Efinity® Programmer User Guide

UG-EFN-PGM-v3.5
November 2024
www.efinixinc.com

Copyright © 2024. All rights reserved. Efinix, the Efinix logo, the Titanium logo, the Topaz logo, Quantum, Trion, and Efinity are trademarks of Efinix, Inc. All other trademarks and service marks are the property of their respective owners. All specifications subject to change without notice.

Contents

Introduction

Efinix 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® 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® software to program Trion®, Topaz, and Titanium FPGAs.

Learn more: For information on generating a bitstream file or on using the Efinity® software, refer to the Efinity Software User Guide.

Software Requirements

- Windows 8.1 or later, x86 and x64 libraries (for 64-bit systems)
- A 64-bit Windows system is required for using the security tools in the Efinity standalone programmer.
- Microsoft Visual C++ 2019 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

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 39 for instructions on installing the drivers.

Installing Patches

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

Windows

1. Download the patch from the Efinity® 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

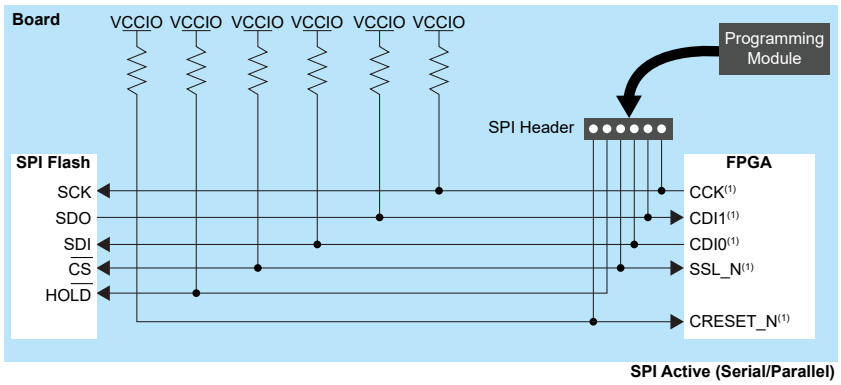
The following table shows the methods you can use to program the configuration bitstream into the flash device on your board. Although you can program the flash directly using the SPI interface, this method requires that you have a SPI header on your board or use an FDTI chip. Therefore, Efinix recommends 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.

Table 2: Flash Programming Modes

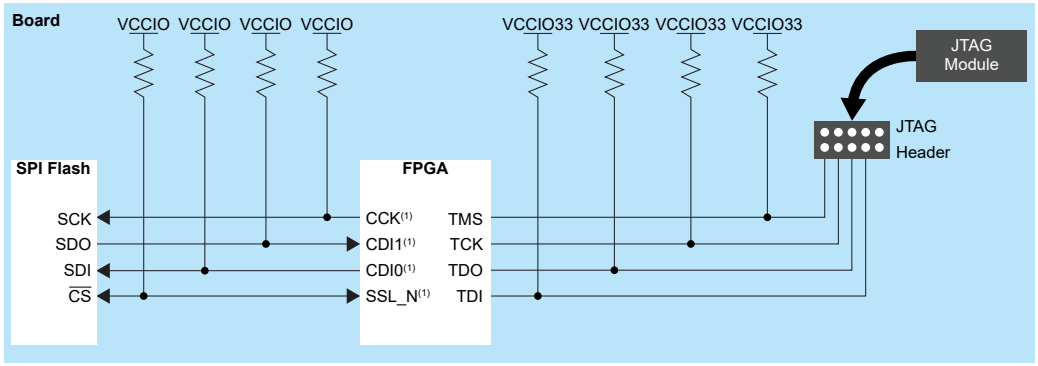
Mode	Description
SPI Active (serial/parallel)	Use the Efinity Programmer and a cable connected to a SPI header on the board.
SPI Active using JTAG Bridge (New)	A improved version of the SPI Active using JTAG Bridge (Legacy) mode with a faster flash programming time.
SPI Active x8 using JTAG Bridge (New)	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 (New) on page 17 for more information.

