UG0602
User Guide
RTG4 FPGA Programming

Microsemi
a Microchip company

Microsemi Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), 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. Learn more at www.microsemi.com.

Content

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 7.0

The following is a summary of changes made in revision 5.0 of this document:

- Updated the information about FlashPro Express. For more information, see Digest, page 10.
- Update the information about power interruption during programming. For more information see, Programming Flow, page 4.

1.2 Revision 6.0

Information about Digest check was updated. See Digest, page 10.

1.3 Revision 5.0

The following is a summary of changes made in revision 5.0 of this document:

- Information about the silicon sculptor was removed. Silicon sculptor programming is not supported for RTG4 devices.
- Information about JTAG_TCK was updated. See JTAG Programming, page 5.

1.4 Revision 4.0

The following is a summary of changes made in revision 4.0 of this document:

- Added a note about standalone erase in the introduction chapter. For more information, see RTG4 FPGA Programming, page 2.
- Deleted the SPI Slave Programming section and all the related information about SPI slave programming in this document.
- Added the Digest Checks section. For more information, see Digest, page 10.
- Replaced the pin name VDDJ with VDDI3 in the RTG4 FPGA Programming chapter. For more information, see RTG4 FPGA Programming, page 2.

1.5 Revision 3.0

The following is a summary of the changes in revision 3.0 of this document.

- Information about RTG4 programming modes was updated. For more information, see Table 1, page 2.
- The RTG4 programming flow was updated. For more information, see Programming Flow, page 4.
- The Programming Using an External Microprocessor section was removed.
- Information about the SPI master was updated.
- The document was changed to the new template.

1.6 Revision 2.0

Removed references of security throughout the document (SAR 66946).

1.7 Revision 1.0

Revision 1.0 was the first publication of this document.

2 RTG4 FPGA Programming

RTG4™ FPGAs offer a variety of programming options to cater to diverse end-user applications. The device supports programming of the following components:

- FPGA fabric
- µPROM

RTG4 devices can be programmed mainly by the following methods:

- JTAG programming
- Production programming

In JTAG programming mode, the device is programmed using an external master such as Microsemi FlashPro4/5 programmer. In this programming mode, the external master fetches the programming data (bitstream) from an external memory. In production programming, the device is programmed through a standalone programmer or an automatic test equipment (ATE) before mounting the device on the board.

The following table lists different programming modes and interfaces. In the following sections, these programming methods are discussed in detail. Microsemi recommends programming the RTG4 devices in radiation-free environments.

Table 1 • **RTG4 Programming Modes**

Mode	Interface	Software	Master	Bitstream Format
JTAG Programming	System Controller dedicated JTAG port	Libero® SoC/FlashPro	FlashPro4/5	Libero SoC default file/STAPL
Production Programming	System Controller dedicated JTAG port	ATE specific test program	ATE	STAPL

Note: Standalone erase is not required to perform prior to programming RTG4 devices because as part of the standard programming procedure, erase is always performed on RTG4 devices prior to programming. Therefore, an additional standalone erase is not needed. If you perform standalone erase, it increases the programming count by two. If standalone erase is performed on flight units, ensure that the programming count stays within the specification range, which is 200 cycles including program and standalone erase cycles. For more information about the specification range, see *DS0131: RTG4 FPGA Datasheet*.

2.1 Bitstream Generation

Libero SoC generates the programming bitstream required to support the different programming modes.

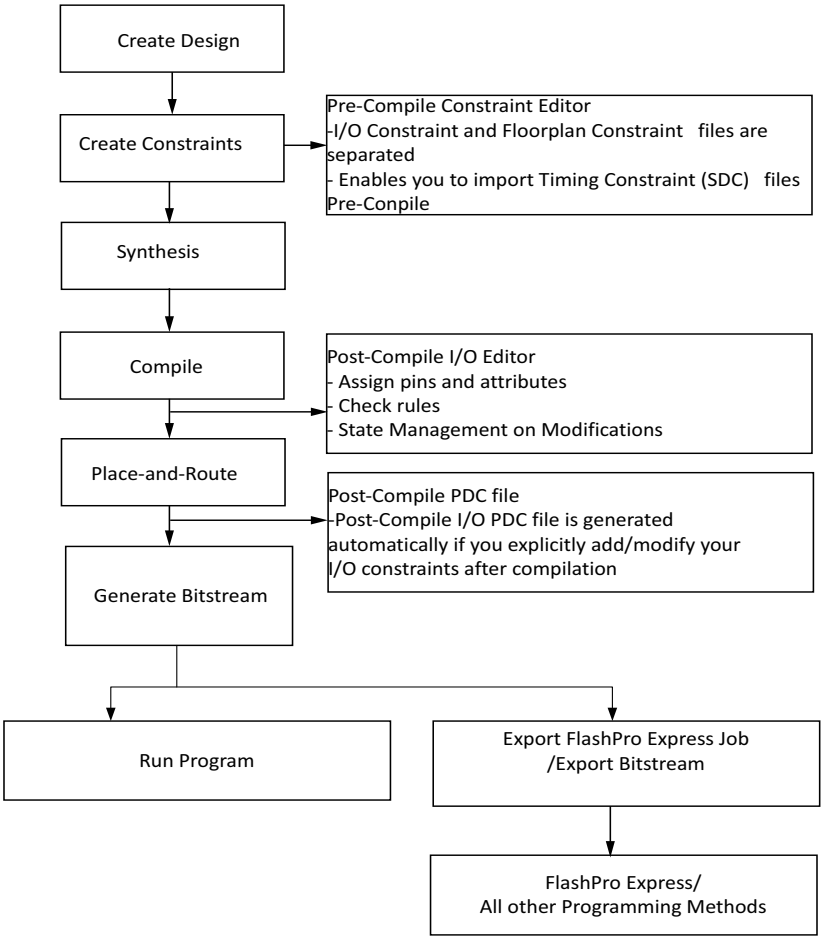
2.1.1 Libero SoC Programming Bitstream Generation Flow

