Efinity® Software User Guide

UG-EFN-SOFTWARE-v18.1
June 2026
www.efinixinc.com

Contents

Figures

Tables

Chapter 1

Introduction

Contents:

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

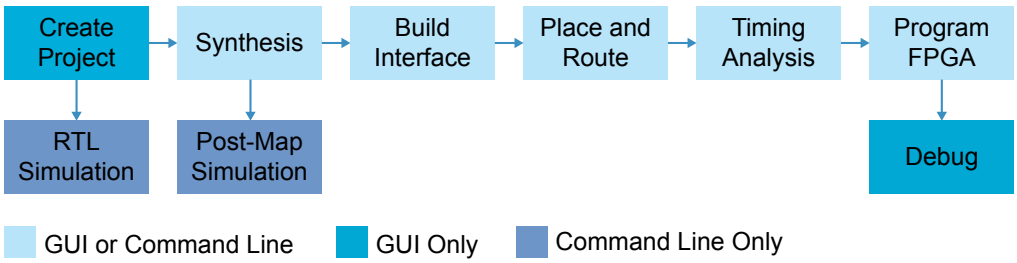


Table 1: Titanium FPGAs Supported in Efinity® Software v2026.1 (or Patches)[1]

FPGA	Package	Bitstream	Timing	Pinout
Ti35	F100, F100S3F2, F225, F256	✓	Final	Final
Ti60	V64, W64, F100, F100S3F2, F225, F256	✓	Final	Final
Ti85	N441, N484, N576, N676	✓	Final	Final
Ti90	J361, G400, J484, L484, G529	✓	Final	Final
Ti95	F225, F324	-	Preliminary	Preliminary
Ti120	J361, G400, J484, L484, G529	✓	Final	Final
Ti125	F225	✓	Preliminary	Final
	M225S4F4, F324	-	Preliminary	Preliminary
Ti135	N441, N484, N576, N576D2F4, N676	✓	Final	Final
Ti165	N484, C529, N900, N1156	✓	Final	Final
Ti180	J361, G400, J484, L484, M484, G529	✓	Final	Final
	J484D1	✓	Final	Final

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

FPGA	Package	Bitstream	Timing	Pinout
Ti240	N484, C529, N900, N1156	✓	Final	Final
Ti375	N484, C529, N900, N1156	✓	Final	Final

Table 2: Topaz FPGAs Supported in Efinity® Software v2026.1 (or Patches)[2]

FPGA	Package	Bitstream	Timing	Pinout
Tz50	F100, F225, F256	✓	Final	Final
Tz75, Tz100	N441, N484, N576, N676	✓	Final	Final
Tz110, Tz170	J361, J484, G400	✓	Final	Final
Tz200, Tz325	C529, N484, N900	✓	Final	Final

Table 3: Trion FPGAs Supported in Efinity® Software v2026.1

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	Final
T55, T85, T120	F324, F484, F576	✓	Final	Final

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

New in v2026.1

The Efinity® software v2026.1 has the following new features and enhancements:

- Introduces support for Ti125 and Ti95 devices
  - Supports enhanced I/O performance and power with the HSIO2 pins
  - Up to 2.5 Gbps MIPI RX/TX performance, and significantly lower power consumption for MIPI HS-RX (compared to HSIO)
  - Up to 1.8 Gbps LVDS RX/TX performance
  - New PHY periphery clock network and distributed clock modifier (CM) resources optimize usage of PLLs and core clock network resources, provide easier peripheral clock skew management, and support higher performance
- New device and package combinations include:
  - Ti125 and Ti95 in F225 packages, supporting easy migration from existing Ti60 and Ti35 F225 designs
  - Ti125M225S4F4, with in-package SPI flash memory and two HyperRAM devices in a small package footprint
  - Ti125 and Ti95 in F324 packages, with more HSIO2, HSIO, HVIO pins
- Enables bitstreams for the following devices
  - Ti135N576, Ti85N576
  - Tz100N576, Tz75N576
- Reduces place and route compile times by about ~ 30%
- Introduces beta support for design partitions
  - Using the Efinity® Partition Planner, users can preserve and export/import synthesis, placement, and routing results of design instances within bounded rectangular partitions in the core fabric
  - Using partitions can ease integration of validated design components into a single FPGA design, and works well with team-based design and incremental compile flows
- Significantly improves responsiveness of the Efinity Floorplan Viewer and Console
- Improves Python toolkit with **efx_py** Python tool that lets you create and modify projects with Python scripting
- Adds Automated Option Explorer (AOE) scripting tool **efx_run_aoe.py**
  - Explore synthesis and placement compiler options
  - Search for combinations that achieve the highest f_{MAX} across system clocks
- Adds GUI Console customization options; limit, filter, or suppress compiler messages
- Allows multiple Interface Designer project files to exist in an Efinity project directory
- Adds a smart bit-generator compiler shortcut to the Interface Designer to generate a new bitstream quickly when changes to do not require re-running place-and route
- Adds dark theme option to Efinity GUI

Example Files

Starting with v2025.2, the Efinity software includes board-specific design files for tutorials and examples in the *<install directory>***/examples** folder. To make it easier for you to use these examples with various Efinix development kits, each example includes Interface Scripting Files (**.isf**). To use these files:

1. Open the project **.xml** file in the Efinity software.
2. Open the Interface Designer.
3. Import the **.isf** for the board you are using, e.g., use the **Ti180J484_kit.isf** to load the interface for the Ti180 J484 development board.

Note: Refer to Importing and Exporting Assignments on page 64 for more information about **.isf**s.

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 Software Installation User Guide
- Efinity Synthesis User Guide
- Efinity Timing Closure User Guide
- Efinity Trion Tutorial
- Efinity Debugger Tutorial
- Efinity Programmer User Guide
- Efinity IP Packager User Guide
- Trion Interfaces User Guide
- Titanium Interfaces User Guide
- Topaz Interfaces User Guide
- Efinity Interface Designer Python API
- Efinity Command-Line Interface User Guide
- Quantum® 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 operating system, at least dual-core
- Your preferred text editor such as Notepad, gVim, Visual Studio
- Machine memory requirements (when compiling Efinity designs):

Table 4: Machine Memory Requirements

These requirements assume up to 16 threads, and include 4 GB for the operating system and background applications.

Product	Model	Memory
Trion	T4, T8, T13, T20, T35	8 GB
	T55, T85, T120	8 GB
Titanium	Ti35, Ti60, Ti85, Ti90, Ti120, Ti135, Ti180	8 GB
	Ti165, Ti240, Ti375	8 GB
Topaz	Tz50, Tz75, Tz100, Tz110, Tz170	8 GB
	Tz200, Tz325	8 GB

Windows Requirements

- Efinity full release or Windows Standalone Programmer: Windows 10 or later, 64-bit operating system
- Microsoft Visual C++ 2022 x64 runtime library (or latest version) redistributable https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170
- Java 64-bit runtime environment; required for configuring some IP cores in the IP Manager (e.g., Sapphire SoC). Efinix has tested the following versions for compatibility:
  - https://www.java.com/en/download/manual.jsp (Java 8)
  - https://developers.redhat.com/products/openjdk/download (OpenJDK 8 or 11)

Note: You may also use other Java software platforms that are available in the market. However, there is a known issue with OpenJDK v24 and higher (see Troubleshooting).

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

Linux Requirements

Supported operating systems:

- Ubuntu v20.04 or later
- Red Hat Enterprise v8.8 or later

Additional software you need to install:

- Libraries:
  - Ubuntu v20.04[3]—**apt install libxcb-cursor0**
  - Red Hat—**yum install xcb-util-cursor**
- Linux X11 or Wayland windowing system (for Efinity® GUI)
- 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 or Red Hat web site to install it. Your path environment variable should include the Java executable.
- Udev device manager for Efinix USB programming cable
  see Installing the Linux USB Driver

[3] The official LTS images for v22.04 and v24.04 include the **libxcb-cursor0** library by default.

Running the Efinity software on Ubuntu in a virtual machine or using Windows WSL requires these additional libraries:

Table 5: Using Efinity and Ubuntu in VM or WSL

Platform	Ubuntu Version	Required Libraries
VM	20.04	**sudo apt install libxcb-cursor0 libnss3 libasound2 libxkbfile1 -y**
	22.04 24.04	No additional libraries required.[3]
WSL v2.3, v2.4[4]	20.04	**sudo apt update** **sudo apt install libxcb-cursor0 libnss3 libasound2 libxkbfile1** **sudo apt install libxcb-xinerama0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 libxcb-shape0 libxkbcommon-x11-0 libegl1 libxdamage1**[5]
	22.04	**sudo apt update** **apt install libxcb-cursor0 libnss3 libasound2 libxkbfile1**
	24.04	**sudo apt update** **sudo apt install libxcb-cursor0 libnss3 libasound2t64 libxkbfile1** **sudo apt install libxcb-xinerama0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0 libxcb-shape0 libxkbcommon-x11-0**[5]

Note: Efinix recommends increasing the memory reservation for your WSL2 machine to avoid degraded performance or out of memory situations. Refer to https://learn.microsoft.com/en-us/windows/wsl/wsl-config#wslconfig.

[4] For the Bitstream Security Key Generator and JTAG SVF Player, you need to set an environment variable. See Table 1.
[5] You can also use the the command **sudo apt install qtwayland5**, however, it installs more libraries than you need, which may not be desired.

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

Tip: *Windows users:* Add the tool paths (`C:\iverilog\bin\` and `C:\iverilog\gtkwave\bin\`) to your system Environment Variables path to ensure the software launches correctly.

Linux installation:

Linux users can use the following commands:

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

Chapter 2

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-ASCII 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
```

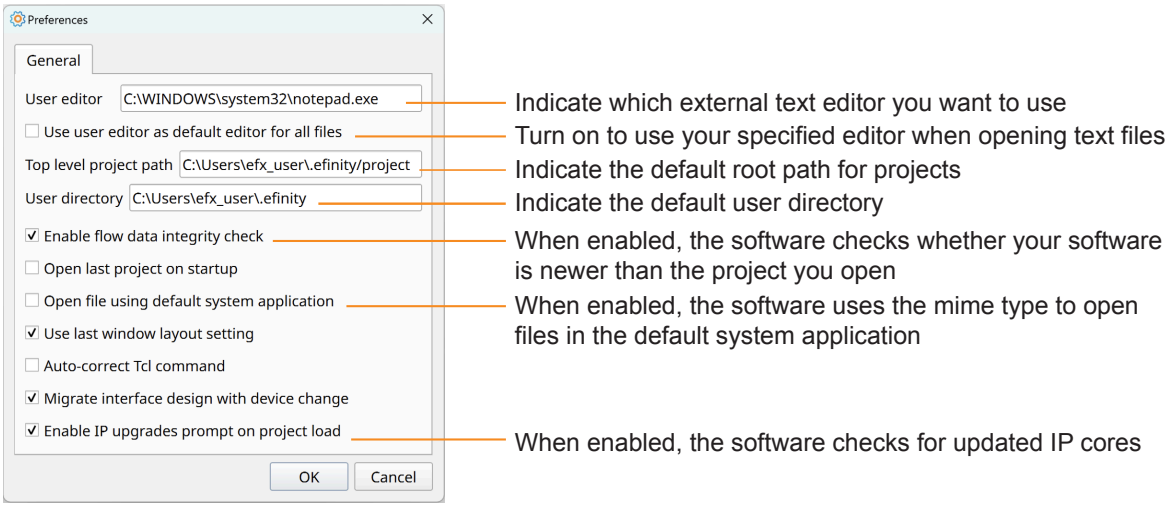
For command-line help:

```
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



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).

Configuring External Text Editors

The Efinity software can launch an external text editor at a specified line. To configure the Efinity software to jump to specified lines in an external text editor, choose **File > Preferences**. In the **Preferences** dialog box, append an option in dual braces ({{ <*option*> }}) to the path in the **User editor** field. The option depends on your external text editor:

- gedit and GVim (Linux): **/usr/bin/gedit {{+%n}}**
- Notepad++ (Windows): **C:\Program Files\Notepad++\notepad++.exe {{-n%n}}**
- Visual Studio Code: **/usr/bin/code {{--goto}} {{%f:%n}}** or **/usr/bin/code {{-g}} {{%f:%n}}**

The supported options include:

- **%n**: Line number.
- **%f**: File name (Optional. Only required for Visual Studio Code when using the `--goto` option).

To open a specified line in your external code editor:

1. Open the Code Editor in the Efinity software. Click **Analyze** on the taskbar in the Code Editor.
2. Double click the row number for an alert in the **Message Browser** panel. The Efinity software will open the project file in your external editor at the specified line.

Setting User and Project Directories

Historically, the Efinity software saved all files, including project files, in the Efinity installation directory by default. Beginning with the Efinity software v2025.1, the default location is in your user directory:

- *Linux*—**/home/** < *user name* > **/.efinity**
- *Windows*—**C:\Users**\ < *user name* > **\.efinity**

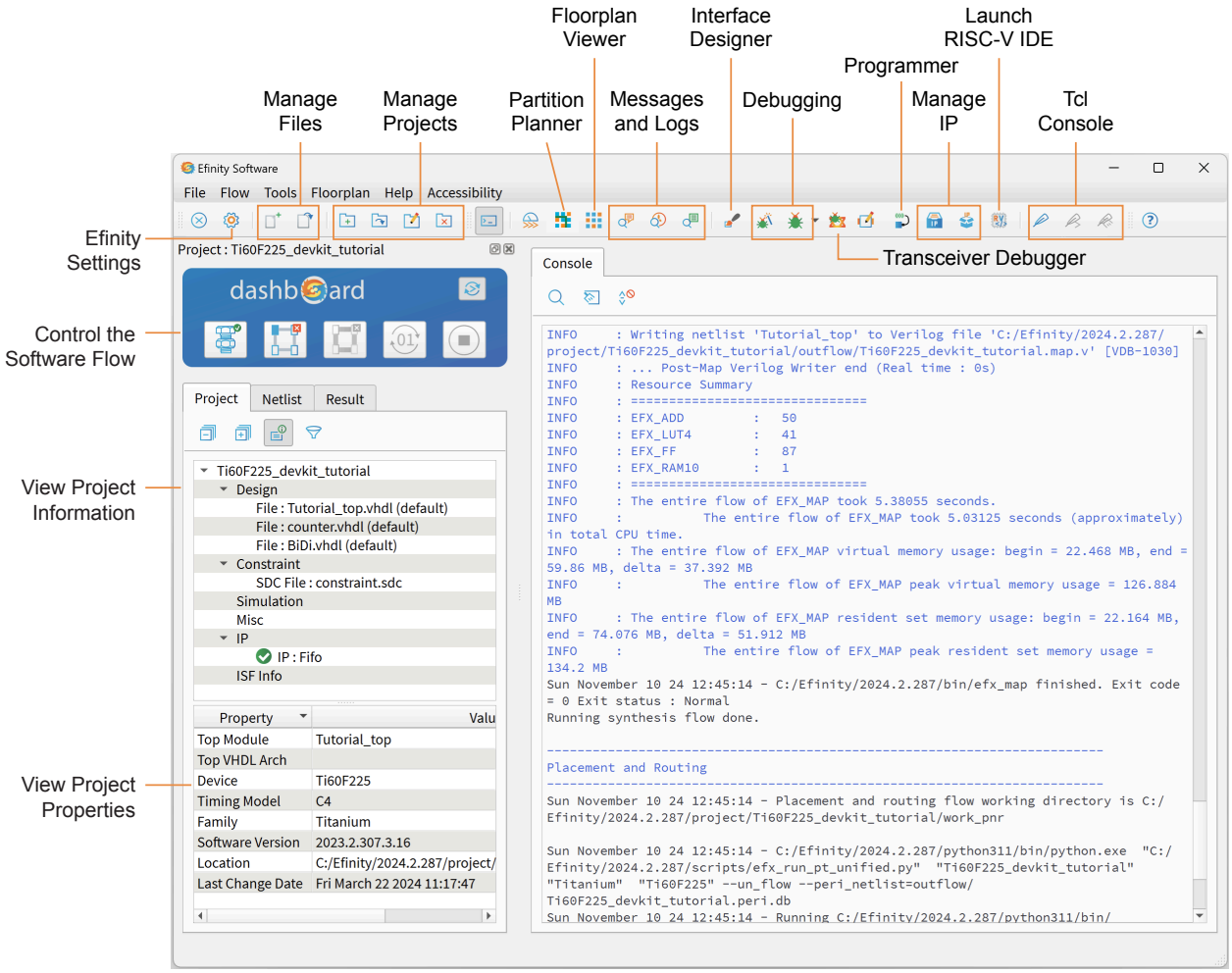
This change makes it easier to manage projects if you have multiple versions of Efinity software installed. The **.efinity** directory is your default project directory and also stores log files and **.ini** files for various tools (IP Manager, IP Packager, Interface Designer, Debugger, Programmer, etc.).

Note: To change the default location, choose **File** > **Preferences**. In the **Preferences** dialog box, change the **Top-level project path** and **User directory** fields for the new path. The Efinity software prompts you to restart the software after you change the path(s).

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 3

Managing Projects

Contents:

Project Editor

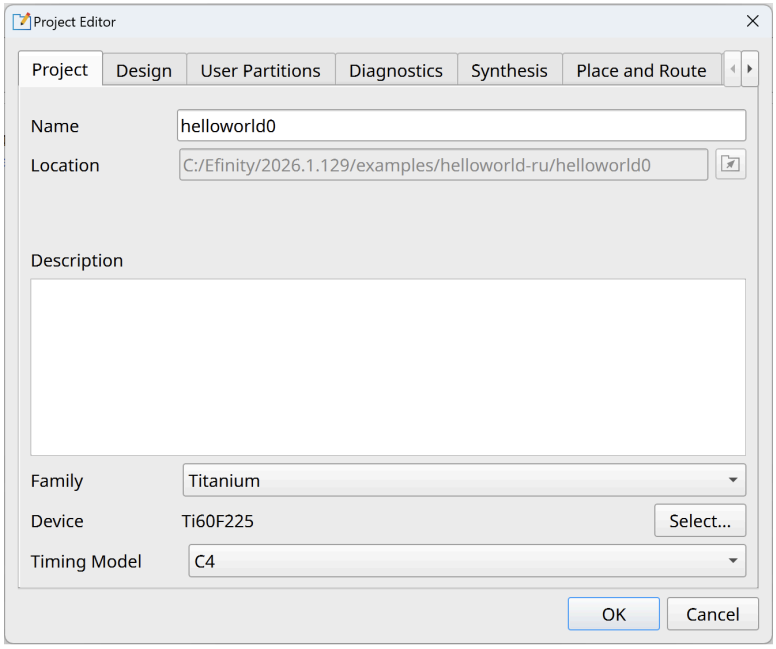
You use the Project Editor to create or modify a project, add files to your project (such as timing constraint files), and choose a device family and device. To create a new project, choose **File > Create Project** or click the Create Project icon.

