



# Generic Flash Programmer User Guide

---

## Intel® Quartus® Prime Standard Edition

Updated for Intel® Quartus® Prime Design Suite: **18.1.1**



**Subscribe**

**Send Feedback**

**UG-20235 | 2019.05.15**

Latest document on the web: [PDF](#) | [HTML](#)



## Contents

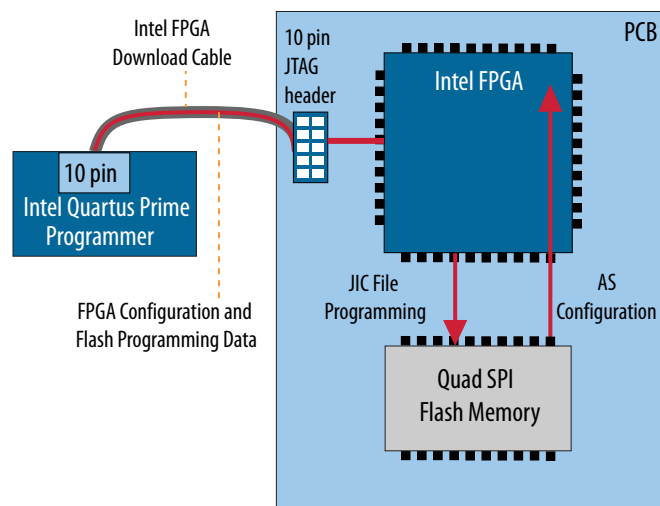
---

<b>1. Generic Flash Programmer User Guide Intel® Quartus® Prime Standard Edition.....</b>	<b>3</b>
1.1. Supported Devices and Configuration Methods.....	4
1.2. Quad SPI Flash Byte-Addressing.....	4
1.3. Generic Flash Programmer Operation.....	5
1.3.1. Generic Flash Programming (Convert Programming File Dialog Box).....	5
1.3.2. Erasing Flash Memory Sectors.....	14
1.4. Generic Flash Programmer Flow Templates (Convert Programming File).....	15
1.4.1. Initialization Flow Templates (Convert Programming File).....	15
1.4.2. Program Flow Template (Convert Programming File).....	16
1.4.3. Erase Flow Template (Convert Programming File).....	17
1.4.4. Verify/Blank-Check/Examine Flow Template (Convert Programming File).....	18
1.4.5. Termination Flow Template (Convert Programming File).....	19
1.4.6. Programming Flow Action Properties.....	20
1.5. Generic Flash Programmer Settings Reference.....	22
1.5.1. Convert Programming File Dialog Box.....	23
1.5.2. Device and Pin Options.....	23
1.5.3. Compression and Encryption Settings (Convert Programming File).....	26
1.5.4. SOF Data Properties Dialog Box (Convert Programming File).....	27
1.6. Generic Flash Programmer User Guide Revision History.....	27

# 1. Generic Flash Programmer User Guide Intel® Quartus® Prime Standard Edition

This document describes how to use the Generic Flash Programmer. You can use the Generic Flash Programmer to load an FPGA configuration bitstream file into a Quad SPI flash memory device. The Quad SPI flash memory device subsequently loads the configuration data into the target FPGA via Active Serial (AS) configuration. You can optionally enable bitstream compression and encryption security to reduce the size and protect the configuration bitstream files.

**Figure 1. Generic Flash Programmer Configuration Example**



The Generic Flash Programmer allows you to send configuration data over a download cable via a JTAG connection to the target FPGA device. The target FPGA then in turn writes the configuration data to the flash memory device. The AS configuration scheme loads the configuration data from the flash memory into the FPGA. For example, this method allows you to configure or reconfigure the FPGA from the flash memory after restoring power to the FPGA after power down.

## Related Information

- [Intel® Arria® 10 Core Fabric and General Purpose I/Os Handbook](#)
- [Cyclone® V Device Handbook: Volume 1: Device Interfaces and Integration](#)
- [Configuration and Remote System Upgrades in Cyclone IV Devices](#)
- [Arria II Device Handbook](#)
- [Arria V Device Handbook: Volume 1: Device Interfaces and Integration](#)
- [Configuring Stratix® III Devices](#)



- Configuration, Design Security, Remote System Upgrades with Stratix® IV Devices
- Intel® Quartus® Prime Standard Edition User Guide: Programmer

## 1.1. Supported Devices and Configuration Methods

The Intel® Quartus® Prime Standard Edition Generic Flash Programmer supports the following FPGA and flash memory devices and configuration methods.

**Table 1. Generic Flash Programmer Device and Configuration Method Support (Intel Quartus Prime Standard Edition)**

	Supported Method
Supported Flash Programming Input File	SRAM Object File (.sof)
Supported Flash Programming Output File	JTAG Indirect Configuration File (.jic)
Supported Configuration Schemes	<ul style="list-style-type: none"><li>• Active Serial</li><li>• Active Serial x4</li></ul>
Supported Devices	Arria, Arria II, Arria V, Cyclone II, Cyclone III, Cyclone IV, Cyclone V, Stratix II, Stratix III, Stratix IV, Stratix V, Intel Cyclone 10 LP, Intel Arria 10

## 1.2. Quad SPI Flash Byte-Addressing

Quad SPI flash devices typically support either 3-byte addressing, 4-byte addressing, or both for programming operations. You can only configure Intel FPGAs with a flash memory device with byte addressing that is compatible with the Intel FPGA that you plan to configure.

The following table specifies the byte-addressing compatibility of Intel FPGAs for supported flash memory devices:

**Table 2. Intel FPGA Required Flash Memory Byte Addressing**

FPGA Devices	Required Flash Memory Byte Addressing
Intel Arria® 10 devices	4-byte addressing
Intel Cyclone® 10 LP devices	3-byte addressing
Arria V, Cyclone V, Stratix® V series devices	<ul style="list-style-type: none"><li>• 3-byte addressing (limited memory address access<sup>(1)</sup>)</li><li>• 4-byte addressing</li></ul>
Cyclone IV, Cyclone II, Cyclone III, Arria GX, Arria II, Stratix II, Stratix III, Stratix IV, Stratix V devices	3-byte addressing

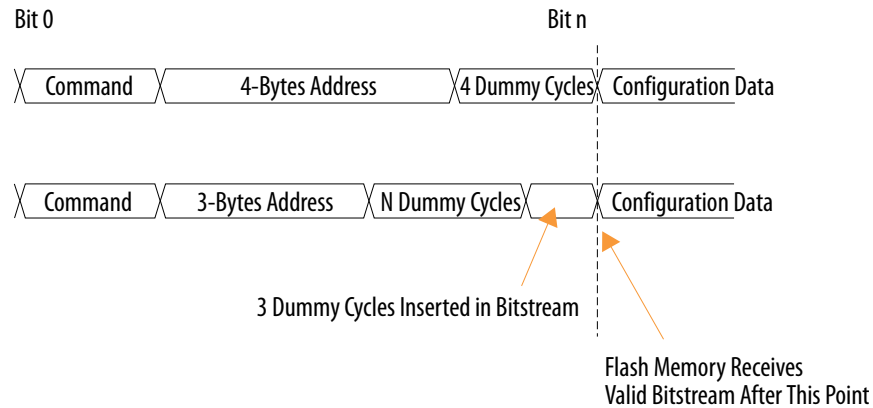
<sup>(1)</sup> Flash devices that exceed 128 megabits (Mb) density require 4-byte addressing to access the memory space higher than 128 Mb. For flash devices that do not support non-volatile, 4-byte addressing setting, the FPGA cannot read the configuration image that has a start address beyond 128Mb. For the remote system update application, the FPGA cannot store images beyond 128Mb.



### Adding Dummy Clock Cycles

Flash memory devices must read either a 24-bit (3-byte) address, or 32-bit (4-byte) address before the flash device can start receiving data to write to the flash memory, or before outputting the data after the flash memory device receives a read command. Therefore, you must specify (or select a flash memory template that specifies) an appropriate dummy clock cycle value for the flash memory device, as [Defining a New Flash Memory Configuration Device](#) on page 11 describes.

**Figure 2. Reading Configuration Data from Flash (3-Byte and 4-Byte Addressing)**



#### Related Information

- [Modifying Programming Flows](#) on page 12
- [Defining a New Flash Memory Configuration Device](#) on page 11

## 1.3. Generic Flash Programmer Operation

The Generic Flash Programmer facilitates the generation of appropriate secondary programming files, with definition of the connected flash memory device, and the corresponding Program, Erase, Verify, Blank-Check, and Examine flows. You can then add the generated programming files to the Intel Quartus Prime Programmer for correct flash programming implementation in hardware.

You can access settings and controls for the Generic Flash Programmer from the Programmer and **Convert Programming File** dialog box.

These user interfaces allow you to generate required programming files, define the flash programming device, and implement your flash programming flow.

Refer to the following configuration steps for the method that you use.