Libero SoC is used to generate the programming bitstream formats needed for different programming modes. The following figure shows the Libero SoC programming bitstream generation flow.

After implementation of the design, the programming bitstream is generated by clicking the **Generate Bitstream** option in Libero. As programming is integrated into the Libero SoC software design flow, you can program the device directly by clicking **Program Design > Run PROGRAM Action**.

You can also program the device using standalone programming tool FlashPro Express. Export *.job file from Libero SoC (Export FlashPro Express Job menu). For more information, see *FlashPro Express User Guide*.

Figure 1 • **Libero SoC Programming Bitstream Generation Flow**



2.1.2 Programming File Size

The programming bitstream contains programming data for FPGA fabric and/or µPROM (partial or full content). The following table lists the typical bitstream size and format for RTG4 devices.

Table 2 • **Programming Bitstream Size**

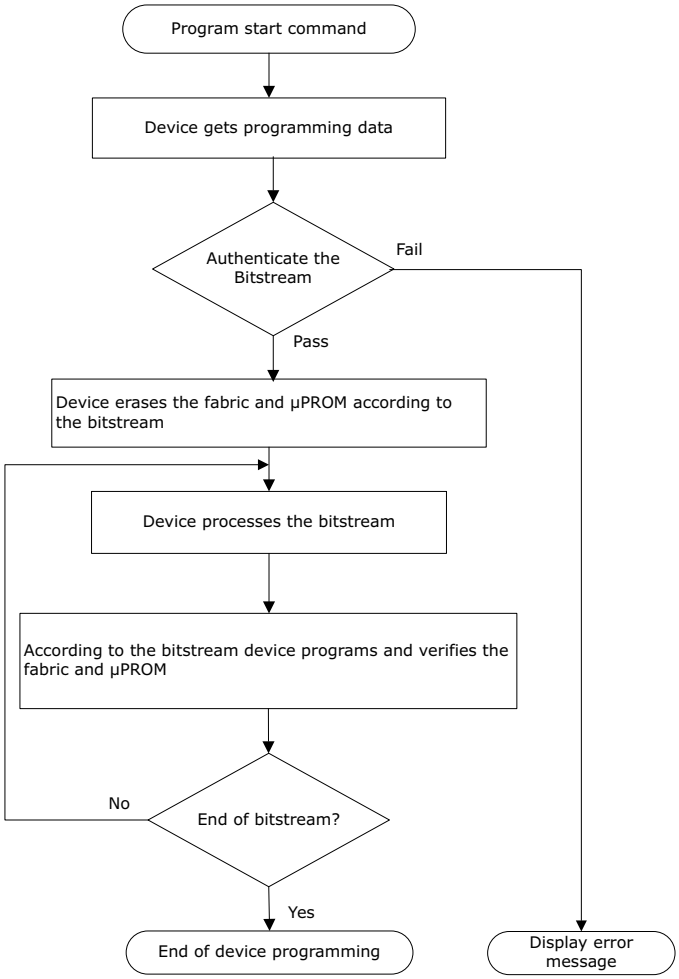
Device	File Type	Fabric + µPROM Full
RT4G150	STAPL	8 MB

2.2 Programming Flow

The programming flow starts when the system controller initiates the device programming start command, and ends when the bitstream data is fully transferred and verified. Verification of the programmed contents are part of the programming flow. It is recommended not to interrupt the programming flow. If the programming flow is interrupted or fails before completion, the device will not be enabled. The FPGA is enabled only after the entire bitstream is successfully programmed. Interrupt during programming may damage the device. The following figure summarizes the RTG4 programming flow.

Note: Power interruption is not supported during programming.

