

Introduction [\(Ask a Question\)](#)

Microchip's SmartFusion[®] 2, IGLOO[®] 2 and PolarFire[®] FPGA families, and the Libero[®] SoC Design suite have the novel capability to generate cryptographic grade digests over the user created design and device programming files. These digests are used to check the integrity of programmed devices and programming files, and allow traceability of programmed device content. This white paper describes how these digests are used to verify the content of a programmed Microchip FPGA device and ensure the integrity of the design.

Table of Contents

Introduction.....	1
1. Microchip FPGA Digests.....	3
2. Libero SoC Design Suite Generated Digest Files.....	4
3. Device Generated Digest Files.....	5
4. JTAG Device Access and Digests.....	6
5. Power-on Reset (POR) and On-Demand Digest Checks.....	7
6. Digest Definitions.....	8
7. Programming File Tracking using Cryptographic Digests.....	10
7.1. Device Specific Digests.....	10
7.2. Use Models.....	10
8. Conclusion.....	14
9. Revision History.....	15
Microchip FPGA Support.....	16
Microchip Information.....	16
Trademarks.....	16
Legal Notice.....	16
Microchip Devices Code Protection Feature.....	17

1. Microchip FPGA Digests [\(Ask a Question\)](#)

The FPGA design process culminates in the generation of a device programming file used to program the FPGA device. The Libero SoC Design suite generates a set of digests during programming file generation. These digests include an overall digest value covering all data to be programmed into the device, as well as individual digests of each programmable component within the device. This includes the FPGA fabric, internal nonvolatile memories and the security settings for the design. Each digest is a SHA-256 hash of a device component data set. Digests generated by the Libero SoC Design suite are referred to as bitstream payload digests. These digests are calculated from the encrypted bitstream data. This encrypted bitstream contains the component programming data along with the additional metadata required to program the device. These digests are unique per user design and are used to verify whether an FPGA device was programmed with a specific programming file.

During device programming, the FPGA device system controller also calculates digests over the programmed device data and stores these in the FPGA. Digests generated by the system controller are known as component digests. These digests are calculated over the programmed component information only and do not include any programming file metadata. As a result, digest values generated by the device system controller do not match the digest values generated by the Libero SoC Design suite. These digests can later be used to verify whether the correct bitstream was programmed into the device. They can also be used as an integrity check mechanism to ensure that there are no failed device flash bits or flash bit states, altered through environmental effects or malicious attacks. This check is enabled to run automatically at FPGA power-on or on-demand by the user.

2. Libero SoC Design Suite Generated Digest Files [\(Ask a Question\)](#)

The Libero SoC Design suite generates digest files during bitstream and programming file generation. Perform the following steps in the Libero SoC Design flow to generate the digests::

1. **Generate Bitstream:** This design flow step generates a `<top_level>.ppd` programming file which can be used to directly JTAG program an FPGA device from the Libero SoC Design suite. The completion of this step results in the generation of a `<top_level>_generateBitstream.log` file. This file contains the bitstream payload digests of all FPGA components enabled by the user. The file also contains the user defined design name and design version, as well as the FPGA fabric checksum.
2. **Export Bitstream:** This design flow step is used to generate standalone device programming files. These files are used by JTAG, DirectC and SPI programming methods, which are available to program the FPGA device. Various file formats can be generated including: `.job`, `.stp`, `.dat`, `.spi`, `.ppd` and `svf`. For each file type generated, an associated `<top-level>_<file_type>.digest` file is generated, which contains the bitstream payload digests of all FPGA components enabled by the user. These files also contain the user defined design name and design version, as well as the FPGA fabric checksum.

This step results in a `<top_level>_exportBitstream.log` file that contains the bitstream payload digests for all generated programming files. The file also contains the user defined design name and design version, as well as the FPGA fabric checksum.

These generated log and digest files are used to verify whether a device is programmed with the intended design file, and to check if the programmed device content has been altered from the original programming configuration.

