Efinity® Software User Guide

UG-EFN-SOFTWARE-v15.0
November 2024
www.efinixinc.com

Contents

Figures

Tables

Introduction

The Efinity® software provides a complete tool flow for designing with Efinix® FPGAs and cores. The graphical user interface (GUI) provides a visual way for you to set up projects, run the software flow, view floorplan information, and build the interfaces that surround the logic portion of your design. You use the command-line to perform simulation and automate the flow using scripts.

Figure 1: Design Flow Overview
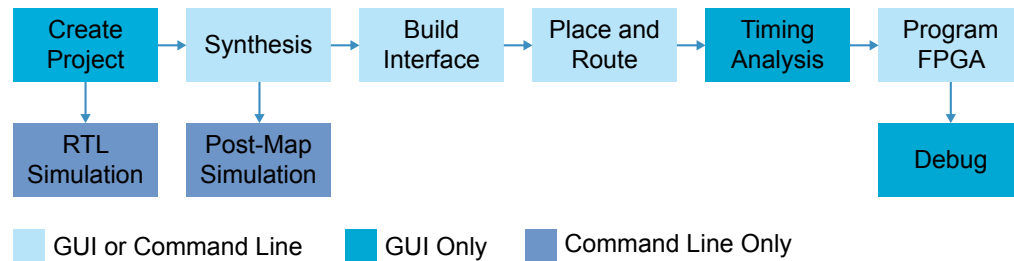
Table 1: Titanium FPGAs Supported in Efinity® Software v2024.2[1]

FPGA	Package	Bitstream	Timing	Pinout
Ti35	F100S3F2, F225	✓	Final	Final
	F100	✓	Final	Final
	F256	✓	Final	Final
Ti60	W64, F100S3F2	✓	Final	Final
	F100	✓	Final	Final
	F225	✓	Final	Final
	F256	✓	Final	Final
Ti85	N676	–	Preliminary	Preliminary
Ti90	J361, G400, J484, L484, G529	✓	Final	Final
Ti120	J361, G400, J484, L484, G529	✓	Final	Final
Ti135	N676	–	Preliminary	Preliminary
Ti165	C529, N1156, N484	–	Preliminary	Preliminary
Ti180	J361, G400, J484, L484, M484, G529	✓	Final	Final
Ti240	C529, N1156, N484	–	Preliminary	Preliminary
Ti375	C529, N1156	✓	Preliminary	Final
	N484	–	Preliminary	Preliminary

---

[1] Refer to the release notes on the web site for the latest support. Software patches often enable new device support.

Table 2: Topaz FPGAs Supported in Efinity® Software v2024.2[2]

FPGA	Package	Bitstream	Timing	Pinout
Tz50	F100, F225, F256	✓	Preliminary	Final
Tz110, Tz170	J361	-	Preliminary	Final
	G400, J484	-	Preliminary	Final

Table 3: Trion FPGAs Supported in Efinity® Software v2024.2

FPGA	Package	Bitstream	Timing	Pinout
T4	F49, F81	✓	Final	Final
T8	F49, F81, Q144	✓	Final	Final
T13	Q100F3, F169, F256	✓	Final	Final
T20	W80, Q100F3, F169, Q144, F256, F324, F400	✓	Final	Final
T35	F324, F400	✓	Final	Final
	F256	-	Final	Preliminary
T55, T85, T120	F324, F484, F576	✓	Final	Final

New in v2024.2

The Efinity® software v2024.2 has the following new features and enhancements:

- Support for Topaz FPGAs
  — Tz50 in F100, F225, and F256 packages
  — Tz110 in J361, G400, and J484 packages
  — Tz170 in J361, G400, and J484 packages
- Improvements to PMA Direct mode support
- 40% reduction in peak memory use for Titanium FPGAs
- Unified design feature
  — Simulation support for basic interface blocks
  — Synthesis inferencing and instantiation support for basic functions
  — Integration with Efinity GUI
- Improved SDC support
  — All functions support Tcl objects
  — New commands `check_timing` and `report_cdc` to help users improve SDC quality

---

[2] Refer to the release notes on the web site for the latest support. Software patches often enable new device support.

Using an Existing Project with a New Software Version

If you are upgrading from an older Efinity version, previously generated compilation files (such as old synthesis and place and route output files) may not be compatible with the new version of Efinity software. If the old files are not compatible, the software will prompt you to re-compile.

Important: When you open an existing project in a newer software version the Efinity software updates the project files with version-specific modeling information; the project files are not backwards compatible. Therefore, Efinix recommends that you make a backup of your project if you may want to open it again in an previous software version.

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.

Hardware and Software Requirements

General Requirements

- Efinity full release: 64-bit, x86 instruction set architecture
- Windows Standalone Programmer.
  — Windows 10: 64-bit x86 instruction set architecture.
  — Windows 11: 64-bit x86 instruction set architecture.
- Computer with a 64-bit operating system.
  — A 64-bit Windows system is required for the Efinity standalone programmer.
  — A 64-bit Windows system is required for using the security tools in the Efinity standalone programmer.
- Your preferred text editor such as Notepad, gVim, Visual Studio
- Machine memory requirements (when running Efinity design compilations):

Table 4: Machine Memory Requirements

Product	Model	Memory
Trion	T4, T8, T13, T20, T35	8 GB
	T55, T85, T120	16 GB
Titanium	Ti35, Ti60	8 GB
	Ti85, Ti90, Ti120, Ti135, Ti180, Ti165, Ti240, Ti375	16 GB
Topaz	Tz50	8 GB
	Tz110, Tz170	16 GB

Windows Requirements

- Windows 10 or later, 64 bit operating system
- 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

  > **Note:** If you want to use the stand-alone Programmer, you also need to install the x86 and x64 libraries (for 64-bit systems).

- Zadig software to install USB drivers
  see **Installing the Windows USB Driver** on page 136
- Java 64-bit runtime environment; required for configuring the Sapphire RISC-V SoC and DMA Controller in the IP Manager; available from:
  — https://www.java.com/en/download/manual.jsp (Java 8)
  — https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)
  — http://jdk.java.net/16/ (OpenJDK 16)

  > **Note:** You may also use other Java software platforms that are available in the market.

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

Linux Requirements

Table 5: Linux Operating System

Operating System	Note
Ubuntu v18.04 or later	You may need to install some graphics related libraries before running the Efinity tools. An example list of libraries (not comprehensive) could be: **apt install libxcb-xinerama0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0** Java 64-bit runtime environment (8 or higher), required for configuring some IP cores in the IP Manager (e.g., Sapphire SoC). Follow the instructions on the **Ubuntu web site** to install it. Your path environment variable should include the Java executable.

Red Hat Enterprise v8.0 or later	You may need to install some graphics related libraries before running the Efinity tools. An example list of libraries (not comprehensive) could be: **yum install libxkbcommon-x11 xcb-util-renderutil xcb-util-keysyms xcb-util-image xcb-util-wm xcb-util**
	
	 ⚠ **Important:** If the GUI is not working, set an environment variable, then launch the tool again using the following commands (applicable for all Linux operating systems):
	
	```source/path/to/efinity/<version>/bin/setup.sh```
	```export QT_DEBUG_PLUGINS=1```
	```efinity```
	
	 The Qt verbose command line output can provide clues about which system libraries are missing and need to be installled via apt/yum. Java 64-bit runtime environment (8 or higher), required for configuring some IP cores in the IP Manager (e.g., Sapphire SoC). Follow the instructions on the **Red Hat web site** to install it. Your path environment variable should include the Java executable.

- Linux X11 windowing system (for Efinity® GUI)
- Udev device manager for Efinix USB programming cable
  see **Installing the Linux USB Driver** on page 135

Installing iVerilog

Icarus Verilog (iVerilog) is a free Verilog simulation tool you can use to compile and simulate Verilog HDL source code. The software is available as source code or as pre-compiled binaries.

Windows installation:

To download the simulator: **bleyer.org/icarus**

ⓘ **Note:** The latest versions of iVerilog are bundled with the GTKWave software, so you only need to download 1 file to get both tools. Refer to the **bleyer.org/icarus** website for more information.

To download the simulator source code: **github.com/steveicarus/iverilog**

Linux installation:

Refer to the Installation Guide for steps to obtain, compile and install Icarus Verilog: **steveicarus.github.io/iverilog/**

ⓘ **Note:** Efinix recommends iVerilog version 11.0 or later.

Installing GTKWave

GTKWave is an open-source tool that analyzes post-simulation dumpfiles and displays the results in a graphical interface. It includes a waveform viewer and RTL source code navigator. You can use GTKWave with the iVerilog simulator to analyze and debug your simulation model, or to view any VCD waveform.

Windows installation:

You can read more at **gtkwave.sourceforge.net**.

Note: If you have downloaded and installed the iverilog setup file (bundled with GTKWave), you do not need to install a separate standalone GTKWave.

To download and run the latest Windows version, follow these steps:

1. You can browse for the software files at **gtkwave - Browse Files at Sourceforge.net**. The Windows files are situated lower down the page.
2. Unzip the downloaded file.
3. *Optional:*

You may need to add the path to GTKWave (`$GTKWave_folder$\bin\`) to your System Variables path for the software to launch correctly.

4. Run the program by executing **gtkwave.exe** in the `<install dir>/bin` directory.

Linux installation:

Linux users can use the following commands:

```
sudo apt-get update
sudo apt-get install gtkwave
```

Chapter 1

Setting Up

Contents:

Efinity Quick Start

To launch the Efinity graphical user interface (GUI), double-click the Efinity desktop icon. To launch and use the Efinity tool from the command line, refer to the following sections.

⚠ **Warning:** Do not use non-English characters in the Efinity project paths.

Windows

Set up your environment and PATH:

```
bin\setup.bat
```

Launch the Efinity GUI from the command line:

```
bin\setup.bat --run
```

Run Efinity from the command line:

```
cd %EFINITY_HOME%\project\<project name>  // Change to project directory
efx_run.bat <project name>.xml            // Run Efinity
```

For command-line help:

```
efx_run.bat --help
```

Linux

Set up your environment and PATH:

```
source bin/setup.sh
```

Launch the Efinity GUI from the command line:

```
efinity
```

Run Efinity from the command line:

```
cd $EFINITY_HOME/project/<project name>  // Change to project directory
efx_run.py <project name>.xml            // Run Efinity
```

For command-line help:

```
efx_run.py --help
```

Setting General Tool Preferences

Before you create a project, set general tool preferences to control the operation of the software.

Figure 2: General Tool Preferences

Indicate which external text editor you want to use

Turn on to use your specified editor when opening text files

Indicate the default root path for projects

When enabled, the software checks whether your software is newer than the project you open

When enabled, the software uses the mime type to open files in the default system application

When enabled, the software checks for updated IP cores

Note: The external text editor defaults to gedit in Linux and Notepad in Windows. The Efinity software also has a built-in Code Editor, which is best used for viewing code instead of as a full editor.

When you double-click a file in the **Project** tab or **Result** tab, the software opens the file in the Code Editor (default) or your specified editor (if you turn on the **Use user editor as default editor for all files** option).

Auto-Load Place-and-Route Data

When you open a project, the Efinity® software loads the messages and data from previous compilations and displays messages in the Console. For large designs, loading the place and route data can take time, and you might want to speed things up a little. The **Auto-load Place and Route Data** option lets you disable (or enable) loading of this data when you open a project.

If you have disabled auto-loading, you cannot view place-and-route results in the Floorplan Editor or Timing Browser, or use the Tcl console. To enable these tools, click the **Load Place and Route Data** button in the main window.

You enable and disable auto-loading in the Efinity® Preferences dialog box (**File > Preferences**).

Efinity Main Window

Use the controls in the Dashboard to run the tool flow, including synthesis, placement, routing, and bitstream generation.

Figure 3: Efinity Main Window

Chapter 2
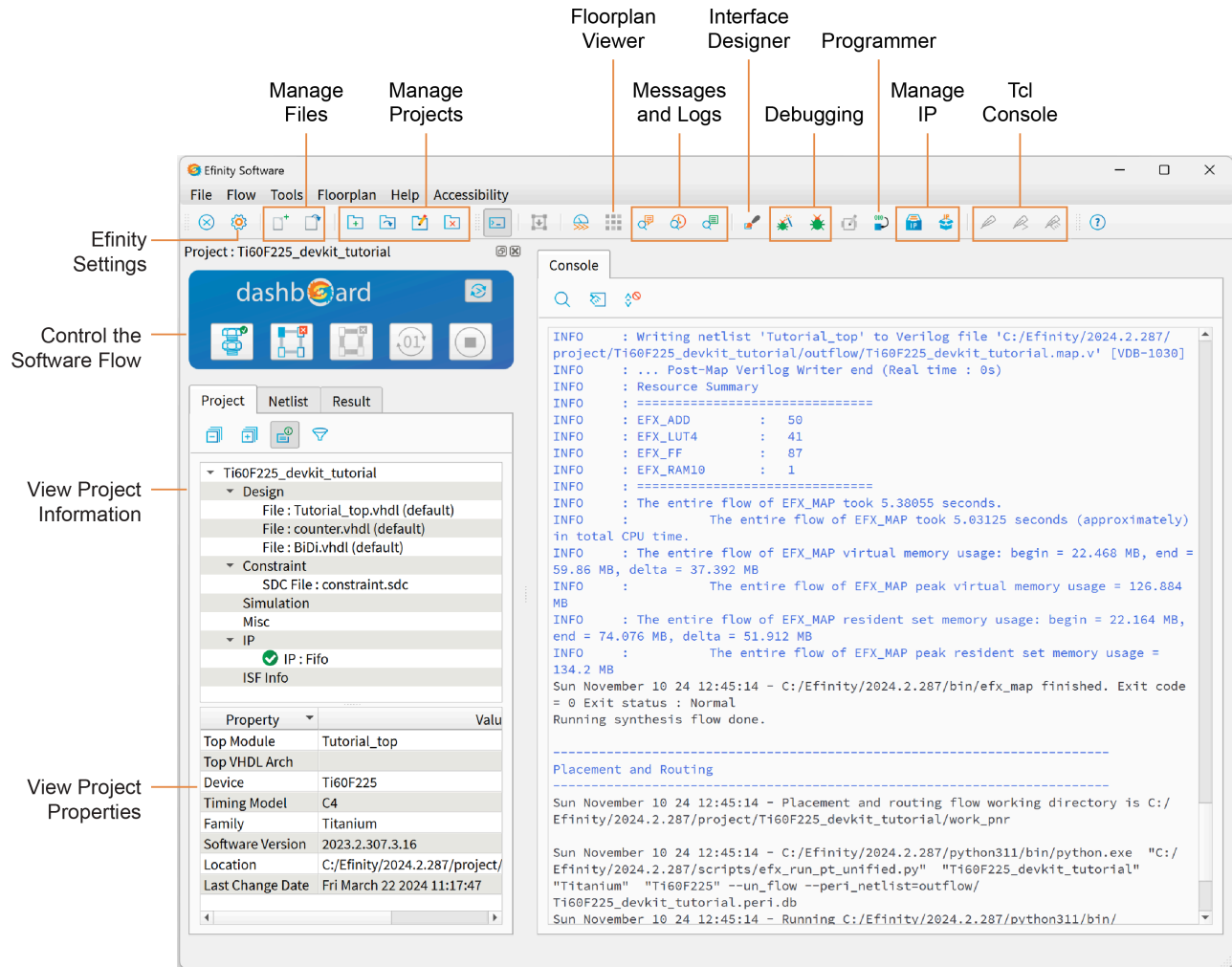
Managing Projects

Contents:

- **Project Editor**
- **Project Tab**
- **Referencing RTL Source Files**
- **Using VHDL Libraries**
- **Packaging Design Files**
- **Migrating a Project to another FPGA**

Project Editor

You use the Project Editor to create or modify a project, add files to your project (such as timing constraint files), and choose a device family and device. To create a new project, choose **File** > **Create Project** or click the Create Project icon.

Project Tab

Use the Project tab to specify the project name, location, optional project description, device family, device, and timing model. The project location defaults to *<install directory>/* **project/***<project name>*.

Figure 4: Project Editor - Project Tab
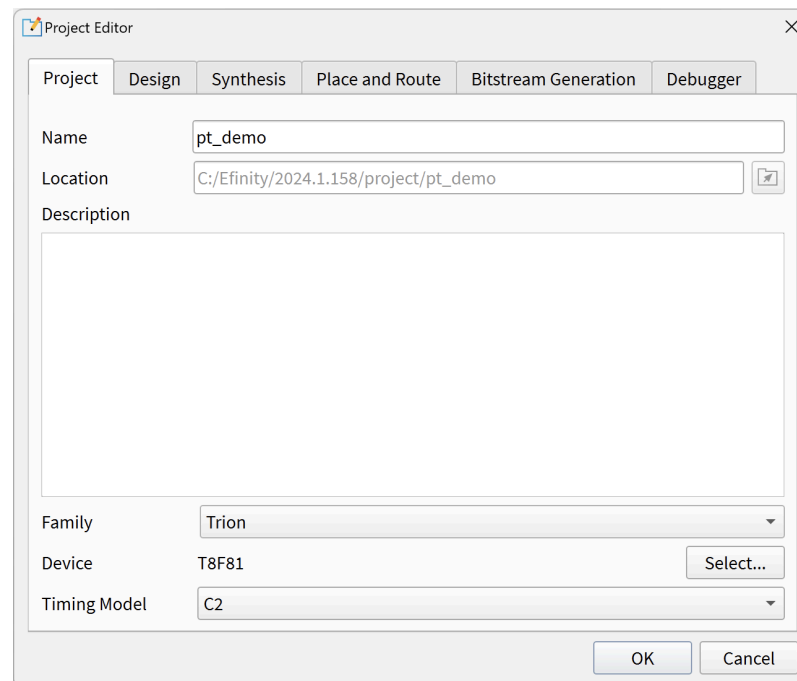
Design Tab

Use the Design tab to add design and constraint files to your project. In Efinity® software 2024.2 and higher, you **must** specify a name for the **Top Module/Entity**. For new projects, the **Top Module/Entity** name defaults to the project name. You can edit the **Top Module/Entity** name without changing the project name.

For existing projects, changing the project name does not automatically update the **Top Module/Entity** name. Therefore, if you want to change the top module or entity name, you must edit it in **Design tab > Top Module/Entity**.

Important: The **Top Module/Entity** name cannot be left blank. If you do not enter a name or if you delete the name, the software issues an error message when you try to save the change.

Figure 5: Project Editor - Design Tab
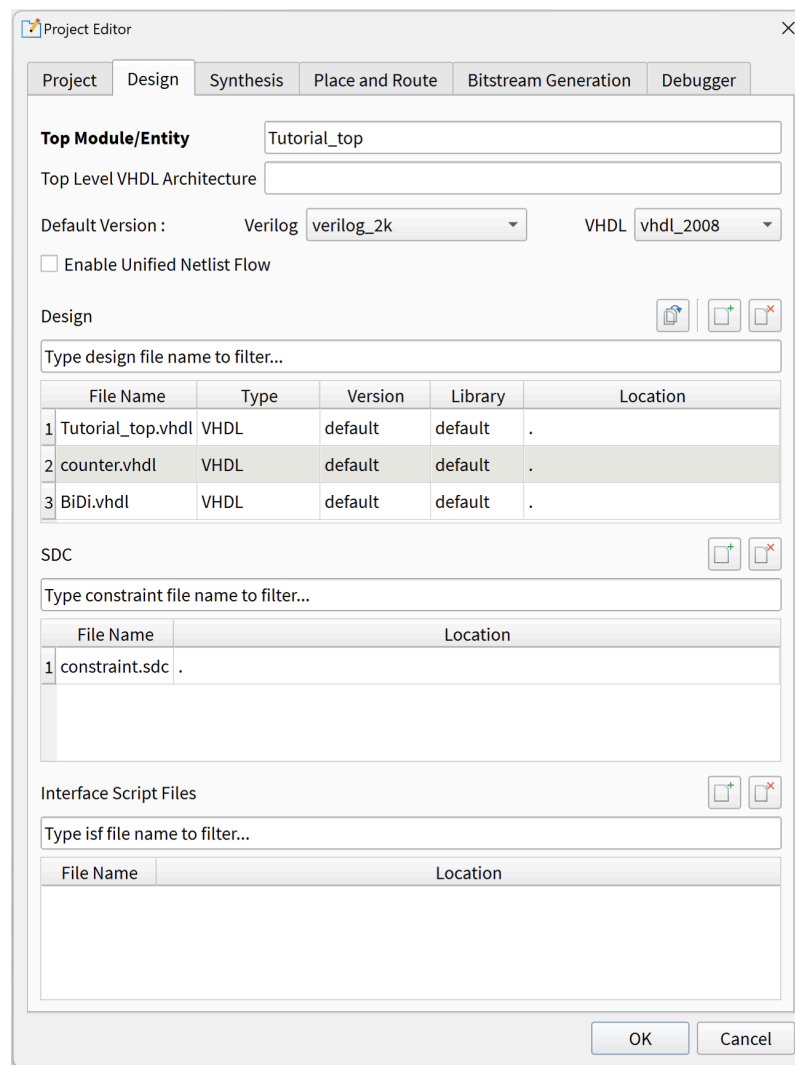
You can choose the top-level VHDL architecture, if desired.

You can set the default Verilog HDL, SystemVerilog, or VHDL version for the design files.

In Efinity® v2020.2 and higher, you can define VHDL libraries and add files to them. See **Using VHDL Libraries** on page 25 for details.

Learn more: For more information about language support, refer to the Efinity Synthesis User Guide.

You can import an entire directory of files into your project or add them one at a time. When you import files:

- Turn on **Copy to Project** to copy the imported files to your project directory. You can choose whether to flatten (copy all files to project root directory) or preserve the directory structure.
- If you do not copy the files to your project, specify whether to reference the files as full or relative paths.
- Choose to import only design files or all files (which includes constraints).

Optionally, you can specify a Synopsys Design Constraints (**.sdc**) file for timing-driven compilation.

Learn more: Refer to **Analyzing Timing** on page 68 and the **Efinity Timing Closure User Guide** for more information on timing analysis.

Synthesis Tab

Optional. The Efinity® software supports options to help you direct the synthesis flow. Use the options on this tab to specify project-specific preferences. If you do not make any settings, the tool uses the defaults.

Figure 6: Project Editor - Synthesis Tab
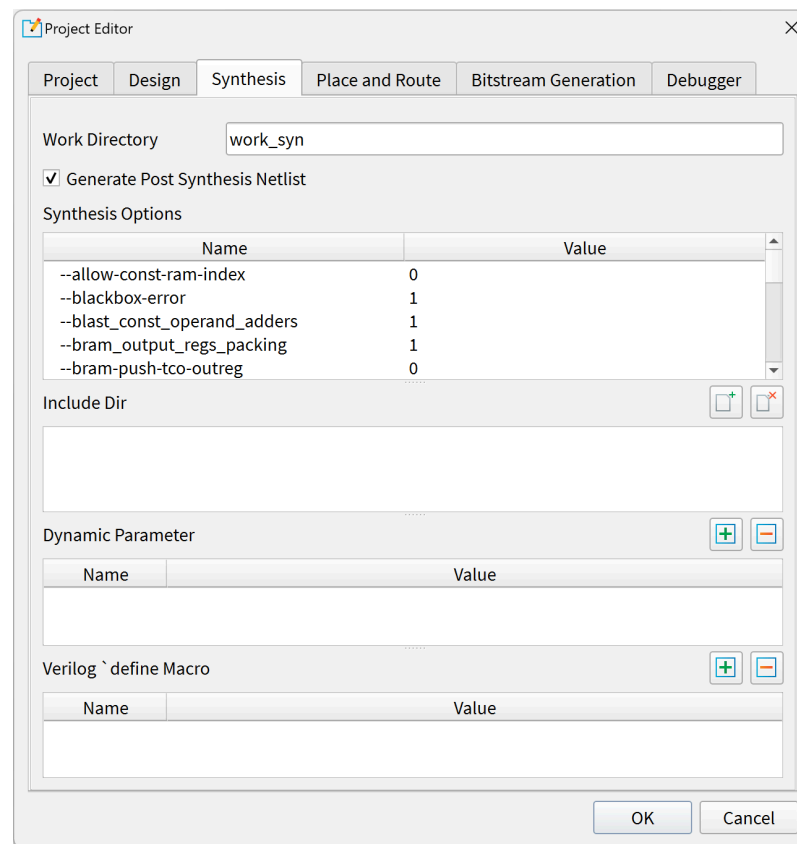
Table 6: Synthesis Project Settings

Setting	Description
Work Directory	Specify a custom directory or use the default (**work_syn**).
Generate post synthesis netlist	Choose whether the software should create this netlist. Default: On

Setting	Description
Synthesis Options	See "Synthesis Options" in the **Efinity Synthesis User Guide**.
Include Dir	Specify directories to include in your project. If you use the IP Manager to add IP, the **ip/**<*module*> directory is listed here. The software searches these locations when you use include statements.
Dynamic Parameter	Use this area to add parameters and values that apply to the top-level module or entity in your project. The value passed into the Dynamic Parameter field must be the same format as that you would use for any variable in VHDL or Verilog HDL. For example, string should be in quotation marks.
Verilog `define Macro	Use this area to add `define macros to your project.    Some FPGA EDA tools automatically create a SYNTHESIS macro. If you want to use the same behavior in the Efinity software, you need to create it here. For example, click the Add Verilog `define Macro button and then enter `SYNTHESIS` in the **NAME** field and `1` in the **Value** field. Then if you want to include simulation only code, use this format: ``` `ifndef SYNTHESIS $display(...) ... some other simulation directives ... `endif ``` You can also use the translate_on and translate_off directives to accomplish similar functionality.

Learn more: Refer to the **Efinity Synthesis User Guide** for more information on these options.

Place and Route Tab

Optional. The options on this tab let you specify project-specific preferences to help close timing. If you do not make any settings, the tool uses the defaults.

- The optimization levels (`--optimization_level`) are settings that control both placement and routing, targeting different metrics.
  - CONGESTION optimization levels may help a congested design meet timing.
  - TIMING optimization levels may help a non-congested design meet timing requirements.
  - POWER optimization levels may help reduce a design's power consumption.

  Often, the default settings are the best choice, as these options will not help all designs.
- The placer effort level (`--placer_effort_level`) is a way to control how much runtime the placer uses when it tries to improve placement quality.
- The number of threads (`--max-threads`) controls how many thread that the placer can launch. The default setting (-1) means that the placer uses the maximum number of available processors.

- The `--seed` option introduces random noise in the placer. The seed is the value you set.

Figure 7: Project Editor - Place and Route Tab
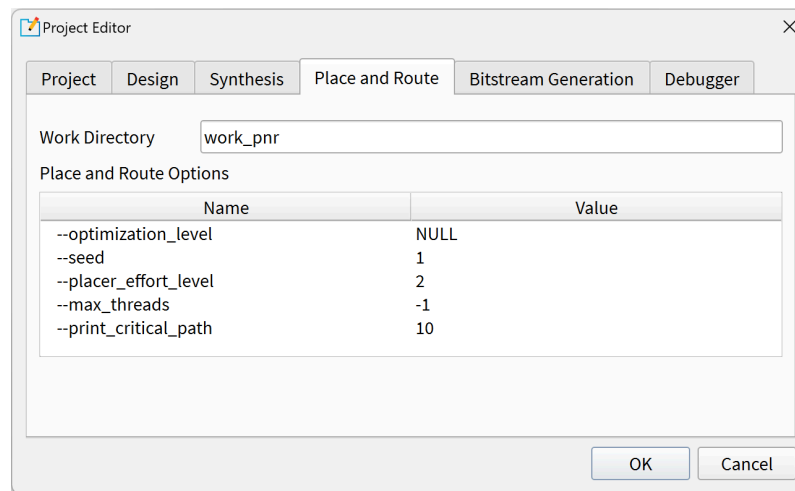
Learn more: Refer to "Place and Route Options" in the **Efinity Timing Closure User Guide** for more details on these options and how to optimize timing.

Bitstream Generation Tab

Optional. Use the options on this tab to specify project-specific preferences such as the programming mode, daisy chaining, and memory initialization. If you do not make any settings, the tool defaults to SPI active programming mode.

Figure 8: Project Editor - Bitstream Generation Tab
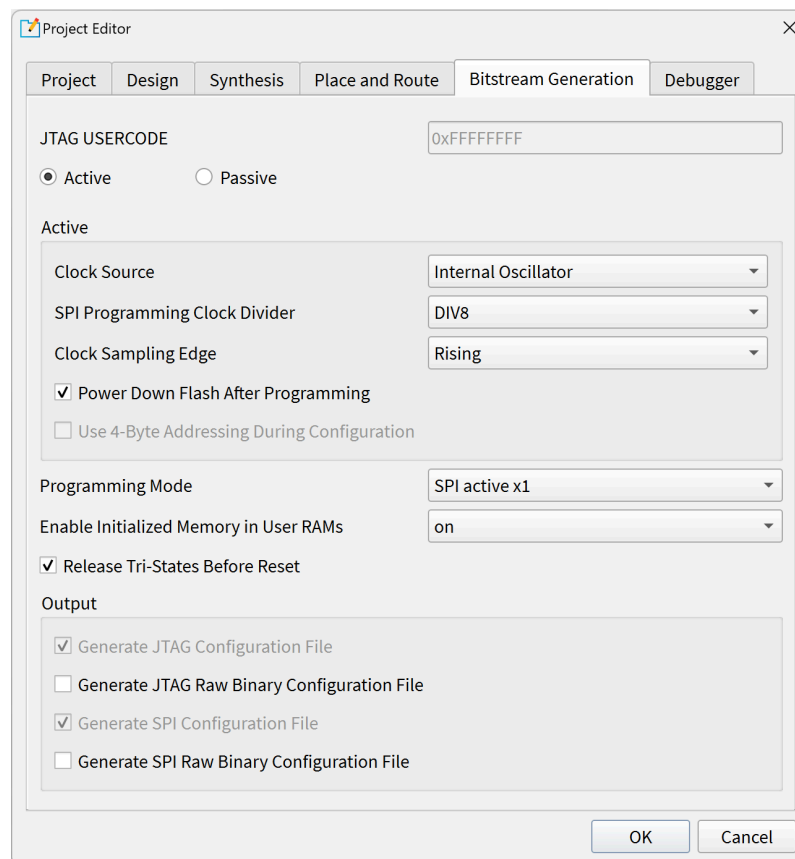
Learn more: Refer to **Project-Based Programming Options** on page 115 for more information on bitstream and programming settings.

Debugger Tab

Optional. This tab is where you enable or disable a debug profile to auto-instantiate in your design and set a working directory for debugging. These settings are used with the Debug Wizard.

Figure 9: Project Editor - Debugger Tab
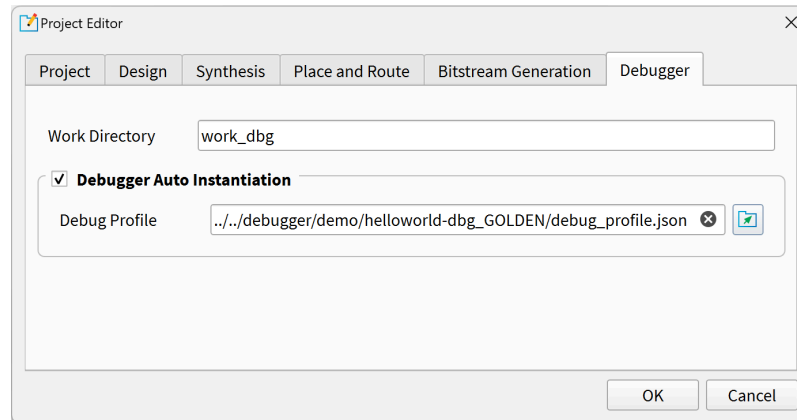
Learn more: Refer to **Debugging** on page 76 for more information.

Project Tab

The Project tab, which is located under the Dashboard, shows all of the files in your design. Double-click a filename in the Project tab to open the file in the Efinity® Code Editor. To open the file in another editor, right-click the file name and select your preferred text editor. You can also choose to show the file in it's containing folder.

Tip: You can resize the Project tab. Grab the blank space between the Project tab and the Console and drag to resize.

Figure 10: Project Tab
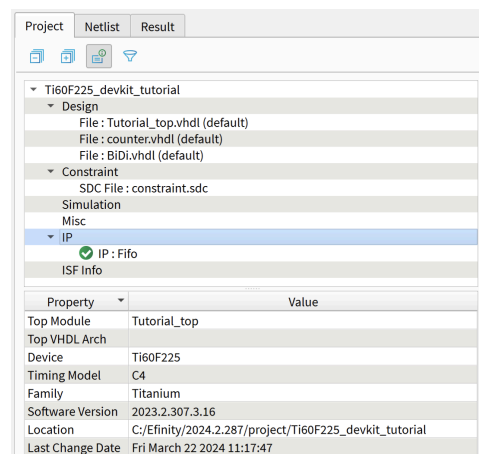
You can right-click the folders in the Project tab to open a context-sensitive menu with shortcut actions. For example, you can right-click **Constraint > Add** to browse for a new **.sdc** file and add it. Refer to **Managing IP in Your Project** on page 45 for more information on the IP context menus.

Referencing RTL Source Files

Instead of adding your RTL files to your project individually, you may want to reference them. The Efinity software v2024.1 and higher lets you reference your design's source files in a reference file list (**.f**). With this method, you can update the **.f** file once and have the changes reflect for all projects that include the **.f** file.

Save the **.f** file in the same directory as your RTL source files. Then add the **.f** file to your project.

The **.f** file is a text file. Add the source files one per line. Any directories should be relative to the location of the **.f** file.

