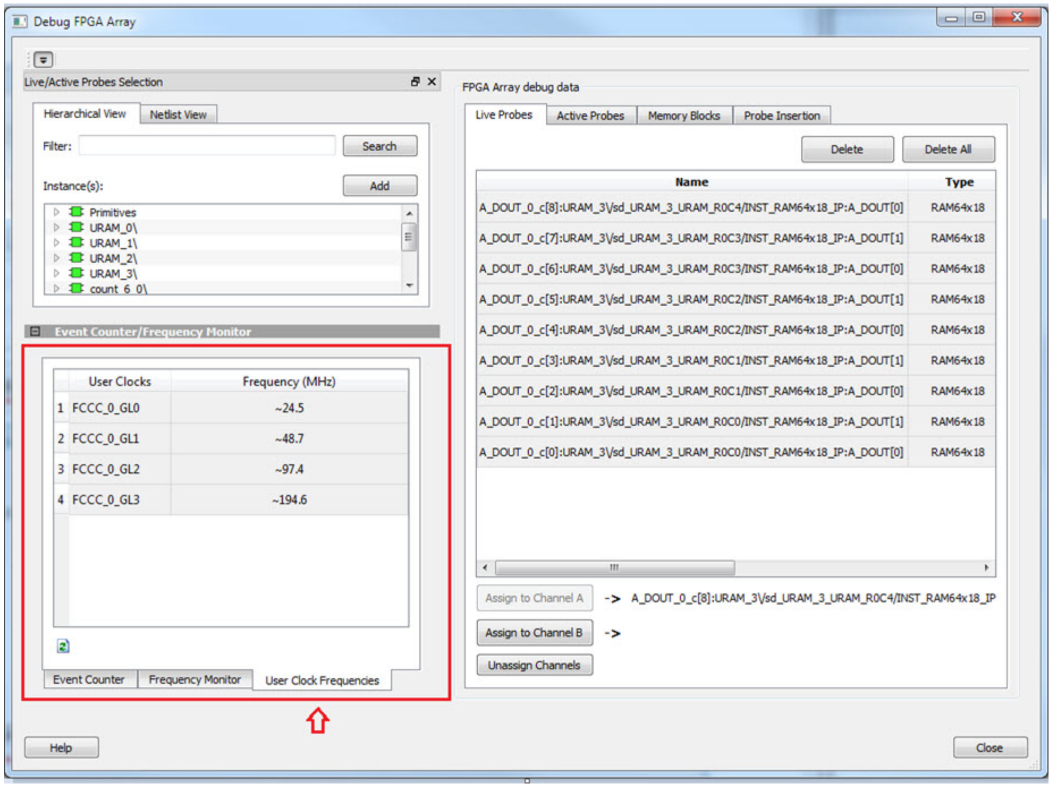
SmartDebug for Libero SoC v11.8 User Guide

How do I monitor or measure a clock?

You can monitor a clock signal from the **Live Probe** tab when the design is synthesized and compiled with FHB Auto Instantiation turned on in Project Settings dialog box (Enhanced Constraint Flow).

In the **Live Probe** tab, SmartDebug allows you to:

1. Measure all the FABCCC GL clocks by clicking the **User Clock Frequencies** tab, as shown in the figure below.



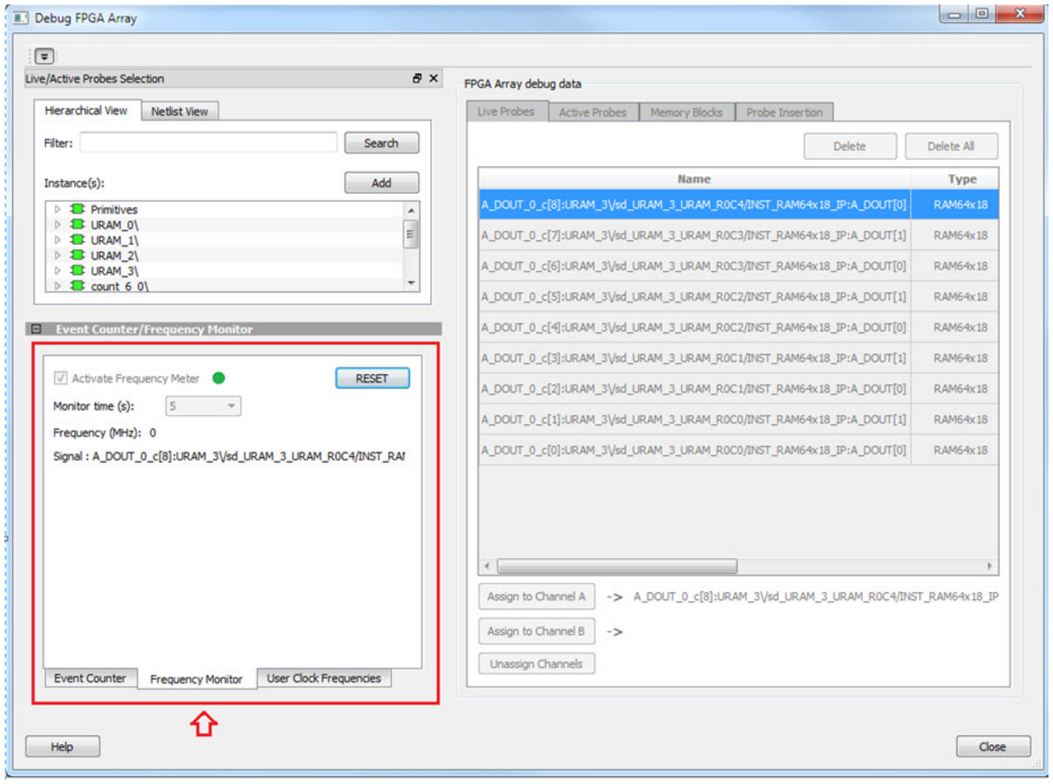
	User Clocks	Frequency (MHz)
1	FCCC_0_GL0	~24.5
2	FCCC_0_GL1	~48.7
3	FCCC_0_GL2	~97.4
4	FCCC_0_GL3	~194.6

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP

Assign to Channel B ->

2. Monitor frequencies of any probe points by:
   a. Assigning the desired signal to Live Probe Channel A.
   b. Selecting the **Frequency Monitor** tab as shown in the following figure and checking the Activate Frequency Meter checkbox.



[✓] Activate Frequency Meter ●

Monitor time (s): 5

Frequency (MHz): 0

Signal : A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAI

Name	Type
A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[7]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[6]:URAM_3\/sd_URAM_3_URAM_R0C3/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[5]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[4]:URAM_3\/sd_URAM_3_URAM_R0C2/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[3]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[2]:URAM_3\/sd_URAM_3_URAM_R0C1/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18
A_DOUT_0_c[1]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[1]	RAM64x18
A_DOUT_0_c[0]:URAM_3\/sd_URAM_3_URAM_R0C0/INST_RAM64x18_IP:A_DOUT[0]	RAM64x18

Assign to Channel A -> A_DOUT_0_c[8]:URAM_3\/sd_URAM_3_URAM_R0C4/INST_RAM64x18_IP

Assign to Channel B ->

How do I perform simple PRBS and loopback tests?

You can perform PRBS and loopback tests using the Debug SERDES option in SmartDebug.