Figure 2 • **Programming Flow**



2.3 JTAG Programming

RTG4 devices have a built-in JTAG controller that is compliant with IEEE 1149.1 and compatible with IEEE 1532. An external programmer, such as FlashPro4/FlashPro5, is used to program the device. The devices can be programmed in both single and chain modes.

The RTG4 devices have JTAG pins in a dedicated bank, which varies depending on the package. For more information about banks and their locations, see *RTG4 FPGA Pin Descriptions*.

JTAG banks can be operated at 1.8 V, 2.5 V, or 3.3 V.

The logic level of the JTAG signals depends on the JTAG bank voltage. The following table lists the JTAG pin names and descriptions.

The JTAG interface is used for device programming and testing or for debugging instantiated soft processor firmware, as listed in the following table. JTAG I/Os are powered by the VDDI3 supply associated with the bank where the I/O reside.

Table 3 • **JTAG Pins**

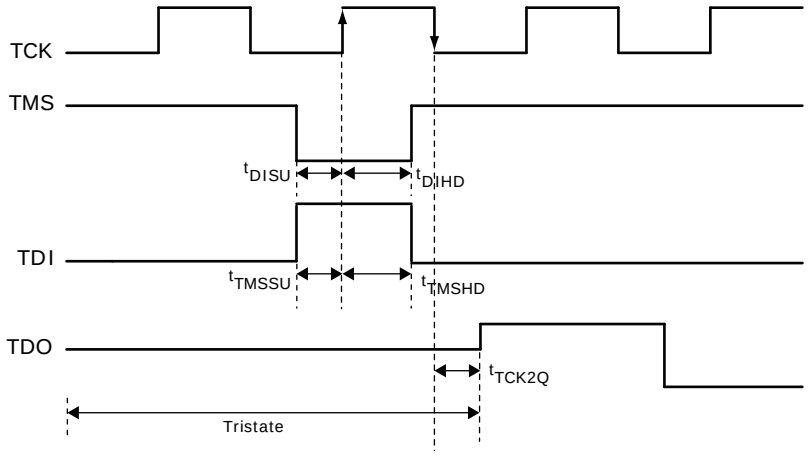
Pin Name	Direction	Weak Pull-up	Description
JTAG_TMS	Input	Yes	JTAG test mode select. Do not connect (DNC) in unused condition.
JTAG_TRSTB	Input	Yes	JTAG test reset. Must be held low during device operation. In unused condition, pull down to VSS through 1 kΩ resistor for upset immunity.
JTAG_TDI	Input	Yes	JTAG test data in. DNC in unused condition.
JTAG_TCK	Input	No	JTAG test clock. When unused, TCK be tied to VSS or VDDI3 through 200 to 1 kΩ resistor on the board as per IEEE 1532 requirements. This prevents totem pole current on the input buffer.
JTAG_TDO	Output	No	JTAG test data out. DNC in unused condition.

For more information about JTAG, see *AC439: Board Design Guidelines for RTG4 FPGA Application Note*.

2.3.1 JTAG Timing

Proper operation of JTAG programming depends on the timing relationship between JTAG pins as shown in the following figure. For the recommended timing values, see the 1532 timing characteristics table of the *DS0131: RTG4 FPGA Datasheet*.

Figure 3 • **JTAG Signals Timing Diagram**



2.3.2 JTAG Programming Architecture

The system controller implements the functionality of a JTAG slave and complies to IEEE 1532 and IEEE 1149.1 standards. The JTAG port communicates with the system controller using:

- a command register that sends the JTAG instruction to be executed.
- a 128-bit data buffer that transfers any associated data.

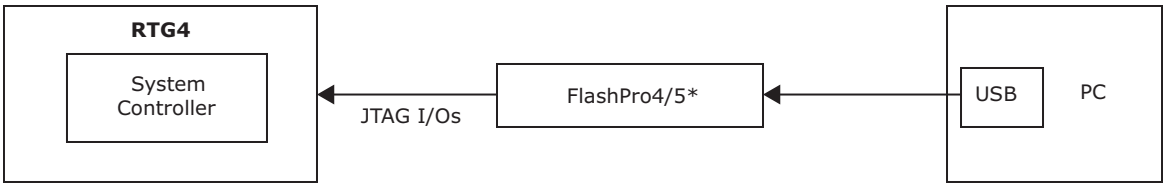
2.3.3 Design Implementation

The FPGA fabric and the µPROM can be programmed using JTAG programming mode.

- During programming, the board must provide power to the following pins: VPP, VDD, and VDDI3. VDDI3 provides power to the JTAG circuitry. For information about voltage ranges, see the *DS0131: RTG4 FPGA Datasheet*.
- An USB-based FlashPro4/FlashPro5 programmer can be used to program the RTG4 device using the dedicated JTAG interface. Libero SoC (or standalone FlashPro Express) executes the programming from a PC connected to the programmer.