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:

- *Linux*—**/home/** < *user name* > **/.efinity**
- *Windows*—**C:\Users** < *user name* > **\.efinity**

Figure 4: Project Editor - Project Tab



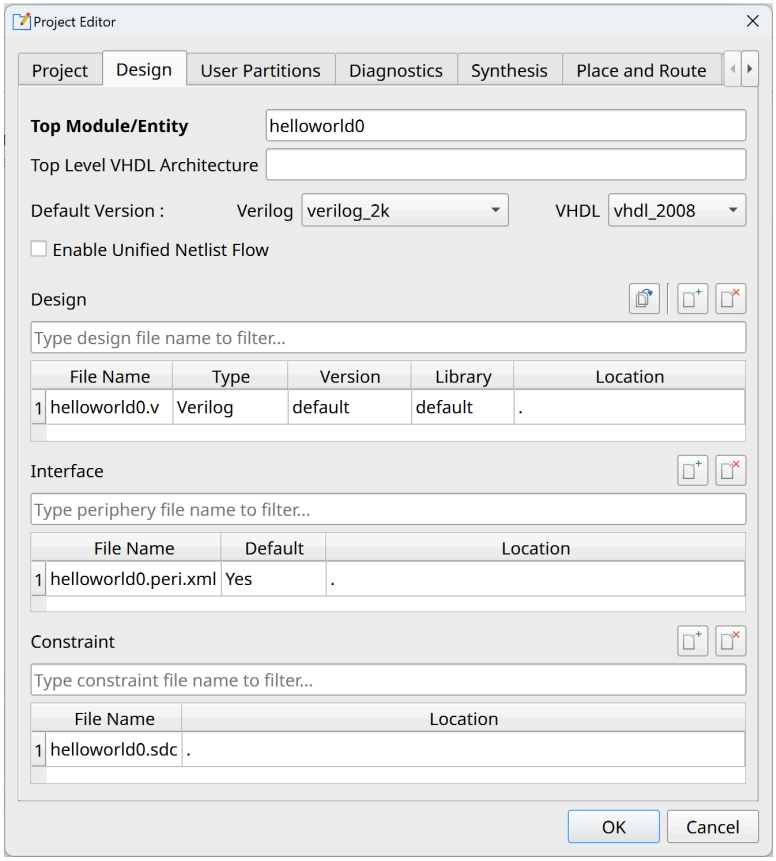
Design Tab

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



Turn on the **Enable Unified Netlist Flow** to let the Efinity software generate most interface blocks for you from the data it interprets from your RTL design. You use this option with the **--peri-syn-inference** and **--peri-syn-instantiation** synthesis options. If you turn on **Enable Unified Netlist Flow** and do not turn on **--peri-syn-inference** or **--peri-syn-instantiation**, the software issues an error message.

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 31 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).

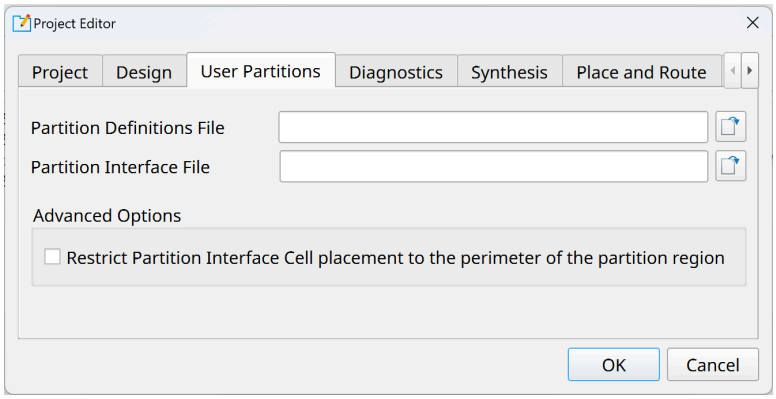
Optionally, in the **Constraint** section you can specify a Synopsys Design Constraints (**.sdc**) file for timing-driven compilation and an Interface Scripting File (**.isf**) that contains all of the Python API commands to re-create your interface.

Learn more: Refer to Analyzing Timing on page 90 and the Efinity Timing Closure User Guide for more information on timing analysis.

User Partitions Tab

The Efinity software v2026.1 and higher has a Partition Planner used for incremental compilation, team-based design, and design re-use. In the **User Partitions** tab, you specify partition-related design files and other options. Refer to Partition Planner (Beta) on page 67 for details.

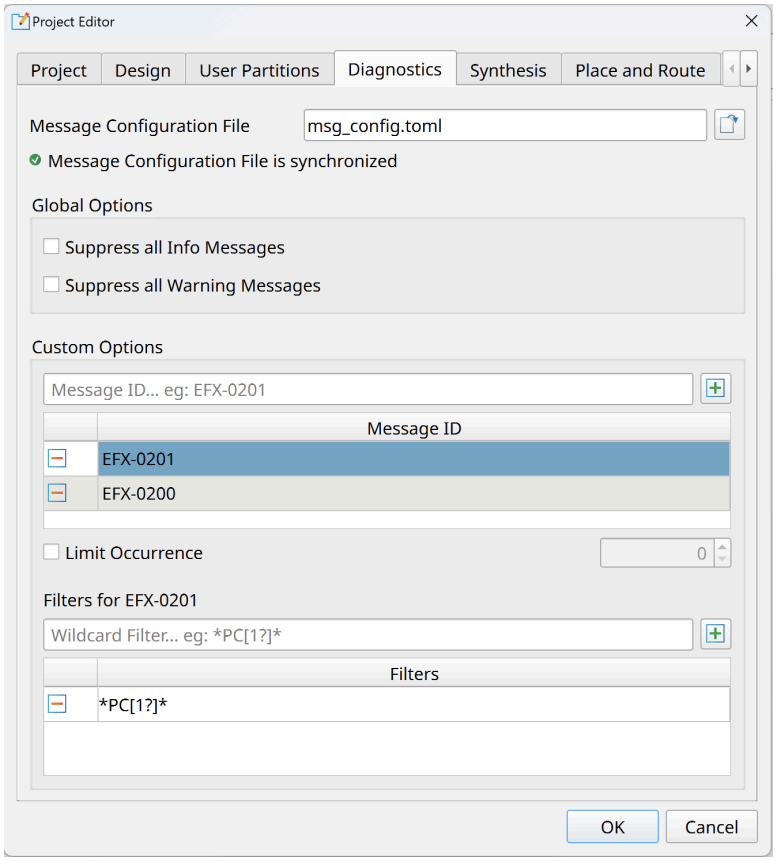
Figure 6: Project Editor - User Partitions Tab



Diagnostics Tab

You use the settings in the **Diagnostics** tab to control how the **Console** displays messages. You can suppress specific message types and/or filter based on message ID numbers. Refer to Supressing Console Output on page 42 for details.

Figure 7: Project Editor - Diagnostics Tab



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 8: Project Editor - Synthesis Tab

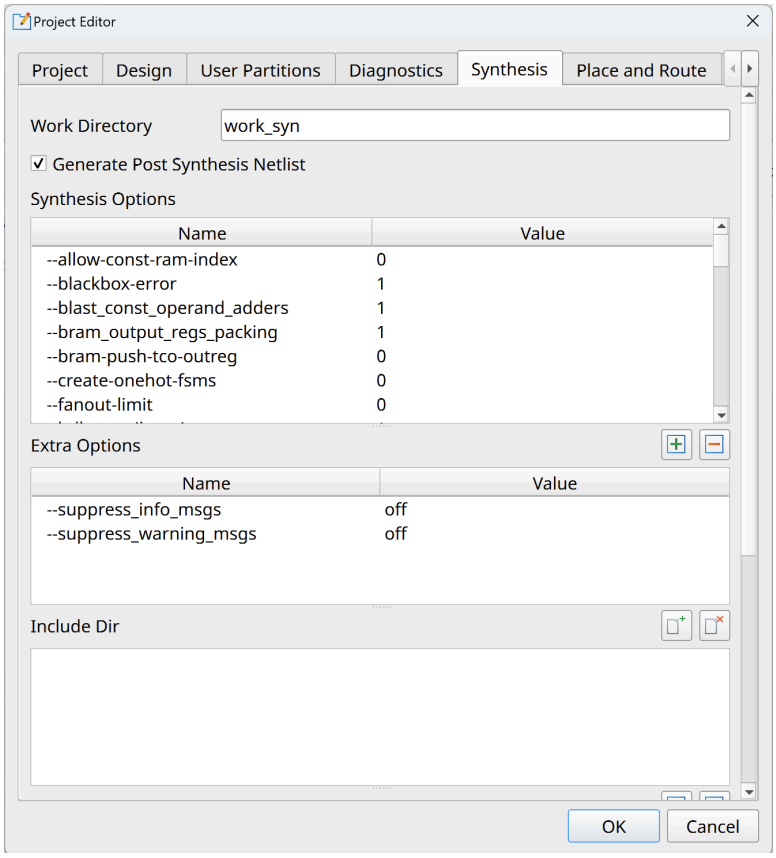


Table 6: Synthesis Project Settings

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
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.

Setting	Description
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

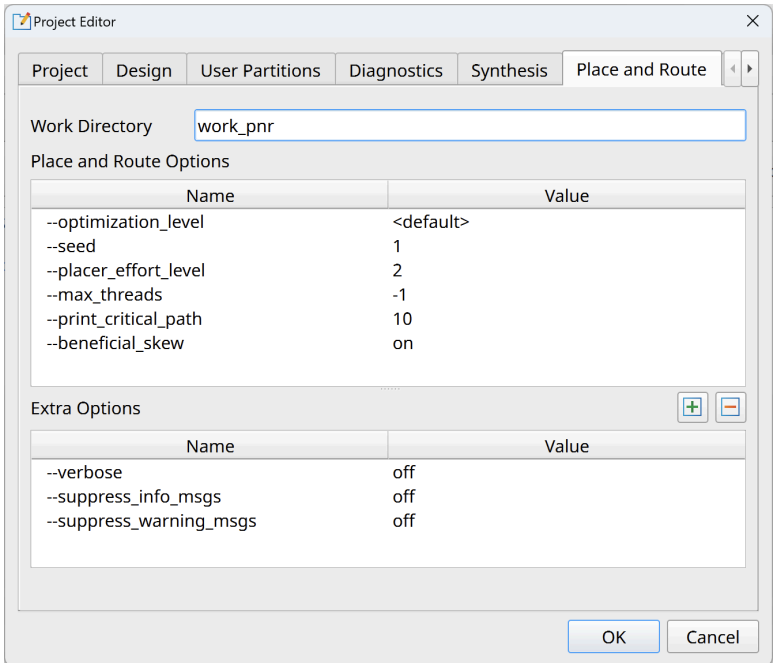
Optional. The options on this tab let you specify project-specific preferences to help close timing. If you do not make any settings, the tool uses the defaults.

- 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 message suppression options (`--suppress_info_msgs`, `--suppress_warning_msgs`, and `--msg_suppression_list`) suppress INFO and WARNING messsages from place and route.

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

Figure 9: Project Editor - Place and Route Tab

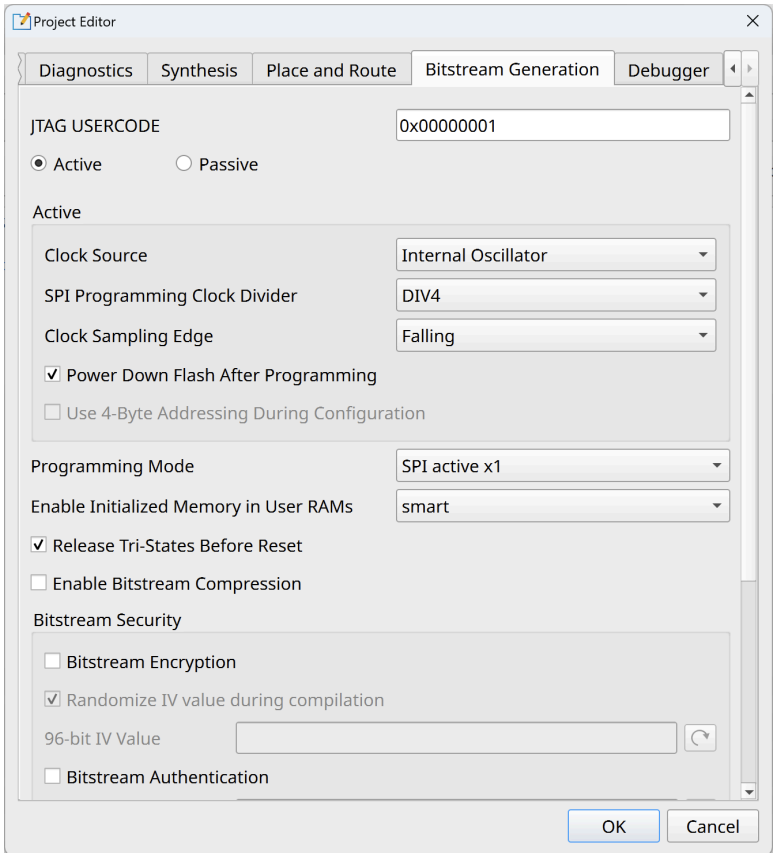


Learn more: Refer to "Place and Route Options" in the for more details on these options and how to optimize timing.

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 10: Project Editor - Bitstream Generation Tab

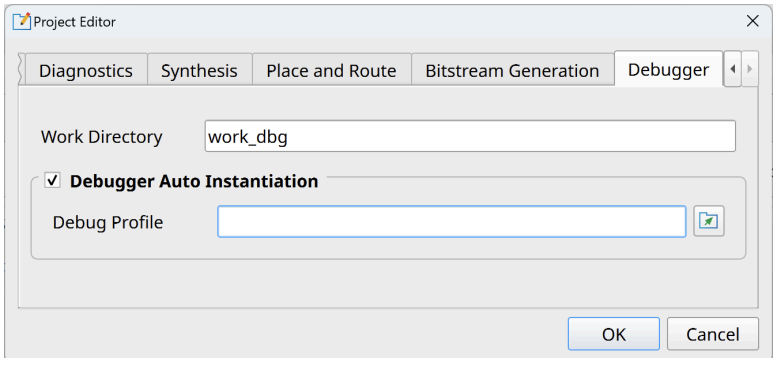


Learn more: Refer to Project-Based Programming Options on page 163 for more information on bitstream and programming settings.

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 11: Project Editor - Debugger Tab



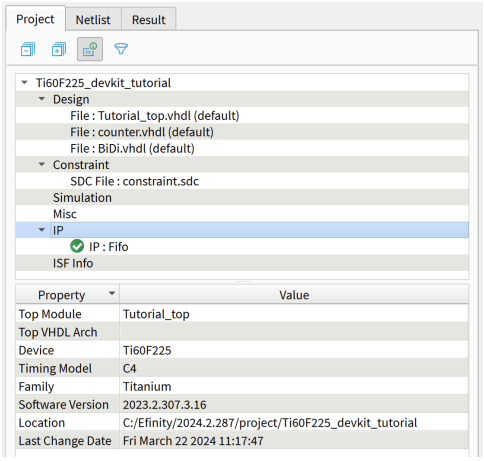
Learn more: Refer to Debugging on page 99 for more information.

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 its 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 12: 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 55 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

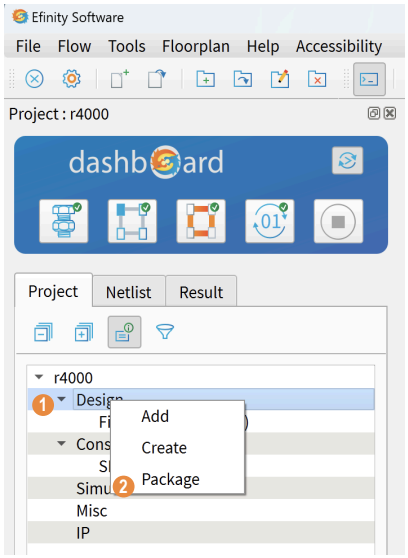
Packaging Design Files

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 13: 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 Efinity 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.

Chapter 4

Running the Tool Flow

Contents:

The Efinity software supports GUI and command line tool flows.

Run the Flow with the Dashboard Controls

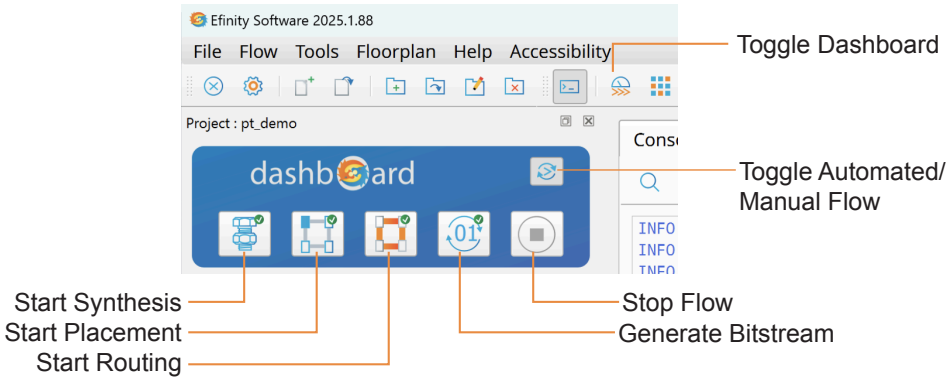
The Dashboard controls the software flow, which operates in two modes: automated and manual. Toggle automated and manual flows using the toggle button.