Figure 1: Flash Programming Board Setup

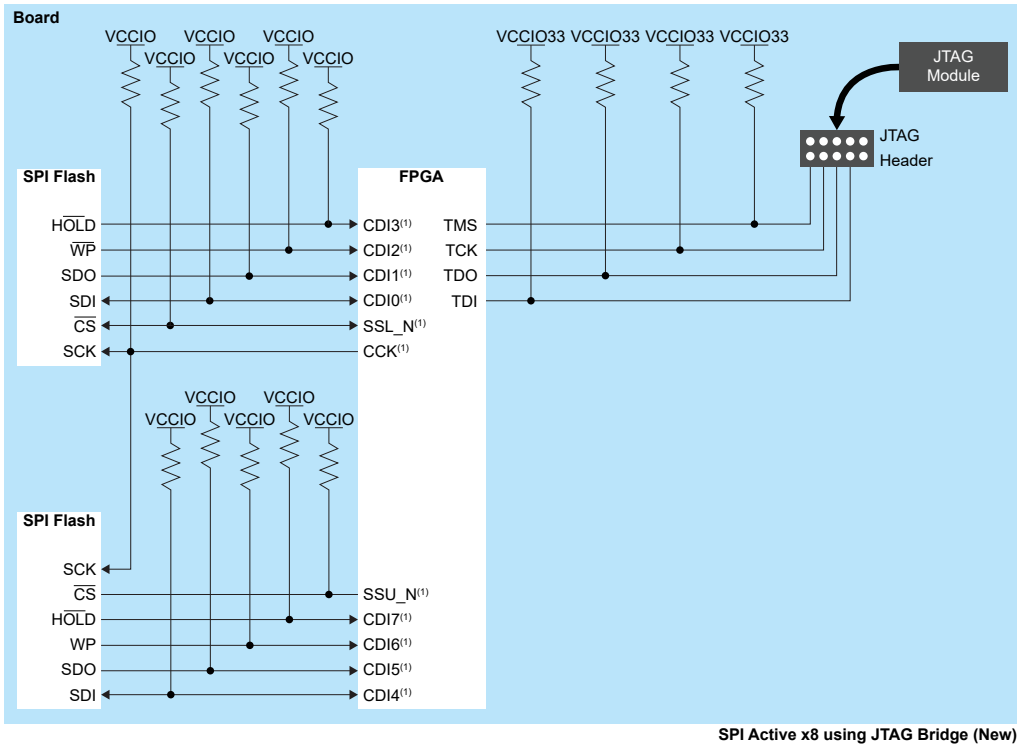


SPI Active (Serial/Parallel)



SPI Active using JTAG Bridge (New)

[1] The external pull-up is optional unless required by an external load.



SPI Active x8 using JTAG Bridge (New)

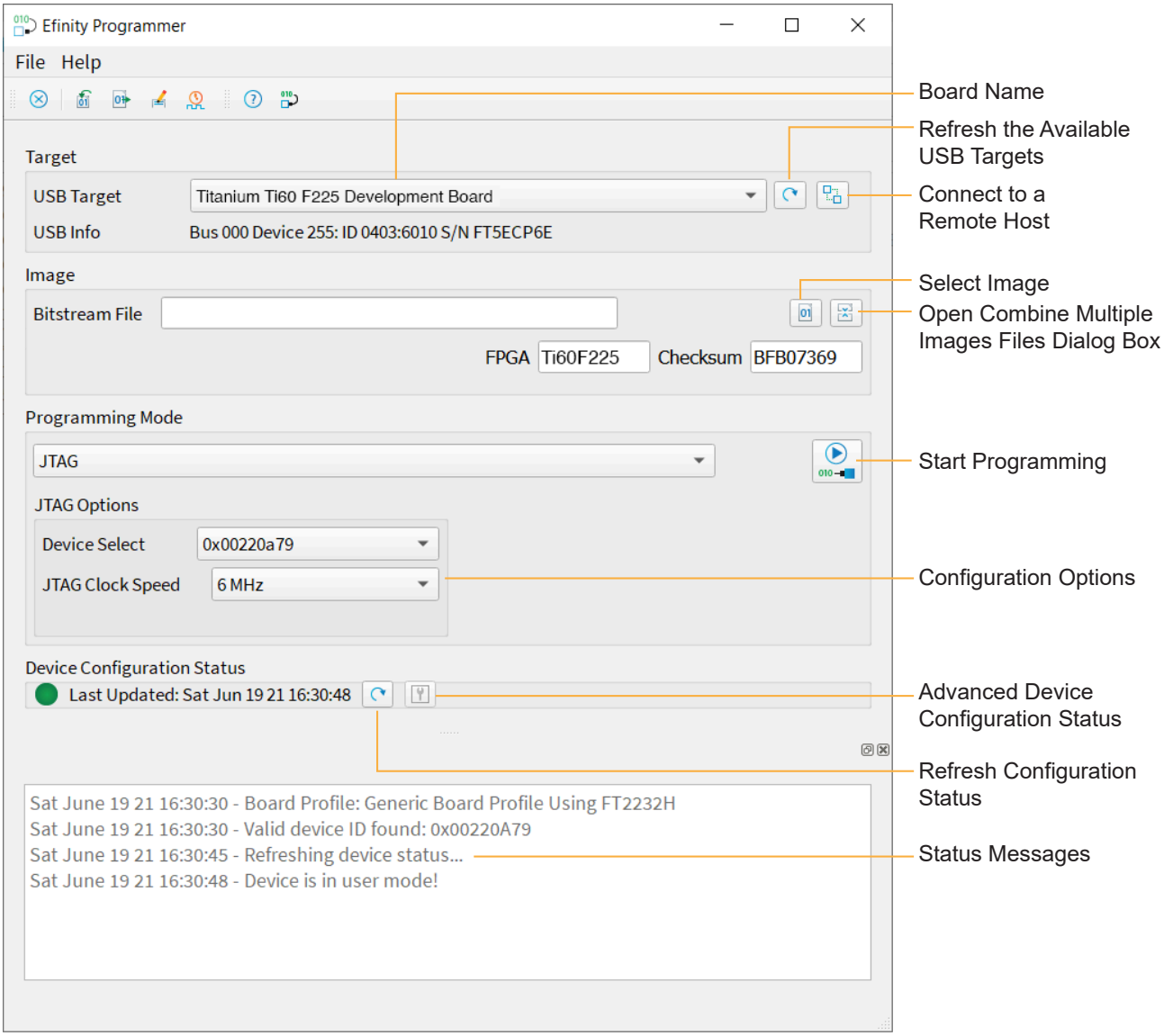
[1] The external pull-up is optional unless required by an external load.

Note: Be sure to hold CRESET_N low to prevent signal contention during SPI Flash Programming.

About the Programmer GUI

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

Figure 2: 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 24.
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 22 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® 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® 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 10.

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 36.

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 10.