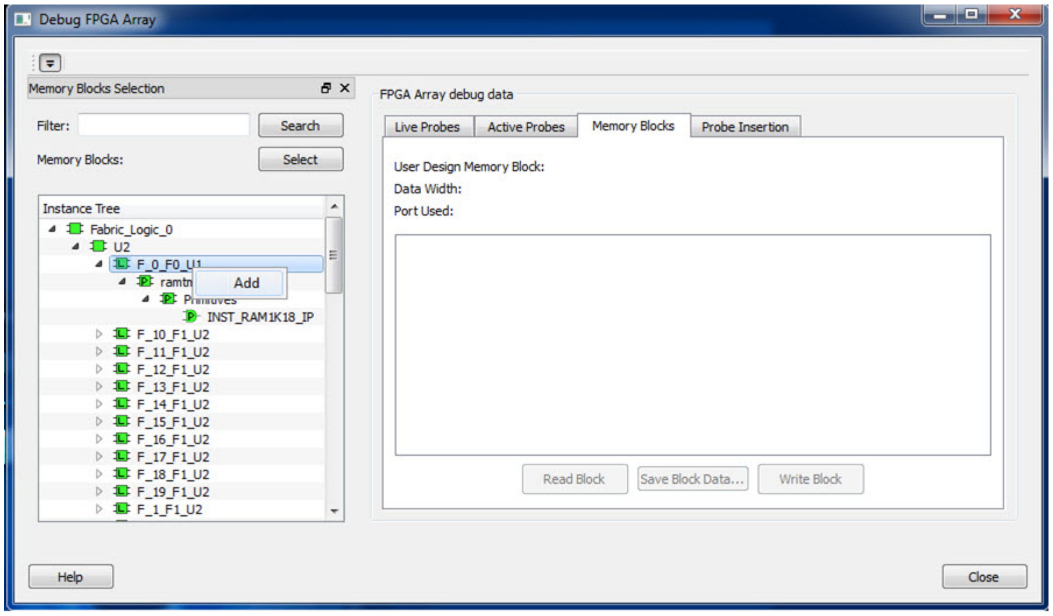
To perform a PRBS test, in the Debug SERDES dialog box, select **PRBS Test** to run a PRBS test on-die or off-die. For more information, see "Debug SERDES – PRBS Test" on page 56.

To perform a loopback test, in the Debug SERDES dialog box, select **Loopback Test** to run a near end serial loopback /far end PMA Rx to Tx loopback test. For more information, see "Debug SERDES – Loopback Test" on page 55.

How do I read LSRAM or USRAM content?

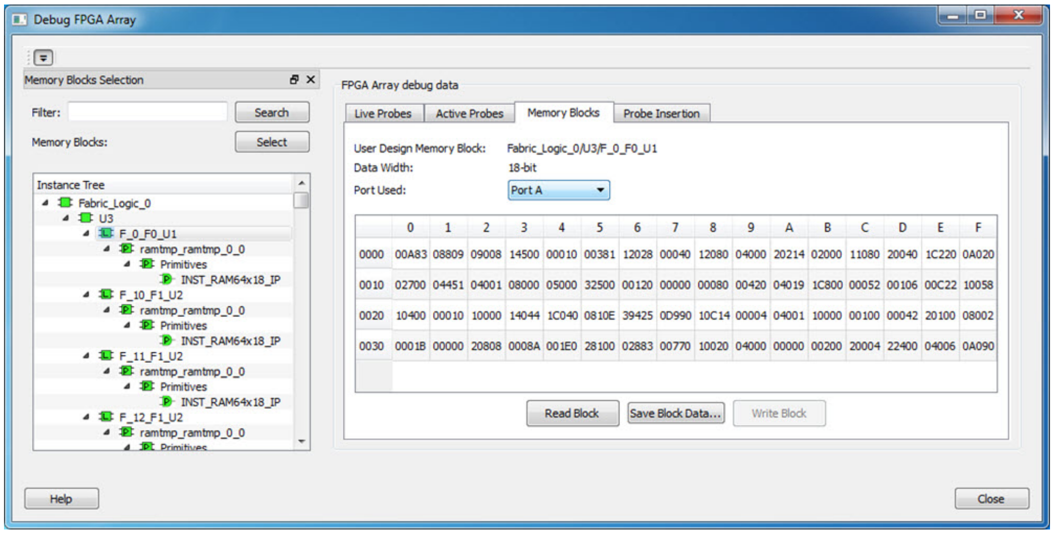
To read RAM content:

1. In the Debug FPGA Array dialog box, click the **Memory Blocks** tab.
2. Select the memory block to be read from the selection panel on the left of the window.



An "L" in the icon next to the block name indicates that it is a logical block, and a "P" in the icon indicates that it is a physical block. A logical block displays three fields in the Memory Blocks tab: User Design Memory Blocks, Data Width, and Port Used. A physical block displays two fields in the Memory Blocks tab: User Design Memory Block and Data Width.

3. Add the block in one of the following ways:
   a. Click **Select**.
   b. Right-click and choose **Add**.
   c. Drag the block to the **Memory Blocks** tab.
4. Click **Read Block** to read the content of the block.



User Design Memory Block: Fabric_Logic_0/U3/F_0_F0_U1
Data Width: 18-bit
Port Used: Port A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	00A83	08809	09008	14500	00010	00381	12028	00040	12080	04000	20214	02000	11080	20040	1C220	0A020
0010	02700	04451	04001	08000	05000	32500	00120	00000	00080	00420	04019	1C800	00052	00106	00C22	10058
0020	10400	00010	10000	14044	1C040	0810E	39425	0D990	10C14	00004	04001	10000	00100	00042	20100	08002
0030	0001B	00000	20808	0008A	001E0	28100	02883	00770	10020	04000	00000	00200	20004	22400	04006	0A090

See Also

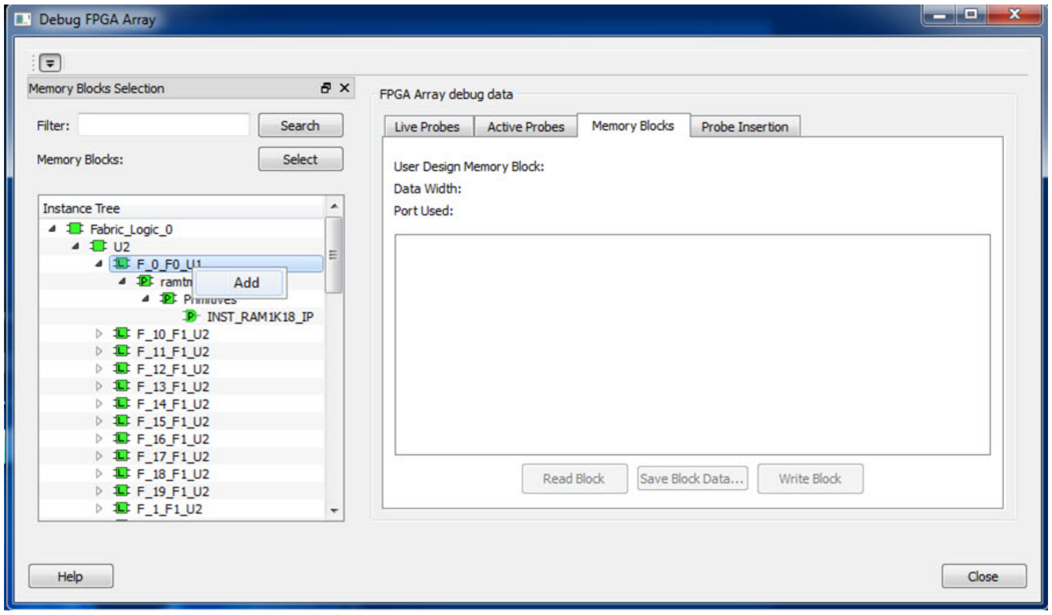
"Memory Blocks (SmartFusion2, IGLOO2, and RTG4)" on page 30

SmartDebug for Libero SoC v11.8 User Guide

How do I change the content of LSRAM or USRAM?