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 14: 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 41.

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)

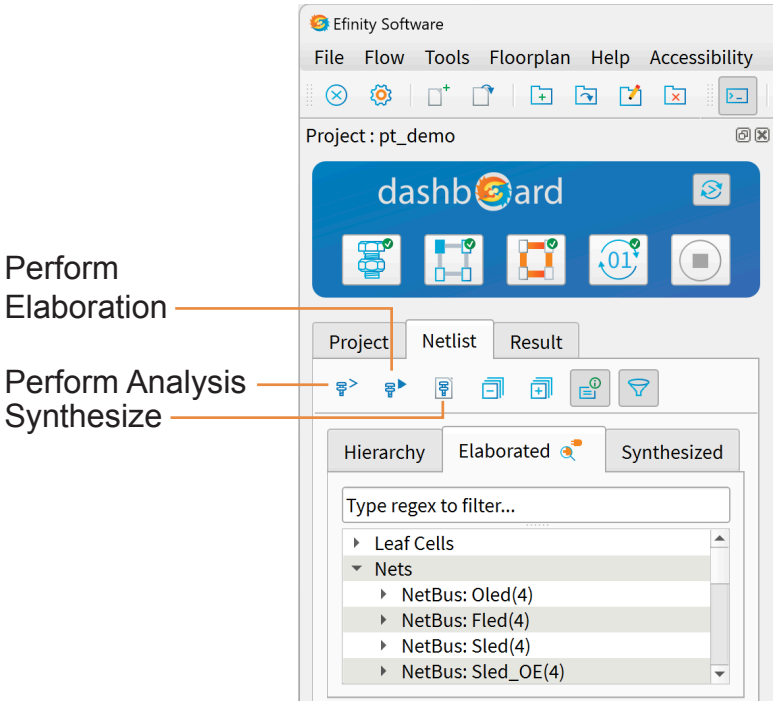
Learn more: Refer to the **Efinity Synthesis User Guide** for more information on synthesis options and attributes as well as design guidelines.

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 15: 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.)

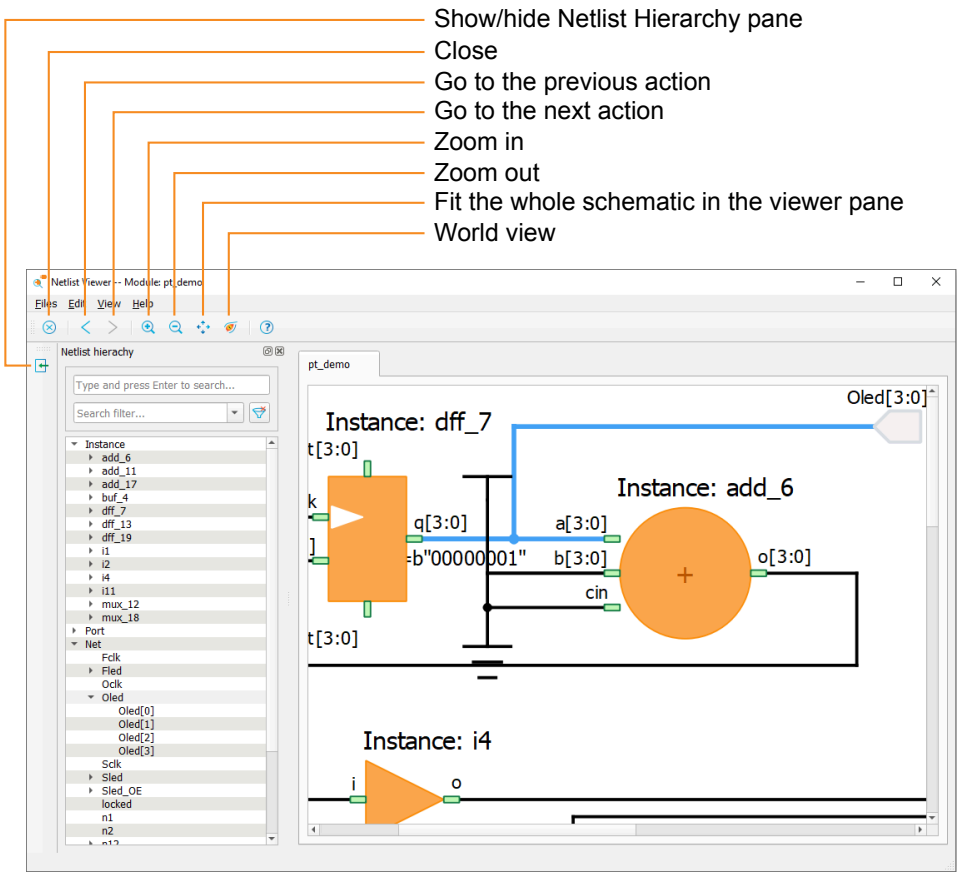
Important: The Netlist Viewer is beta in the Efinity software v2023.1.

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

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

Figure 16: Netlist Viewer

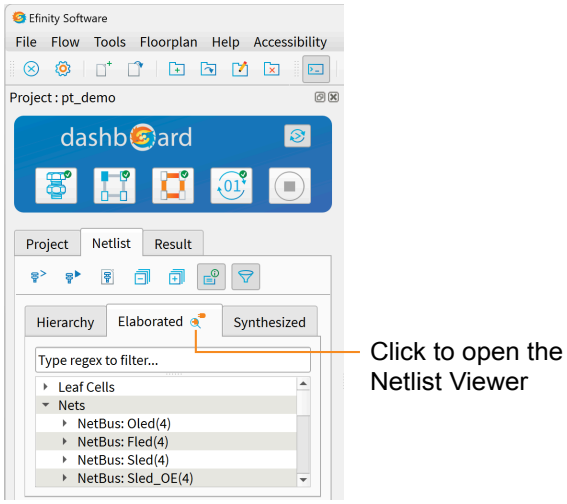


Opening the Netlist Viewer

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 17: Opening 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.

Note: You can also highlight or mark design elements and then zoom to them. See Highlighting and Marking on page 39.

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.

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.

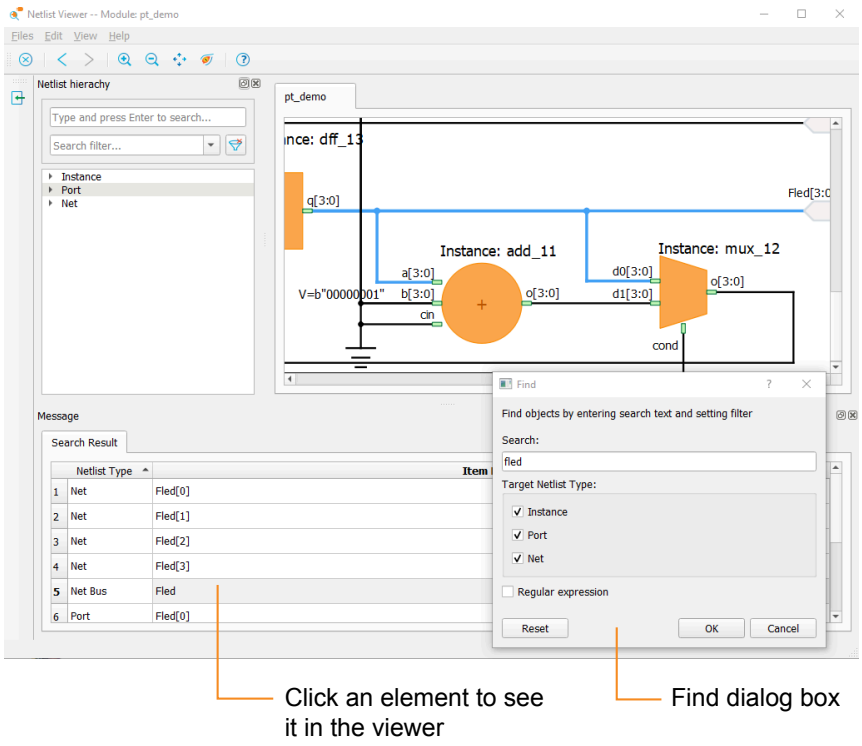
Finding Elements

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.

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

Figure 18: Finding Elements



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 highlighed 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 90 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.

Supressing Console Output

During compilation, the Efinity software generates (a lot) of messages and displays them in the Console. You may want to suppress some of these messages to make it easier to see the ones you are most intersted in. With the Efinity software v2026.1 and higher you can suppress Console output messages using the GUI or a message configuration file (**.toml**).[6]

Message suppression only affects messages printed in the Console. Synthesis and place-and-route log files always contain the full compiler output.

Many messages have ID numbers in brackets []. For example, the following message has ID `EFX-0200`.

```
WARNING : Removing redundant signal : rd_underflow. (Fifo.v:998) [EFX-0200]
```

Important: You can only filter messages with ID numbers.

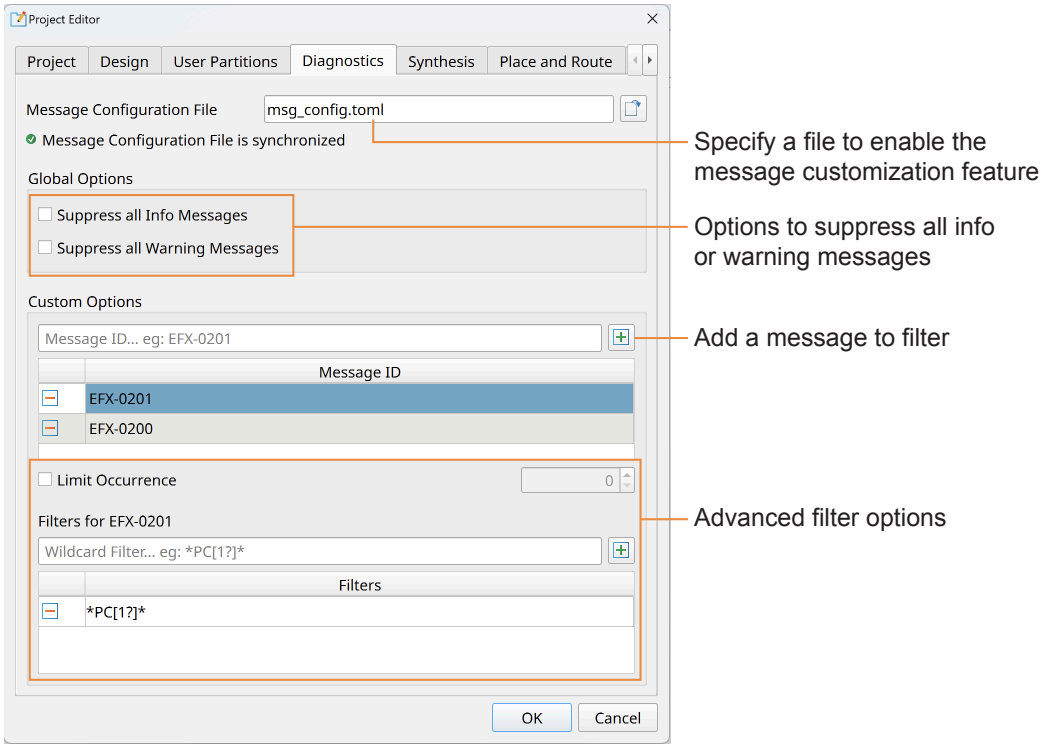
Supressing with the GUI

You filter the Console output using the **Diagnostics** tab in the **Project Editor**.

1. Choose **File > Project Editor** or click the Project Editor toolbar icon.
2. Click the **Diagnostics** tab.
3. Enter the **Message Configuration File** name to enable the feature.
4. *OR*, click the Select Message Configuration File button to open the **Select Message Configuration File** dialog box. Click **Open** to use the default (**msg_config.toml**).
5. If you want to suppress all info or warning messages, turn on those options.
6. (Optional) Suppress specific messages.
   - Enter the message ID in the message ID box under **Custom Options** and click the Add Custom Message Rule button.
   - (Optional) Limit how many times the message displays by turning on **Limit Occurance** and choosing a number.
   - (Optional) Filter out messages that match the message ID based on wildcard strings. Supported wildcard tokens include:
     - * Matches zero or any number of placeholder character(s).
     - ? Matches exactly one placeholder character.

[6] See **https://toml.io/en/** for the TOML syntax.

Figure 19: Diagnostics Tab



Supressing with a File

You can create the message configuration file manually using a text editor instead of specifying the options and filters in the GUI. The following table shows the syntax.

Table 7: Message Configuration File Syntax

Type	Setting	Options	Description
Global	info.suppress	true false (default)	Show/hide info messages.
	warning.suppress	true false (default)	Show/hide warning messages.
Message specific	suppress	true, false	Hide all messages with the specified tag name. This setting is the same as using filter-out = [*]
	filter-out	Expression	Wildcard filter. *–Matches zero or any number of placeholder character(s). ?–Matches exactly one placeholder character.
	limit	Integer	The number of messages of this type to show.

Note: Efinix recommends that you always surround the message ID with quotes (" ") to avoid incorrect TOML parsing.

The following code shows an example configuration message file:

```
[global]
info.suppress    = true   # Info message setting; defaults to false
warning.suppress = false  # Warning message setting; defaults to false
```

```
["FlowMsg::InvalidOptimizationLevel"]
suppress  = true   # Equivalent to filter-out = [ * ]

["VERI-1209"]
filter-out = [ *eth_tx.v*, *cntrl.v ]
limit      = 50     # Defaults to unlimited
```

After you create the message configuration file, add it to your project.

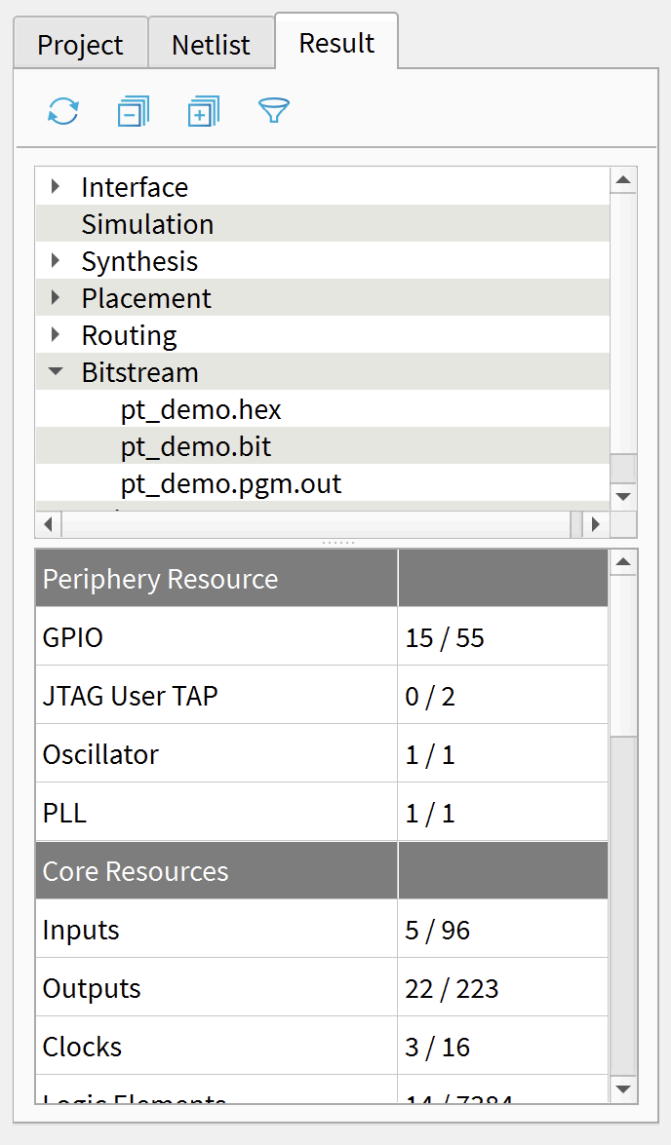
1. Choose **File > Project Editor > Diagnostics tab**.
2. Click the Select Message Configuration File button to open the **Select Message Configuration File** dialog box.
3. Browse to your file and click **Open**.

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 18).

Figure 20: 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 273 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 8: 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.

Category	File	Description
	<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 9: Interface Designer Files

The Interface Designer generates these files when you click the Generate Interface Output 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.
<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.

File	Description
<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.