```
module1.v
module2.v
top.v
```

Add directories you want to include, The Efinity software searches the include directory for files specified with the `include directive. The software can also find modules that are not specified in the source file list. For example, the software can find module m1 specified in m1.v if the file m1.v is located in a specified include directory.

```
+incdir+<include_directory>
```

You can also list another **.f** file:

```
-f module1.f
-f module2.f
top.v
```

You can specify the HDL version for the source file. For Verilog HDL files, the available versions are: verilog_95, verilog_2k, sv_05, and sv_09. For VHDL files, the available options are vhdl_1993, vhdl_2008, and vhdl_2019.

```
file1.v, t:verilog_2k
file2.vhd, t:vhdl_2019
```

To include a custom VHDL library, use this format:

```
file1.vhd, l:<custom library>
```

Using VHDL Libraries

In the Efinity® software v2020.2 and higher, you can use VHDL libraries to organize and reference commonly used packages and entities.

Create a Library

To create a library for your project:

1. Open the Project Editor.
2. Click the **Design** tab.
3. Add the design file(s) that have the packages you want to use. You can add multiple files.
4. Double click the cell under **Library**.
5. In the drop-down menu, choose **Add New**.
6. Enter the library name and click **OK**.

> **Note:** In VHDL, the **work** library refers to the current library in the design. When assigning a library name to a VHDL design file, you are encouraged not to use the word **work** as the library name only (instead use a variable like name, example: **my_work**). Doing so will cause an error in synthesis. Leave it blank (or default) if the file is part of the current library in the design project.

7. (Optional) If you add more than one library file to your project, double-click in the **Library** cell for each file and either choose the library name or add a new one.

Library names are saved across projects.

Add a File to a Library

You add a file to a library in the **Project Editor > Design** tab. Double-click the Library cell for the file and choose the name from the drop-down list.

Reference a Library

You use the `library` and `use` VHDL language constructs to reference your new library. The following simple code example shows a new library file for the package `mylibrary`:

Example: mylibrary.vhd

```
--! Use standard library
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package mylibrary is
 --! factor width
 constant DF_WIDTH : integer := 12;
end package mylibrary;
```

After you add this file to your project and create a library for it, you can refer to the file in your code:

Example: Referring to the Package

```
--! Use standard library
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--! Use costum library
library mylibrary;
use mylibrary.mylibrary.all;

--! Multiplier entity brief description
```

```
--! Detailed description of this
--! multiplier design element.
entity multiplier is
    port (
        a     : in  signed (DF_WIDTH-1 downto 0);      --! Multiplier first factor
        b     : in  signed (DF_WIDTH-1 downto 0);      --! Multiplier second factor
        result    : out signed (2*DF_WIDTH-1 downto 0)    --! Multiplier result
    );
end entity;
```

Reference Trion and Titanium Primitive Libraries

The Efinity® software includes VHDL libraries for Trion and Titanium primitives. You use the `library` and `use` VHDL language constructs to reference these libraries:

```
library efxphysicallib;
use efxphysicallib.efxcomponents.all;
```

📖 **Learn more:** The following documents provide example code for these libraries:

> **Quantum Titanium Primitives User Guide**
> **Quantum Trion Primitives User Guide**

Packaging Design Files

The IP Packager is a tool that lets you "package" design file(s) as standalone IP for use in the IP Manager. This feature is helpful for using the same code in multiple projects.

ⓘ **Note:** The IP Packager is available in the Efinity software v2024.1 and higher.

To launch the IP Packager, right-click **Project > Design** and choose **Package**.

Figure 11: Opening IP Packager
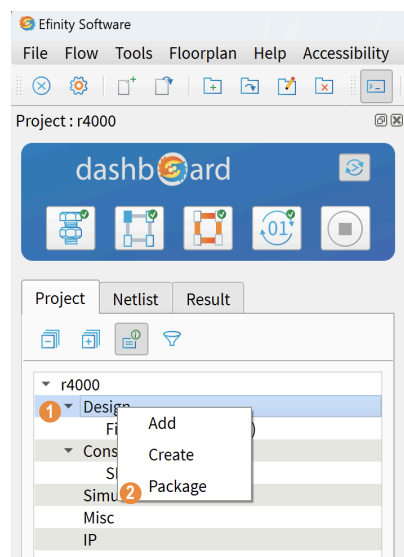
To open the IP Packager:

1. Right-click **Design**.
2. Choose **Package**.

📖 **Learn more:** For instructions on using IP Packager, refer to the IP Packager User Guide.

Migrating a Project to another FPGA

You choose an FPGA when you create your project. But later, you may want to migrate your project to a different FPGA. The new FPGA you select may not have the same features as the existing one, and some resources may not have the same names. So when you choose a new FPGA in the Project Editor, the Efinity® software automatically performs a design check in the background to help you migrate the design. You see a message at the bottom of the main window that the software is checking for migration issues.

Note: Trion and Titanium FPGAs have different configuration settings. Therefore, when migrating designs from the Trion family to the Titanium family, the software resets any incompatible configuration settings to the default.

The software generates a detailed log of the interface changes it makes during the migration and saves it into the *<project>*/**work_pt** directory.

Automatic Migration

If the new FPGA you choose is similar to the existing one (for example, you want to change from the T13 in the BGA256 to the T20 in the BGA256), the software can migrate all the assignments automatically and gives a message that migration completed successfully. You do not need to do anything else.

If the two FPGAs have different interface resources (GPIO, PLLs, etc.) but they are pin and package compatible, the software migrates the assignments automatically. The user design instances will be preserved but some resources may be automatically reassigned.

Migrate Design Wizard

If the software cannot migrate automatically, it launches the Migrate Design wizard. This wizard helps you decide how to handle the changes. In the first pane, the wizard:

* Shows the issues it found, for example, GPIO feature differences.
* Asks if you want to create a new interface design or update your current one.
* Lets you back up your existing interface design so you can go back to it if needed.

In the second pane, the wizard shows the assignments that have problems. If you decide to continue migration, the wizard opens the Interface Designer so you can fix the problems. You can also cancel to stop migration.

Note: If you cancel migration and keep the new FPGA setting, the Migrate Design wizard opens again the next time you run the Interface Designer.

Note: For help understanding the messages, refer to the "Design Check" topics in the Titanium Interfaces User Guide. These topics describe the messages the Interface Designer generates and gives suggestions on how to fix errors and warnings.

The outcome of design migration depends on the FPGAs involved because each FPGA has its own unique interface and resources, and each interface block supports specific features. Therefore, migrating the design from one FPGA to another is limited to the interface block support available in the destination FPGA.

Migrating a design from one family to another requires manual modification in the post-migrated design. Different families have different architectures, and therefore different features. However, the software tries to preserve the instances that have already been created if the interface block is supported in both FPGAs despite having a different feature set. In this case, some of the configuration settings may be reset to the default in the migrated design.

The possible outcomes for instances when migrating a design are:

- The instance is retained but resource assignments are removed.
  - In most cases, the interface block instance is preserved but the assigned resource is removed because the resource does not exist in the destination FPGA or you are migrating between families.
  - The instance is retained but the feature set is different. Refer to the migration log in the wizard to understand the differences. For example, when migrating from Trion FPGAs to Titanium FPGAs, the GPIO instances are preserved but different, additional, or incompatible features are set to the default.
- The instance is removed.
  - If the new FPGA does not support the interface block, the block is removed. For example, migrating a T20F324 design with LVDS RX instantiated to the T8F81 (which does not support LVDS RX) results in the LVDS RX instance being removed.
  - If the new FPGA has the same block but the block's features are completely different, the block is removed. For example, migrating a T120F576 design with DDR instantiated to the Ti180G529 (which supports DDR but with a completely different configuration) results in the DDR instance being removed.

The Device Setting stores information related to I/O banks and FPGA settings. In Titanium FPGAs, it also includes the Clock/Control configuration.

Device Settings are migrated as part of the design migration process. The outcome of the migration depends on the setting compatibility between the FPGAs. If the setting is not compatible, it is reset to the default value for the destination FPGA.

- I/O Bank configuration is migrated if the setting is valid or applicable in the destination FPGA. For example, if Bank 1A exists in the destination FPGA, the voltage indicated for this bank is migrated if the destination FPGA Bank 1A supports the voltage value. If the destination FPGA Bank 1A does not support the voltage, Bank 1A in the migrated design is reset to the default voltage.
- Clock/Control configuration does not exist in Trion FPGAs. Therefore, you cannot migrate any settings between Trion and Titanium FPGAs.

Running the Tool Flow

Contents:

- **Run the Flow with the Dashboard Controls**
- **Run the Flow from the Command Line**
- **About Efinity Synthesis**
- **Netlist Tab**
- **Netlist Viewer (Beta)**
- **Viewing Messages and Logs**
- **Result Tab**
- **Viewing Place-and-Route Results**

The Efinity software supports GUI and command line tool flows.

Run the Flow with the Dashboard Controls

The Dashboard controls the software flow, which operates in two modes: automated and manual. Toggle automated and manual flows using the toggle button.

> **Note:** The automated flow is on by default.

- **Automated flow**—Use automated mode to run the full flow from start to finish. Additionally, you can start the flow from any point and run it to the end. For example, after completing the full flow you can restart the automated flow at the placer stage and run the flow to the end.
- **Manual flow**—Disable the automated flow to run each stage manually.

When a stage completes, a marker indicates whether the stage completed successfully, with warnings, or with errors. You can stop and restart the flow at any point.

Figure 12: Dashboard Controls
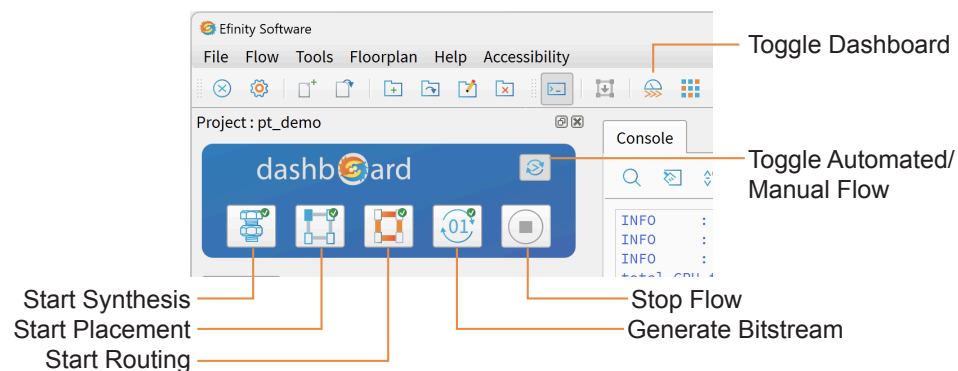
Run the Flow from the Command Line

You can run the software flow from the command line using the **efx_run.py** Python 3 script. This script is available in the **scripts** directory.

Compilation commands:

- `--flow compile` performs synthesis, place and route, and generates a bitstream hex file (default)
- `--flow map` performs synthesis
- `--flow pnr` performs place and route
- `--flow full` runs the full flow, including RTL and post-synthesis simulation
- `--flow interface` generates the interface constraint files
- `--flow sta_tclsh` enters the Tcl Console

Simulation only commands:

- `--flow rtlsim` performs RTL simulation on the design's source files
- `--flow mapsim` performs simulation with the post-synthesis netlist file

Programming commands:

- `--flow pgm` creates the bitstream hex file used to configure the device
- `--flow program` programs the target device

Tip: Use `--help` to display the command-line help and additional options.
The Console supports color output. See **Console** on page 35.

The following example command runs the complete flow on the **helloworld** design:

Example: Run Complete Flow from Command Line

Linux:

```
> efx_run.py helloworld.xml --flow full
```

Windows:

```
> efx_run.bat helloworld.xml --flow full
```

About Efinity® Synthesis

The first stage after you complete your RTL design is synthesis. During synthesis, the compiler takes your design and turns it into a gate-level netlist. The software supports synthesis options and attributes so you can optimize your design.

The software supports the synthesizable subset of the following languages:

- SystemVerilog and Verilog HDL
- VHDL
- Mixed languages (any combination of the above)

Learn more: Refer to the **Efinity Synthesis User Guide** for more information on synthesis options ans attributes as well as design guidelines.

Netlist Tab

The Netlist tab, which is under the Dashboard, shows the design hierarchy and helps you browse through the elaborated design and synthesized netlist. You can only view the synthesized netlist after you have performed synthesis. You can right-click the items in the Netlist tab to open a context-sensitive menu with shortcut actions.

Tip: You can resize the Netlist tab. Grab the blank space between the Netlist tab and the Console and drag to resize.

Figure 13: Using the Netlist Tab
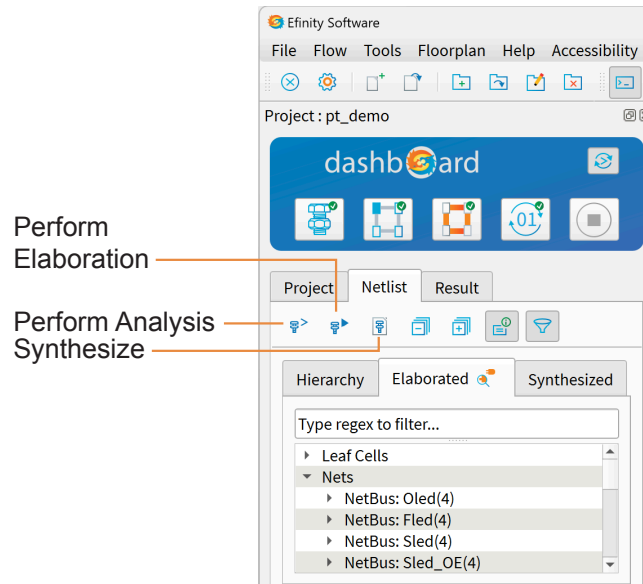
Netlist Viewer (Beta)

The Netlist Viewer tool displays and analyzes your design's netlist, including all components and their connections (nodes and nets). You use the Netlist Viewer to examine an elaborated netlist visually. (The Netlist Viewer does not support post-mapping netlists in the Efinity software v2023.1.)

! **Important:** The Netlist Viewer is beta in the Efinity software v2023.1.

With the Netlist Viewer you can:

- Visualize and analyze netlists in a graphical format
- Understand the connections between components
- Identify potential issues in your design

i **Note:** The Netlist Viewer is supported in the Efinity software v2023.1 and higher.

Figure 14: Netlist Viewer
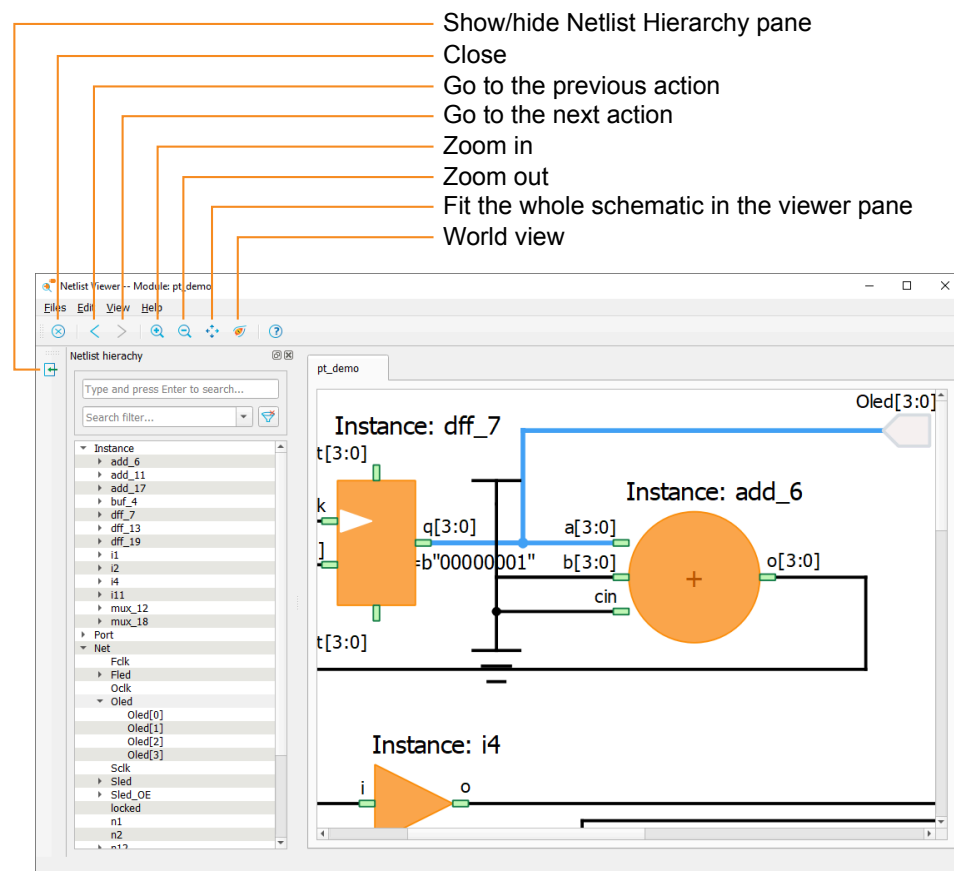
Opening the Netlist Viewer

1. If you have not already done so, synthesize your design or choose **Netlist > Elaborate All** to generate the elaborated netlist for your design.
2. Click **Netlist > Elaborated > Show Elaborated Netlist in Viewer** or choose **Tools > Show Elaborated Netlist in Viewer**.

> **(!) Important:** If the design is large and flattened, the Netlist Viewer will take a long time to load. This performance problem is a known issue for the beta.

Figure 15: Opening the Netlist Viewer

Click to open the Netlist Viewer
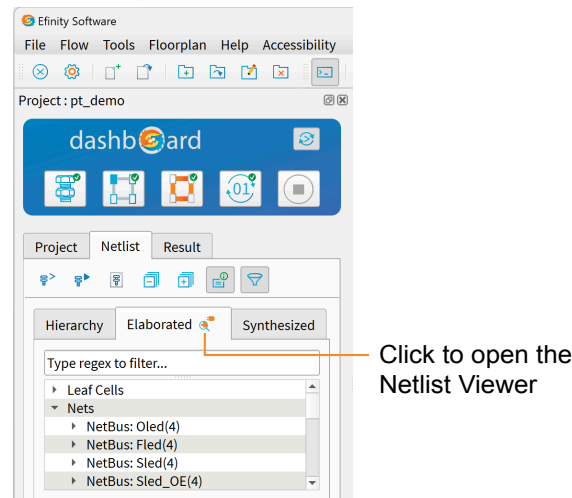
Zooming

There are several ways to zoom in or out:

- Use the toolbar buttons.
- Use the mouse to click and drag:
  — Drag down to the right to draw a box around the area you want to zoom in on.
  — Drag up to the right to zoom out or drag down to the left to zoom in. The zoom level is dependent on the length of the line you draw. A pop-up indicator shows the zoom level represented by the line length, e.g., zoom -1.0 or zoom +0.5.
  — Drag up and to the left to fit the whole design in the viewer (zoom fit).
- Right-click in the viewer. In the pop-up menu, choose **View > <option>** to zoom in, out, or fit.
- If you are zoomed in and want to see where you are, open the World View.
  — Choose **View > World View**.
  — Click the World View button.
  — Right-click in the viewer and choose **View > World View** from the pop-up menu.

> **(i) Note:** You can also highlight or mark design elements and then zoom to them. See **Highlighting and Marking** on page 33.

Highlighting and Marking

For large netlists, you may want to use a visual indicator or marker to keep track of where specific design elements are located in the viewer. The Netlist Viewer has two ways to put an indicator onto an element: highlighting and marking,

- Highlighting adds a colored line around the element.

- Marking adds a colored dot in the center of the element and on each port connected to the element.

> ℹ **Note:** The software does not save the highlights or marks; they are discarded when you close the Netlist Viewer. (This feature is planned for a future version.)

You can adjust the zoom to fit the highlighted or marked items in the view. Right-click in the viewer and choose **View** > **Fit highlighted** or **View** > **Fit marked**.

Viewing the Netlist Hierarchy

The Netlist hierarchy pane shows the instances, ports, and nets of the current module. You can browse through the list or use the filter to find specific elements.

You can toggle the Hierarchy pane's visibility by clicking the Show/Hide Netlist Hierarchy icon.

Finding Elements

For large netlists, it can be difficult to browse elements with the Netlist Hierarchy. Instead you can use the Find function to search for elements by name or with a regular expression. You can choose whether to search instances, ports, nets, or a combination.

1. Choose **Edit** > **Find** to open the Find dialog box.
2. Enter the search criteria.
3. Click **OK**.
4. The Search Result pane opens and displays the results. Click on an element name to see it in the viewer.

> ℹ **Note:** For buses, you need to click on the bus name not the signal name to see it in the viewer.

Figure 16: Finding Elements
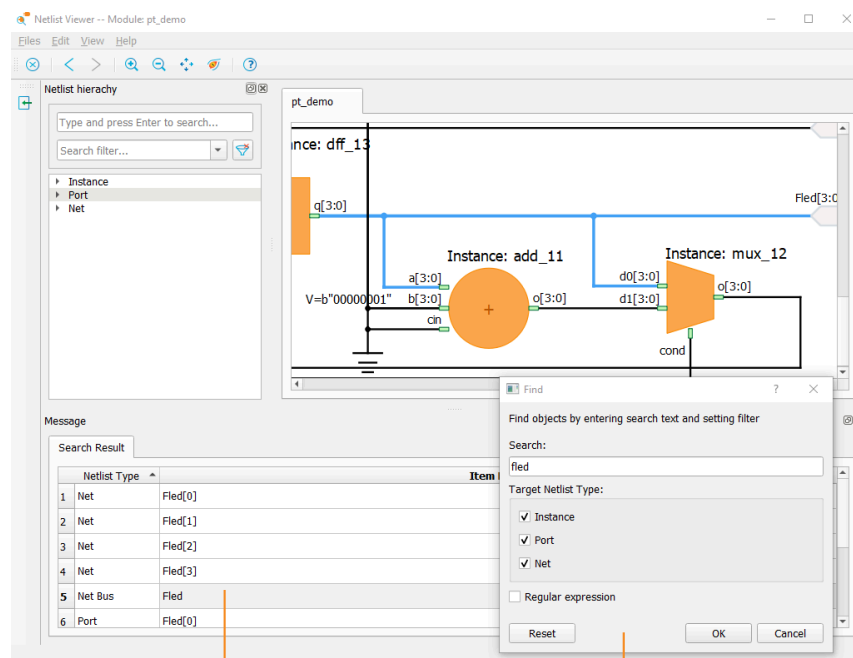
Click an element to see it in the viewer

Find dialog box

Viewing a User-Defined Element

The viewer colors manually instantiated primitives blue (instead of orange). Double-click on the element to view the internal structure of the primitive.

ⓘ **Note:** You cannot view the internal structure for encrypted elements. If you mouse over a blue element and the cursor changes to a hand, you can double-click to view it. If the cursor does not change, the element is encrypted.

Viewing an Element's Connectivity

You can see all of the nets that connect to a specific element.

1. Right-click the element.
2. Choose **Show Connectivity**. The viewer colors all of the nets connected to the element in blue.

Viewing the Action History

As you work in the Netlist Viewer, it saves a history of all your actions. Then, you can go backwards in the history to remove actions and forward in the history to perform them again. This feature can be useful to see a previous state when you are marking and highlighting elements.

Viewing Messages and Logs

The Efinity software has several methods for viewing messages and log entries that result from the compilation flow.

Console

The Console provides verbose messages and reports for all aspects of the tool flow. Additionally, it functions as the Tcl message console when you turn on the Tcl Command Console.

*   You can clear the Console and remove all messages.
*   You can prevent the Console from scrolling when the tool issues new messages.
*   You can enable/disable text and background color (Efinity software v2024.1 and higher).

The Console supports output in color. Info, warning, and error messages are highlighted in different colors (blue, purple, and red, respectively). To turn color on or off, use **Accessibility > Console Color**. You can also turn on a dark mode for the Console background using **Accessibility > Console Dark Theme**.

If you are running the Efinity software at the command line, the output is also in color by default. Use these environment variables to control the output color:
*   *Use color*—`EFINITY_COLOR_PRINTING=1`
*   *Do not use color*—`EFINITY_COLOR_PRINTING=0`

Tip: It can be hard to find specific messages in the Console as they scroll by. Instead, use the Message Browser or Log Browser to see specific messages like warnings and errors. Additionally, you can use the Search button in the Console to jump to a specific string.

Message Browser

The Message Browser gives synthesis-specific messages that result when you elaborate the netlist.

Timing Browser

The Timing Browser shows the critical paths in your design. Refer to **Analyzing Timing** on page 68 for more details.

Log Browser

The Log Browser gives you a way to sort and browse through all of the messages resulting from the compilation flow. It shows some of the same content as the Console (the Console is more verbose). You can filter by where the message appeared in the flow (synthesis, placement, routing, programming) and the type of message (info, warning, or error). The Log Browser lets you search messages using keywords or regular expressions.

Result Tab

The Result tab, which is under the Dashboard, shows all of the reports and files that result from compilation. Double-click on any file to view it in the Code Editor. Additionally, it shows a summary table of the resources used. You can right-click the items in the Result tab to open a context-sensitive menu with shortcut actions. If you double-click a filename, the file opens in the Code Editor (default) or the editor you set in your Efinity preferences (see **Setting General Tool Preferences** on page 16).

Figure 17: Using the Result Tab
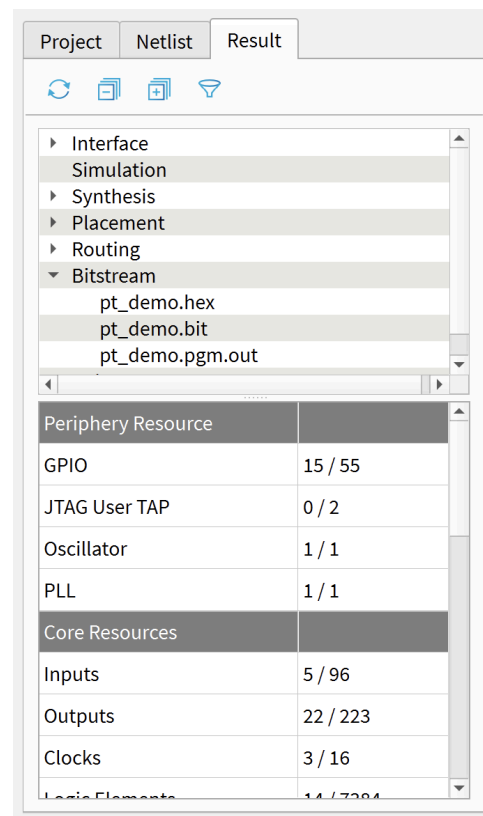
The numbers of inputs and outputs in the Core Resource section represent the connections between the core and the periphery; they are not package pins. See **<project>.place.rpt** on page 150 for more details.

Tip: In Efinity® v2020.2 and higher you can resize the Result tab. Grab the blank space between the Result tab and the Console and drag to resize.

Table 7: Compilation Files and Reports

The software generates these files when you run the flow.

Category	File	Description
Synthesis	<project>.map.v	Post-mapping netlist file for simulation.
	<project>.map.core.v	Post-mapping core netlist file for simulation with the unified design flow.
	<project>.map.peri.v	Post-mapping interface netlist file for simulation with the unified design flow.
	<project>.map.rpt	Synthesis report file; gives a summary of the resources your design uses.
	<project>.map.out	Messages output to the Console during synthesis; includes any synthesis warnings or errors.
	<project>.res.csv	Provides the resource usage for all of the modules in the design.
Placement	<project>.place	Detailed placement report.
	<project>.place.rpt	Resource summary report.
	<project>.place.out	Messages output to the Console during placement.
Routing	<project>.pnr.rpt	Provides the resource summary for inputs, outputs, clocks, LEs, memory, and multipliers (Trion) or DSP Blocks (Titanium and Topaz).
	<project>.route.rpt	Routing report.
	<project>.timing.rpt	Static timing analysis report.
	<project>.route.out	Messages output to the Console during routing.
Bitstream	<project>.hex	Use this file to program in SPI active or passive mode.
	<project>.bit	Use this file for JTAG programming.
	<project>.pgm.out	Messages output to the Console during bitstream generation.

Table 8: Interface Designer Files

The Interface Designer generates these files when you click the Generate Efinity Constraint Files button and when you do a full compile.

File	Description
<project>.interface.csv	Constrains the FPGA design pins used in the interface between the core and the periphery.
<project>.pt.rpt	Interface Design report file with details of the blocks used, I/O banks, global connections, clock region usage, GPIO and dual-function configuration pins used, etc.
<project>.pinout.rpt	Has the board design pinout with pin number, signal name, pin name, I/O bank, etc. in a nicely formatted text file format.
<project>.pinout.csv	Pinout report file formatted as **.csv**.
<project>.pt_timing.rpt	Timing report for the Trion®, Topaz, and Titanium interface logic.
<project>.pt.sdc	Template SDC file to constrain the FPGA design pins based on the interface configuration.
<project>_or.ini	Contains option register information the Programmer uses.
<project>_template.v	Template Verilog HDL file defining the FPGA design pins based on the interface configuration.

File	Description
<project>.unified.isf	ISF file that creates design instances and set their properties based on the interface logic discovered by synthesis in the unified design flow.
<project>.auto_asg.isf	ISF file with interface logic automatically assigned to resources in the unified design flow.
<project>.peri_rtl.v	Interface netlist file (blocks inferred by synthesis) for simulation in the unified design flow.
<project>.peri_pt.v	Interface netlist file (blocks from Interface Designer) for simulation in the unified design flow.

Viewing Place-and-Route Results

You view place-and-route results in the Console pane, in the Result pane, and in the Floorplan Editor.

- The Console displays messages generated during compilation. For example, if the design has too many I/O pins to fit in the target device, compilation will stop and the Console will show the error message.
- The Result pane shows the output and report files for each stage in the flow. Additionally, the Report pane displays a table of the interface resources the design uses; if your design has a debug core, it also shows a table of resources used by the debugger.

> (i) **Note:** Double-click on a file name to open it in the Efinity Code Editor.

- After you have run a project through synthesis, placement, and routing, you can open the Floorplan Editor tool to see a representation of the tiles in the FPGA and the placement of logic, memory, I/O, and other blocks. Click the Floorplan Editor icon in the main toolbar to open the Floorplan Editor.

Tip: Detach the Floorplan Editor tool from the main Efinity window for better viewing.

Figure 18: Floorplan Editor

Click on a Block to Show Information (Unused Blocks Are Empty)

ⓘ **Note:** If you have disabled auto-loading, you cannot view place-and-route results in the Floorplan Editor or Timing Browser, or use the Tcl console. To enable these tools, click the **Load Place and Route Data** button in the main window. Refer to Auto-Load Place-and-Route Data on page 16 for details.

📖 **Learn more:** Refer to the Efinity Trion Tutorial for more information on how to use the Floorplan Editor.

Using the IP Manager

Contents:

- **Supported IP Cores by Family**
- **Using the IP Configuration Wizard**
- **Generated Files**
- **Instantiating IP in Your Project**
- **Managing IP in Your Project**
- **IP Settings File**
- **Getting Updated IP**

The Efinity® IP Manager is an interactive wizard that helps you customize and generate Efinix® IP cores. The IP Manager performs validation checks on the parameters you set to ensure that your selections are valid. When you generate the IP core, you can optionally generate an example design targeting an Efinix development board and/or a testbench. This wizard is helpful in situations in which you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core for different projects.

Note: Not all Efinix IP cores include an example design or a testbench.

The IP Manager is included with the Efinity software v2020.2 and higher.

The IP Manager consists of:
- *IP Catalog*—Provides a catalog of IP cores you can select. Open the IP Catalog using the toolbar button or using **Tools > Open IP Catalog**.
- *IP Configuration*—Wizard to customize IP core parameters, select IP core deliverables, review the IP core settings, and generate the custom variation.
- *IP Editor*—Helps you manage IP, add IP, and import IP into your project.

Supported IP Cores by Family

Not all IP cores work with all Titanium or Trion FPGAs. For example, IP cores that connect to DDR DRAM memory will not work with FPGA that does not have a hard DDR DRAM interface. The following table shows which IP is supported in which FPGA.

Note: Refer to the FPGA Selector Guide for more information about supported blocks in each FPGA package.

Table 9: IP Cores Supported by Family

Refer to the IP core user guide for the IP version history.

IP Core	Trion		Titanium		Topaz	
	Supported	Not Supported	Supported	Not Supported	Supported	Not Supported
AXI Infrastructures						
AXI Data FIFO	All		All		All	
AXI Interconnect	All		All		All	
AXI4-Stream Switch	All		All		All	
Arithmetic						
CORDIC	All		All		All	
Divider	All		All		All	
Integer Square Root	All		All		All	
Bridges and Adaptors						
ABP Interconnect	All		All		All	
APB3 to AXI4 Lite Converter	All		All		All	
Data Pipeline	All		All		All	
Direct Memory Access	All		All		All	
Ethernet						
Triple Speed Ethernet MAC	All		All		All	
10G Ethernet MAC		All	Ti85, Ti135, Ti165, Ti240, Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
Foundation IP						
PLL Dynamic Reconfiguration		All	Ti85, Ti135, Ti165, Ti240, Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
Trion PLL Auto-Reset	All others	T4F49, T4F81, T8F49, T8F81		All		All
Memory						
BRAM Wrapper	All		All		All	

IP Core	Trion		Titanium		Topaz	
	Supported	Not Supported	Supported	Not Supported	Supported	Not Supported
FIFO	All		All		All	
Memory Controllers						
ASMI SPI Flash Controller	All		All		All	
DDR Hard Memory Controller-Reset	T20, T35, T55, T85, T120	T4, T8, T13		All		All
DDR3 Soft Controller		All	All		All	
HyperRAM Controller	All		All		All	
JTAG SPI Flash Loader	All		All		All	
SDRAM Controller	All			All		All
SD Host Controller	All		All		All	
Trion DDR Calibration and Debug	T20 (BGA324 and BGA400 only), T35, T55, T85, T120 FPGAs	T4, T8, T13		All		All
MIPI						
MIPI 2.5G CSI-2 RX Controller MIPI 2.5G CSI-2 TX Controller		All	Ti90, Ti120, Ti180	Ti35, Ti60	Tz110, Tz170	Tz50
MIPI 2.5G DSI TX Controller		All	All			
MIPI D-PHY BIDIR RX Controller MIPI D-PHY BIDIR TX Controller		All	All		All	
MIPI CSI-2 RX Controller MIPI CSI-2 TX Controller		All	All		All	
MIPI D-PHY RX Controller MIPI D-PHY TX Controller		All	All		All	
MIPI DSI RX Controller MIPI DSI TX Controller		All	All		All	
Processors and Peripherals						
Sapphire SoC	All except T4	T4	All			
Sapphire High-Performance SoC			Ti165, Ti240, Ti375		Tz200, Tz325	
Serial Interface Protocols						
I^{2}C	All		All		All	
UART	All		All		All	

Table 10: End of Life IP Cores by Family

IP Core	End of Life in Version	Replaced By	Trion		Titanium	
			Supported	Not Supported	Supported	Not Supported
Jade SoC	2022.1	Sapphire SoC	T8, T13, T20, T35, T55, T85, T120	T4	All	
Opal SoC	2022.1	Sapphire SoC	All except T4	T4	All	
Ruby SoC	2022.1	Sapphire SoC	T35, T55, T85, T120	T4, T8, T13, T20	All	
DDR Hard Memory Controller-Calibration	2022.1	DDR Hard Memory Controller-Calibration and Reset	T20, T35, T55, T85, T120	T4, T8, T13		All
DDR Hard Memory Controller-Calibration and Reset	2024.2	Trion DDR Calibration and Debug	T20, T35, T55, T85, T120	T4, T8, T13		All
FIFO (Legacy)	2021.1	FIFO	All		All	

Using the IP Configuration Wizard

The following steps explain how to customize an IP core with the IP Configuration wizard.

1. Open the IP Catalog.
2. Choose an IP core and click **Next**. The **IP Configuration** wizard opens.
3. Enter the module name in the **Module Name** box.

> **Note:** You cannot generate the core without a module name.

4. Customize the IP core using the options shown in the wizard. For detailed information on the options, refer to the IP core's user guide or on-line help.
5. (Optional) In the **Deliverables** tab, specify whether to generate an IP core example design targeting an Efinix® development board and/or testbench. For SoCs, you can also optionally generate embedded software example code. These options are turned on by default.
6. (Optional) In the **Summary** tab, review your selections.
7. Click **Generate** to generate the IP core and other selected deliverables.
8. In the **Review configuration generation** dialog box, click **Generate**. The Console in the **Summary** tab shows the generation status.

> **Note:** You can disable the **Review configuration generation** dialog box by turning off the **Show Confirmation Box** option in the wizard.

9. When generation finishes, the wizard displays the **Generation Success** dialog box. Click **OK** to close the wizard.

The wizard adds the IP to your project and displays it under **IP** in the Project pane.

Generated Files

The IP Manager generates these files and directories:

- **<module name>_define.vh**—Contains the customized parameters.
- **<module name>_tmpl.v**—Verilog HDL instantiation template.
- **<module name>_tmpl.vhd**—VHDL instantiation template.
- **<module name>.v**—IP source code.
- **settings.json**—Configuration file.
- **<kit name>_devkit**—Has generated RTL, example design, and Efinity® project targeting a specific development board.
- **Testbench**—Contains generated RTL and testbench files.

> **(i) Note:** For encrypted IP, the ModelSim software version of 2022.4 or later is required for successful simulation. For other simulators, the latest version is required.

Instantiating IP in Your Project

The IP Manager creates these template files in the *<project>*/**ip/***<module name>* directory:

- *<module name>*.**v_tmpl.v** is the Verilog HDL module.
- *<module name>*.**v_tmpl.vhd** is the VHDL component declaration and instantiation template.

To use the IP, copy and paste the code from the template file into your design and update the signal names to instantiate the IP.

> **(!) Important:** When you generate the IP, the software automatically adds the module file (*<module name>*.**v**) to your project and lists it in the **IP** folder in the Project pane. Do not add the *<module name>*.**v** file manually (for example, by adding it using the Project Editor); otherwise the Efinity® software will issue errors during compilation.

Do not manually add IP to the Design folder

IP Manager adds generated IP to the IP folder (and your project) automatically

Project	Netlist	Result

- ▼ Ti60F225_devkit_tutorial
  - ▼ Design
    - File : Tutorial_top.vhdl (default)
    - File : counter.vhdl (default)
    - File : BiDi.vhdl (default)
  - ▼ Constraint
    - SDC File : constraint.sdc
  - Simulation
  - Misc
  - ▼ IP
    - ✓ IP : Fifo
  - ISF Info

Property ▼	
Top Module	Tutoria...
Top VHDL Arch	
Device	Ti60F225

IP Editor
New IP
Import IP
Generate All
Check Upgrades

Managing IP in Your Project

You can manage your project's IP in the Project pane under the Dashboard. When you right-click the IP folder, the software shows a context-sensitive menu with these options:

- **IP Editor**—Launches the IP Editor window, which shows the IP instances in your project. You can use this window to add additional IP to your project or to import IP. You import IP using a **settings.json** file, see **IP Settings File** on page 46 for details.
- **New IP**—Launches the IP Catalog.
- **Import IP**—Import an existing IP core using a settings file.
- **Check Upgrades**—Checks the Efinity® directory structure to see if there are any updates to the IP (for example, from a software patch). The Efinity® software checks for available upgrades by default. You can change this setting in the Preferences window. See **Setting General Tool Preferences** on page 16.

Icon	Meaning
✅	The IP core is up to date.
🔼	There is an optional update to the IP core. Efinix recommends that you launch the IP Configuration wizard for the core to make any changes and then re-generate the deliverables.
❗	There is a required update to the IP core. You must launch the IP Configuration wizard for the core to make any changes and then re-generate the deliverables.

Figure 19: Project Tab > IP Folder Context-Sensitive Menu
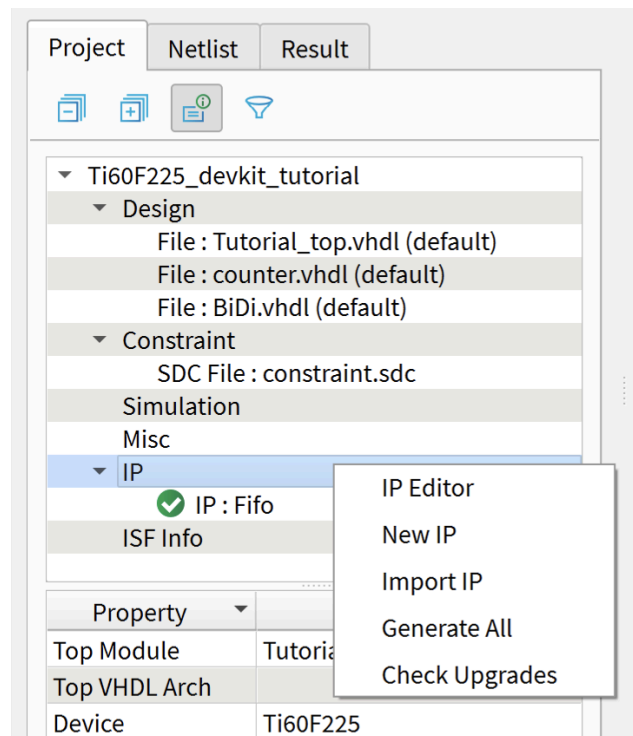
Importing IP

To make it easier for you to re-use IP, you can import a configured IP core. Import the IP by:

1. Right-clicking the IP folder in the Project tab and choosing **Import IP** or clicking the Import an IP button in the IP Editor.
2. Browse for the **settings.json** file of an existing IP core you want to import.
3. The IP Manager launches the **IP Configuration** window.
4. Enter a name in the **Module Name** box.
5. (Optional) Change any of the IP settings (for example, you might not want to generate the example designs).
6. Click **Generate**.

The software generates the imported IP core and adds it to your project.

Managing an IP Core

When you right click the IP core name under the IP folder, the software shows a context-sensitive menu with these options:

- **Open Folder**—Opens the folder containing the IP deliverables.
- **Configure**—Launches the IP Configuration wizard. You can make any changes to the IP parameters and then re-generate it.
- **Generate**—Generates the IP deliverables.
- **Remove**—Remove the IP core and delete all deliverables from your project directory.
- **Open Documentation**—Launches the help for the IP core.

IP Settings File

When you generate an IP core, the IP Manager creates a **settings.json** file. This file contains all of the parameter settings for the customized IP.

You can use this settings file to create another instance of the core with the same settings, or you can modify it to create another core with slightly different settings. For example, you can quickly create FIFOs of varying depths by re-using an existing **settings.json** file.

To create another, modified, instance of the IP core:

1. Right-click the IP folder in the Project pane.
2. Choose Import IP from the pop-up menu.

   (i) **Note:** You can also import IP in the IP Editor window.

3. Browse to the **settings.json** file for the IP core you want to use as a starting point and click **Open**. The IP Configuration wizard opens.
4. Enter the module name for the IP core.
5. Configure the IP core as usual and generate the IP.

Getting Updated IP

Starting with v2020.2, IP is typically delivered with the Efinity® software:

- IP is included and installed as part of the Efinity® software.
- Updated IP with new features is distributed as patches to the Efinity® software.
- Bug fixes (if any) to the IP cores is distributed as patches to the Efinity® software.

New IP cores may be released as beta versions (**.zip** file) in the Support Center before being rolled into the next major Efinity® release.

Chapter 5

Constraining Logic and Assigning Pins

Contents:

- **About the Interface Designer**
- **Get Oriented**
- **Using the Resource Assigner**
- **Resource View**
- **Importing and Exporting Assignments**
- **Scripting an Interface Design**
- **Viewing the Package Pinout**
- **Constraining Logic and Routing Manually (Beta)**

The tools in the Efinity® main window help you design the logic portion of your design. You use the Interface Designer to constrain logic and assign pins to the blocks in the periphery. In the Interface Designer, you connect the signals from your logic design to the pins in the device interface blocks, and then output a constraint file. During compilation, the Efinity® software uses the constraint file to constrain your design to the interface blocks.

Learn more: The **Efinity Trion Tutorial** gives step-by-step instructions on using the Interface Designer with an example **helloworld** design.

The **Titanium Interfaces User Guide** and **Trion Interfaces User Guide** provide instructions on how to use the Interface Designer to configure each block as well as technical details about the interface.

The **Efinity Interface Designer Python API** describes how to create an interface design using scripting.

About the Interface Designer

Trion®, Topaz, and Titanium FPGAs wrap a Quantum®-accelerated core with a periphery that sends signals out to the device pins. The core contains the logic, embedded memory, and multipliers. The device periphery includes blocks such as GPIO pins, LVDS, MIPI, DDR, and PLLs.

The tools in the Efinity® main window help you design the logic portion of your design. You use the Efinity Interface Designer to build the peripheral portion of your design.

Figure 20: Conceptual View of Interface Blocks

Programmable Core Fabric:
Create your RTL design for the core fabric using Efinity design tools.

Interface Blocks:
Use the Efinity Interface Designer to create and define these blocks and to connect them to your RTL design via the signal interface.

Signal Interface:
Connects the core fabric to the interface blocks

Note: The number and locations of blocks are shown for illustration purposes only. The actual number and position depends on the device.

Get Oriented

The Interface Designer has four main sections:

- *Design Explorer*—Provides a list view of the interface blocks you have in your design organized by block type. It also includes device-wide settings for the I/O banks and configuration options. Select a block to display it's summary and editor.
- *Block Summary*—Displays the current settings for the selected block.
- *Block Editor*—Provides options and settings for the selected block. The editor may have more than one tab, depending on the block.
- *Resource Assigner*—Provides an easy, tabular method for assigning resources. View by instance (default) or resource.

Figure 21: Interface Designer
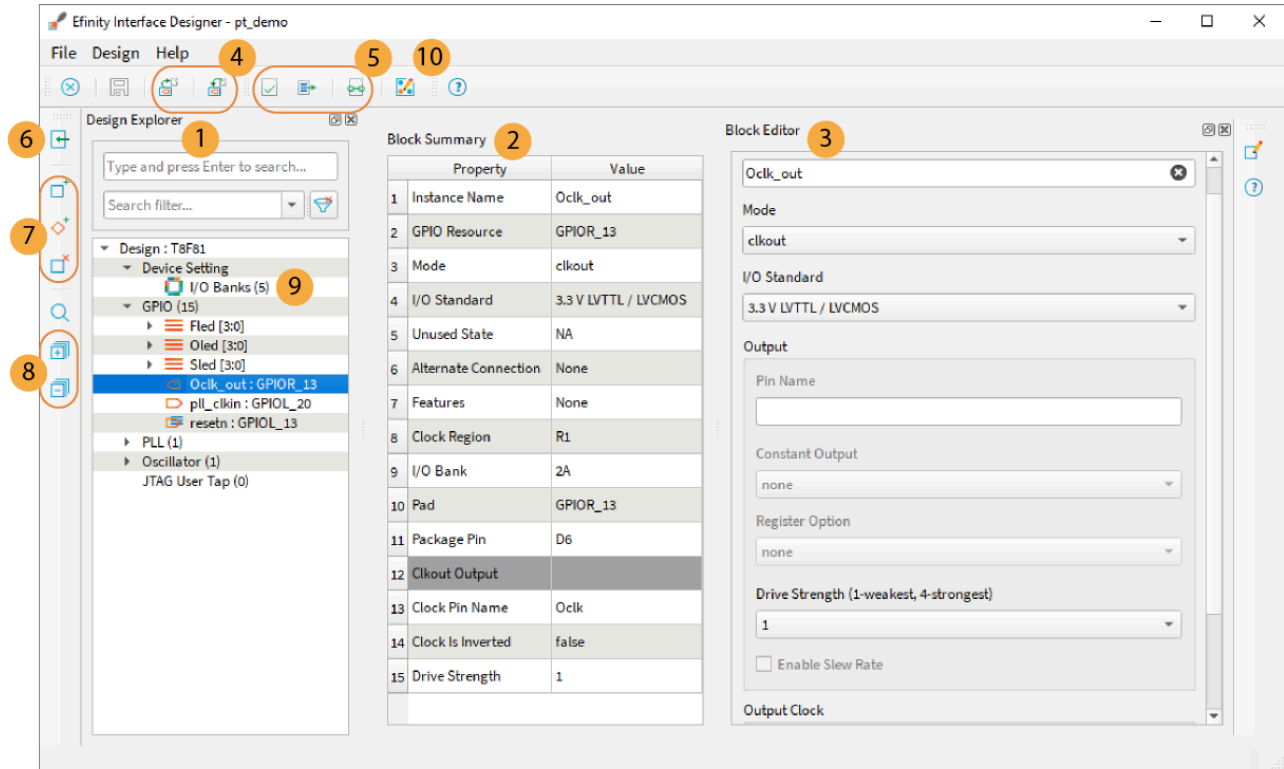
Notes:
1. The Design Explorer shows the interface blocks in your design. They are organized by block type.
2. The block summary shows the settings for the block selected in the Design Explorer.
3. Use the Block Editor to add or change settings for the interface block.
4. You can import or export GPIO resource assignments using a **.csv** or **.isf** file.
5. Use the project management tools to perform design checks, view reports, generate constraints, etc.
6. Click Show/Hide Resource Assigner to toggle a tabular view of assignments.
7. Use the block tools to add or delete blocks and buses.
8. Expand or collapse the Design Explorer folders.
9. The number in parentheses shows the number of used blocks.
10. The Package Planner lets you see the pins and assignments graphically.

When you first open the Interface Designer for your project, the Design Explorer shows the Device Settings folder (with default settings) and empty folders for the interface blocks your chosen device supports. You need to add blocks as required for your design.

Figure 22: Resource Assigner
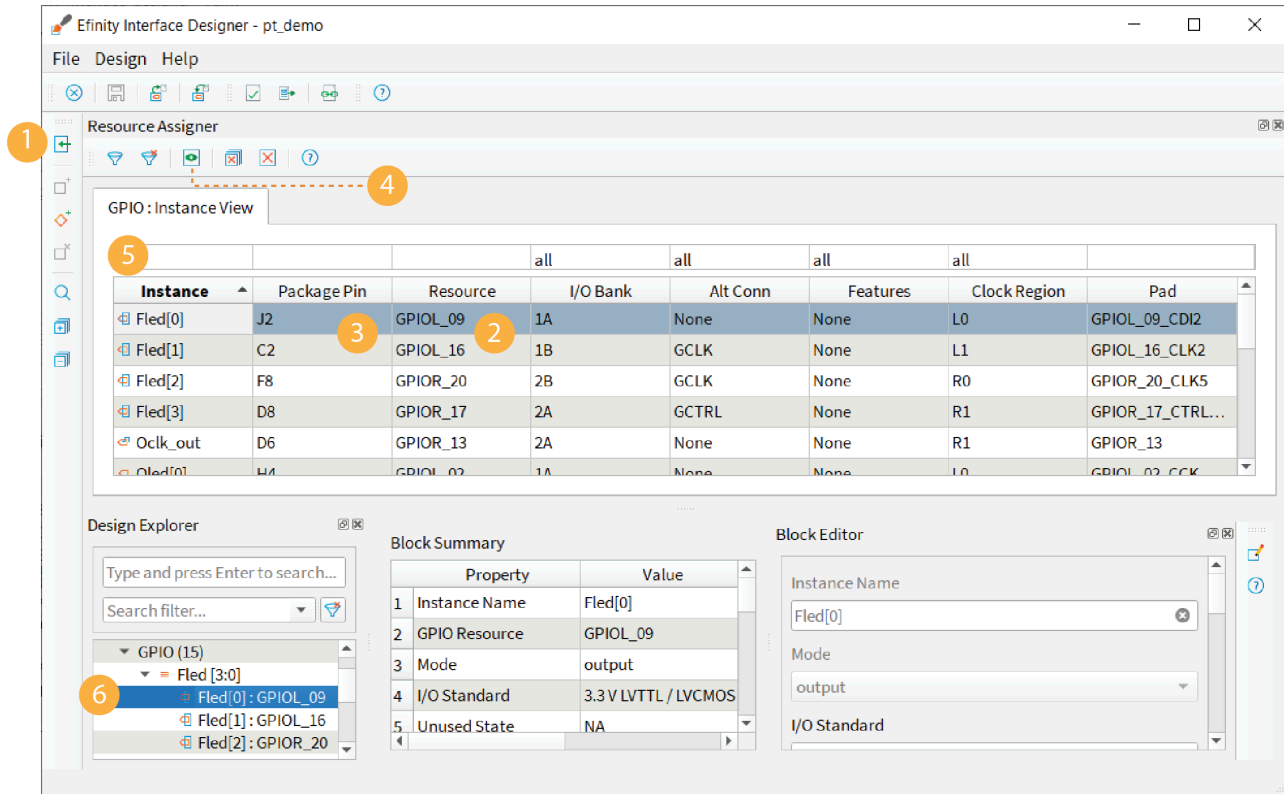
Notes:
1. Show or hide the Resource Assigner.
2. Double-click in the Resource cell to open the list of available resources.
3. Double-click in the Package Pin cell to open the list of available pins.
4. Click the Switch View button to toggle between Instance View and Resource View.
5. Type in the filter cell above the column you want to filter.
6. Selecting a block in the Design Explorer highlights it in the Resource Assigner.

Using the Resource Assigner

⬅ Resource Assigner	◉ Switch View	✕ Clear Selected Resource	✕ Clear All Resources
▽ Show/Hide Filter	▽ Reset Filter		

The Resource Assigner provides a tabular view of all GPIO resources in your chosen FPGA and information about them, such as whether they are used, the I/O bank, pad, and package pin, and the instance assigned to the resource.

- The **GPIO: Instance View** shows all GPIO instances in your project.
- The **GPIO: Resource View** shows all GPIO, LVDS, and MIPI RX or TX lane resources and the resources to which you assigned them.

ⓘ **Note:** In the Efinity® software v2021.1, you can only view the resources used for LVDS and MIPI lanes in the Resource Assigner. You cannot change or assign resources in this view.

To assign a resource:

1. Open the Resource Assigner by clicking the Show/Hide Resource Assigner button. The software opens to the Instance View, which lists all instances in the design.

   ⓘ **Note:** Click Switch View to toggle between instance view and resource view.

2. In instance view, you can assign pins or resources to the instance. Double-click in the table cell for the item you want to assign. The software displays a drop-down list of available selections.
3. Select an unused resource, instance, or pin.

   ⓘ **Note:** If you select a used resource, instance, or pin, the software makes the new assignment, which replaces the previous assignment.

4. Press Enter.

ⓘ **Note:** When LVDS resources are used for both LVDS and GPIO within the same bank, they must be separated by 2 unused pairs of LVDS pins to avoid any unwanted interference. The Efinity software issues an error if you do not leave this separation. Refer to Table 1.

ⓘ **Note:** Titanium: When using HSIO pins as GPIO, make sure to leave at least 1 pair of unassigned HSIO pins between any GPIO and HSIO pins in the same bank. This separation reduces noise. The Efinity software issues an error if you do not leave this separation.

Resource View

When assigning GPIO, sometimes you want to know which resource can be used as a global clock, global control, or other special function. You can look it up in the pin table for the FPGA and package you are targeting, but an easier way is to use the Resource View in the Resource Assigner.

1. Click the Switch View button to open the Resource View.
2. Double-click in the filter box above the **Alt Conn** column and choose the connection type, for example, **GCLK**.

Figure 23: Resource View
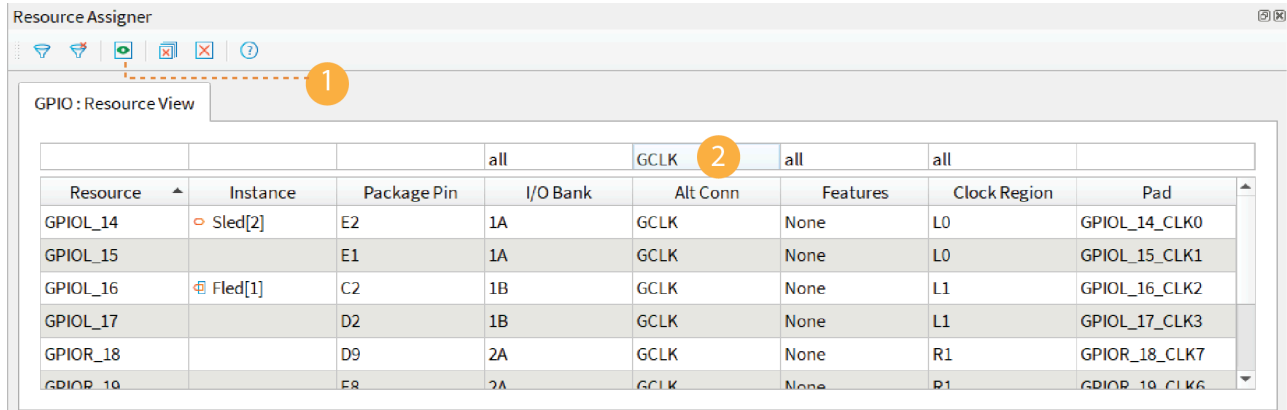
Importing and Exporting Assignments

Although it is nice to use a GUI for adding blocks, in some cases it may be easier to use another format. The Interface Designer lets you import and export assignments using an Interface Scripting File (**.isf**) or comma separated values (**.csv**) file.

When the software reads an imported **.isf**, it processes the entire imported file and shows any issues it found. The import only fails for catastrophic errors. The software:

- Creates new instances defined in the file that do not already exist in the GUI
- Overwrites assignments for existing instances with settings from the file
- Does not delete instances that are in the GUI but were not defined in the file

When the software reads an imported **.csv** file, it compares the imported assignments to the original assignments and reports any issues. If the software finds warnings, it displays them but allows you to finish the import. If it finds errors, it will not finish the import. When importing, the software:

- Deletes instances that you removed
- Creates newly defined instances
- Replaces instances you renamed with the new name

Learn more: For help understanding messages, refer to the "Design Check" topics in the Titanium Interfaces User Guide. These topics describe the messages the Interface Designer generates and gives suggestions on how to fix errors and warnings.

Interface Scripting File

The Interface Scripting File (**.isf**) contains all of the Python API commands to re-create your interface. You can export your design to an **.isf**, manipulate the file, and then re-import it back into the Efinity® software. Additionally, you can write your own **.isf** if desired.

In addition to using the API, you can export and import an **.isf** in the Interface Designer GUI. Click the Import GPIO or Export GPIO buttons and choose **Interface Scripting File (.isf)** under **Format**.

Example: Example Interface Scripting File

```
# Efinity Interface Configuration
# Version: 2020.M.138
# Date: 2020-06-26 14:22
#
# Copyright (C) 2017 - 2020 Efinix Inc. All rights reserved.
#
# Device: T8F81
# Package: 81-ball FBGA (final)
# Project: pt_demo
# Configuration mode: active (x1)
# Timing Model: C2 (final)

# Create instance
design.create_output_gpio("Fled",3,0)
design.create_inout_gpio("Sled",3,0)
design.create_output_gpio("Oled",3,0)
design.create_clockout_gpio("Oclk_out")
design.create_pll_input_clock_gpio("pll_clkin")
design.create_global_control_gpio("resetn")

# Set property, non-defaults
design.set_property("Fled","OUT_REG","REG")
design.set_property("Fled","OUT_CLK_PIN","Fclk")
design.set_property("Sled[0]","IN_PIN","")
design.set_property("Sled[0]","OUT_PIN","Sled[0]")
design.set_property("Sled[1]","IN_PIN","")
design.set_property("Sled[1]","OUT_PIN","Sled[1]")
design.set_property("Sled[2]","IN_PIN","")
design.set_property("Sled[2]","OUT_PIN","Sled[2]")
design.set_property("Sled[3]","IN_PIN","")
design.set_property("Sled[3]","OUT_PIN","Sled[3]")
design.set_property("Oclk_out","OUT_CLK_PIN","Oclk")