#### Related Information

[Generic Flash Programming \(Convert Programming File Dialog Box\)](#) on page 5

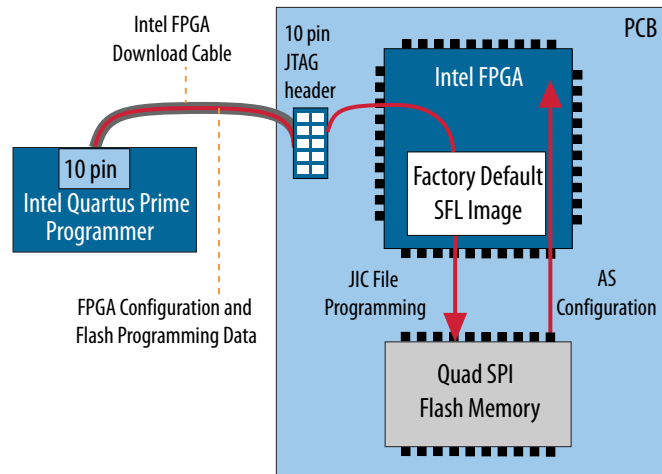
### 1.3.1. Generic Flash Programming (Convert Programming File Dialog Box)

In generic flash programming with the **Convert Programming File** dialog box, you generate the necessary device programming and configuration files, and perform JTAG programming of the flash memory via the Factory default SFL image.

The Factory default SFL image enables communication between the JTAG pins and the flash memory device's serial interface. The SFL is an instance of the Serial Flash Loader Intel FPGA IP that is optimized for this function.

After generating the files, you use the Intel Quartus Prime Programmer to program the flash, which in turn configures the FPGA via AS configuration.

**Figure 3. Flash Programming Configuration (Intel Arria 10 Example)**



Generic flash programming with the **Convert Programming File** dialog box includes the following high level steps that this section describes in detail:

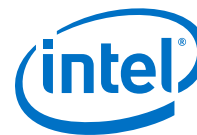
1. [Step 1: Generate Primary Device Programming Files](#) on page 6—use the Intel Quartus Prime Assembler to generate the .sof FPGA configuration file.
2. [Step 2: Generate Secondary Programming Files \(Convert Programming Files\)](#) on page 7—use the **Convert Programming File** dialog box to generate the .jic that you program into your flash memory device to store .sof configuration data.
3. [Step 3: Program the Flash Memory Device](#) on page 13—use the Intel Quartus Prime Programmer and connected Intel FPGA download cable to program the .jic configuration data into the flash memory device and the .sof into the FPGA via Active Serial JTAG configuration.

### 1.3.1.1. Step 1: Generate Primary Device Programming Files

The Intel Quartus Prime Assembler generates the .sof FPGA configuration file once design compilation is complete. Prior to running the Assembler, you can specify device and pin options that impact the .sof and subsequent .jic file generation.

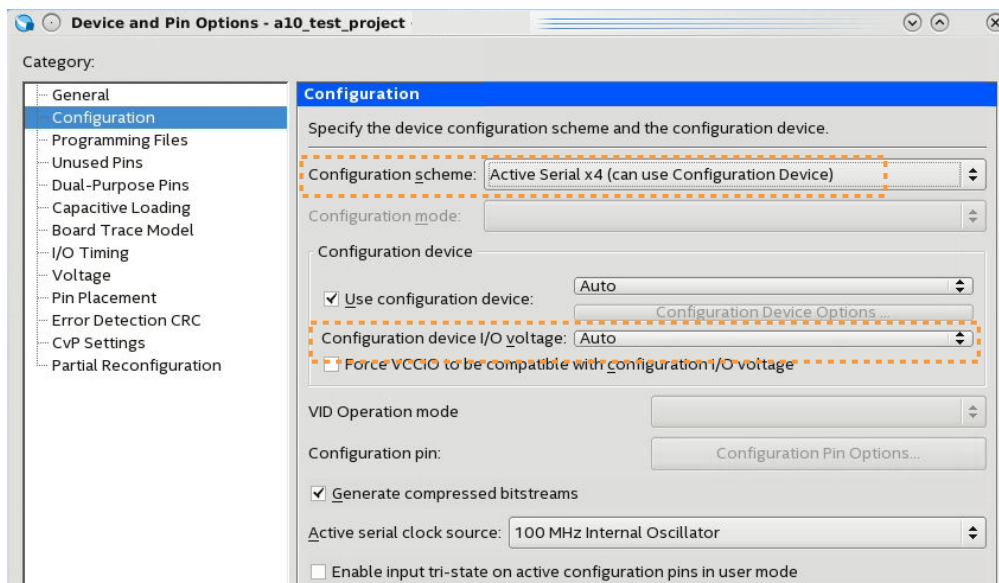
Follow these steps to generate a .sof file for use in generic flash programming:

1. Before running the Assembler, click **Assignments > Device > Device & Pin Options** to specify options for FPGA configuration pins and other hardware settings that the .sof file preserves. The following options are particularly relevant to generic flash programming. For option descriptions, refer to [Device and Pin Options](#) on page 23.



- **General** tab—specify the **JTAG user code**, configuration clock source, and options for specific configuration pins.
- **Configuration** tab—specify **Active Serial** or **Active Serial x4** for the FPGA **Configuration scheme**. Select **Auto** for **Configuration device I/O voltage**, which you can specify with precision at a later time.

Figure 4. Device & Pin Options Dialog Box



2. To generate primary device programming files, click **Processing > Start > Start Assembler**. The Compiler confirms that prerequisite modules are complete, and launches the Assembler to generate the programming files.

### Related Information

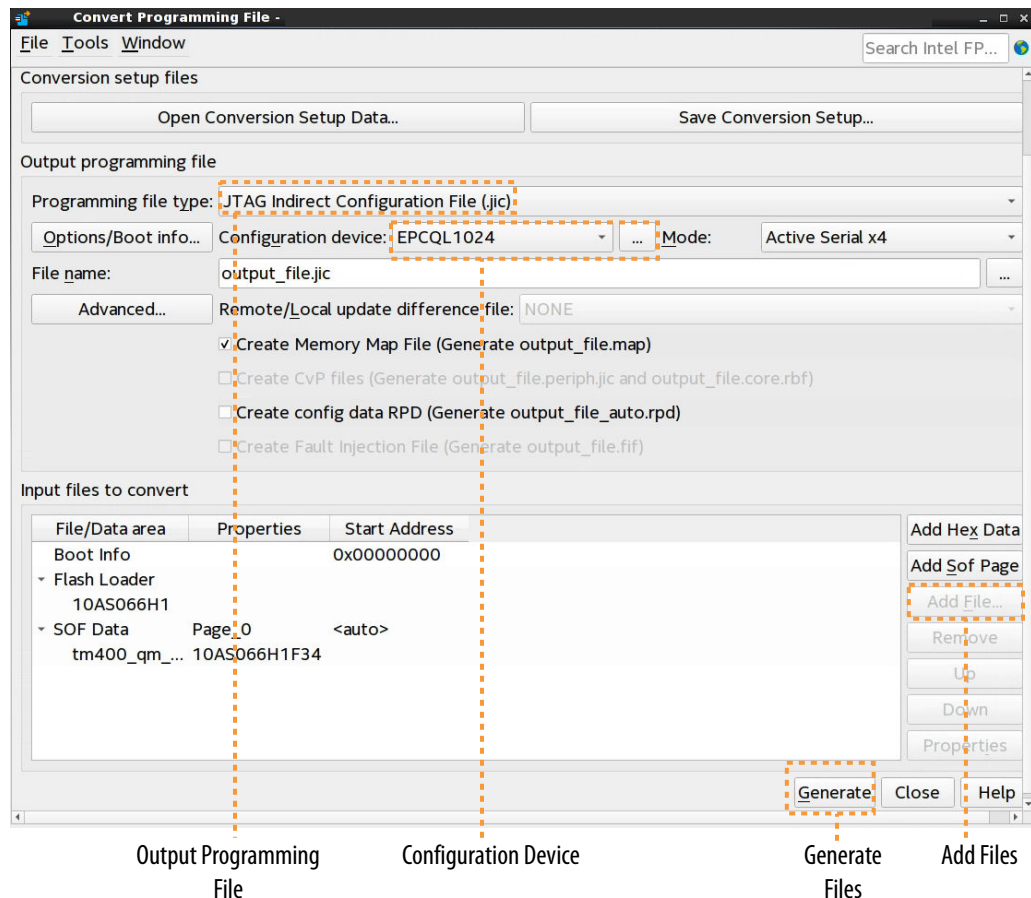
[Device and Pin Options](#) on page 23

#### 1.3.1.2. Step 2: Generate Secondary Programming Files (Convert Programming Files)

You can use the **Convert Programming File** dialog box to generate secondary programming files for alternative device programming methods. For example, generating the .jic file for flash programming, the .rbf file for partial reconfiguration, or the .rpd file for a third-party programmer configuration.

The options available in the **Convert Programming File** dialog box change dynamically, according to your device and configuration mode selection.