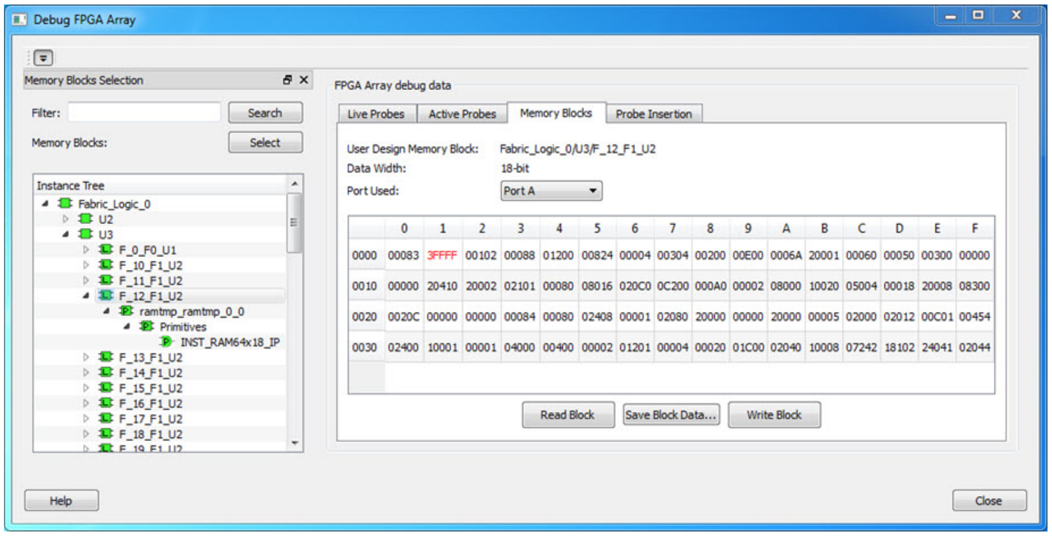
To change the content of LSRAM or USRAM:

1. In the SmartDebug window, click **Debug FPGA Array**.
2. Click the **Memory Blocks** tab.
3. Select the memory block from the selection panel on the left of the window.



An "L" in the icon next to the block name indicates that it is a logical block, and a "P" in the icon indicates that it is a physical block. A logical block displays three fields in the Memory Blocks tab: User Design Memory Blocks, Data Width, and Port Used. A physical block displays two fields in the Memory Blocks tab: User Design Memory Block and Data Width.

4. Add the memory block in one of the following ways:
   a. Click **Select**.
   b. Right-click and choose **Add**.
   c. Drag the block to the **Memory Blocks** tab.
5. Click **Read Block**. The memory content matrix is displayed.
6. Select the memory cell value that you want to change and update the value.
7. Click **Write Block** to write to the device.



User Design Memory Block: Fabric_Logic_0/U3/F_12_F1_U2
Data Width: 18-bit
Port Used: Port A

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	00083	3FFFF	00102	00088	01200	00824	00004	00304	00200	00E00	0006A	20001	00060	00050	00300	00000
0010	00000	20410	20002	02101	00080	08016	020C0	0C200	000A0	00002	08000	10020	05004	00018	20008	08300
0020	0020C	00000	00000	00084	00080	02408	00001	02080	20000	00000	20000	00005	02000	02012	00C01	00454
0030	02400	10001	00001	04000	00400	00002	01201	00004	00020	01C00	02040	10008	07242	18102	24041	02044

See Also

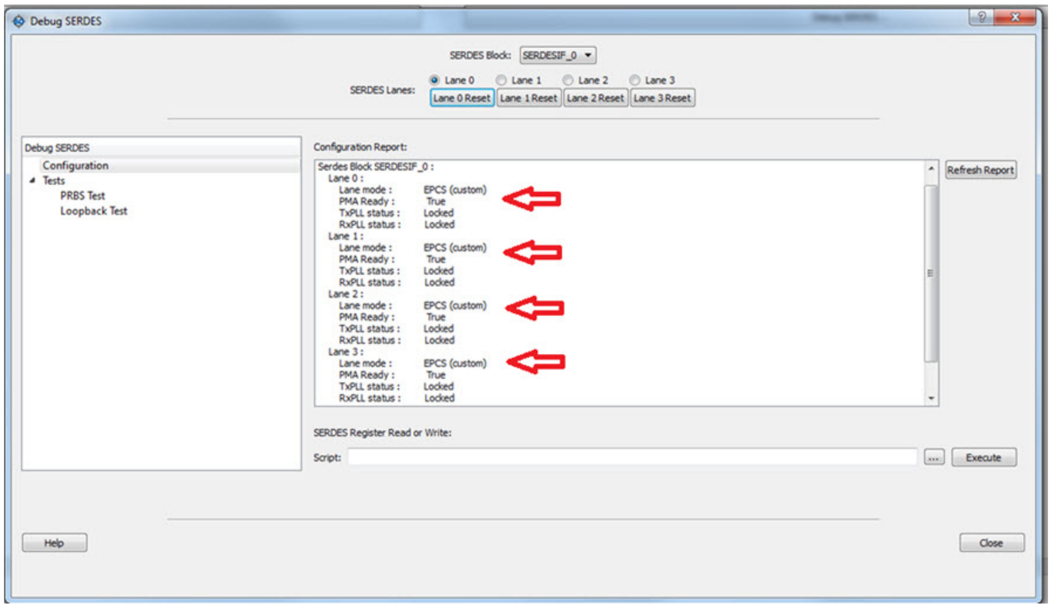
"Memory Blocks (SmartFusion2, IGLOO2, and RTG4)" on page 30

SmartDebug for Libero SoC v11.8 User Guide

How do I read the health check of the SERDES?

You can read the SERDES health check using the following Debug SERDES options:

1. Review the **Configuration Report**, which returns PMA Ready, TxPLL status, and RxPLL status. For SERDES to function correctly, PMA ready should be true, and TxPLL and RxPLL status should be locked. The Configuration Report can be found in the Debug SERDES dialog box under Configuration. See Debug SERDES (SmartFusion2, IGLOO2, and RTG4).



Configuration Report:

```
Serdes Block SERDESIF_0 :
  Lane 0 :
    Lane mode :     EPCS (custom)
    PMA Ready :     True
    TxPLL status :  Locked
    RxPLL status :  Locked
  Lane 1 :
    Lane mode :     EPCS (custom)
    PMA Ready :     True
    TxPLL status :  Locked
    RxPLL status :  Locked
  Lane 2 :
    Lane mode :     EPCS (custom)
    PMA Ready :     True
    TxPLL status :  Locked
    RxPLL status :  Locked
  Lane 3 :
    Lane mode :     EPCS (custom)
    PMA Ready :     True
    TxPLL status :  Locked
    RxPLL status :  Locked
```

2. Run the **PRBS Test**, which is a Near End Serial Loopback tests on selected lanes. This should result in 0 errors in the Cumulative Error Count column. See "Debug SERDES – PRBS Test" on page 56.

