



# Efinity<sup>®</sup> Programmer User Guide

---

UG-EFN-PGM-v4.2  
March 2026  
[www.efinixinc.com](http://www.efinixinc.com)



# Contents

<b>Introduction.....</b>	<b>4</b>
<b>Hardware and Software Requirements.....</b>	<b>4</b>
<b>Installing.....</b>	<b>4</b>
Installing Patches.....	4
<b>FPGA Configuration Modes.....</b>	<b>5</b>
<b>Flash Programming Modes.....</b>	<b>6</b>
<b>About the Programmer GUI.....</b>	<b>7</b>
Working with Bitstreams.....	8
Edit the Bitstream Header.....	9
Bitstream Compression.....	9
Export to Raw Binary Format.....	9
Export to .svf Format.....	10
Convert to Intel Hex Format at the Command Line.....	11
Combine Bitstreams and Other Files.....	12
Combine Bitstreams at the Command Line.....	13
SPI Programming.....	13
Program a Single Image.....	13
Program Multiple Images (CBSEL).....	14
Program Multiple Images (Internal Reconfiguration).....	15
Program Multiple Images (Bitstream and Data).....	16
Program a Daisy Chain.....	16
JTAG Programming.....	17
Trion Family JTAG Device IDs.....	17
Titanium Family JTAG Device IDs.....	17
Topaz Family JTAG Device IDs.....	18
Program a Single Image.....	18
Program Using a JTAG Chain.....	19
Program using a JTAG Bridge.....	20
JTAG Programming with FTDI Chip Hardware.....	22
FTDI Programming at the Command Line.....	22
Identifying FTDI URLs.....	26
Programmer Messages.....	27
Using the Command-Line Programmer.....	32
Configuration Status Register.....	32
<b>Verifying Configuration with the Programmer.....</b>	<b>34</b>
<b>Verified Flash Devices.....</b>	<b>34</b>
<b>Working with Remote Hardware.....</b>	<b>35</b>
<b>Securing Titanium Bitstreams.....</b>	<b>37</b>
Using the Efinity Bitstream Security Key Generator.....	40
Blowing Fuses with the SVF Player.....	42
Encrypt or Sign Bitstreams from the Command Line.....	44
Workflow for Using Security Features.....	45
Verifying Security Settings.....	47
JTAG Command Support with Security Enabled.....	48
<b>Working with JTAG .svf Files.....</b>	<b>49</b>
Using the Efinity SVF Player.....	49
Export JTAG Operations at the Command Line.....	50

- Where to Learn More..... 52**
- Appendix: Installing USB Drivers.....53**
  - Installing the Windows libusbK Driver.....53
  - Troubleshooting Driver Installation..... 54
- Appendix: Program using a JTAG Bridge (Legacy)..... 56**
- Revision History.....57**

# Introduction

Efnix provides a standalone Windows Programmer for use on lab machines or in a manufacturing environment. This tool has the same features as the Programmer provided with the Efinity<sup>®</sup> software, and works on 32- and 64-bit Windows operating systems.

The standalone Programmer uses a bitstream file (**.hex** for SPI programming or **.bit** for JTAG programming) that you generate with the Efinity<sup>®</sup> software to program Trion<sup>®</sup>, Topaz, and Titanium FPGAs.



**Learn more:** For information on generating a bitstream file or on using the Efinity<sup>®</sup> software, refer to the [Efinity Software User Guide](#).

## Hardware and Software Requirements

- Windows 10 or later, 64-bit operating system
- Microsoft Visual C++ 2022 x64 runtime library (or latest version) redistributable  
<https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>
- Zadig software to install USB drivers  
[zadig.akeo.ie](http://zadig.akeo.ie)

## Installing

Double-click the **efinity-*<version>*-windows-x64-pgm.msi** installer package and follow the on-screen instructions.

When the software finishes installing, the following applications are added to your Windows **Start** menu in the **Efinity Programmer *<version>*** folder:

- Efinity JTAG SVF Player *<version>*
- Efinity Key Generator *<version>*
- Efinity Programmer *<version>*



**Note:** Refer to [Appendix: Installing USB Drivers](#) on page 53 for instructions on installing the drivers.

## Installing Patches

You download Efinity<sup>®</sup> patches separately from the software and then install them into your existing Efinity<sup>®</sup> installation directory.

### Windows

1. Download the patch from the Efinity<sup>®</sup> page in the Support Center.
2. Unzip the patch into any temporary directory by double-clicking the patch filename in the Windows Explorer and choosing **Extract all** or by using the command `unzip efinity-<version>-patch.zip` at a command prompt.

3. Setup the environment variables by typing these commands at a command prompt:

```
> <path to Efinity>\<version>\bin\setup.bat
```

4. Run the patch installer by typing these commands at a command prompt:

```
> cd efinity-<version>-patch
> run.bat
```



**Note:** The path <drive>:\Windows\System32 must exist in %PATH% if you have a customized environment variable.

## Linux

1. Download the patch from the Efinity® page in the Support Center.
2. Open a terminal window.
3. Unzip the patch into any temporary directory:

```
> unzip efinity-<version>-patch.zip
```

4. Setup the environment variables:

```
> source /path/to/efinity/<version>/bin/setup.sh
```

5. Run the patch installer:

```
> cd efinity-<version>-patch
> ./run.sh
```

# FPGA Configuration Modes

Trion®, Topaz, and Titanium FPGAs have dedicated configuration pins. You select the configuration mode by setting the appropriate condition on the input configuration pins. Trion®, Topaz, and Titanium FPGAs support the following configuration modes.

**Table 1: FPGA Configuration Modes**

Mode	Description
SPI Active (serial/parallel)	The FPGA loads the bitstream itself from non-volatile SPI flash memory.
SPI Passive (serial/parallel)	An external microprocessor or microcontroller sends the bitstream to the FPGA using the SPI interface.
JTAG	A host computer sends instructions through a download cable to the FPGA's JTAG interface using JTAG instructions.

# Flash Programming Modes

The following table shows the methods you can use to program the configuration bitstream into the flash device on your board. Although you can program the flash directly using the SPI interface, this method requires that you have a SPI header on your board or use an FDTI chip. Therefore, Efinix recommends that you use a JTAG bridge, because that method only requires a JTAG header, which you would typically have on your board for other purposes anyway.



**Important:** If you are using the security feature (Titanium or Topaz only), you can no longer use the JTAG bridge flash programming modes after you disable JTAG access. Refer to [Securing Titanium Bitstreams](#) on page 37 for details.

*Table 2: Flash Programming Modes*

Mode	Description
SPI Active (serial/parallel)	Use the Efinity Programmer and a cable connected to a SPI header on the board.
SPI Active using JTAG Bridge (New)	A improved version of the SPI Active using JTAG Bridge (Legacy) mode with a faster flash programming time.
SPI Active x8 using JTAG Bridge (New)	A improved version of the SPI Active x8 using JTAG Bridge (Legacy) mode with a faster flash programming time.

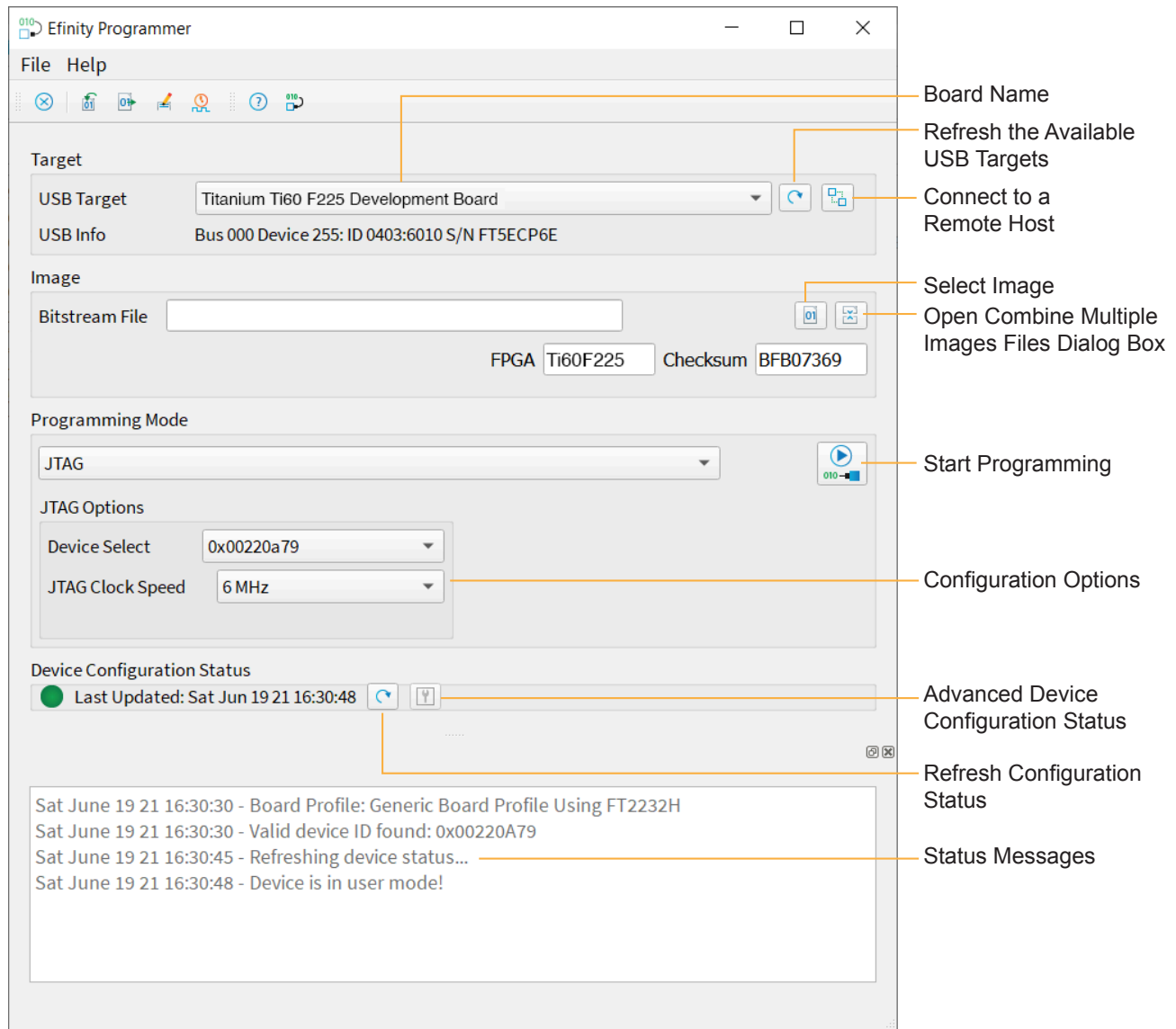


**Learn more:** Refer to [Program using a JTAG Bridge](#) on page 20 for more information.

# About the Programmer GUI

The graphical user interface makes it easy to select bitstream images and program Efinix FPGAs.

Figure 1: Programmer



To use the Programmer:

1. Choose a target. Click the Edit Remote Host List button to connect to a board attached to a remote host. See [Working with Remote Hardware](#) on page 35.
2. Choose a bitstream file. Use a **.hex** file for SPI modes or a **.bit** file for JTAG mode. After you select a bitstream, the Programmer reads the bitstream and displays data in the **FPGA** and **Checksum** fields. The checksum excludes the pre-header and ignores whether characters are uppercase or lowercase; therefore, it is a checksum of the bitstream content, not a file checksum.

**Tip:** You can also get the checksum from the command line using the command:

```
%EFINITY_HOME%\bin\python3 %EFINITY_HOME%\pgm\bin\efx_pgm\generate_checksum.py <bitstream>
```

### 3. Choose the programming mode and then select options.

Mode	Options
SPI Active	Starting Flash Address Flash Length Erase Before Programming Verify After Programming
SPI Passive	Clock Speed
JTAG	Device Select JTAG Clock Speed
SPI Active using JTAG Bridge (Legacy) SPI Active using JTAG Bridge (New) SPI Active x8 using JTAG Bridge (Legacy) SPI Active x8 using JTAG Bridge (New)	Starting Flash Address Flash Length Erase Before Programming Verify After Programming Device Select JTAG Clock Speed

### 4. Click the Program FPGA (SPI Passive or JTAG) or Program Flash (all other modes) button.

The Programmer has status information that gives you diagnostics:

- The FPGA or flash device's configuration status displays in the Device Configuration Status area. Click the Refresh button to refresh the status and display messages in the console.
- Use the Advanced Device Configuration Status button to get diagnostics that can be helpful when debugging why configuration is failing. Refer to [Configuration Status Register](#) on page 32 for more information.



**Note:** For detailed information on how to use configuration modes and set up your circuit board for configuration, refer to [AN 006: Configuring Trion FPGAs](#) or [AN 033: Configuring Titanium FPGAs](#).

## Working with Bitstreams

You can use the Efinity Programmer to manipulate a bitstream before programming an FPGA or flash device.

## Edit the Bitstream Header

You can use the Programmer to edit the bitstream header information, for example, to add project or revision information. To edit the header:

1. In the Programmer, choose **File > Edit Header...** or click the toolbar icon to open the **Edit Image Header** dialog box. The window shows the default header information.
2. Edit the header.
3. Click **Save**.



**Important:** When editing the bitstream header, if you remove any of the auto-generated information (such as `Device: <name>`), the Programmer may not be able to recognize the bitstream. Efinix recommends that you only append a small amount of information to the auto-generated data if you want to customize or annotate the header. The header can be a maximum of 256 characters, including the auto-generated text.

If you want to write your own program to detect which device the bitstream targets (e.g., using a microprocessor and SPI passive mode), be sure to keep all of the auto-generated header, specifically the `Device: <name>` string.

## Bitstream Compression

When you generate a bitstream for Titanium Topaz FPGAs, the Efinity<sup>®</sup> software compresses the bitstream by default. This compression results in a bitstream size that is about half of the maximum size.

Refer to [AN 033: Configuring Titanium FPGAs](#) for the bitstream sizes.



**Important:** If you are using the Titanium or Topaz security features (AES-256 encryption and/or asymmetric authentication), the software cannot compress the bitstream. Therefore, compression is disabled when you use these features.

## Export to Raw Binary Format

The Efinity<sup>®</sup> software v2018.4 and later supports raw binary (**.bin**) format for use with third-party flash programmers. To export to this format:

1. Open the Programmer.
2. Select the bitstream file.
3. Click **Export**.
4. Specify the filename.
5. Click **Save**.

You can also convert the file to **.bin** at the command line as described in [Convert to Intel Hex Format at the Command Line](#) on page 11.