The following figure shows the FlashPro4/5 programmer connected to the JTAG ports of a RTG4 device. Only this programming mode is supported by the current version of the Libero SoC software.

Figure 4 • **JTAG Programming Mode**



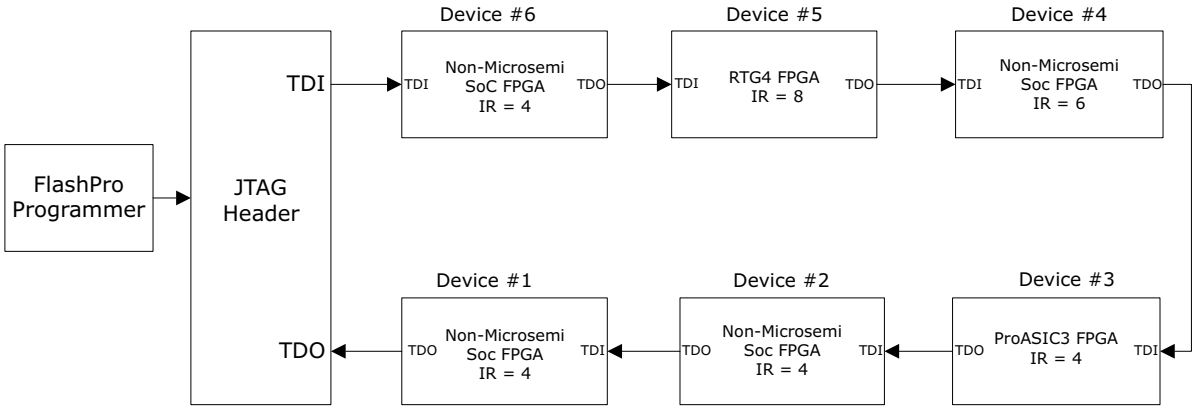
*The FlashPro3 programmer can be used in JTAG programming mode but it has been discontinued.

A single FlashPro4/5 programmer can also program multiple Microsemi FPGAs from the same family or from different families in a single JTAG chain.The TDO pin of the JTAG header represents the beginning of the chain. The TDI pin of the last device is connected back to the JTAG header, thus completing the chain.

A non-Microsemi FPGA can also be included in the chain as shown in the Figure 5, page 6. When a Microsemi FPGA within a chain is programmed, all non-Microsemi FPGAs in the chain are placed in bypass mode. When a device is in bypass mode, the device's data register length is automatically set to 1 and the device stops responding to any programming instructions.

For more information about JTAG chain programming, see the *FlashPro Express User Guide*.

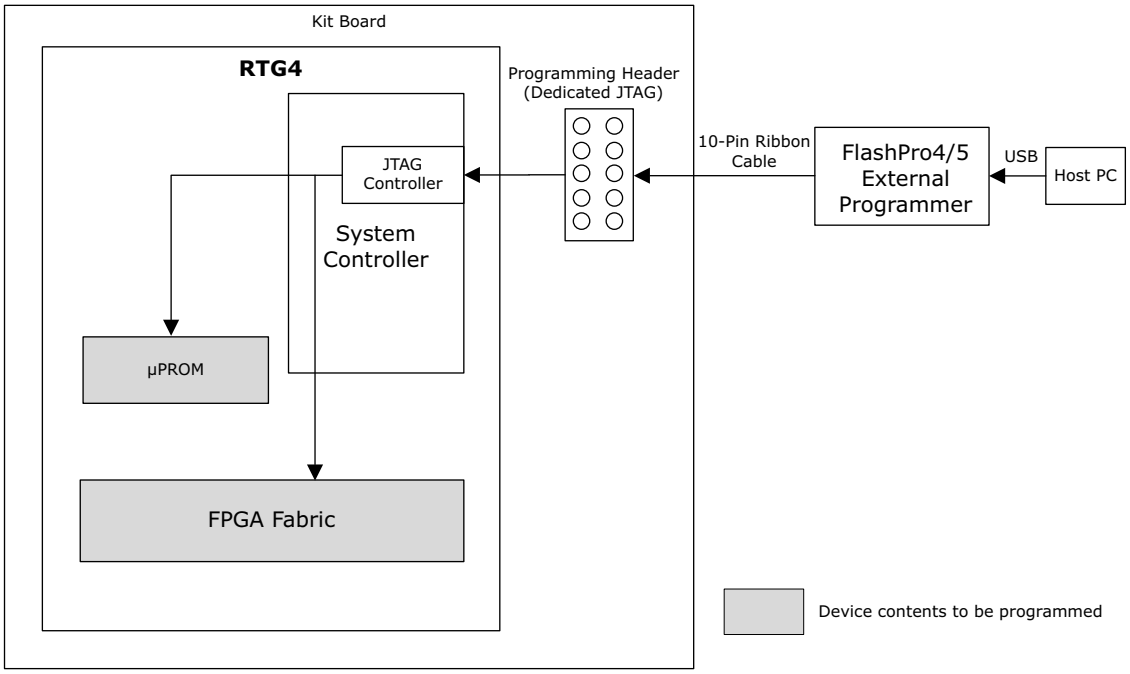
Figure 5 • **Device Programming in JTAG Chain**



2.3.4 Programming Using an External Programmer

RTG4 devices can be programmed using an external programmer through the dedicated JTAG port. When programming with the Libero SoC or standalone FlashPro Express software, FlashPro4/5 cable must be connected to the JTAG pins of the device through a 10-pin programming header, as shown in the following figure. The target board must provide power to the VPP, VDD, and VDDI3 pins.