Convert to Intel Hex Format at the Command Line

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

```
export_bitstream.py [-h] [--family <Trion®, Topaz, and Titanium>] [--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 or off>]
    <format> <input filename> <output filename>
```

Where <*format*> is:

- hex_to_bin
- hex_to_intelhex
- bin_to_hex
- intelhex_to_hex
- hex_to_svf

For example:

```
C:\Efinity\2021.1\bin\setup.bat
python3 C:\Efinity\2021.1\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 3: Modes when Combining Images

Mode	Use For	Number of Images	Refer to
Selectable Flash Image	Multi-image configuration	Up to 4	Program Multiple Images (CBSEL) on page 11
	Internal reconfiguration	Up to 4	Program Multiple Images (Internal Reconfiguration) on page 12
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors	Program a Daisy Chain on page 13
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	Program Multiple Images (Bitstream and Data) on page 13

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 **External Flash 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 programming multiple images, refer to **Example Design: Configuring a Trion Development Board with Multiple Images** on the Downloads page in the Support center.

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 **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**.

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.

JTAG Device IDs

The following tables listtable lists the Topaz, Titanium, and Trion JTAG device IDs.

Table 4: 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

Table 5: 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, G400, G529	0x00690A79
	L484	0x00684A79
Ti240	All	0x006A2A79
Ti375	All	0x006A0A79

Table 6: Trion JTAG Device IDs

FPGA	Package	JTAG Device ID
T4, T8	BGA81	0x0
T8	QFP144	0x00210A79
T13	All	0x00210A79

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

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®, 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: 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**.

Program using a JTAG Bridge (New)

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 (New) mode, is an improved version of the legacy SPI Active using JTAG Bridge mode, and is available in the Efinity software v2023.2 and higher. This mode is substantially faster than the legacy mode and 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.

Notice: If you would like to incorporate the RTL files for the new and improved flash loader into your own design, the files are located in the *<Efinity directory>***/pgm/rtl/spiloaderv2** directory.

The Trion®, Topaz, and Titanium**.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 the new JTAG bridge, as well as the legacy JTAG bridge. The files are not interchangeable; therefore, if you are choosing the **.bit** file yourself, make sure that you choose the correct one for the JTAG bridge mode you are using. Legacy ones are in the *<Efinity version>***/pgm/fli/titanium/legacy/pgm/fli/topaz/legacy** directory.

For Trion FPGAs, you need to specify the pre-built file to use.

Note: Efinix strongly recommends that you use the default new mode for JTAG bridge programming. However, if you would like guidance on using the legacy JTAG bridge, please refer to the **Efinity Software User Guide**.

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 (New)** or **SPI Active x8 using JTAG Bridge (New)** 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. (Optional) Specify the **.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.

   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. 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 29 for information on signing existing **.bit** files.

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

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®, 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. (Optional) 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 29 for information on signing existing **.bit** files.

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**.

FDTI Programming at the Command Line

The Efinity® includes a script, **ftdi_program.py**, which you can use for command-line programming with FTDI modules. The command is in the format:

```
ftdi_program.py <filename>.bit -m <mode> --url <url> --aurl <url>
```

<*mode*> is the programming mode:

- `active`, `passive`
- `jtag`, `jtag_chain`
- `jtag_bridge_new`, or `jtag_bridge_x8_new`[1] (new mode, see **Program using a JTAG Bridge (New)** on page 17)
- `jtag_bridge`, or `jtag_bridge_x8`[2] (legacy mode, see **Program using a JTAG Bridge (Legacy)** on page 18)

Note: To use the JTAG bridge modes, you must have already configured the FPGA with the JTAG SPI flash loader.

The Efinity software v2023.2 and higher includes pre-built flash loader **.bit** files in *<Efinity installation directory>***/pgm/fli/***<family>*.

Refer to the **JTAG SPI Flash Loader Core User Guide** for information on using the legacy flash loader.

Important: You only need to specify the `--url` and `--aurl` 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.

<*url*> is in the format:

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

where:

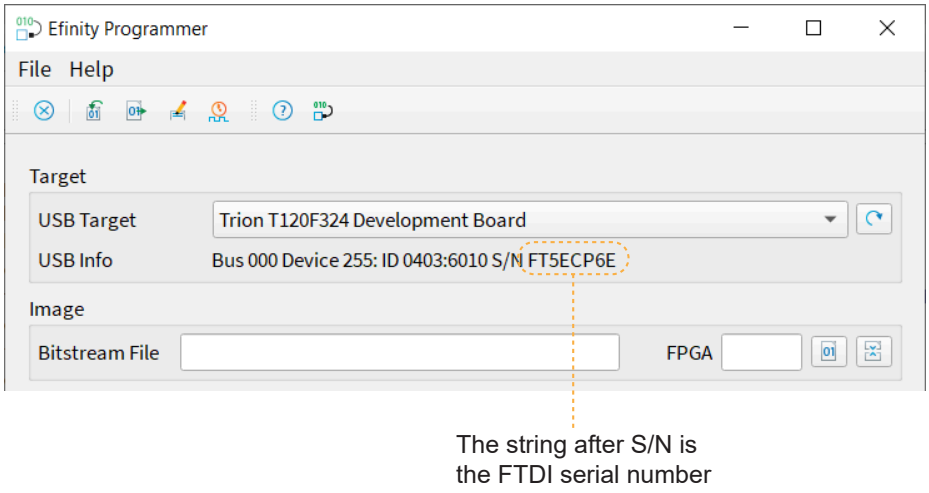
<*product*> is the USB product ID of the device

[1] The `jtag_bridge_x8_new` mode is only supported in some Titanium Topaz FPGAs. Refer to the data sheet for the modes your FPGA support
[2] The `jtag_bridge_x8` mode is only supported in some Titanium Topaz FPGA s. Refer to the data sheet for the modes your FPGA supports.

<product>	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

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

- If you only have one Efinix® development board or FTDI device connected to your computer, you do not need to specify the serial number.
- In the Efinity® 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`.



<*interface*> is the interface number. For Efinix® development boards, <*interface*> is always 1.

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, use:

```
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, use:

```
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, use:

```
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, use:

```
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, use:

```
%EFINITY_HOME%\bin\python3 %EFINITY_HOME%\pgm\bin\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, use:

```
%EFINITY_HOME%\bin\python3 %EFINITY_HOME%\pgm\bin\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, use:

```
%EFINITY_HOME%\bin\python3 %EFINITY_HOME%\pgm\bin\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, use:

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

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
```

Windows:

```
> efx_run.bat <project name>.xml --flow program
```

(Optional) Use these options:

- `--pgm_opts mode` specifies the configuration mode. The available modes are:
  - `active`—SPI active configuration
  - `passive`—SPI passive configuration
  - `jtag`—JTAG programming
  - `jtag_bridge`—SPI active using JTAG bridge mode
  - `jtag_bridge_x8`—SPI active x8 using JTAG bridge mode (used with two flash devices)[3]

  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.

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 confugration issues. View the register values in the **Advanced Device Configuration Status** dialog box, which you open by clicking the button of the same name.

Table 7: Configuration Status Register

Name	Description
IN_USER	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. **Note:** This bit is not supported in Ti60ES FPGAs.

[3] Used with two flash devices. Only supported in some Titanium Topaz FPGAs. Refer to the data sheet for the modes your FPGA supports.

Name	Description
CDONE	Configuration done, has the same value as the CDONE output pin. 0: The FPGA is not configured. 1: Configuation is complete.
NSTATUS	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.
CRC32_ERROR_CORE	0: No CRC errors were detected in the core configuration bits. 1: One or more CRC errors were detected in the core configuration bits.
RMUPD_ERROR	0: No errors occurred during remote update. 1: An error occurred during remote update configuration. Has the same value as the remote update error status signal sent to the core fabric.
CONFIG_END	0: Configuration is not complete. 1: Configuration completed (whether successful or not).
SYNC_PAT_FOUND	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.
SEU_ERROR	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.
CRC32_ERROR_PERIPH	0: No CRC errors were detected in the interface configuration bits. 1: One or more CRC errors were detected in the interface configuration bits.
AES256_PASS	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. **Note:** This bit is not supported in Ti60ES FPGAs.
RSA_PASS	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. **Note:** This bit is not supported in Ti60ES FPGAs.
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.

Name	Description
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.

Supported Flash Devices

Table 8: Supported Flash Devices

Manufacturer	Family Part Number
GigaDevice	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

Note: Efinix recommends using SPI NOR flash memories.

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 [-h] [-a <address>] [-p <port>]
```

Where:

- -h shows help.
- <*address*> is the 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*> is the 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 <Ffinity path>/pgm/bin/efx_pgm/efinity_hw_server.py
```

Stop the Efinity Hardware Server

In the terminal or Command Shell, 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 19.)

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.

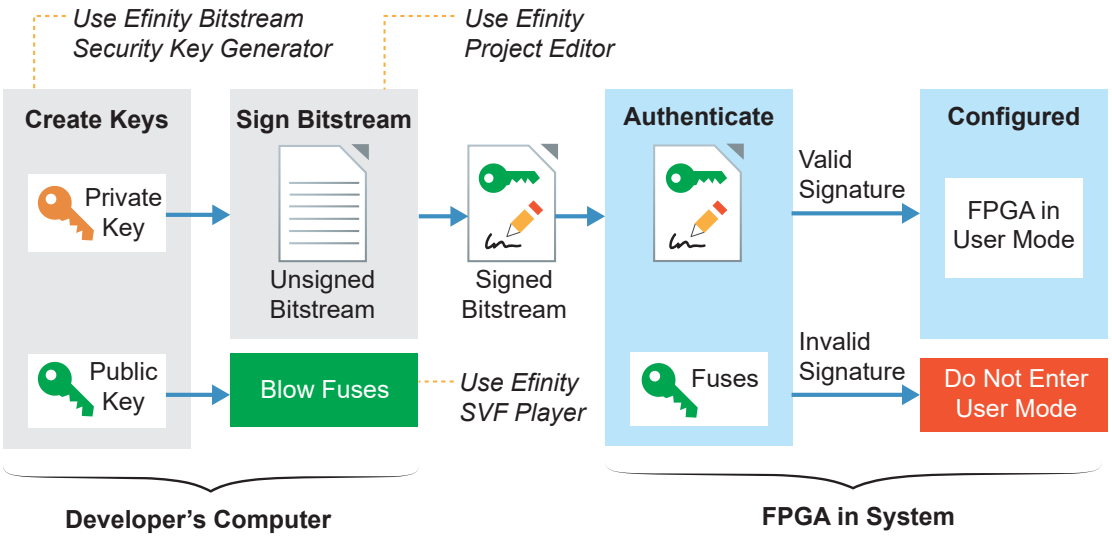
- *Encryption*—Encrypt your bitstream using an AES-256 key.
- *Authentication*—Sign your bitstream with an RSA-4096 private key.
- *Disable JTAG*—Disable all JTAG instructions except for `IDCODE`, `DEVICE_STATUS`, and `BYPASS`.

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

Table 9: 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. After you blow the JTAG fuse, you cannot use any JTAG command except `IDCODE`, `DEVICE_STATUS`, and `BYPASS`.
 Project Editor	Turn on bitstream encryption and/or authentication, and specify the **.bin** file created by the Bitstream Security Key Generator. Turn on bitstream authentication and specify the private key (**.pem**) file to sign the bitstream. **Note:** You need the full version of Efinity software to work with projects and to generate bitstreams. The Windows Standalone Programmer does nbot 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 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

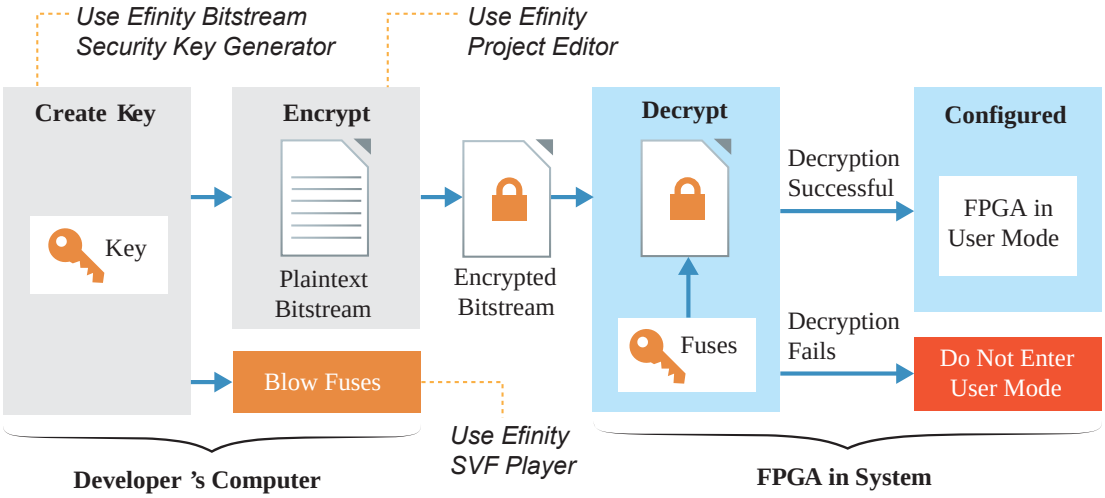