# Set resource assignment
design.assign_pkg_pin("Fled[0]","J2")
design.assign_pkg_pin("Fled[1]","C2")
design.assign_pkg_pin("Fled[2]","F8")
design.assign_pkg_pin("Fled[3]","D8")
design.assign_pkg_pin("Sled[0]","E6")
design.assign_pkg_pin("Sled[1]","G4")
design.assign_pkg_pin("Sled[2]","E2")
design.assign_pkg_pin("Sled[3]","G9")
design.assign_pkg_pin("Oled[0]","H4")
design.assign_pkg_pin("Oled[1]","J4")
design.assign_pkg_pin("Oled[2]","A5")
design.assign_pkg_pin("Oled[3]","C5")
design.assign_pkg_pin("Oclk_out","D6")
design.assign_pkg_pin("pll_clkin","C3")
design.assign_pkg_pin("resetn","F1")
```

.csv File for GPIO Blocks

For larger designs with lots of GPIO, it can be simpler to use a spreadsheet application to make assignments. The Resource Assigner allows you to import and export GPIO block assignments using a comma separated values (**.csv**) file. The **.csv** file includes the package pin and pad name, the instance name, and the mode. You can use this method for any type of GPIO, including LVDS pins used as GPIO or HSIO pins used as GPIO.

Table 11: Example GPIO .csv File

Package Pin-Pad Name	Instance Name	Mode
G5-GPIOL_00		
J4-GPIOL_01_SS_N		
H4-GPIOL02_CCK		
G4-GPIOL_03_CDI4	led[0]	output
F4-GPIOL04_CDI0	led[1]	output
J3-GPIOL_05_CDI5	rstn	input
H3-GPIOL_06_CDI1		
...		
(3)	led[6]	inout

When working with the **.csv** file:
- Add your assignments to the **Instance Name** and **Mode** columns.
- Do not modify the package pin-pad names.
- For the mode, specify: input, output, inout, clkout, or none

Note: You cannot make advanced settings such as alternate connections or registering. To make these settings, use the Block Editor.

When the software reads an imported **.csv** file, it performs a comparison between the **.csv** assignments and the original GPIO block assignments and reports any issues. If the software finds warnings, it displays them but allows you to finish the import. If it finds errors, it will not finish the import. When importing, the software:
- Deletes instances that you removed
- Creates newly defined instances
- Replaces instances you renamed with the new name

Scripting an Interface Design

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.[4] Efinix distributes a copy of Python 3 with the Efinity® software to support point tools such as the Debugger and to allow users to write scripts to control compilation.

You use the Efinity® Interface Designer to build the peripheral portion of your design, including GPIO, LVDS, PLLs, MIPI RX and TX lanes, and other hardened blocks. Efinix provides a Python 3 API for the Interface Designer to let you write scripts to control the interface design process. For example, you may want to create a large number of GPIO, or

---

[3] Unassigned instances have a blank field for the Package Pin-Pad Name column.
[4] Source: **What Is Python? Executive Summary**

target your design to another board, or export the interface to perform analysis. This user guide describes how to use the API and provides a function reference.

Learn more: Refer to the Python web site, **www.python.org/doc**, for detailed documentation on the language.

Learn more: For more information on using the Python API to script an interface, refer to the **Efinity Interface Designer Python API**.

Viewing the Package Pinout

The Package Planner provides a visual representation of the FPGA package pins. Each pin is color coded by function (such as GPIO, configuration, power, etc.) letting you easily see which package pin has which function. Additionally, you can highlight I/O banks, PLL reference clocks, global clocks, and global controls so you can quickly find a specific pin that has the feature you need. This tool is helpful when planning how to map the signals in your design to package pins.

Figure 24: Package Planner
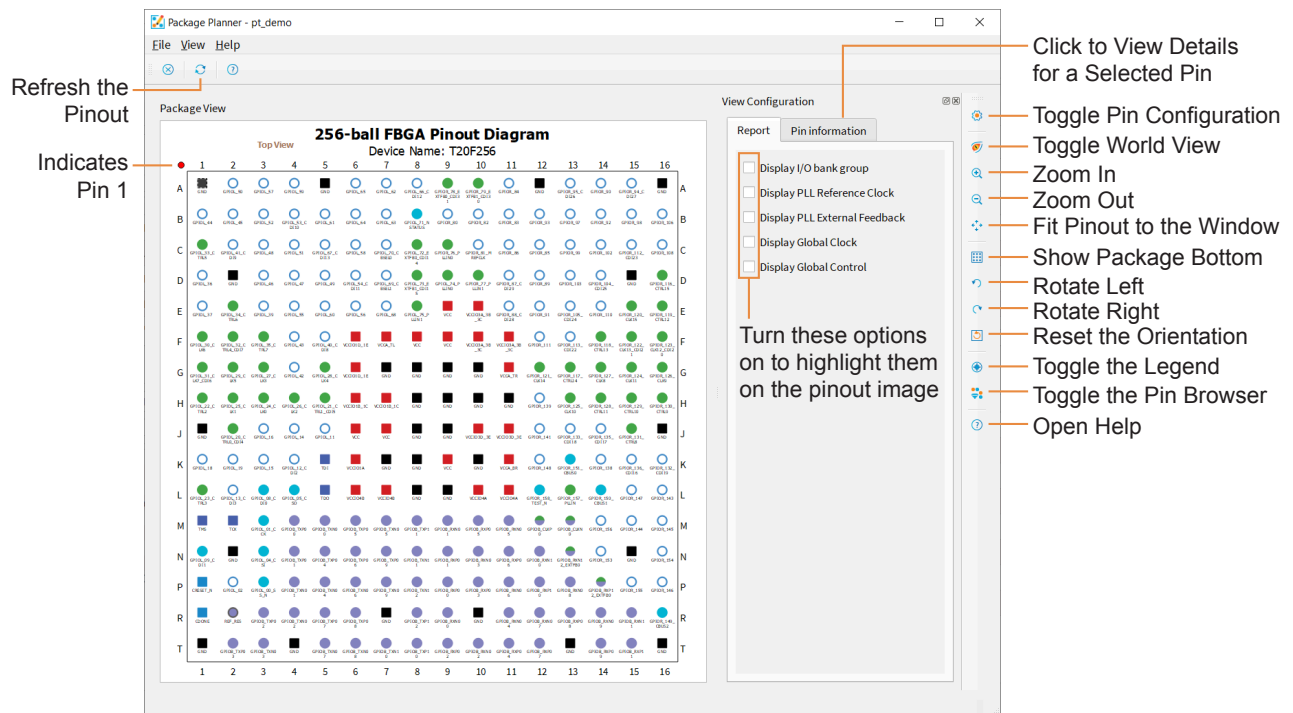
Selecting a Pin

Click a pin in the pinout to highlight it. The **Pin Information** tab opens to show the details about the selected pin. Open the Legend to view the meaning of the pins' color coding.

Figure 25: Selected Pin
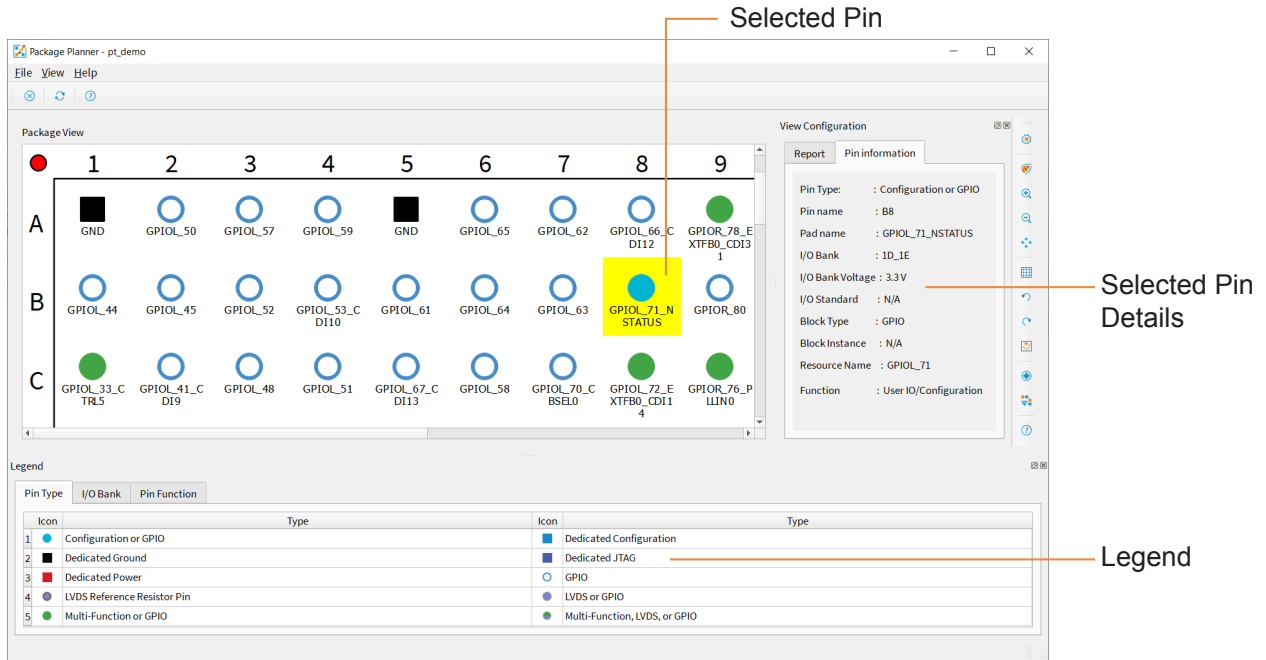
You can also hover over a pin for a quick view of the pin details.

Figure 26: Pin Quick View
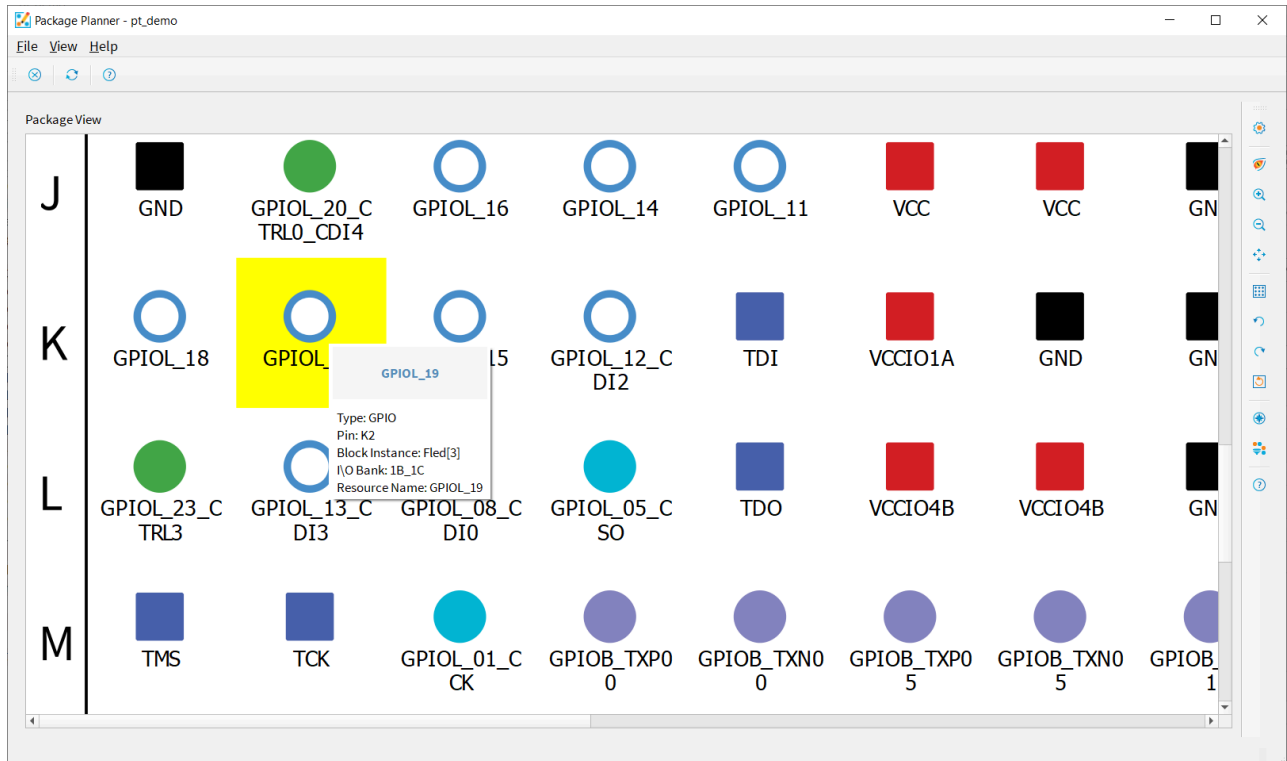
Browsing for Pins

The Package Planner has a **Pin Browser**, which has a table view similar to the **Resource Assigner**. You can filter pins and then select them in the **Pin Browser**. The selected pin is highlighted in the pinout.

Figure 27: Browsing for Pins

Constraining Logic and Routing Manually (Beta)

The Efinity software v2022.1 and higher lets you assign logic to a specific location in the FPGA's core. With this method, you can place your design's logic manually instead of letting the software place it for you.

In v2022.2 and higher, you can also manually constrain routing to specific paths. When you constrain routing you also need to constraint the logic to which the nets connect.

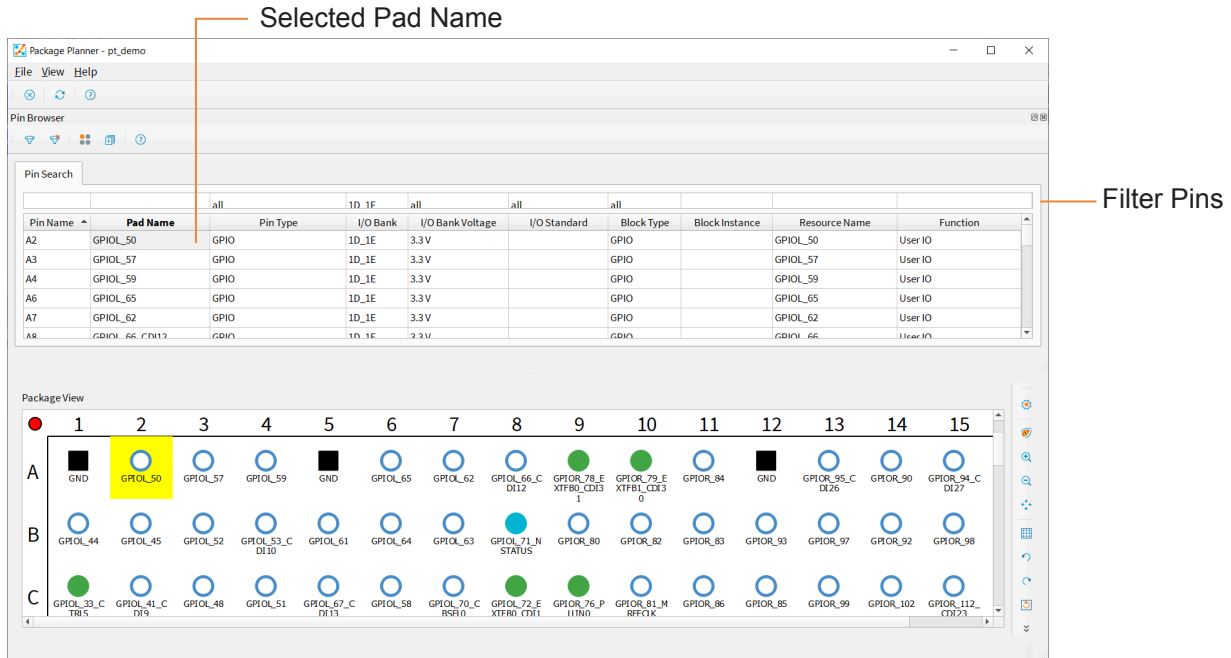Placing logic and/or routing manually is an advanced technique, so make sure that you fully understand the rules and restrictions as described in the following sections.

Important: These features are beta.

Tiles

The FPGA is made up of a grid of tiles. Most tiles are for logic/routing and others are for functions like RAM, multipliers, or DSP. The following table shows the types of tiles by family and their use.

Table 12: FPGA Tile Types

Tile	Trion	Titanium	Used for
EFT	✓	✓	Logic and routing with register
EFL	✓		Logic and routing without register
EFM		✓	Logic, routing, register, and shift register
RAM	✓	✓	RAM blocks
MULT	✓		Multiplier blocks
DSP48		✓	DSP blocks

When you view your design's placement in the Floorplan Editor, you can click on a tile to view its type and other details. In the following figure, the selected blue tile is an EFT and is used for logic.

Tip: The Floorplan Editor provides a graphical way to find logic you want to constrain.

Figure 28: Tiles in the Floorplan Editor
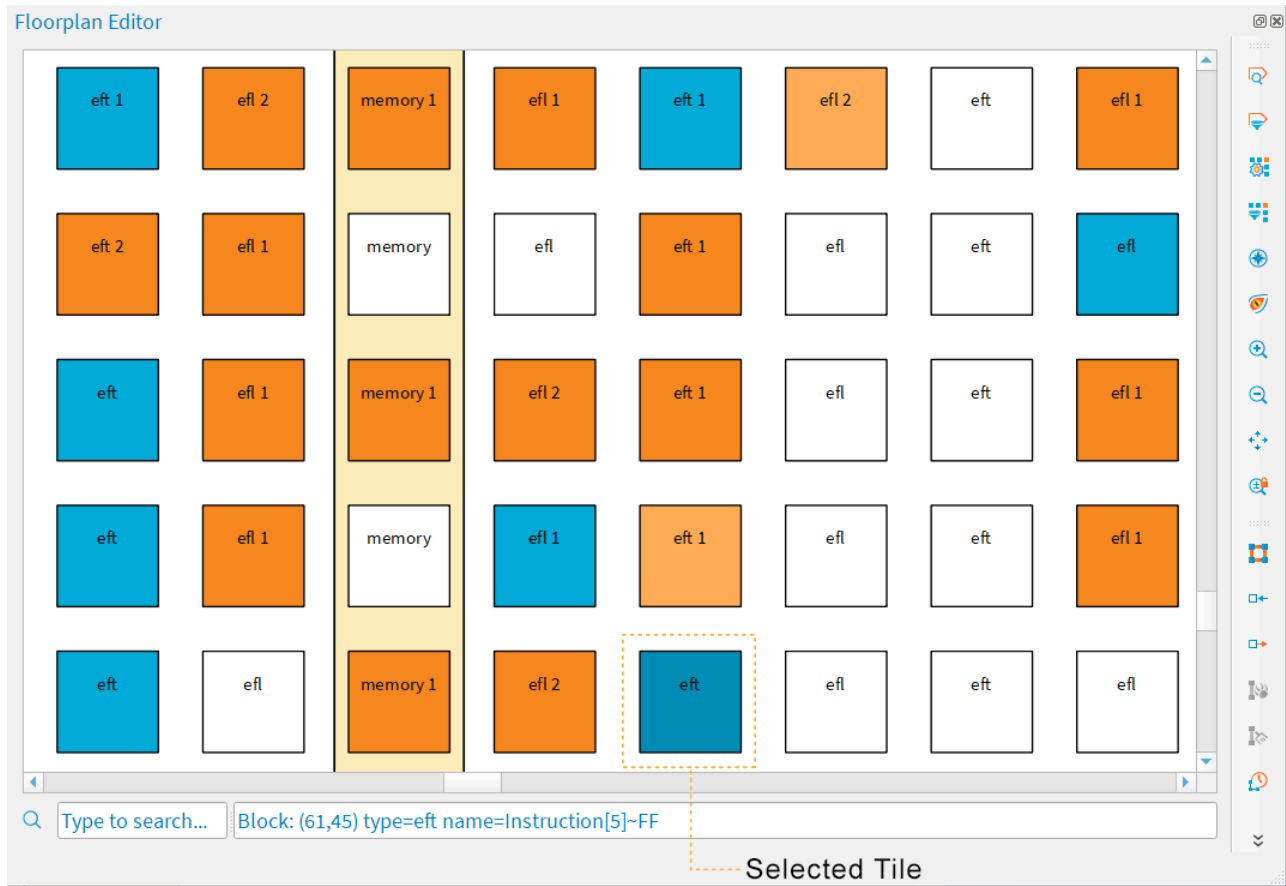
Notice that some tiles in the floorplan have a number. This number indicates how many routing lines are used in that tile. A tile used for logic (blue) can also be used as routing (indicated by the number). Orange means a tile is only used as routing.

Working with Primitives

During synthesis, the software maps your design's logic—LUTs, RAM, flipflops, etc.—to primitives. These primitives occupy specific locations (tiles or groups of tiles). Each tile has one or more sub-blocks in which to place a primitive. Placing multiple primitives into the same tile is called *packing*.

The following tables show the types of primitives, the tiles where you can place them, and the sub-blocks they can occupy.

Table 13: Mapping Trion Primitives to Tiles and Sub-Blocks

Tile	Sub-Block			
	0	1	2	3
EFT	EFX_LUT4 EFX_ADD	–	EFX_FF	–
EFL	EFX_LUT4 EFX_ADD	–	–	–
RAM	EFX_RAM_5K EFX_DPRAM_5K	Reserved	–	–
MULT	EFX_MULT	–	–	–

Table 14: Mapping Titanium Primitives to Tiles and Sub-Blocks

Tile	Sub-Block			
	0	1	2	3
EFT	EFX_LUT4 EFX_ADD EFX_COMB4	Reserved	EFX_FF	–
EFM	EFX_LUT4 EFX_ADD EFX_COMB4 EFX_SRL8	Reserved	EFX_FF	–
RAM	EFX_RAM10 EFX_DPRAM10	Reserved	–	–
DSP48	EFX_DSP48 EFX_DSP24 EFX_DSP12	EFX_DSP24 EFX_DSP12	EFX_DSP12	EFX_DSP12

The following table shows another view of the same mappings.

Table 15: Mapping Primitives to Tiles

Primitive	Compatible Tiles		Allowed Sub-Block Indices
	Trion	Titanium	
EFX_LUT4	EFT, EFL	EFT, EFM	0
EFX_ADD	EFT, EFL	EFT, EFM	0
EFX_COMB4	EFT, EFL	EFT, EFM	0
EFX_FF	EFT	EFT, EFM	2
EFX_SRL8	–	EFM	0
EFX_RAM_5K	RAM	–	0
EFX_DPRAM_5K	RAM	–	0
EFX_RAM10	–	RAM	0
EFX_DPRAM10	–	RAM	0
EFX_MULT	MULT		0
EFX_DSP48	–	DSP48	0
EFX_DSP24	–	DSP48	0, 1
EFX_DSP12	–	DSP48	0, 1, 2, 3

Finding Primitive Cell Names

When the software maps your design to primitives, it assigns a cell name to each instance. To view the primitive cell names:

- In the Dashboard's **Netlist** tab, click the Load Synthesized Netlist icon and expand **Leaf Cells**.
- Open the *<project>***.map.v** file (in the Dashboard, go to **Result pane > Synthesis**). This file is in the project's **outflow** directory.

Enabling Manual Assignments

Because manual assignments are beta in the Efinity software v2022.1, v2022.2, and 2023.1, you must enable them with an **.ini** file.

1. Create a text file named **efx_pnr_settings.ini** and save it in your project directory.
2. Add the following line to the **.ini** file:

```
loc_assignment = <filename>.placeloc
```

When you synthesize your design, the software uses the assignments in the *<filename>***.placeloc** file.

Assignment Rules

Follow these rules when creating assignments.

General Rules

- You can only constrain logic in the core (use the Interface Designer for I/O constraints).
- You can only constrain primitive cells. If two primitives cells *can* be packed together, you can assign them to the same location. The sub-block index must be unique for each primitive cell in a location. For example, if you assign four EFX_DSP12 primitives to the same tile, they must each have a different sub-block.
- The software does not pack manually assigned cells with unassigned cells. For example, if you place a EFX_DSP12 into a DSP tile at sub-block 0 and do not assign any other sub-blocks, the software will not pack any other DSP logic into that tile, leaving sub-blocks 1, 2, and 3 empty. Similarly, only assigning flipflops (which use sub-block 2) uses more overall resources because sub-block 0 is left empty.

> **(!) Important:** Because assigned and unassigned cells are not packed together, make sure to "fill up" the tile with logic. Otherwise you can end up using more tiles than needed.

Flipflops

- An EFX_FF can be packed alone or with its driver cell (EFX_LUT4, EFX_SRL8, EFX_ADD, or EFX_COMB4).
- An EFX_FF can only be packed with an EFX_SRL8 if they share CE and CLK inputs and if the EFX_FF does not have an inverted input.
- An EFX_FF cannot be packed if it has an inverted input connected to a multi-fanout net.

RAM, Multiplier, and DSP

- EFX_MULT, EFX_DSP48, and all RAM primitives cannot share a tile with any other cells.
- Two EFX_DSP24 primitives or up to four EFX_DSP12 primitives **not** connected by CASCIN/CASCOUT signals can be packed together and share a location.

Chains

EFX_DSP48, EFX_DSP24, EFX_DSP12, EFX_ADD, and EFX_SRL8 can form chains. If one cell in the chain is assigned a location, every other cell in the chain must also be assigned a location, in the correct order.

Creating a Location Assignment File

The location assignment file is a text file with the extension **.placeloc**. Each assignment is on a single line with tabs or spaces between the data:

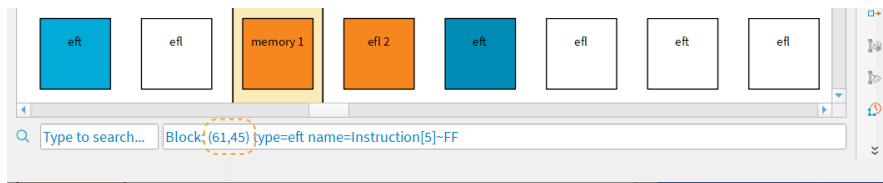```
<block name>   <x>   <y>   <subblk>
```

- *< block name >* is the primitive cell name.
- *< x >* is the horizontal location.
- *< y >* is the vertical location.
- *< subblk >* is the sub-block location.

You must include all data for each assignment.

Any text following a # character is ignored (treated as a comment).

**Tip:** Use the Floorplan Editor to help you find the *x, y* coordinates for a tile. When you click a tile the coordinates are shown in ().



*x,y coordinates for the selected tile*

To make it easier for you to create assignments, the Efinity software can dump all placement data into a file when placement finishes. You can copy and paste the primitive cells you want to constrain into your **.placeloc** file and then modify the *x, y* coordinates.

To dump the placement data, add the following line to your **efx_pnr_settings.ini** file and re-run the placer.

```
dump_placeloc = on
```

Important: Do NOT simply copy and paste the entire dump file into your **.placeloc** file or the software may not be able to perform placement efficiently. Only copy the primitives you want to constrain.

Example: LUT and Flipflop

The example packs an EFX_FF with its driver, LUT_A, an EFX_LUT4.

```
#block name   x   y    subblk
#----------  --  --  ------
LUT_A        3    3    0
FF_B         3    3    2    # LUT_A drives FF_B
```

Example: SRL8 Chain

This example assigns locations to every cell in an SRL8 chain.

```
#block name   x     y    subblk
#----------  --    --  ------
first_srl8    5     4     0
second_srl8   5     5     0
third_srl8    5     6     0
fourth_srl8   5     7     0
```

Example: Parallel Cascaded DSP Block

This example assigns locations to every EFX_DSP24 across two chains. There can be two EFX_DSP24 cells per DSP tile.

```
#block name      x    y    subblk
#----------      --   --  ------
chain0_dsp24_0   17   2     0
chain1_dsp24_0   17   2     1
chain0_dsp24_1   17   22    0
chain1_dsp24_1   17   22    1
```

Constraining Routing Manually (Titanium Only, Beta)

With the Efinity software v2022.2 and higher the router lets you manually constrain routing traces for Titanium FPGAs. This feature is beta.

! **Important:** The feature is only supported for Titanium FPGAs.

After you compile your design once, you can lock down (or constrain) specific nets to specific paths. For any future compilations, the software routes these constrained nets in the same way. To constrain nets, you also need to constrain the logic to which the nets connect. See **Constraining Logic and Routing Manually (Beta)** on page 58 for information on making logic constraints.

You can combine constrained logic and constrained routing to preserve the placement and routing of a small part of your design, letting the rest change as you compile. This feature can be useful when logic (such as a sampling delay line) with very specific routing requirements must be locked down early in the design cycle. Additionally, this feature lets you preserve place and route for connections that have difficult timing constraints.

Routing Constraint Flow

To use routing constraints, follow this procedure:

1. Determine which nets and cells should be constrained.
2. Run the Efinity software, adjusting your design for each iteration, until the nets meet timing.
3. When the nets meet timing, use an **.ini** file to tell the software to save the placement and routing data to templates. (See **Generate .rcf Template** on page 65)
4. Do not make any changes to the design and re-compile.
   The software creates these template files:
   - Placement template *<project>*.**out.placeloc**
   - Routing template *<project>*.**rcf.template**

   The software also creates a routing traces file *<project>*.**troutingtraces**.
5. Move these three files out of the **outflow** directory, for example, move them up one level to the main project directory.
6. Copy and paste the cells and nets you want to constrain from the two template files to your own files. You do not want to copy everything! (See **Creating a Routing Constraint File** on page 65 and **Creating a Location Assignment File** on page 62)
7. Add your new constraint files to an **.ini** file. (See **Enabling Routing Constraints** on page 64)
8. Continue to change your design as needed. When you compile, the software will place and route the constrained logic and nets as defined in the constraint files.

Enabling Routing Constraints

Because routing constraints are beta in the Efinity software v2022.2, you must enable them with an **.ini** file. Because routing constraints are used with logic constraints, you enable them both.

1. Create a text file named **efx_pnr_settings.ini** and save it in your project directory.
2. Add the following lines to the **.ini** file:

```
loc_assignment = <path>/<filename>.placeloc
rcf_file = <path>/<filename>.rcf
```

When you synthesize your design, the software uses the assignments in the specified files.

Generate .rcf Template

You tell the software to generate templates in the **efx_pnr_settings.ini** file. Because routing constraints are used with logic constraints, you enable templates for both.

1. If you do not already have one, create a text file named **efx_pnr_settings.ini** and save it in your project directory.
2. Add the following lines to the **.ini** file:

```
dump_placeloc = on
generate_rcf_template = on
```

When you compile your design, the software generates the *<project>*.**out.placeloc** and *<project>*.**rcf.template** files.

> **Important:** Do not generate these templates until you are ready to lock down the routing.

Creating a Routing Constraint File

The routing constraint file is a text file with the extension **.rcf**. The file format is line-oriented; each command is on a single line with spaces between the data.

To make it easier for you to create assignments, the Efinity software can dump all routing data into a template file when routing finishes. (See **Generate .rcf Template** on page 65) You copy and paste the nets you want to constrain into your own **.rcf**.

> **Important: Do NOT** simply copy and paste the entire template file into your **.rcf** or the software may not be able to perform routing efficiently. Only copy the nets you want to constrain.

The **.rcf** has these components:

- routeTraceFile *<path>*/*<filename>*.**troutingtraces** is the file that has the saved net traces you want to use.
- restoreNetFromTraceFile *<net>* is the net you want to constrain
- Lines beginning with # are comments

```
# The constrained router flow will use the following trace file to restore constrained nets
routeTraceFile  <path>/<project>/<filename>.troutingtraces

# Here is a list of available nets that can be restored from the trace file
# You can use (#) to comment any net that you would like to exclude
restoreNetFromTraceFile  rst_i
restoreNetFromTraceFile  net_1
restoreNetFromTraceFile  net_2
# restoreNetFromTraceFile  net_3    # this net is ignored
```

Best Practices for Constraining Routing

Follow these guidelines when constraining routing to ensure consistency for register and signal names when you re-compile.

- Use a consistent naming convention, such as `netname_LOCKED`, for all constrained nets. This methodology lets you identify them in the template files more easily.
- Limit routing constraints (if possible) to named single-fanout signals between named registers.
- Use the `syn_keep` synthesis attribute—for all locked registers and the signals between locked registers—to tell synthesis to keep the signals during optimization. If you do not use `syn_keep`, the software might optimize away the net you want to constrain.

```
(* syn_keep = "true" *) wire netname_LOCKED;
```

- In your **.rcf**, **do not** point to the **.troutingtraces** file in the project **outflow** directory. This file is overwritten each time you compile. Instead, move the **.troutingtraces** file into another directory and point to it in that location.
- Use routing constraints sparingly; excessive constraints make it hard to close timing.
- Implement constrained routing as late in the design cycle as possible (when you have fewer changes to your design).

Note: Although you can use constrained routing on combinational paths, primitive cell names (for example LUT names) on these paths may change if you modify unrelated sections of the design and re-run synthesis. As a result, you may need to update your *<project>*.**out.placeloc** file to reflect the new primitive cell names.

Example Flow

Assume that your design has the following register path: `rlock0` to `rlock1` to `rlock2`, and that this path meets timing. We want to constrain this path while we modify another part of the design (that is independent from this constrained path).

1. To prevent synthesis from optimizing away the registers and wires, use `syn_keep` in the Verilog HDL design:

    ```
    (* syn_keep = "true" *) reg rlock0;
    (* syn_keep = "true" *) reg rlock1;
    (* syn_keep = "true" *) reg rlock2;
    (* syn_keep = "true" *) wire rlock0_net;
    (* syn_keep = "true" *) wire rlock1_net;
    (* syn_keep = "true" *) wire rlock2_net;
    ```

2. Run place and route with the options `dump_placeloc = on` and `generate_rcf_template = on`. You add these options to a **efx_pnr_settings.ini** file, one option per line, and save the file in the project folder.

3. Examine the generated file *<project>***.out.placeloc** to identify the placed location of the locked registers:

    ```
    rlock0~FF 16 49 2
    rlock1~FF 16 50 2
    rlock2~FF 16 44 2
    ```

4. Examine the generated file *<project>***.rcf.template** to find the nets between the registers in the **.rcf.template** file:

    ```
    restoreNetFromTraceFile rlock0_net
    restoreNetFromTraceFile rlock1_net
    ```

5. Remove all cells except the locked ones from the *<project>***.out.placeloc** file and save it as your own file called **my.placeloc**. Similarly, remove all nets except the constrained ones from *<project>***.rcf.template** and save it as your own file called **my.rcf** file.

6. Add the following settings to your **efx_pnr_setting.ini** file:

    ```
    loc_assignment = my.placeloc
    rcf_file = my.rcf
    ```

You can now modify any other part of the design and re-run the synthesis and place and route. The software constrains the paths you specified.

Analyzing Timing

You use static timing analysis (STA) to measure the timing performance of your design. The software generates a timing report based on the design's place and route results and the project's SDC file. The software provides several tools for viewing and cross-probing timing results:

- The Timing Browser helps you explore your design's critical paths and the cells of those paths.
- The Floorplan tool shows the locations of the paths and cells in the fabric.
- The Tcl Console helps you analyze and explore timing.

After analyzing your design's timing, you can update your SDC file if needed. To apply the new SDC settings to see how they affect placement and routing, re-run the flow from synthesis to the end.

Learn more: For detailed information on performing timing analysis and closing timing, refer to the **Efinity Timing Closure User Guide**.

Chapter 7

Simulating

Contents:

You can use the command line flow to perform RTL simulation on your design's source files as well as simulation on the post-synthesis netlist file.

Note: In the Efinity software v2024.2 and higher you can also simulate the simple interface blocks, such as GPIO, and PLLs. The supported interfaces blocks are listed in the Trion, Titanium, and Topaz primitives user guides. Simulation support for additional blocks will be available in upcoming releases.

Simulation involves the following steps:

1. Perform behavioral RTL simulation to ensure that the RTL design matches your testbench functionality. You can include multiple Verilog HDL design files. Use the `--flow rtlsim` flag.
2. Run the mapper to synthesize your design using the `--flow map` flag. The software creates the *<project name>***.map.v** file in the **outflow** directory, which you use for post-synthesis simulation.
3. Perform post-map simulation using the top-level testbench and the **.map.v** file using the `--flow mapsim` flag.
4. After generating the interface constraints with the unified netlist option you can simulate the interface logic:

   a. Simulate the interface using the `--flow ptsimrtl` flag.
   b. Perform full-chip simulation using the `--flow ptsimfc` flag.

The following example shows the commands for these three steps:

Example: Simulating at the Command lIne

Linux:

```
efx_run.py <project name>.xml --flow rtlsim
efx_run.py <project name>.xml --flow map
efx_run.py <project name>.xml --flow mapsim
efx_run.py <project name>.xml --flow ptsimrtl
efx_run.py <project name>.xml --flow ptsimfc
```

Windows:

```
efx_run.bat <project name>.xml --flow rtlsim
efx_run.bat <project name>.xml --flow map
efx_run.bat <project name>.xml --flow mapsim
efx_run.bat <project name>.xml --flow ptsimrtl
efx_run.bat <project name>.xml --flow ptsimfc
```

The software saves simulation results into the **outflow** directory.

Simulation Models

The Efinix core primitive models are located in the directory *<installation directory>*/
sim_models/verilog.

Table 16: Core Primitive Simulation Models

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_ADD	Simple Full Adder	✓	✓	✓	efx_add.v
EFX_COMB4	Simple 4-Input LUT ROM plus Simple Adder		✓	✓	efx_comb4.v
EFX_DPRAM5K	5 Kbit True-Dual-Port RAM Block	✓			efx_dpram5k.v
EFX_DPRAM10	10 Kbit True-Dual-Port RAM Block		✓	✓	efx_dpram10.v
EFX_DSP12	Quad-Mode 4 x 4 DSP Block		✓	✓	efx_dsp12.v
EFX_DSP24	Dual-Mode 8 x 8 DSP Block		✓	✓	efx_dsp24.v
EFX_DSP48	Full Function DSP Block		✓	✓	efx_dsp48.v
EFX_FF	D Flip-flop with Clock Enable and Set/Reset Pin	✓	✓	✓	efx_ff.v
EFX_GBUFCE	Global Clock Buffer	✓	✓	✓	efx_gbufce.v
EFX_LUT	Simple 4-Input LUT ROM	✓	✓	✓	efx_lut4.v
EFX_MUL	18 x 18 Multiplier	✓			efx_mult.v
EFX_RAM_5K	5 Kbit RAM Block	✓			efx_ram_5k.v
EFX_RAM10	10 Kbit RAM Block		✓	✓	efx_ram10.v
EFX_SRL8	8-Bit Shift Register		✓	✓	efx_srl8.v

The Efinix interface primitive models are located in the directory *<installation directory>*/
pt/sim_models/verilog

Table 17: Interface Primitive Simulation Models

Although additionally model files are located in the **/pt/sim_models/verilog** directory, only the ones listed in this table are supported in v2024.2

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_CLKOUT	Clock Output Buffer	✓	✓	✓	EFX_CLKOUT.v
EFX_FPLL_V1	Fractional PLL		✓	✓	EFX_FPLL_V1.v
EFX_GPIO_V1	Basic GPIO	✓			EFX_GPIO_V1.v
EFX_GPIO_V2	GPIO with Double Data I/O Function	✓			EFX_GPIO_V2.v
EFX_GPIO_V3	HVIO and HSIO Used as GPIO		✓	✓	EFX_GPIO_V3.v
EFX_IBUF	Single-Ended Input Buffer	✓	✓	✓	EFX_IBUF.v
EFX_IDDIO	Input Double Data I/O Register	✓	✓	✓	EFX_IDDIO.v
EFX_IOREG	Single-Ended Bi-Directional Register	✓	✓	✓	EFX_IOREG.v
EFX_IO_BUF	Single-Ended Bi-Directional Buffer	✓	✓	✓	EFX_IO_BUF.v
EFX_IREG	Single-Ended Input Register	✓	✓	✓	EFX_IREG.v

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_JTAG_CTRL	JTAG Interface	✓	✓	✓	EFX_JTAG_CTRL.v
EFX_JTAG_V1	JTAG User TAP Interface	✓	✓	✓	EFX_JTAG_V1.v
EFX_OBUF	Single-Ended Output Buffer	✓	✓	✓	EFX_OBUF.v
EFX_ODDIO	Output Double Data I/O Register	✓	✓	✓	EFX_ODDIO.v
EFX_OREG	Single-Ended Output Register	✓	✓	✓	EFX_OREG.v
EFX_OSC_V1	Oscillator	✓			EFX_OSC_V1.v
EFX_OSC_V3	Oscillator		✓	✓	EFX_OSC_V3.v
EFX_PLL_V1	Simple PLL	✓			EFX_PLL_V1.v
EFX_PLL_V2	Advanced PLL	✓			EFX_PLL_V2.v
EFX_PLL_V3	Full-Featured PLL		✓	✓	EFX_PLL_V3.v

Changing the Default Testbench Names

The simulation flow assumes that:

- Your testbench file is named *<project name>*_**tb.v**
- The top module in your testbench is named **sim**

To use a different testbench name, use the `--tb` option.