Figure 6 • **JTAG Programming using External Programmer**



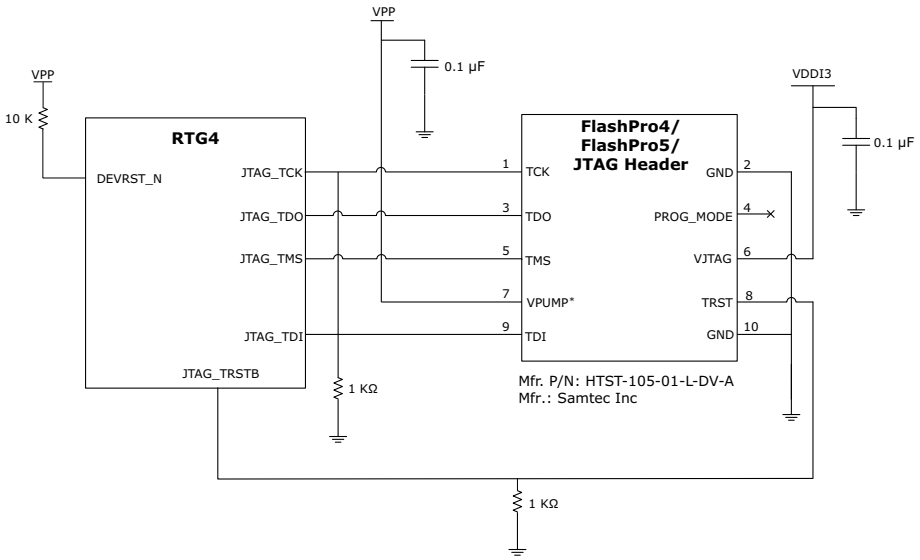
2.3.4.1 JTAG Power Supply Requirements

The power supplies must conform to the specification as defined in the *DS0131: RTG4 FPGA Datasheet*. Measures must be taken to ensure signal integrity, so that power supplies and JTAG signals are free from noise.

RTG4 FPGAs require a single programming voltage to be applied at the VPP pin during programming. This voltage must be supplied from the board. When the device is not programmed under normal operating conditions, VPP must be connected to its supply range. For more information about VPP tolerance range, see the *DS0131: RTG4 FPGA Datasheet*. The board must supply JTAG voltage to the VDDI3 pin of the device and the VJTAG pin of the programmer header.

Microsemi recommends that VPP and JTAG power supply lines (VDDI3) are kept separate with independent filtering capacitors. See the following figure for capacitor requirements. The bypass capacitor must be placed within 2.5 cm of the device pins. The following figure shows the connections between the programming header and the device.

Figure 7 • **JTAG Programming of a Single RTG4 Device**



Note: From Libero SoC v11.8 SP3 and later, FlashPro does not detect or drive VPUMP for SmartFusion2, IGLOO2, and RTG4 devices. FlashPro detects and drives VPUMP only when ProASIC3/IGLOO/Fusion/SmartFusion device is detected in the chain.

- FlashPro does not detect VPP on SmartFusion2, IGLOO2, and RTG4 devices. The VPUMP pin on these boards must be left floating.
- If the board contains ProASIC3/IGLOO/Fusion/SmartFusion along with SmartFusion2/IGLOO2/RTG4 in the JTAG chain, then connect VPUMP of the ProASIC3/IGLOO/Fusion/SmartFusion to the JTAG header of the VPUMP pin.

2.4 State of RTG4 Components During Programming

The following table lists the state of each ASIC block and I/O type during the programming mode described in this user guide. You can configure the I/O state during JTAG programming in Libero SoC, as shown in the following figure. For more information about device I/O states during programming, see the *Libero SoC User Guide*.