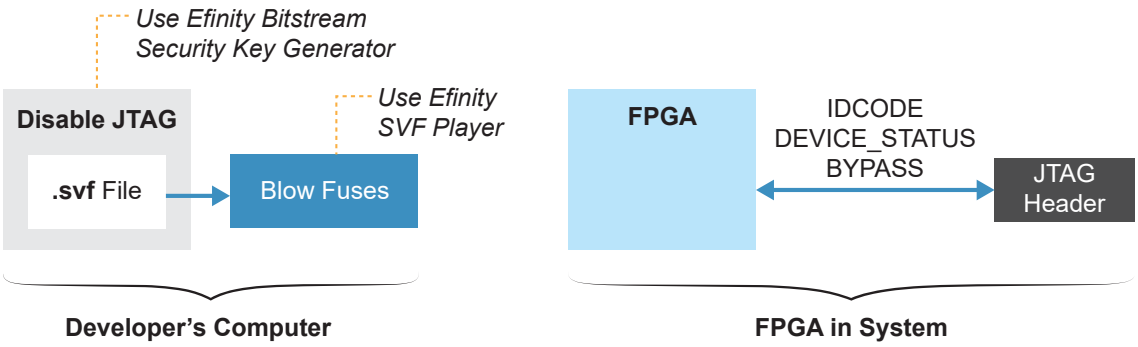


Figure 5: Disabling JTAG



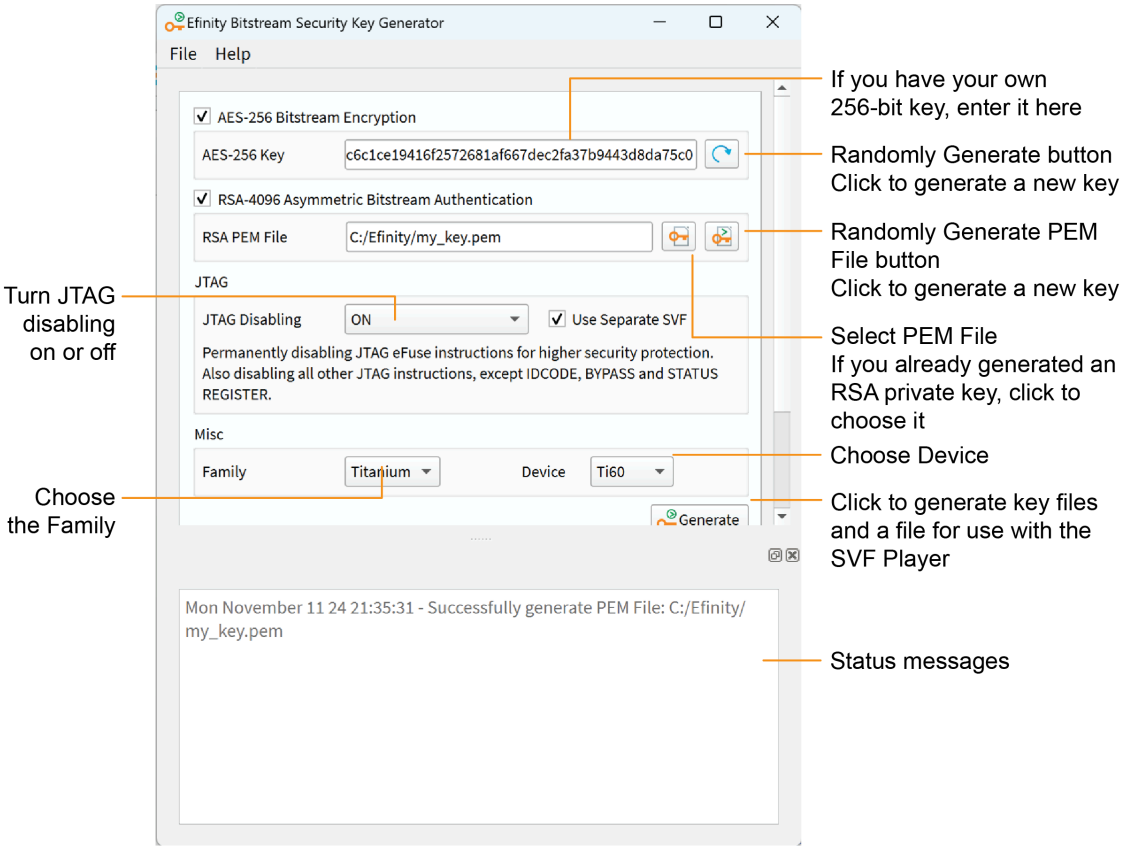
The following sections describe how to use each of these tools to enable security features.

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 34 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
OFF	No JTAG disabling. Efinix strongly recommends that you use **ON** or **DISABLE_EFUSE_ONLY** to disable access to the JTAG efuse instructions for added security.
ON	Permanently disables the JTAG efuse instructions as well as all other JTAG instructions except `IDCODE`, `BYPASS`, `SAMPLE/PRELOAD`, and `DEVICE_STATUS`.
DISABLE_EFUSE_ONLY	Available for all Titanium and Topaz FPGAs except: • Ti35 • Ti60 • Tz50 Permanently disables the JTAG efuse instructions only. Other JTAG instructions are not affected, for example, you can still perform debugging.

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: Efinix 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 34). 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 *n*232H 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.

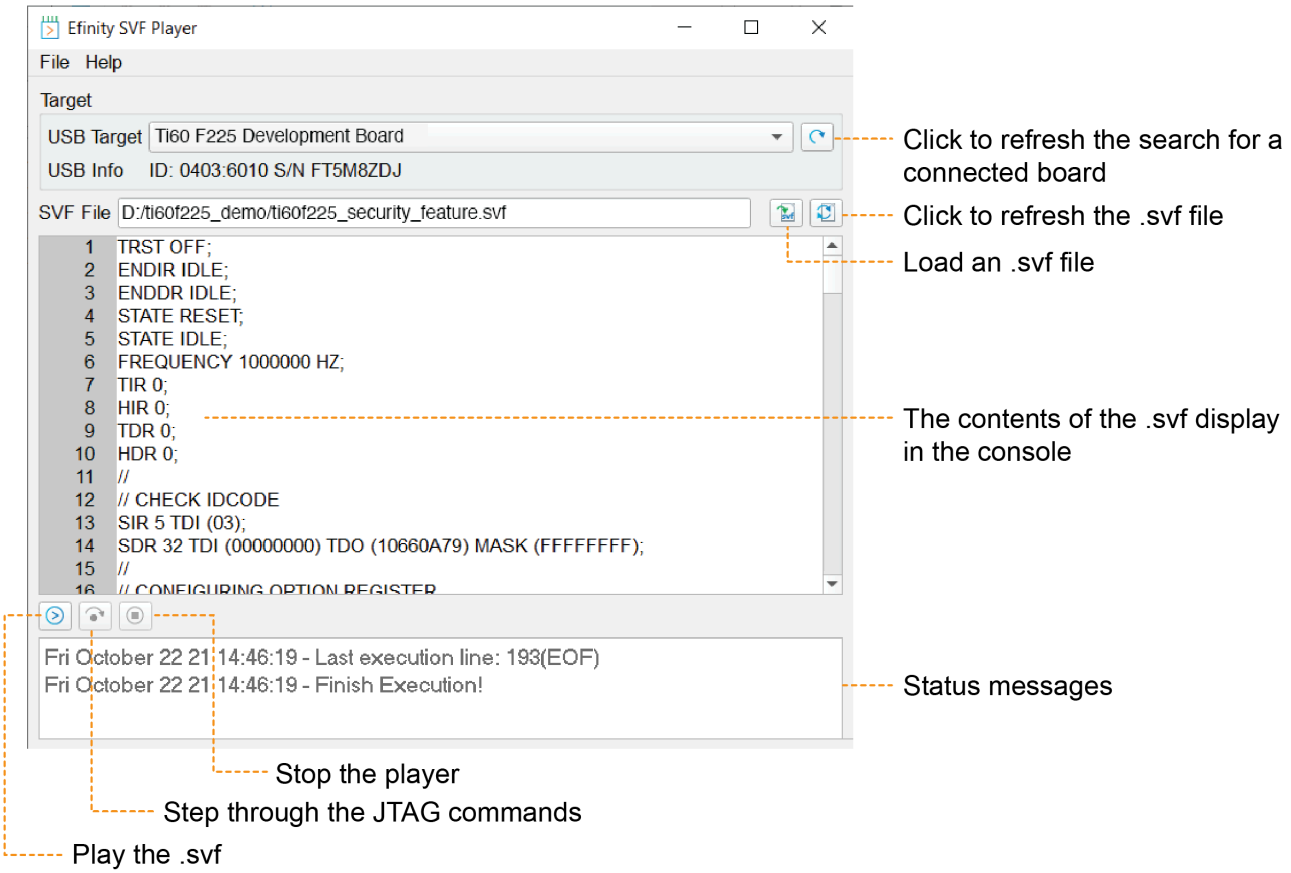
The **.svf** used for blowing fuses performs a variety of JTAG commands.

- 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 that you can use to encrypt and/or sign bitstreams from the command line. You use the script *<Efinity directory>*/**security/bin/AddSecurityTitanium.py**.