## Export to .svf Format

The Efinity® software v2021.1 and later supports serial vector format (**.svf**) files for use with third-party JTAG programmers. To export to this format:

1. Open the Programmer.
2. Select a bitstream file.
3. Click **Export**.
4. Specify the filename.
5. Choose **Serial Vector Format (\*.svf)** as the **Files of type**.
6. Click **Save**.



**Note:** For more information on using this bitstream format, refer to [Working with JTAG .svf Files](#) on page 49.

You can also convert the file to **.hex** at the command line as described in [Convert to Intel Hex Format at the Command Line](#) on page 11.

---

## Convert to Intel Hex Format at the Command Line

You can convert a bitstream file to Intel Hex and other formats at the command line using this command:

```
export_bitstream.py [--help] [--family FAMILY] [--idcode IDCODE] [--freq FREQ]
  [--sdr_size SDR_SIZE] [--tir_length TIR_LENGTH] [--hir_length HIR_LENGTH]
  [--tdr_length TDR_LENGTH] [--hdr_length HDR_LENGTH] [--enter_user_mode {on, off}]
  format input_file output_file
```

**Table 3: export\_bitstream.py Positional Arguments**

Argument	Input	Description
format	hex_to_bin, hex_to_intelhex, bin_to_hex, intelhex_to_hex, hex_to_svf	Conversion type.
input_file	Filename	Image file source.
output_file	Filename	Image file destination.

**Table 4: export\_bitstream.py Options**

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--family	N/A	Family name	Device family (SVF only)
--idcode	N/A	Identification code	JTAG IDCODE (SVF only).
--freq	N/A	Number	JTAG frequency (SVF only).
--sdr_size	N/A	Number	Approximate JTAG <code>shift_dr</code> size before cycling to idle state (SVF only).
--tir_length	N/A	Number	JTAG bypass trailer instruction register length (SVF only).
--hir_length	N/A	Number	JTAG bypass header instruction register length (SVF only).
--tdr_length	N/A	Number	JTAG bypass header data register length (SVF only).
--enter_user_mode	N/A	on, off	Enter user mode after JTAG configuration (SVF only).

The following example shows conversion of the bitstream **hex** file to **bin** format:

### Example: Converting Hex to Bin

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\export_bitstream.py hex_to_bin new_project.hex test2.bin
```

## Combine Bitstreams and Other Files

You may want to store multiple bitstreams or other data into the same flash device on your board. For example, you can combine files for:

- Multi-image configuration using the CBSEL pins
- Internal reconfiguration
- Programming FPGAs in a daisy chain
- Programming a bitstream and other files such as a RISC-V application binary

You use the **Combine Multiple Image Files** dialog box to choose files to combine into a single file for programming. Choose one of the following modes:

**Table 5: Modes when Combining Images**

Mode	Use For	Number of Images	Notes
Selectable Flash Image	Multi-image configuration	Up to 4	Use this mode if you want the CBSEL pins to control which image the FPGA loads. For this mode, you also need to choose <b>Image Type &gt; External Controller Flash Image</b> . See <a href="#">Program Multiple Images (CBSEL)</a> on page 14
	Internal reconfiguration	Up to 4	Use this mode if you want the internal reconfiguration pins to determine which image the FPGA loads. For this mode, you also need to choose <b>Image Type &gt; Remote Update Flash Image</b> . See <a href="#">Program Multiple Images (Internal Reconfiguration)</a> on page 15
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors <sup>(1)</sup> .	See <a href="#">Program a Daisy Chain</a> on page 16
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	See <a href="#">Program Multiple Images (Bitstream and Data)</a> on page 16



**Note:** When you combine images for an MCU-controlled system or SPI passive daisy chain, the Programmer adds padding between the images as needed. Therefore, you can send the entire bitstream continuously until all devices in the chain are configured.

<sup>(1)</sup> Efinity Programmer does not apply any constraints for combining multiple images. Efinix recommends that you run the IBIS simulation to check the signal integrity if you need to connect more than 4 devices in the same daisy chain.

## Combine Bitstreams at the Command Line

If you want to use a script to combine images at the command line, you can use the **multi\_image\_merger.py** script in the `%EFINITY_HOME%\pgm\bin` directory. The command is:

```
multi_image_merger.py [--help] [--mode MODE] [--type TYPE] [-ifile IFILE] [-iaddr IADDR]
  [--outfile OUTFILE]
```

**Table 6: BRAM Initial Content Updater CLI Options**

Option (long)	Option (short)	Input	Description
--help	-h	None	Show the help.
--mode	-m	generic_comb_image, daisy_chain, image, selectable_flash_image	Specifies which multi-image mode to use. Default: selectable_flash_image
--type	-t	internal, external	Specifies which type to use. For selectable_flash_image mode only. Default: external
--ifile	N/A	Filename	Image files. Can be specified multiple times; use this flag for each file you want to combine.
--iaddr	N/A	Hex number	Starting address. Can be specified multiple times.
--outfile	-o	Filename	Output bitstream file.

### Example: Combining Bitstream Images

```
multi_image_merger.py -m selectable_flash_image -t external -ifile file1.hex -iaddr 0x00
  -ifile file2.hex -iaddr 0x380000 -o output.bit
multi_image_merger.py -m selectable_flash_image -t internal -ifile file1.hex
  -ifile file2.hex -o output.bit
multi_image_merger.py -m daisy_chain_image -ifile file1.hex -ifile file2.hex -o output.bit
multi_image_merger.py -m generic_comb_image -ifile file1.hex -ifile file2.hex -iaddr 0x00
  -iaddr 0x380000 -o output.bit
```

## SPI Programming

You can program Efinix FPGAs using the SPI interface and a **.hex** file.

### Program a Single Image

In single image programming mode, you configure one FPGA with one image.

1. Click the **Select Image File** button.
2. Browse to the **outflow** directory and choose `<project name>.hex`.
3. Choose **SPI Active** or **SPI Passive** configuration mode.
4. Click **Start Program**. The console displays programming messages.

## *Program Multiple Images (CBSEL)*

In this programming mode, you specify up to four images that can configure one FPGA. You then use the FPGA's CBSEL pins to select which image to use. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Selectable Flash Image**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Choose **Image Type > External Flash Image**. This setting tells the FPGA to use the CBSEL pins.
6. Click in the table row corresponding to the position for which you want to add an image.
7. Click **Add Image**.
8. Select the image file to place in that location.
9. Click **OK**.
10. Repeat steps 6 through 9 as needed. You can add up to four images.
11. Click **Apply** to generate the combined image file.
12. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
13. Click **Start Program**.

## Program Multiple Images (Internal Reconfiguration)

In this programming mode, you specify up to four images that can configure one FPGA. You then use the FPGA's internal reconfiguration interface to select which image to use. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Selectable Flash Image**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Choose **Image Type > Remote Update Flash Image**.



**Note:** When using internal reconfiguration, you **must** choose **Remote Update Flash Image**. If you choose **External Flash Image**, the FPGA reconfigures with the first image as specified by the CBSEL pins instead of the golden image.

6. Click in the table row corresponding to the position for which you want to add an image.
7. Click **Add Image**.
8. Select the image file to place in that location.
9. Click **OK**.
10. Repeat steps 6 through 9 as needed. You can add up to four images.
11. Click **Apply** to generate the combined image file.
12. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
13. Click **Start Program**.



**Note:** For more information on using the internal reconfiguration feature, refer to [AN 010: Using the Internal Reconfiguration Feature to Update Efinix FPGAs Remotely](#).



**Note:** For Ti375, Ti240, and Ti165 internal reconfiguration, SPI active x1 with SPI Programming Clock Divider DIV is not supported.



**Note:** For Tz200 and Tz325 internal reconfiguration, SPI active x1 with SPI Programming Clock Divider DIV is not supported.

## Program Multiple Images (Bitstream and Data)

In this programming mode, you specify one bitstream and one or more data files to combine into a single file for programming. You can only use active mode.

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Generic Image Combination**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Click **Add Image**.
6. Select the image file to place in that location.
7. Click **Open**. The image file and flash length are displayed in the table.
8. Specify the flash address.
9. Repeat steps 5 through 8 as needed.



**Note:** If you want to combine a bitstream and a RISC-V binary, use 0x00000000 as the bitstream's flash address and 0x00380000 as the binary's flash address.

10. Click **Apply** to generate the combined image file.
11. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
12. Click **Start Program**.

## Program a Daisy Chain

In this programming mode, you specify any number of images to configure a daisy chain of FPGAs. You can choose active or passive configuration for first FPGA; the rest are in passive mode.

1. Click the **Combine Multiple Images** button.
2. Select **Daisy Chain** as the **Mode**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Click **Add Image** to add a file to the daisy chain.
6. Repeat step 5 to add as many files as you want to the chain. Use the up/down arrows to re-order the images if needed.
7. Click **Apply** to generate the combined image file.
8. Click **Close** to return to the Programmer, which displays the combined image file as the image to use for programming.
9. Click **Start Program**.

## JTAG Programming

You can program Efinix FPGAs using the JTAG interface and a **.bit** file.

### *Trion Family JTAG Device IDs*

The following table lists the Trion JTAG device IDs.

**Table 7: Trion JTAG Device IDs**

FPGA	Package	JTAG Device ID
T4, T8	BGA81	0x0
T8	QFP144	0x00210A79
T13	All	0x00210A79
T20	WLCSP80, QFP100F3, QFP144, BGA169, BGA256	0x00210A79
T20	BGA324, BGA400	0x00240A79
T35	All	0x00240A79
T55, T85, T120	All	0x00220A79

### *Titanium Family JTAG Device IDs*

The following table lists the Titanium JTAG device IDs.

**Table 8: Titanium JTAG Device IDs**

FPGA	Package	JTAG Device ID
Ti35	All	0x10661A79
Ti60	All	0x10660A79
Ti85	All	0x006C2A79
Ti90	J361, J484, G400, G529	0x00691A79
	L484	0x00688A79
Ti120	J361, J484, G400, G529	0x00692A79
	L484	0x0068CA79
Ti135	All	0x006C0A79
Ti165	All	0x006A1A79
Ti180	M484	0x00680A79
	J361, J484, J484D1, G400, G529	0x00690A79
	L484	0x00684A79
Ti240	All	0x006A2A79
Ti375	All	0x006A0A79

## Topaz Family JTAG Device IDs

The following table lists the Topaz JTAG device IDs.

**Table 9: Topaz JTAG Device IDs**

FPGA	Package	JTAG Device ID
Tz50	All	10668A79
Tz75	All	006C8A79
Tz100	All	006C9A79
Tz110	All	00698A79
Tz170	All	00699A79
Tz200	All	006A8A79
Tz325	All	006A9A79

### Program a Single Image

In single image programming mode, you configure one FPGA with one image.

1. Click the **Select Image File** button.
2. Browse to the **outflow** directory and choose *<project name>.bit*.
3. Choose the **JTAG** configuration mode.
4. Click **Start Program**. The console displays programming messages.

## Program Using a JTAG Chain

You can program an FPGA that is part of a JTAG chain. The chain can include Trion<sup>®</sup>, Topaz, and Titanium FPGAs as well as other devices. You define your JTAG chain using a JTAG chain file. You import the JTAG chain file into the Programmer to perform programming. The JTAG chain file is an XML file (.xml) that includes all of the devices in the chain. For example:

Trion FPGA example:

```
<?xml version="1.0"?>
<chain>
  <device chip_num="1" id_code="0x00210a79" ir_width="4" istr_code="1100" />
  <device chip_num="2" id_code="0x00210a79" ir_width="4" istr_code="1100" />
  <device chip_num="3" id_code="0x00210a79" ir_width="4" istr_code="1100" />
</chain>
```

Titanium Topaz FPGA example:

```
<?xml version="1.0"?>
<chain>
  <device chip_num="1" id_code="0x10661A79" ir_width="5" istr_code="11000" />
  <device chip_num="2" id_code="0x10661A79" ir_width="5" istr_code="11000" />
  <device chip_num="3" id_code="0x10661A79" ir_width="5" istr_code="11000" />
</chain>
```

where:

- chip\_num is the device order starting from position 1.
- id\_code is the hexadecimal JEDEC device ID (all lowercase letters)
- ir\_width is the width of the instruction register in bits
- istr\_code is the binary IDCODE instruction



**Note:** To create a Test Data In (TDI) connection, use chip\_num="1" as the first device.



**Note:** For Trion FPGAs, use 1100 as the istr\_code.



**Note:** For Titanium Topaz FPGAs, use 11000 as the istr\_code.

To program using a JTAG chain:

1. Create a JTAG Chain File using a text editor.
2. Open the Programmer.
3. Choose your **USB Target** and **Image**.
4. Select **JTAG** as the **Programming Mode**.
5. Click the Import JCF toolbar button.
6. Browse to your JTAG Chain File and click **Open**.
7. Select which device you want to program in the drop-down list next to the **JTAG Programming Mode** option.
8. Click **Start Program**.



**Note:** If you implement both the daisy chain and JTAG chain together, ensure that the daisy chain is fully completed before executing the JTAG chain. Because the daisy chain requires CSIs to be connected to CSOs, the JTAG chain will only configure successfully when the CSIs are high.

## Program using a JTAG Bridge

Programming with a JTAG bridge is a two-step process: first you configure the FPGA to turn it into a flash programmer (**.bit**) and second you use the FPGA to program the flash device with the bitstream (**.hex**).

The SPI Active using JTAG Bridge mode (formerly named SPI Active using JTAG Bridge (New)) has pre-built flash loader (**.bit**) files that you can use. These **.bit** files do not require an external clock source. You can still use your own **.bit** file if you choose to do so.



**Note:** The JTAG bridge modes were changed in the Efinity software v2025.1. If you are using an older version of software and want to use the SPI Active using JTAG Bridge (Legacy) mode, refer to **Appendix: Program using a JTAG Bridge (Legacy)** on page 56.

The JTAG bridge bitstream files bundled with v2025.1 and higher can only be used with v2025.1 or higher. You cannot use older bundled JTAG bridge bitstream files with v2025.1 or higher, and you cannot use the v2025.1 or higher bundled files with older software versions. If you need to use the older bundled files, use the Programmer v2024.2.

**Tip:** If you would like to incorporate the RTL files for the new flash loader into your own design, use the JTAG to SPI Flash Bridge core in the IP Manager.

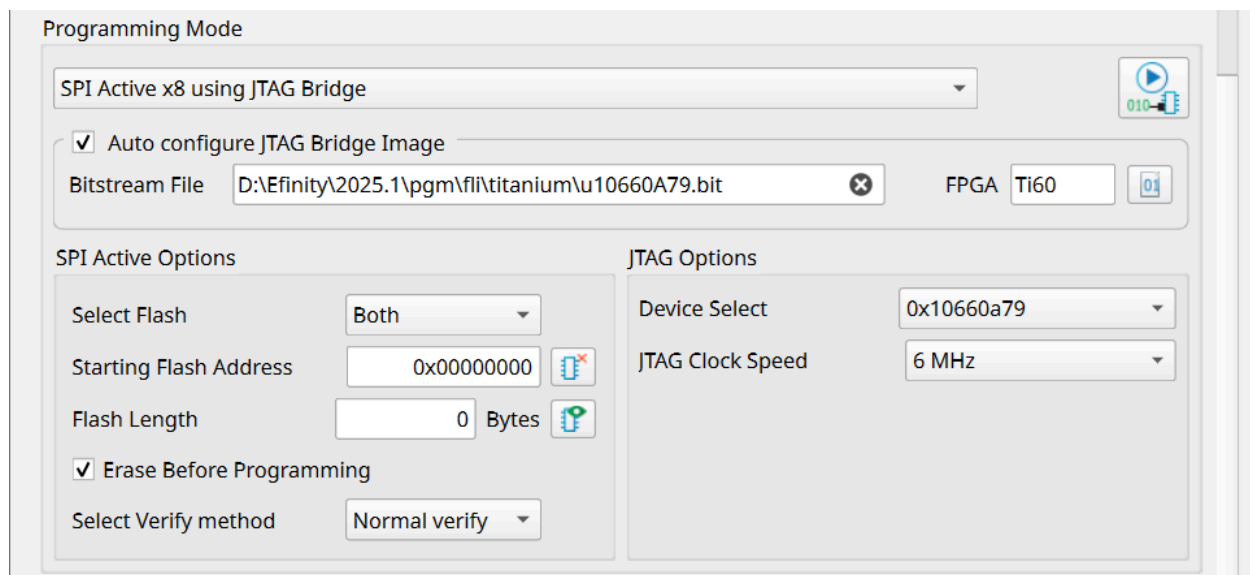
The Titanium and Topaz **.bit** files include a custom JTAG USERCODE in the bitstream:

- Single flash **.bit** files—0x96C09A03
- Dual flash **.bit** files—0xC07FCFE2



**Note:** For Titanium Topaz FPGAs, the Programmer automatically loads the **.bit** file based on the FPGA target. The Programmer has separate pre-built **.bit** files for JTAG bridge mode. For Trion FPGAs, you need to specify the pre-built file to use.

Figure 2: SPI Active Using JTAG Bridge Options



To program using a JTAG bridge:

1. Choose the **USB Target**.
2. In the **Image** box, click the **Select Image File** button to browse for the **.hex** file to program the flash device.
3. Choose the **SPI Active using JTAG Bridge** or **SPI Active x8 using JTAG Bridge** programming mode.

4. Turn on the **Auto configure JTAG Bridge Image** option.  
For Titanium Topaz FPGAs, the Programmer automatically loads the **.bit** file. Skip step 5 if you want to use the pre-loaded **.bit** file.
5. Specify your own **.bit** file.
  - a) In the **Programming Mode** box, click **Select Image File**.
  - b) The **Open Image File** dialog box opens to a directory of available pre-built **.bit** files.  
Choose the file for your FPGA (Trion), or browse to find your own **.bit** file.  
The Programmer remembers which file you specify and uses it automatically the next time you run the Programmer.
6. Choose the **SPI Active Options** and **JTAG Options**.

Option	Description
<b>Select Flash</b>	x8 mode only. Choose whether to use the upper flash, lower flash, or both.
<b>Starting Flash Address</b>	Specify the address if other than the default.
<b>Flash Length</b>	Specify the length if other than the default.
<b>Erase Before Programming</b>	Default: on. When turned on, the Programmer erases the flash device before re-programming it.
<b>Select Verify Method</b>	<p><b>Normal verify</b>—The FPGA computes an on-chip hash from the read back flash data to perform verification. <b>Normal verify</b> is <b>significantly</b> faster than in the Programmer v2024.2 and lower (so much faster that you might think it did not do anything).</p> <p><b>Fast verify</b>—Similar to normal verify, but requires a SPI x4 width (quad mode). The Programmer cannot detect whether your board is using quad mode; if your board is not using it and you try to use fast verify, programming will fail.</p> <p><b>Skip verify</b>—Do not verify the flash.</p>
<b>Device Select</b>	Choose the JTAG device ID of the FPGA to program.
<b>JTAG Clock Speed</b>	Choose a speed or specify a custom one.

7. Click **Start Program**. The Programmer first configures the FPGA and then programs the flash device.



**Important:** If you are using the Titanium Topaz RSA bitstream authentication security feature, you need to use a signed **.bit** file. Copy the bundled **.bit** file from `<Efinity version>/pgm/fli/titanium/pgm/fli/topaz` to another directory and sign it. Then point to the signed **.bit** file in the Programmer. You can also create your own **.bit** file if you prefer.

Refer to [Using the Efinity Bitstream Security Key Generator](#) on page 40 for information on signing existing **.bit** files.

Efnix strongly recommends you to disable JTAG if you are using the security features to achieve the highest security level. While disabled, you can still program the flash with JTAG Bridge by connecting to a soft JTAG tap IP and four GPIOs. Refer to:

- [Blowing Fuses with the SVF Player](#) on page 42
- [JTAG Command Support with Security Enabled](#) on page 48

## JTAG Programming with FTDI Chip Hardware

These instructions describe how to program Trion®, Topaz, and Titanium FPGAs using the FTDI Chip FT2232H and FT4232H Mini Modules. Efinix® has tested the hardware for use with Trion®, Topaz, and Titanium FPGAs.



**Note:** Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures.

1. Open the Efinity® software.
2. Open the Efinity® Programmer.
3. Click the Select Bitstream Image button.
4. Browse to your image and click **OK**.
5. Choose one of the following in the **USB Target** drop-down list:
  - **Dual RS232 HS** for FT2232H Mini Module
  - **FT4232H\_MM** for FT4232H Mini Module
6. Choose **JTAG** from the **Programming Mode** drop-down list.
7. Click **Start Program**.

## FTDI Programming at the Command Line

The Efinity software includes a Python script you can use for programming FTDI modules at the command line.

```
ftdi_pgm.py [--help] [--mode MODE] [--output_file OUTPUT_FILE] [--url URL] [--aurl AURL]
[--xml XML] [--num NUM] [--board_profile BOARD_PROFILE] [--address ADDRESS]
[--num_bytes NUM_BYTES] [--burst_size BURST_SIZE] [--jtag_bridge mode JTAG_BRIDGE_MODE]
[--jtag_clock_freq JTAG_CLOCK_FREQ] [--verify_method VERIFY_METHOD]
[--check_flash_if_supported CHECK_FLASH_IF_SUPPORTED] [--spi_active_freq SPI_ACTIVE_FREQ]
[--spi_passive_freq SPI_PASSIVE_FREQ] [--list_usb] [input_file]
```

**Table 10: ftdi\_pgm.py Positional Arguments**

Argument	Description
input_file	HEX file generated from efx_pgm.

**Table 11: ftdi\_pgm.py Options**

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.

Option (Long)	Option (Short)	Input	Description
--mode	-m	passive, active, jtag, jtag_chain, erase_flash, read_flash, jtag_bridge, jtag_bridge_x8	<p>Programming mode.</p> <p>See the <a href="#">Efinity Programmer User Guide</a>.</p> <p>In Efinity software versions prior to v2025.1, the <code>jtag_bridge</code> and <code>jtag_bridge_new</code> options were named <code>jtag_bridge_new</code> and <code>jtag_bridge_x8_new</code>, respectively. (2)</p> <p>To use the JTAG bridge modes, you must have already configured the with the JTAG SPI flash loader.</p> <p>The Efinity software v2023.2 and higher includes pre-built flash loader.bit files in <code>&lt;installation directory&gt;/pgm/fli/&lt;family&gt;</code>. Refer to the <a href="#">JTAG SPI Flash Loader Core User Guide</a> for information on using the legacy flash loader.</p>
--output_file	-o	Filename	Output file used for <code>read_flash</code> mode.
--url	-u	URL	FTDI URL (see <a href="#">Identifying FTDI URLs</a> on page 26).
--aurl	-a	URL	Alternative URL ( <i>Deprecated</i> ).
--xml	-x	Filename	XML file for JTAG programming.
--num	-n	Number	Chip target number for JTAG chain programming.
--board_profile	-b	Generic Board Profile Using FT232, Digilent JTAG-HS3, FireAnt Development Board, Generic Board Profile Using FT2232H, ISX Programming Cable, Titanium Ti180J484 Dev Board, Titanium Ti180M484 Development Kit, Generic Board Profile Using FT4232, JinChen Programming Cable, TJ180A484S Development Kit, Xyloni Development Board, Generic Board Profile Using FT4234HA	Name of the board profile used.
--address	N/A	Hex number	Starting flash address for flash read and write operations.
--num_bytes	N/A	Number	Number of bytes to erase or read. For modes <code>erase</code> and <code>read</code> only.
--burst_size	N/A	Number	Individual read or write burst size in multiples of 256 bytes. For legacy JTAG bridge modes only ( <code>jtag_bridge</code> and <code>jtag_bridge_x8</code> ).
--jtag_bridge_mode	N/A	Erase, write, erase_and_write, read, all, all_no_erase	JTAG bridge programming mode.

(2) The `jtag_bridge_x8` mode is only supported in some Titanium and Topaz FPGAs. Refer to the data sheet for the modes your FPGA supports.

Option (Long)	Option (Short)	Input	Description
--jtag_clock_freq	N/A	Number	JTAG clock frequency.
--verify_method	N/A	None, onchipx1, onchipx2, onchipx4	The method used to verify the downloaded bitstream. Default: onchipx2 (On-chip hash calculation with SPI x2 mode)
--check_flash_if_supported	N/A	Hex string	Check if flash is supported using the JEDEC ID hex string (e.g., C84012).
--spi_active_freq	N/A	Number	Set SPI active frequency. Default: 6000000 Hz
--spi_passive_freq	N/A	Number	Set SPI passive frequency. Default: 3000000 Hz
--list_usb	-l	None	List the available USB target's URL.

## Linux Examples

To program in Linux:

1. Open a terminal and change to the Efinity® installation directory.
2. Type: `source ./bin/setup.sh` and press enter.
3. Use the `ftdi_program.py` command.

**Example:** Titanium Ti60 F225 Development Board as the only board attached to your computer:

```
ftdi_program.py <filename>.bit -m jtag
```

**Example:** Titanium Ti60 F225 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
ftdi_program.py <filename>.bit -m jtag --url ftdi://ftdi:4232h:FT5ECP6E/1
--aurl ftdi://ftdi:4232h:FT5ECP6E/1
```

**Example:** Xyloni Development Board as the only board attached to your computer:

```
ftdi_program.py <filename>.bit -m jtag
```

**Example:** Trion T120 BGA324 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
ftdi_program.py <filename>.bit -m jtag --url ftdi://ftdi:2232h:FT5ECP6E/1
--aurl ftdi://ftdi:2232h:FT5ECP6E/1
```

## Windows Examples

To program in Windows:

1. Open a command prompt and change to the Efinity® installation directory.
2. Type: `.\bin\setup.bat` and press enter.
3. Use the `ftdi_program.py` command.

**Example:** Titanium Development board as the only board attached to your computer:

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\ftdi_program.py <filename>.bit -m jtag
```

**Example:** Titanium Ti60 F225 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\ftdi_program.py <filename>.bit
-m jtag --url ftdi://ftdi:4232h:FT5ECP6E/1 --aurl ftdi://ftdi:4232h:FT5ECP6E/1
```

**Example:** Xyloni Development Board as the only board attached to your computer:

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\ftdi_program.py <filename>.bit -m jtag
```

**Example:** Trion T120 BGA324 Development Board with serial number FT5ECP6E when another board with an FTDI chip is connected to your computer:

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\ftdi_program.py <filename>.bit
-m jtag --url ftdi://ftdi:2232h:FT5ECP6E/1 --aurl ftdi://ftdi:2232h:FT5ECP6E/1
```

## Identifying FTDI URLs

Certain Efinity<sup>®</sup> scripts contain the `--url` and `--aur1` options, which require the input of an FTDI URL.



**Important:** You only need to specify the `--url` and `--aur1` options if you have more than one board with an FTDI chip connected to your computer.

Only supported in T20 (BGA324 and BGA400), T35, T55, and T120 FPGAs.

The FTDI URL is in the format:

```
ftdi://ftdi:<product>:<serial>/<interface>
```

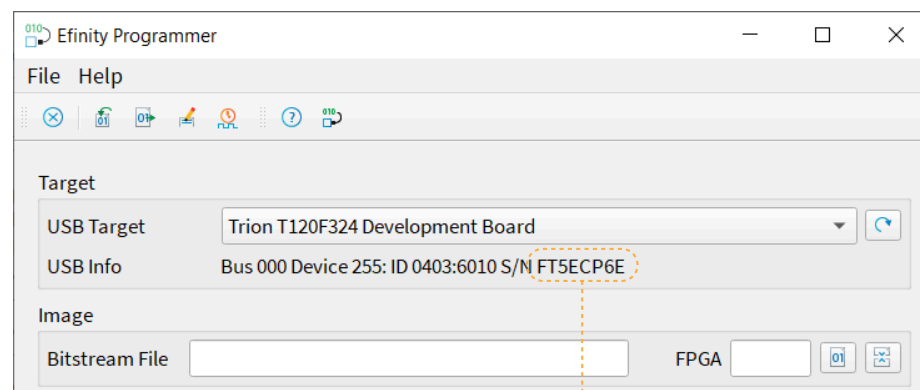
where:

`<product>` is the USB product ID of the device

<code>&lt;product&gt;</code>	Board
232h	Trion T8 Development Board
2232h	Trion T20 MIPI Development Board Trion T20 BGA256 Development Board Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board
4232h	Xyloni Development Board
4232h	Titanium Ti60 BGA225 Development Board Titanium Ti375C529 Development Board Titanium Ti375N1156 Development Board
2232h	Titanium Ti180J484 Development Board
2232h	Topaz Tz170J484 Development Board

`<serial>` is the serial number of the FTDI chip. (Optional)

- If you only have one Efnix<sup>®</sup> development board or FTDI device connected to your computer, you do not need to specify the serial number.
- In the Efinity<sup>®</sup> software v2020.2 and higher, the Programmer displays the serial number of the FTDI device in the **USB Info** string. The serial number is a string beginning with FT.



The string after S/N is the FTDI serial number

`<interface>` is the interface number. For Efnix<sup>®</sup> development boards, `<interface>` is always 1.

## Programmer Messages

The following section lists warning and error messages that the software may display and explains how to fix them.

<b>Message</b>	on_program Device is not available
<b>Reason</b>	1. Board not connected or powered off. 2. USB driver is not installed.
<b>To fix</b>	1. Connect board to host and power on the board. 2. Install USB driver.
<b>Message</b>	USBError(2, 'Entity not found')
<b>Reason</b>	USB driver is not installed.
<b>To fix</b>	Install USB driver.
<b>Message</b>	*Cannot get JTAG url, Please check your board profile configuration *int() argument must be a string, a bytes-like object or a real number, not 'NoneType'-- (idcode=None)
<b>Reason</b>	USB driver is not installed in interface 1 (JTAG).
<b>To fix</b>	Install USB driver for interface 1.
<b>Message</b>	ERROR: Incompatible file extension for programming mode, please use .bit file for JTAG programming
<b>Reason</b>	JTAG chosen as programming mode but <b>.hex</b> file specified in bitstream file.
<b>To fix</b>	Specify correct the <b>.bit</b> bitstream file.
<b>Message</b>	ERROR: Incompatible file extension for programming mode, please use .hex file for SPI Active programming
<b>Reason</b>	SPI Active chosen as programming mode but <b>.bit</b> file specified in bitstream file.
<b>To fix</b>	Specify correct the <b>.hex</b> bitstream file.
<b>Message</b>	ERROR: Incompatible file extension for programming mode, please use .hex file for SPI Passive programming
<b>Reason</b>	SPI Passive chosen as programming mode but <b>.bit</b> file specified in bitstream file.
<b>To fix</b>	Specify correct the <b>.hex</b> bitstream file.
<b>Message</b>	ERROR: Incompatible file extension for programming mode, please use .hex file for SPI Active using JTAG Bridge programming
<b>Reason</b>	SPI Active using JTAG Bridge chosen as programming mode but <b>.bit</b> file specified in bitstream file.
<b>To fix</b>	Specify correct the <b>.hex</b> bitstream file.
<b>Message</b>	ERROR: Incompatible file extension for programming mode, please use .hex file for SPI Active x8 using JTAG Bridge programming
<b>Reason</b>	SPI Active x8 using JTAG Bridge chosen as programming mode but <b>.bit</b> file specified in bitstream file.
<b>To fix</b>	Specify correct the <b>.hex</b> bitstream file.

<b>Message</b>	ERROR: Check board is plugged in, and then click on "Refresh USB Targets"
<b>Reason</b>	Board disconnected or powered off during programming.
<b>To fix</b>	Reconnect the board and click the refresh button.
<b>Message</b>	Image file not found
<b>Reason</b>	Bitstream file not loaded.
<b>To fix</b>	Load the bitstream file.
<b>Message</b>	Failure to configure was detected
<b>Reason</b>	Programmer internally failed to enter configuration mode.
<b>To fix</b>	Program again.
<b>Message</b>	Unable to configure from flash device.
<b>Reason</b>	JTAG state failed to enter USER mode.
<b>To fix</b>	Reprogram the bitstream.
<b>Message</b>	Unable to determine status of device.
<b>Reason</b>	<ol style="list-style-type: none"> <li>1. JTAG programming in unknown state; potentially hardware issue.</li> <li>2. Different width chosen to program the bitstream file. For example, the bitstream is x1 width but SPI Active x8 is chosen.</li> </ol>
<b>To fix</b>	<ol style="list-style-type: none"> <li>1. Try to reprogram the bitstream. If using C323HM cable, check the connectivity.</li> <li>2. Choose the correct width or run the bitgen again.</li> </ol>
<b>Message</b>	ERROR: Flash verify unsuccessful... mismatch found
<b>Reason</b>	<ol style="list-style-type: none"> <li>1. Programmed flash does not match with the selected bitstream.</li> <li>2. The board does not support the selected verify method.</li> </ol>
<b>To fix</b>	<ol style="list-style-type: none"> <li>1. Reprogram the bitstream.</li> <li>2. Select "Normal verify."</li> </ol>
<b>Message</b>	ERROR: Unable to verify JTAG interface, cannot determine configuration status
<b>Reason</b>	JTAG mode used to program the board, but JTAG interface 1 is unstable or the JTAG connection using C2323HM is incorrect or disconnected.
<b>To fix</b>	Check the driver for the JTAG interface or check the wire connection.
<b>Message</b>	ERROR: Unknown error trying to read flash device, aborting. Aborting flash programming FtdiProgram error: could not get flash device
<b>Reason</b>	Attempted to program the board via SPI Active or SPI Passive while interface 0 (SPI) is disabled.
<b>To fix</b>	Check the driver for the SPI interface. If it is unsupported (using C232HM), then it is not possible to program with SPI.
<b>Message</b>	FtdiProgram error: Device is in CONFIGURATION_FAIL state instead of user mode after programming JTAG Bridge Image!
<b>Reason</b>	Wrong or incomplete JTAG Bridge image specified.
<b>To fix</b>	Specify the correct JTAG Bridge image.

<b>Message</b>	Unsupported JTAG Bridge version: 0.0. Please choose the latest bundled JTAG Bridge image and then try again., aborting flash programming FtdiProgram error:
<b>Reason</b>	Used an older or unsupported version of the flash loader.
<b>To fix</b>	Use the supported version of the flash loader.
<b>Message</b>	ERROR: JTAG Bridge Image not found. Please specify correct file path. - ERROR: File = ""
<b>Reason</b>	JTAG Bridge image not specified when SPI Active using JTAG Bridge is selected as programming mode.
<b>To fix</b>	Specify the correct JTAG Bridge image.
<b>Message</b>	ERROR: Invalid speed entered, please only input numbers
<b>Reason</b>	Invalid character entered for Custom JTAG Clock Speed.
<b>To fix</b>	Use a numerical speed value.
<b>Message</b>	ERROR: Invalid speed entered, out of range, please enter a number between 1000 and 30,000,000"
<b>Reason</b>	Invalid character entered for Custom JTAG Clock Speed.
<b>To fix</b>	Use a valid value between 1000 and 30,000,000.
<b>Message</b>	ERROR: The FPGA given in the bitstream file does not match the FPGA you are trying to program. Check that you are using the correct bitstream file.
<b>Reason</b>	Wrong bitstream file specified for the board in use.
<b>To fix</b>	Specify the correct bitstream file.
<b>Message</b>	Detected 4Byte flag in bitstream but flash is smaller or equal to 16MiB Aborting flash programming FtdiProgram error:Detected 4Byte flag in bitstream but flash is smaller or equal to 16MiB
<b>Reason</b>	1. Erase or read flash attempted with a starting flash address greater than the flash capacity. 2. Erase length specified greater than the flash capacity.
<b>To fix</b>	1. Specify the correct starting address. 2. Specify the correct length.
<b>Message</b>	ERROR: Unable to retrieve flash status Check board is plugged in, and then click on ""Refresh USB Targets"" Unrecognized Flash device. Will use Generic Flash profile. Please contact support if you face any problem.
<b>Reason</b>	Programmed bistream contains an incorrect or mismatched header with the board.
<b>To fix</b>	Verify that the bitstream header is correct.
<b>Message</b>	ERROR: Export SVF feature is disabled for T8/T20 bitstreams
<b>Reason</b>	T8 or T20 bitstream exported to SVF.
<b>To fix</b>	SVF is not supported for T8 or T20.
<b>Message</b>	ERROR: Cannot edit SPI Active clock settings for .bit file, please use the .hex file
<b>Reason</b>	Opened Edit SPI Active Clock on a .bit file.
<b>To fix</b>	Edit SPI Active Clock only works for .hex files.

<b>Message</b>	ERROR: Input file has been corrupted, unable to determine target device
<b>Reason</b>	Opened Edit SPI Active Clock on a corrupted <b>.hex</b> file.
<b>To fix</b>	Replace the corrupted <b>.hex</b> file to a valid one.
<b>Message</b>	ERROR: Unable to read input image file, file maybe have been corrupted
<b>Reason</b>	Opened Edit SPI Active Clock on a corrupted <b>.hex</b> file.
<b>To fix</b>	Replace the corrupted <b>.hex</b> file to a valid one.
<b>Message</b>	ERROR: Device code for JTAG Bridge image CANNOT be Unknown. Please ensure you are using correct bitstream file
<b>Reason</b>	<ol style="list-style-type: none"> <li>1. Programmer is unable to detect the device code.</li> <li>2. Connectivity issue.</li> <li>3. USB driver missing for the JTAG interface.</li> <li>4. Programmer is unable to detect non-Efinix device.</li> </ol>
<b>To fix</b>	Ensure there is a valid device ID at Device Select.
<b>Message</b>	ERROR: Export feature only works with Efinity bitstreams
<b>Reason</b>	Export failed because the bitstream is not originally from Efinity.
<b>To fix</b>	Bitsream exports not originally from Efinity are not supported.
<b>Message</b>	ERROR: Cannot detect JTAG chain setup. Please import JTAG chain file
<b>Reason</b>	Programmer is unable to detect non-Efinix device.
<b>To fix</b>	Import the JCF.
<b>Message</b>	Calculated IR width is invalid. Please import JTAG chain file
<b>Reason</b>	A board in the chain is powered off.
<b>To fix</b>	Power on the board and click the refresh button.
<b>Message</b>	Total IR width of the previous JCF does not match actual total IR width. Please import JTAG chain file again.
<b>Reason</b>	Programmer is unable to detect non-Efinix device and cannot auto-detect IR length of the board in the JTAG chain.
<b>To fix</b>	Import the JCF.
<b>Message</b>	ERROR: The Programmer cannot detect the FPGA in the JTAG chain. Check the JTAG cable or header for connectivity issues
<b>Reason</b>	Incorrect connection of the chain or improperly connected wire.
<b>To fix</b>	Check the wire connectivity and check the IDCODE through the SVF Player.
<b>Message</b>	ERROR: Invalid ASCII character detected in header, cannot display header text
<b>Reason</b>	Bitstream header is incorrectly formatted.
<b>To fix</b>	Run the bitstream again to generate a new bitstream.
<b>Message</b>	ERROR: JTAG chain file does not match XSD standard
<b>Reason</b>	The JCF file in the wrong format.
<b>To fix</b>	Fix the format of the JCF.

<b>Message</b>	Error occurred. OpenocdNotRunning("An error occurred when waiting for response from the main loop. OpenocdNotRunning('Failed to add a user due to: OpenOCD Error: no device found; Return code: 1')")
<b>Reason</b>	Occurs in co-debug mode when the board is disconnected and reconnected.
<b>To fix</b>	Close and re-open the Programmer.
<b>Message</b>	Failed to detect number of JTAG TAP. JTAG chain connection may be broken or number of TAP is greater than 128
<b>Reason</b>	Failed to auto-detect the board in the JTAG chain.
<b>To fix</b>	Import the JCF.
<b>Message</b>	ERROR: Invalid output file <file name> specified for image generation
<b>Reason</b>	Name of the output file not specified when using Combine Multiple Image Files.
<b>To fix</b>	Specify the file in Output File.
<b>Message</b>	["", "", "", ""] ERROR: All input files for image generation must be targeted to the same device
<b>Reason</b>	1. No file added to the field at Combine Multiple Image Files. 2. Image mixed with a different target device.
<b>To fix</b>	1. Add at least one image to use the tool. 2. Only the same targeted device bitstream file can be used to combine the image.
<b>Message</b>	ERROR: Flash address '' is not a valid hexademical number
<b>Reason</b>	Flash address unspecified or incorrect at Generic Image Combination.
<b>To fix</b>	Specify the flash address in the correct hexadecimal format.
<b>Message</b>	ERROR: No input files for image combination
<b>Reason</b>	No input file specified for the image combination at Generic Image Combination.
<b>To fix</b>	Specify at least one image.
<b>Message</b>	ERROR: First flash address '0x00380000' is not equal to 0x00000000
<b>Reason</b>	First flash address specified at Generic Image Combination does not start with 0.
<b>To fix</b>	For the first image, the flash address must start with 0.
<b>Message</b>	ERROR: Flash address '0x00000000' should be greater than or equal to next available flash address '0x00121000'
<b>Reason</b>	Second or later flash address specified at Generic Image Combination starts with 0.
<b>To fix</b>	For the second and later images, the flash address cannot start with 0.

## Using the Command-Line Programmer

To run the Programmer using the command line, use the command:

### Example: Command-Line Programmer

Linux:

```
efx_run.py <project name>.xml --flow program [--pgm_opts [mode=MODE] [settings_file]]
```

Windows:

```
efx_run.bat <project name>.xml --flow program [--pgm_opts [mode=MODE] [settings_file]]
```

### Options

`--pgm_opts mode` specifies the configuration mode. The available modes are:

**Table 12: --pgm\_opts Modes**

Mode	Description
active	SPI Active configuration.
passive	SPI Passive configuration.
jtag	JTAG programming. See the <a href="#">Efinity Programmer User Guide</a> for more information about programming with the JTAG interface.
jtag_bridge	SPI Active using JTAG bridge mode.
jtag-bridge_x8	SPI Active x8 using JTAG bridge mode (used with two flash devices). <sup>(3)</sup>

In active mode, the FPGA configures itself from flash memory; in passive mode, a CPU drives the configuration. If you do not specify the mode, it defaults to active. For example, to use JTAG mode, use the command:

```
efx_run.py <project name>.xml --flow program --pgm_opts mode=jtag
```

`--pgm_opts settings_file` specifies a file in which you have saved all of the programming options. A settings file is useful for performing batch programming of multiple devices.






**Note:** See [Programming Options](#) for more programming options.

## Configuration Status Register

Titanium Topaz FPGAs have a configuration status register. You can use the Efinity Programmer to monitor the values in this register to help debug configuration issues. View the register values in the **Advanced Device Configuration Status** dialog box, which you open by clicking the button of the same name.

<sup>(3)</sup> Used with two flash devices. Only supported in some Titanium Topaz FPGAs. Refer to the data sheet for the modes your FPGA supports.

Table 13: Configuration Status Register

Name	Description
IN_USER	<p>0: The FPGA is not in user mode. 1: The FPGA is in user mode. IN_USER waits for all internal resets and tri-states to be released before it goes high.</p> <hr/> <p> <b>Note:</b> This bit is not supported in Ti60ES FPGAs.</p>
CDONE	<p>Configuration done, has the same value as the CDONE output pin. 0: The FPGA is not configured. 1: Configuration is complete.</p>
NSTATUS	<p>Configuration status, has the same value as the active-low NSTATUS output pin if the NSTATUS pin is not driven by user when the FPGA is in user mode. 0: Indicates that the FPGA received a bitstream that was targeted for a different configuration mode or width, or a CRC error is detected during configuration. NSTATUS can also go low if there is a mismatch between the bitstream and the FPGA encryption/authentication keys. 1: During configuration, indicates that the FPGA is in configuration mode.</p>
CRC32_ERROR_CORE	<p>0: No CRC errors were detected in the core configuration bits. 1: One or more CRC errors were detected in the core configuration bits.</p>
RMUPD_ERROR	<p>0: No errors occurred during remote update. 1: An error occurred in the golden image during remote update configuration.</p>
CONFIG_END	<p>0: Configuration is not complete. 1: Configuration completed (whether successful or not).</p>
SYNC_PAT_FOUND	<p>0: Indicates that the FPGA is not receiving the expected synchronization pattern at start of the bitstream. Check for board or power issues. 1: Indicates that the FPGA detected a synchronization pattern at start of the bitstream, and the clock and data connections to the FPGA are acceptable. Any configuration problems are likely digital or logical in nature. After successful configuration the status will return to 0.</p>
SEU_ERROR	<p>0: No SEU detection errors were found. 1: An SEU detection error was found when reading back the SEU CRAM. Has the same value as the SEU detection error status signal to the core fabric.</p>
CRC32_ERROR_PERIPH	<p>0: No CRC errors were detected in the interface configuration bits. 1: One or more CRC errors were detected in the interface configuration bits.</p>
AES256_PASS	<p>For an encrypted bitstream: 0: Decryption failed. The encryption keys used in to program the fuses may not match the ones used to encrypt the bitstream 1: The encrypted bitstream was decrypted successfully. If the bitstream is not encrypted, this register is always a 1.</p> <hr/> <p> <b>Note:</b> This bit is not supported in Ti60ES FPGAs.</p>
RSA_PASS	<p>When using RSA authentication: 0: The signature check failed. The RSA keys used to program the fuses may not match the ones used to sign the bitstream in the Efinity project. 1: The bitstream signature was verified successfully If RSA authentication is not used, this register is always a 1.</p> <hr/> <p> <b>Note:</b> This bit is not supported in Ti60ES FPGAs.</p>

Name	Description
AES_ACTIVE	After the FPGA is configured, you can check this status bit for encryption: 0: AES is disabled in the current design. 1: AES is enabled in the current design.
RSA_ACTIVE	After the FPGA is configured, you can check this status bit for authentication: 0: RSA is disabled in the current device. 1: RSA is enabled in the current device.
USERCODE	Displays the 32-bit hex JTAG USERCODE.

## Verifying Configuration with the Programmer

After you program the flash or configure the FPGA, you can confirm that the bitstream is loaded and the user design is running successfully using the Programmer. You can also use a microcontroller or LEDs to verify configuration. Refer to "Verifying Configuration" in [AN 006: Configuring Trion FPGAs](#) or [AN 033: Configuring Titanium FPGAs](#).

## Verified Flash Devices

The following table lists third-party flash devices tested and verified by Efinix. Unverified flash devices may still be compatible with your FPGA. Refer to the list of Supporting Commands in [AN 070: Understanding SPI Flash Operations in SIP Devices](#) to determine whether your flash device will work with your FPGA.

*Table 14: Tested Flash Devices*

Manufacturer	Family Part Number
GigaDevice <sup>(4)</sup>	GD25Q, GD25WQ, and GD25LQ
Macronix	MX25L, MX25U, MX25V, MX75L, and MX75U
Puya Semiconductor	P25Q
Winbond	W25Q
Micron	M25P and MT25Q
XTX	XT25F
Atmel (Adesto Technologies)	AT25SF
ISSI	IS25LP128 and IS25WP512M



**Note:** Efinix recommends using SPI NOR flash memories.

<sup>(4)</sup> Does not support 4-byte address mode.

# Working with Remote Hardware

The Efinity software includes the Efinity Hardware Server that allows you to communicate with a development board that is attached to a remote host machine. For example, you may want to use your Efinix development board in a lab environment and let several developers access it from their own computers. With the Efinity Hardware Server, you can connect the board to the lab machine and then program or debug it from a remote networked computer. The Efinity Hardware Server is supported in the Programmer, Debugger, and SVF Player.



**Important:** The Efinity Hardware Server is beta in the Efinity software v2021.2, v2022.1, and 2023.1. Please excuse any random bugs, we will fix them.

**Known issue:** Currently, the hardware server does not arbitrate between multiple requests. Therefore, if more than one person tries to connect to the board, there will be a conflict and all users will see errors in the Programmer console or the Programmer may crash or hang. If the board is in the middle of programming when multiple requests occur, programming aborts in an unfinished state.

## Start the Efinity Hardware Server

You start the Efinity Hardware Server using the `efinity_hw_server.py` command-line tool.

```
efinity_hw_server.py [--help] [--addr ADDR] [--port PORT]
```

Table 15: `efinity_hw_server.py` Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--addr	-a	Address	Server address. If you do not specify an address, the Efinity Hardware Server defaults to 0.0.0.0 (that is, all IPv4 addresses on the local machine).
--port	-p	Number	Server port number. If you do not specify a port, the Efinity Hardware Server defaults to 8080.

The tool issues the message `Running Server at <IP address>:<port>` when the Efinity Hardware Server begins running.

### Windows:

Use the following commands in a Command Prompt to start the server:

```
<Efinity path>\bin\setup.bat
<Efinity path>\bin\python3.bat <efinity path>\pgm\bin\efx_pgm
\efinity_hw_server.py
```

### Linux:

Use the following commands in a terminal to start the server:

```
source <Efinity path>/bin/setup.sh
python3 <Efinity path>/pgm/bin/efx_pgm/efinity_hw_server.py
```

## Stop the Efinity Hardware Server

In the terminal or Command Prompt, enter `Ctrl+C` to stop the server.

## Connect the Board to the Server

For Efinix development boards, connect the board to the server using a USB cable. When you connect to the remote host from your computer, the board name appears in the Programmer's **USB Target** list.

For your own board, use a JTAG Mini-Module or JTAG cable to connect the board to the server. When you connect to the remote host from your computer, the module or cable name appears in the Programmer's **USB Target** list. (Refer to **JTAG Programming with FTDI Chip Hardware** on page 22.)

## Connect to a Remote Host

You use the **Edit Remote Host** dialog box to manage the list of remote server hosts. You access this dialog box from Programmer, Debugger, or SVF Player tools.

1. Click the Edit Remote Host List button to open the **Edit Remote Host** dialog box.
2. Press the + button.
3. Double-click the cell under **Address** and enter the server's IP address.
4. Double-click the cell under **Port** and enter the port.
5. Click the + button to add another row. Click the - button to remove a selected row.
6. Click **OK**.

The software refreshes the **USB Target** list; any boards connected to remote hosts appear in the list. Simply choose the board that you want to program or debug as usual.

# Securing Titanium Bitstreams

Titanium FPGAs have built-in security features to help you protect your intellectual property and to prevent tampering.

- *Encryption*—Encrypt your bitstream using an AES-256 key.
- *Authentication*—Sign your bitstream with an RSA-4096 private key.
- *JTAG Disable*—Permanently disables all JTAG instructions except for those used to get device information.
- *JTAG Disable Efuse Only*—Permanently disables the JTAG efuse instructions only.



**Note:** Refer to **JTAG Command Support with Security Enabled** on page 48 for details on the JTAG disabling modes and which commands they support.

You use the following Efinity tools to implement these bitstream security features:

**Table 16: Efinity Tools Used for Securing Bitstreams**


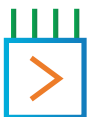


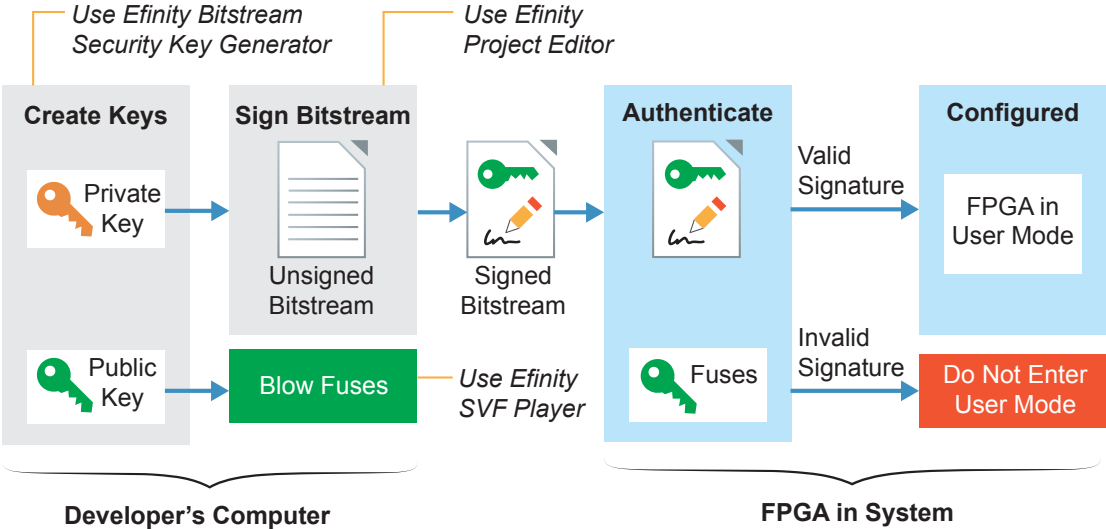
Tool	Used for
 Bitstream Security Key Generator	Create or specify an AES-256 key. Create or specify an RSA-4096 private key. Specify whether to disable JTAG.
 SVF Player	Program the fuses in the Titanium FPGA with the AES-256 key and/or RSA certificate data. After you blow the fuses with an RSA key, the FPGA only accepts a bistream signed with the correct private key. After you blow fuses with an AES-256 key, the FPGA only accepts a plaintext bitstream or a bitstream signed with the correct key. Program the JTAG fuse to disable JTAG function.
 Project Editor	Turn on bitstream encryption and/or authentication, and specify the <b>.bin</b> file created by the Bitstream Security Key Generator. Turn on bitstream authentication and specify the private key ( <b>.pem</b> ) file to sign the bitstream.   <b>Note:</b> You need the full version of Efinity software to work with projects and to generate bitstreams. The Windows Standalone Programmer does not support these features.

Figure 3: Bitstream Authentication



The public key is derived from the private key; the **.pem** is essentially a private/public key pair. The private key only exists in the **.pem**. The software uses it to sign the bitstream, but the bitstream and fuses only contain the public key information. The FPGA uses the public key to validate the bitstream's signature; it cannot be used to re-sign a modified bitstream.

Figure 4: Bitstream Encryption

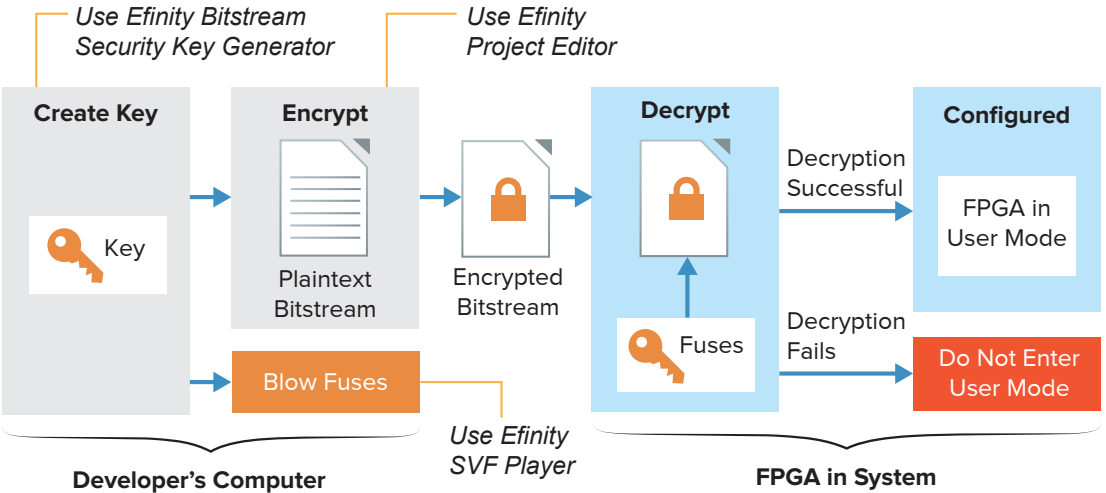
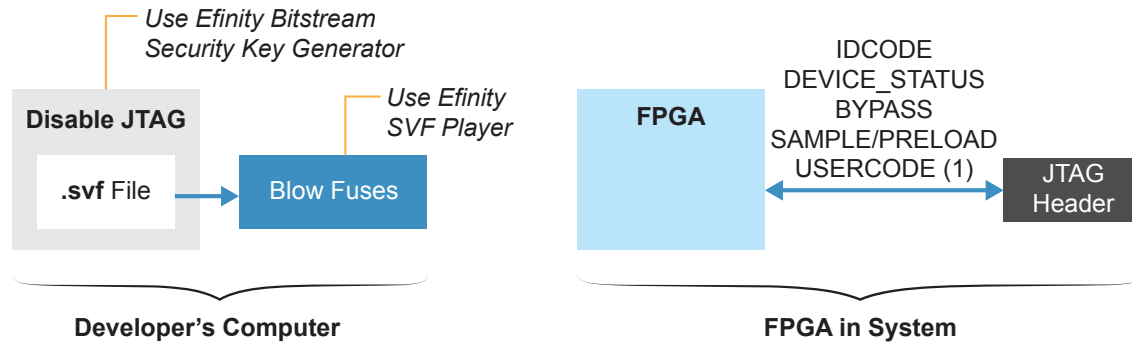


Figure 5: Disabling JTAG



Note:

1. Not supported in some FPGAs, see "JTAG Command Support with Security Enabled" topic.

The following sections describe how to use each of these tools to enable security features.

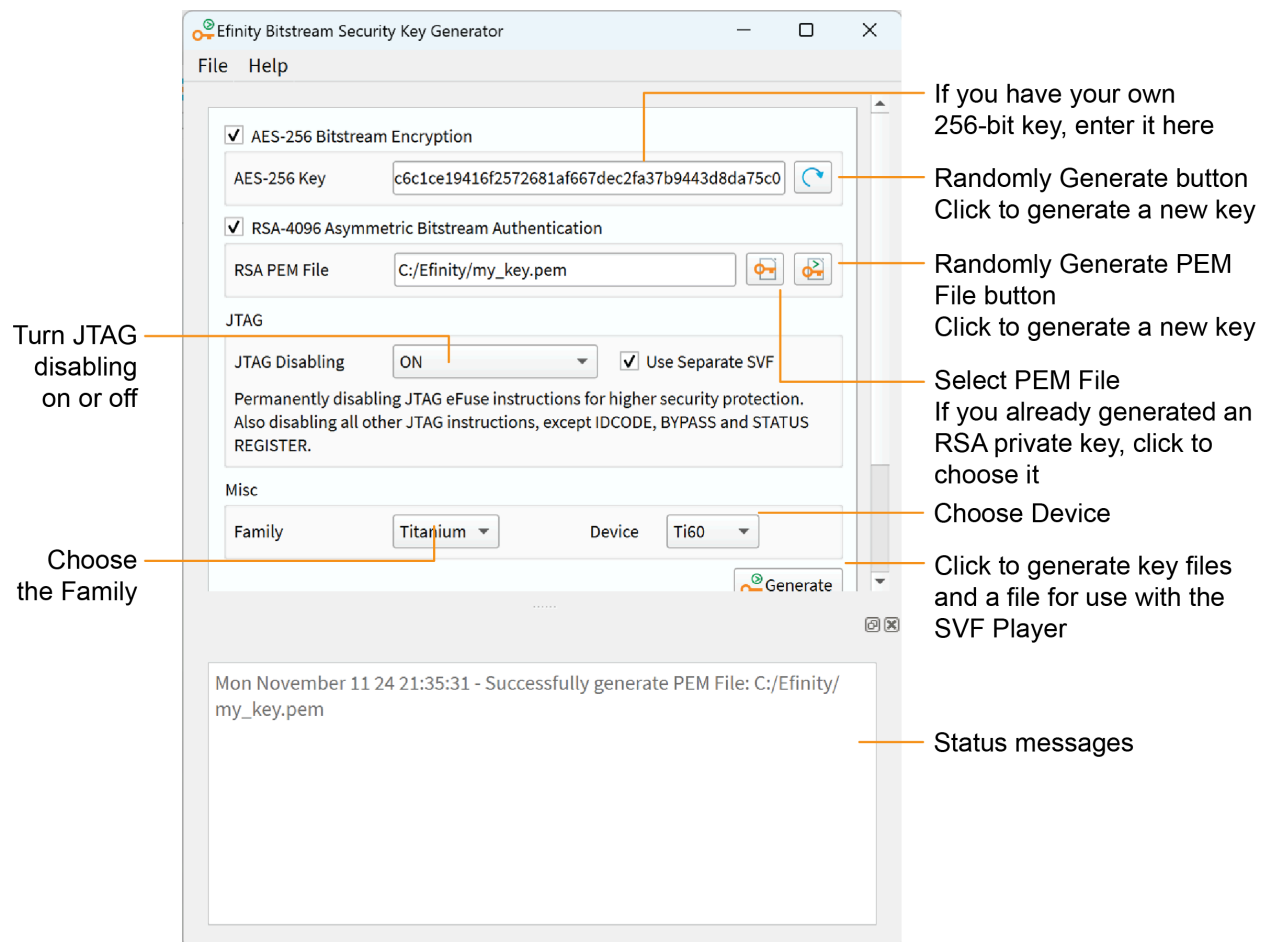
## Using the Efinity Bitstream Security Key Generator

The key generator tool simplifies the process of creating encryption keys and generating RSA certificates. You access this tool in the Efinity main menu at **Tools > Open Key Generator**. You can use the key generator without opening a project.



**Note:** You can use the Efinity Bitstream Security Key Generator iteratively. That is, you can first use encryption and later add in RSA authentication, and even later disable JTAG commands. Refer to [Workflow for Using Security Features](#) on page 45 for more information.

Figure 6: Efinity Bitstream Security Key Generator



1. If you want to use encryption:
  - a) Turn on **AES-256 Bitstream Encryption**.
  - b) Click the Randomly Generate button to generate a 256 bit key. The software populates the **AES-256 Key** box with the generated key.
  - c) Alternatively, if you already have a key, you can enter it into the **AES-256 Key** box.
2. If you want to use authentication:
  - a) Turn on **RSA-4096 Asymmetric Bitstream Authentication**.
  - b) Click the Randomly Generate PEM File button.
  - c) In the **Generate AND Save PEM File** dialog box, choose a location to save the **.pem** file and type a filename in the **File name** box.

- d) Click **Open**. The tool generates the private key and displays a message in the status box.
- e) Alternatively, click the Select PEM File button to load a private key (**.pem**) that you created already.



**Note:** If you use another tool to create a private key, be sure to use the RSA-4096 algorithm. Titanium FPGA's only support authentication with this algorithm.

3. If you are ready to turn off JTAG, choose **ON** or **DISABLE\_EFUSE\_ONLY** for **JTAG Disabling**. Otherwise, leave it set to **OFF**.

Option	Description
<b>OFF</b>	No JTAG disabling. Efinix strongly recommends that you use <b>ON</b> or <b>DISABLE_EFUSE_ONLY</b> to disable access to the JTAG efuse instructions for added security.
<b>ON</b>	Permanently disables the JTAG efuse instructions as well as all other JTAG instructions except for those used to get device information.
<b>DISABLE_EFUSE_ONLY</b>	Permanently disables the JTAG efuse instructions only. Other JTAG instructions are not affected, for example, you can still perform debugging.



**Note:** See **JTAG Command Support with Security Enabled** on page 48 for details.

If you turn on the **Use Separate SVF** option, the software creates two SVFs: one for AES and/or RSA (**<keyname>.svf**) and one for JTAG disabling (**<keyname>\_jtag\_disable.svf**). Two files make it easy to use the key generator iteratively, and when you are done to disable JTAG.

When the **Use Separate SVF** option is tuned off, the software creates one **<keyname>.svf**, which contains all applicable AES, RSA, and JTAG disabling commands.



**Important:** Do not permanently disable JTAG unless you are really ready, that is, you are finished with all JTAG debugging and configuration tasks. After you disable JTAG, you cannot undo it. Use **DISABLE\_EFUSE\_ONLY** if you still want to perform debugging.

4. Choose your FPGA.
5. Click **Generate**.
6. In the **Select Output File** dialog box, choose the location to save the **.bin** (key data) file and type a filename in the **File name** box.
7. Click **Open**.

The tool creates the following files:

- **<filename>.bin**—This file contains key information. You specify it in the Project Editor when you turn on bitstream encryption and/or authentication.
- **<filename>.pem**—This file contains your RSA private key. You use this file to sign the bitstream by specifying it in the Project Editor.
- **<filename>.svf**—This file contains JTAG commands and key information. You use it with the Efinity SVF Player to blow the FPGA fuses.



**Note:** Efinity recommends that you save the 256-bit encryption key in a safe place so you have it in case you want to generate another **.svf** later (see [Workflow for Using Security Features](#) on page 45). You need to copy it from the **AES-256 Key** box and save it into a text file.

## Blowing Fuses with the SVF Player

The Efinity SVF Player is a JTAG SVF player that sends JTAG commands to an FPGA. The player reads the JTAG commands from a serial vector format (**.svf**) file. You can use the SVF Player without opening a project. The Efinity SVF Player requires a JTAG cable or mini-module with the FTDI *m232H* chipset.

The Efinity Bitstream Security Key Generator creates an **.svf** that you use with the SVF Player to blow fuses in Titanium FPGAs. These fuses contain key information for bitstream encryption and/or RSA authentication, and also control JTAG access to the FPGA.

The **.svf** used for blowing fuses performs a variety of JTAG commands.

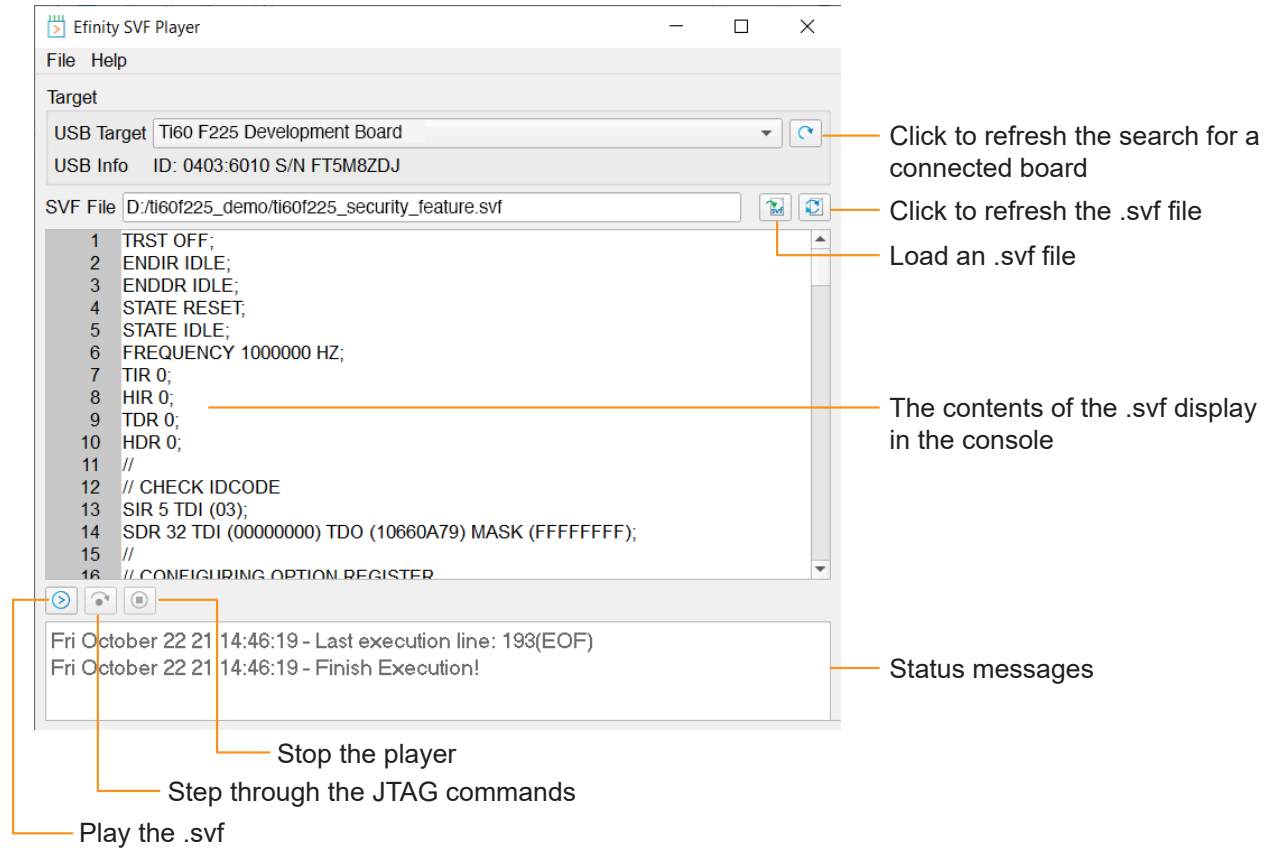
- It checks the FPGA's IDCODE and compares it to the **.svf** to ensure that the player is targeting the correct FPGA.
- For AES encryption, the key is sent in eight 32-bit words, followed by a validation step.
- For RSA authentication, the key is sent in twelve 32-bit words, followed by a validation step.
- It has commands to blow the JTAG fuse.

The **.svf** only has commands for the bitstream security features that you turned on in the Efinity Bitstream Security Key Generator.



**Important:** You can only blow the fuses once, and you cannot undo it after you have blown them. So make sure that you are really ready before you take this step.

Figure 7: SVF Player



To blow fuses with the SVF Player:

1. Choose a **USB Target**. Ensure that your board is connected to your computer and turned on. Click the Refresh button to search for newly connected boards.
  2. Click the Open SVF File button to load the **.svf** that you generated with the Efinity Bitstream Security Key Generator. The content of the **.svf** displays in the console.
- Note:** If you make changes to the **.svf**, you can reload it using the Reload button.
3. Click the Play button to play the **.svf** file. It takes a very short amount of time to blow fuses.
  4. Toggle `CRESET_N` or power cycle your board for the new fuse settings to take effect.



**Important:** Do not try to blow the same fuses a second time (for example, do not run the same **.svf** twice in a row).

Typically, you will not receive any errors when running the SVF Player. However, you may receive a TDO mismatch error in the following situations:

- You are trying to blow fuses that are already blown.
- You are trying to blow fuses for the wrong FPGA, that is, the FPGA you selected in the Efinity Bitstream Security Key Generator is not the same as the one on your board.

## Encrypt or Sign Bitstreams from the Command Line

The Efinity software includes a Python script you can use to encrypt and/or sign bitstreams from the command line. You use the script `$EFINITY_HOME/security/bin/AddSecurityTitanium.py`.

```
AddSecurityTitanium.py [--help] [--sign] [--encrypt] [--iv IV] [--output OUTPUT]
  [--verbose] [--timeout TIMEOUT] [--keypair KEYPAIR] [--passphrase PASSPHRASE]
  [--public_key PUBLIC_KEY] [--signature_file SIGNATURE_FILE]
  bitstream keyfile
```

**Table 17: AddSecurityTitanium.py Positional Arguments**

Argument	Description
bitstream	Bitstream hex file name.
keyfile	Keyfile name.

**Table 18: AddSecurityTitanium.py Options**

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--sign	-s	None	RSA sign the bitstream. Required if target device has enabled RSA in non-volatile memory. With this option, you must also specify the RSA PEM key file containing the RSA private key.
--encrypt	-e	None	Encrypt the bitstream. Optional regardless if target device has had decryption key programmed in non-volatile memory.
--iv IV	-i IV	None	Manually specify 96-bit bit IV value, for obfuscation. If not specified, one will be auto-generated. Ignored if encryption not used.
--output	-o	Filename	Use the specified output security-enabled HEX file name instead of default name.
--verbose	N/A	None	Print out detailed information.
--timeout	N/A	Number	Timeout in seconds, defaults no timeout.
--keypair	-p	Key pair	RSA keypair PEM file (must match that used with GenKeyFileTitanium.py tool).
--passphrase	-x	Pass phrase	Passphrase associated with RSA private key, contained in RSA PEM key pair file. If the private key is passphrase-protected, then this option is required.
--public_key	N/A	Filename	RSA public key PEM file.

### Example: Sign and Encrypt a File

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\security\bin\AddSecurityTitanium.py --sign --encrypt
  --iv 0123456789ABCDEF01234567 --output my_secured_bitstream.hex --device_version 1
```

```
--keypair my_private_key.pem my_raw_unsecured_bitstream.hex my_keyfile.bin
```

## Workflow for Using Security Features

This topic describes some of the potential workflows you might use when developing applications that include bitstream security. You do not have to use all of the bitstream security features simultaneously. You can enable them sequentially or only use some of the features if that suits your workflow.