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 <Libero 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 <Libero 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 - <Libero IDE project>/Smartgen/AnalogBlock
- SmartFusion devices - <Libero 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.

What happens if invalid firmware is loaded into eNVM in SmartFusion2 devices?

When invalid firmware is loaded into eNVM in SmartFusion2 devices, Cortex-M3 will not be able to boot and issues reset to MSS continuously. eNVM content using View Flash Memory content will read zeroes in SmartDebug.

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

Device Status Report: 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)	Vccnvm 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

Security Status Info	Description
	feature is enable for encrypted programmed. This allows field design 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	Check if the page-write count is within the expected range. The expected write count is greater than or equal to: 6,384 - SmartFusion devices 2,288 - Fusion devices 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. 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. Since the threshold can only be set in multiples of 4096 (2^12), 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.

Addendum - FHB IP Design Specification

Introduction

FPGA Hardware Breakpoint (FHB) is silicon technology introduced by Microsemi in its SmartFusion2 and IGLOO2 FPGAs.

FHB uses the Live Probe and Active Probe features in SmartDebug to enable the selection of any flip-flop in a user design as a trigger to temporarily “pause” the design. This allows the synchronous elements to be tapped for their state information through the debug port built into the device.

This document describes the various features of the IP, the RTL involved, timing and block diagrams, pin-out information, and other features.

Features

The FPGA Hardware Breakpoint (FHB) IP includes the following features:

- DUT execution control with Active Probe
  - Halt
  - Run
  - Step
- DUT execution control halt
  - Trigger input (Live Probe)
  - Software control (SmartDebug)
- Delayed triggering capability up to 256 clock cycles

DUT Execution Control with Active Probe

DUT execution control is achieved with three inputs to the FHB IP through Active Probe write, as described in the following sections.

Halt

When this signal is asserted, it halts DUT execution by pausing the clock and therefore freezing all synchronous elements in their current states. These elements can then be captured with Active Probe read. This signal is given directly with Active Probe write whenever the user needs to pause the DUT. DUT execution can also be paused through the trigger signals from the DUT, which are connected through Live Probe.

When this signal is asserted, it passes through the clock synchronization and pos-edge detection circuits, and is then given to the pulse delay module, which holds the signal for the necessary number of clock cycles, as specified by the user. The output signal GLx_ENB is then de-asserted and the DUT is paused.

Run

This feature is used to “unpause” the DUT. Essentially, it performs the reverse operation of the Halt feature. This signal is also similarly issued by the user through Active Probe write. Once this signal is asserted, it releases the hold on the DUT and resumes its operations from the current state.

Similar to the Halt feature, the signal is passed through a series of synchronizer and pos-edge detection blocks before being sent to the combinational logic circuit that enables the value of GLx_ENB. This resumes the DUT functionality.

Step

This feature is used by the user to observe/control values over a single clock cycle. Functionally, this works as a combination of Halt and Run signals being asserted in a space of one clock cycle. Therefore, once the Step is asserted, GLx_ENB is disabled for one clock cycle, and DUT values can be captured once it is paused. This value can also be asserted through Active Probe write.

Logically, the Step function is achieved by synchronizing and detecting the positive edge of the Active Probe write signal. It is then sent to a multiplexer circuit that controls the value of GLx_ENB to switch over for one clock period.

Controlled execution of this signal can extract valuable chronological data on the DUT and an output waveform (similar to a simulation waveform) can be constructed by multiple clock cycle step executions.

Soft Reset Capabilities

This feature allows the user to reset the state of the FHB at any point in the design/debug process without having to restart the DUT or chip. The Soft_Reset signal is asserted through Active Prove write.

Delayed Triggering Capabilities

With this feature, an input parameter induces a delay into the system response for the trigger. For example, if the user wants to pause the DUT 20 clock cycles after observing a certain output or a counter value, he can input 20 into the delay register in the FHB IP by using an Active Probe write command. Once the event is triggered and the appropriate signal is sent, the delayed trigger then waits for 20 clock cycles before de-asserting the GLx_ENB and INT signals that would Halt/pause the DUT.

The user can also choose not to induce any delay into the trigger mechanism to Halt the DUT, in which case the trigger goes through without any delay. If the user wants to delay the pulse, the minimum possible delay is four clock cycles due to the presence of three additional registers in the counter path. The mechanism to implement the delay logic is accomplished through a “down counter”. With a zero value as the default delay, the input pulse is always sent through without any delay. Through SmartDebug, the user can choose to write in a non-zero value as the delay using Active Probe write, and the subsequent trigger will be delayed by the specified number of clock cycles. The output pulse is a “NOR” of all the bits of the count. Therefore, once the count is zero, a pulse is generated to be the output. However, the required inclusion of the NOR gate in the path of the trigger will lead to a higher propagation and routing delay due to the inclusion of the LUT in the path. Unless the user chooses to add a delay, for a 50MHz signal, there will not be any delay in triggering.

Global Halt Capabilities

A single FHB block can produce a single GLxEB signal. This signal can be used to control one or more GLx signals in an FCCC Module. However, if the user wants to have a different control signal for each different clock in the design, they can instantiate more than one FHB module to accomplish this. To enable this global halt capability, all the FHB blocks’ Halt signals are tied together to the Live Probe feedback signal.

The Halt signal from the Liveprobe module is given controlled access to the FHB through the MuxA and MuxB settings. The FHB can be “armed” or “disarmed” with these settings. Therefore, to implement the global halt capabilities, the user can set all the muxes in the design to select the Liveprobe Halt feature. Once this mode is selected, all FHB blocks will respond to a positive edge from this path.

CLK INT Mux Selection

The following figure shows CLK INT Mux selection connections.

This block accommodates the user’s GLX_ENB signal, if it is used.