3. **Device Generated Digest Files** [\(Ask a Question\)](#)

SmartFusion 2, IGLOO 2 and PolarFire FPGA family of devices generate digests when programmed by a bitstream. These component digests, which are generated by the device system controller, are stored securely. These component digests can be used by the FPGA device to self-verify that the programmed content of the device has not changed since programming. The digests can also be exported from the device for inspection.

4. JTAG Device Access and Digests [\(Ask a Question\)](#)

JTAG actions have the capability to report both programming file bitstream payload digests and FPGA-generated component digests. The following JTAG actions result in a log file that reports digests and other valuable artifacts of the device state:

- **PROGRAM:** JTAG programming the device results in the generation of a `<top_level>_PROGRAM.log` file. This log file contains the bitstream payload digests from the programming file and the component digests generated by the FPGA device during the programming process. This log also contains the user-defined design name, design version, FPGA fabric checksum and the FPGA Device Serial Number (DSN). This is an important file that helps you to verify whether a specific device was programmed by a specific bitstream, and allow traceability to the programming file. This file provides direct mapping of the bitstream generated by the Libero SoC Design suite to the component digests generated by the FPGA device.
- **DEVICE_INFO:** This JTAG action returns a report of the current state of the FPGA. The action results in the generation of a `<top_level>_DEVICE_INFO.log` file. This log file includes the component digests, user-defined design name, design version, FPGA fabric checksum and the FPGA DSN. This action is useful to understand the current state of the device and help trace back the programmed device content to the actual programming file used for programming. This action is non-intrusive to the FPGA flash array; this means that there is no wear on the device flash configuration cells when executing this action. For more information about the FPGA configuration memory endurance limits, see the respective device datasheet.
- **VERIFY_DIGEST:** This JTAG action instructs the device system controller to regenerate the component digests from the programmed device component configurations and compare these to the component digests stored in the device during device programming. This action results in the generation of `<top_level>_VERIFY_DIGEST.log` file. This file provides a PASS or FAIL result for each FPGA component in the user design, along with the recalculated component digest values. Other data provided in this log includes user-defined design name, design version, FPGA fabric checksum and the FPGA DSN. This action verifies that the device programmed content has not changed since it was programmed into the device. This action must access the device flash configuration cells so the flash cells will experience wear, similar to device programming. For more information about the FPGA configuration memory endurance limits, see the respective device datasheet.
- **VERIFY:** This JTAG action executes a bit-by-bit check of the programming file against the data programmed into the device. The result of this action is the generation of the `<top_level>_VERIFY.log` file. This file provides a simple PASS or FAIL from the result of this comparison to validate if the device is programmed with the expected bitstream. This action must access the device flash configuration cells so the flash cells will experience wear, similar to device programming. For more information about the FPGA configuration memory endurance limits, see the respective device datasheet.

5. Power-on Reset (POR) and On-Demand Digest Checks [\(Ask a Question\)](#)

The digests are used to detect device tampering and ensure device integrity by verifying that the programmed content in the device has not changed due to malicious actions or natural aging and wear.

SmartFusion 2, IGLOO 2 and PolarFire FPGA family of devices can be configured to execute a digest check at power-up or DEVRSTn. After each device power-up or DEVRSTn, the device automatically regenerates the component digests from the programmed device component configurations and compares these to the component digests stored in the device during device programming. If there is any mismatch in this comparison, then a tamper macro flag will be asserted into the FPGA fabric. The user can create logic to handle this event which may include locking down the device to zeroizing the device. Digest checking can also be invoked on-demand by calling the digest check system service (PolarFire) or Non-Volatile Memory (NVM) Data integrity check service (SmartFusion 2 or IGLOO 2). This action must access the device flash configuration cells so the flash cells will experience wear, similar to device programming. For more information about the FPGA configuration memory endurance limits, see the respective device datasheets. Finally, the JTAG action, VERIFY_DIGEST, can be executed to check digest as described earlier in this document.