**Figure 5. Convert Programming File Dialog Box**

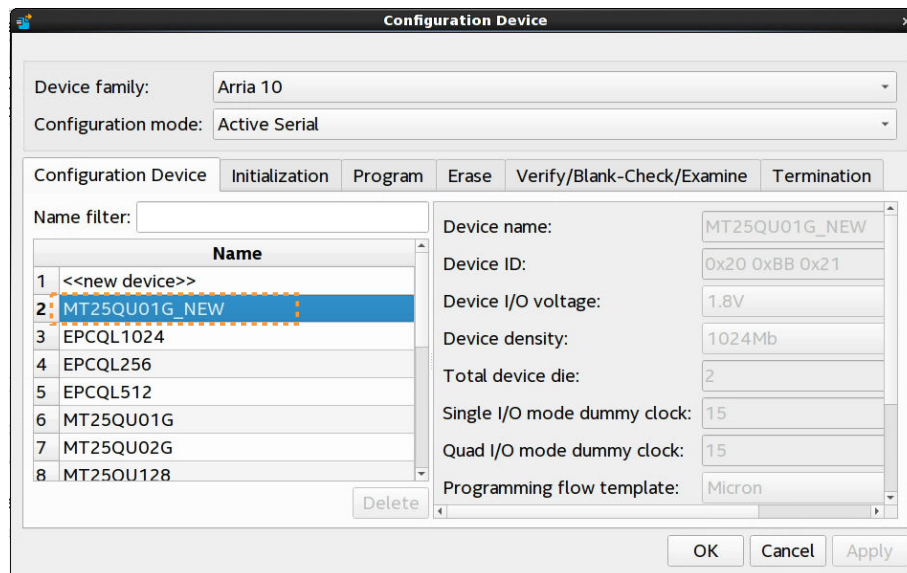


1. Generate the primary programming files for your design, as [Step 1: Generate Primary Device Programming Files](#) on page 6 describes.
2. Click **File ► Convert Programming Files**.
3. Under **Output programming file**, select the **JTAG Indirect Configuration File (.jic)** as the **Programming file type** that you generate. The Generic Flash Programmer supports only this file type.
4. Specify the **File name** and output directory (...) for the .jic file you generate.
5. For the configuration **Mode**, select **Active Serial x4** or **Active Serial**.
6. To specify the **Configuration device**, click the (...) button to select a supported flash memory device and predefined programming flow. When you select a predefined device, you cannot modify any setting. Alternatively, click **<<new device>>** to define a new flash memory device and programming flow, as [Defining a New Flash Memory Configuration Device](#) on page 11, and [Modifying Programming Flows](#) on page 12 describe.



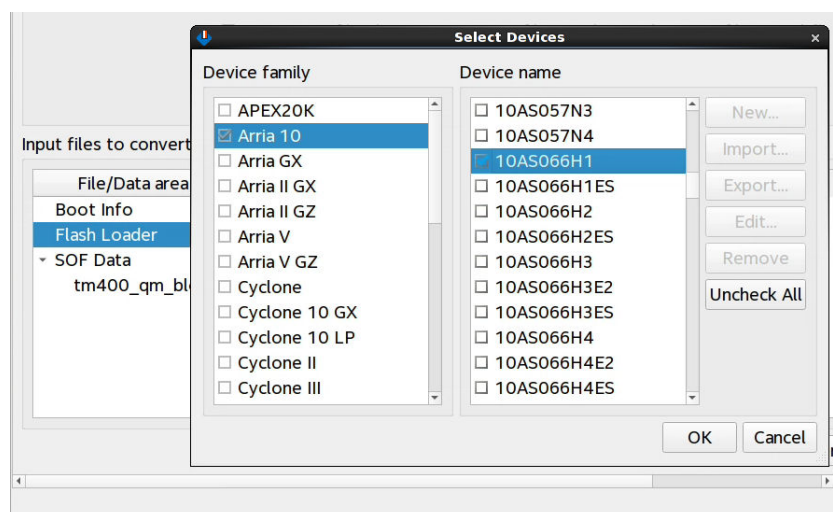


Figure 6. Configuration Device Dialog Box



7. Under **Input files to convert**, select the **SOF Data** item, and then click the **Add File** button. Specify the .sof file that contains the configuration bitstream data. To include raw data, click **Add Hex Data** and specify a .hex file.
8. To enable bitstream compression or encryption security settings, select the .sof file and click **Properties**, as [Enabling Bitstream Compression or Encryption \(Convert Programming File Dialog Box\)](#) on page 10 describes.
9. Select the **Flash Loader** text, and then click the **Add Device** button. Select the device that controls loading of the flash device.

Figure 7. Selecting the Flash Loader Device



10. After you specify all options in the **Convert Programming File** dialog box, click the **Generate** button to create the files.

## Related Information

- [Defining a New Flash Memory Configuration Device](#) on page 11
- [Enabling Bitstream Compression or Encryption \(Convert Programming File Dialog Box\)](#) on page 10
- [Modifying Programming Flows](#) on page 12

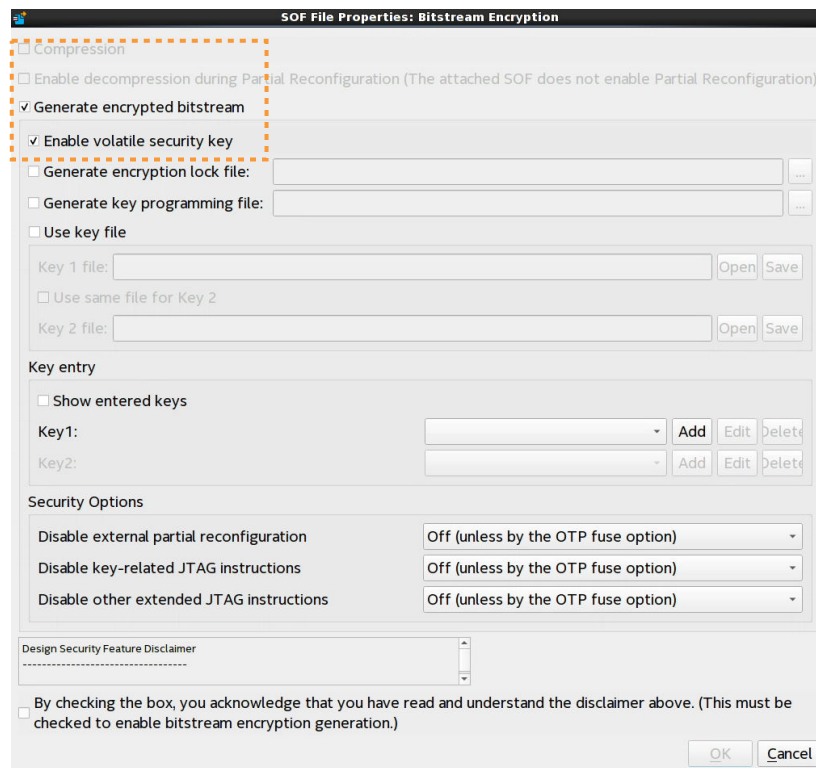
### 1.3.1.2.1. Enabling Bitstream Compression or Encryption (Convert Programming File Dialog Box)

Optionally enable bitstream compression to reduce the size of your programming bitstream file. Enable encryption key programming files and user-defined 256-bit security key to protect and authenticate the configuration bitstream. These options are mutually exclusive.

Follow these steps to enable bitstream file compression or encryption:

1. Generate a .jic file for flash programming, as this document describes.
2. In the **Convert Programming File** dialog box, select the .sof file under **Input files to convert**.
3. Click the **Properties** button. The **SOF File Properties: Bitstream Encryption** dialog box appears.

**Figure 8. Enabling Bitstream Compression or Encryption (Intel Arria 10 and Intel Cyclone 10 GX Designs)**



**SOF File Properties: Bitstream Encryption**

☐ Compression

☐ Enable decompression during Partial Reconfiguration (The attached SOF does not enable Partial Reconfiguration)

☒ Generate encrypted bitstream

☒ Enable volatile security key

☐ Generate encryption lock file:

☐ Generate key programming file:

☐ Use key file

Key 1 file:

☐ Use same file for Key 2

Key 2 file:

**Key entry**

☐ Show entered keys

Key1:

Key2:

**Security Options**

Disable external partial reconfiguration:

Disable key-related JTAG instructions:

Disable other extended JTAG instructions:

**Design Security Feature Disclaimer**

☐ By checking the box, you acknowledge that you have read and understand the disclaimer above. (This must be checked to enable bitstream encryption generation.)