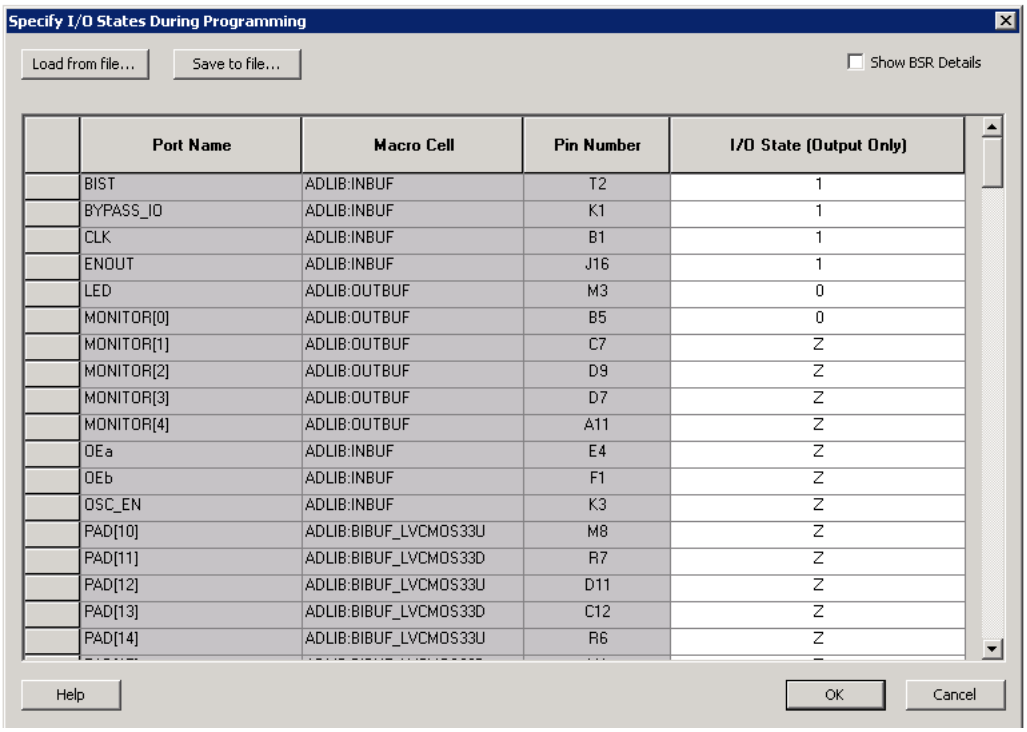
Note: I/Os are tristated in the blank devices, that is, factory shipped unprogrammed devices.

Table 4 • **ASIC Block and I/O State During Programming**

	Programming Methodology
Components	**JTAG**
Shared I/O (fabric/FDDR)	Tristated by system controller
Dedicated Fabric I/O	Tristated by system controller
SerDes I/O	Unaffected by programming
FDDR Block	Fabric interfaces gated off by a flash bit (so no transactions can occur at config APB, AHB/AXI interfaces to fabric)
SerDes Block	Fabric interfaces gated off by a flash bit (so no transactions can occur at config APB, AHB/AXI interfaces to fabric)

The following figure shows the various I/O states of RTG4 components during programming.

Figure 8 • **I/O States During Programming**



2.5 Digest

Digest is a 32-bit Multiple Input Shift Register (MISR) CRC code generated by a hardware circuit in the System controller. It is generated for Fabric/µPROM of the FPGA. Digest is used for protecting the data integrity.

The digest is printed during bitstream generation and bitstream programming. When a user creates a design in Libero and then exports the FlashPro Express job, the fabric digest is printed in the Libero log window and saved in a digest file under the export folder. The digest file is a text file containing the 32-bit digest value. The name of the digest file will match the name of the FlashPro Express job exported, and will be appended with a ".digest" extension.

The digest is also printed at the end of programming operation in the Libero and FlashPro Express log. The user can compare this printed digest with the same in the exported digest file to ensure the intended programming of the FPGA.

2.5.1 Digest Check – VERIFY_DIGEST

During programming operation, the digest is calculated simultaneously and stored in the fabric user segment. When VERIFY_DIGEST is performed, the digest is recalculated and compared against the stored value in the fabric user segment. This compare ensures the image programmed in the device has not changed since the last time the device was programmed.

There is no digest check on power-up, and a digest check (that is, VERIFY_DIGEST) has to be initiated through Libero or FlashPro Express.

Note: It is recommended to keep the track of how many times VERIFY_DIGEST and/or verify are performed as Libero SoC does not keep a track of the count (impacts the data retention limit). For more information about maximum number of counts, see *DS0131: RTG4 FPGA Datasheet*.

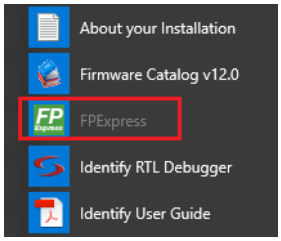
2.5.1.1 Use Model

Use the VERIFY_DIGEST setting if there is any concern about bitstream modification. User may consider disabling this function for security concerns. See Figure 14, page 14 on how to disable this action in Libero.