This iterative process has two parts: blowing fuses and securing the bitstream.

### Blowing Fuses Iteratively

You can blow fuses in any order, and blow only some of them in any iteration. For example, you can:

1. Blow fuses for only AES-256.
2. Blow fuses for only RSA authentication.
3. Blow fuses for AES-256 after doing step 2.
4. Blow fuses for RSA authentication after doing step 1.
5. Blow fuses for both AES-256 and RSA authentication, but do not blow JTAG fuse.
6. Blow fuses for AES-256 and RSA authentication, and blow JTAG fuse (*all in* mode where you turn on everything).
7. Blow JTAG fuse after doing steps 1, 2, 3, 4, or 5.



**Important:** Once you blow the JTAG fuse (steps 6 or 7), you cannot perform any further iterations!

Each time you want to blow fuses for a new iteration, you use the Efinity Bitstream Security Key Generator to create a new **.svf** file with the new options that you want to enable.



**Important:** Do not enable options that you have already turned on. For example, if you already blew the AES-256 fuses, do not try to blow them again.

**Example 1: Blow Fuses for AES-256 First, Fuses for RSA Authentication Later**

You already blew fuses for AES-256 and now you want to blow fuses for RSA authentication:

1. Open the Efinity Bitstream Security Key Generator.
2. Turn *off* the **AES-256 Bitstream Encryption** option.
3. Turn *on* the **RSA-4096 Asymmetric Bitstream Authentication** option and generate or select a **.pem**.
4. Click **Generate** to create a new **.svf**; discard the **.bin** file.
5. Use the new **.svf** with the SVF Player to blow the RSA fuses; discard the **.bin** file.

**Example 2: Blow Fuses for AES-256 and RSA Authentication First, Fuse for Disabling JTAG Later**

You already blew fuses for AES-256 and RSA authentication and now you want to blow the JTAG fuse:

1. Open the Efinity Bitstream Security Key Generator.
2. Turn *off* the **AES-256 Bitstream Encryption** option.
3. Turn *off* the **RSA-4096 Asymmetric Bitstream Authentication** option.
4. Choose **ON** for **JTAG Disabling**.
5. Click **Generate** to create a new **.svf**; discard the **.bin** file.
6. Use the new **.svf** with the SVF Player to blow the JTAG fuse.

**Securing Bitstreams Iteratively**

You can secure the bitstream with encryption and/or authentication. When you enable either option (or both) in the Project Editor, you need to specify the **.bin** file you create with the Efinity Bitstream Security Key Generator.



**Note:** When working iteratively, you need to make sure that you use the same key data that you used in the previous iteration.

**Example 3: Secure Bitstream for AES-256 First, RSA Authentication Later**

You already enabled for AES-256 and now you want to enable RSA authentication:

1. Open the Efinity Bitstream Security Key Generator.
2. Turn on the **AES-256 Bitstream Encryption** option and enter the key from the previous iteration (this is why you should save it).
3. Turn on the **RSA-4096 Asymmetric Bitstream Authentication** option and generate or select a **.pem**.
4. Click **Generate** to create a new **.bin** file; discard the **.svf** file.
5. Specify the new **.bin** file in the Project Editor.
6. Generate the bitstream.

Example 1 and Example 3 both start with AES-256 and later add RSA authentication. However, you *turn off* AES-256 for Example 1 and *turn on* AES-256 for Example 3. Therefore, you need to run the Efinity Bitstream Security Key Generator twice: the first time with settings for blowing fuses; the second time with settings for bitstream security.

Example 2 only blows the JTAG fuse, so you use the **.svf** file with the SVF Player and discard the **.bin** file.

## Verifying Security Settings

You may want to verify that your Titanium FPGA is correctly using the security features that you enabled. You can use the **Advanced Device Configuration Status** dialog box (Programmer) to view the security status signals. See [Configuration Status Register](#) on page 32 for details.



**Note:** With the AES encryption feature enabled, Titanium FPGAs accept both encrypted and unencrypted bitstreams as valid. So you can configure the FPGA with a plaintext bitstream even after you blow its fuses with an AES key.

Conversely, if you have blown fuses for RSA authentication, the FPGA only accepts a bitstream signed with the private key you blew into the fuses.

Figure 8: Advanced Device Configuration Status Security Signals

Signal	Status
0 IN_USER	0
1 CDONE	0
2 NSTATUS	0
3 CRC32_ERRO...	0
4 RMUPD_ERROR	0
5 CONFIG_END	1
6 SYNC_PAT_FO...	1
7 SEU_ERROR	0
8 CRC32_ERRO...	0
9 AES256_PASS	1
10 RSA_PASS	0
11 RSA_ACTIVE	0
12 AES_ACTIVE	0

Update

You can also test out the bitstream security features by trying to program the FPGA with a bitstream that you signed with the wrong RSA key, an unsigned bitstream, or a bitstream encrypted with the wrong key. If the Titanium FPGA detects a key mismatch, it will not go into user mode.

## JTAG Command Support with Security Enabled

Titanium and Topaz FPGAs support additional bitstream security by letting you disable JTAG commands completely or partially:

- *JTAG Disable*—Permanently disables the JTAG efuse instruction as well as all other JTAG commands except for the ones used to read device information.
- *JTAG Disable Efuse*—Permanently disables the JTAG efuse instructions only. Other JTAG instructions are not affected, for example, you can still perform debugging.

The following table shows the commands supported for each mode.

**Table 19: Allowed JTAG Commands with Security Enabled**

Command	JTAG Disable				JTAG Disable Efuse		
	Ti35, Ti60, Tz50	Ti85, Ti135, Tz75, Tz100	Ti90, Ti120, Ti180, Tz110, Tz170	Ti165, Ti240, Ti375, Tz200, Tz325	Ti85, Ti135, Tz75, Tz100	Ti90, Ti120, Ti180, Tz110, Tz170	Ti165, Ti240, Ti375, Tz200, Tz325
BYPASS	✓	✓	✓	✓	✓	✓	✓
DEVICE_STATUS	✓	✓	✓	✓	✓	✓	✓
EFUSE_PREWRITE	-	-	-	-	-	-	-
EFUSE_USER_WRITE	-	-	-	-	-	-	-
EFUSE_WRITE_STATUS	-	-	-	-	-	-	-
ENTERUSER	-	-	-	-	✓	✓	✓
EXTEST	-	-	-	-	✓	✓	✓
IDCODE	✓	✓	✓	✓	✓	✓	✓
INTEST	-	-	-	-	✓	✓	✓
JTAG_USER1	-	-	-	-	✓	✓	✓
JTAG_USER2	-	-	-	-	✓	✓	✓
JTAG_USER3	-	-	-	-	✓	✓	✓
JTAG_USER4	-	-	-	-	✓	✓	✓
PROGRAM	-	-	-	-	✓	✓	✓
SAMPLE/PRELOAD	✓	✓	✓	✓	✓	✓	✓
USERCODE	-	✓	✓	✓	✓	✓	✓

Refer to the following topics for details:

- [Securing Titanium Bitstreams](#) on page 37
- [Using the Efinity Bitstream Security Key Generator](#) on page 40

# Working with JTAG .svf Files

The JTAG serial vector format (**.svf**) file is a vendor-independent ASCII text file of JTAG commands. You can use an **.svf** file for JTAG debugging, boundary-scan testing, and programming with any **.svf**-compatible JTAG hardware.

The Efinity Programmer can convert a bitstream file to **.svf** so that you can use third-party JTAG hardware to program an Efinix FPGA. Refer to [Export to .svf Format](#) on page 10.

JTAG programming with an **.svf** file is supported in all Efinix FPGAs *except* for:

- T4, T8, and T13 in any package
- T20 in W80, Q144, F169, and F256 packages

## Using the Efinity SVF Player

The Efinity SVF Player is a JTAG SVF player that sends JTAG commands to an FPGA. The player reads the JTAG commands from a serial vector format (**.svf**) file. You can use the SVF Player without opening a project. The Efinity SVF Player requires a JTAG cable or mini-module with the FTDI *n232H* chipset.

You can use the SVF Player to execute any JTAG commands on the following Efinix FPGAs:

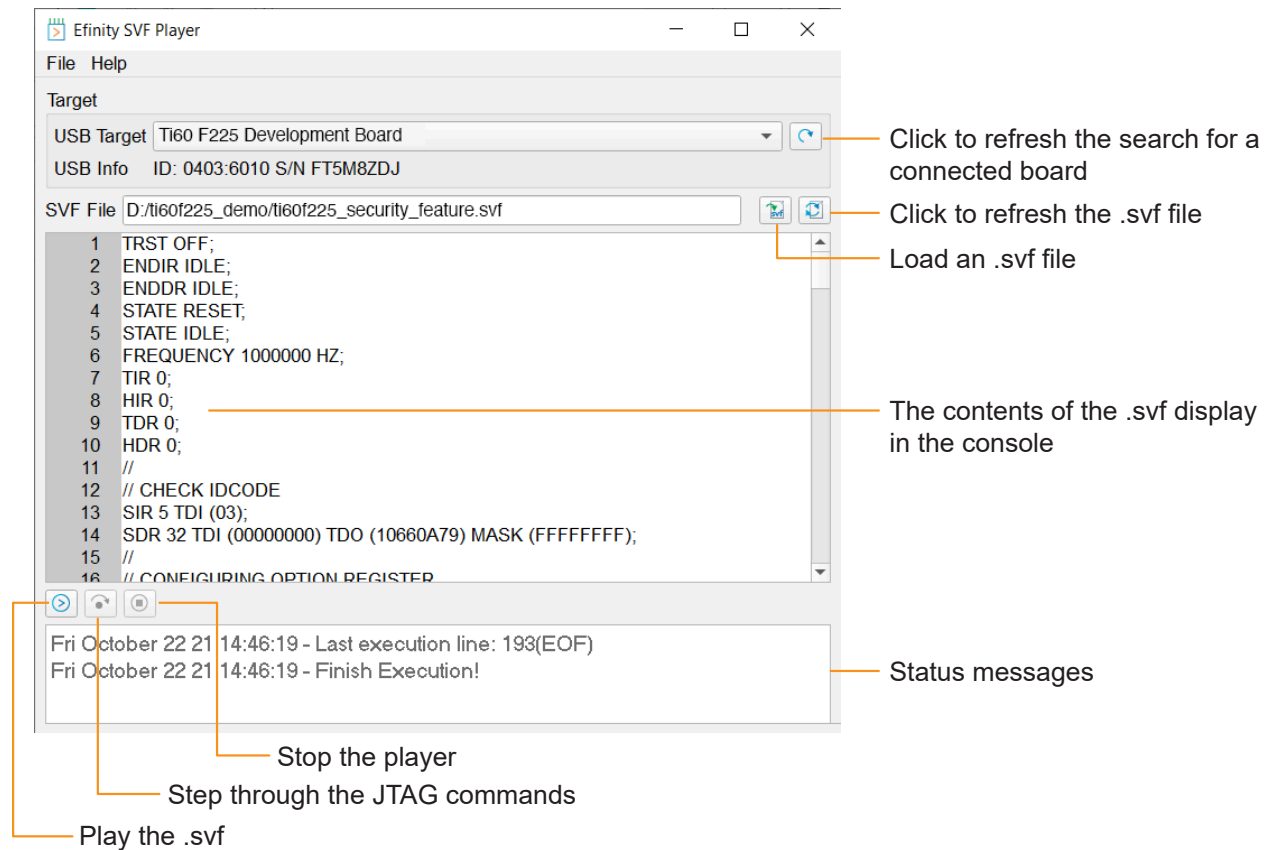
- Trion T20 in F324 and F400 packages
- Trion T35 in any package
- Trion T55, T85, and T120 in any package
- All Titanium and Topaz FPGAs in any package

You can use the the SVF Player to execute any JTAG command *except* PROGRAM for the following Trion FPGAs:

- T4, T8, and T13 in any package
- T20 in W80, Q144, F169, and F256 packages

You can also use the SVF Player to execute JTAG commands for non-Efinix devices in a JTAG chain.

Figure 9: SVF Player



To use the SVF Player:

1. Choose a **USB Target**. Ensure that your board is connected to your computer and turned on. Click the Refresh button to search for newly connected boards.
2. Click the Open SVF File button to load the **.svf**. The content of the **.svf** displays in the console.



**Note:** If you make changes to the **.svf**, you can reload it using the Reload button.

3. Click the Play button to play the **.svf** file.

You can also step through the **.svf** file line by line using the Step Over button. This feature is useful for debugging. To stop playing the file, click the Stop button.

## Export JTAG Operations at the Command Line

The Efinity software includes a Python script you can use to export JTAG operations of the JTAG bridge to an SVF file at the command line.

When using the **export\_bitstream\_flashloaderv3.py** script, you must connect to the FPGA board using a download cable.

```
export_bitstream_flashloaderv3.py [--help] --jtag bridge file JTAG BRIDGE FILE [--freq FREQ]
[--start_address START_ADDRESS] [--target TARGET] [--chip_num CHIP_NUM]
[--jcf JCF] [--url URL] [--verify_method VERIFY_METHOD] input_file output_file
```

Table 20: `export_bitstream_flashloaderv3.py` Positional Arguments

Argument	Description
input_file	Image file source.
output_file	Image file destination.