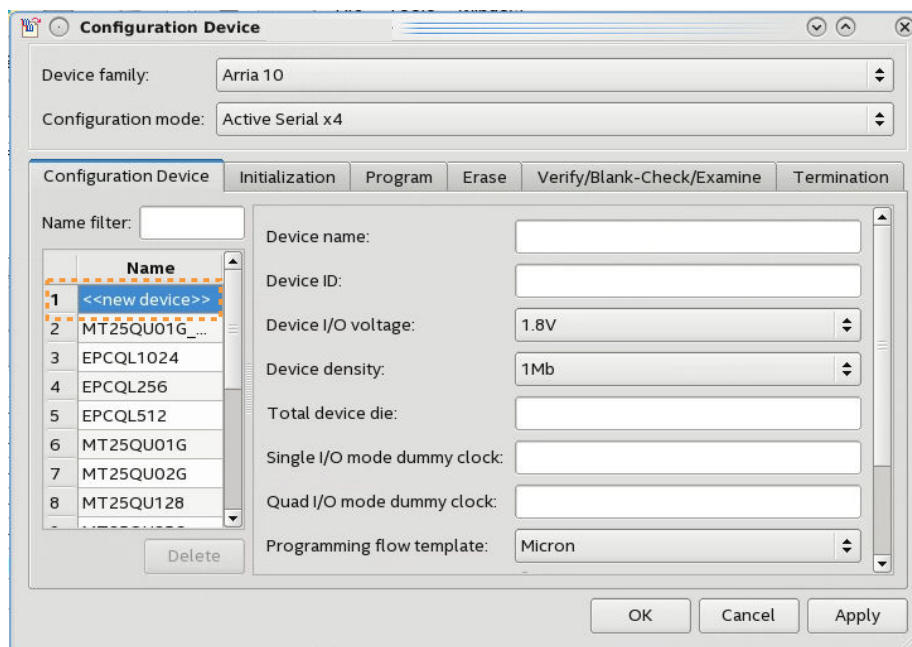
4. To enable compression, turn on the **Compression** option. All encryption options disable.
5. To enable bitstream file encryption:
  - a. Turn off the **Compression** option.
  - b. Turn on the **Generate encrypted bitstream** option.
  - c. Specify options for programming file key decryption, and **Security Options**, as [Compression and Encryption Settings \(Convert Programming File\)](#) on page 26 describes.
6. Click **OK**.

### 1.3.1.2.2. Defining a New Flash Memory Configuration Device

You can define and store the settings for a new flash memory device, based on the programming flow template of an existing supported flash memory device. You can customize and preserve the properties of the new flash memory device for subsequent reuse, and define other flash memory devices based on one you define.

When you define a new flash memory device, Intel Quartus Prime software stores the collection of settings in an .xml file automatically in the **Device database directory** location that you can specify.<sup>(2)</sup>

**Figure 9. Define New Flash Device (Intel Arria 10 Example)**



Follow these steps to define a new flash memory device:

1. Perform one of the following to generate a .jic file for flash programming:

<sup>(2)</sup> Storage of the .xml file in the default installation directory requires system administration rights. If you install a later version of the Generic Flash Programmer, you may need to move or copy any .xml files for use or exclusion from the later version.



- [Step 2: Generate Secondary Programming Files \(Convert Programming Files\)](#) on page 7
2. For the **Configuration Device** option, select **<<new device>>**. The settings on this and other tabs become available.
  3. For **Programming flow template**, select an existing flash memory device template for the new device initial settings, or define a flash programming flow for a different flash memory vendor based on an existing template.
  4. Specify the remaining settings on the **Configuration Device** tab:

**Table 3. Configuration Device Tab Settings**

Option	Description
<b>Device name</b>	Specify a unique name for the flash not already listed in the <b>Name</b> column. The <b>Name</b> must not contain any empty string (space) or special characters (except "_").
<b>Device ID</b>	Specify the 3-byte ID that the Programmer Auto-Detect operation uses to detect the flash programming device, such as 0x20 0xBB 0x21.
<b>Device I/O voltage</b>	Specify <b>1.8V</b> or <b>3.0/3.3V</b> to match your memory device specification.
<b>Device density</b>	Select the total density that corresponds with your flash memory device size.
<b>Total device die</b>	Specify the total number of die for a stacked device (where applicable).
<b>Single I/O mode dummy clock</b>	Specify the Fast Read dummy clock cycle for flash device in single I/O protocol. The programming file generation uses this setting to determine if the configuration requires bit shifting to compensate for the actual dummy clock cycle during Active Serial configuration.
<b>Quad I/O mode dummy clock</b>	Specify the Fast Read dummy clock cycle for flash device in Quad I/O protocol. The programming file generation uses this setting to determine if the configuration requires bit shifting to compensate for the actual dummy clock cycle during Active Serial configuration.
<b>Device database directory</b>	Specifies the location of the .xml file that preserves a flash memory device definition. <i>Note:</i> When you specify a non-default folder for the <b>Device database directory</b> location, place the .sof and .jic files in the same folder as the .xml file to avoid missing a defined flash database or corruption of the .jic file.

5. For supported FPGA devices, optionally modify any of the default programming flows for the flash memory device, as [Modifying Programming Flows](#) on page 12 describes.

*Note:* When you modify a programming flow, all .jic files using this programming flow are affected. For example, you can define a new micron\_lgb flow, and then use this device to define the micro\_lgb\_partA.jic file. Later, you modify the micron\_lgb flow, and then use this flow to create micro\_lgb\_partB.jic. In this example, micro\_lgb\_partA programming flow reflects the latest modifications to micron\_lgb.

### 1.3.1.2.3. Modifying Programming Flows

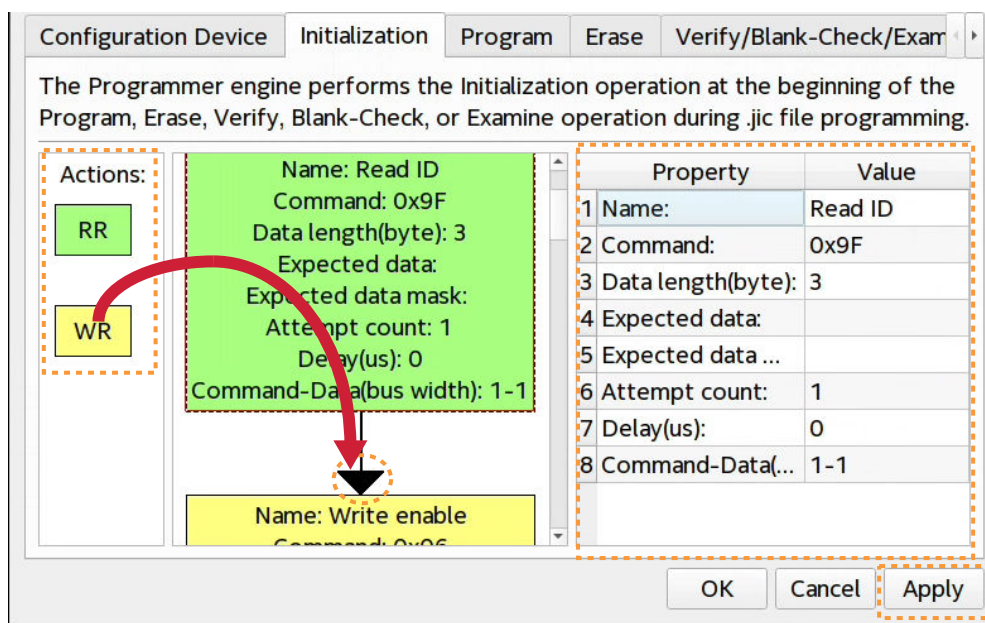
You can modify and preserve the flows for Initialization, Program, Erase, Verify/Blank-Check/Examine, and Termination operations for a flash memory device for supported devices. For each operation, you can drag and drop **Actions** (such as **Read Register**) into locations in the flow to match the programming requirements of your flash memory device. Modifying a programming flow affects all .jic files that use that programming flow.



Follow these steps to modify a default programming flow:

1. Define a new configuration device, as [Defining a New Flash Memory Configuration Device](#) on page 11 describes.
2. In the **Configuration Device** dialog box, click an available **Action** on the **Initialization**, **Program**, **Erase**, **Verify/Blank-Check/Examine**, and **Termination** tabs.
3. With the **Action** selected, drag the action to an arrow point in the flow graphic. The arrow icon changes to indicate when the placement is legal, at which point you can release the mouse to place the **Action** in the flow. Alternatively, right-click in the flow graphic to add, delete, copy, or paste an **Action** from the menu.

**Figure 10. Modifying Initialization Flow**



4. To modify the properties of an **Action**, click the action in the flow graphic. The editable properties appear in the adjacent pane, as [Programming Flow Action Properties](#) on page 20 describes.
5. When you are satisfied with the modifications, click **Apply**. The flow preserves with your new configuration device definition, and applies during the flash device programming.