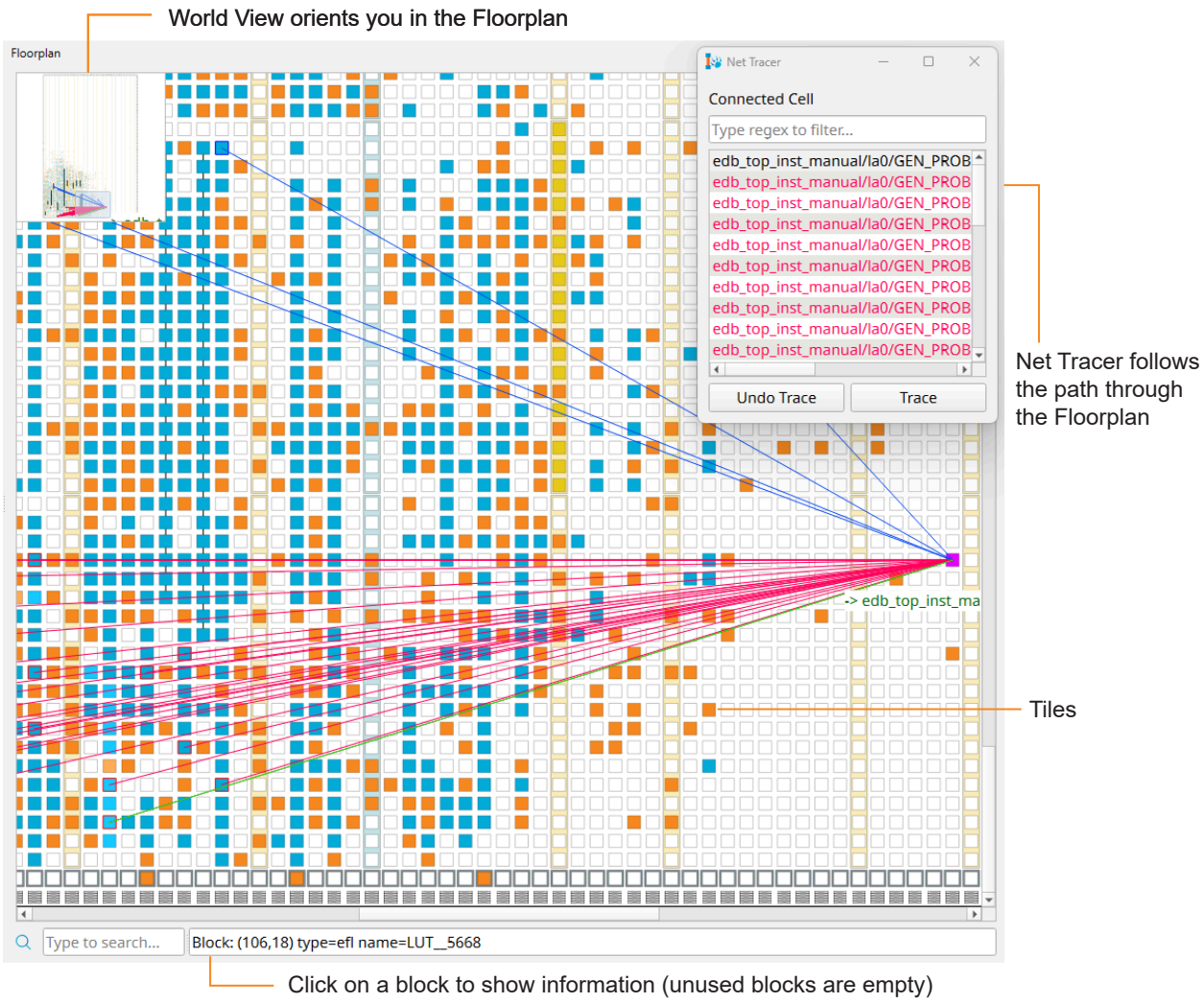
- The Console displays messages generated during compilation. For example, if the design has too many I/O pins to fit in the target device, compilation will stop and the Console will show the error message.
- The Result pane shows the output and report files for each stage in the flow. Additionally, 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.

  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 21: Floorplan Editor



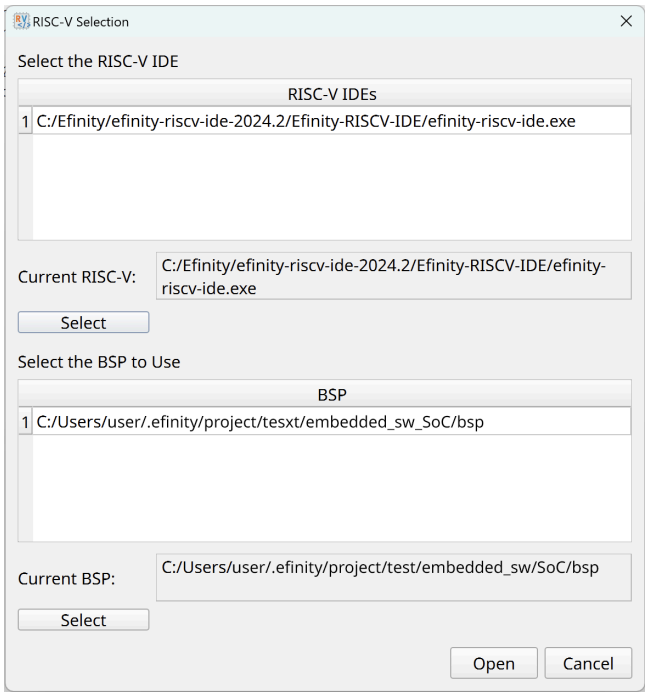
Learn more: Refer to the Efinity Trion Tutorial for more information on how to use the Floorplan Editor.

Efinity RISC-V Embedded Software IDE

The Efinity RISC-V Embedded Software IDE is an Eclipse-based Integrated Development Environment (IDE) powered by Ashling's *RiscFree*™ IDE. The IDE is a separate package that you can download from the Supprot Center.

In the Efinity software v2025.1 and higher you can launch the IDE from within the Efinity GUI. Click the Open RISC-V IDE button in the main toolbar to open the **RISC-V Selection** dialog box.

Figure 22: RISC-V Selection Dialog Box



The dialog box shows the IDE versions you have installed, and it looks in your project for any available BSPs. If you want to use one that is not listed, click **Select** to browse for a different IDE version or BSP.

Chapter 5

Using the IP Manager

Contents:

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 when you use several IP cores, multiple instances of an IP core with different parameters, or the same IP core across 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.

Tip: Refer to Resolving IP Manager Issues on page 57 if you get an RPC server error when launching the IP Manager.

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 10: 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_2	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	
PCIe Scatter-Gather Direct Memory Access		All	Ti85, Ti135, Ti165, Ti240, and Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
PMA 64b66b GearBox		All	Ti85, Ti135, Ti165, Ti240, and Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
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
USXGMII Multirate Ethernet Core		All	Ti85, Ti135, Ti165, Ti240, Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
Foundation IP						

IP Core	Trion		Titanium		Topaz	
	Supported	Not Supported	Supported	Not Supported	Supported	Not Supported
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
PRBS Generator/Checker		All	All		All	
Memory						
BRAM Wrapper	All		All		All	
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 to SPI Flash	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 RX Controller 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	

IP Core	Trion		Titanium		Topaz	
	Supported	Not Supported	Supported	Not Supported	Supported	Not Supported
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		All	
Sapphire High-Performance SoC		All	Ti85, Ti135, Ti165, Ti240, Ti375	All others	Tz75, Tz100, Tz200, Tz325	All others
Sapphire RV64 SoC	All except T4 and T8	T4, T8	All		All	
Serial Interface Protocols						
I^2C	All		All		All	
UART	All		All		All	
JTAG	All		All		All	

Table 11: End of Life IP Cores by Family

IP Core	End of Life in Version	Replaced By	Trion		Titanium		Topaz	
			Supported	Not Supported	Supported	Not Supported	Supported	Not Supported
Divider	2025.2.288.4.15	Divider_2	All		All		All	
JTAG SPI Flash Loader	2025.1	JTAG Bridge	All		All			
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.

In Efinity software v2025.2 and higher, the IP Manager can generate synthesized files for out-of-context (OOC) synthesis flows. See "Out-of-Context Synthesis" in the Efinity Synthesis User Guide for more details. The generated OOC files are:

- <*module name*>**.ooc.vdb**—Synthesized IP core.
- <*module name*>**.stub.v**—Defines the boundaries of the synthesized circuit. The software uses this file when synthesizing your top-level design that includes the OOC archive.

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.

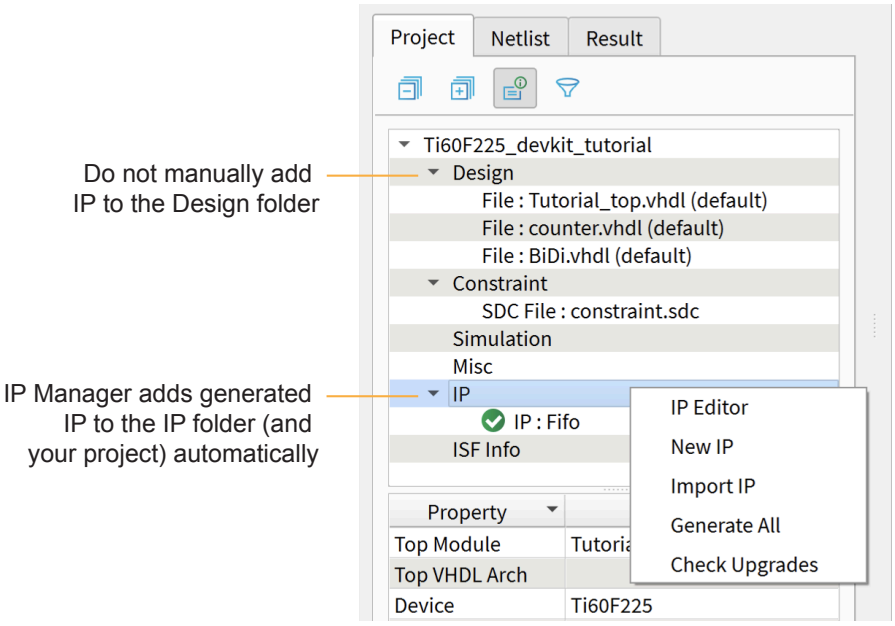
Instantiating IP in Your Project

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

- *<module name>*tmpl.v is the Verilog HDL module.
- *<module name>*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.



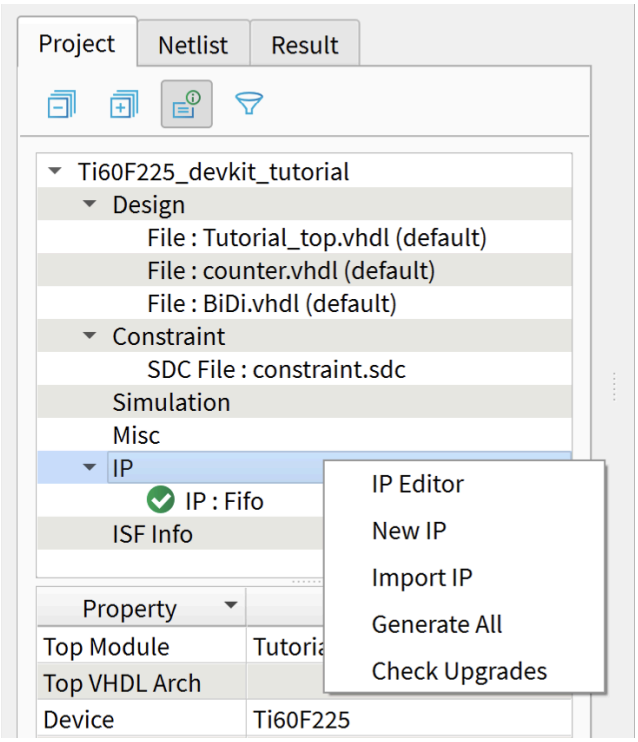
Managing IP in Your Project

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 57 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 18.

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 23: 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.

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.

Resolving IP Manager Issues

When you launch the Efinity software, it creates two server instances on localhost for the IP Manager. The IP Manager uses these server instances for inter-process communications. These server instances use randomly chosen port numbers based on preset upper and lower bounds. The server instances are:

- IPM_Server = 127.0.0.1:<*port number*>
- RPC_Server= 127.0.0.1:<*port number*>

If your computer has closed the default port ranges (for example, your IT department has closed ports for security reasons), the two RPC server instances cannot connect to localhost and the software issues the message **IP Manager RPC Server Not Connected**.

To resolve this issue you need to:

1. Talk to your IT department about opening ports for the IP Manager.

2. Set specific default ports for the software to use instead of a range (so the IT department only has to open 2 ports).

The software defaults to using the port range 49152 – 65535 because these ports are dynamic, private, or ephemeral ports. They are open for any application to use and are not reserved by the Internet Assigned Numbers Authority (IANA).

To change the default port range to specific ports:

1. In a text editor, open the **app_session.ini** file, which is located in the Efinity **bin** directory (<*install directory*>**/bin**). The contents are:

```
[config]
ipm_server_port_lower_bound = 49152
ipm_server_port_upper_bound = 65535
efxg_rpc_port_lower_bound = 49152
efxg_rpc_port_upper_bound = 65535
```

2. Change the configuration to use a single port by changing the upper and lower bound numbers. The software selects ports using the scheme *port_lower_bound* < = *port_number* < *port_upper_bound*. Therefore, set the lower bound as the port you want to use and the upper nound as the next higher number; the software uses the lower bound number for the port setting. For example:

```
[config]
ipm_server_port_lower_bound = 53123
ipm_server_port_upper_bound = 53124
# Software uses port number 53123

efxg_rpc_port_lower_bound = 49153
efxg_rpc_port_upper_bound = 49154
# Software uses port number 49153
```

3. Save the **app_session.ini** file and then re-launch the Efinity software.

Chapter 6

Constraining Logic and Assigning Pins

Contents:

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 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.

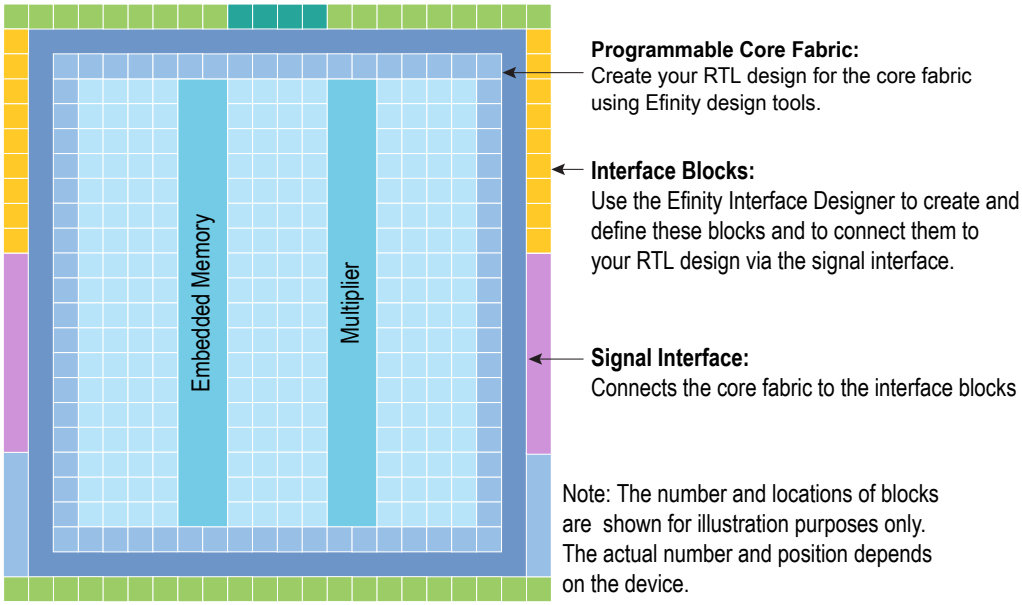
The describes how to create an interface design using scripting.

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 24: Conceptual View of Interface Blocks

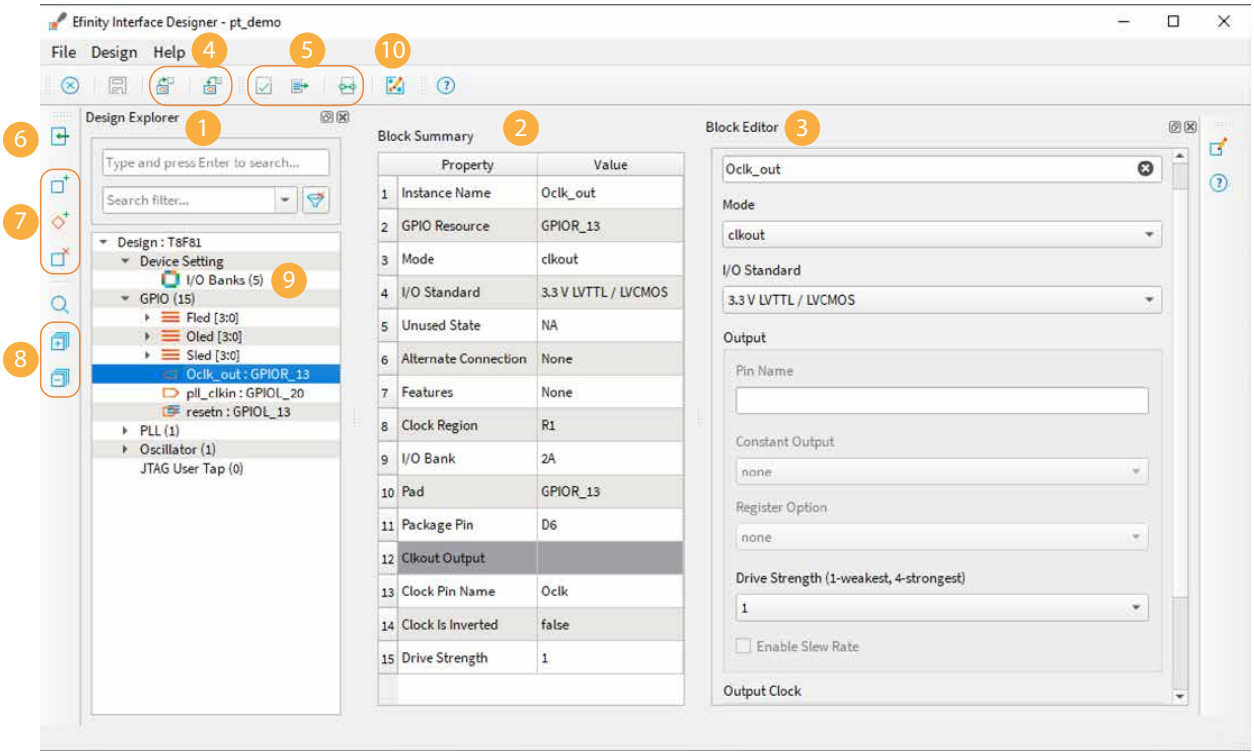


Get Oriented

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 25: Interface Designer

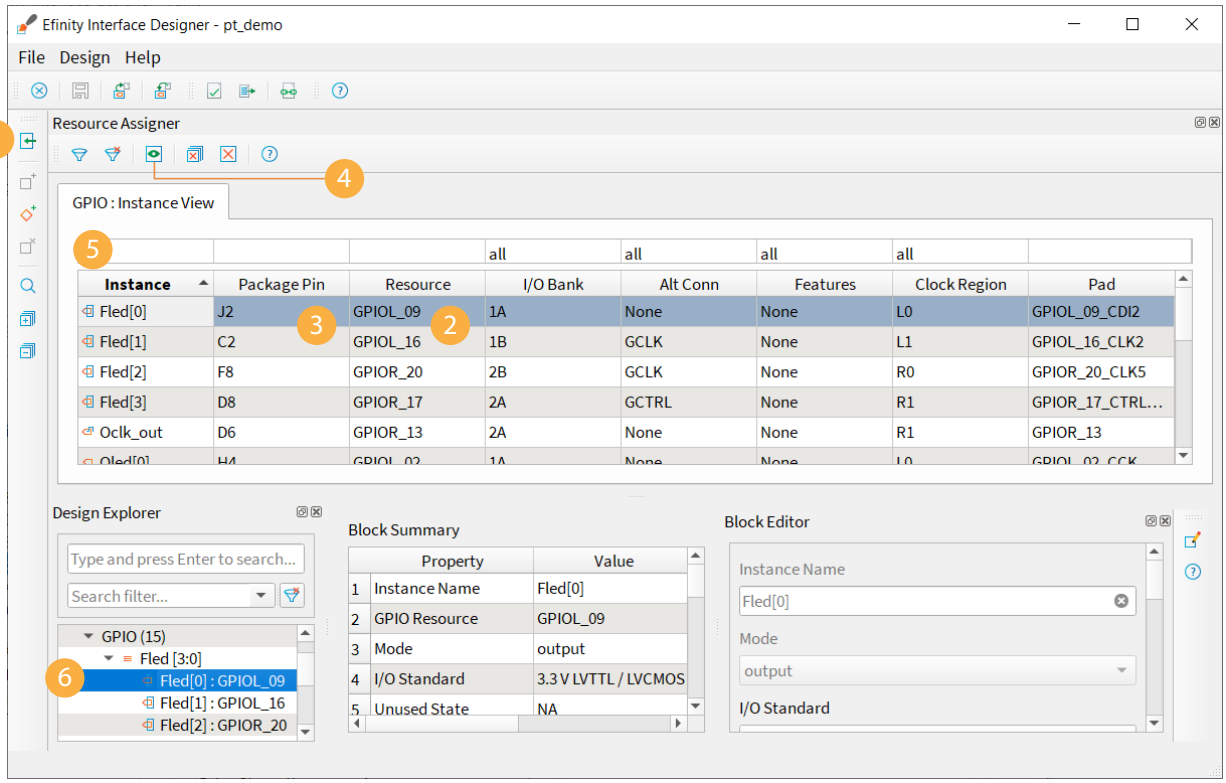


Notes:

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 26: 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.