6. Digest Definitions [\(Ask a Question\)](#)

Following is a list of bitstream payload digests and component digests important to programming verification. For more information, such as for a comprehensive list of all digests available in the device, see the [SmartFusion 2 and IGLOO 2 FPGA Security Best Practices User Guide](#) and the [PolarFire Family Security User Guide](#).

The bitstream payload digests, generated by the Libero SoC Design suite tool for SmartFusion 2 and IGLOO 2 devices, are as follows:

- Security component digest: Digest of all configured device security settings
- Fabric component digest: Digest of the FPGA fabric configuration
- eNVM0 component digest: Digest of all ROM content of eNVM0
- eNVM1 component digest: Digest of all ROM content of eNVM1
- Entire bitstream digest: Combined digest of all components

The bitstream component digests, generated by the SmartFusion 2 or IGLOO 2 device system controller, are as follows:

- Fabric digest: Digest of the FPGA fabric configuration.
- eNVM0 digest: Digest of all ROM content of eNVM0
- eNVM1 digest: Digest of all ROM content of eNVM1
- User security policies digest: Digest of configured security policies
- User key set x digest: Digest of configured user encryption key sets
- Factory row and factory key digest: Digest of factory key
- Fabric configuration digest: Digest of fabric configuration settings

The bitstream payload digests, generated by Libero SoC Design suite tool for PolarFire family devices, are as follows:

- Security component digest: Digest of all configured device security settings
- Entire bitstream digest: Combined digest of all components
- eNVM component digest: Digest of all ROM content of eNVM (PolarFire SoC only)
- sNVM component digest: Digest of all ROM content of sNVM
- Fabric component digest: Digest of the FPGA fabric configuration

The bitstream component digests, generated by PolarFire family device system controller, are as follows:

- Fabric digest: Digest of fabric configuration
- Fabric configuration data digest: Digest of fabric configuration parameters such as cycle count, design version and back level protection value
- sNVM ROM pages digest: Digest of sNVM pages marked as ROM
- User security segment digest: Digest of user security settings
- UKDIGEST0: Digest of user key segment containing SRAM-PUF data
- UKDIGEST1: Digest of user key segment containing KUP (User EC key)
- UKDIGEST2: Digest of user key segment containing UPK1
- UKDIGEST3: Digest of user key segment containing UEK1
- UKDIGEST4: Digest of user key segment containing DPK
- UKDIGEST5: Digest of user key segment containing UPK2

- UKDIGEST6: Digest of user key segment containing UEK2
- UPDIGEST (UPERM Digest): Digest of permanent lock security segments
- FDIGEST (SYS Digest): Digest of factory lock segment, factory key segment in pNVM and System Controller ROM
- UKDIGEST7: Digest of One-Way Passcode HWM (for PolarFire SoC FPGA only)
- ENVMDIGEST: Digest of eNVM (for PolarFire SoC FPGA only)
- UKDIGEST8: Digest of MSS Boot mode Information (for PolarFire SoC FPGA only)
- UKDIGEST9: Digest of SNVM_RW_ACCESS_MAP (for PolarFire SoC FPGA only)
- UKDIGEST10: Digest of Secure Boot Image Certificate (SBIC) (for PolarFire SoC FPGA only)

7. Programming File Tracking using Cryptographic Digests [\(Ask a Question\)](#)

Cryptographic digests generated by the Libero SoC Design suite, and SmartFusion 2, IGLOO 2 and PolarFire FPGA device families, are used to confidently track device programmed content and the programming files used for programming. The following sections explain how digests can help with this tracking.

7.1. Device Specific Digests [\(Ask a Question\)](#)

In general, device component digest generated by a particular device family density will be the same for all devices programmed with the same programming file. However, there are some exceptions:

- For the PolarFire Family of devices, any sNVM user data storage pages set up as ROM and as authenticated or authenticated/encrypted data, results in the device-generated sNVM ROM pages digest to be unique for each device. This is because the information on these pages will be protected by a device-specific PUF. Also, user and factory key segments data is protected by the PUF, resulting in unique, per device digests for these segments.
- SmartFusion 2 and IGLOO 2 060/090/150 density devices have user and factory key sets protected by device-specific PUFs. These digests will be unique per device. The 060 density of these devices include some device specific information in the calculation of the eNVM0 digest. This results in a unique eNVM0 digest per device.