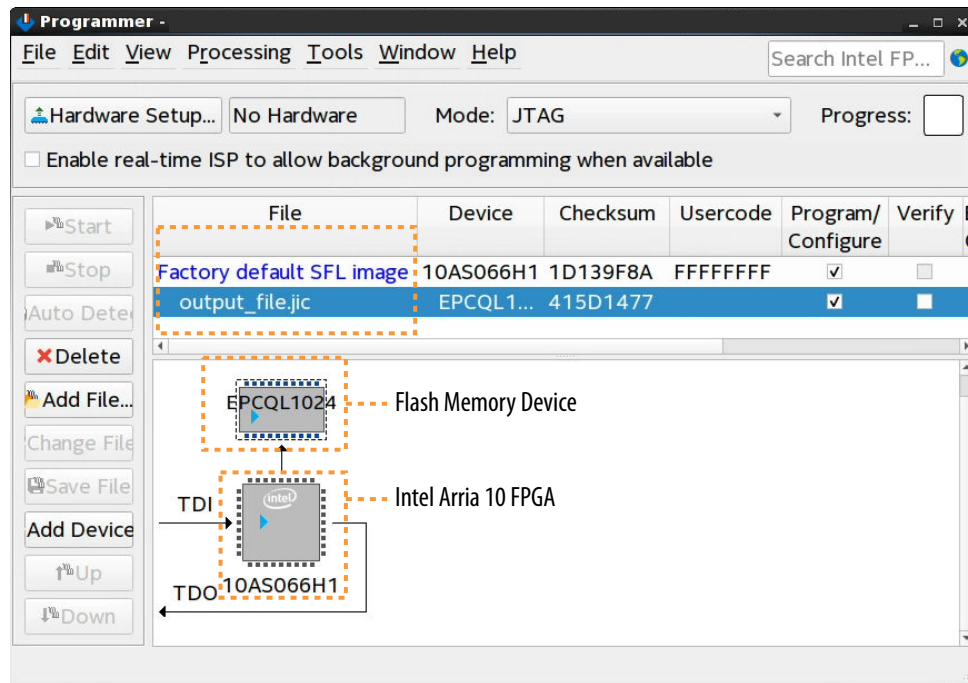
### 1.3.1.3. Step 3: Program the Flash Memory Device

Follow these steps to program the flash memory device using the Intel Quartus Prime Programmer and Intel FPGA download cable over a JTAG connection via Active Serial programming.

1. Generate a .jic file, as [Step 2: Generate Secondary Programming Files \(Convert Programming Files\)](#) on page 7 describes.
2. Open the Intel Quartus Prime software, and then click **Tools** ► **Programmer**.
3. In the Programmer, click **Hardware Setup**, and then select a connected Intel FPGA Download Cable.

4. Click **Add File**, and then select the .jic file you generate at the previous step.

**Figure 11. JIC Loaded for Flash Programming in Programmer**



5. Enable the **Program/Configure** option for the .jic file. The FPGA **Device** row **Program/Configure** option automatically enables, and displays the **Factory default SFL image** as the FPGA configuration data. This entry indicates that the programming flow first configures the FPGA with the SFL image, which subsequently controls the programming of the flash memory device.
6. Click **Start** and wait for the progress bar to reach 100%. The programming flow executes according to your specifications in the .jic file.

### 1.3.2. Erasing Flash Memory Sectors

When performing flash memory erase operations via JTAG and a .jic file, the Intel Quartus Prime Programmer erases only the flash memory sectors that the .jic specifies.

For example, if you specify a .jic file containing only a 13.6Mbits FPGA image on an EPCQ64A device, the Programmer erases only the bottom 13.6Mbits, and does not erase the remaining 50.4Mbits of data.

To erase the entire flash memory device contents, do not specify a .jic file for flash programming. Rather, manually add the flash device to the associated FPGA device chain by following these steps:

1. In the Programmer, right-click the target FPGA device, and then click **Edit ► Attach Flash Device**.
2. Select the appropriate flash device from the list.



## 1.4. Generic Flash Programmer Flow Templates (Convert Programming File)

For supported devices, you can modify and preserve the default programming flows, as [Modifying Programming Flows](#) on page 12 describes.

The following describe the templates for each programming flow:

### Related Information

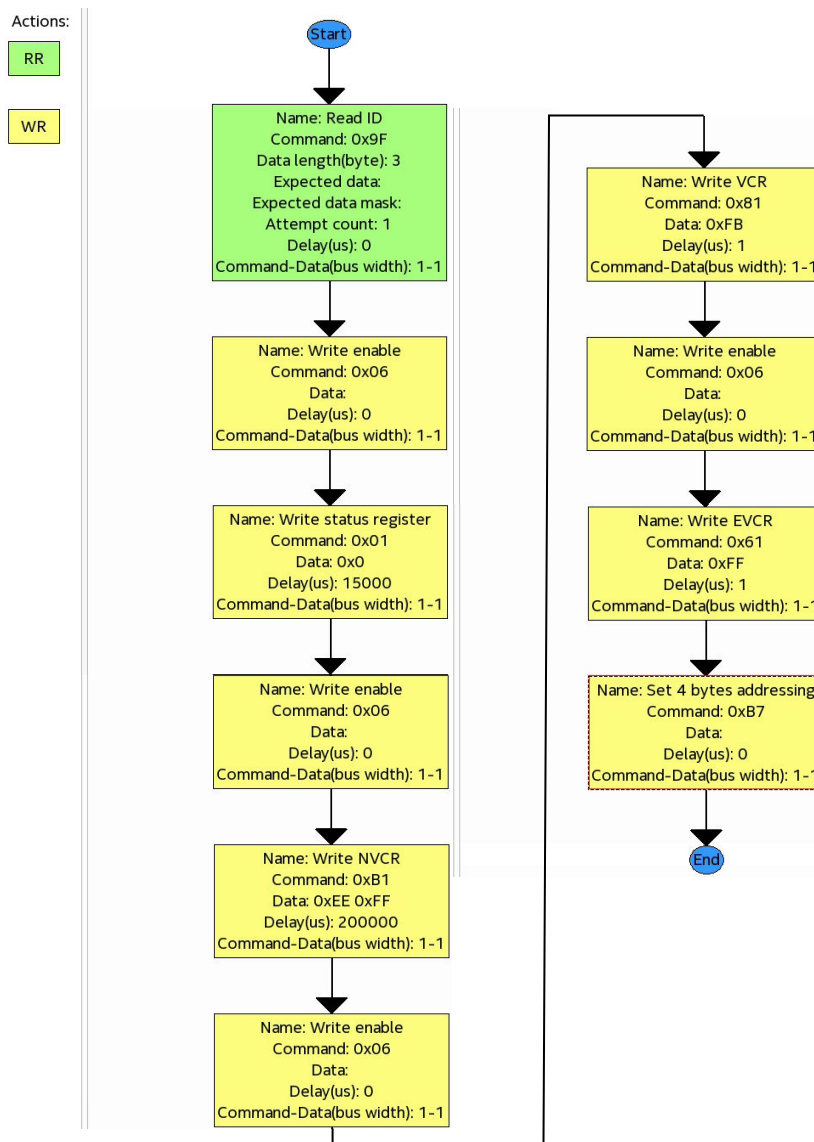
- [Initialization Flow Templates \(Convert Programming File\)](#) on page 15
- [Program Flow Template \(Convert Programming File\)](#) on page 16
- [Erase Flow Template \(Convert Programming File\)](#) on page 17
- [Verify/Blank-Check/Examine Flow Template \(Convert Programming File\)](#) on page 18
- [Termination Flow Template \(Convert Programming File\)](#) on page 19
- [Programming Flow Action Properties](#) on page 20

### 1.4.1. Initialization Flow Templates (Convert Programming File)

The Initialization flow template has the following **Actions**.



**Figure 12. Initialization Flow Template (Micron Example)**



**Note:** Example and **Action** properties are for reference only as specific actions vary by flash memory device.

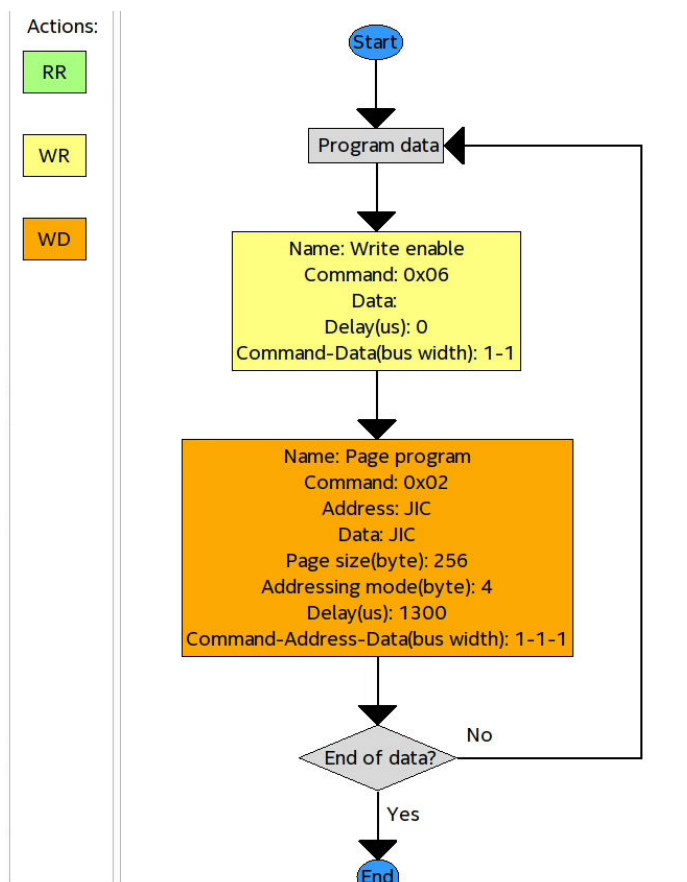
### 1.4.2. Program Flow Template (Convert Programming File)

The Program flow template has the following **Actions**.