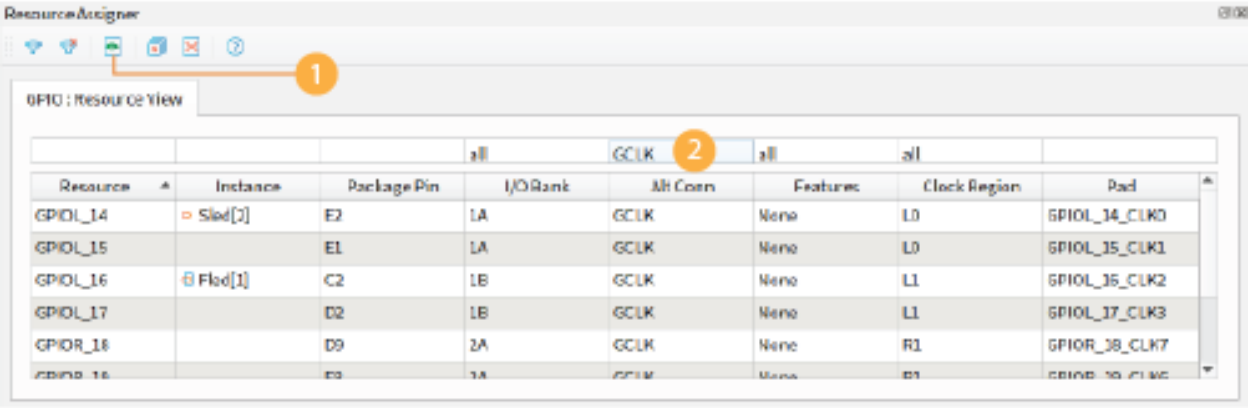
Note: Titanium: When using HSIO pins as GPIO, make sure to leave at least 1 pair of unassigned HSIO pins between any GPIO and HSIO pins in the same bank. This separation reduces noise. The Efinity software issues an error if you do not leave this separation.

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 27: 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

With the Efinity software v2025.1 and higher you can add an **.isf** to your project in the **Project Editor > Design tab**.

Learn more: For help understanding messages, refer to the "Design Check" topics in the interfaces user guides. 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 12: 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		
...		
(7)	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.(8) 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

(7) Unassigned instances have a blank field for the Package Pin-Pad Name column.
(8) 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: For more information on using the Python API to script an interface, refer to the Efinity Interface Designer Python API.

Partition Planner (Beta)

A design partition represents a portion of the RTL hierarchy located in a specific area in the device floorplan. Partitions ensure the quality and reproducibility of logic placement, routing, and timing results[9] across project and/or design iterations. Additionally, they enforce floorplanning requirements with placement constraints. Partitions support the following workflows:

- Team-based design (top-down methodology) where multiple users develop different parts of the design in parallel, reducing the overall development time.
- Design re-use (bottoms-up preservation flow) in which you use partitions across projects and/or designs.
- Incremental compilation where you re-use compilation results thereby reducing compile times across project and/or design iterations.

The Efinity Partition Planner, which is beta in v2026.1, allows you to create and use partitions to support these workflows. As with a non-partition-based design, you should have a good idea of the design's interface and I/O requirements at the beginning of the design process. Ideally, you define partitions after determining the design interface.

Note: The Efinity software supports partitions for Titanium and Topaz FPGAs.

What Is a Partition?

A partition is a portion of your design hierarchy that is assigned to a specific area in the device floorplan. Each partition has a unique name made up of alphanumeric characters plus hypens (-) and underscores (_), and has a *bounding box*, which is a user-specified rectangular area in the floorplan.

Note: Partition names cannot be empty strings. `RootPartition` is a reserved name used for the portions of the design that are not assigned to a partition. You cannot constrain the root partition.

Table 13: Partition Types

Type	Description
Region	The partition defines a placement region constraint. The software places the specified XLR cells within the specified partition's bounding box. However, the routing paths between the partition's XLR cells can go outside the partition boundary.
Export	You can export this partition type for re-use in a subsequent compile or project. The software constrains the partition's XLR cells within the specified bounding box. The software also constrains the routing paths between the partition's XLR cells within the bounding box. The corresponding RTL module's input and output ports become the partition interface (or ports). This interface is the naming convention the software uses to merge the netlists when you later import the partition to a new design. You can manually place the partition interface using the **Partition Interface Assignments** tool.

[9] The software preserves timing results as long as you use the same timing constraints across projects/designs.

Type	Description
Import	You can import a previously exported partition into the specified design hierarchy. The software merges the imported netlist at the given hierarchy by matching partition interface (port) names. The software removes all existing RTL hierarchy from the destination hierarchy and replaces it with the imported partition. Imported partitions have the same boundng box as the originally created partition. When you import a partition, the software preserves the partition's XLR cell placement and routing from the previous compilation. Assuming your design uses the same timing models and project settings as the imported partition, the software preserves the timing characteristics of cell-to-cell paths from the previous compilation. Additionally, if your design's clock settings and timing constraints are the same as the imported partition, the software preserves the timing delays of clock pin to clock ports on imported XLR cells from the previous compilation. For imported partitions, you specify the filename and path of a previously exported partition (**.part**).

Hierarchy Path

A hierarchy path is the sequence of RTL instance names that uniquely identify a design sub-hierarchy. For example, given the following Verilog HDL design:

```
module Top(a, b, c, d, o);
    input a, b, c, d;
    output o;

    And4 and4(.a(a), .b(b), .c(c), .d(d), .o(o));
endmodule

module And4(a, b, c, d, o);
    input a, b, c, d;
    output o;

    wire ab;
    wire cd;

    And2 and2_ab(.in1(a), .in2(b), .o(ab));
    And2 and2_cd(.in1(c), .in2(d), .o(cd));
    And2 and2_abcd(.in1(ab), .in2(cd), .o(o));
endmodule

module And2(in1, in2, o);
    input in1, in2;
    output o

    assign o = in1 & in2;
endmodule
```

- The empty hierarchy path `""` refers to the entire design.
- The hierarchy path `"and4"` includes all logic contained within the single instantiation of the `And4` module (`and2_ab`, `and2_cd`, `and2_abcd`) as well as any associated wires/nets and primitives.
- The hierarchy path `"and4|and2_cd"` includes all wires/nets and primitives required to implement the AND operation between inputs `c` and `d`.

You cannot create a partitions with the top hierarchy path `""` or reference Efinity primitives , e.g., `"and4|and2_cd|LUT123"`.

For import and export partitions, the software uses the hierarchy module's top-level input and output ports as the interface. The software maps the partition interface to `EFX_LUT4` primitives during sythesis. In the previous example, the partition for `and4|and2_cd` has `in1` and `in2` (inputs) and `o` (output) interface ports.

Exclusive Partitions

If a partition is *exclusive*, the software cannot place unconstrained XLR cells from the root partition within the partition's bounding box. However, even if a partition is exclusive, the software can route other connections through the partition.

Exclusive regions may improve design routability at the cost of reduced performance and reduced device resources available for XLR cells in the root partition.

Note: Efinix does not recommend making a partition exclusive unless you experience routing difficulties. By default, partitions are not exclusive to maximize resource availability.

When you import a partition, you can choose whether is it exclusive or not, even if the original partion was exclusive.

Size and Placement

In Titanium and Topaz FPGAs, the regional clock regions are arranged in rows that are 80 XLR cells tall. Therefore, a partition's bounding box must be in increments of 80 rows and align with the regional clock boundaries. When you draw a partition's bounding box in the floorplan, the software restricts the box dimensions and placement by snapping to the closest legal XLR cell on the Y-axis.

Partition bounding boxes can touch, but they cannot overlap. Additionally, partition hierarchies cannot overlap. For example, using the code in **Hierarchy Path** on page 68, you cannot have a partition for `and4` and `and4|and2_cd` simultaneously.

Working with Imported Partitions

For team-based design or design re-use you import a **partition output file (.part)** into the Partition Planner. An imported partition is compatible with the assigned design hierarchy path if all partition interface port names and directions (input/output) match. However, the RTL module names and hierarchy paths do not need to match. For example, you can import a partition from `top|FifoEXP` (module `Fifo_v1.0`) into `Top|Perip|FIFOImp` (module `fifo_stub`) as long as the interfaces match.

When importing a partition, the software only looks at the interface ports of the assigned design hierarchy. The software ignores the module's contents. To make the import process easier, Efinix recommends that you define *stub* module instances in your RTL design as targets for imported partitions as shown in the following code example.

```
module top(a, b, o);
    input a, b;
    output o;

    wire imp_a, imp_b;
    wire imp_o;

    ...
    my_stub imported(.a(imp_a), .b(imp_b), .o(imp_o));
    ...
endmodule

module my_stub(a, b, o);
    input a, b;
    output o;
    // No internal logic
endmodule
```

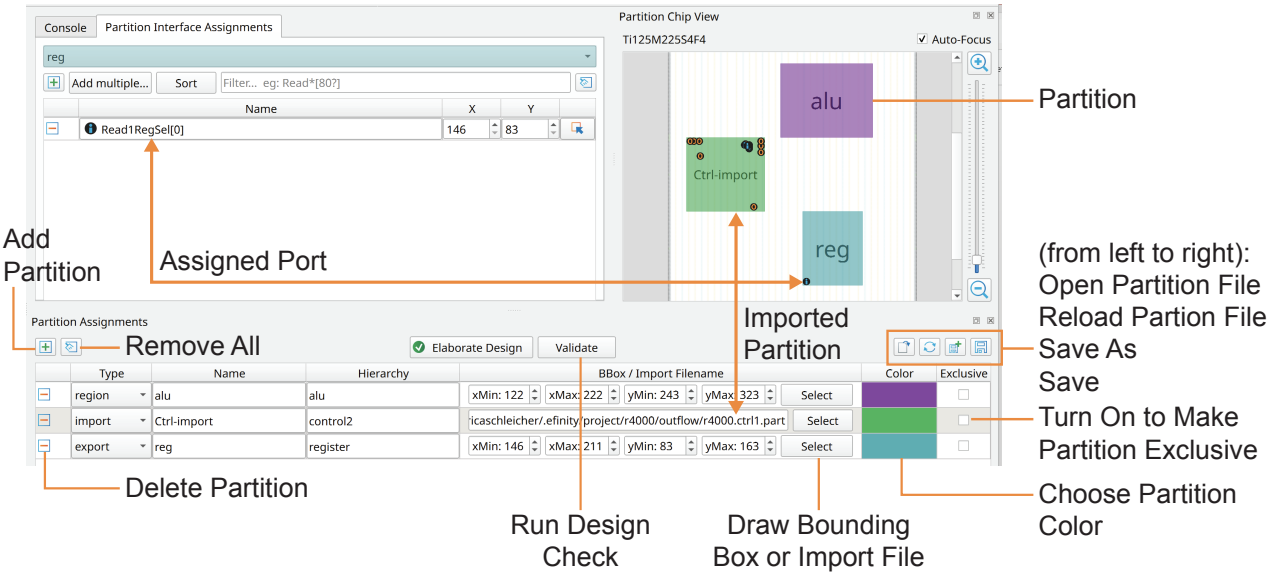
Given a sufficiently constrained partition, the placer correctly optimizes the placement of partition interface cells for quality of results for design re-use, placement regions, and incremental compilation flows. However, with a top-down methodology such as team-based design (where the partition design may be sparsely populated during early stages of co-development), the placer may not be able to produce a placement configuration that works

well during final integration compile. To minimize this effect, you can manually assign placement locations for partition interface cells to get better results. Typically, assigning a small subset of interfaces on the top-K most critical paths is usually sufficient to overcome most of the performance loss due to an under-constrained partition.

Using the Partition Planner

Open the Partition Planner using the **Tools > Open Partition Planner** menu item or by clicking the toolbar icon. The tool opens in the Efinity main window.

Figure 28: Partition Planner



The Partition Planner GUI has three parts:

- *Partition Assignments*—Shows the partitions in a table with one partition per row.
- *Partition Chip View*—Shows a floorplan view of the XLR cells and any assigned partitions.
- *Partition Interface Assignments*—Allows you to place Interface cells in specific XLR locations manually within an export partition.

Note: Before using the Partition Planner. specify the partition definition file (.pdefn) and partition interface file (.pifce) in the **Project Editor** > **User Partitions** tab. The software prompts you to enter these filenames if you have not done so already.

Create a Partition

The **Partition Assignments** tool is where you add (or remove) partitions and specify their settings. To add a partition:

1. Click **Elaborate Design**. The software elaborates the design and the resulting hierarchy is available for you to choose in the **Hierarchy** column.
2. Click the Add Partition button. The software adds a new row to the table.
3. Choose the partiton type: **region**, **export**, or **import** (see What Is a Partition? on page 67).
4. Enter a descriptive name in the **Name** box.
5. If you elaborated the design, choose the hierarchy name in the **Hierarchy** column.
6. Click **Select**.
   - For region or export partition types, draw the partition's bounding box on the **Partition Chip View** or specify the *x,y* coordinates (see Size and Placement on page

69). Tip: You can update the height or width of a region by grabbing the side of the bounding box and dragging it.

- For import partition types, the **Select Partition Archive** dialog box opens and you can choose a **.part** file to import.

7. (Optional) Click the cell in the **Color** column and use the slider to choose a new color.
8. (Optional) Turn on **Exclusive** (see Exclusive Partitions on page 69).
9. Click **Validate**. The software performs a design check and notifies you of any errors.
10. Click the Save button when you are finished.

Assign Interface Ports

Before you export a partition you may want to assign (or constrain) the interface ports to specific XLR cells. Later, when you import the partition into another design, the software constrains the ports to the assigned location(s). The software automatically places any unassigned ports.

Note: You must assign each bit individually.

To assign a port:

1. Select the export partition's name in the drop-down list box.
2. Click the Add interface button. A new row displays.
3. Start typing a port name in the **Name** cell and then choose the name from the drop-down list box. Alternatively:
   a. Click **Add multiple...** to open a dialog box with a list of port names.
   b. Shift-click to choose multiple ports.
   c. Click **OK**. The software adds them to the table.
4. Click the Pick and Drop button and then click an XLR cell box on the **Partition Chip View** floorplan. Alternatively, specify the *x,y* coordinates. The software places an icon with an i (for input) or o (for output) in the cell. Use the magnifying glass buttons or the slider to zoom the floorplan in or out.

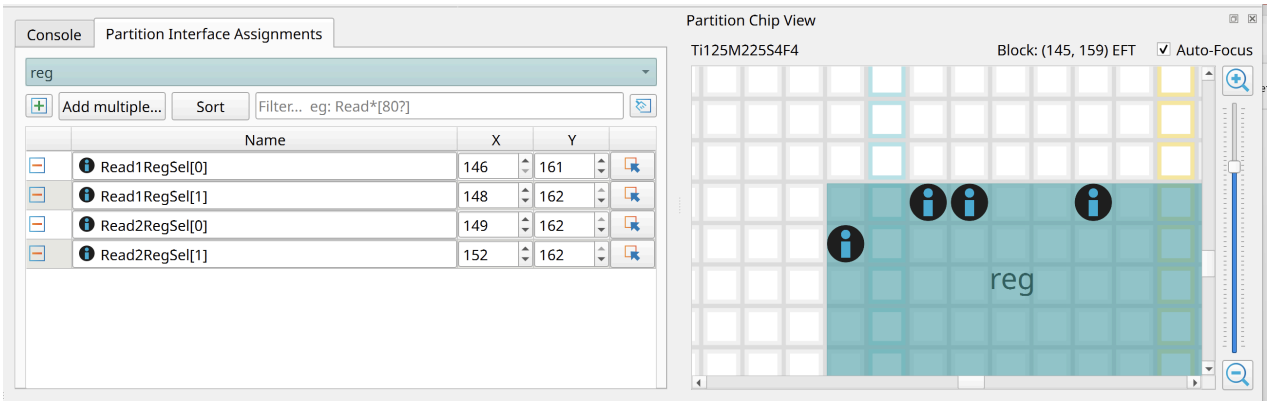
Tip: To move a port to a new spot, click it's icon, move the cursor to a new location, and then click again.