To use a different name for the top-level module, specify it with the `--tb_top` option.

Example: Changing Default Names

Linux:

```
efx_run.py <project name>.xml --flow rtlsim|mapsim --tb <testbench name>
efx_run.py <project name>.xml --flow rtlsim|mapsim --tb_top <top-level module name>
```

Windows:

```
efx_run.bat <project name>.xml --flow rtlsim|mapsim --tb <testbench name>
efx_run.bat <project name>.xml --flow rtlsim|mapsim --tb_top <top-level module name>
```

(i) **Note:** If the testbench file is not located at the root level of the project directory, you need to specify the path. For example:

```
efx_run.py helloworld.xml --flow rtlsim --tb src\helloworld_tb.v
```

Simulate with the iVerilog Simulator

By default, the Efinity® software calls the iVerilog simulator. Use the `--flow rtlsim|mapsim` flag.

(i) **Note:** You can download the free Icarus Verilog (iVerilog) simulator from **iverilog.icarus.com**.

(i) **Note:** *Windows:* You may need to add the path to iVerilog (`$iVerilog_folder$\bin\`) to your System Variables path for the software to launch correctly.

For example, the commands to simulate are:

Example: Simulate with iVerilog

Linux:

```
> efx_run.py <project name>.xml --flow rtlsim    // Behavioral simulation
> efx_run.py <project name>.xml --flow map       // Synthesize the design
> efx_run.py <project name>.xml --flow mapsim     // Post-synthesis simulation
```

Windows:

```
> efx_run.bat <project name>.xml --flow rtlsim    // Behavioral simulation
> efx_run.bat <project name>.xml --flow map       // Synthesize the design
> efx_run.bat <project name>.xml --flow mapsim     // Post-synthesis simulation
```

The simulator responds with
- PASS if the simulation is successful.

- a Python exception warning if the simulation is unsuccessful.

The software saves simulation results (<*project name*>**.rtl.simlog** and <*project name*>**.map.simlog**) and error messages (<*project name*>**.log**) in your project's **outflow** directory.

View Waveforms

To use GTKWave to view a waveform:

1. Add the following lines to your testbench to generate the dumpfiles:

    ```
    $dumpfile("outflow/<file name>.vcd");
    $dumpvars(0, sim);
    ```

2. Simulate with the iVerilog simulator.
3. Use this command to view the output waveform:

    ```
    gtkwave outflow/<project name>.vcd
    ```

Simulate with the ModelSim Simulator

By default, the Efinity® software calls the iVerilog simulator. Use the `--modelsim` option to target the ModelSim simulator instead.

ⓘ **Note:** The simulator must be in your path for the simulation to run properly.

For example, the commands to simulate are:

Example: Simulate with ModelSim

Linux:

```
> efx_run.py <project name>.xml --flow rtlsim --modelsim      // Behavioral simulation
> efx_run.py <project name>.xml --flow map                    // Synthesize the design
> efx_run.py <project name>.xml --flow mapsim --modelsim      // Post-synthesis simulation
```

Windows:

```
> efx_run.bat <project name>.xml --flow rtlsim --modelsim     // Behavioral simulation
> efx_run.bat <project name>.xml --flow map                   // Synthesize the design
> efx_run.bat <project name>.xml --flow mapsim --modelsim     // Post-synthesis simulation
```

The simulator responds with
- PASS if the simulation is successful.
- FAIL if the simulation is unsuccessful.

The software saves simulation results (<*project name*>**.rtl.simlog** and <*project name*>**.map.simlog**) and error messages (<*project name*>**.log**) in your project's **outflow** directory.

Simulate with the ModelSim GUI

The ModelSim GUI uses a macro file of your simulation files and workspace for simulation.

1. Create a new macro file <*project name*>**.do** in your project directory.
2. For behavioral simulation, define your workspace and include your source code.
3. For post-synthesis simulation, define your workspace, include the post-mapping synthesis file, and include the simulation models for Efinix primitives.

4. Add the `vsim -t ps <work space>.<test bench module name>` command to start simulation in the ps timeframe.

5. Add the `run <number>us` command to generate a waveform up to *<number>* µs.

6. Run the ModelSim software in the Transcript console.

7. Change to the project root directory.

8. Use the `do` command to execute the macro (`do <name>.do`).

9. Add signals to the waveform in the **Objects** tab.

10. View the simulation result in the **Wave** tab.

The following examples show the macro files for behavioral and post-synthesis simulation for the **helloworld** design provided with the Efinity® software.

Figure 29: Bevhavioral Simulation Example .do Macro

```
vlib work
vmap work work

vlog "helloworld.v"
vlog "led.v"
vlog "reset.v"
vlog "helloworld_tb.v"

vsim -t ps work.sim
run 1us
```

Figure 30: Post-Synthesis Simulation Example .do Macro

The Efinity software provides additional primitives, but they are not used for simulation.

```
vlib work
vmap work work

vlog "outflow/helloworld.map.v"

vlog "<path to Efinity>/sim_models/verilog/efx_add.v"
vlog "<path to Efinity>/sim_models/verilog/efx_dpram_5k.v"
vlog "<path to Efinity>/sim_models/verilog/efx_ff.v"
vlog "<path to Efinity>/sim_models/verilog/efx_gbufce.v"
vlog "<path to Efinity>/sim_models/verilog/efx_lut4.v"
vlog "<path to Efinity>/sim_models/verilog/efx_mult.v"
vlog "<path to Efinity>/sim_models/verilog/efx_ram_5k.v"

vlog "helloworld_tb.v"

vsim -t ps work.sim
run 1us
```

Simulate with the NCSim Simulator

By default, the Efinity® software calls the iVerilog simulator. Use the `--ncsim` option to target the NCSim simulator instead.

Note: The simulator must be in your path for the simulation to run properly.

For example, the commands to simulate are:

Example: Simulate with NCSim

Linux:

```
> efx_run.py <project name>.xml --flow rtlsim --ncsim     // Behavioral simulation
> efx_run.py <project name>.xml --flow map                // Synthesize the design
> efx_run.py <project name>.xml --flow mapsim --ncsim     // Post-synthesis simulation
```

Windows:

```
> efx_run.bat <project name>.xml --flow rtlsim --ncsim     // Behavioral simulation
> efx_run.bat <project name>.xml --flow map                // Synthesize the design
> efx_run.bat <project name>.xml --flow mapsim --ncsim     // Post-synthesis simulation
```

The simulator responds with
- PASS if the simulation is successful.
- FAIL if the simulation is unsuccessful.

The software saves simulation results (*<project name>*.**rtl.simlog** and *<project name>*.**map.simlog**) and error messages (*<project name>*.**log**) in your project's **outflow** directory.

Debugging

Contents:

- **Profile Editor Perspective**
- **Debug Wizard**
- **Debug Perspective**
- **Debugger Options**
- **Concurrent Debugging**
- **Resource Usage**
- **Disable the Debug Core**

The Efinity® software includes a hardware Debugger to probe signals in your FPGA design via the JTAG interface. The Debugger has two perspectives: *Profile Editor* and *Debug*. The Profile Editor perspective is where you add debug cores manually. You can also view the settings of a Logic Analyzer core that you created with the **Debug Wizard**. The Debug perspective is where you perform debugging.

The Debugger includes two debug cores, **Virtual I/O (`vio`)** and a **Logic Analyzer (`la`)**. You use a manual flow and the Profile Editor to configure Virtual I/O cores. You can use a manual flow or the Debug Wizard's automated flow to configure Logic Analyzer cores.

Debugging involves the following general steps:

1. Create a debug profile with the Virtual I/O and/or Logic Analyzer debugger core(s).
2. Generate the debug design file and add it to your project.
3. Compile.
4. Program the FPGA.
5. Run the Debugger GUI and observe the values on the probes.

ⓘ **Note:** The minimum operating frequency of the debug cores is 2 times the JTAG TCK frequency.

Profile Editor Perspective

Choose **Perspectives > Profile Editor** to open the editor. If you created a debug profile using the Debug Wizard, the editor loads it automatically. You can import an existing profile; if you do not have an existing debug profile, you add Virtual I/O or Logic Analyzer cores and then configure them.

Figure 31: Debugger Profile Editor Perspective

Icon legend:

- Add Debug Core
- Add Probe
- Remove Probe or Source
- Import Profile
- Add Source
- Remove Debug Core
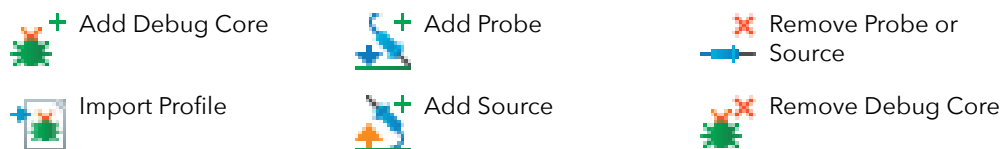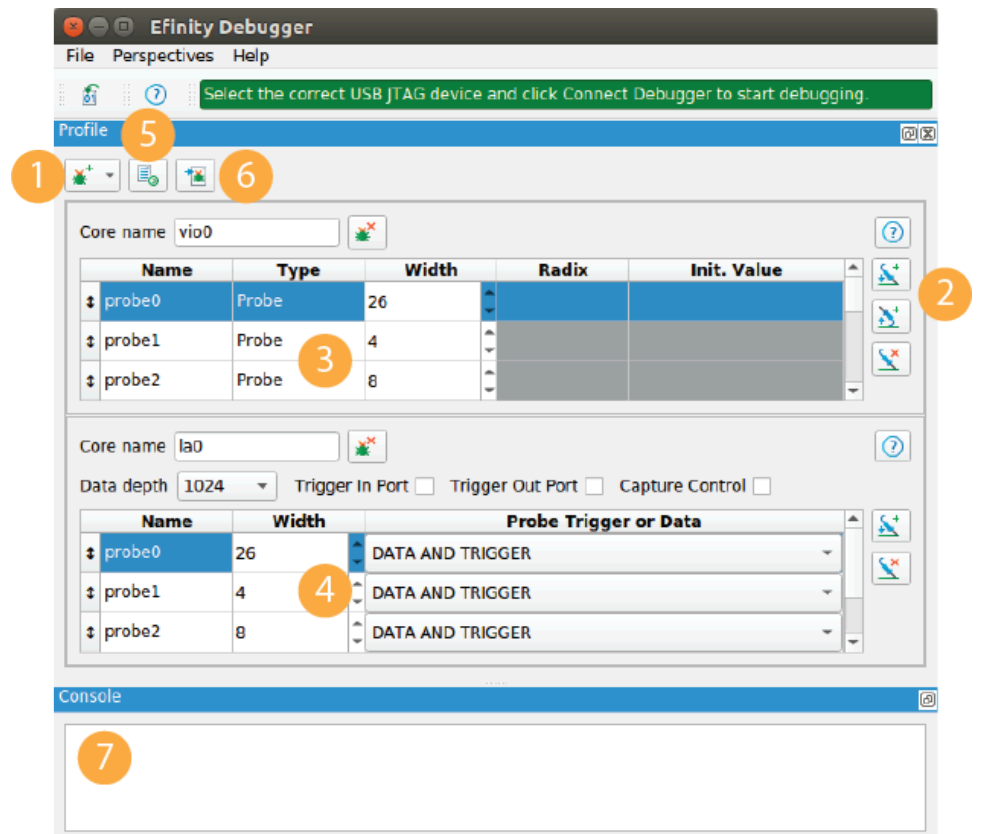
1. Click **Add Debug Core** to add a Logic Analyzer (`la`) or Virtual I/O (`vio`) core manually. You can also use the Debug Wizard for Logic Analyzer cores.
2. For `vio`, add probes and sources; for `la`, add probes.
3. For `vio`, specify the signal name and width; for sources you can also specify a radix and initial value.
4. For `la`, specify the signal name, width, and whether the signal is for collecting data, triggering, or both.
5. Click **Generate Core RTL** to create the debug module and instantiation template.
6. If you created a debug profile with the Debug Wizard, click **Import Profile** to load it.
7. The Console displays messages.

Virtual I/O Debug Core

The Virtual I/O (`vio`) core lets you monitor and drive the FPGA signals using the Debugger. You can use it to capture instantaneous data from connected wires or registers, and you can edit values of connected wires or register. This debug core is useful for triggering reset or control signals in real time. For example, you could use the Virtual I/O core to trigger a reset instead of using a pushbutton; or, you can use it to monitor a data bus to ensure that the data is what you expect. You manually configure and instantiate the Virtual I/O core.

Functional Description

The Virtual I/O core has an interface to the JTAG User Tap block, a clock, and user-specified probes and sources.

Figure 32: Virtual I/O Core Block Diagram
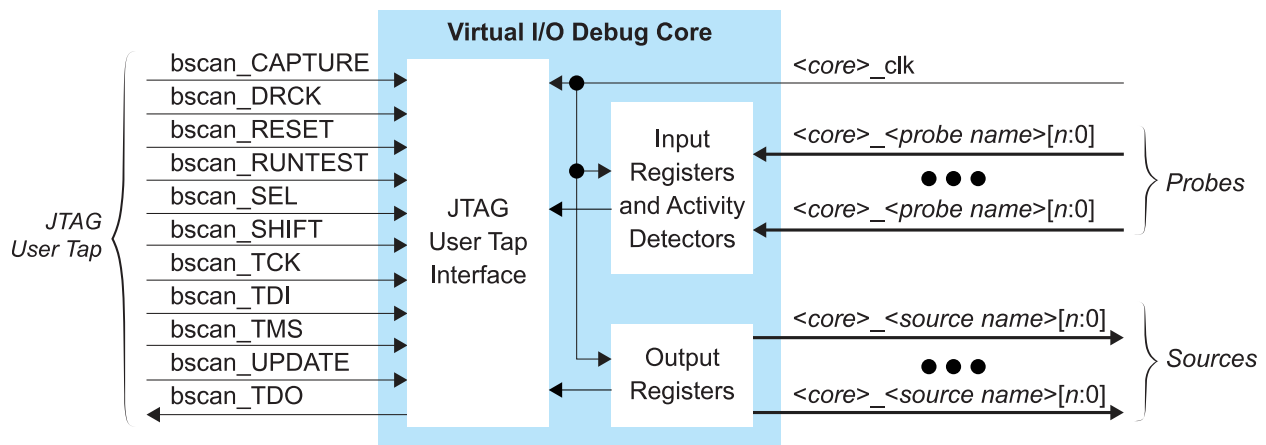
Table 18: Virtual I/O Core Ports

Port	Direction	Description
*<core>*_clk	Input	Clock to register input and output ports.
*<core>*_*<probe name>*[n:0]	Input	Probes you add in the Profile Editor. You can add a maximum on 64 probes; the maximum probe width is 256 bits.
*<core>*_*<source name>*[n:0]	Output	Sources you add in the Profile Editor. You can add a maximum on 64 sources; the maximum source width is 256 bits.
bscan_CAPTURE	Input	Capture output from the TAP controller.
bscan_DRCK	Input	Gated TCK output.
bscan_RESET	Input	Reset output for the TAP controller.
bscan_RUNTEST	Input	Output asserted when the TAP controller is in the Run Test / Idle state.
bscan_SEL	Input	USER instruction active output.
bscan_SHIFT	Input	SHIFT output from TAP controller.
bscan_TCK	Input	JTAG test clock input (TCK).
bscan_TDI	Input	JTAG test data input (TDI).
bscan_TMS	Input	JTAG test mode select input (TMS).
bscan_UPDATE	Input	UPDATE output from TAP controller.
bscan_TDO	Output	JTAG test data output (TDO).

Adding a Virtual I/O Core

1. Open the Debugger.
2. Choose **Perspectives > Profile Editor**.
3. Choose **Add Debug Core > VIO**.
4. Specify the core name.
5. Add sources (inputs to your design from the JTAG interface) and probes (outputs from your design to the JTAG interface).
   - For probes, choose a width and specify the signal to which you want to connect the probe in your design.
   - For sources, choose a width and specify the signal to which you want to connect the source in your design; you can set an initial value and choose a radix for how to display the data.
6. Click **Generate Core RTL**. The Efinity® software saves the debug profile in your project directory as **debug_profile.json**. The software also creates a debug template (**debug_TEMPLATE.v**), which includes the module for the debug profile you created and **debug_top.v**, which is the RTL logic for the debug core.
7. Add the **debug_top.v** file to your project.

Tip: In the **Project** pane, right-click **Design** and choose **Add** to open a dialog box to find the file and add it.

---

8. Add a JTAG User Tap block in the Interface Designer. Choose **JTAG_USER1** as the **JTAG Resource**.

   (i) **Note:** the debug template uses the default signal names prefixed with jtag_inst1. If you use a different name, then you should also change it in the module instantiation.

   ---

9. Add the debug logic into your design using these steps:
   a. Add all of the JTAG input and output pins to the project's top module. Refer to the JTAG User TAP block pin names in the Interfaces Design to get the pin list.
   b. Instantiate the debug core in the project's top module. You can copy the example code from the generated **debug_TEMPLATE.v** or **debug_TEMPLATE.vhd** file in the project folder.
   c. Connect the nets that you want to monitor and drive the FPGA signals. You need to map the net (input, output, wire, register, and/or signal) to the port of the instantiated debug core (edb_top_inst).
10. Compile the design.

When compilation completes, you can launch the Debugger to perform debugging.

Logic Analyzer Debug Core

You use the Logic Analyzer core (`la`) to monitor the signals in your design. You can capture connected wire or register values over a specified time period or after a specific number of times the trigger condition occurs (the default is 1). During runtime, the core samples the signals and saves the data into the FPGA's block RAM. You can specify the number of probes, the buffer depth, and the width for each probe input. Additionally, you can set global AND, OR, NAND, and NOR trigger conditions as well as segment triggers.

You add a Logic Analyzer core manually or using the Debug Wizard, compile your design, and program the FPGA. Then, you use the Debugger to set trigger events. When a trigger occurs, the core fills the sample buffer and loads the results into the Debugger's Debug Perspective. You can view this data using the GTK waveform viewer.

> **(i)** **Note:** Learn how to use the Debug Wizard >

Functional Description

The Logic Analyzer core has an interface to the JTAG User Tap block, a clock, user-specified probes and trigger-related signals.

Figure 33: Logic Analyzer Core Block Diagram
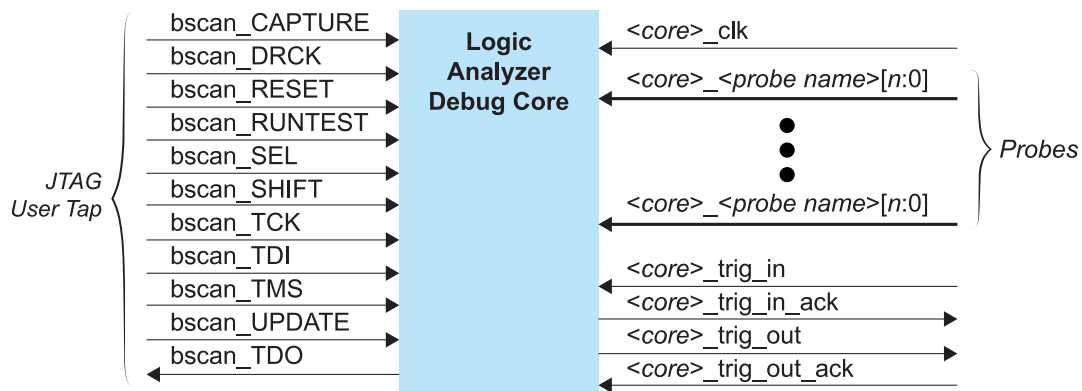
Table 19: Logic Analyzer Core Ports

Port	Direction	Description
*<core>*_clk	Input	Clock for triggers. At a minimum, this clock should run at twice the speed of the JTAG clock. The Debugger uses a JTAG clock of 3 MHz, so this clock should be 6 MHz or higher.
*<core>*_*<probe name>*[n:0]	Input	Probes you add in the Profile Editor. You can add a maximum on 64 probes; the maximum probe width is 256 bits.
*<core>*_trig_in	Input	Input trigger. You can connect this port to another Logic Analyzer core to build a cascading chain of triggers. Alternatively, you can connect it to an external source such as an oscilloscope.
*<core>*_trig_in_ack	Output	Input trigger acknowledge.
*<core>*_trig_out	Output	Output trigger. This trigger can be generated from an external trigger condition or from the *<core>*_trig_in port of another Logic Analyzer core.
*<core>*_trig_out_ack	Input	Output trigger acknowledge.
bscan_CAPTURE	Input	Capture output from the TAP controller.
bscan_DRCK	Input	Gated TCK output.

Port	Direction	Description
bscan_RESET	Input	Reset output for the TAP controller.
bscan_RUNTEST	Input	Output asserted when the TAP controller is in the Run Test / Idle state.
bscan_SEL	Input	USER instruction active output.
bscan_SHIFT	Input	SHIFT output from TAP controller.
bscan_TCK	Input	JTAG test clock input (TCK).
bscan_TDI	Input	JTAG test data input (TDI).
bscan_TMS	Input	JTAG test mode select input (TMS).
bscan_UPDATE	Input	UPDATE output from TAP controller.
bscan_TDO	Output	JTAG test data output (TDO).

Adding a Logic Analyzer Core Manually

1. Open the Debugger.
2. Choose **Perspectives > Profile Editor**.
3. Choose **Add Debug Core > Logic Analyzer**.
4. Specify the core name.
5. Select the data depth. This settings lets you control how much data is saved for the probes. The more data you save, the more on-chip memory is used.
6. Turn on Trigger In Port and/or Trigger Out Port to enable those signals.
7. Turn on Capture Control if you want to change the capture mode in the **Capture Setup** tab during debugging (see Debug Perspective on page 83 for details). If you turn this option on, the Logic Analyzer uses more FPGA resources.
8. Add probes (outputs from your design to the JTAG interface).

   a. Choose a width and specify the signal to which you want to connect the probe in your design.

   b. Choose **Data and Trigger** (default) to save data and can trigger when to capture.

   Choose **Data Only** to save data.

   Choose **Trigger Only** to trigger when to capture data. Tigger only signals do not display in the resulting waveform.

9. Click **Generate Core RTL**. The Efinity® software saves the debug profile in your project directory as **debug_profile.json**. The software also creates a debug template (**debug_TEMPLATE.v**), which includes the module for the debug profile you created and **debug_top.v**, which is the RTL logic for the debug core.
10. Add the **debug_top.v** file to your project.

Tip: In the **Project** pane, right-click **Design** and choose **Add** to open a dialog box to find the file and add it.

---

11. Add a JTAG User Tap block in the Interface Designer. You can choose either JTAG resource.

   (i) **Note:** the debug template uses the default signal names prefixed with `jtag_inst1`. If you use a different name, then you should also change it in the module instantiation.

   ---

12. Add the debug logic into your design using these steps:

   a. Add all of the JTAG input and output pins to the project's top module. Refer to the JTAG User TAP block pin names in the Interfaces Design to get the pin list.

    b. Instantiate the debug core in the project's top module. You can copy the example code from the generated **debug_TEMPLATE.v** or **debug_TEMPLATE.vhd** file in the project folder.

    c. Connect the nets that you want to monitor and drive the FPGA signals. You need to map the net (input, output, wire, register, and/or signal) to the port of the instantiated debug core (`edb_top_inst`).

13. Compile the design.

When compilation completes, you can launch the Debugger to perform debugging.

Note: For complex designs with multiple levels of hierarchy, it can be time-consuming to implement the Logic Analyzer core manually. Instead, use the Debug Wizard. **Learn about the Debug Wizard >**

Debug Wizard

The Debug Wizard provides an automated flow for adding a logic analyzer core to your design. You launch the wizard from the Efinity main icon bar. This wizard is helpful for complex projects with multiple levels of hierarchy. You select signals and nets from the post-map netlist and specify the probe type. Then, the wizard automatically creates a debug profile, adds the debug core to your project, connects the nets that you want to debug to the probe ports of the debug instance, and adds the JTAG User Tap block to your interface design. When the wizard completes its processing, you simply compile and start debugging.

Using the Wizard

1. Launch the Debug Wizard.

2. Choose the buffer depth. The buffer uses on-chip RAM, therefore, a larger buffer uses more RAM.

3. Optionally enable capture control. Enabling this option lets you change the capture mode in the **Capture Setup** tab during debugging (see **Debug Perspective** on page 83 for details). If you turn this option on, the logic analyzer uses more FPGA resources.

4. Select the JTAG User TAP (**USER1** or **USER2**) to connect to the Debugger in the **Connection Settings** box.

5. In the **Signals from** list, choose **Elaborated Netlist** to browse for signals in the pre-map netlist, or **Post-Map** to use signals from the post-map netlist.

6. Select signals and add them using the forward arrows. You can filter the signal list with regular expressions.

> **Note:** Signals with an **Undefined** clock domain are not driven by any clock in the post-map netlist. If you want to capture the waveform of a signal with an undefined clock domain, you need to manually add the Logic Analyzer core.

7. Specify the probe type (Data and Trigger, Data Only, or Trigger Only) for each signal.

8. Click **Next**. The wizard generates the core and hooks it up to your design.

9. Turn on **Enable Auto Instantiation** to have the wizard enable the logic analyzer in your project.

10. Click **Finish**. The Efinity® software saves the debug profile in your project directory as **debug_profile.wizard.json**. The software also creates a debug template (**debug_TEMPLATE.v**), which includes the module for the debug profile you created and **debug_top.v**, which is the RTL logic for the debug core.

Note: The wizard's automated flow requires the `JTAG_USER1` or `JTAG_USER2` resource. If you are using the block for the Debugger, you cannot use it for any other JTAG function; otherwise, you will receive an error during placement.

If you did not turn on **Enable Auto Instantiation**, you can manually enable the wizard-created debug profile:

1. Open the Project Editor.
2. Click the **Debugger** tab.
3. Select your project's **debug_profile.wizard.json** in the **Debug Profile** box.
4. Turn on **Debugger Auto Instantiation**.

Turn off **Debugger Auto Instantiation** in the **Debugger** tab to disable the debugger profile.

Debug Perspective

The Debug perspective is where you perform debugging. From this view, you can program the FPGA, set triggers, and open the GTKWave waveform viewer to see the results.

When you close the Debugger, it asks you if you want to save settings. Click **Yes** if you want to save values you have entered (such as trigger values, radix, window depth, etc.). This feature lets you open and close the Debugger without losing your work.

The Debugger provides basic error checking. When you program the FPGA, the Debugger checks to make sure that the bitstream you chose matches the FPGA you are trying to program. Additionally, the Debugger verifies that the debug profile in your Efinity project matches the debug core in the bitstream you are using. If the Debugger finds any mismatches, it gives an error message.

You can open multiple Debugger windows. Choose **Tools > Open Debugger** multiple times or click the Debugger icon multiple times to open additional windows. When you close the Efinity software, all Debugger window close as well.

Note: Download and install the GTKWave software from **gtkwave.sourceforge.net**. *Windows:* You may need to add the path to GTKWave (`$GTKWave_folder$\bin\`) to your System Variables path for the software to launch correctly.

Logic Analyzer Perspective

The following figure shows the Debug perspective for the Logic Analyzer.

Figure 34: Debug Perspective GUI - Logic Analyzer

To perform debugging using the Logic Analyzer:

1. Select the bitstream and program the FPGA.
2. Connect the Debugger.
3. Add triggers. If you turned on **Capture Control** in the Debug Wizard, you can adjust the capture mode in the **Capture Setup** tab. Set the **# Trigger** option if you want the trigger to occur after *n* occurances (default is 1).
4. Click **Run** to run the code. The Debugger waits for the trigger conditions you set and then captures data.

   Click **Run Immediate** to begin capturing data immediately.
5. The **Core status** and **Capture status** areas show the progress.
6. The Console shows messages.
7. Click the Select Waveform File button to choose a waveform file.

Understanding Capture Control

The Logic Analyzer core supports a capture control option. When you turn on this option in the debug profile, the **Capture Setup** tab becomes available during debugging.

The **Capture Mode** option selects which condition the Debugger evaluates before each sample is captured:

- **Always**—Stores a data sample during a given clock cycle regardless of any capture conditions you set.
- **Basic**—Only stores a data sample during a given clock cycle if the capture condition evaluates as true. Select this option to add nets and set capture conditions.

You can subdivide the capture data buffer into one or more segments. The Debugger automatically suggests a window depth depending on the number of segments you choose. Additionally, you can set the position of the trigger in the window.

Virtual I/O Perspective

The following figure shows the Virtual I/O Debug perspective.

Figure 35: Virtual I/O Debugger
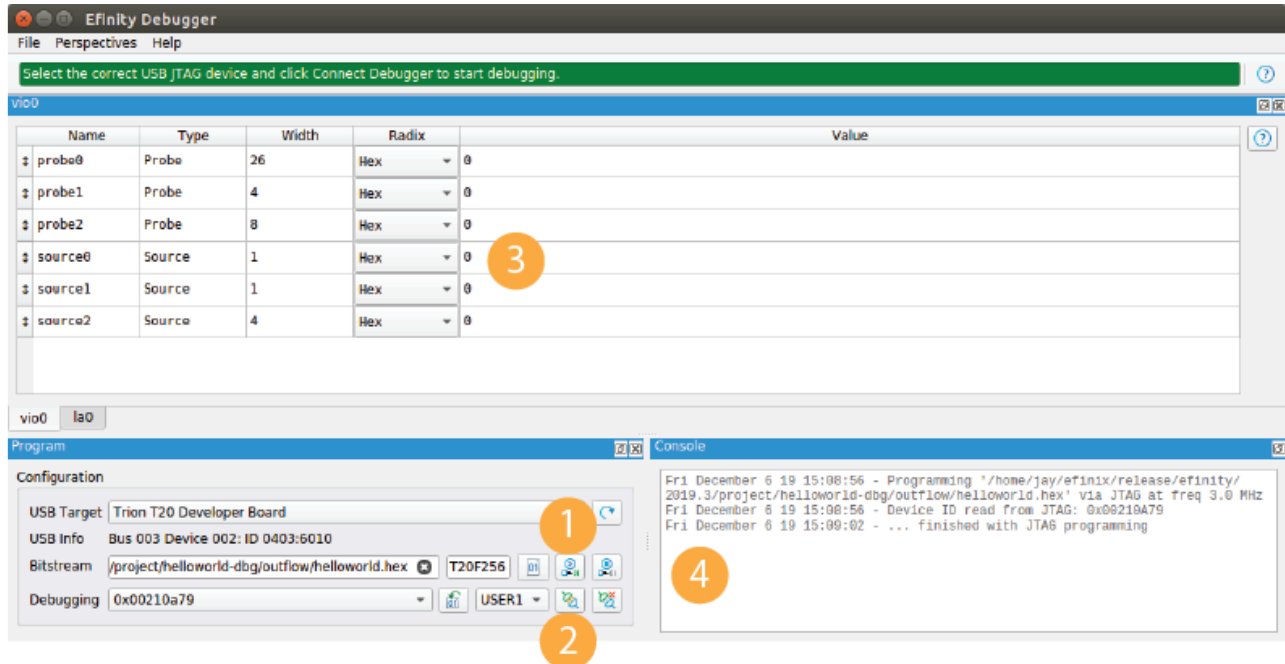
To perform debugging using Virtual I/O cores:

1. Select the bitstream and program the FPGA.
2. Connect the Debugger.
3. Enter values for the sources and observe the values for the probes.
4. The Console shows messages.

Debugger Options

The Debugger has these options, which you turn on or off in the **Options** menu:

Table 20: Debugger Options

Option	Description
Allow .bit/.hex for all configuration modes	With this option turned on, you can program the FPGA using a **.hex** file in JTAG mode. This option supports legacy behavior for Trion FPGAs.
Always launch new waveform viewer window	Turn this option on if you want the Debugger to open a new waveform window each time you launch a debug session. Existing windows remain open.

Concurrent Debugging

The Debugger has the concurrent debug feature where you can open multiple debug windows and connect to different JTAG USER TAP interfaces at the same time. This feature lets you perform debugging more easily. For example, you can set up a trigger in one Debugger window and then cause the event to happen in the second Debugger window.