Table 21: `export_bitstream_flashloaderv3.py` Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--jtag_bridge_file	N/A	File path	The file path of the JTAG-to-SPI Bridge bitstream.
--freq	N/A	Number	JTAG frequency. Default: 6e6
--start_address	N/A	Hex number	Starting flash address for flash read and write operations, excluding 0x.
--target	N/A	lower, upper, both	Target flash. <code>both</code> is used for the x8 mode.
--chip_num	N/A	Number	1-based device position in the JTAG chain. Default: 1
--jcf	N/A	File path	JTAG chain file in XML format. Not needed if there is only one device.
--url	-u	URL	FTDI URL (see <a href="#">Identifying FTDI URLs</a> on page 26).
--verify_method	N/A	none, onchipx1, onchipx2, onchipx4	The method used to verify the downloaded bitstream. Default: onchipx2 (On-chip hash calculation with SPI x2 mode)

**Example: Export JTAG Bridge Operations to an SVF File**

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\efx_pgm\export_bitstream_flashloaderv3.py bitstream.hex
output.svf --jtag_bridge_file %EFINITY_HOME%\pgm\fli\titanium\u006A0A79.bit
```



**Note:** Load the JTAG Bridge bitstream into the FPGA before using the exported SVF file. The output file of the `export_bitstream_flashloaderv3.py` script does not include the JTAG Bridge bitstream.

## Run the `export_bitstream_flashloaderv3` Python Script

To create an SVF file:

1. Generate your bitstream file.
2. Run the `export_bitstream_flashloaderv3.py` script. It will download the JTAG bridge into the FPGA, then program the SPI flash with the bitstream file. Only the SPI flash programming actions of the JTAG operations will be captured.



**Note:** If you are using an Efinix development board, you can connect to the board using a USB cable. If not, use an external download cable.

To use the created SVF file:

1. Download the JTAG bridge into the FPGA.

2. Execute the created SVF file to program the SPI flash with your project bitstream.

## Where to Learn More

The Efinity<sup>®</sup> software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software. You can also access the latest versions of PDF documentation in the [Support Center](#):

- [Efinity Software User Guide](#)
- [Efinity Software Installation User Guide](#)
- [Efinity Synthesis User Guide](#)
- [Efinity Timing Closure User Guide](#)
- [Efinity Trion Tutorial](#)
- [Efinity Debugger Tutorial](#)
- [Efinity Programmer User Guide](#)
- [Efinity IP Packager User Guide](#)
- [Trion Interfaces User Guide](#)
- [Titanium Interfaces User Guide](#)
- [Topaz Interfaces User Guide](#)
- [Efinity Interface Designer Python API](#)
- [Efinity Command-Line Interface User Guide](#)
- [Quantum<sup>®</sup> Trion Primitives User Guide](#)
- [Quantum<sup>®</sup> Titanium Primitives User Guide](#)
- [Quantum<sup>®</sup> Topaz Primitives User Guide](#)

In addition to documentation, Efinix field application engineers have created a series of videos to help you learn about aspects of the software. You can view these videos in the [Support Center](#).

## Appendix: Installing USB Drivers

To program Trion<sup>®</sup>, Topaz, and Titanium FPGAs using the Efinity<sup>®</sup> software and programming cables, you need to install drivers.

Efinix development boards have FTDI chips (FT232H, FT2232H, or FT4232H) to communicate with the USB port and other interfaces such as SPI, JTAG, or UART. Refer to the Efinix development kit user guide for details on installing drivers for the development board.



**Note:** If you are using more than one Efinix development board, you must manage drivers accordingly. Refer to [AN 050: Managing Windows Drivers](#) for more information.



**Notice:** The Trion T8 BGA81 Development Boards do not have FTDI chip for USB communication. Refer to the T8 BGA81 Development Kit User Guide for more information about installing its Windows USB driver.

For your own development board, Efinix suggests using the FTDI Chip FT2232H or FT4232H Mini Modules for JTAG programming Trion<sup>®</sup>, Topaz, and Titanium FPGAs. (You can use any JTAG cable for JTAG functions other than programming.)



**Note:** Efinix does not recommend the FTDI Chip C232HM-DDHSL-0 programming cable due to the possibility of the FPGA not being recognized or the potential for programming failures.

*Table 22: USB Programming Connections*

Board	Connect to Computer with
Efinix development boards	USB cable
Your own board	FTDI x232H programming kit. For example: <ul style="list-style-type: none"> <li>• FT2232H Mini Module</li> <li>• FT4232H Mini Module</li> </ul>



**Note:** The FTDI Chip Mini Module supports 3.3 V I/O voltage only. Refer to the [FTDI Chip website](#) for more information about the modules.

## Installing the Windows libusbK Driver

Efinix provides a custom Windows libusbK driver installer for enabling FTDI communication with Efinix development boards (before March 2026 you had to download software from Zadig). You install drivers for:

- Interface 1 (JTAG communication)
- Interface 0 (SPI communication)



**Important:** For some Efinix development boards, Windows automatically installs drivers for some interfaces when you connect the board to your computer. You do not need to install another driver for these interfaces. Refer to the user guide for your development board for specific driver installation requirements.

**Tip:** Download the [Windows FTDI driver](#) from the [Support Center](#).

To install the driver:

1. Unzip the driver files into a temporary directory.
2. Right click **efx\_libusbk\_amd64\_i1.inf**.
3. Choose **Install** option from the pop-up menu.
4. Click **Install** when prompted by Windows Security to install the device software.
  - (Optional) Turn on the **Always trust software from 'Efinix, Inc.'** option.
5. Click **OK** when prompted that operation has completed successfully.
6. (Optional) Enable Interface 0 by repeating steps 1 - 4 with the **efx\_libusbk\_amd64\_i0.inf** file.

The new driver(s) appear in the Windows **Device Manager Universal Serial Bus** folder.

## Troubleshooting Driver Installation

If the installed drivers do not automatically bind to the target device, try these steps:

1. Open the Windows **Device Manager**
2. Identify the target device for driver binding (e.g., Ti60F225 Development Kit (Interface 1)).
3. Double-click the device name to open the **Properties** window.
4. Click the **Driver** tab.
5. Click **Update Driver**.
6. Choose **Browse my computer for drivers** and **Let me pick from a list of available drivers on my computer**.
7. Select the newly installed driver, e.g., Efinix libusbK USB Device (FT4232H).



**Note:** Depending on the target development kit, a different FTDI device may display. See the "Supported FTDI Devices" section below for a full list of devices.

8. Click **Next** to update drivers for the device.
9. Click **Close** once prompted that drivers have been successfully updated.

If you are cannot identify which device or interface is listed in the Device Manager:

1. Open the Device Manager.
2. Double-click the device to open the device properties.
3. Click the **Details** tab.
4. Choose **Hardware Ids** in the **Property** dropdown menu.
5. Identify the FTDI device and interface number using the following schemes based on the **Value**:
  - See the following table for a list of supported devices and their respective VIDs and PIDs.
  - For multi-interface devices:
    - Interface 0: ends in MI\_00
    - Interface 1: ends in MI\_01
  - Example `Hardware Ids` entry for FT4232H (Interface 1): USB \VID\_0403&PID\_6011&MI\_01

*Table 23: Supported FTDI Devices*

<b>Device</b>	<b>VID</b>	<b>PID</b>
FT232H	0x0403	0x6014
FT232HP	0x0403	0x6045
FT233HP	0x0403	0x6044
FT2232H	0x0403	0x6010
FT2232HP	0x0403	0x6042
FT2233HP	0x0403	0x6040
FT4232H	0x0403	0x6011
FT4232HP	0x0403	0x6043
FT4233HP	0x0403	0x6041
FT4232HA	0x0403	0x6048

## Appendix: Program using a JTAG Bridge (Legacy)

Programming with a JTAG bridge is a two-step process: first you configure the FPGA to turn it into a flash programmer (**.bit**) and second you use the FPGA to program the flash device with the bitstream (**.hex**).

The Trion<sup>®</sup>, Topaz, and Titanium **.bit** files include a custom JTAG USERCODE in the bitstream:

- Single flash **.bit** files—0x6212E80D
- Dual flash **.bit** files—0xFA828A14

To program using a JTAG bridge:

1. Choose the **USB Target**.
2. In the **Image** box, click the **Select Image File** button to browse for the **.hex** file to program the flash device.
3. Choose the **SPI Active using JTAG Bridge (Legacy)** or **SPI Active x8 using JTAG Bridge (Legacy)** mode.
4. Turn on the **Auto configure JTAG Bridge Image** option.

For Titanium Topaz FPGAs, the Programmer automatically loads the **.bit** file. Skip step 5 if you want to use the pre-loaded **.bit** file.

5. Specify the **.bit** file.
  - a) In the **Programming Mode** box, click **Select Image File**.
  - b) The **Open Image File** dialog box opens. Browse to find your own **.bit** file.
6. Click **Start Program**. The Programmer first configures the FPGA and then programs the flash device.



**Notice:** Refer to the [JTAG SPI Flash Loader Core User Guide](#) for instructions on creating the **.bit** file.



**Important:** If you are using the Titanium Topaz RSA bitstream authentication security feature, you need to use a signed **.bit** file. Copy the bundled **.bit** file from the appropriate source folder to another directory and sign it. Then point to the signed **.bit** file in the Programmer. You can also create your own **.bit** file with the JTAG Flash Loader IP core if you prefer. Depending upon your board, the source folder is:

- `<Efinity version>/pgm/fli/titanium`
- `<Efinity version>/pgm/fli/topaz`

Refer to [Using the Efinity Bitstream Security Key Generator](#) on page 40 for information on signing existing **.bit** files.

# Revision History

Table 24: Revision History

Date	Version	Description
March 2026	4.2	Added <b>Programmer Messages</b> on page 27. (DOC-2862) Added instructions for installing the Efinix <b>Windows libusbK driver</b> and <b>Troubleshooting Driver Installation</b> on page 54.
December 2025	4.1	For Titanium and Topaz FPGAs, the default setting for the <b>Project Editor &gt; Bitstream Generation tab &gt; Clock Sampling Edge</b> option is <b>Falling</b> (not <b>Rising</b> ). (DOC-2804)
November 2025	4.0	Added note about padding in combined bitstream images in <b>Combine Bitstreams and Other Files</b> on page 12.
September 2025	3.9	Added GigaDevice GD25LQ256 to list of supported flash devices. (DOC-2711)
August 2025	3.8	Added <b>Export JTAG Operations at the Command Line</b> on page 50. (DOC-2569) Added options to <b>FTDI Programming at the Command Line</b> on page 22. Moved <b>Identifying FTDI URLs</b> on page 26 into a separate topic. (DOC-2658).
May 2025	3.7	Corrected path to ftdi_program.py. (DOC-2555)
May 2025	3.6	Updated hardware and software requirements.
November 2024	3.5	Fixed various typos.
November 2024	3.4	Updated entry for SYNC_PAT_FOUND in <b>Table 13: Configuration Status Register</b> on page 33. (DOC-2181) Update JTAG IDs for Ti375, Ti135, and Ti85. (DOC-1851) Corrected link to latest Microsoft Visual C++ Redistributable downloads. (DOC-2045) Added topic about encrypting and/or signing bitstreams at the command line.
June 2024	3.3	The software has separate <b>.bit</b> files for JTAG Bridge (New) and JTAG Bridge (Legacy), and they are not compatible with each other. The <b>.bit</b> files do not require an external clock. (DOC-1789)
May 2024	3.2	Added Ti165 and Ti240 FPGAs, replacing the Ti135 and Ti240, respectively.
January 2024	3.1	Added JTAG device IDs for Ti135, Ti240, and Ti375. (DOC-1662) Added Ti135 and Ti240 to machine memory requirements. Added instructions on installing patches. Added note about Windows %PATH% variable. (DOC-1687)
December 2023	3.0	Added G400 package support. (DOC-1393) Added Program using a JTAG Bridge (New) modes. (DOC-1542) 64-bit operating system is required. 32-bit systems are not supported.
June 2023	2.9	Added JTAG device ID for Trion Q100F3, and Titanium J361, J484, and G529 packages. (DOC-1165)

Date	Version	Description
May 2023	2.8	<p>Added note about referring to AN050: Managing Windows Drivers in the Installing USB Drivers topic. (DOC-977)</p> <p>Removed Verifying Configuration with Programmer topic. The topic is in AN006 and AN033.</p> <p>Added IS25LP128 to list of supported flash devices. (DOC-1247)</p>
November 2022	2.7	<p>Updated supported flash devices. (DOC-896)</p> <p>Corrected faint/missing callout lines in Figure 2 and 3. (DOC-976)</p>
August 2022	2.6	<p>Added installation instructions.</p> <p>Added instructions on using the JTAG SVF Player.</p> <p>Added topics on the Bitstream Security Key Generator.</p> <p>Clarified that when using internal reconfiguration you must use <b>Programmer &gt; Combine Multiple Image Files &gt; Image Type &gt; Internal Flash Image</b> option. (DOC-874)</p> <p>Added topic on verifying configuration with the Programmer.</p> <p>When editing the bitstream header, do not remove any auto-generated data or the Programmer may not recognize the bitstream. (DOC-868)</p> <p>Removed support for C232HM-DDHSL-0 cable. (DOC-860)</p> <p>Updated Installing USB Drivers topics.</p>
April 2022	2.5	<p>Added Program using a JTAG Bridge topic.</p> <p>Added topic on combining a bitstream and other data into a single file for programming.</p> <p>Re-organized topics about working with bitstreams.</p>
December 2021	2.4	<p>Updated machine memory requirements (RAM).</p> <p>Added information about connecting to remote hosts.</p> <p>Added Macronix MX75L and MX75U to supported flash devices. (DOC-573)</p> <p>Added support for FTDI FT4232H Mini Module. (DOC-597)</p> <p>With the Efinity software v2021.2 and higher, you <b>must</b> use <b>.hex</b> for SPI and <b>.bit</b> for JTAG. (DOC-638)</p>
October 2021	2.3	<p>Added topic on the Titanium configuration status registers. (DOC-487)</p> <p>Added topic on flash programming modes.</p> <p>Added note about FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495)</p> <p>Added XT25F family to list of supported flash devices. (DOC-529)</p>
June 2021	2.2	<p>Updated Windows driver installation instructions.</p> <p>Added SPI Active x8 over JTAG Bridge mode.</p> <p>Added Titanium JTAG IDs.</p> <p>Updated list of supported flash devices.</p>
January 2021	2.1	<p>Corrected JTAG chain file code example. (DOC-368)</p>

Date	Version	Description
December 2020	2.0	<p>Added the requirement to install the Microsoft Visual C++ 2015 x64 and x86 runtime libraries for the standalone Programmer.</p> <p>Added JTAG device IDs for T20BGA324 and T20BGA400.</p> <p>Added FTDI cable and module connection for T20BGA400.</p> <p>Removed the FTDI2232 from About USB Drivers topic making the description applicable to other FTDI chips.</p> <p>Updated instructions on installing USB drivers for Windows.</p> <p>Added list of supported flash devices.</p> <p>Corrected JTAG Mini Module pin names for T4, T8, T13, T20BGA256, and T20BGA169 connection setup.</p>
June 2020	1.0	Initial release.