Figure 13. Program Flow Template

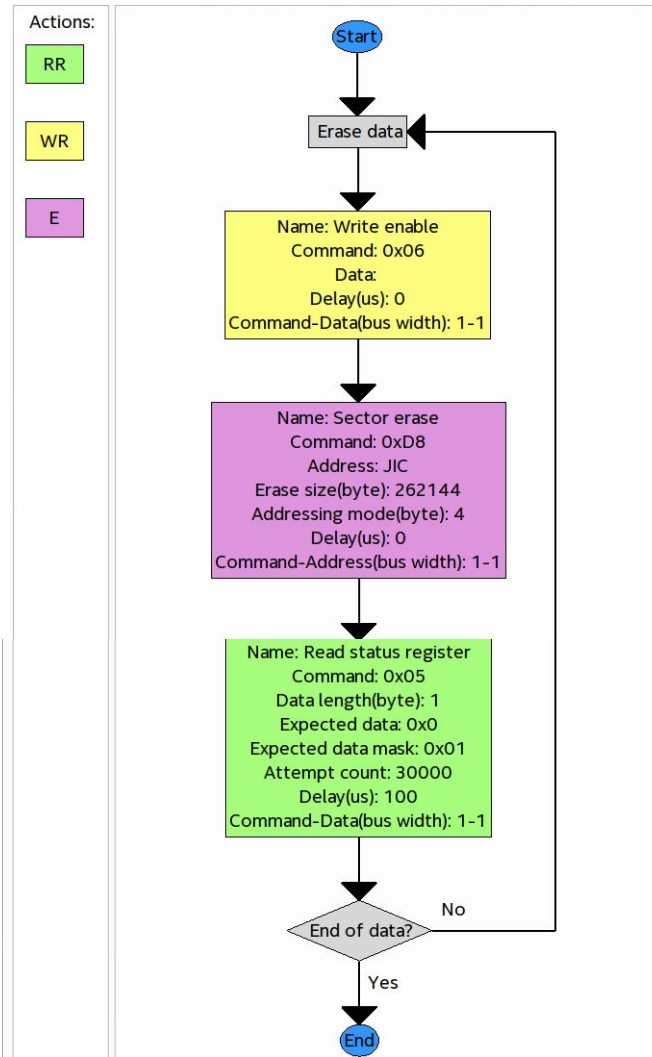


*Note:* **Action** properties are for reference only as specific properties vary by application.

### 1.4.3. Erase Flow Template (Convert Programming File)

The Erase flow template has the following **Actions**.

**Figure 14. Erase Flow Template**



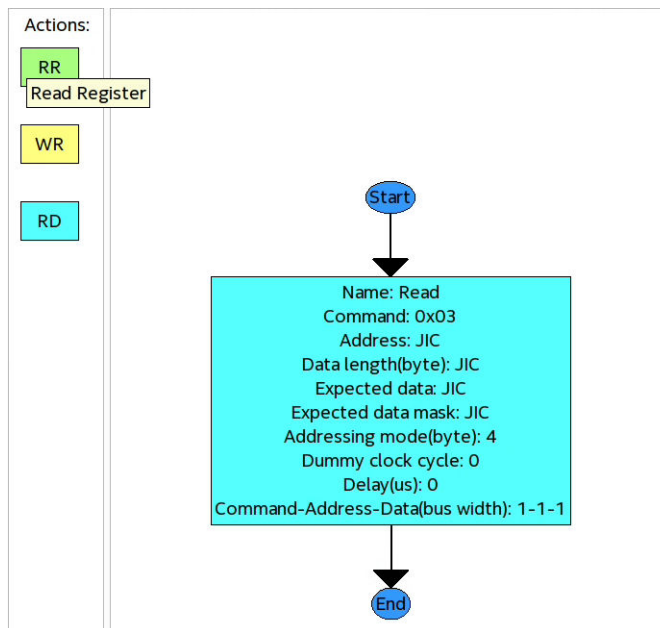
**Note:** **Action** properties are for reference only as specific properties vary by application.

#### 1.4.4. Verify/Blank-Check/Examine Flow Template (Convert Programming File)

The Verify/Blank-Check/Examine flow template has the following **Actions**.



**Figure 15. Verify/Blank-Check/Examine Flow Template**

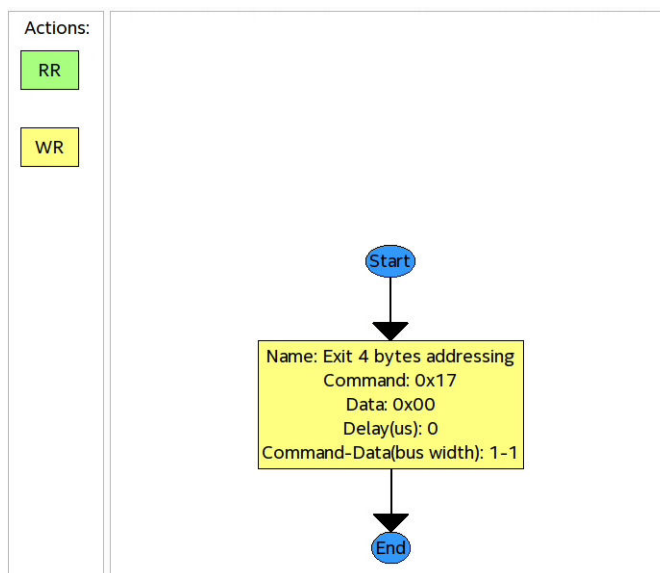


*Note:* **Action** properties are for reference only as specific properties vary by application.

#### 1.4.5. Termination Flow Template (Convert Programming File)

The Termination flow template has the following **Actions**.

**Figure 16. Termination Flow Template**



*Note:* **Action** properties are for reference only as specific properties vary by application.



### 1.4.6. Programming Flow Action Properties

The available programming flow **Actions** have the following properties:

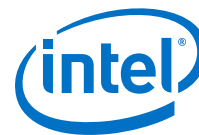
**Table 4. Read Register (RR) Action Properties**

Symbol	Action	Property	Description
<b>Read Register (RR)</b>	Performs a read operation that returns a list of data register bytes.	<b>Name</b>	Displays the name of the command that executes as part of the <b>Action</b> , such as <b>Read ID</b> and <b>Read flag status register</b> .
		<b>Command</b>	Hex value that executes the command, such as 0x70.
		<b>Data length (byte)</b>	Expected length of returned data. To read according to the .jic file, enter the value as JIC.
		<b>Expected data</b>	(Optional) For use with <b>Expected data mask</b> . The following are example entries: <ul style="list-style-type: none"><li>• 1 Byte: 0xAB</li><li>• 2 Bytes: 0xAB 0xCD</li><li>• 3 Bytes: 0xAB 0xCD 0xEF</li></ul> To compare the actual read back data against the .jic file, enter the value as JIC.
		<b>Expected data mask</b>	(Optional) Specify this value if you specify the <b>Expected data</b> . The default value is zero. The operation compares the actual read back register values against the expected value and mask value, and reports error on mismatch. The following are example entries: <ul style="list-style-type: none"><li>• 1 Byte: 0xFF</li><li>• 2 Bytes: 0xFF 0xFF</li></ul>
		<b>Delay (us)</b>	Delay after <b>Command</b> executes.
		<b>Attempt count</b>	The number of times to repeat the command. You can use the <b>Expected data</b> , <b>Expected data mask</b> , <b>Delay</b> and <b>Attempt count</b> to create a polling operation on a register value.
		<b>Command-Data (bus width)</b>	Ratio of command to data bus widths.

**Table 5. Write Register (WR) Action Properties**

Action	Description	Property	Description
<b>Write Register (WR)</b>	Performs a write operation to a volatile or non-volatile register.	<b>Name</b>	Displays the name of the command that executes as part of the <b>Action</b> , such as <b>Write enable</b> or <b>Write status register</b> .
		<b>Command</b>	Hex value that executes the command, such as 0x70.
		<b>Data</b>	Data that you want to write into the register. If the command doesn't require data (for example, Write Enable), leave this property empty. The following are example entries:

*continued...*



Action	Description	Property	Description
			<ul style="list-style-type: none"> <li>1 Byte: 0xAB</li> <li>2 Bytes: 0xAB 0xCD</li> </ul>
		Delay (us)	Delay after <b>Command</b> executes.
		Command-Data (Bus Width)	Ratio of command to data bus widths.