The concurrent debug feature requires you to connect to the board using the Efinity Hardware Server. You can use the same computer for the server and client. Launch the server with the board connected to your computer and then connect to the server from the Debugger client using the same IP address.

See **Working with Remote Hardware** on page 133 for instructions on setting up a Hardware Server.

To open more than one Debugger window, choose **Tools > Open Debugger** or click the Debugger icon multiple times.

Resource Usage

In Efinity version 2020.1 and higher, you can view the resources used by the debug cores in the Dashboard's Results pane in the Debugger table. The software reports:
- Whether auto-instantiation is turned on or off
- Whether the debug target is the elaborated or post-map netlist
- Number of flipflops used
- Number of adders used
- Number of LUTs used
- Number of memory blocks used

Note: The resoure usage is an estimation, and is meant to give you a general guideline about the usage for reference purposes.

Disable the Debug Core

If you want to remove a debug core from your project:

1. Open the Project Editor.
2. Click the **Debugger** tab.
3. Turn off the **Debugger Auto Instantiation** option.
4. Click **OK**.
5. Re-compile the design.

The software removes the debug profile from your design, but does not remove it from disk. So you can re-enable the debug profile again by turning on the **Debugger Auto Instantiation**, specifying the profile name, and recompiling.

Chapter 9

Debugging Transceivers (Beta)

Contents:

- **Launching the Transceiver Debugger**
- **Using the Transceiver Debugger**
- **Transceiver Debugger Commands**
- **Interpreting the Results**

The Efinity software v2024.1 and higher includes the Efinity Transceiver Debugger tool that tests and displays the signal quality of the transceiver signals; it does not test the actual data itself. This tool is **not** an oscilloscope.

⚠ **Important:** This tool is Beta in v2024.1, and has limited features. For example, it can only access channel 0 for the eye diagram plot.

The left side of the Transceiver Debugger has programming options and buttons. Its functionality is similar to the Programmer, only simplified. A Console displays messages such as the device ID, board profile, and device status. It also shows any error messages.

The right side of the window has an Eye Diagram Plot. The **Status** tab shows the PHY and link status; the **Configuration** tab has options to customize the plot.

Figure 36: Efinity Transceiver Debugger
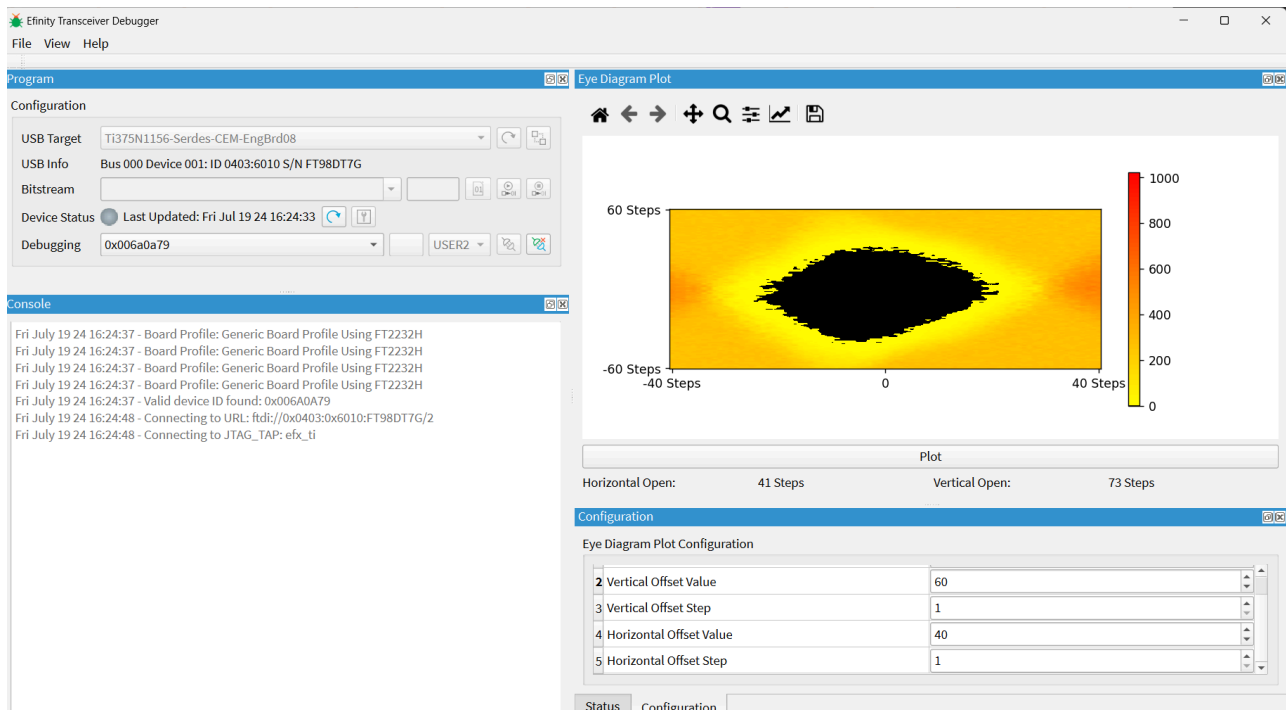
Launching the Transceiver Debugger

In the Efinity software v2024.1 you launch the Transceiver Debugger using a batch file (Windows) or shell script (Linux).

Windows—Execute the file *<Efinity version>***debugger\serdes_debug_tool\bin\efinity_serdes_dbg.bat**.

Linux—Execute the file *<Efinity version>*/**debugger/serdes_debug_tool/bin/efinity_serdes_dbg.sh**.

Using the Transceiver Debugger

This topic assumes you already know how to program an FPGA using the Efinity Programmer. To use the Transceiver Debugger:

1. Connect an Efinix transceiver-capable board, e.g., a board with the Ti375 N1156 FPGA. If you are using the Ti375 PCIe Early Access Board, it comes pre-loaded with an example design.

2. Click Refresh USB Target if the **USB Target** field does not display the board name.

3. If the FPGA is not programmed already, select a bitstream file and program it as you normally would.

4. Select **JTAG > USER2** and connect the Transceiver Debugger.

   > **(i)** **Note:** Disconnect all other debug cores before connecting the Transceiver Debugger.

5. Review the **PCIE Status** table in the **Status** tab. This table shows the link status and speed.

6. Review the **PHY Status** table in the **Status** tab. This table show whether the lanes are locked and ready.

7. In the **Configuration** tab, adjust the settings for the eye diagram plot.

Setting	Description
Vertical Offset Value	Indicates how many vertical positions there are to the right and left of center. Higher values result in a plot with high resolution, at the expense of longer drawing time. Default: 40
Vertical Offset Step	Indicates how many vertical positions to include in the same color pixel. For example, for an offset value of 40 and a step of 2, the software plots a total of 20 blocks on each side of the center line. A higher number results in less drawing time. Default: 2
Horizontal Offset Value	Indicates how many horizontal positions there are to the right and left of center. Higher values result in a plot with high resolution, at the expense of longer drawing time. Default: 40
Horizontal Offset Step	Indicates how many horizontal positions to include in the same color pixel. For example, for an offset value of 40 and a step of 2, the software plots a total of 20 blocks on each side of the center line. A higher number results in less drawing time. Default: 2

Setting	Description
Accumulation Period	The accumulation period is how long to perform the test for each pixel. A higher value provides more saturated results. Each period in the nanoseconds range.

8. When you are finished setting values, click **Plot**. The tool begins sampling data and displaying the results on the plot.

Transceiver Debugger Commands

The following table outlines some of the commands you can use in the Efinity Transceiver Debugger tool.

Table 21: Transceiver Debugger Commands

Command	Example	Comment
write:*<24-bit hex register>,<32-bit hex value>*	write:0x2041A0,0x00003800	-
read:*<24-bit hex register>*	read:0x2041A0	Return value is 32 bits.
sleep:*<seconds>*	sleep:5	Only available in Mass Mode.

For a list of transceiver registers, see:

- **Titanium PCIe® Controller Registers User Guide**
- **Titanium Ethernet 10GBase-KR User Guide** ("Register Map")
- **Titanium PMA Direct User Guide** ("Register Map")

Interpreting the Results

The **PCIE Status** table shows information about the link. You can review the data to see whether the link is operating at the correct speed with the correct number of lanes.

The eye plot gives a visual representation of the link quality. Areas that are black have no errors reported. The yellow, orange, and red colors indicate how many errors are being found (yellow is less while red is more).

You can use the plot to see if the error-free area (or eye) is sufficient for your application. For example, for a PCIe Gen4x4 interface, the unit interval (UI) for PCIe Gen4 is 62.5 picoseconds (refer to the PCI-SIG's PCI Express® Base Specification for more information). In the PCIe Gen4x4 plot:

- One horizontal step is approximately three millivolts.
- One unit interval is 64 steps. A UI is a full window (between the X of the eye diagram).
- A step, therefore, is close to 1 picosecond.

Chapter 10

Configuring an FPGA

Contents:

When you have finished running your design through the flow, you are ready to configure a device. You configure devices using the standalone GUI or command-line Programmer tool and a USB cable attached to your board. You can download the bitstream file into the device itself or into flash memory. Before you begin configuration, install the USB drivers for the programming cable (see **Appendix: Installing USB Drivers** on page 135).

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 22: 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 23: 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 for more information.

Figure 37: Flash Programming Board Setup

SPI Active with SPI Header

SPI Active using JTAG Bridge (Legacy/New) with JTAG Header
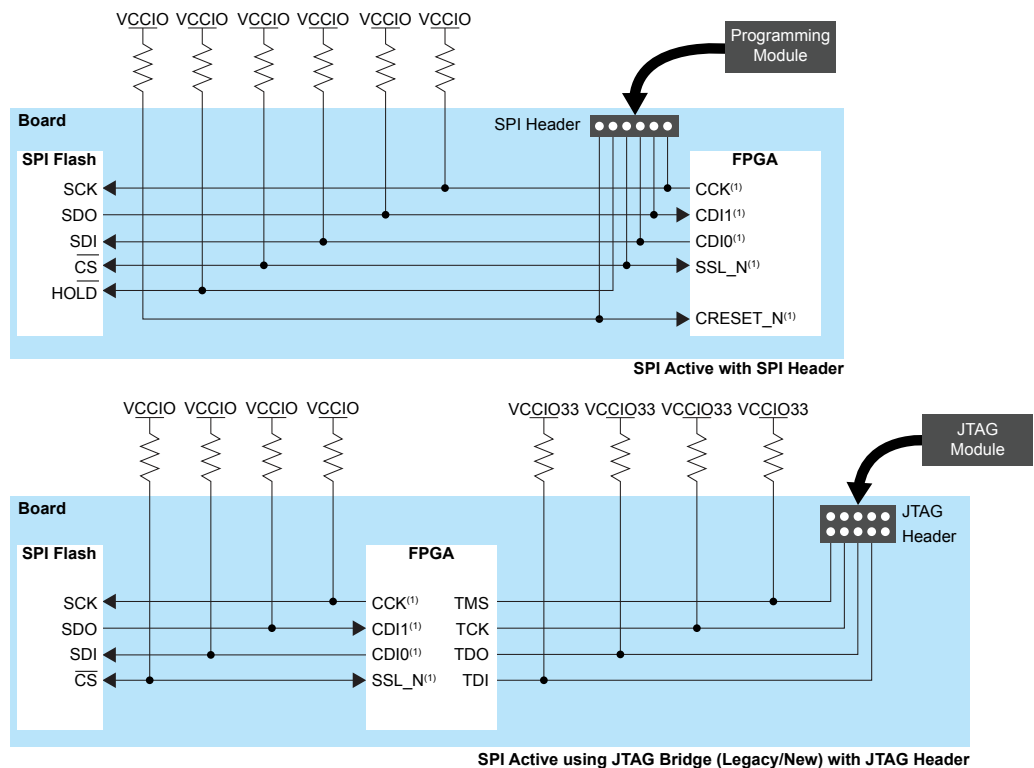
¹ The external pull-up is optional unless required by an external load.
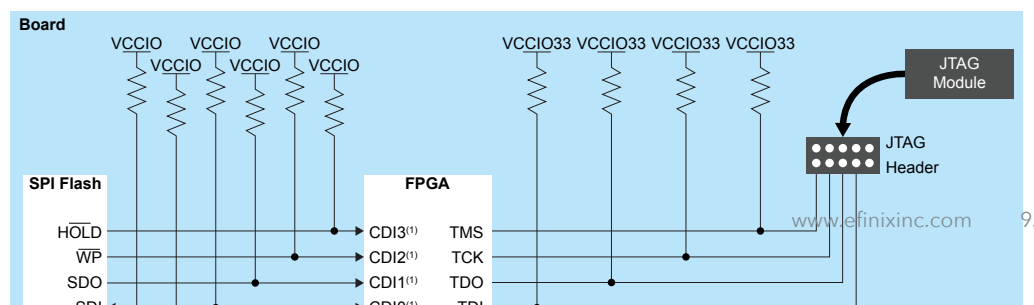
About the Programmer GUI

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

Figure 38: Programmer
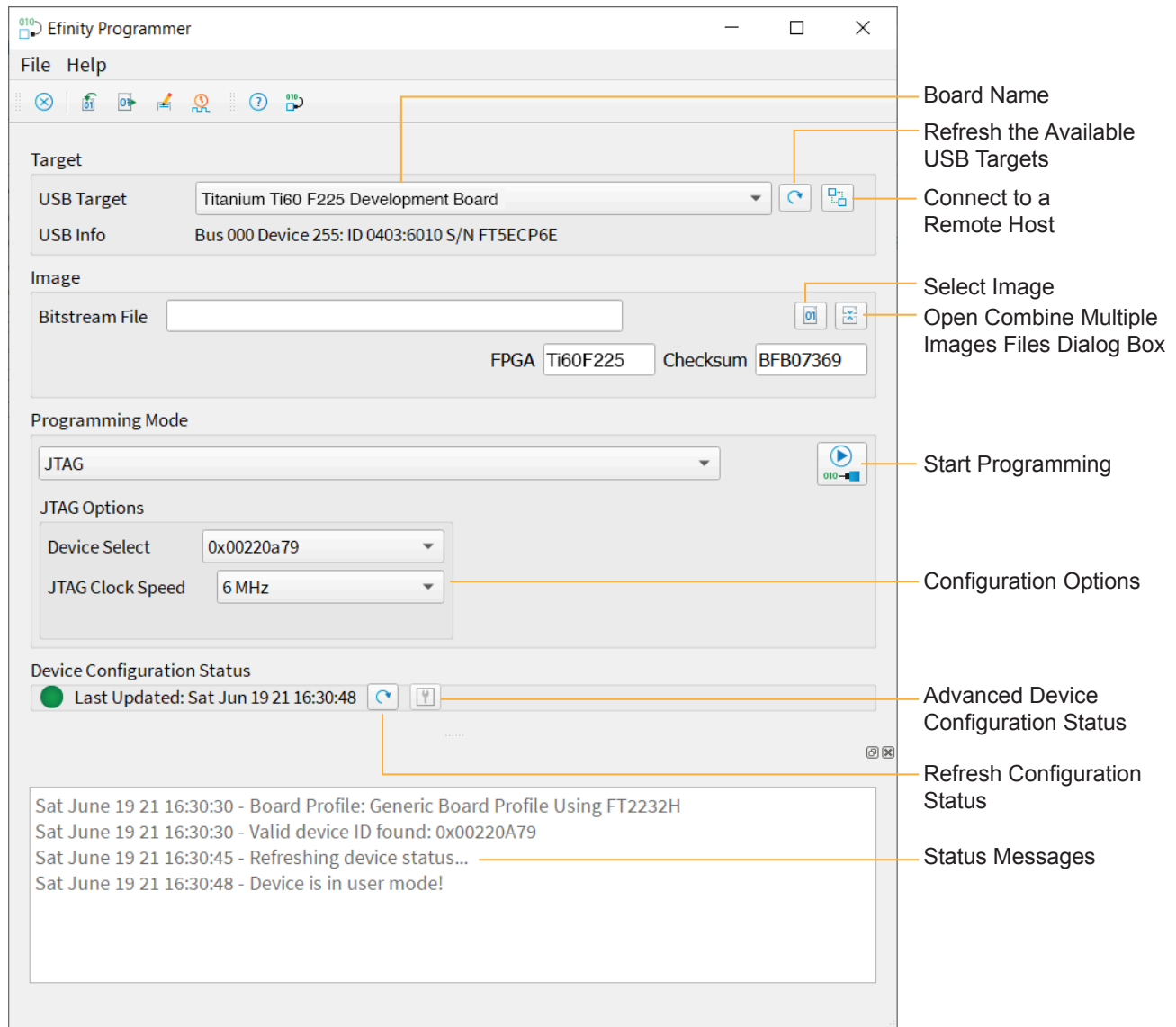
To use the Programmer:

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 133.
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)	Starting Flash Address
SPI Active using JTAG Bridge (New)	Flash Length
SPI Active x8 using JTAG Bridge (Legacy)	Erase Before Programming
SPI Active x8 using JTAG Bridge (New)	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 118 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**.

Edit the SPI Active Clock

An internal oscillator generates the internal clocks the FPGA uses during configuration. In SPI active configuration mode, configuration starts operating at the default frequency (10 MHz) and then switches to the user-selected clock to minimize configuration time (assuming the SPI flash device supports the faster f_{MAX}).

You set the configuration clock frequency in the Efinity® software.

Table 24: Internal Oscillator Clock Settings

SPI Clock Divider	Frequency (MHz)
DIV4	20
DIV8	10

To change the clock frequency:

1. Choose **File > Edit SPI Active Clock** or click the toolbar icon to open the **Edit SPI Active Clock Settings** dialog box.
2. Choose the divider value with **DIV Select**.
3. Click **Apply** and **Close** to save your changes.

You can also set the clock frequency for the project in the **Project Editor > Bitstream Generation** tab. Any setting you make in the **Edit SPI Active Clock Settings** dialog box overrides what you set for the project.

Note: T20 (Q144, F324, F400 packages) and T35 (all packages) support negative edge sampling. Click **Enabled** to turn it on. Then, specify the number of extra clock cycles to insert between the time when the default clock changes to the specified clock and when the FPGA continues configuration. You can add up to 7 extra clock cycles.

Generate a Bitstream (Programming) File

When you run the automated flow, the software automatically generates bitstream files that you can use to configure your target device. You can also generate the bitstream files manually. To generate bitstream files from the command line, use the following command:

Example: Generate a Bitstream File from the Command Line

Linux:

```
> efx_run.py <project name>.xml --flow pgm
```

Windows:

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

The software generates these files in the **outflow** directory:
- **.hex** file as *<project name>*.**hex**. Use this file to program in SPI active or passive mode.
- **.bit** file as *<project name>*.**bit**. Use this file for JTAG programming.

! **Important:** With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG.

The bitstream file includes programming options you set for your project (e.g., to initialize user memory or set configuration mode). If you change these options you must regenerate the bitstream file. See **Project-Based Programming Options** on page 115.

i **Note:** The software does not generate bitstream files for preliminary devices.

About the BRAM Initial Content Updater

The BRAM Initial Content Updater is a tool that lets you quickly update the initial memory saved in the FPGA's BRAM without performing a full compile. For example, you can use this tool if you want to:

- Update RISC-V application code in the on-chip memory
- Update sensor parameters in on-chip memory

To open the BRAM Initial Content Updater, click its icon in the toolbar or choose **Tools > Open BRAM Initial Content Updater**.

Figure 39: Using the Netlist Pane
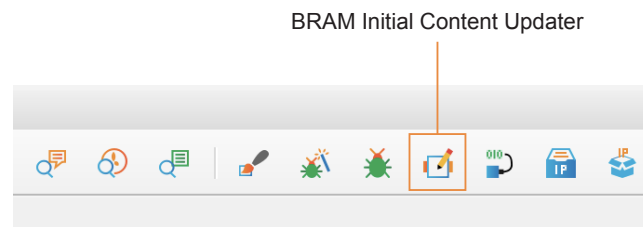
In the tool, you select the elaborated netlist memory that you want to update, not the post-map memory. Therefore, you do not need to know how synthesis decomposes and maps the memories to use this tool. Because this tool bypasses the full compilation flow, it does not update *<project>*.**map.v** and other intermediate compiler output files. As a result, the updated bitstream file will be out of sync with your other project files.

The format of the initial memory files is the same format used by Verilog HDL parsers and and the `$readmemh` or `$readmemb` Verilog HDL functions.

> **ⓘ**
> **Note:** The `--optimize-zero-init-rom` synthesis option tells the software to optimize away zero value ROMs. If your design has zero-value ROMs and you plan to use the BRAM Initial Content Updater later, disable this synthesis option in your project settings.

Figure 40: BRAM Initial Content Updater
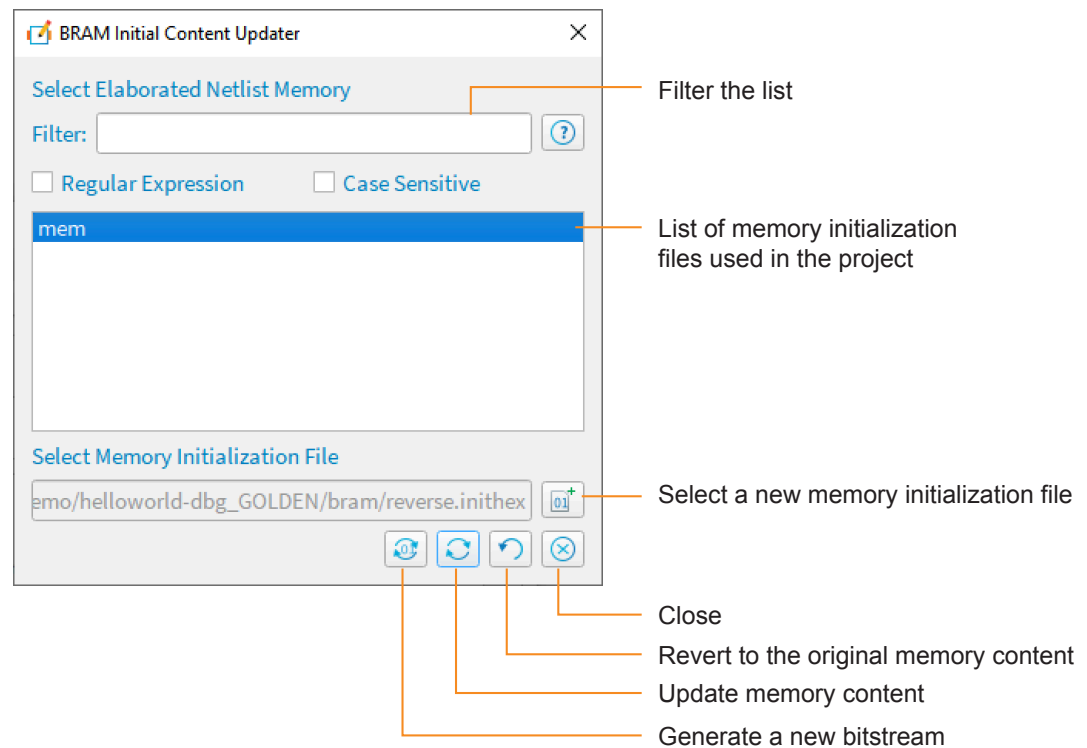
Updating the BRAM Initial Content

To use the tool:

1. Compile your project if you have not already done so. (The BRAM Initial Content Updater tool is disabled if you have not compiled.)
2. Create a new **.hex** file or update an existing one with the new memory content.
3. Open the tool by choosing **Tools > Bram Update** or by clicking the toolbar icon.
4. Select the memory you want to update. Use the filter options to narrow the list.
5. Click the Select Memory Initialization File button.
6. Choose the new **.hex** file.
7. Click **Open**.
8. Click the Update Memory Content button.
9. Click the Regenerate Bitstream button.
10. Close the BRAM Initial Content Updater.

Configure the FPGA with your updated bitstream.

If you want to revert the bitstream back to the original one:

1. Open the BRAM Initial Content Updater.
2. Click the Revert Memory Content Updates button.
3. Click the Regenerate Bitstream button.

Using the Example Files

If you have a Trion T20 BGA256 Development Board, you can use the example files provided with the Efinity software to experiment with the BRAM Initial Content Updater.

1. Connect the board to your computer.
2. Open the helloworld project in the *<Efinity install path>***/debugger/demo/ helloworld-dbg_GOLDEN** directory.
3. Compile the project.
4. Configure the FPGA using the Programmer, JTAG mode, and the **.bit** file located in the project's **outflow** directory.
5. Open the BRAM Initial Content Updater.
6. Click the memory named **mem** to select it.
7. Click the Select Memory Initialization File button.
8. Select the **reverse.inithex** file in the **helloworld-dbg_GOLDEN** directory.
9. Click **Open**.
10. Click the Update Memory Content button.
11. Click the Regenerate Bitstream button.
12. Close the BRAM Initial Content Updater.
13. In the Programmer, click Start Ptrogram (use the same bitstream file). When configuration completes, the LEDs on the board blink in the opposite direction, showing the changed initial memory state.

You can use the other files in the **helloworld-dbg_GOLDEN** directory to update the bitstream to show other blinking patterns. Additionally, you can create your own **.hex** file to change the blinking pattern.

Command-Line Interface

In addition to the GUI. you can rus the BRAM Initial Content Updater from the command line. With this method you can perform iterative work, without having to go through GUI for every iteration.

Usage:

```
efx_bram_update --project <project name> --memory <logical memory name>,<initialization file>
 <options>
```

Where:

Table 25: BRAM Initial Content Updater CLI Options

Option	Shorthand	Description	Type	Example
--help	-h	Show the help.	Optional	--help
--project arg	-j	Specify the Efinity project file.	Required	-j pt_demo.xml
--mem_info arg	-i	Specify the memory file in protocol buffer format.	Optional	-i new_mem.hex
--place arg	-p	Specify a placement file.	Optional	-p pt_demo.place
--lbf arg	-l	Specify the Logical Bit File (**.lbf**).	Optional	-l pt_demo.lbf
--output arg	-o	Specify the name for the updated bitstream file.	Optional	-o pt_demo.bit
--family arg	-f	Indicate the FPGA family, trion or titanium	Optional	-f trion
--verbose	-v	Print out verbose messages.	Optional	-v
--memory arg	-b	Specify the logical memory you want to update and the memory initialization file. Use the format <memory>,<init file>	Required	-b mem,new_mem.hex
--mode arg	-m	Indicate the mode for the update tool. update: Default. Use to updatie the memory with a new file. read: Reads the current initial content data in the bitstream and displays it in the console. revert: Go back to the original initial memory content.	Optional	-m revert

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 Topaz security features (AES-256 encryption and/or asymmetric authentication), the software cannot compress the bitstream. Therefore, compression is disabled when you use these feaatures.

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 102.

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**.

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 26: 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 103
	Internal reconfiguration	Up to 4	Program Multiple Images (Internal Reconfiguration) on page 104
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors	Program a Daisy Chain on page 105
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	Program Multiple Images (Bitstream and Data) on page 104

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 Remotely Update Trion® and Titanium FPGAs**.

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 27: Topaz JTAG Device IDs

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

Table 28: 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 29: 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

FPGA	Package	JTAG Device ID
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.

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 123 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 Trion®, Topaz, and Titanium 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 with the JTAG Flash Loader IP core if you prefer.

Refer to **Using the Efinity Bitstream Security Key Generator** on page 123 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.

> (i) **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`[5] (new mode, see **Program using a JTAG Bridge (New)** on page 109)
- `jtag_bridge`, or `jtag_bridge_x8`[6] (legacy mode, see **Program using a JTAG Bridge (Legacy)** on page 110)

> (i) **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

---

[5] The `jtag_bridge_x8_new` mode is only supported in some Titanium Topaz FPGAs. Refer to the data sheet for the modes your FPGA supports.
[6] 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`.
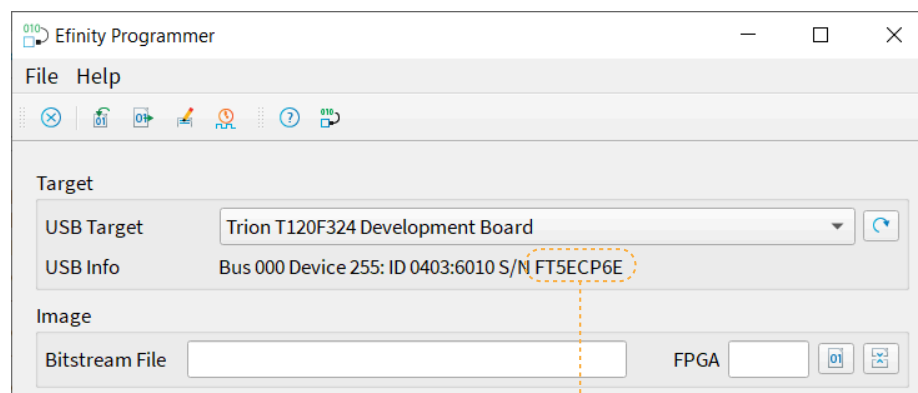
The string after S/N is
the FTDI serial number

<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)[7]

  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.

---

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

Project-Based Programming Options

You specify project-based programming options in the **Project Editor > Bitstream Generation** tab in the Efinity® software. Efinix FPGAs support active and passive configuration in a variety of modes.

Note: Some of these project settings affect bits in the bitstream. Therefore, when you program an FPGA with the Programmer, the setting you make in the Project Editor should match what you intend to use in the Programmer.

Table 30: Project-Specific Programming Options

Option	Notes
Active/Passive	Active: SPI active mode. Passive: SPI passive mode. Your choice of active or passive affects the pinout and determines which choices are available in the Programming Mode box.
JTAG USERCODE	Use this field to specify a 32-bit user electronic signature. The USERCODE is included in the bitstream. You can read it from the FPGA via the JTAG interface, and you can view the JTAG USERCODE in the Programmer's Advanced Device Status dialog box. Default:Fixed at: 0xFFFFFFFF
Clock Source	For Titanium Topaz FPGAs, choose whether you want to use the FPGA's internal oscillator or an external clock source as the configuration clock. For Trion FPGAs, this option is always **Internal Oscillator**.
SPI Programming Clock Divider	Choose the divider for the SPI clock. This setting is reflected in the bitstream file. Default: DIV8
Clock Sampling Edge	For Titanium Topaz FPGAs, choose whether the configuration clock should sample on the rising or falling edge. The default is **Rising**. For Trion FPGAs, this option is always **Rising**.
Power down flash after programming	Enable this option to power down the flash device after the FPGA finishes programming. This setting is reflected in the bitstream file, and you can only set it here. Default: On
Use 4-byte addressing during configuration	(Titanium Topaz only). When you turn this option on, the control block issues 4-byte addresses when it configures the FPGA. This option is not supported for all Trion Ti35 or Ti60Tz50 FPGAs.
Programming mode	Choose the programming mode and width; the choices depend on the FPGA and package you are targeting. This setting is reflected in the bitstream file, and you can only set it here. Default: SPI *<active or passive>* x1
Enable Initialized Memory in User RAMs	This setting is reflected in the bitstream file, and you can only set it here. **on**: The bitstream has initialized memory. **off**: The bitstream does not have initialized memory. **smart**: For the Trion family, this option has the same effect as **on**. For the Titanium Topaz family, this option gives a slightly smaller bitstream. Default: smart

Option	Notes
Release Tri-States before Reset	During configuration, core signals are held in reset and the I/O pins are tri-stated. These states are released when the FPGA enters user mode. On: (default) I/O pins are released from tri-state before the core is released from reset (use this option when the application is core sensitive). Off: Core signals are released from reset before the I/O pins are released from tri-state (use this option when the application is I/O sensitive).
Enable Bitstream Compression	(Titanium Topaz only) When turned on (default), the software compresses the bitstream. If you choose **Bitstream Encryption** or **Bitstream Authentication**, this option is turned off and disabled because you cannot compress a bitstream and use the security features simultaneously.
Bitstream Encryption	(Titanium Topaz only) On: The software generates an encrypted bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box. Off: (default) The software generates a plaintext bitstream.
Randomize IV value during compilation	(Titanium Topaz only) This option is used with bitstream encryption. The encryption/decryption uses a 96-bit initial vector (IV). If you want the software to generate a random IV for every compilation, leave this option turned on. If you want to specify an IV, turn this option off and specify the value in the **96-bit IV Value** box. On (default): Let the software generate the IV value. (The bitstream will be different every time you compile, even if nothing has changed in your design.) Off: The software does not generate the IV value; the user will specify it in the **96-bit IV Value** box. (If nothing has changed in your design, when you recompile, the bitstream remains the same)
96-bit IV Value	(Titanium Topaz only) Click the refresh button next to this box to generate a random IV value. You can also enter a value you generate with another program.
Bitstream Authentication	(Titanium Topaz only) On: The software generates a signed bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box and the RSA private key (**.pem**) file in the **RSA Private Key** box. Off: (default) The software generates an unsigned bitstream.
FPGA Key Data File	(Titanium Topaz only) Specify the location and name of the **.bin** file you generated with the Efinity Bitstream Security Key Generator.
RSA Private Key	(Titanium Topaz only) Specify the location and name of the RSA private key file (**.pem**).
Generate JTAG configuration file	On (always): Generate a **.bit** file for JTAG configuration.
Generate JTAG raw binary configuration file	On: Generate a **.bin** file (raw binary) for JTAG configuration. Off (default): Do not generate a **.bin** file.
Generate SPI configuration file	On (always): Generate a **.hex** file for SPI programming.
Generate SPI raw binary configuration file	On: Generate a **.bin** file (raw binary) for SPI programming. Off (default): Do not generate a **.bin** file.