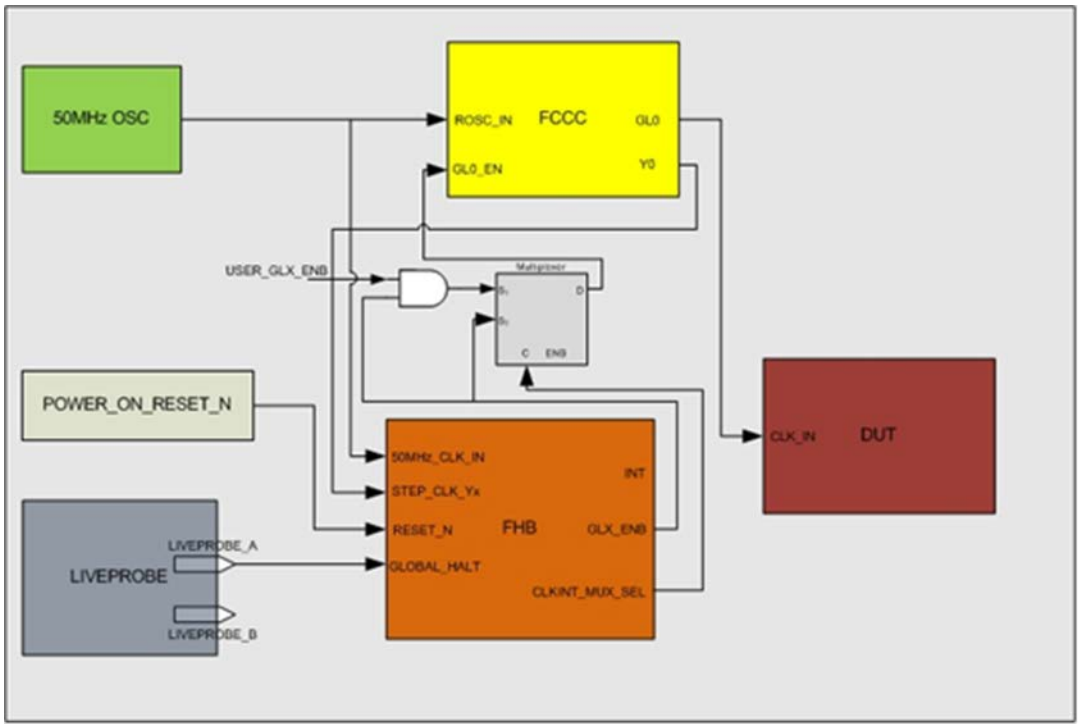


Figure 60 · CLK INT Mux Selection Connections

FHB Halt Status Monitor

The FHB Interrupt monitor is logic that must be added to either each individual FHB module or to a single monitor for the whole design. The Interrupt monitor lets the user monitor which clocks are paused and which clocks are active. This logic should be able to communicate with SmartDebug through the JTAG interface without interrupting or interfering with normal FHB operation. Therefore, ActiveProbe reads cannot be used to monitor all of the INT register values, as that may possibly break existing LiveProbe connections.

For this function, all INT bits from all FHB modules in the design are written into a single USRAM block, and SmartDebug reads to the USRAM location by using the SII bus. This will not terminate the Liveprobe connection, because it uses a different bus. External logic to the FHB modules must be implemented, and all INT signals must be connected to it. This logic uses one USRAM block and additional LUTs and registers for the user logic.

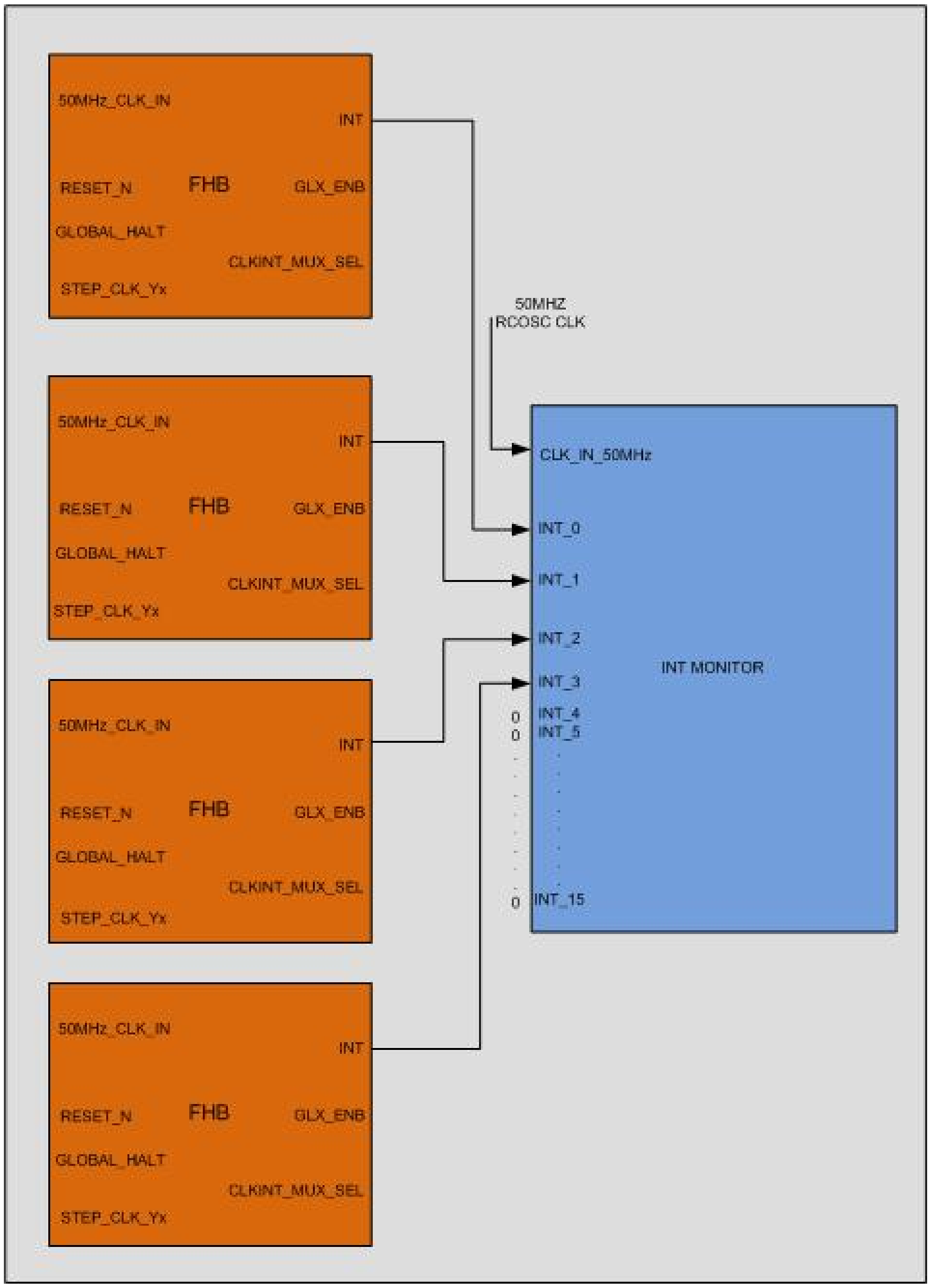


Figure 61 · Halt Status Monitor Connections

Note: *The INT monitor block and all of its subsequent connections are external to all FHB modules, because each individual block is connected and this monitor block cannot be placed inside a FHB module. The total number of FHB blocks is dependent on the number of FCCC clocks the user has in their design. Additionally, any unused inputs to this block are tied "low". For these reasons, the INT monitor block is placed external to all FHB modules, as shown in the diagram above.*

Event Counter and Frequency Monitor

The Event Counter and Frequency Monitor operations can be performed by a block that is placed external to the main FHB module, and can be used to count the number of "events" or toggles of a given signal within a user-specified time period. Using the Liveprobe feedback macro, the user selects a signal during real time/debug time to connect to the Event Counter. These blocks are free running counters that count the number of edges of the input signal. Therefore, in conjunction with the Active Probe read feature, this can be quickly used to determine the pseudo-staticity of a signal, or to find any "stuck-at" errors.

The Frequency Monitor block can be used to calculate the frequency of a fabric clock in real time. This block uses Live Probe to route the clock that the user wants to calculate the frequency of to the Frequency Monitor block. The frequency is measured through two free running counters, one at 50 MHz and the other connected through Live Probe. Using internal control logic, both counters start running when an input clock has been detected through the Liveprobe feedback macro. The software is programmed to automatically stop the count operation after a specified period of time and calculate the ratio between the two counts (i.e., the unknown clock's count with respect to the 50MHz clock source). This calculates the frequency of an unknown signal.