To perform a VERIFY_DIGEST,

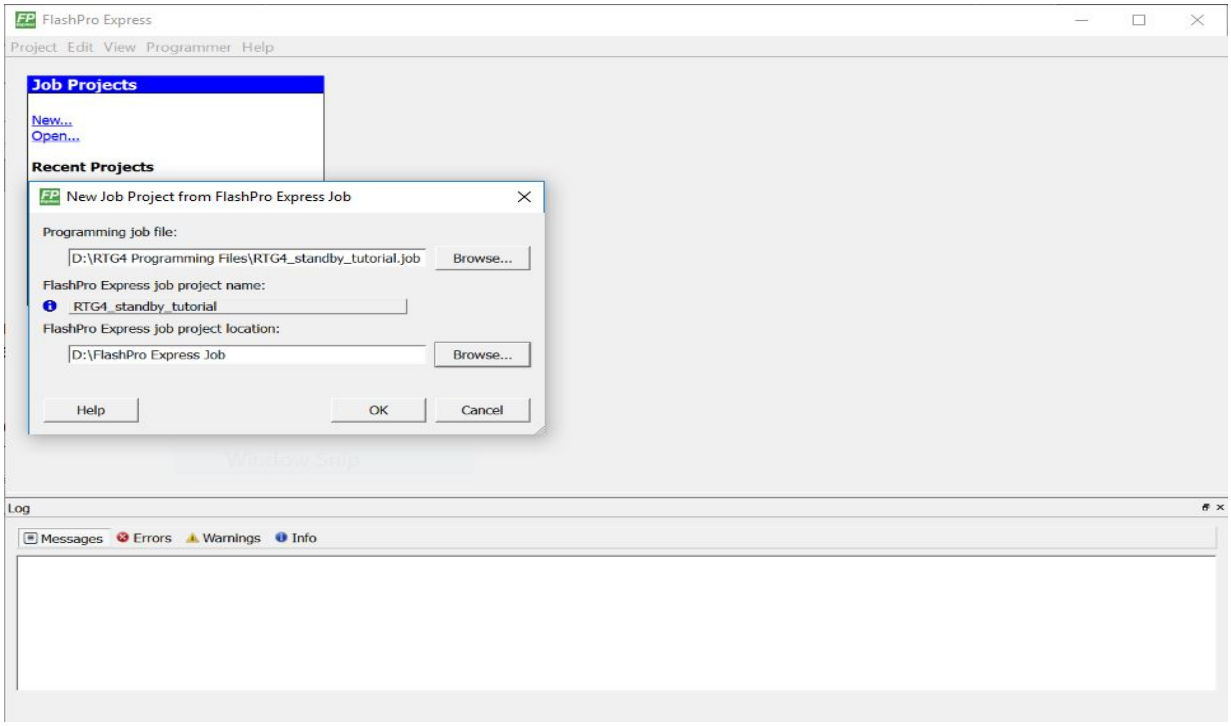
1. Go to **Start** > **Microsemi** > **FP Express.** The **Libero SoC v12.0** window is displayed.

Figure 9 • **Launching FlashPro Express**



2. Click **New** or select **New Job Project** from FlashPro Express Job from Project menu to create a new job project.
3. Enter the following in the New Job Project from FlashPro Express Job dialog box:
   - Programming job file: Click **Browse**, and navigate to the location where the .job file is located and select the file.
   - FlashPro Express job project location: Click **Browse** and navigate to the location where you want to save the project.
4. Click **OK**.