When you change one of these options, you can simply re-run the bitstream generation flow step. You do not need to recompile the design.

Figure 41: Setting Programming Options (Trion)

Figure 42: Setting Programming Options (Titanium Topaz)

Notice: Refer to the data sheet for your FPGA for information on which configuration options it supports. Refer to **AN 006: Configuring Trion FPGAs** or **AN 033: Configuring Titanium FPGAs** for information on configuration modes, timing, and board considerations.

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.

Table 31: 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.
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.

Name	Description
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.
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**.

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 32: 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.

Figure 43: Bitstream Authentication
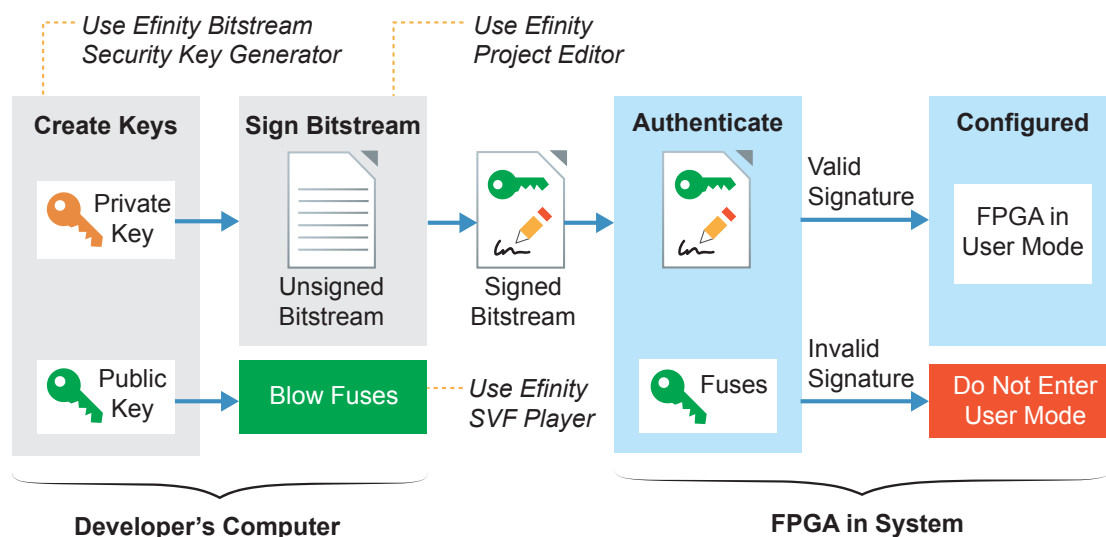
The public key is derived from the private key; the **.pem** is essentially a private/public key pair. The private key only exists in the **.pem**. The software uses it to sign the bitstream, but

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 44: Bitstream Encryption
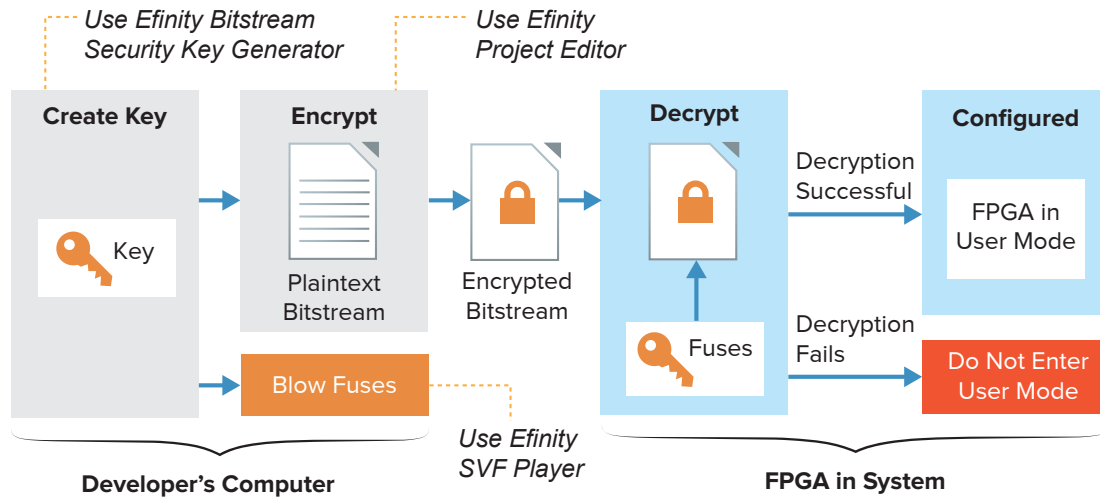
Figure 45: Disabling JTAG
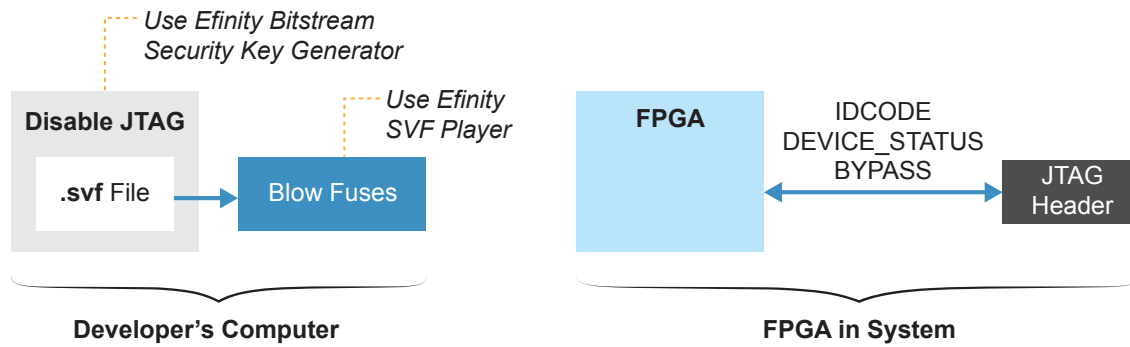
The following sections describe how to use each of these tools to enable security features.

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 128 for more information.

Figure 46: Efinity Bitstream Security Key Generator
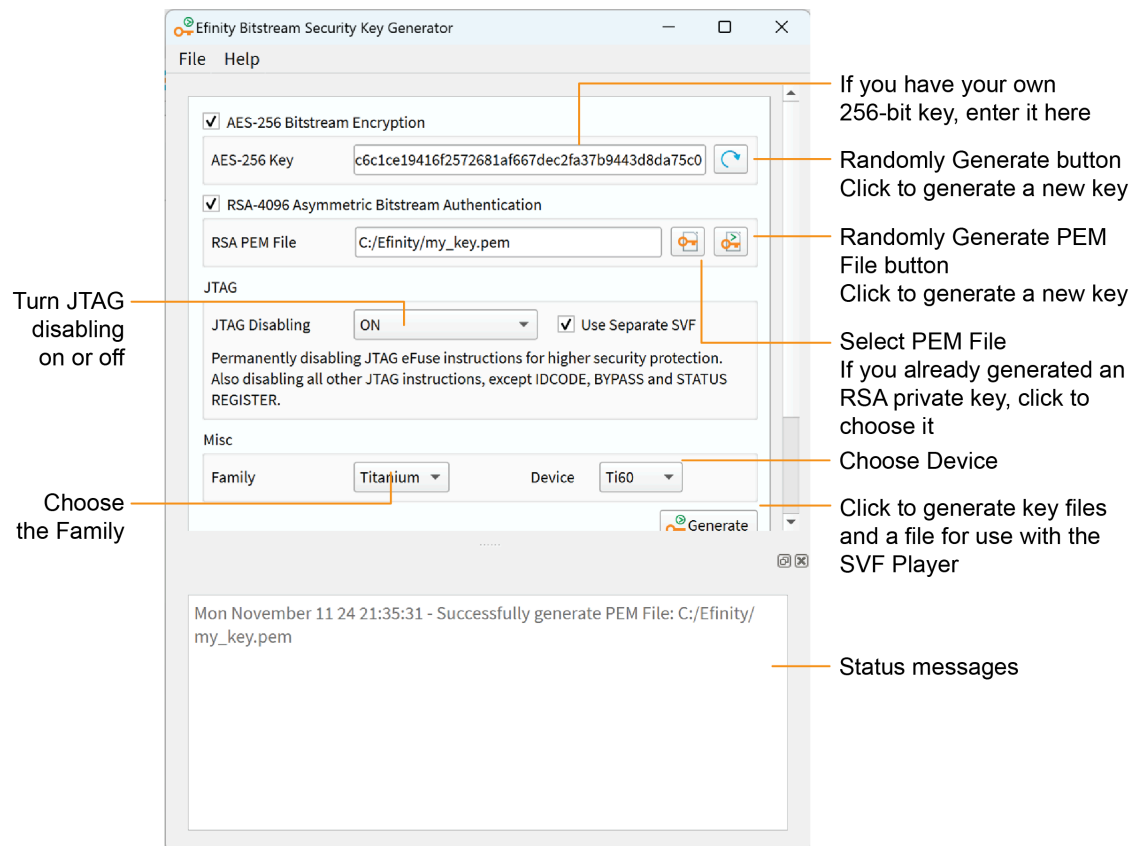
1. If you want to use encryption:

   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, and 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 128). 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 47: SVF Player
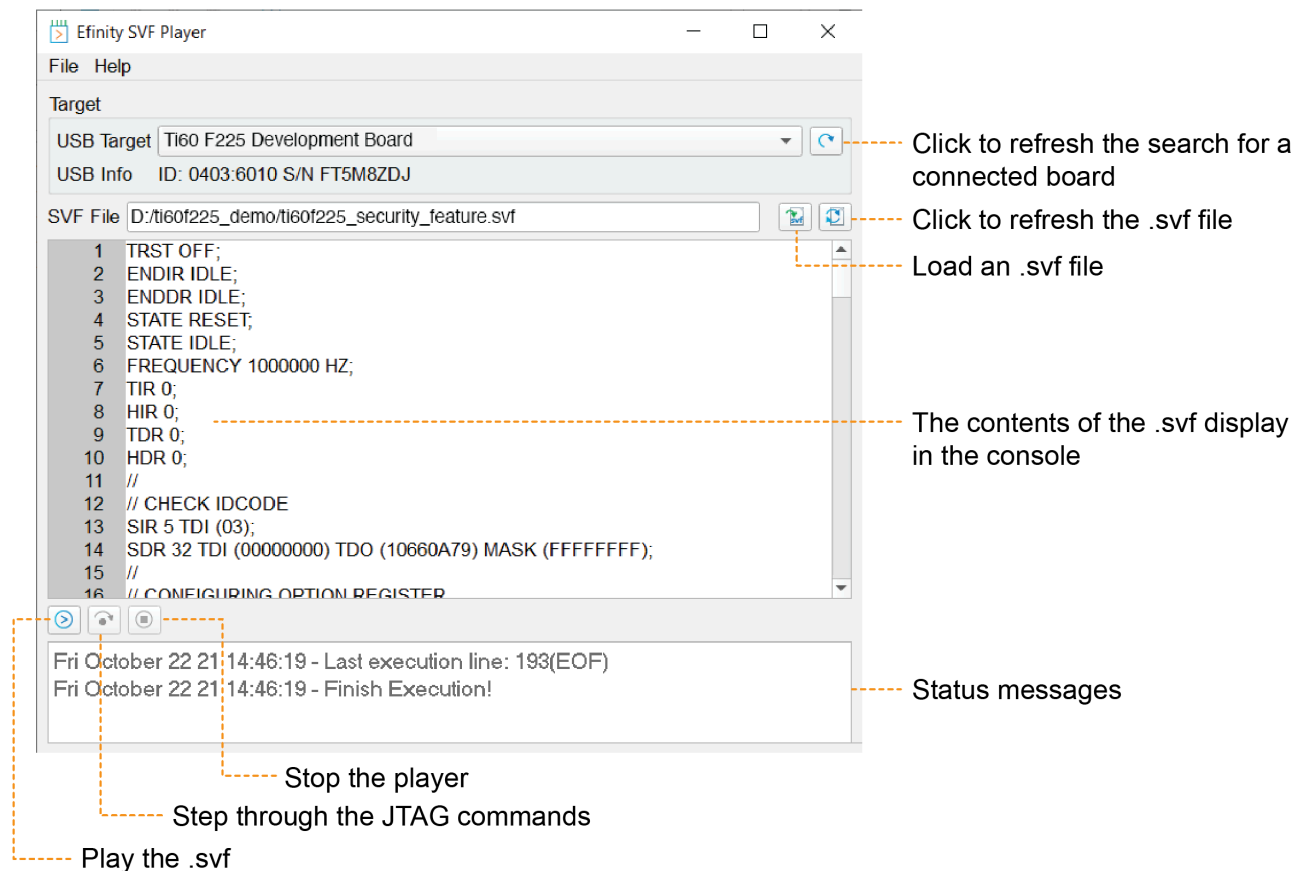
To blow fuses with the SVF Player:

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.

> (i) **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.

Enabling Security for Your Project

You set bitstream security options for your project in the Project Editor. After you enable these options, you only need to generate a new bitstream to apply them. You do not need to re-compile the design.

Table 33: Project Options for Security

Option	Description
Bitstream Encryption	(Titanium Topaz only) On: The software generates an encrypted bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box. Off: (default) The software generates a plaintext bitstream.
Bitstream Authentication	(Titanium Topaz only) On: The software generates a signed bitstream. You also need to specify the **.bin** file in the **FPGA Key Data File** box and the RSA private key (**.pem**) file in the **RSA Private Key** box. Off: (default) The software generates an unsigned bitstream.
FPGA Key Data File	(Titanium Topaz only) Specify the location and name of the **.bin** file you generated with the Efinity Bitstream Security Key Generator.
RSA Private Key	(Titanium Topaz only) Specify the location and name of the RSA private key file (**.pem**).

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 34: 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, Tz50, Tz110, Tz170 2: Ti85, Ti135, Ti165, Ti240, Ti375
--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.
--signature_file	N/A	Filename	Signature file generated from third-party signing utility.
--raw_content_output_file	N/A	Filename	Generated output file intended as main data input to third-party signing utility.

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 118 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 48: 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.

Chapter 11

Working with JTAG .svf Files

Contents:

- **Using the Efinity SVF Player**

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 101.

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 49: SVF Player
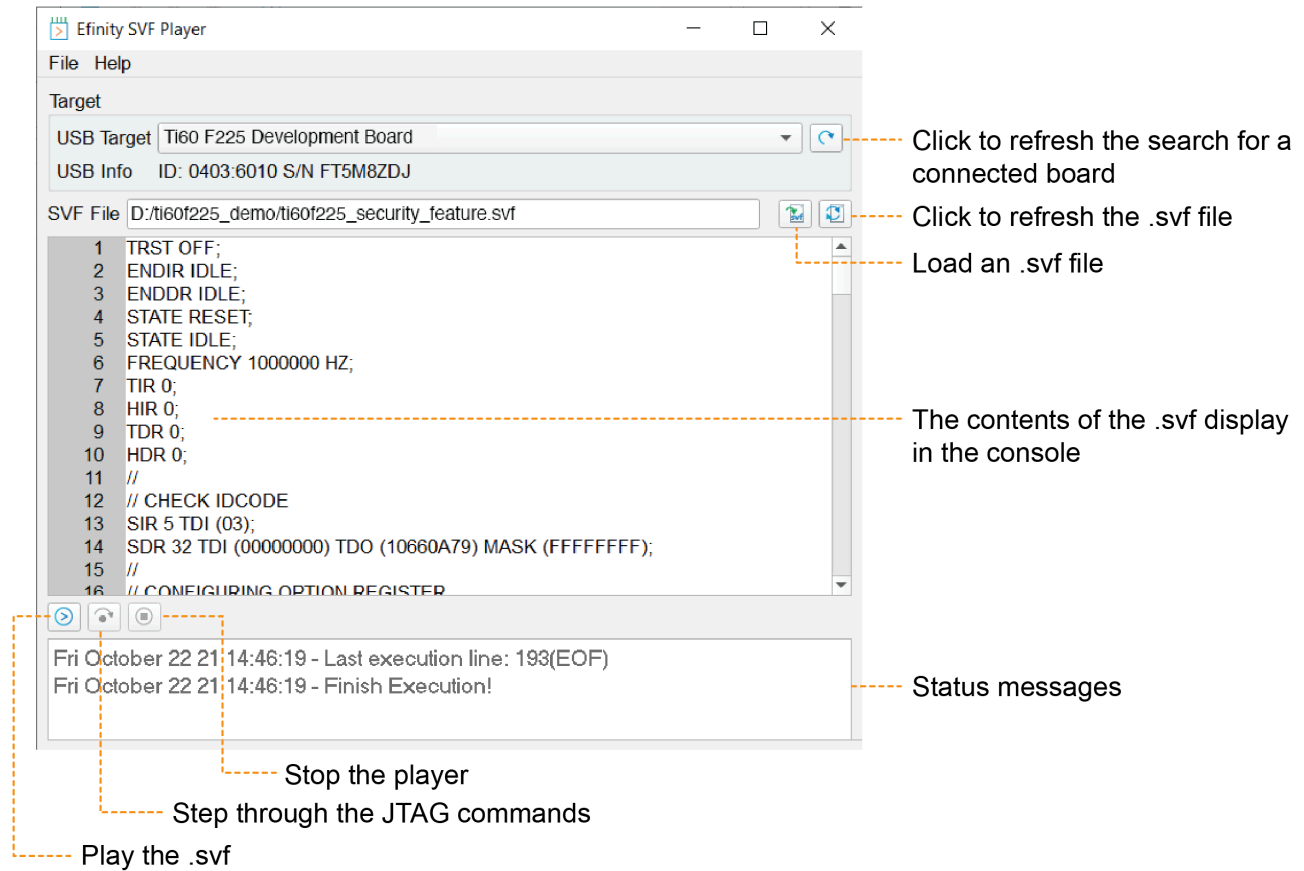
To use the SVF Player:

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.

Chapter 12

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 111.)

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.

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 35: 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 Linux USB Driver

The following instructions explain how to install a USB driver for Linux operating systems.

1. Disconnect your board from your computer.
2. In a terminal, use these commands:

```
> sudo <installation directory>/bin/install_usb_driver.sh
> sudo udevadm control --reload-rules
```

Note: If your board was connected to your computer before you executed these commands, you need to disconnect it, then re-connect it.

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.

   (i) **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.

(i) **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.

Appendix: Connecting Programming Hardware

SPI Programming Connections

The following figure illustrates the connection required when programming the SPI flash with FTDI FT2232H and FT4232H Mini Module.

Figure 50: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Trion FPGAs)

Note:
1 FTDI supports SPI x1 only.
2 The external pull-up is optional unless required by an external load.
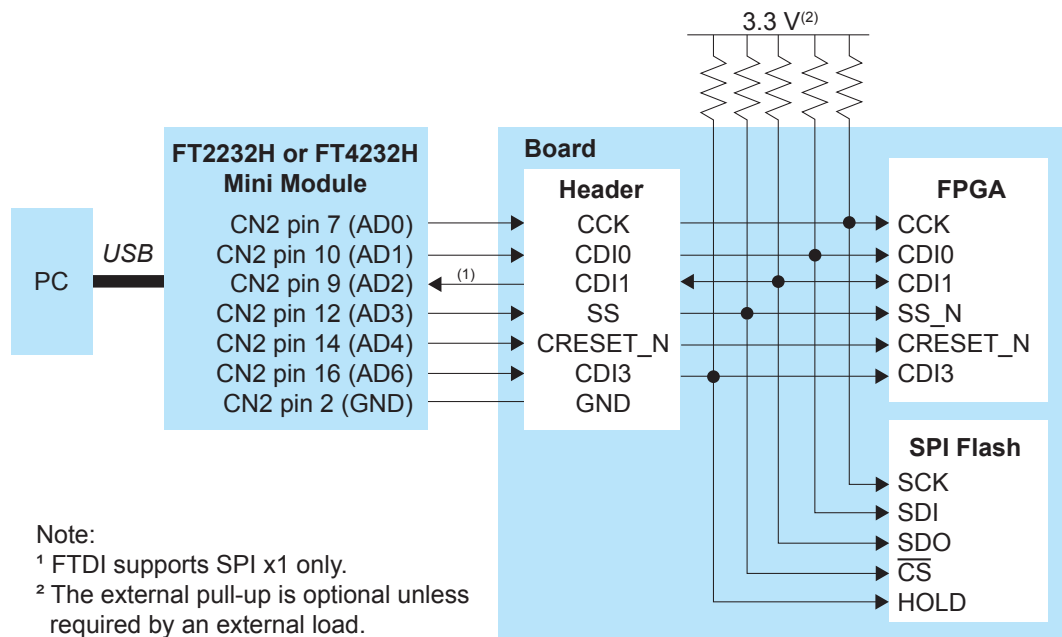
Figure 51: SPI Flash 1.8V Programming with FTDI FT2232H and FT4232H Mini-Module Connections

Note:
1 The external pull-up is optional unless required by an external load.
2 FTDI supports SPI x1 only.
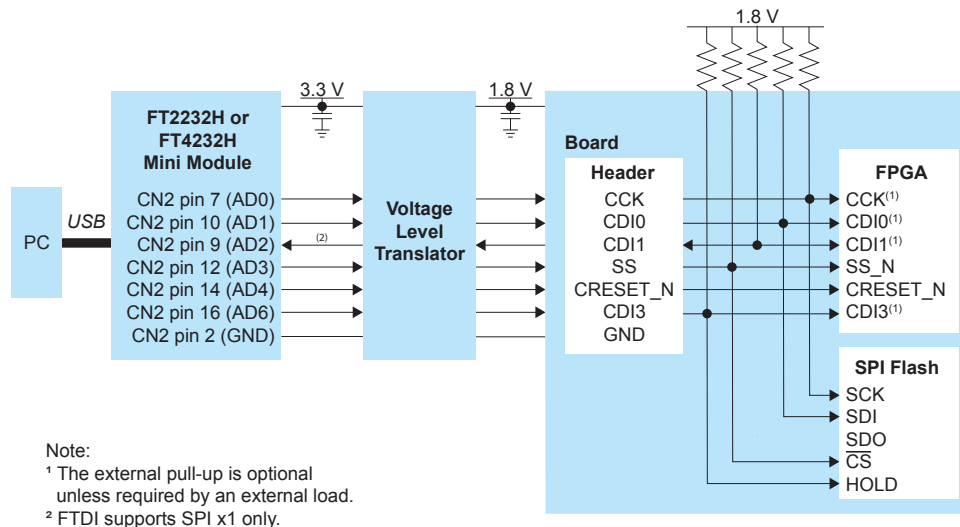
Figure 52: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Titanium Topaz FPGAs)

Refer to the voltage level translator data sheet for the capacitor values.
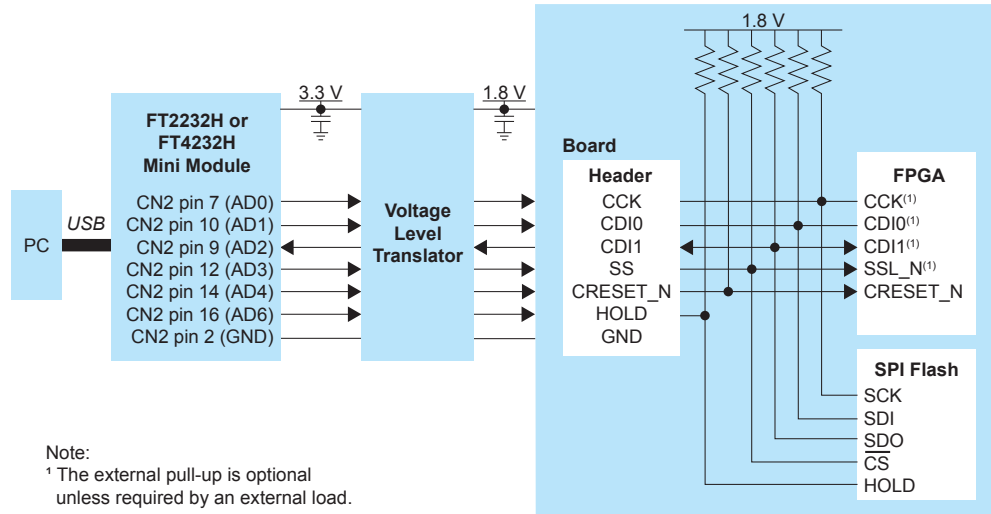
JTAG Programming Connections (Trion FPGAs)

Connecting a JTAG Cable

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

Connecting a JTAG Mini Module

When programming T4, T8, T13, T20WLCSP80, T20QFP100F3, T20QFP144, T20BGA256, and T20BGA169 FPGAs, use this connection:

Figure 53: Connect FT2232 Mini Module to JTAG Pins plus CRESET_N and SS_N
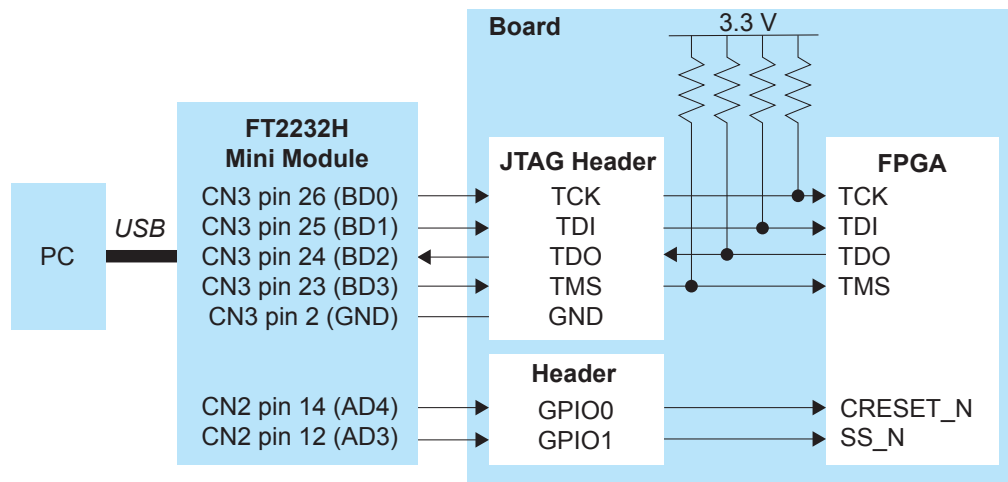
Figure 54: Connect FT4232 Mini Module to JTAG Pins plus CRESET_N and SS_N
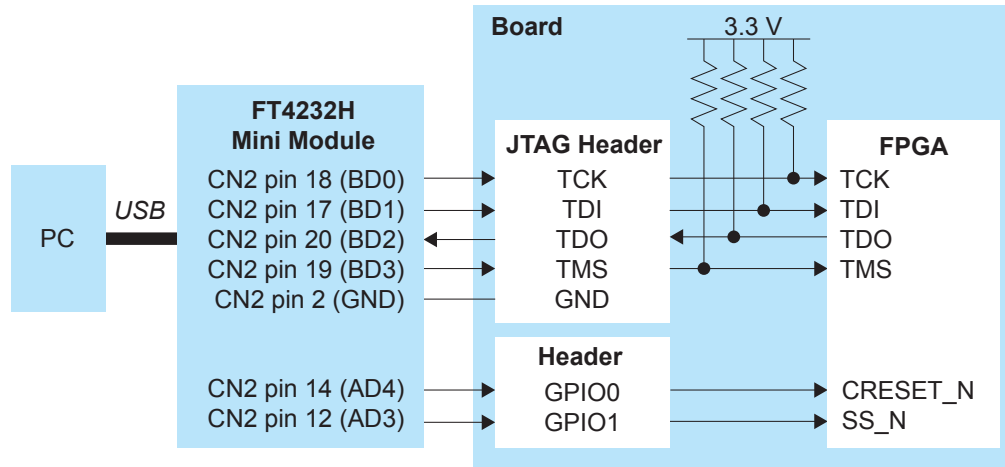
> **ℹ Note:** This figure uses the `CRESET_N` and `SS_N` pins in addition to the standard JTAG pins. However, this setup is only needed for JTAG configuration. You can use the standard 4 JTAG pins and any cable for other JTAG functions.

When programming T20BGA324, T20BGA400, T35, T55, T85, and T120 FPGAs, use this connection:

Figure 55: Connect FT2232 Mini Module to JTAG Pins
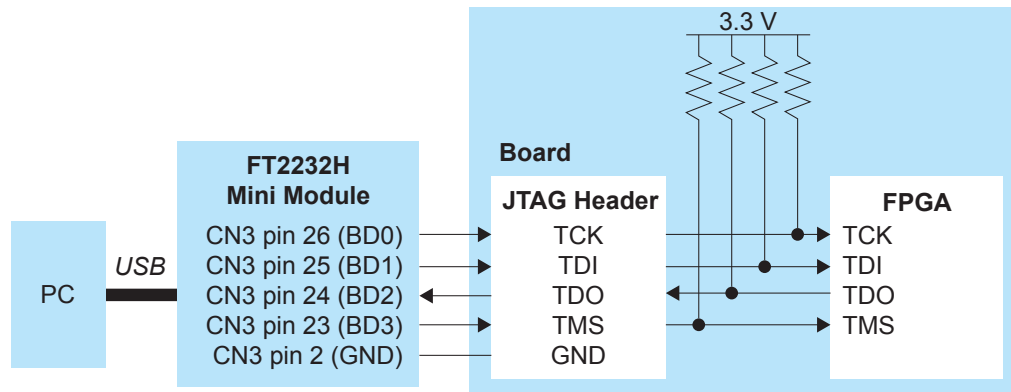
Figure 56: Connect FT4232 Mini-Module to JTAG Pins
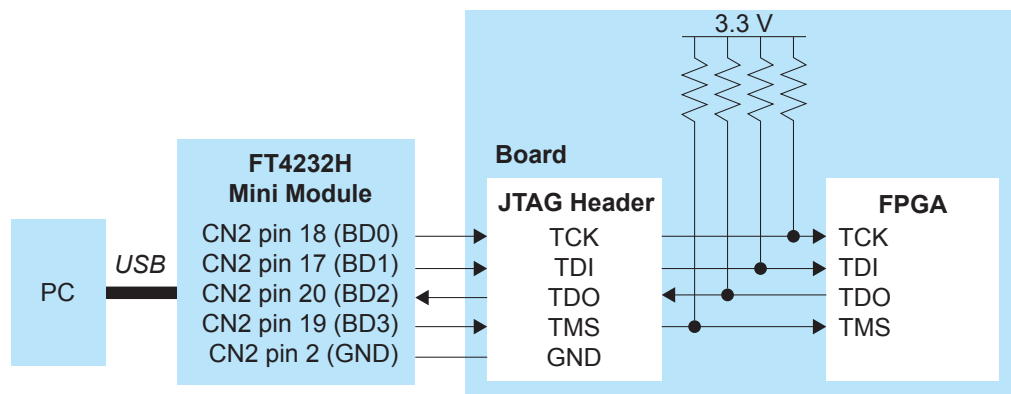
JTAG Programming Connections (Titanium Topaz FPGAs)

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

Connecting a JTAG Mini Module

When programming Titanium Topaz FPGAs with a JTAG Mini Module, use this connection:

Figure 57: Connect FT2232 Mini-Module to JTAG Pins
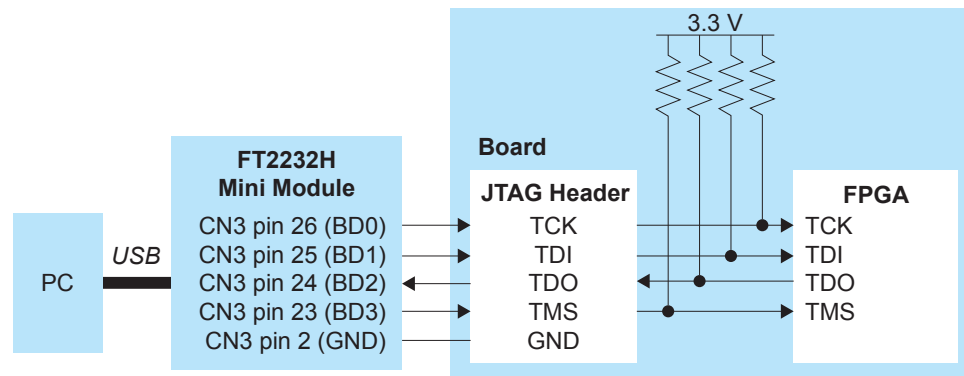
Figure 58: Example of FT2232 Mini-Module Connected to JTAG Pins Using 1.8V
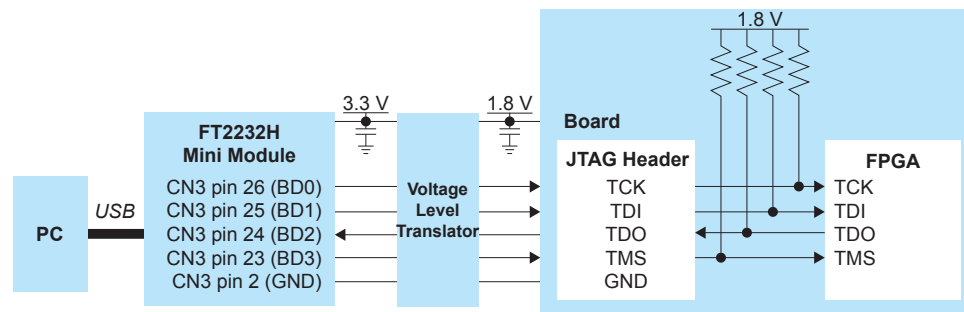
Figure 59: Connect FT4232 Mini-Module to JTAG Pins
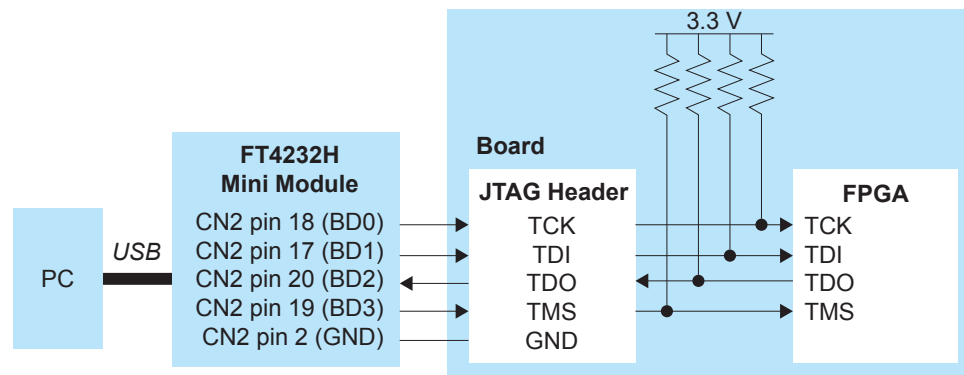
Figure 60: Example of FT4232 Mini-Module Connected to JTAG Pins Using 1.8V
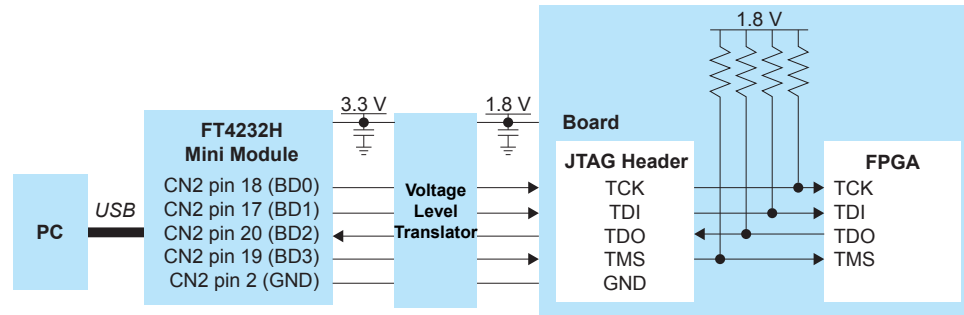
Appendix: Efinity Tools

This topic provides a list of tools included with the Efinity software.