```
AddSecurityTitanium.py [-h] [-s] [-e] [-i IV] [-o OUTPUT]
                       [--device_version {1,2}] [--verbose]
                       [--timeout TIMEOUT] [-p KEYPAIR] [-x PASSPHRASE]
                       [--public_key PUBLIC_KEY]
                       bitstream keyfile
```

Table 10: 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.
--device_version	N/A	1, 2	Device security version. 1: Ti35, Ti60, Ti90, Ti120, Ti180, , , 2: , , Ti165, ,
--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.

The following example shows how to 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.

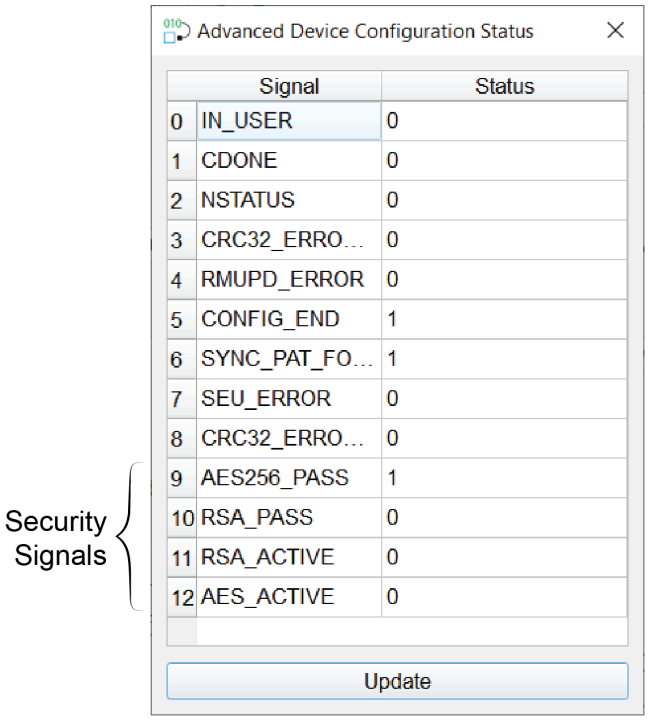
Verifying Security Settings

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 22 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



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.

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 *n*232H chipset.

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

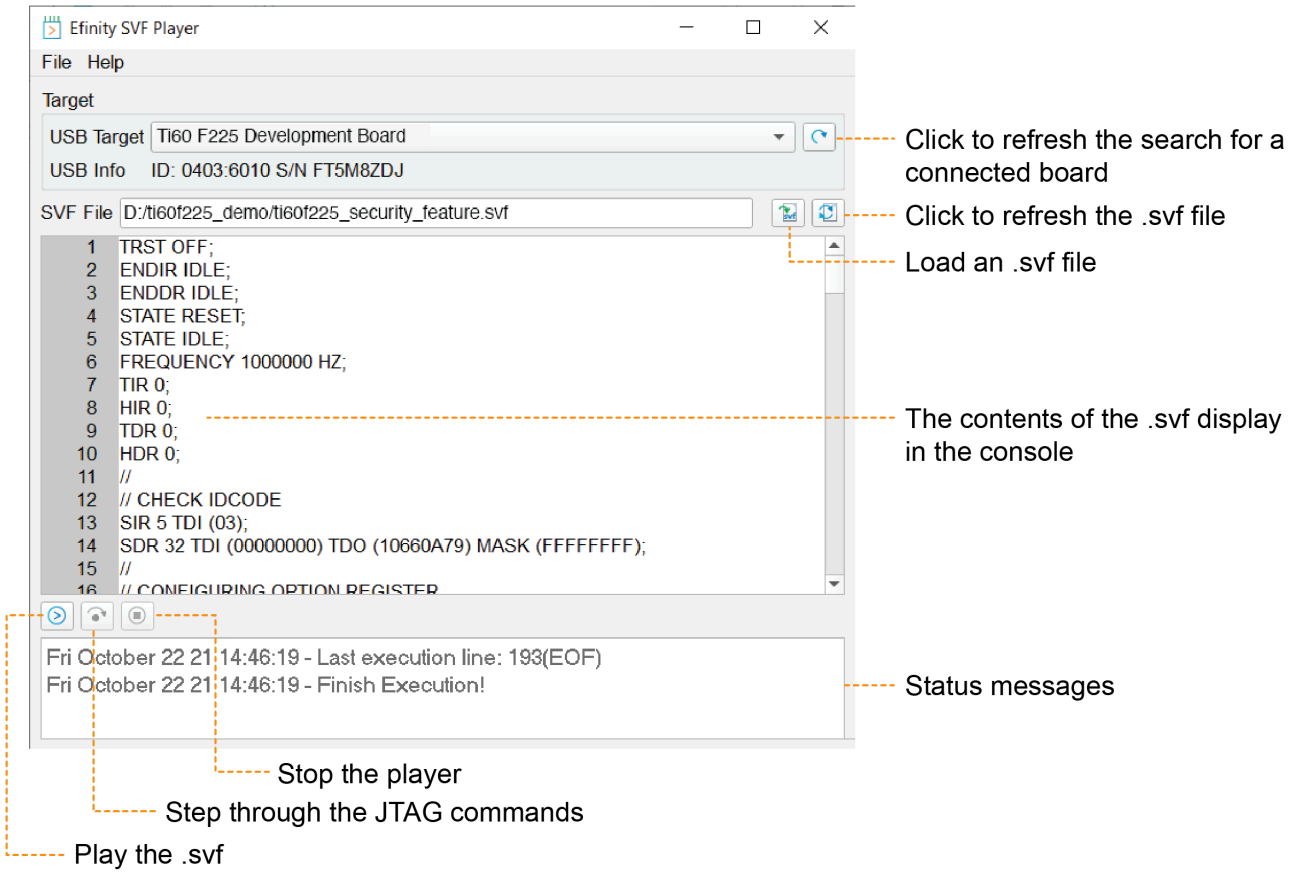
- Trion T20 in F324 and F400 packages
- Trion T35 in any package
- Trion T55, T85, and T120 in any package
- All Titanium FPGAs in any package

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.

Where to Learn More

The Efinity® 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 Synthesis User Guide
- Efinity Timing Closure User Guide
- Efinity Software Installation User Guide
- Efinity Trion Tutorial
- Efinity Debugger Tutorial
- Titanium Interfaces User Guide
- Trion Interfaces User Guide
- Efinity Interface Designer Python API
- Quantum® Trion Primitives User Guide
- Quantum® Titanium Primitives User Guide
- Quantum® 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®, Topaz, and Titanium FPGAs using the Efinity® 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®, 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 11: USB Programming Connections

Board	Connect to Computer with
Efinix development boards	USB cable
Your own board	FTDI *x*232H programming kit. For example: • FT2232H Mini Module • FT4232H Mini Module

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 USB Driver

On Windows, you use software from Zadig to install drivers. Download the Zadig software (version 2.7 or later) from zadig.akeo.ie. (You do not need to install it; simply run the downloaded executable.)

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.

To install the driver:

1. Connect the board to your computer with the appropriate cable and power it up.

2. Run the Zadig software.

   Note: To ensure that the USB driver is persistent across user sessions, run the Zadig software as administrator.

3. Choose **Options > List All Devices**.
4. Repeat the following steps for each interface. The interface names end with *(Interface N)*, where *N* is the channel number.
   - Select **libusb-win32** in the **Driver** drop-down list.
   - Click **Replace Driver**.
5. Close the Zadig software.

Note: This section describes how to install the libusb-win32 driver for each interface separately. If you have previously installed a composite driver or installed using libusbK drivers, you do not need to update or reinstall the driver. They should continue to work correctly.

Revision History

Table 12: Revision History