Table 6. Write Data (WD) Action Properties

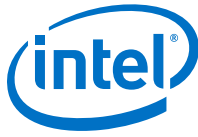
Action	Description	Property	Description
<b>Write Data (WD)</b>	Performs a write operation of data into the flash memory.	<b>Name</b>	Displays the name of the command that executes as part of the <b>Action</b> , such as <b>Page program</b> .
		<b>Command</b>	Hex value that executes the command, such as 0x70.
		<b>Address</b>	Specifies the flash memory address that you want to start writing. To flash the .jic file, enter the value as JIC.
		<b>Data</b>	Data that you want to program into the flash device. <sup>(3)</sup> To flash the .jic file, enter the value as JIC.
		<b>Page size (byte)</b>	Page size of this flash memory device. Typical value is 256 for most flash devices.
		<b>Addressing mode (byte)</b>	Specifies the number of bytes the flash memory device expects for an address (switch between 3 or 4 bytes). The mode must match the addressing mode that you specify in the Initialization flow.
		<b>Delay (us)</b>	Delay after <b>Command</b> executes.
		<b>Command-Address-Data (bus width)</b>	Ratio between command, address, and data bus widths.

Table 7. Read Data (RD) Action Properties

Symbol	Action	Property	Description
<b>Read Data (WD)</b>	Performs a read operation of data from the flash memory.	<b>Name</b>	Displays the name of the command that executes as part of the <b>Action</b> , such as <b>Read</b> .
		<b>Command</b>	Hex value that executes the command, such as 0x70.
		<b>Address</b>	Specifies the flash memory address that you want to start reading. To read according to the .jic file, enter the value as JIC.
		<b>Data length (byte)</b>	Expected length of returned data. To read according to the .jic file, enter the value as JIC.

continued...

<sup>(3)</sup> You can specify the **Data** value in hex, decimal, or a mixture of both. The max entry cannot exceed 32727. For hex entry, 1 byte (0xAA) equals 5 characters for a max entry of 6553 hex characters. The Data value you specify must match the number of bytes for the page size of 256.



Symbol	Action	Property	Description
		<b>Expected data</b>	(Optional) For use with <b>Expected data mask</b> . The following are example entries: <ul style="list-style-type: none"><li>• 1 Byte: 0xAB</li><li>• 2 Bytes: 0xAB 0xCD</li><li>• 3 Bytes: 0xAB 0xCD 0xEF</li></ul> To compare the actual read back data against the .jic file, enter the value as JIC.
		<b>Expected data mask</b>	If you specify the <b>Expected data mask</b> , then the operation compares the actual read back data against the expected value and mask value, and reports an error on mismatch. If you want to compare against the .jic file, enter the value as JIC.
		<b>Addressing mode (byte)</b>	Specifies the number of bytes the flash memory device expects for an address (switch between 3 or 4 bytes). The mode must match the addressing mode that you specify in the Initialization flow.
		<b>Dummy clock cycle</b>	Specifies the number of dummy clock cycles subsequent to the <b>Command</b> .
		<b>Delay (us)</b>	Delay after <b>Command</b> executes.
		<b>Command-Address-Data (bus width)</b>	Ratio between command, address, and data bus widths.

Table 8. Erase (E) Action Properties

Symbol	Action	Property	Description
<b>Erase (E)</b>	Performs an erase of data from the flash memory.	<b>Name</b>	Displays the name of the command that executes as part of the <b>Action</b> , such as <b>Erase sector</b> .
		<b>Command</b>	Hex value that executes the command, such as 0x70.
		<b>Address</b>	Specifies the flash memory address that you want to start erasing. The address must be aligned with <b>Erase size</b> . If you want to flash the .jic file, enter the value as JIC.
		<b>Erase size</b>	Specifies the erase size (in Bytes).
		<b>Addressing mode</b>	Specifies the number of bytes the flash memory device expects for an address (switch between 3 or 4 bytes). The mode must match the addressing mode that you specify in the Initialization flow.
		<b>Delay</b>	Delay after <b>Command</b> executes.
		<b>Command-Address (Bus Width)</b>	Ratio of command to address bus widths.

## 1.5. Generic Flash Programmer Settings Reference

The following topics describe Generic Flash Programmer settings.

### Related Information

- [Device and Pin Options](#) on page 23



- [SOF Data Properties Dialog Box \(Convert Programming File\)](#) on page 27
- [Compression and Encryption Settings \(Convert Programming File\)](#) on page 26
- [Convert Programming File Dialog Box](#) on page 23

### 1.5.1. Convert Programming File Dialog Box

Allows you to convert or combine one or more secondary programming files that support alternative device configuration schemes, such as flash programming, partial reconfiguration, or remote system update.

**Table 9. Convert Programming File Dialog Box Settings**

Setting	Description
<b>Programming file type</b>	Allows you to specify a secondary programming file format for conversion of a primary programming file. The Generic Flash Programmer supports only the .jic file type.
<b>Configuration device</b>	Allows you to select a predefined or define a new configuration device. Click the (...) button to define a new device and programming flow.
<b>Mode</b>	Allows you to select the method of device configuration. The Generic Flash Programmer supports only the <b>Active Serial</b> or <b>Active Serial x4</b> modes.
<b>Output file</b>	Specifies the location of the files that <b>Convert Programming File</b> generates. By default this location is the top-level project directory.
<b>Input files to convert</b>	Specifies one or more primary programming files for conversion or combination into one or more secondary programming files for alternative programming methods.

### 1.5.2. Device and Pin Options

The following tables describe **Device & Pin Option** settings that impact Generic Flash Programmer. To access, click **Assignments > Device > Device & Pin Options**.

#### General Device Options

Allow you to specify basic device configuration options that are independent of a specific configuration scheme. To access these settings, click **Assignments > Device > Device and Pin Options > General**.

**Table 10. General Device Options**

Option	Description
<b>Options</b>	<ul style="list-style-type: none"> <li>• <b>Auto-restart configuration after error</b>—restarts the configuration process automatically if a data error is encountered. If this option is turned off, you must externally direct the device to restart the configuration process if an error occurs. This option is available for passive serial and active serial configuration schemes.</li> <li>• <b>Release clears before tri-states</b>—releases the clear signal on registered logic cells and I/O cells before releasing the output enable override on tri-state buffers. If this option is turned off, the output enable signals are released before the clear overrides are released.</li> <li>• <b>Enable user-supplied start-up clock (CLKUSR)</b>—uses a user-supplied clock on the CLKUSR pin for initialization. When turned off, external circuitry is required to provide the initialization clock on the DCLK pin in the Passive Serial and Passive Parallel Synchronous configuration schemes; in the Passive Parallel Asynchronous configuration scheme, the device uses an internal initialization clock.</li> </ul>
<i>continued...</i>	



Option	Description
	<ul style="list-style-type: none"><li>• <b>Enable device-wide reset (DEV_CLRn)</b>—enables the DEV_CLRn pin, which allows all registers of the device to be reset by an external source. If this option is turned off, the DEV_CLRn pin is disabled when the device operates in user mode and is available as a user I/O pin.</li><li>• <b>Enable device-wide output enable (DEV_OE)</b>—enables the DEV_OE pin when the device is in user mode. If this option is turned on, all outputs on the chip operate normally. When the pin is disabled, all outputs are tri-stated. If this option is turned off, the DEV_OE pin is disabled when the device operates in user mode and is available as a user I/O pin.</li><li>• <b>Enable INIT_DONE output</b>—enables the INIT_DONE pin, which allows you to externally monitor when initialization is complete and the device is in user mode. If this option is turned off, the INIT_DONE pin is disabled when the device operates in user mode and is available as a user I/O pin.</li><li>• <b>Enable JTAG Pin Sharing</b>—enables the JTAG pin sharing feature. The JTAGEN pin is enabled and becomes a dedicated input pin in user mode. JTAG pins (TDO, TCK, TDI, and TMS pins) are available as test pins when the JTAGEN pin is pull low. JTAG pins are dedicated when the JTAGEN pin is high. If this option is turned off, the JTAGEN pin is disabled when the device operates in user mode and is available as a user I/O pin. JTAG pins are retained as dedicated JTAG pins.</li><li>• <b>Enable nCONFIG, nStatus, and CONF_DONE pins</b>—enables the major configuration pins, nCONFIG, nSTATUS, and CONF_DONE pin in user mode. If this option is turned off, the nCONFIG, nSTATUS, and CONF_DONE pins are disabled when the device operates in user mode and are available as user I/O pins.</li><li>• <b>Enable OCT_DONE</b>—enables the OCT_DONE pin, which controls whether the INIT_DONE pin is gated by OCT_DONE pin. If this option is turned off, the INIT_DONE pin is not gated by the OCT_DONE pin.</li><li>• <b>Enable security bit support</b>—enables the security bit support, which prevents data in a device from being obtained and used to program another device. This option is available for supported device (MAX® II, and MAX V) families.</li><li>• <b>Set unused TDS pins to GND</b>—sets the unused temperature sensing diode TSD pins, TEMPDIODEp and TEMPDIODEn to GND in the pin. By default, TSD pins are available for connection to an external temperature sensing device; however, you must manually connect the pins to GND if they are not connected. When turned on, this option updates the information in the .pin file and does not affect FPGA behavior.</li><li>• <b>Enable CONFIG_SEL pin</b>—enables the BOOT_SEL pin in user mode. If this option is turned off, the BOOT_SEL pin is disabled when the device operates in user mode and is available as a user I/O pin.</li><li>• <b>Enable nCEO pin</b>—enables the nCEO pin. This pin should be connected to the nCE of the succeeding device when multiple devices are being programmed. If this option is turned off, the nCEO pin is disabled when the device operates in user mode and is available as a user I/O pin.</li><li>• <b>Enable autonomous PCIe HIP mode</b>—releases the PCIe HIP after peripheral configuration, before device core configuration completes. This option only takes effect if CvP mode is disabled.</li><li>• <b>Enable the HPS early release of HPS IO</b>—releases the HPS shared I/O bank after the IOCSR programming.</li></ul>
<b>Auto usercode</b>	Sets the JTAG user code to match the checksum value of the device programming file. The programming file is a .pof for non-volatile devices, or an .sof for SRAM-based devices. If you turn on this option, the <b>JTAG user code</b> option is not available.
<b>JTAG user code</b>	Specifies a hexadecimal number for the device selected for the current Compiler settings. The JTAG user code is an extension of the option register. This data can be read with the JTAG USERCODE instruction. If you turn on <b>Auto usercode</b> , this option is not available.
continued...	