5. Click **Validate**. The software performs a design check and notifies you of any errors.
6. Click the Save button when you are finished.

You can sort the signals in the table or filter them using a wildcard expression.

The following figure shows assigned ports for the `reg` partition.

Figure 29: Assigned Ports



Best Practices

Follow the guidelines in this topic when working with partitions.

Size Requirements

When deciding how big to make a partition, Efinix recommends adding 15-25% additional logic resources to the number of expected XLR cells used. The equation to calculate the partition size is:

```
<partition area> >= (1.0 + <margin>) * (<expected XLR cell usage> + <# of
 partition interfaces>)
```

where $<margin> \in [0.1, 0.2] \rightarrow [0.15, 0.25]$

I/O and Interface Resources

Efinix FPGAs have a distinct boundary between the core logic and the I/O and interface resources. Partitions only include XLR cells (core logic) and cannot contain any I/O or interface resources. The Partition Planner's **Partition Chip View** window shows all of the core logic, and you can partition all of it.

When you import a partition, you need to assign the I/O and other interface resources. Use the same interface resources across compilations to ensure that the imported partition(s) behave as expected.

Clock Interfaces

To avoid unintended clock gating, the software removes partition interfaces for clock ports at the beginning of place and route. Therefore, you cannot assign placement locations for clock ports.

Routing Path Preservation

When you import a partition, the software preserves all routing paths between each pair of input and output ports in the partition interface. Because the software removes clock interfaces, the software re-routes clock pins (on the device boundary) to clock ports. If the new design has the same timing settings and constraints as the original partition, the resulting clock pin to clock port timing is identical; however, the routing paths between the pin and interface port may be different.

Timing Constraint Perservation

An exported partition does not preseve timing constraints; therefore, when you import it, the partition does not include timing constrains from the original project. You need to migrate and adapt the necessary timing constraints to new the project when you import the partition.

Cross-Partition Optimization

To support partition reuse (exporting and importing), the software treats each partition as a distinct block and does not perform synthesis optimizations across partition boundaries. As described in **Hierarchy Path** on page 68, the software inserts XLR cells for each export and import partition port. Because those interface XLR cells are mapped to `EFX_LUT4` primitives, you may notice additional delay overhead for signals crossing the partition boundary.

Partitions and Out-of-Context Synthesis

Partitions and out-of-context synthesis are separate but complimentary features.

Out-of-context synthesis applies at the RTL module level: you instantiate pre-compiled RTL modules in your Verilog HDL or VHDL design. These pre-compiled modules do not have placement or routing associated with them. The Efinity place-and-route tool handles out-of-context modules and your regular RTL modules the same way.

Although you define partitions before synthesis (where additional logic may be inserted), partitions are physical constraints that tell the place-and-route tool where to place the XLR cells. Because the place-and-route tool does not distinguish between pre-compiled modules and your regular RTL design, a design partition may contain a mix of XLR cells from both sources.

Table 14: Comparing Out-of-Context Module and Import Partitions

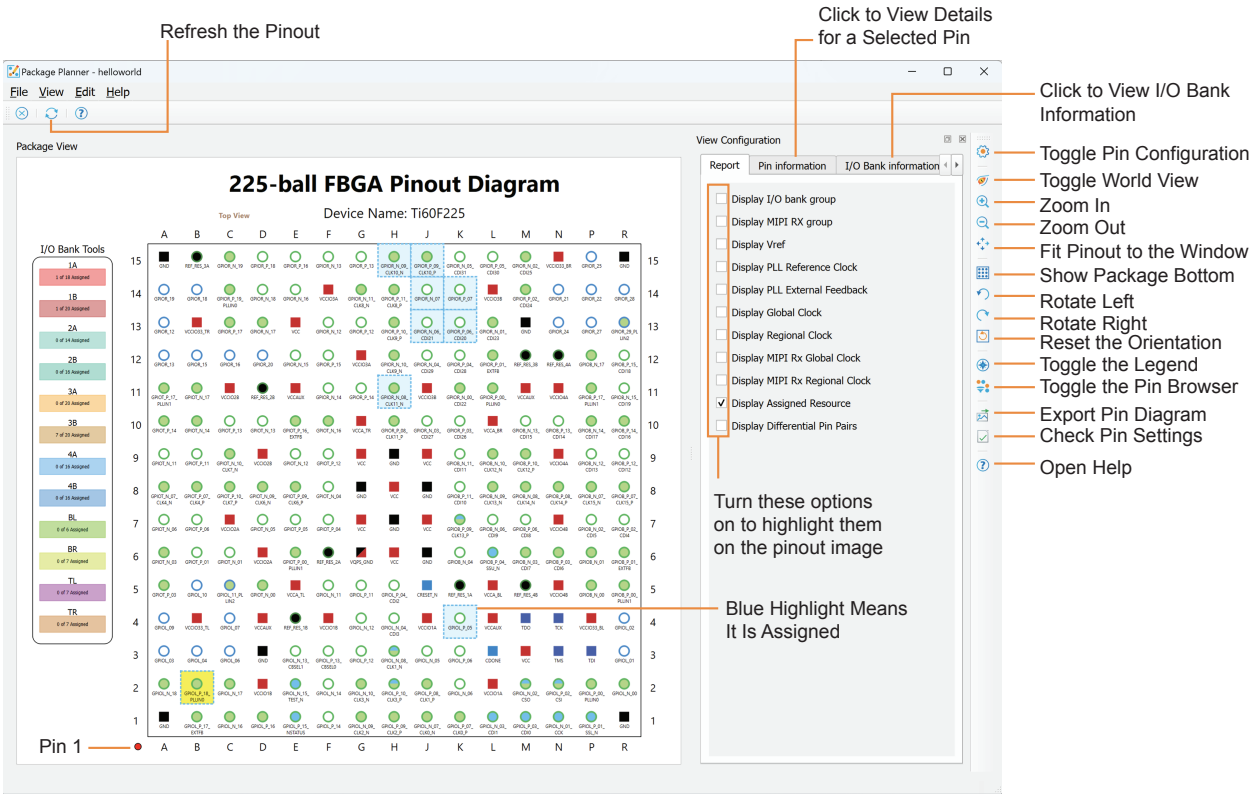
Workflow	Out-of-Context Module	Imported Partition
What is preserved	Synthesized and optimized netlist.	Synthesized and optimized netlist as well as placement and routing.
Limits on re-use	You can instantite is one or more times in the same RTL design.	You can only import the partition once because the placement and routing are fixed.

Editing and 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.

The Package Planner has a variety of options to display various types of pins. For example, you can highlight VREF pins, MIPI RX groups, and clocks. In v2025.2 and higher, you can also view the assigned pins and differential pin pairs.

Figure 30: Package Planner



Selecting a Pin

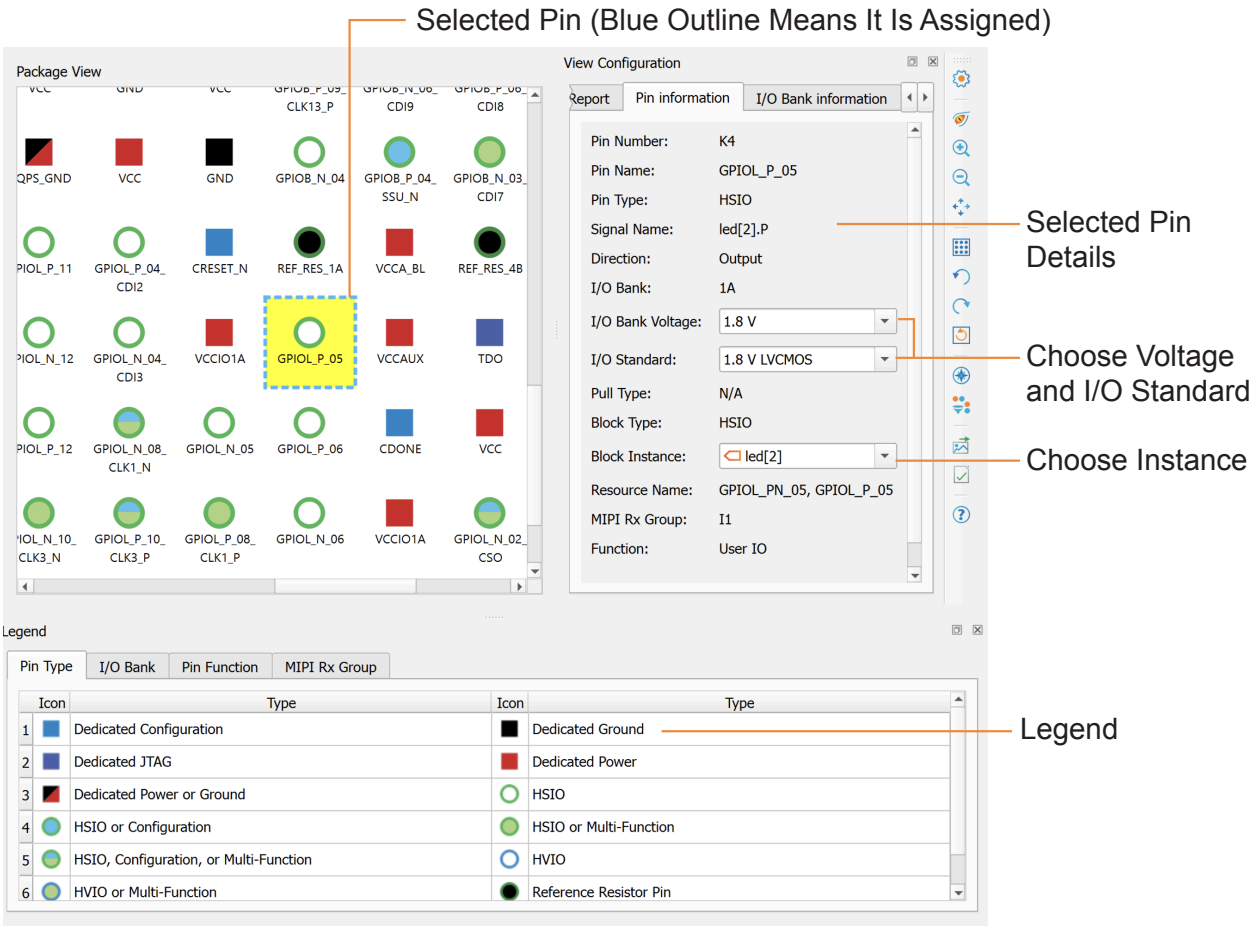
Click a pin in the pinout to highlight it. The **Pin Information** tab opens to show the details about the selected pin. Open the Legend to view the meaning of the pins' color coding. Select multiple pins by pressing the Control key and clicking (however, the **Pin Information** tab only shows data for the first pin you click).

In v2025.2 and higher, you can use the Package Planner for pin assignment:

1. Click the pin.
2. In the **Pin Information** tab, choose the instance in the **Block Instance** drop-down list box.
3. Choose the I/O bank voltage and I/O standard.

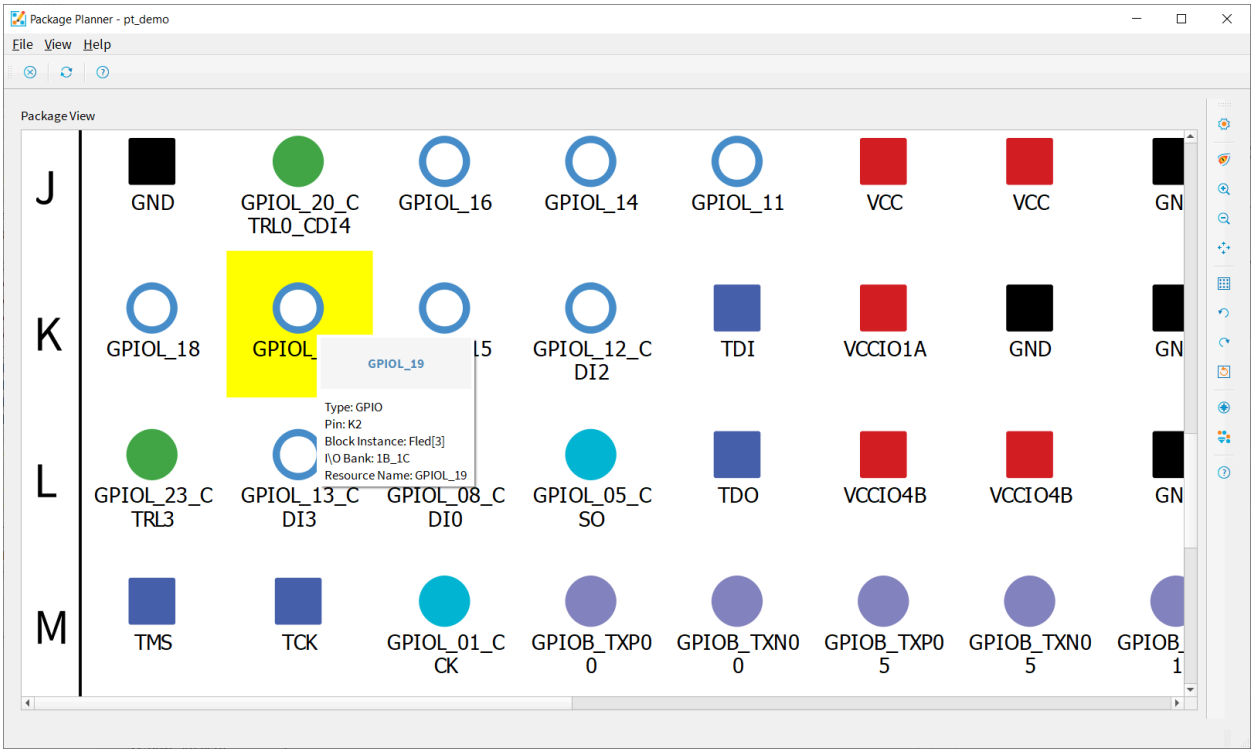
Note: You can also set the voltage for the entire I/O bank in the **I/O Bank Information** tab. If the voltage etting for the entire I/O bank does not match what you choose for the pin, the software highlights the error and displays an error message.

Figure 31: Selected Pin



You can also hover over a pin for a quick view of the pin details.

Figure 32: Pin Quick View

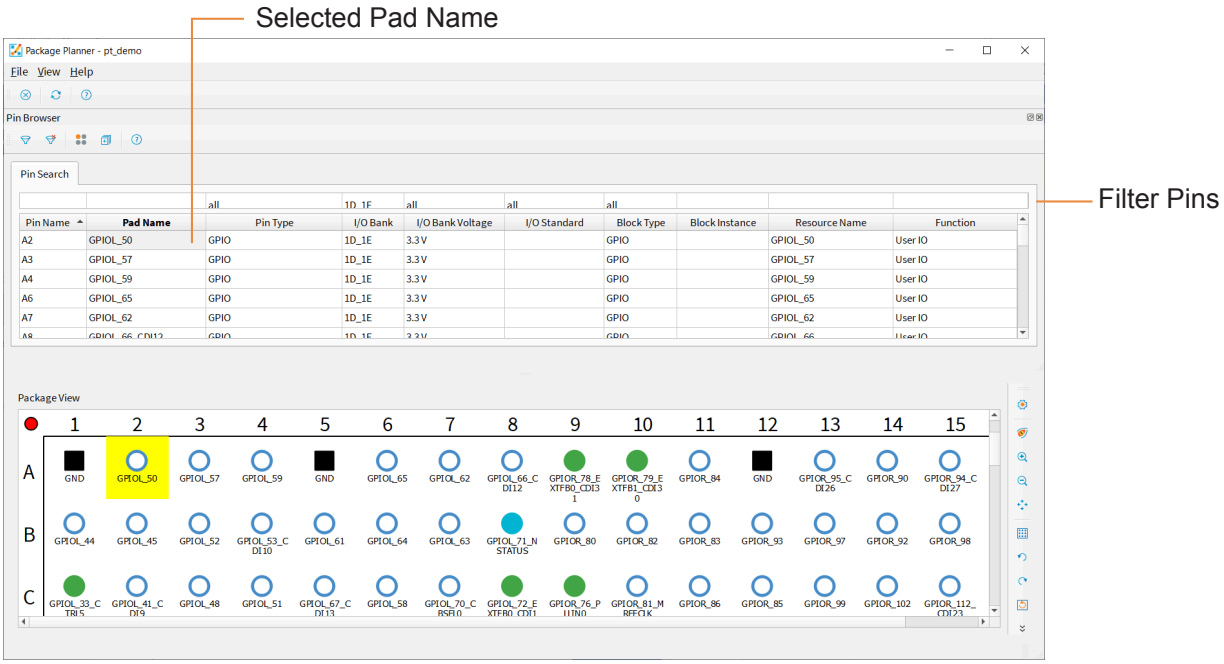


Browsing for Pins

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.

In v2025.2 and higher, you can view multiple selected pins in the **Pin Browser**. Press the Control key and click the pins, then open the **Pin Browser**. Similarly, you can select multiple pins in the **Pin Browser** to highlight them in the package view. Shift-click to select pins in consecutive cells, Control-click to select ones that are not.