The connections to this block are as follows:

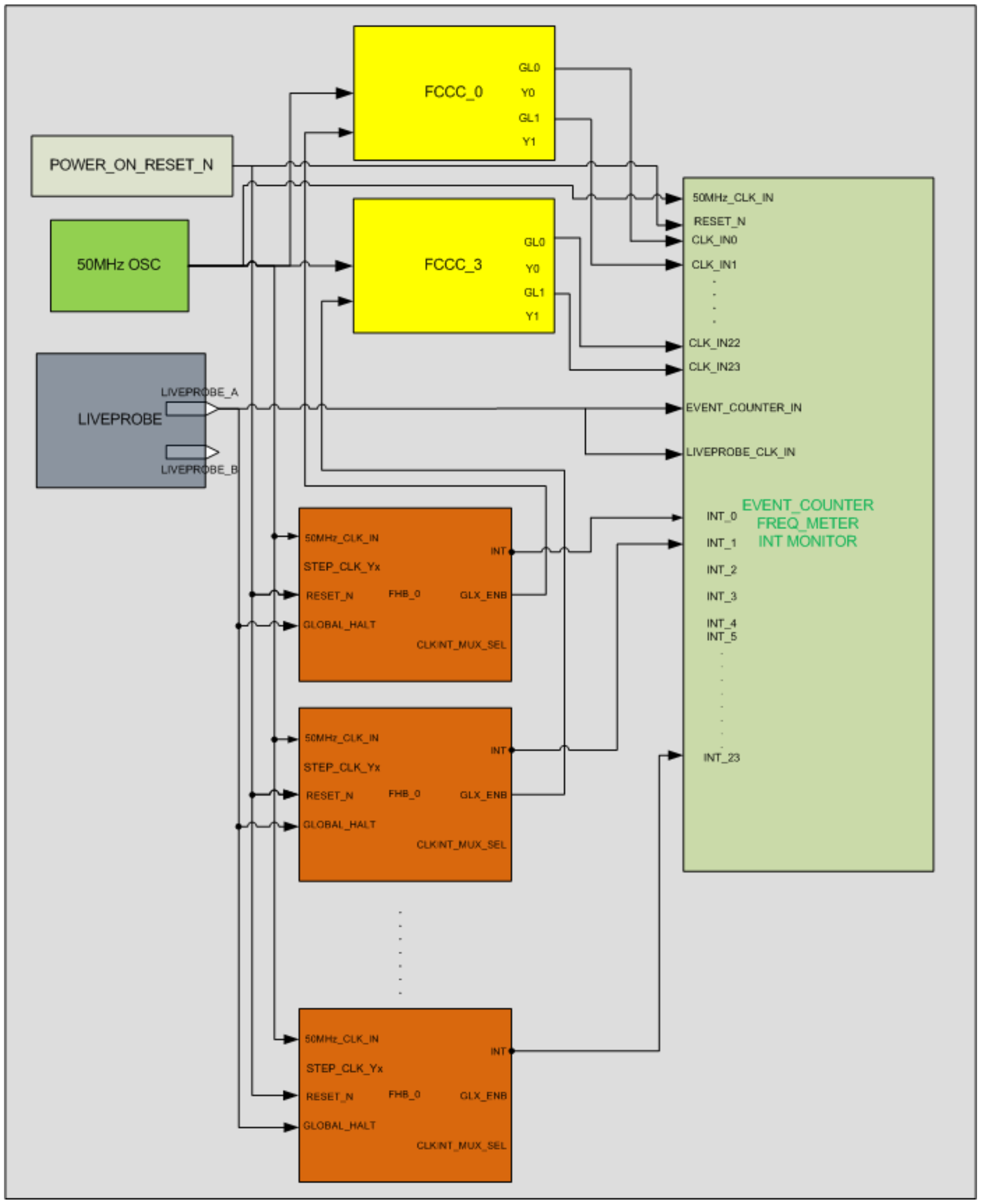


Figure 62 · Event Counter, Frequency Monitor, and INT Monitor Merged in One Block

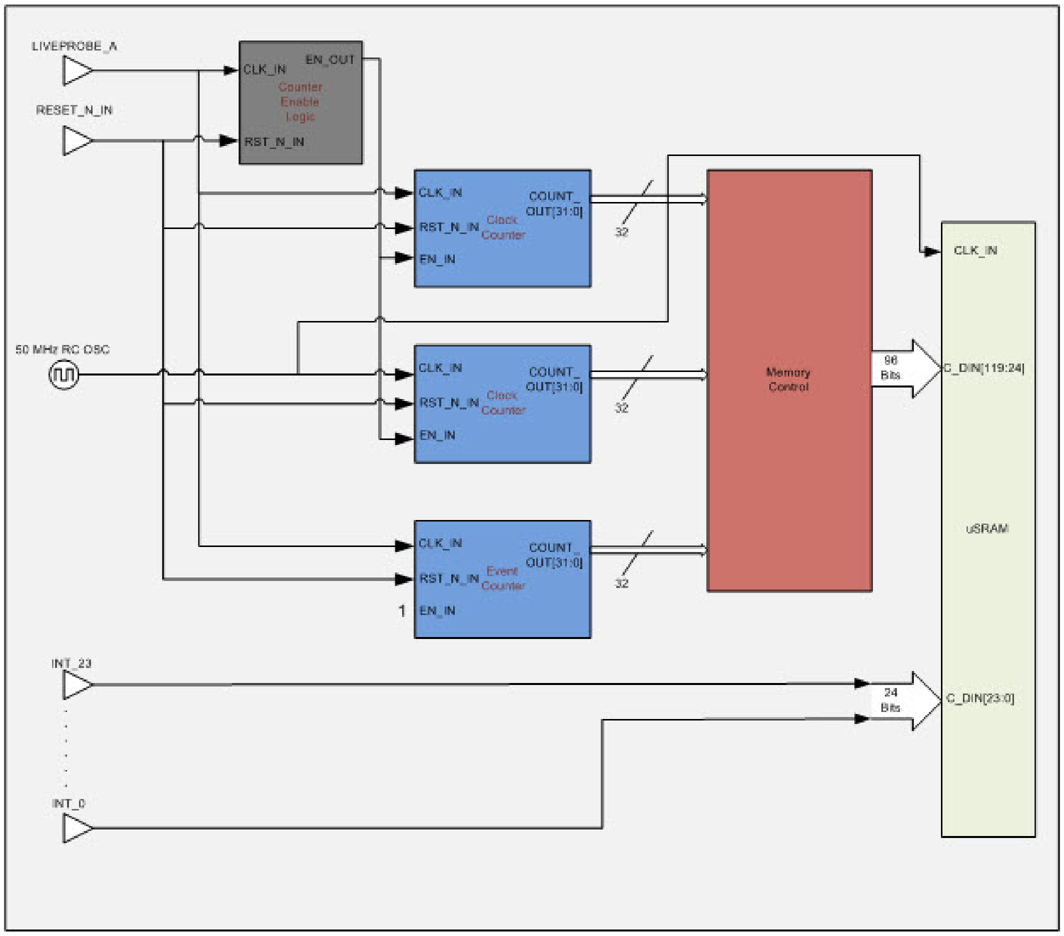


Figure 63 · Clock Counter Connections for Frequency Monitor and Event Counter

Input/Output Port Descriptions

The inputs and the outputs of the block are summarized in the following tables.