Date	Version	Description
November 2024	3.5	Fixed various typos.
November 2024	3.4	Updated entry for `SYNC_PAT_FOUND` in Table 7: Configuration Status Register on page 22. (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 **.bit** files for JTAG Bridge (New) and JTAG Bridge (Legacy), and they are not compatible with each other. The **.bit** files do not require an external clock. (DOC-1789)
May 2024	3.2	Added Ti165 and Ti240 FPGAs, replacing the Ti135 and Ti200, respectively.
January 2024	3.1	Added JTAG device IDs for Ti135, Ti200, and Ti375. (DOC-1662) Added Ti135 and Ti200 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)
May 2023	2.8	Added note about referring to AN050: Managing Windows Drivers in the Installing USB Drivers topic. (DOC-977) Removed Verifying Configuration with Programmer topic. The topic is in AN006 and AN033. Added IS25LP128 to list of supported flash devices. (DOC-1247)

Date	Version	Description
November 2022	2.7	Updated supported flash devices. (DOC-896) Corrected faint/missing callout lines in Figure 2 and 3. (DOC-976)
August 2022	2.6	Added installation instructions. Added instructions on using the JTAG SVF Player. Added topics on the Bitstream Security Key Generator. Clarified that when using internal reconfiguration you must use **Programmer** > **Combine Multiple Image Files** > **Image Type** > **Internal Flash Image** option. (DOC-874) Added topic on verifying configuration with the Programmer. When editing the bitstream header, do not remove any auto-generated data or the Programmer may not recognize the bitstream. (DOC-868) Removed support for C232HM-DDHSL-0 cable. (DOC-860) Updated Installing USB Drivers topics.
April 2022	2.5	Added Program using a JTAG Bridge topic. Added topic on combining a bitstream and other data into a single file for programming. Re-organized topics about working with bitstreams.
December 2021	2.4	Updated machine memory requirements (RAM). Added information about connecting to remote hosts. Added Macronix MX75L and MX75U to supported flash devices. (DOC-573) Added support for FTDI FT4232H Mini Module. (DOC-597) With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG. (DOC-638)
October 2021	2.3	Added topic on the Titanium configuration status registers. (DOC-487) Added topic on flash programming modes. Added note about FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495) Added XT25F family to list of supported flash devices. (DOC-529)
June 2021	2.2	Updated Windows driver installation instructions. Added SPI Active x8 over JTAG Bridge mode. Added Titanium JTAG IDs. Updated list of supported flash devices.
January 2021	2.1	Corrected JTAG chain file code example. (DOC-368)
December 2020	2.0	Added the requirement to install the Microsoft Visual C++ 2015 x64 and x86 runtime libraries for the standalone Programmer. Added JTAG device IDs for T20BGA324 and T20BGA400. Added FTDI cable and module connection for T20BGA400. Removed the FTDI2232 from About USB Drivers topic making the description applicable to other FTDI chips. Updated instructions on installing USB drivers for Windows. Added list of supported flash devices. Corrected JTAG Mini Module pin names for T4, T8, T13, T20BGA256, and T20BGA169 connection setup.
June 2020	1.0	Initial release.