Important: These exceptions do not impact device integrity checking because this process checks the programmed device content against the digest stored in the device during programming. Here device specific digests have no impact. These exceptions can impact some of the programming file tracking use models, as described in the following sections. These will be noted in the use models.

7.2. Use Models [\(Ask a Question\)](#)

This section provides detailed information about the programming file tracking use models.

7.2.1. Contract Manufacturer (CM) Programming Verification [\(Ask a Question\)](#)

Users often use a Contract Manufacturer (CM) to program FPGA devices. You can verify which bitstream file was programmed on the devices by running a JTAG VERIFY action on each device that was programmed. This is a costly and time-consuming process that also wears the FPGA flash cells. A more efficient process is using digest files generated during the programming file generation and programming of devices. These digests are compared to verify that all the devices were programmed with the correct bitstream file.

The bitstream file content is divided into three major component sections:

- FPGA fabric
- NVM
- Security

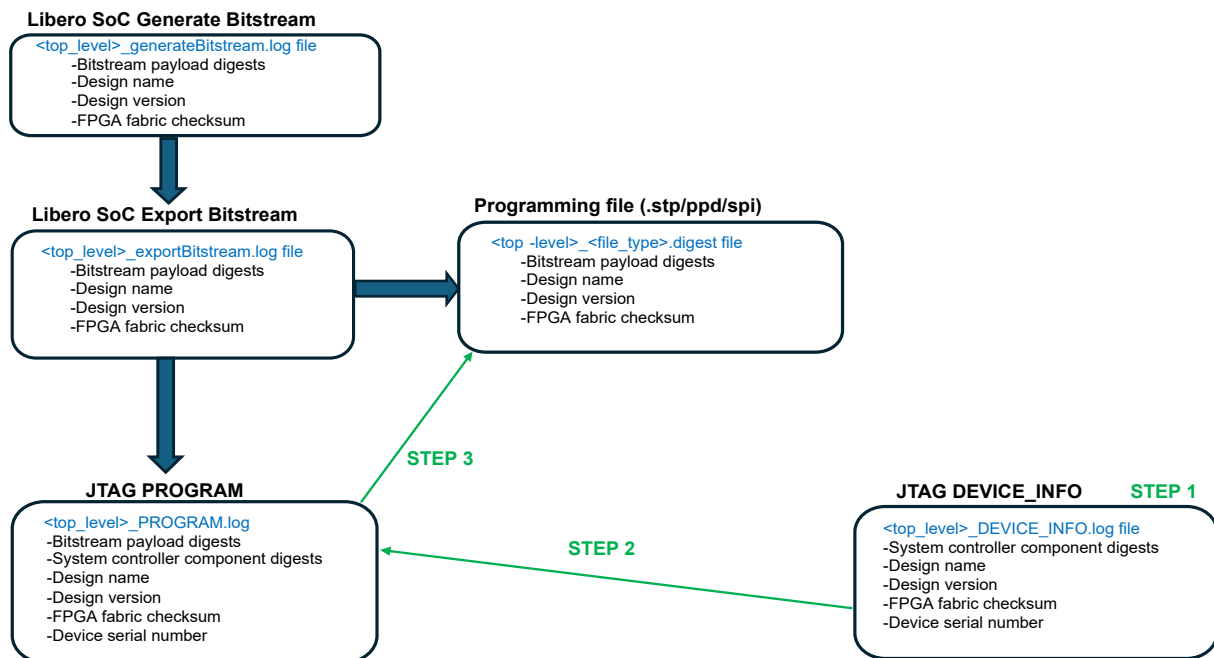
A valid bitstream contains a combination of any of the three preceding primary bitstream components. When a user creates a design in the Libero SoC Design suite and exports the `STAPL` file (for FlashPro) or programming `job` (for FlashPro Express), the bitstream payload digest of each of the primary components used in the design is saved into the log and `.digest` files as described earlier. The customer sends the `STAPL/programming job` files to a contract manufacturer for device programming. When the devices are programmed, the JTAG PROGRAM action log file is generated, which contains the bitstream payload digests, component digests and DSN. The contract manufacturer saves these log files and sends the devices, along with log files, back to the customer.