Table 36: Efinity Tools

Tool	Description	Read More
Bitstream Security Key Generator	Simplifies the process of creating encryption keys and generating RSA certificates (for Titanium FPGAs only).	Using the Efinity Bitstream Security Key Generator on page 123
BRAM Initial Content Updater	Lets you quickly update the initial memory saved in the FPGA's BRAM without performing a full compile.	About the BRAM Initial Content Updater on page 97
Code Editor	Basic editor for viewing code or report files. You should use your own editor for real coding work.	
Debugger	Use to probe signals in your FPGA design via the JTAG interface.	Debugging on page 76
Debug Wizard	Provides an automated flow for adding a logic analyzer core to your design.	Debug Wizard on page 82
Floorplan Editor	Provides a graphical view of the logic and routing in your design.	
Interface Designer	Used to build the peripheral portion of your design such as PLLs, GPIO, MIPI, DDR, etc.	About the Interface Designer on page 48
IP Packager	Use this tool to "package" design files for re-use in the IP Manager.	Packaging Design Files on page 26
IP Manager	Interactive wizard that helps you customize and generate Efinix IP cores.	Using the IP Manager on page 40
JTAG SVF Player	JTAG SVF player that sends JTAG commands to an Efinix FPGA.	Using the Efinity SVF Player on page 131
Log Message	Tool to sort and browse through all of the messages resulting from the compilation flow.	Viewing Messages and Logs on page 35
Message Browser	Shows synthesis-specific messages that result when you elaborate the netlist.	Viewing Messages and Logs on page 35
Netlist Viewer	Displays and analyzes your design's netlist, including all components and their connections (nodes and nets).	Netlist Viewer (Beta) on page 32
Package Planner	Provides a visual representation of the FPGA package pins.	Viewing the Package Pinout on page 55
Programmer	Select bitstream images and program the FPGA directly or the flash device on a board.	About the Programmer GUI on page 93
Tcl Command Console	Enter Tcl commands to analyze and explore timing.	
Timing Browser	Helps you explore your design's critical paths and the cells of those paths.	
Transceiver Debugger	Lets you test and display the signal quality of the FPGA's transceiver signals.	Debugging Transceivers (Beta) on page 88

Appendix: Efinity Project Files

The following sections describe the important files the Efinity software uses and generates. Files in the **work*** directories are typically intermediate files used by the tools, and do not provide useful information for the user.

Efinity Source Files for Version Control

If you want to put your project under revision control, the files you need to store (in addition to your RTL) are:

- **<project>.xml**
- **<project>.peri.xml**
- **<project>.sdc**
- **debug_profile.wizard.json**
- **dbg_top.v**
- **<module>.v**
- **settings.json**

Bitstream Generation

<project>.hex

In GUI	Result pane > Bitstream menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the bitstream generation step
Design source?	No

The Efinity software creates this file during the bitstream generation step. This file is the **.hex** file you use to program in SPI active or SPI passive modes.

<project>.bit

In GUI	Result pane > Bitstream menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the bitstream generation step
Design source?	No

The Efinity software creates this file during the bitstream generation step. This file is the **.bit** file you use to program in JTAG mode.

<project>.pgm.out

In GUI	Result pane > Bitstream menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the bitstream generation step
Design source?	No

The software creates this file after bitstream generation; it contains all of the messages output to the Console.

Debugger

debug_profile.wizard.json

In GUI	Project Editor > Debuging tab
In file system	*<project>*
Created by	Efinity Debug Wizard
Design source?	Yes

The Efinity software creates this file when you use the Debug Wizard to add a logic analyzer core to your design. This file contains all of the settings you made in the wizard. For more information in the wizard and settings, refer to **Debug Wizard** on page 82.

dbg_top.v

In GUI	Result pane > Debugger menu
In file system	*<project>*/**outflow/work_dbg**
Created by	Efinity Debug Wizard
Design source?	Yes

The Efinity software creates this file when you use the Debug Wizard to add a logic analyzer core to your design. This file has the RTL logic for the debug core.

debug_TEMPLATE.v

In GUI	–
In file system	*<project>*/**outflow/work_dbg**
Created by	Efinity Debugger
Design source?	Yes

The Efinity Debugger creates this file when you create a logic analyzer or virtual I/O debug core manually. This file has the module for the debug profile you created.

Refer to **Virtual I/O Debug Core** on page 78 or **Logic Analyzer Debug Core** on page 80 for more information.

Interface Designer

<project>.peri.xml

In GUI	–
In file system	*<project>*
Created by	Interface Designer when you create an interface
Design source?	Yes

The Interface Designer creates this file when you create a new interface for your project. This file contains all of the settings that you specified in the Interface Designer for I/O banks, GPIO, LVDS, PLLs, MIPI, DDR, etc. You should not edit this file directly!

<project>.interface.csv

In GUI	**Result pane > Interfaces menu**
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file shows the constraints for the FPGA design pins used in the interface between the core and the periphery in a comma-separated values (**.csv**) file.

<project>.pt.rpt

In GUI	**Result pane > Interfaces menu**
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file provides details of the blocks used in the interface, including I/O banks, global connections, clock region usage, GPIO and dual-function configuration pins used, PLLs, LVDS, etc.

<project>.pinout.rpt

In GUI	**Result pane > Interfaces menu**
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file provides the board design pinout with pin number, signal name, pin name, I/O bank, etc. in a nicely formatted text file format.

<project>.pinout.csv

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file provides the board design pinout with pin number, signal name, pin name, I/O bank, etc. in a comma-separated values (**.csv**) format.

<project>.pt_timing.rpt

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file shows the interface's timing requirements based on the *<project>*.**pt.sdc**. The report has these sections:

- *PLL Timing Report*—Shows period and phase shift of output clocks from PLL.
- *GPIO Timing Report*—The report shows the following GPIO data:
  — The clock network delay, including the delay from GPIO_GCLK_IN to the core's global network and the delay from the PLL's clkout to GPIO_GCLK_OUT.
  — The output delay for GPIO configured as clock outputs (GPIO_CLK_OUT).
  — The delays for non-registered GPIO.
  — The delays for registered GPIO, including Timing Requirement of both Setup time and Hold time for path from FPGA pins to FPGA interface and the path delay from FPGA interface to FPGA pins.
- *JTAG Timing Report*—If you added a debug core to your design, this section shows the JTAG signal delay.

<project>.pt.sdc

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. The file is a template SDC that you use to create your own SDC file. You copy and paste the constraints into your own SDC and modify it as needed.

There are several types of contraints:

- *Clock constraints*—These constraints define the clocks and virtual clocks in your design. The file has create_clock constraints for the PLL clocks (the SDC file defines a clock period) and any GPIO clocks, that is, GPIO used as GCLK (you need to define the clock period for these).

- *GPIO constraints*—These constraints define the input delay and output delay from registered IO to core as well as input delay and output delay from non-registered IO to core.
- *Periphery constraints*—These are constraints for any interfaces, such as LVDS, MIPI, DDR, etc.

📖 **Learn more:** See **Copy and Paste the Interface Constraints** in the **Efinity Timing Closure User Guide** for instructions on using this file.

<project>_or.ini

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file contains bitstream configuration settings (option register settings) related to features you enable in the Interface Designer such as SEU and remote update. The Programmer uses this file when creating the bitstream.

<project>_template.v

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints
Design source?	No

The Interface Designer creates this file when you generate constraints. This file provides the a template Verilog HDL file defining the FPGA design pins based on the interface configuration. You can use this file as the starting point for the Efinity synthesis top-level target. The port list in the file matches the **Interface Designer-generated SDC constraint file**. To use this file:

1. Save the file with a different name to the directory where you keep your source files, such as your project directory.
2. Add the new file to you project as a design file.
3. Change the top-level entity in the Efinity project to be the module name given in this file. For example, if the module name is `pt_demo`, change the top-level entity name to `pt_demo` in **Project Editor > Design tab > Top Module/Entity**.
4. Add the design content.

Unified Design Flow

<project>.unified.isf

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Synthesis when generating constraints in the unified design flow
Design source?	No

The mapper creates this file when you generate constraints in the unified design flow. This file is an ISF that creates design instances and sets their properties based on the interface logic discovered by synthesis.

<project>.auto_asg.isf

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Synthesis when generating constraints in the unified design flow
Design source?	No

The mapper creates this file when you generate constraints in the unified design flow. This file is as ISF with the interface logic automatically assigned to resources. This file is only generated when a custom user ISF resource assignment file is not included.

<project>.peri_rtl.v

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints in the unified design flow
Design source?	No

The Interface Designer creates this file when you generate constraints in the unified design flow. This file is the interface netlist file for simulation. This file only contains the interface logic discovered by synthesis.

<project>.peri_pt.v

In GUI	Result pane > Interfaces menu
In file system	*<project>*/**outflow**
Created by	Interface Designer when generating constraints in the unified design flow
Design source?	No

The Interface Designer creates this file when you generate constraints in the unified design flow. This file is the interface netlist file for simulation. This file only contains the interface logic generated using the Interface Designer.

IP

<module>.define

In GUI	–
In file system	*<project>*/**ip/***<module>*
Created by	IP Configuration wizard
Design source?	No

When you generate an IP core, the IP Configuration wizard creates this file. The file contains all of the settings you specified for the IP core.

settings.json

In GUI	–
In file system	*<project>*/**ip**/*<module>*
Created by	IP Configuration wizard
Design source?	Yes

When you generate an IP core, the IP Configuration wizard creates this file. The file has the configuration settings for the IP core.

You can use this settings file to create another instance of the core with the same settings, or you can modify it to create another core with slightly different settings. For example, you can quickly create FIFOs of varying depths by re-using an existing **settings.json** file.

Learn more: Refer to **IP Settings File** on page 46 for instructions on using this file to create another instance of an IP core.

<module>_tmpl.v

In GUI	–
In file system	*<project>*/**ip**/*<module>*
Created by	IP Configuration wizard
Design source?	No

When you generate an IP core, the IP Configuration wizard creates this file. The file has the Verilog HDL template you can use to instantiate the IP in your RTL design.

<module>_tmpl.vhd

In GUI	–
In file system	*<project>*/**ip**/*<module>*
Created by	IP Configuration wizard
Design source?	No

When you generate an IP core, the IP Configuration wizard creates this file. The file has the VHDL template you can use to instantiate the IP in your RTL design.

<module>.v

In GUI	–
In file system	*<project>*/**ip**/*<module>*
Created by	IP Configuration wizard
Design source?	Yes

When you generate an IP core, the IP Configuration wizard creates this file. The file source code for the IP core.

Placement

<project>.place

In GUI	Result pane > Placement menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the placement step
Design source?	No

The Efinity software creates this file during the placement step. This file has the detailed placement report (block name, x,y coordinates, sub-block, and block number) for all of the logic blocks in the design shown in a nicely formatted text layout.

<project>.place.rpt

In GUI	Result pane > Placement menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the placement step
Design source?	No

The Efinity software creates this file during the placement step. This file shows the resources used after placement for inputs, outputs, clocks, LEs, memory, and multipliers (Trion) or DSP Blocks (Titanium).

The report's Resource Summary section shows how many core resources (inputs, outputs, clocks, etc.) the design uses. In this context, the inputs and outputs are the connections between the core and the periphery (or interfaces); they do not represent package pins. Different versions of software model these connections differently, which can cause the number of available input or output connections to change from one release to the next.

For example, the Efinity software v2022.2 includes a `-reference_pin` option for the `set_input_delay` and `set_output_delay` constraints for Trion FPGAs. To model these pins, the software adds more clock connections from the core to the interface to the report's I/O counts. Therefore, for the same design, you may notice a higher number of inputs and outputs in the report file in the Efinity software v2022.2 or higher.

To find the number of GPIO used (meaning the number of package I/O pins as inputs and outputs), refer to the Result pane's GPIO Periphery Resource field or *<project>*.**pt.rpt** in the **outflow** folder.

<project>.place.out

In GUI	Result pane > Placement menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the placement step
Design source?	No

The Efinity software creates this file during the placement step. This file shows the messages output to the Console during placement. Review all SDC messages and adjust your constraints as needed. Warning messages flag issues that can affect timing closure.

Project

<project>.sdc

In GUI	Project pane > Constraint menu
In file system	*<project>*
Created by	User defined
Design source?	Yes

This file is the Synopsys Design Constraints (**.sdc**) file you use to constraint your design to meet timing requirements. It is too much information to explain how to do that here, so refer to **Efinity Timing Closure User Guide** for the full details.

<project>.xml

In GUI	–
In file system	*<project>*
Created by	Efinity software when you create a project
Design source?	Yes

The Efinity software creates this file when you create a new project. This file contains all of the information about your project, including any settings you make in the Project Editor. You should not edit this file directly!

Routing

<project>.pnr.rpt

In GUI	Result pane > Routing menu
In file system	*<project>***/outflow**
Created by	Efinity software during the routing step
Design source?	No

The Efinity software creates this file during the routing step. This file shows the resources used after placement and routing for inputs, outputs, clocks, LEs, memory, and multipliers (Trion) or DSP Blocks (Titanium).

<project>.route.rpt

In GUI	Result pane > Routing menu
In file system	*<project>***/outflow**
Created by	Efinity software during the routing step
Design source?	No

The Efinity software creates this file during the routing step. This file shows the routing report, including global control information.

<project>.route.out

In GUI	Result pane > Routing menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the route step
Design source?	No

The software creates this file after routing; it contains all of the messages output to the Console. This file shows detailed information about the number of routing iterations and cost time when routing your design. If your design does not route, you can try adjusting the place-and-route optimization level for congestion.

Learn more: See **Place-and-Route Options** in the **Efinity Timing Closure User Guide** for information on using the options for optimization level.

<project>.timing.rpt

In GUI	Result pane > Routing menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the route step
Design source?	No

The software creates this file after routing. This static timing analysis report contains detailed information about your design's critical paths. The report has severl sections:

- *Clock Frequency Summary*—Shows a summary of the clocks in your design and their constraints. It uses the critical paths to show the maximum clock frequency that each clock can achieve. In the summary you can check the clock constraints defined in your SDC file, the maximum frequency of the clocks in your design, and the edge of the launch clocks and capture clocks.
- *Clock Relationship Summary*—Lists the related clocks, their constraints, and the slack. The report shows measurements using the active clock edge. This report shows how many pairs of launch clocks and capture clocks are involved when routing your design and the slack of the most critical setup path among related clocks. If any of the clock relationships have negative slack, your design *has not* closed timing.
- *Path Details for Max Critical Paths*—Shows the critical paths for the maximum (setup) critical paths. The report only shows the most critical path for each relationship. This section gives detailed information for the Launch Clock Path, Capture Clock Path, Data Path (including Clock To Q + Data Path Delay). Usually, the most efficiency way to reduce the data path delay is to fix negative slack.
- *Path Details for Min Critical Paths*—Shows the critical paths for the minimum (hold) critical paths. The report only shows the most critical path for each relationship.

Learn more: Refer to **Interpreting Timing Results** and **Tools for Exploring Timing** in the **Efinity Timing Closure User Guide**.

Synthesis

<project>.map.v

In GUI	Result pane > Synthesis menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the synthesis step
Design source?	No

The Efinity software creates this file during the synthesis step. This file has the post-mapping netlist that you use for simulation.

<project>.map.core.v

In GUI	Result pane > Synthesis menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the synthesis step.
Design source?	No

The Efinity software creates this file during the synthesis step. This file has the post-mapping core netlist that you use for simulation in the unified design flow. This file is only generated in the unified design flow and contains only the core logic.

<project>.map.peri.v

In GUI	Result pane > Synthesis menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the synthesis step.
Design source?	No

The Efinity software creates this file during the synthesis step. Post-mapping netlist file for simulation with the unified design flow. This file is only generated in the unified design flow and contains the interface logic as well as the core netlist..

<project>.map.rpt

In GUI	Result pane > Synthesis menu
In file system	*<project>*/**outflow**
Created by	Efinity software during the synthesis step
Design source?	No

The Efinity software creates this file during the synthesis step. This file contains all of the reporting for the synthesis step. It give the top-level entioty, the files in the design as well as results of pre-optimization and mapping, post-optimization and re-synthesis, and estimates for the resource usage for each module.

<project>.map.out

In GUI	Result pane > Synthesis menu
In file system	<project>/outflow
Created by	Efinity software during the synthesis step
Design source?	No

The Efinity software creates this file during the synthesis step. This file has all of the messages the synthesis tool outputs to the Console, including any synthesis warnings or errors.

<project>.res.csv

In GUI	Result pane > Synthesis menu
In file system	<project>/outflow/work_syn
Created by	Efinity software during the synthesis step
Design source?	No

The Efinity software creates this file during the synthesis step. This file has the resource usage for all of the modules in the design. When you double-click on this file in the **Result pane > Synthesis** menu, the report opens with two tabs:

- The **Hierarchy** tab shows a tablular view of the modules and the resources used. You can filter the list using the filter field at the top.
- The **Text** tab shows a plain text view of the same data.

Appendix: Shortcuts

This section provides a list of shortcuts when working with the Efinity software.

Table 37: Shortcuts

Action	Shortcut
To launch the Efinity GUI with a project already loaded	Use this command in a terminal or command prompt: ``` $Efinity$\bin> setup.bat --run efinity --project <project_name>.xml ```
To add a design file to your project	Go to **Dashboard** > **Project pane** and right-click **Design**. Choose **Add** from the pop-up menu, browse for the file, and click **Open**.
To add an SDC file to your project	Go to **Dashboard** > **Project pane** and right-click **Constraint**. Choose **Add** from the pop-up menu, browse for the file, and click **Open**.
To open the file system folder where your project resides	Go to **Dashboard** > **Project pane** and right-click the filename. Choose **Open Containing Folder**.
To open a design file with your chosen text editor	Go to **Dashboard** > **Project pane** and right-click the filename. Choose **Open with User Editor**. (Make sure you have set the path to your editor first in **File** > **Preferences** > **External text editor**.)
To delete a file from your project	Go to **Dashboard** > **Project pane** and right-click the filename. Choose **Delete**.

Appendix: Icon List

Dashboard Icons

	Synthesize	Place	Route	Generate Hex Bitstream	Stop
Active					
Success					
Error					
Warning					
Disabled					

Toggle Automated Flow

General Icons

Filter

Search

General Tool Preferences

Help

Exit

Property Icons

Expand All

Collapse All

Toggle Properties Panel

Log Icons

Message Browser

Log Browser

Project Icons

Open Project

New Project

Close project

Edit Project

Choose Project Directory

Remove a File from a Project

Add a file to a Project

Import Design & Constraint Files

Save

Results are out
of sync with settings

Flow Icons

Run Complete Flow

Synthesize

Place

Route

Place & Route

Stop

Generate Bitstream File

Floorplan Icons

View Floorplan	Zoom In	Show Timing Delay
Show Cell Browser	Zoom Out	Show All Nets
Show Cell Browser Filter	Fit View	Trace Nets (Open Net Tracer)
Toggle Floorplan Filter/Legend	Show Fanin	Clear Net Trace
Filter Floorplan View	Show Fanout	World View (Dragon's Eye View)
Floorplan Legend	Show Timing Path	Load Place and Route Data

Timing & Console Icons

Toggle Timing Browser	Lock Scrolling
Toggle Tcl Command Console	Toggle Console
Run Tcl Command	Clear Console
Clear Tcl Command	Load Place and Route Data

Interface Designer Icons

Interface Designer	Export GPIO Assignments	Resource Assigner
Add Block	Import GPIO Assignments	Toggle Instance View and Resource View
Create a GPIO bus	Clear Design	Clear Resource
Delete Block	Check Design for Errors	Clear All Resources
Show or Hide Block Editor	Export Settings	Show/Hide Filter
Resource Assigner	Generate Constraints File	Clear Filter
	Package Planner	

Tools Icons

IP Catalog	Key Generator	SVF Player
IP Editor	Randomly Generate PEM File	Open SVF File
Netlist Viewer	Select PEM File	Reload
Show Dashboard		Run
		Step Over
		Stop

Programmer Icons

Programmer	Add Image File	Remote Hardware Server
Import JTAG Chain File (JCF)	Delete Image File	Advanced Device Configuration Status
Export	Combine Image Files	
Edit Header	Select Image File	

Debugger Icons

Debugger	Add Probe	Add Net
Debug Wizard	Add Source	Remove Net
Add Debug Core	Remove Probe or Source	Active High
Remove Debug Core	Import Debug Profile	Active Low
Connect Debugger	Refresh USB Target	Toggle High
Disconnect Debugger	Select Waveform File	Toggle Low
	Advanced Device Configuration Status	

Package Planner Icons

Exit	Show/Hide View Config	Left Rotate
Refresh	World View	Right Rotate
Help	Zoom In	Reset Orientation
	Zoom Out	Legend
	Fit to Window	Pin Browser
	Show Bottom View	Export Diagram
	Show Package Top	

Revision History

Table 38: Document Revision History

Date	Version	Description
November 2024	15.0	Added new features in 2024.2. Updated device support. **Project Editor** > **Design tab** > **Top Module/Entity** cannot be left empty. (DOC-2137) Updated Table 4: Machine Memory Requirements on page xii. (DOC-2052) Corrected link to latest Microsoft Visual C++ Redistributable downloads. (DOC-2045) An EFX_FF primitive cannot be placed in a Trion ELF tile. Described **.f** files for referencing RTL source code. (DOC-2072) Updated screen shots for unified design options. (DOC-2184) Added Linux requiremens fro Java. (DOC-2056) Added new # Trigger option for Debugger Logic Analyzer. (DOC-1886) Double-clicking a file in the Project or Result pane opens it in the default or user editor. (DOC-2146) Added topic about encrypting and/or signing bitstreams at the command line.
August 2024	14.1	Added Ti85 and Ti135 FPGAs.
June 2024	14.0	Updated device support for v2024.1. Added topic on Packaging Design Files on page 26. Red Hat support is v8.0 and higher. Removed support for v7.4. (DOC-1648) The Console supports color output and dark mode. (DOC-1762) Added the `sta_tclsh` flow option to open the Tcl Console form the command line. (DOC-1881) Added chapter on the Efinity Transceiver Debugger. (DOC-1941) 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	13.6	Added Ti165 and Ti240 FPGAs, replacing the Ti135 and Ti200, respectively.
April 2024	13.5	Added bitstream support for F100 package for the Ti35 and Ti60 FPGAs. Pinout is final for F100 package for the Ti35 and Ti60 FPGAs. Q3 timing model is final for the Ti90, Ti120, and Ti180 FPGAs in the J484 package.
March 2024	13.4	Added F100 and F256 packages for the Ti35 and Ti60 FPGAs. Added F256 package for the T35 FPGAs.
February 2024	13.3	Updated table of supported Titanium FPGAs.
February 2024	13.2	Added note in Generated Files-Testbench. (DOC-1691)
January 2024	13.1	Added Ti135 and Ti200 to device support and machine memory requirements. (DOC-1660) Added JTAG device IDs for Ti135, Ti200, and Ti375. Added note explaining that you should make a backup of your existing project before opening it in a newer software version because the project files are not backwards compatible. (DOC-1632) Added note about Windows %PATH% variable to Hardware and Software Requirements on page xii. (DOC-1687)

Date	Version	Description
December 2023	13.0	Updated device support and new in v2023.2. Updated machine memory requirements. For Windows, a 64-bit operating system is required. 32-bit systems are not supported. Added explanation about the input and output numbers listed in the Core Resoures section of the **Result** pane and in the *<project>*.**place.rpt** file. You can open multiple Debugger windows by clicking the Debugger icon multiple times. Added information on how to reference Trion and Titanium VHDL primitive libraries.
November 2023	12.2	Added bitstream support for G400 packages. Added note to use only ASCII characters. (DOC-1522)
August 2023	12.1	Added G400 package support. (DOC-1393)
June 2023	12.0	Updated device support and new in v2023.1. Added section about the Netlist Viewer tool. Added section about the BRAM Initial Content Updater. Updated description for **Preferences** dialog box. Added topic on how to preserver place-and-route for a portion of your design. Added additional information on design migration. Added appendix describing all tools included with the Efinity software.
December 2022	11.0	Updated device support and new in v2022.2. Added section on constraining routing manually. EFX_COMB4 not available in Trion FPGAs. (DOC-1074) Added description of Debugger **Options** menu. (DOC-1029) Added topics on how to constrain routing (beta).
September 2022	10.1	Updated Project-Based Programming Options topic for new options. Updated PFGA support for Efinity patch 2022.1.226.1.9.
August 2022	10.0	Added new project-based programming option for 4-byte addressing. Updated the available options for the**Project Editor** > **Place and Route** tab. (DOC-889) Clarified the instructions for instantiating debug cores. (DOC-883) 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. Removed support for C232HM-DDHSL-0 cable. (DOC-860) Added a topic on the concurrent debug feature. Updated supported IP cores. Updated Installing USB Drivers topics. Updated supported IP cores.
June 2022	9.2	Pointed to new sourceforge location for GTKWave download. (DOC-797)

Date	Version	Description
April 2022	9.1	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.
		Moved topics on installing USB drivers and connecting programming hardware to the appendix.
		The minimum operating frequency of the debug cores is 2 times the JTAG TCK frequency. (DOC-754)
		Added CORDIC core to the list of supported IP (included with Efinity patch v2021.2.323.2.18).
December 2021	9.0	Added Efinity Hardware Server documentation. (DOC-598)
		Added support for FTDI FT4232H Mini Module. (DOC-597)
		Added the JTAG USERCODE option to the Project-Based Programming Options topic.
		With the Efinity software v2021.2 and higher, you **must** use **.hex** for SPI and **.bit** for JTAG. (DOC-638)
		When importing an IP configuration **.json** file, specify the module name in the IP Configuration wizard. (DOC-611)
		Updated machine memory requirements (RAM).
		You may need to re-compile when upgrading from an older version.
		Added appendix of project file definitions.
November 2021	8.2	Added instructions on using the Titanium bitstream security features.
		Added instructions for using the Efinity SVF Player.
		Described how to export a bitstream to serial vector format (**.svf**).
		When using the stand-alone Programmer on 64-bit Windows, install both the x86 and x64 libraries. (DOC-576)
		Added instructions for importing IP cores. (DOC-584)
October 2021	8.1	Added topic on flash programming modes.
		Added topic on the Titanium configuration status registers. (DOC-487)
		Added note that FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495)
		Added description of command to convert bitstream files from **.hex** to **.bin** to Exporting to Raw Binary Format topic. (DOC-527)
		JRE required for running the DMA Controller in the IP Manager. (DOC-549)
		Added a note that you need to specify the path when simulating with testbench files that are not in the project's root directory. (DOC-468)
June 2021	8.0	Added support for Titanium family.
		Supported Ubuntu version is v18.04 or higher. v16.04 is end of life. (DOC-433)
		Added the Java runtime environment as a software requirement for configuring the Sapphire SoC in the IP Manager.
		Described more detail on the Enable Initialized Memory in User RAMs option in the **Project Editor** > **Bitstream Generation** tab. (DOC-458)
		Added table of IP cores supported by family.
		Updated the FTDI command-line programming topic. Added the command-line programmer configuration mode options. (DOC-430)
January 2021	7.1	Corrected JTAG chain file code example. (DOC-368)

Date	Version	Description
December 2020	7.0	Added a new chapter on using the IP Manager. Added instructions on using VHDL libraries. Explained how to resize the Project, Netlist, and Result panes. Described the context-sensitive menus in the Project, Netlist, and Result panes. Added requirement to install the Microsoft Visual C++ 2015 x86 runtime library for the standalone Programmer. (DOC-315) Updated instructions for performing JTAG programming at the command line. (DOC-323) Corrected JTAG Mini Module pin names for T4, T8, T13, T20BGA256, and T20BGA169 connection setup. Clarified Undefined clock domain signals in the Debug Wizard. Added table of files shown in the Result pane. (DOC-277) Interface scripting file now supports PLL.
November 2020	6.1	Updated instructions on installing Windows USB drivers. Added FTDI cable and module connection for T20BGA400. Added JTAG device IDs for T20BGA324 and T20BGA400. Removed the FTDI2232 from About USB Drivers topic making the description applicable to other FTDI chips. Corrected the command for using --pgm_opts with the command-line programmer.
June 2020	6.0	Updated for v2020.1 release. Windows 7, Red Hat v6, and CentOS v6 no longer supported. Removed the chapters on SDC constraints and Tcl commands. This content is now in the Efinity Timing Closure User Guide. Added a topic on Efinity synthesis. Added a topic on project migration. Updated Programmer content to reflect new GUI and features. Consolidated and updated content on installing USB drivers for boards, C232HM-DDHSL-0 cable and FTDI FT2232H module. Added support for FTDI FT2232H module for JTAG programming. Added a topic on the various ways to view messages and logs. Added topic on the Interface Designer/s Resource View. Added a topic on using an API for scripting an interface design. Added topic on Interface Scripting File (**.isf**).
December 2019	5.0	Updated for v2019.3 release. Added chapter on using the Debugger. Added explanation that 2 unassigned pairs of LVDS pins should be located between and GPIO and LVDS pins in the same bank.
August 2019	4.5	Updated for v2019.2 release. Added information on enhanced Resource Assigner. Added information on JTAG programming. Added command-line instructions for using the Windows **efx_run.bat** file.
April 2019	4.4	Updated for v2019.1 release. Added information on new project manager capabilities. Updated `set_false_path` usage.

Date	Version	Description
January 2019	4.3	Updated for v2018.4 release. Added more information on simulation and waveform viewing. Added instructions for installing Windows USB driver. Updated Programming information.
October 2018	4.2	Added a note pointing to AN 006: Configuring Trion FPGAs for more information about using multiple images and daisy chaining for configuration. Added Python 3 to the software requirements list as an option. For Windows, if you do not have a full version of Python, the **.py** extension may not be correctly associated with Python.
June 2018	4.1	Removed Python requirement; as of this release, Python is included with the software. Added the requirement that Windows users install the Microsoft Visual C++ 2015 x64 runtime library.
April 2018	4.0	Updated for v2018.0 release. Added "Constraining Logic and Assigning Pins" topic, which replaces section on fine-tuning your design. Updated information on device configuration.
Novenber 2017	3.1	Minor updates.
May 2017	3.0	Updated for v2017.0 release. Described new Floorplan Editor tools. Updated SDC constraint information.
May 2016	2.0	Updated for v2016.0 release. Documented the Timing Browser. Documented the Tcl Command Console and available Tcl commands. Updated SDC constraint information.
July 2015	1.1	Minor updates.
May 2015	1.0	Initial release.