Table 8 · Input Signals of FHB IP

INPUT SIGNAL (All signals in blue are Probewr signals that are sent through the JTAG)	**DESCRIPTION**
Halt	A signal that can be issued from the JTAG to manually override the internal "Triggers" to halt the clock of the DUT 1:Posedge-Set backdoor input value of pulse synchronizer H to '1' 0:Negedge-Set backdoor input value of pulse synchronizer H to '0'
Run	A signal that restores the DUT clock and asserts the value of GLx_ENB to '1'. This is used to bring the circuit out of the Spatial Debug mode and restore functionality. This signal can be manipulated along with the Step signal for Multi-Snapshot Debugging 1: Posedge-Set backdoor input value of pulse synchronizer I to '1' 0: Negedge-Set backdoor input value of pulse synchronizer I to '0'
Edge Select	A bit that can be used to change the polarity of the trigger. The user can program either 1 or 0 onto this select bit through Active Probe write command and the edge is triggered accordingly. By default FHB is triggered on the positive edge of the input pulse '0' to make the circuit trigger on the positive edge of the input pulse '1' to make the circuit trigger on the negative edge of the input pulse

Step	This signal can be used to halt the DUT Clock by just one clock cycle. (i.e. turning off GLx_ENB to '0' for a period of just one clock cycle). 1: Posedge-Set backdoor input value of pulse synchronizer J to '1' 0: Negedge-Set backdoor input value of pulse synchronizer J to '0'
Pulse Delay	This is a feature that can be customized by the user to control the number of clock cycles before which the clock is frozen after the trigger is applied. This is very useful when the user wishes to apply the trigger before the snapshot needs to be taken. For example, if the user wishes to take the snapshot 20 clock cycles after a certain event has occurred, he can customize this value to disable the GLx_ENB signal accordingly. Parameterized input value that can be set by the user to range from 0-256. The parameter value bits also have to pass through the synchronizer logic before they can be used in the circuit.
CLKINT_MUX_SEL	This register is used to switch control of the GLx_ENB signal between the user logic and the FHB module. It is tied to "Low" by default, hence enabling the user controlled GLx_ENB to take effect. Once it is overwritten by Smartdebug through Activeprobe write to be "high", and then it gives control to FHB.
Global Halt	This is a signal that is added to implement a "Global Halt" when multiple FHB Blocks are implemented in a design. If the user intends to use multiple clock domains, they will have to use a different FHB block for each clock, but they can use this signal to halt all the clocks and debug them individually as necessary. To enter this mode, the following Mux settings need to be implemented: MuxA to 1 MuxB to 0
POWERUP_RESETN (Active low)	This is the master reset of the IP block which is used to reset all the synchronous elements in the block to ensure that there are no unknown or tri-state values in the design that may be propagated to cause jitter and meta stability in the circuit. This signal needs to be driven by POR (Power on Reset) of the SmartFusion2 Device.
50MHz OSC	The 50MHz OSC is used to run the FHB Halt circuit. This is not the same clock that controls the DUT. Maximum operating frequency of the IP is 50MHz, but that can be timing close for the DUT+FHB IP.

Table 9 · Output Signals of FHB IP

OUTPUT SIGNAL	DESCRIPTION
GLx_ENB	The control signal for the DUT clock which is used to halt the clocks in the DUT to enable the user to take a snapshot of all the synchronous elements in the DUT when triggered. This is sent as a select signal for the glitchless mux as already described. 1: Posedge-Set the 2nd input value of MuxA to '1' 0: Negedge-Set the 2nd input value of MuxA to '1'
INT	This is an Interrupt signal that is sent to the interrupt monitor block to inform the software if the clock has been paused or not.
CLKINT_MUX_SEL	The CLKINT_MUX_SEL signal can be used to switch control of the GLx_ENB signal between the user logic and the FHB module. This is only used in the case where there is a user driven logic also controlling the GLx_ENB signal. Without any selection, the user is still capable of using the FHB's GLx_ENB signal since it is logically coupled (AND gate) with the user driven signal. This selection is necessary only when the user wants to use the Step functionality of the FHB i.e. when they want to run the DUT by a single clock pulse at a time. For this to work, any user DUT signal cannot interfere with the GLx_ENB signal. 0: Default value, control is given to both user driven GLx_ENB and FHB driven

	value 1: Removes control of the user driven GLx_ENB and gives sole control to the FHB module for implementing the Step functionality
STEP_CLK_Y0	The "step" function controls the GLX_ENB signal to perform the execution of the DUT by 1 DUT clock cycle at a time. To generate this single clock pulse in the correct frequency, the corresponding ungated clock (Yx) of the FCCC is connected to the FHB.

The required outputs can be achieved by manipulating these input signals. The final result is to achieve control over the DUT clock through the manipulation of GLx_ENB.

High Level Block Diagram

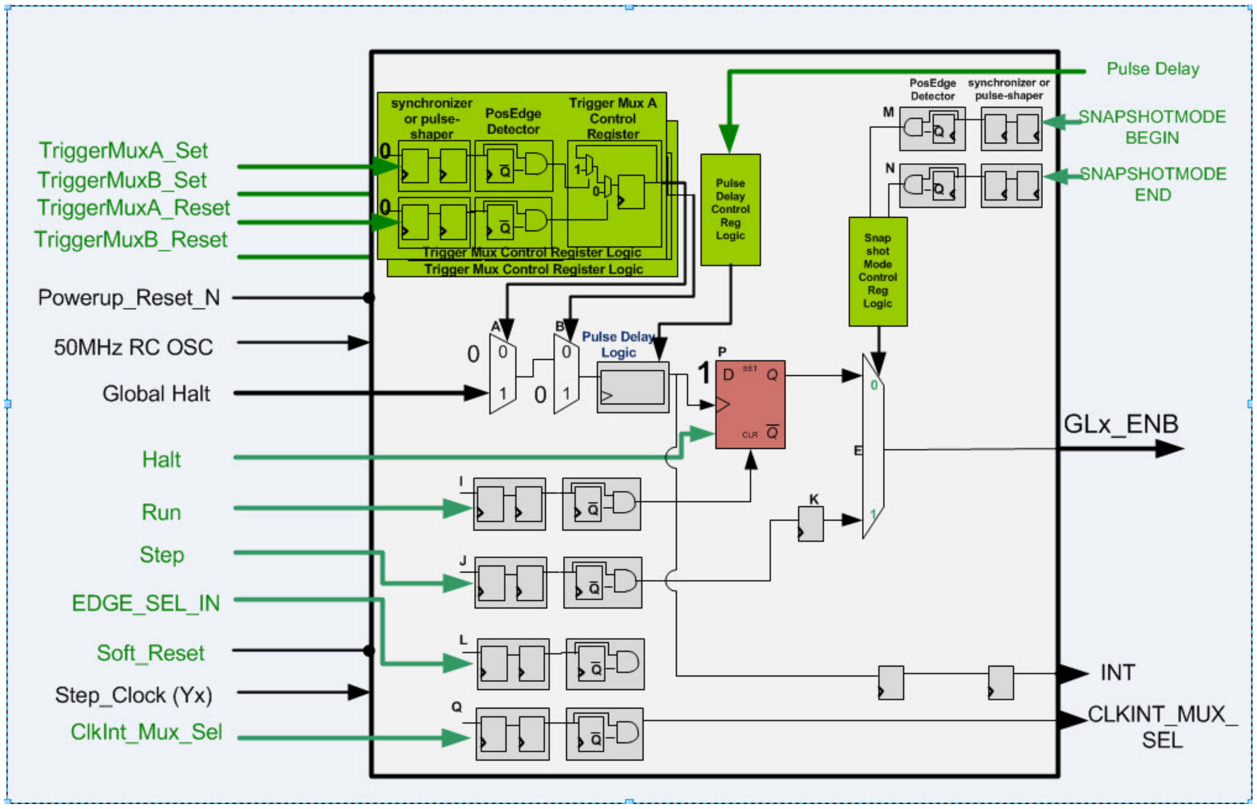


Figure 64 · FHB IP Block Diagram

RTL Architecture

Clock Gating

As described previously, the concept of spatial debug relies on the technique of "pausing" the system in a particular state and then capturing all the states through back-door logic embedded in the FPGA architecture. Clock gating is used to achieve this pseudo-static state.