Option	Description
<b>In-system programming clamp state</b>	<p>Allows you to specify the state that the pins take during in-system programming for used pins that do not have an in-system programming clamp state assignment. Unused pins and dedicated inputs must always be tri-stated for in-system programming. Used pins are tri-stated by default during in-system programming, which electrically isolates the device from other devices on the board. At times, however, in order to prevent system damage you may want to specify the logic level for used pins during in-system programming. The following settings are available:</p> <ul style="list-style-type: none"> <li>• <b>Tri-state</b>—the pins are tri-stated.</li> <li>• <b>High</b>—the pins drive VCCIO.</li> <li>• <b>Low</b>—the pins drive GND.</li> <li>• <b>Sample and Sustain</b>—the pins drive the level captured during the SAMPLE / PRELOAD JTAG instruction.</li> </ul>
<b>Configuration clock source</b>	<p>Specifies the clock source for device initialization (the duration between CONF_DONE signal went high and before INIT_DONE signal goes high). For AS x1 or AS x4 configuration mode, you can select either <b>Internal Oscillator</b> or <b>CLKUSR</b> pin only. The DCLK pin is an illegal option for AS mode. In 14 nm device families, only <b>Internal Oscillator</b> or <b>OSC_CLK_1</b> pins are available.</p>
<b>Device initialization clock source</b>	<p>Specifies the clock source for device initialization (the duration between CONF_DONE signal went high and before INIT_DONE signal goes high). For AS x1 or AS x4 configuration mode, you can select either <b>Internal Oscillator</b> or <b>CLKUSR</b> pin only. The DCLK pin is an illegal option for AS mode. In 14 nm device families, only <b>Internal Oscillator</b> or <b>OSC_CLK_1</b> pins are available.</p>

### Configuration Options

Allow you to specify the configuration scheme, configuration device and pin options, serial clock source, and other options for subsequent device configuration with your programming bitstream. To access these settings, click **Assignments > Device > Device and Pin Options > Configuration**. Disabled options are unavailable for the current device or configuration mode.

**Table 11. Configuration Options**

Option	Description
<b>Configuration mode</b>	Specifies the <b>Standard</b> or <b>Remote</b> configuration mode.
<b>Configuration Device</b>	<p>Allows you to specify options for an external configuration device that stores and loads configuration data.</p> <ul style="list-style-type: none"> <li>• <b>Configuration Device Options</b>—allows you to define the properties of an external configuration device.</li> <li>• <b>Configuration device I/O voltage</b>—specifies the VCCIO voltage of the configuration pins for the current configuration scheme of the target device. This option is available for supported device families.</li> <li>• <b>Force VCCIO voltage to be compatible with configuration I/O voltage</b>—forces the VCCIO voltage of the configuration pins to be the same as the configuration device I/O voltage. If you turn off this option, the VCCIO voltage of the configuration pins may vary depending on the I/O standards used in the I/O banks containing the configuration pins. This option is available for supported device families.</li> </ul>
<b>Configuration Pin Options</b>	Enables or disables operation of specific device configuration pins for status monitoring, SEU error detection, CvP, and other configuration pin options.
<b>Generate compressed bitstreams</b>	Generates compressed bitstreams and enables bitstream decompression in the target device.

*continued...*



Option	Description
Active serial clock source	Specifies the configuration clock source for Active Serial programming. Options range from 12.5 MHz to 100 MHz.
VID Operation Mode	Enables Voltage IDentification logic in the target device with selected operation mode. The available options are <b>PMBus Master</b> or <b>PMBus Slave</b> .
Disable Register Power-Up Initialization	Tri-states the Active Configuration pins in user mode. The Compiler ignores this setting if the configuration scheme you select is not an Active Configuration scheme.

### 1.5.3. Compression and Encryption Settings (Convert Programming File)

The compression and encryption settings allow you to specify options for compression and encryption key security for the device configuration SRAM Object File (.sof). To access these settings, select the .sof in the **Input files to convert** list in the **Convert Programming File** dialog box, and click **Properties**.

**Table 12. SOF File Properties: Bitstream Encryption Dialog Box (Convert Programming File Dialog Box)**

Setting	Description
Compression	Applies compression to the bitstream to reduce the size of your programming file. The Intel Quartus Prime Assembler can generate a compressed bitstream image that reduces configuration file size by 30% to 55% (depending on the design). The FPGA device receives the compressed configuration bitstream, and then can decompress the data in real-time during configuration. This option is unavailable whenever <b>Generate encrypted bitstream</b> is enabled.
Enable decompression during partial reconfiguration	Enables the option bit for bitstream decompression during Partial Reconfiguration.
Generate encrypted bitstream	Generates an encrypted bitstream configuration image. You then generate and specify an encryption key file (.ekp) for device configuration. This option is unavailable whenever <b>Compression</b> is enabled.
Enable volatile security key	Allows you to encrypt the .sof file with volatile (enabled) or non-volatile (disabled) security key.
Generate encryption lock file	Specifies the name of the encryption lock file (.elk) that <b>Convert Programming File</b> generates.
Generate key programming file	Specifies the name of the key programming file (.key) that <b>Convert Programming File</b> generates.
Use key file	<ul style="list-style-type: none"><li>• <b>Key 1 file</b>—specifies the name of Key 1 .key file.</li><li>• <b>Key 2 file</b>—specifies the name of Key 2 .key file.</li></ul>
Key entry	Allows you to enter the keys for decryption.
Security options	<ul style="list-style-type: none"><li>• <b>Disable partial reconfiguration</b>—specifies options for prevention of partial reconfiguration.</li><li>• <b>Disable key-related JTAG instructions</b>—disables JTAG instructions for the period of time you specify.</li><li>• <b>Disable other extended JTAG instructions</b>—disables other JTAG instructions for the period of time you specify.</li></ul>
Design Security Feature Disclaimer	Acknowledges required acceptance of Design Security Disclaimer.



### 1.5.4. SOF Data Properties Dialog Box (Convert Programming File)

Allows you to define flash memory pages that store configuration data. To access from the **Convert Programming File** dialog box, click the **SOF Data** item and click the **Properties** button.

The following settings are available:

**Table 13. SOF Data Properties Dialog Box Settings**

Setting	Description
<b>Pages</b>	Configuration devices can store multiple configuration bitstreams in flash memory, called pages. CFI configuration devices can store up to eight configuration bitstreams. Some Intel FPGA devices can store multiple configuration bitstreams, including the factory image.
<b>Address mode for selected pages</b>	The options are: <ul style="list-style-type: none"><li>• <b>Auto</b>—automatically allocates a block in the flash device to store the data.</li><li>• <b>Block</b>—specify the start and end address of the flash partition.</li><li>• <b>Start</b>—specify the start address of the partition. The tool assigns the end address of the partition based on the input data size.</li></ul>
<b>Start address</b>	Specifies the start address of the partition. Only enabled when <b>Address Mode</b> is <b>Block</b> or <b>Start</b> .
<b>End address</b>	Specifies the end address of the partition. Only enabled when <b>Address Mode</b> is <b>Block</b> .

### 1.6. Generic Flash Programmer User Guide Revision History

Document Version	Intel Quartus Prime Version	Changes
2019.05.15	18.1.1	First public release.