Figure 33: Browsing for Pins



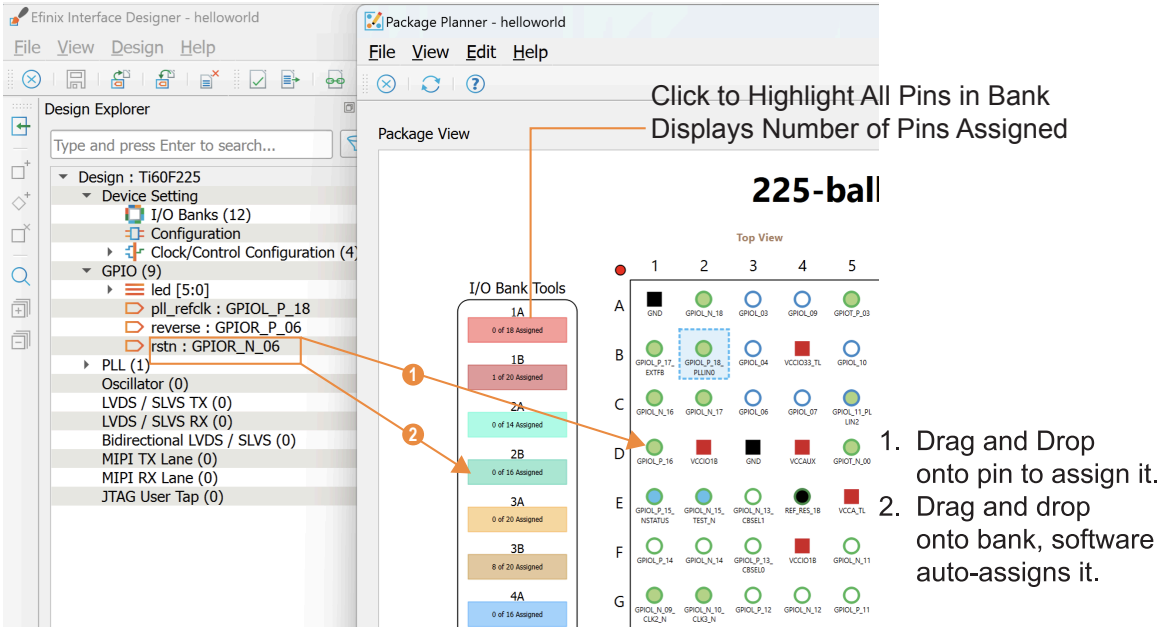
Drag-and-Drop Assignments

In v2025.2 you can use drag-and-drop to assign instances to a pin or bank. Position the Interface Design and Package Planner windows so that they are side by side or overlapping. Click the instance name and drag it onto the Package Planner. Drag to a pin to assign the instance to that pin. Drag to a block in the I/O Bank Tools and the software auto-assigns it to a pin in the chosen bank.

In v2025.2 you can drag and drop GPIO, LVDS, MIPI lanes, MIPI D-PHY, DDR, SPI flash, and PLL SSC blocks. For the multi-pin instances, drag the instance onto one of the pins. For example for MIPI D-PHY, drag the instance onto any of the MIPI pins. Additional instance support (such as buses) is planned for a future software release.

Note: When you drag an instance onto the I/O bank to auto-assign it, the software does a design check to confirm that there is a pin available. If the software cannot find a valid assignment, it displays a message prompting you to choose a different bank.

Figure 34: Drag-and-Drop Instances



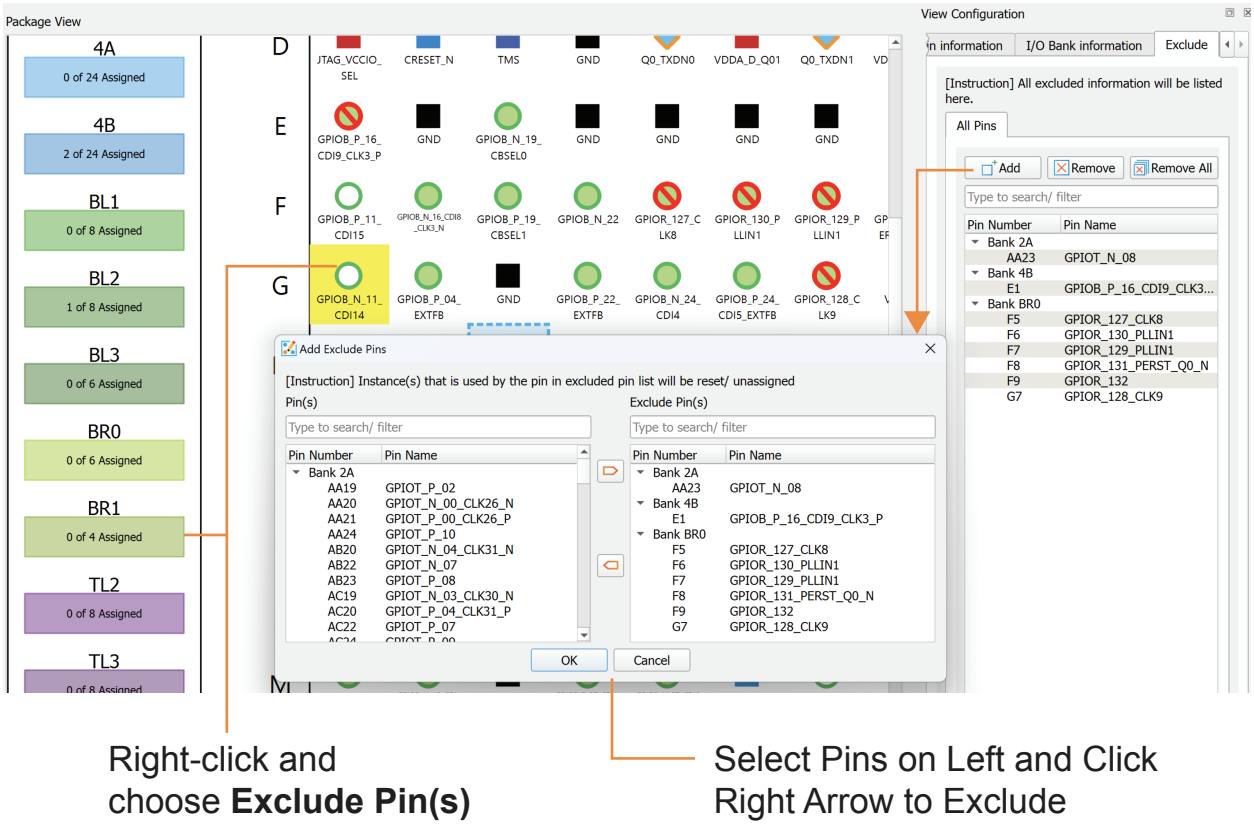
Excluding Pins and Banks

In v2025.2 and higher you can exclude pins or I/O banks from auto-assignment. The **Exclude** tab shows the list of excluded pins. It also has buttons to clear the setting or add more pins to the exclusion list.

- Right-click on a pin or back and choose **Exclude Pin(s)** from the pop-up menu. If you have excluded a pin, you can right click and choose **Clear Excluded Pin(s)** to remove the setting.
- Alternatively, choose the **Exclude** tab and click **Add**. In the Add Exclude Pins dialog box, select the pins in the **Pin(s)** box on the left and click the arrow to move them to the **Exclude Pin(s)** box on the right. You can include pins by moving them back to the **Pin(s)** box.
- If a pin has an assignment and you exclude it, the software issues a warning that it will remove the assignment.
- If a pin is excluded, then you cannot change any settings for it, e.g., I/O voltage and standard, and instance assignments.

Note: Be careful when excluding pins that must be used together, such as differential pairs. If you exclude one of the pins in a pair, the other can still be used as a GPIO; however, it cannot be used for LVDS because LVDS requires both pins. Turn on **Report** > **Turn on Differential Pin Pairs** to see which pins are used together for LVDS.

Figure 35: Drag-and-Drop Instances



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 in 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.

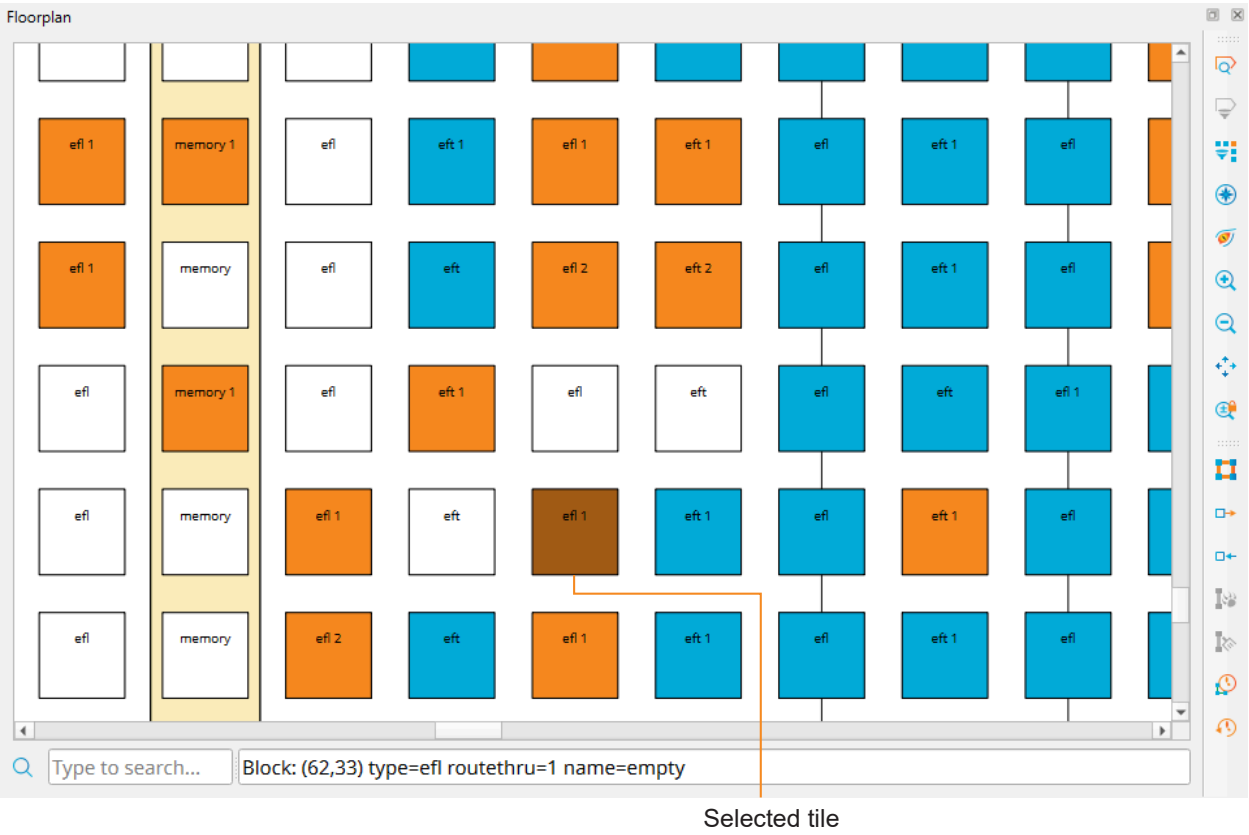
Table 15: FPGA Tile Types

Tile	Trion	Titanium	Topaz	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 36: 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 16: 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 17: Mapping Titanium and Topaz 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 18: Mapping Primitives to Tiles

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

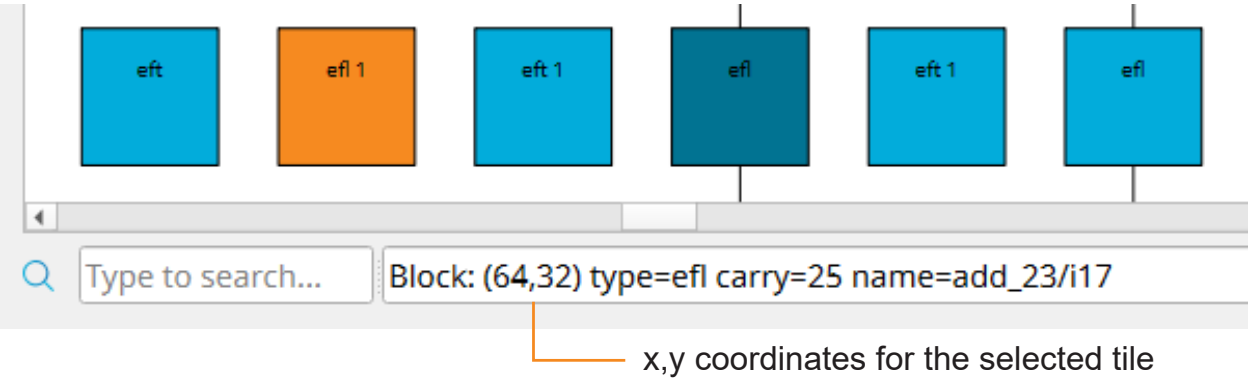
```
<block name>    <x>    <y>    <subblk>
```

- <*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 ().



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 (Beta)

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

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 80 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 87)
   a) For Trion devices, set `--route_dump_constraint_file="on"` to generate **<*project*>.route2**, which is later used in the constraint flow.
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 routing traces file is **<*project*>.troutingtraces** for Titanium and Topaz, and **<*project*>.route2** for Trion.

Additionally, the process generates <project>.placer_keepout, which is later used by the placer in the constraint flow.

5. Move these 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 87 and Creating a Location Assignment File on page 84)
7. Add your new constraint files to an **.ini** file. (See Enabling Routing Constraints on page 88)
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.

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 87) 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.

  Note: Can only be used by Titanium and Topaz.

Remember: For Trion, the routing trace file is passed using a command-line option: **routing_constraint_file="{***project***}.route2"**

- `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
```

Enabling Routing Constraints

You must enable routing constraints with an **.ini** file in the Efinity software v2022.2 and above. 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
placer_keepout_file = <path>/<filename>.placer_keepout
```

Note: Only for Trion devices, **routing_constraint_file=<*path*>/<*filename*>.route2**

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

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.

  Note: Titanium and Topaz only.

- 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.

Note: For Trion, add `route_dump_constraint_file` to the Trion flow and set `route_dump_constraint_file = on`.

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
placer_keepout_file=my.placer_keepout
(Trion) routing_constraint_file= my.route2
```

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.

Chapter 7

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 8

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 interface blocks. 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 core and portion of the interface design defined in the source RTL using the `--flow ptsimrtl` flag.
   b. Simulate the core design and all of the interface logic defined in the source RTL or the Interface Designer using the `--flow ptsimfc` flag. This requires a new testbench targeted at the full chip module called <*project*>`~chip`.

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 interface
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 interface
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 19: 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_LUT4	Simple 4-Input LUT ROM	✓	✓	✓	efx_lut4.v
EFX_MULT	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 20: Interface Primitive Simulation Models

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

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 Single-Ended GPIO		✓	✓	EFX_GPIO_V3.v

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_GPIO_DIFF_V3	HVIO and HSIO Used as Differential GPIO		✓	✓	EFX_GPIO_DIFF_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
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
EFX_LVDS_RX_V1	LVDS Reciever	✓			EFX_LVDS_RX_V1.v
EFX_LVDS_RX_V2	LVDS Receiver		✓	✓	EFX_LVDS_RX_V2.v
EFX_LVDS_TX_V1	LVDS Transmitter	✓			EFX_LVDS_TX_V1.v
EFX_LVDS_TX_V2	LVDS Transmitter		✓	✓	EFX_LVDS_TX_V2.v
EFX_LVDS_BIDIR_V2	LVDS Transmitter/Receiever		✓	✓	EFX_LVDS_BIDIR_V2.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 <file>
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 <file>
efx_run.bat <project name>.xml --flow rtlsim|mapsim --tb_top <top-level module name>
```

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. This free simulator has limitiations: it cannot decrypt code (such as IP from the IP Manager), and it does not support SystemVerilog.

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

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 (<*flow*>**.rtl.simlog** and <*flow*>**.map.simlog**) and error messages (<*flow*>**.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 (<*flow*>**.rtl.simlog** and <*flow*>**.map.simlog**) and error messages (<*flow*>**.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 37: 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 38: 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 (<*flow*>**.rtl.simlog** and <*flow*>**.map.simlog**) and error messages (<*flow*>**.log**) in your project's **outflow** directory.

Simulate with the Aldec Active HDL or Riviera-PRO Simulator

By default, the Efinity® software calls the iVerilog simulator. Use the `--aldec` option to target the Active HDL or Riviera-PRO simulators 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 Aldec Active HDL or Riviera-PRO

Linux:

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

Windows:

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

Note: The Aldec Active HDL simulator can be used in GUI mode with the `--sim_opts gui` option.

Linux:

```
efx_run.py <project name>.xml --flow rtlsim --aldec --sim_opts gui
```

Windows:

```
efx_run.bat <project name>.xml --flow rtlsim --aldec --sim_opts gui
```

The simulator responds with

- PASS if the simulation is successful.
- FAIL if the simulation is unsuccessful.