The JTAG PROGRAM action log files are important because they provide a direct mapping of the bitstream payload digests to the component digests. The customer can verify that the correct design was programmed on the device by matching the digests in the log file. The process would be as follows:

1. Obtain the DSN and the component digests programmed into the device by running a JTAG DEVICE_INFO action on the CM programmed FPGA.
2. Verify that associated PROGRAM log file DSN and component digests match the values returned by the DEVICE_INFO action.
3. Verify that the PROGRAM log file bitstream payload digest matches the bitstream payload digest, provided by the `<top-level>_<file_type>.digest` files generated during the Export Bitstream process used to create the programming file.

This process flow is shown in the following figure.

Figure 7-1. Contract Manufacturer (CM) Programming Verification



Important: There are methods to securely program devices and generate reports from an untrusted contract manufacturer by employing Microchips Secure Production Programming Solution (SPPS). This solution requires the use of Hardware Security Modules (HSM) at the FPGA design house and at the untrusted contract manufacturer. For more information contact Microchip.

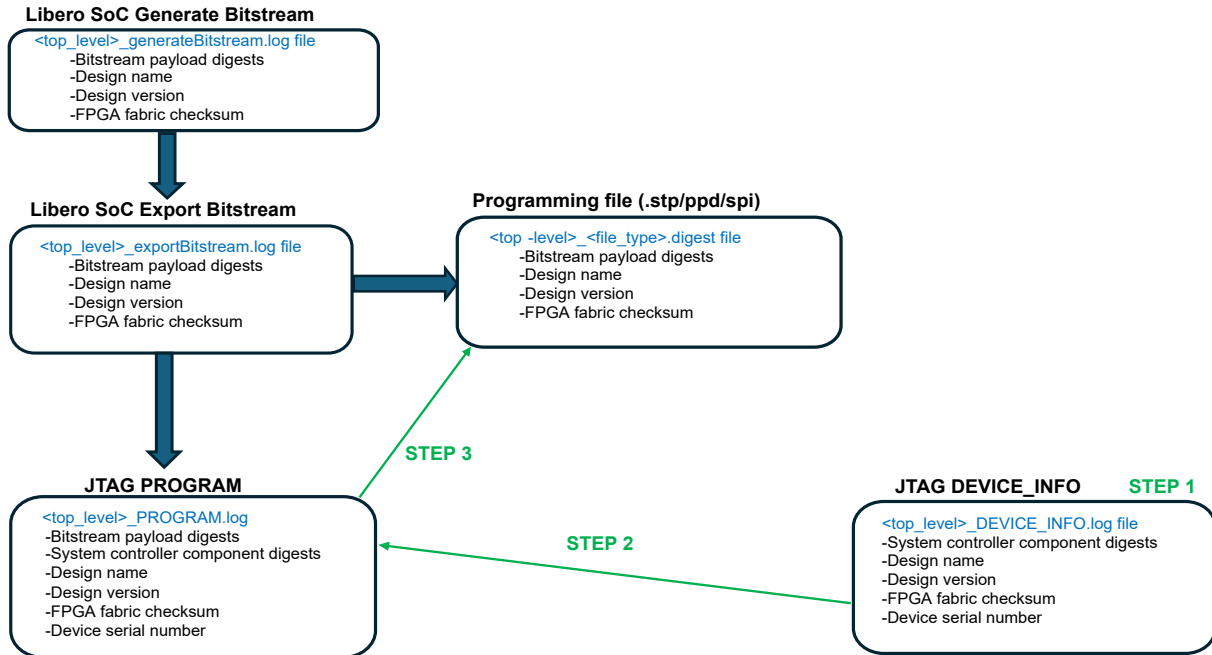
7.2.2. Internal Development/Return Material Authorization (RMA) Device Tracking [\(Ask a Question\)](#)