Figure 10 • **Configuring the Digest Check**

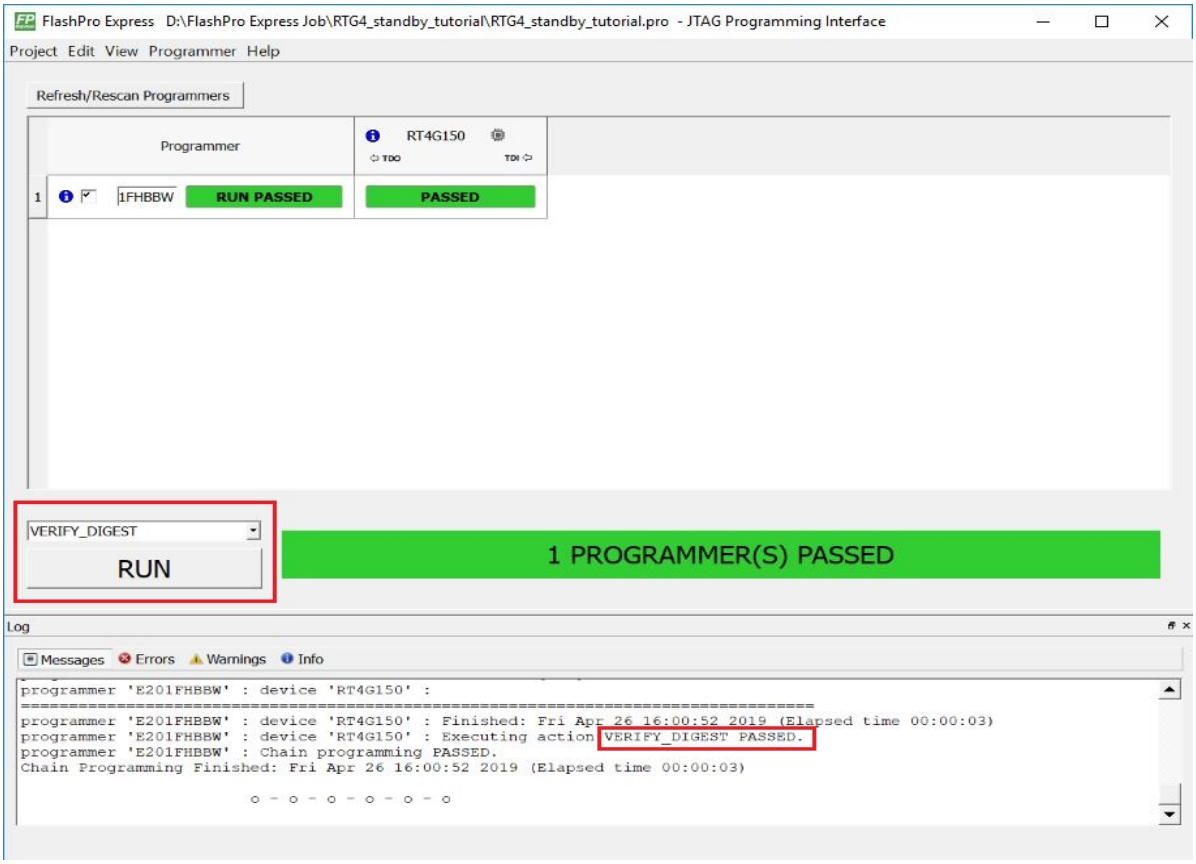


5. From the Drop Down select **VERIFY_DIGEST** from the list.

Figure 11 • **Verifying Digest**

6. Click **VERIFY_DIGEST**. The log window displays the digest check status.

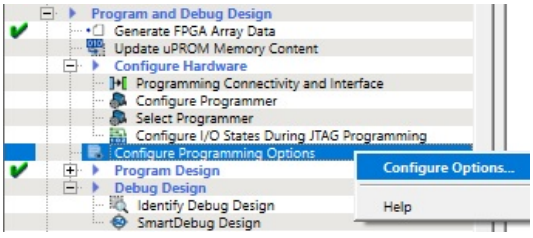
Figure 12 • **Verifying Digest – Log Details**



To disable a digest check:

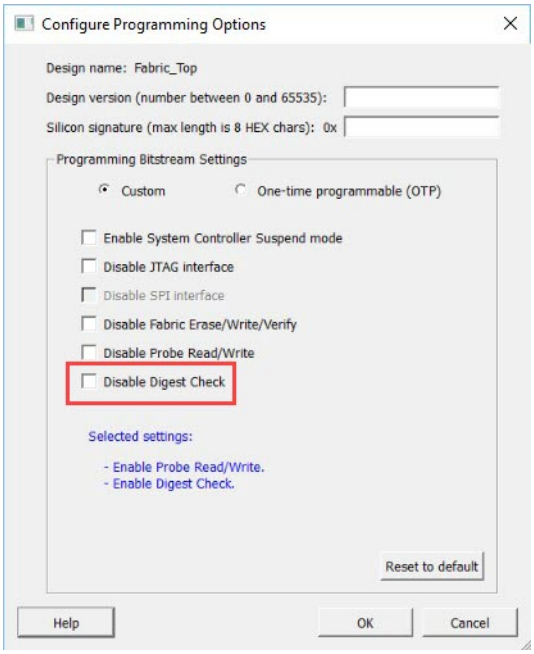
1. In the **Design Flow** window, double-click **Configure Programming Options** or right-click and click **Configure Options**.

Figure 13 • **Configuring Programming Options**



2. Select the **Disable Digest Check** check box under **Programming Bitstream Settings**.

Figure 14 • **Disabling Digest Check**



Note: Check this box disable verify and read of digest in DEVICE_INFO. To disable the digest check, you need to first program the device with this option and power cycle the device with TRSTB pin tied to LOW.

2.5.2 Digest Check – DEVICE_INFO

DEVICE_INFO action is available in both Libero and FlashPro Express software. Design name, Design checksum and Fabric digest are printed in the DEVICE_INFO log. These can be manually compared to the checksum in the STAPL file and digest in the .digest file to ensure the device is programmed with the intended design. There is no restriction on number of times DEVICE_INFO is performed (that is, no impact on retention limit).

2.5.2.1 Use Model

DEVICE_INFO is a quick way to check the status of a programmed device and to ensure that the intended design has been programmed into the device. DEVICE_INFO also prints Cycle Count in the log file, which can be used to track the retention limit as shown in the following figure.

Figure 15 • **DEVICE_INFO – Log Details**