Clock gating uses a glitchless mux which multiplexes the clock signal with a “logical low” level input to give the output GLx. This is the gated clock that drives the user design or the DUT. The timing diagram for clock, GLx_ENB and output GLx is shown below in the following figure.

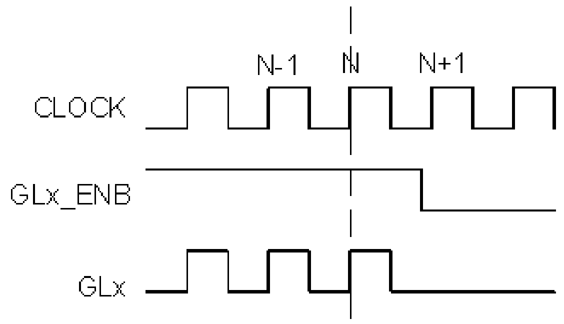


Figure 65 · Clock Gating Circuit Timing Diagram

The function of this circuit is the key to capturing the state of all the elements in the design. The trigger signal GLx_ENB controls the functioning of this circuit. When this signal is triggered, the entire DUT is paused. Therefore, this trigger signal must be chosen carefully by the user, so they can see the states of the flip-flops and understand why bugs may have occurred.

Triggering

Triggering is a crucial part of the FHB technology, since this is the control signal that enables the entire capture of the DUT. Triggering is user driven logic, and can be a signal from within the DUT, from the testbench logic, from the firmware, or from an FPGA pin. This allows the user to select the triggers and halt the process through various sources.

The trigger is supplied from FPGA fabric logic through Live Probe, which is labeled as “Trigger”. Therefore, a small control and synchronization circuit must be designed to ensure that the correct trigger is selected and sent through to halt the clock of the DUT. This logical element is labeled as “Trigger Mux Control Register Logic,” as shown in the following figure.

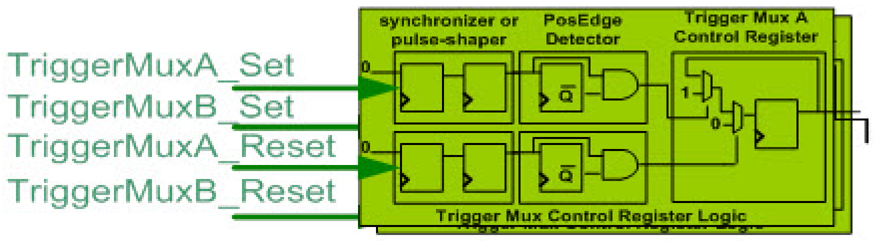

TriggerMuxA_Set
TriggerMuxB_Set
TriggerMuxA_Reset
TriggerMuxB_Reset
synchronizer or pulse-shaper
PosEdge Detector
Trigger Mux A Control Register
Trigger Mux Control Register Logic

Figure 66 · Trigger Mux Control Register Logic

The trigger mux control logic includes the following elements:

- Synchronizer
- Positive Edge Detector
- Trigger Mux Control Registers

Synchronizer

The synchronizer circuit consists of a “pseudo-latch” followed by a clock synchronizer circuit that latches onto the ‘probewr’ value and passes it through a series of flip-flops to ensure a glitchless operation. When the JTAG command comes in, the back door logic writes the value onto the latch, and that is propagated into the circuit. The latch logic is brought about by using a flip-flop with an always-high clock that ensures that the input value is latched onto the output. This also does not induce any timing violations into the circuit that might be seen with a latch, because the logic is implemented with a flip-flop. The same logic is applied to both the Set and the Reset signals. The schematic of this circuit is as shown in the following figure.

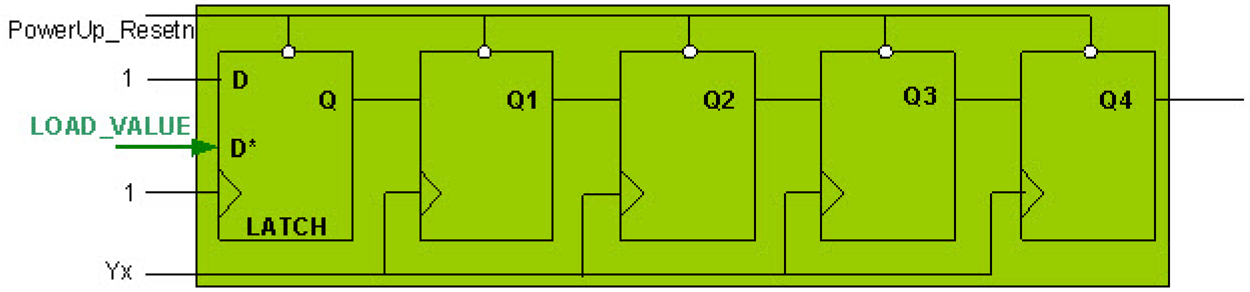


Figure 67 · Probewr Synchronizer Circuit

Positive Edge Detector

Once the probewr signal is synchronized and propagated through the synchronizer circuit, it is necessary to make sure that the circuit detects a change in the value of the input. This is accomplished by using a positive edge detector circuit. This is a fairly straightforward logical element with an “AND” gate logic used to detect a change in the value of the input from “0” to “1”. Once the circuit determines a rising edge for either the set (or the reset) signal, it is then propagated into the two multiplexers that control the final output.

Control Register

This part of the circuit has two multiplexers driving a flip-flop that gives the final output for the block. This value is then sent to the external mux. The select bits of these two multiplexers are the Set and the Reset outputs from the positive edge detector block, as shown in Figure 67. This logic block functions similar to that of a JK Flip Flop. When the Set bit is ‘1’, the output of the block is ‘1’. When the Reset bit is ‘1’, the output of the block is ‘0’. If there is no input from either the Set or the Reset bits, the previous value of Q is retained because of the feedback loop. It is not possible for both the Set and the Reset bits to be ‘1’.

Disarm Trigger

Once the FHB has been “armed” (i.e. it is ready to take a trigger input from the Liveprobe input to HALT the DUT), it is also possible to “disarm” it. The disarm operation reverses the settings that make the FHB sensitive to this input. It does not react to the Liveprobe signal. This option is useful when the user either wishes to cancel the trigger due to inactivity or if they wish to change the trigger signal.

Utilization Figures

The utilization figures for the FHB module (including Event Counter, Frequency Monitor, delay block, and the Interrupt Monitor) will vary, depending on the features that are implemented and used. The utilization figures in the following table are for SmartFusion2 and IGLOO2 only.

IP	Fabric 4LUT	Fabric DFF	Interface 4LUT	Interface DFF	Single-Ended I/O	uSRAM 1K	PLLs and CCCs
FHB	77	218	0	0	0	0	0
Halt status Monitor, Event Counter, and Frequency Monitor	227	162	36	36	0	1	0