It is desirable for designers to be able to track the programmed content of devices under development or devices returned from the field (RMA). The approach described, in the preceding section, for CMs are used in these cases, provided the device was programmed using JTAG. There can be some relaxation of the requirement for needing the exact PROGRAM action log used to program the specific device to track the DSN. It may be sufficient in these cases to just get a high confidence of what is programmed into the device. In this case, any PROGRAM action log generated using the bitstream suspected to be programmed into the device will do. DSN matching will fail but digests should match. A similar process is then used:

1. Obtain the component digests programmed into the device by running a JTAG DEVICE_INFO action on the programmed FPGA.
2. Verify the PROGAM log file component digests match the values returned by the DEVICE_INFO action.
3. Verify that the PROGAM log file bitstream payload digest matches the bitstream payload digest provided by the `<top-level>_<file_type>.digest` files generated during the Export Bitstream process used to create the programming file.

To add more confidence, the user may also compare the design name, design version and FPGA fabric Cyclic Redundancy Check (CRC) provided in these files.

Figure 7-2. Internal Development/RMA Device Tracking



Due to the exceptions described in section [Device Specific Digests](#), PolarFire devices with sNVM user data storage pages, configured as ROM and as authenticated or authenticated/encrypted data, as well as SmartFusion 2 or IGLOO 2 060 density devices, need the programming log file for the specific device to verify if a particular bitstream was programmed into them.

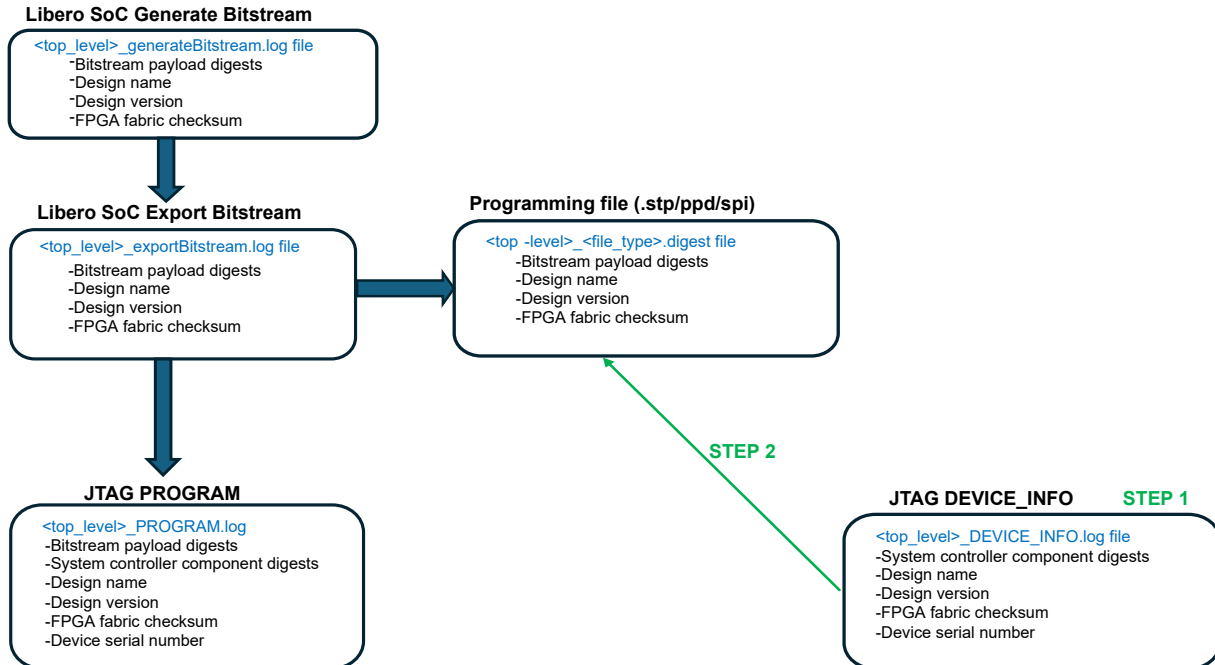
7.2.3. IAP/SPI Initiator Device Programming [\(Ask a Question\)](#)

SmartFusion 2, IGLOO 2 and PolarFire FPGA family of devices have the option to auto-program themselves from an attached SPI flash memory device (IAP/AutoUpdate/AutoProgram). When this process is used, there is no PROGRAM action log file generated, so the direct mapping of the bitstream payload digests to the component digests is lost. Without this artifact, tracking which design is programmed into the device becomes more challenging.

One option to solve this challenge is to JTAG program a surrogate device to generate the PROGRAM action log file artifact. With this file available, the processes described in the previous sections can be followed. A second option is to closely manage the design name and design version in the designs and use those fields, along with the FPGA fabric checksum, to track the design programming instead of the digests. The FPGA fabric checksum is a 16-bit CRC that has relatively low collision resistance, but using this value along with the user-created design name and design version, should provide sufficient information to accurately determine the programming file used to program the device. Device programming file tracking, using this second option, would follow this process:

1. Run a JTAG DEVICE_INFO action on the FPGA to obtain design name, design version and FPGA fabric checksum programmed into the device.
2. Compare this information with the information available in the `<top-level>_<file_type>.digest` files generated during the Export Bitstream process used to create the programming file.

Figure 7-3. Tracking Programming Files through Use of Design Name, Design Version and Fabric Checksum



Because of the expectations described in section Device Specific Digests, PolarFire devices with sNVM user data storage pages, configured as ROM and as authenticated or authenticated/encrypted data, and SmartFusion 2, IGLOO 2 060 density devices will require the programming log file for the specific device to verify if a specific bitstream was programmed into these devices.

7.2.4. Multiple Programming Files [\(Ask a Question\)](#)

The Libero SoC Design suite allows flexibility in device programming methodologies. This flexibility comes with a price which is that the user must closely manage programming files during development and device programming. One available programming option is to program different components in the FPGA device, using different programming files. For example, a device may be initially programmed with a programming file containing only security and FPGA fabric content; later, the device eNVM could be programmed alone, where the existing security and FPGA content are not modified. In this scenario, the device has been programmed with two programming bitstreams, which complicates programming file tracking. In this scenario, multiple JTAG PROGRAM log files would need to be compared to the DEVICE INFO log. Designers need to carefully manage device programming files to ensure the correct files are programmed into devices.

If the design name or version methodology is adopted to track device programming files, be aware that the design name or version are programmed into the device with the FPGA fabric content. Programming of eNVM only, for example, would not update these fields in the device. A possible method to track eNVM content in this scenario would be to manually calculate and record the CRC of the eNVM content prior to programming. The user can optionally append this CRC to the eNVM content and program it into the device. This recorded CRC can subsequently be used to verify the content of a specific device's eNVM, by reading the eNVM content with the Libero SoC Design suite built-in SmartDebug tool and comparing it to the previously recorded CRC value.

8. Conclusion [\(Ask a Question\)](#)

Microchip's Libero SoC Design suite and devices offer a wide range of programming options. Digests are used to manage device-programmed content and files. Programmed FPGA device integrity is checked utilizing device-initiated digest checks. Programming files are tracked through digest extracted from the device and programming file logs. A well thought out programming file methodology should be followed to properly manage programming files and programmed device content.

9. Revision History [\(Ask a Question\)](#)

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

Revision	Date	Description
A	12/2024	Initial release

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

ISBN: 979-8-3371-0337-2

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.