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

Chapter 9

Debugging

Contents:

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:

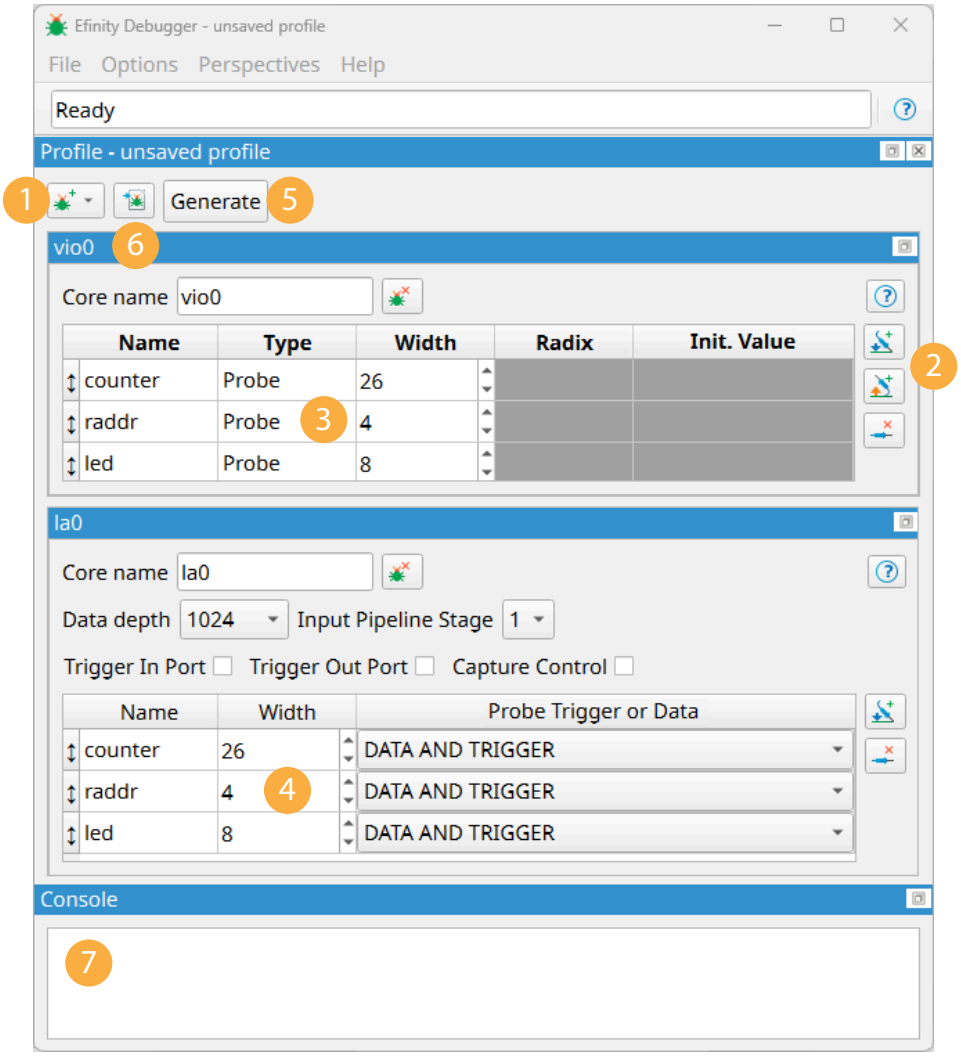
1. Create a debug profile with the Virtual I/O and/or Logic Analyzer debugger core(s).
2. Generate the debug design file and add it to your project.
3. Compile.
4. Program the FPGA.
5. Run the Debugger GUI and observe the values on the probes.

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 39: Debugger Profile Editor Perspective



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 40: Virtual I/O Core Block Diagram

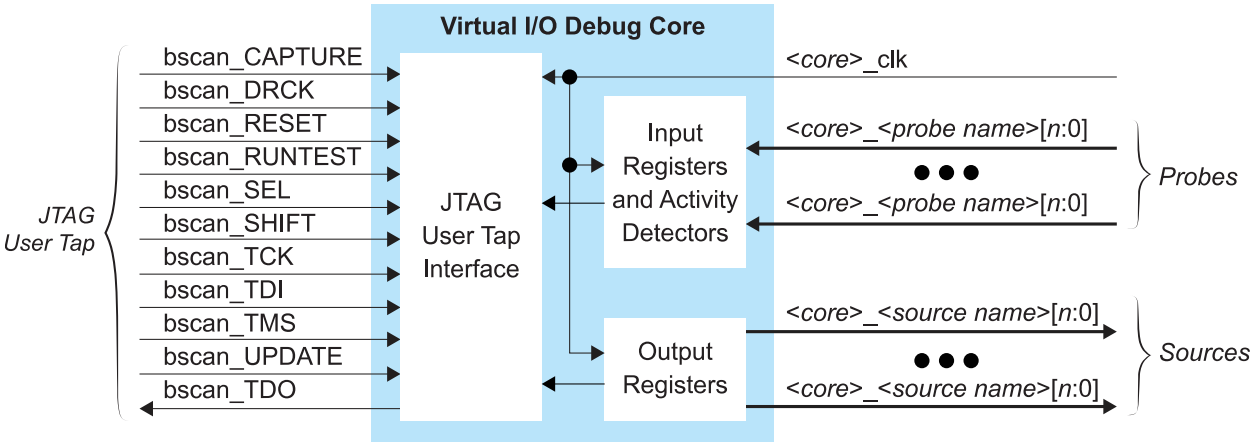


Table 21: Virtual I/O Core Ports

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. Input a debug profile name into the **Profile name** field.
4. Choose **Add Debug Core > VIO**.
5. Specify the core name.
6. 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.
7. Click **Generate Core RTL**. The Efinity® software saves the debug profile in your project directory as *<profile name>***.json**. The software also creates a debug template (*<profile name>***_TEMPLATE.v**), which includes the module for the debug profile you created and *<profile name>***_top.v**, which is the RTL logic for the debug core.
8. Add the *<profile name>***_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.

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

   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.

10. 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 *<profile name>***_TEMPLATE.v** or *<profile name>***_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`).
11. 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.

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 41: Logic Analyzer Core Block Diagram

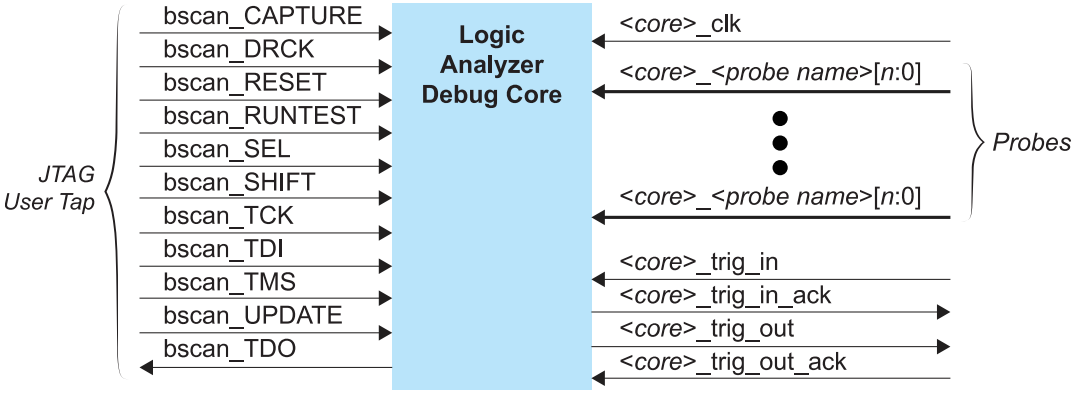


Table 22: Logic Analyzer Core Ports

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. Input a debug profile name into the **Profile name** field.
4. Choose **Add Debug Core > Logic Analyzer**.
5. Specify the core name.
6. 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.
7. Turn on Trigger In Port and/or Trigger Out Port to enable those signals.
8. Turn on Capture Control if you want to change the capture mode in the **Capture Setup** tab during debugging (see Debug Perspective on page 107 for details). If you turn this option on, the Logic Analyzer uses more FPGA resources.
9. 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.
10. Click **Generate Core RTL**. The Efinity® software saves the debug profile in your project directory as <*profile name*>**.json**. The software also creates a debug template (<*profile name*>**_TEMPLATE.v**), which includes the module for the debug profile you created and <*profile name*>**_top.v**, which is the RTL logic for the debug core.
11. Add the <*profile name*>**_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.

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

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.

13. 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 <*profile name*>**_TEMPLATE.v** or <*profile name*>**_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`).

14. 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 specifiy 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 107 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.

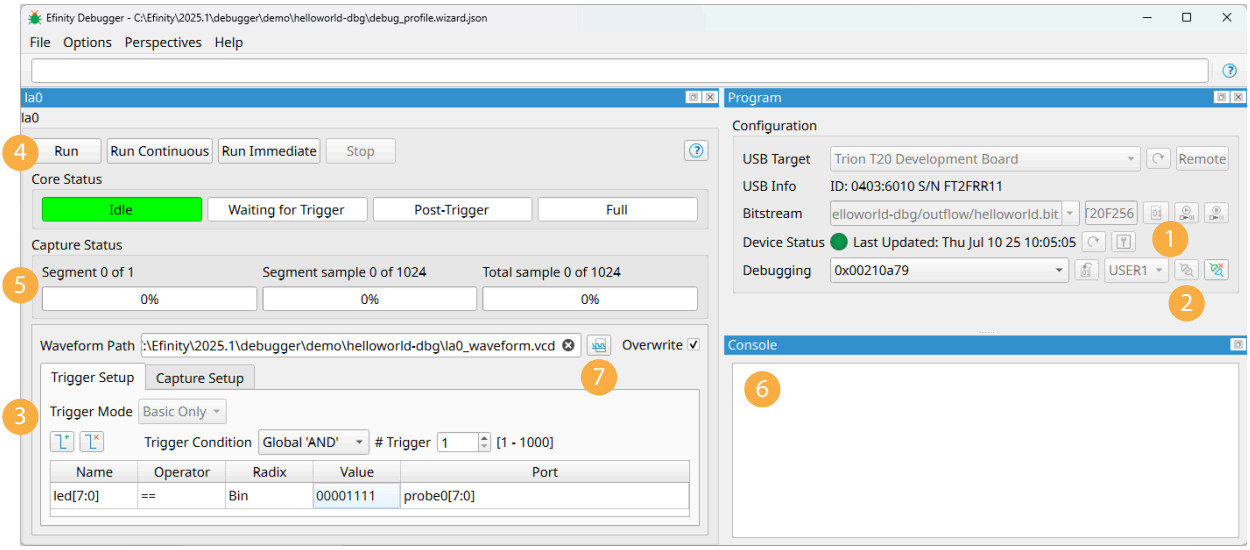
You can open multiple Debugger windows. Choose **Tools > Open Debugger** multiple times or click the Debugger icon multiple times to open additional windows. When you close the Efinity software, all Debugger window close as well.

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 42: Debug Perspective GUI - Logic Analyzer



Select Bitsteam | Connect Debugger | Disconnect Debugger

Start Programming | Add Net | Remove Net

Stop Programming | Select Waveform File

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:

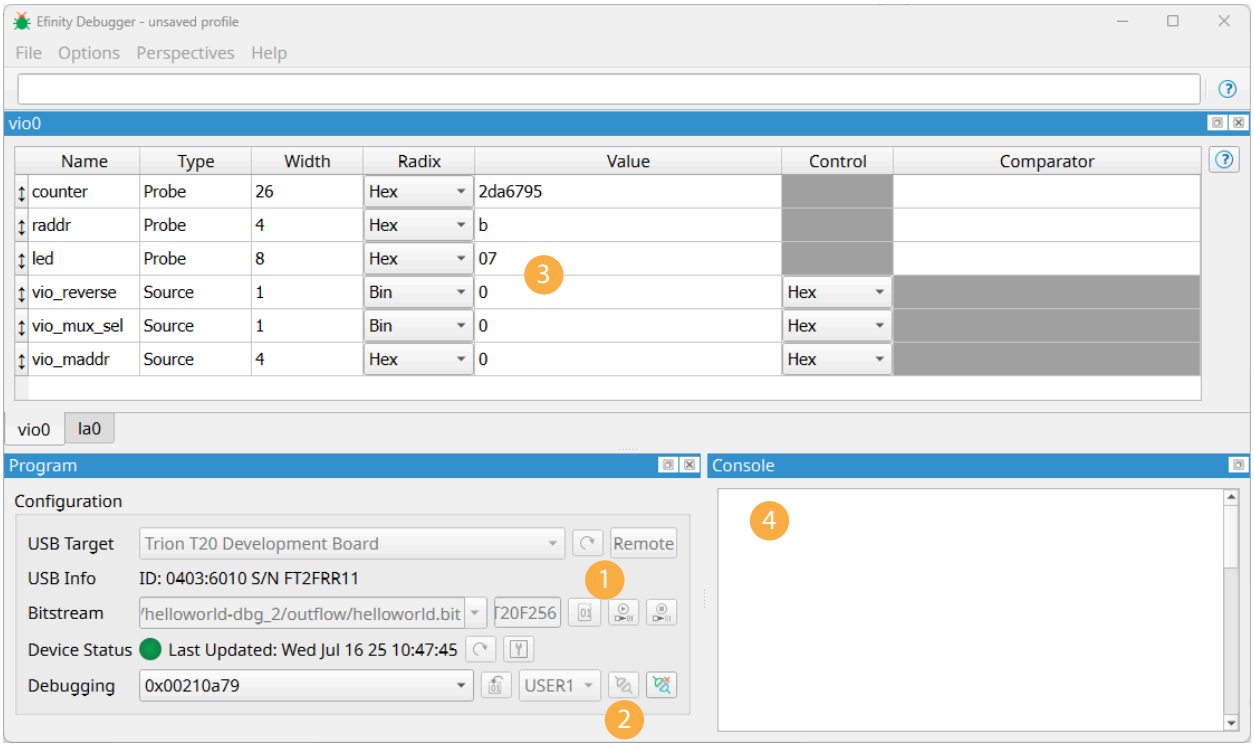
- **Always**—Stores a data sample during a given clock cycle regardless of any capture conditions you set.
- **Basic**—Only stores a data sample during a given clock cycle if the capture condition evaluates as true. Select this option to add nets and set capture conditions.

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 43: 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 23: 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.

Using the mark_debug Synthesis Attribute

You can use the attribute, `mark_debug`, to mark the debug nets for auto debug probe insertion. You include the `mark_debug` synthesis attribute in your RTL and set it to `true` or `1`. If the attribute is set to `true` or `1`, synthesis tool writes out the selected signal to a default file, *<project dir>*/**outflow/debug_profile.mark_debug.json**. In the **Project Editor**, you can enable or disable the `mark_debug` output by setting the synthesis option **enable-mark-debug** to `1` (default, enabled) or `0` (disabled).

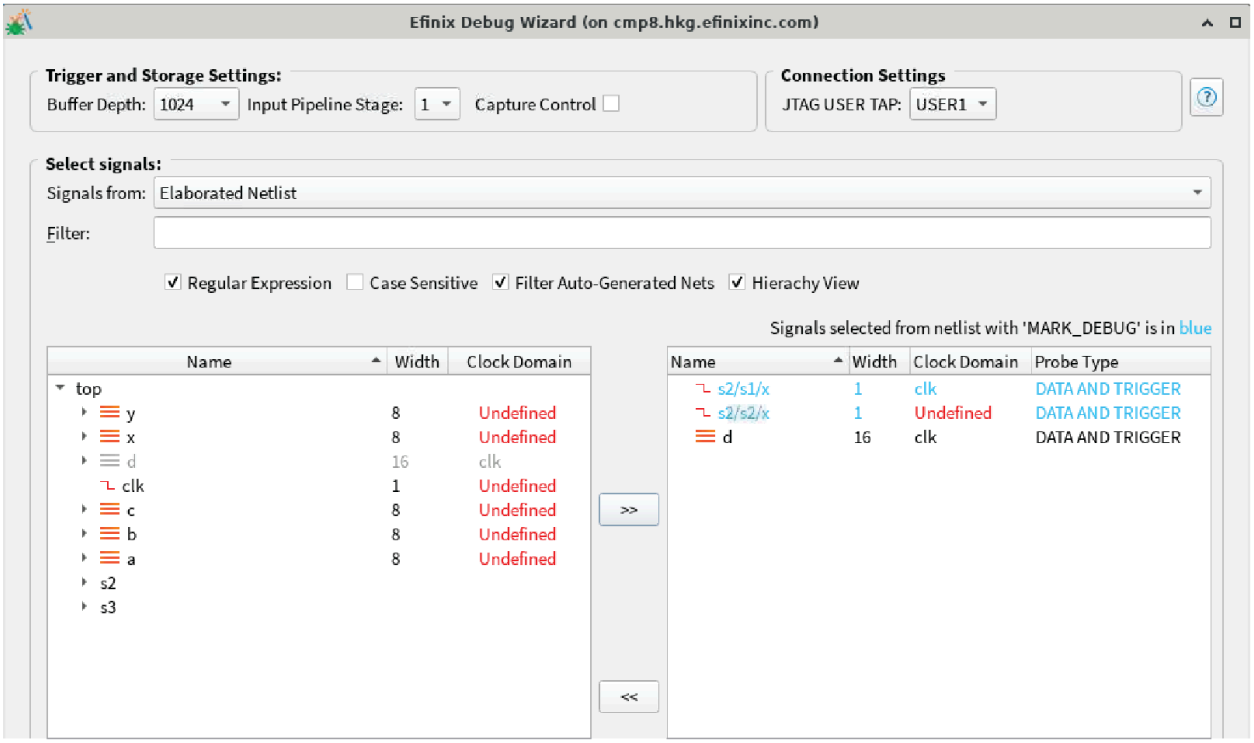
Note: See "mark_debug" in the .

After synthesis, you can open the Debugger Wizard to monitor the selected `mark_debug` signals (highlighted in blue in the following figure). In the Debug Wizard, you can:

- Edit the attributes related to the selected signal, e.g., clock domain and trigger type.
- Add additional signals from the design to be probed or remove the selected `mark_debug` signals.

When you finish using the Debug Wizard, the tool writes the **debug_profile.wizard.json** as usual. You can continue to debug as you normally would.

Figure 44: mark_debug Signals in the Debug Wizard



Note: If you have changed which signals have the `mark_debug` attribute in your RTL, you must re-run the Debug Wizard to reimport the selected or changed signals. The changes are retained if the same signal is selected and if the debug attribute has changed before.

In batch (command-line) workflows, the software generates the same *<project dir>*/**outflow/debug_profile.mark_debug.json** file. You need to edit the **project.xml** to add the `efx:debugger` section manually:

1. Turn on **auto instantiation**.
2. Specify the debug profile value as **outflow/debug_profile.mark_debug.json**.

```
<efx:debugger>
   <efx:param name="work_dir" value="work_dbg" value_type="e_string"/>
   <efx:param name="auto_instantiation" value="on" value_type="e_bool"/>
   <efx:param name="profile" value="outflow/debug_profile.mark_debug.json"
 value_type="e_string"/>
   </efx:debugger>
```

The **efx_run.py** script picks up the specially named file, **debug_profile.mark_debug.json** and calls **efx_run_dbg.py** with a new **mark_debug** option. This option triggers a call to a **efx_dbg/DbgWizard.py** function to process and generate the connection profile. The **debug_top.v** is required for the debugger auto insertion flow.

Note: In this command-line flow, all debugger attributes are set to default when the **debug_profile.mark_debug.json** file is first generated. Then, you can edit the JSON file to alter the following attributes:

- `data_depth`, default = `1024`
- `capture_control`, default = `false`
- `input_pipeline`, default = `1`
- `jtag_user`, default = `USER1`

For each of the probe signals, the clock domain is deduced automatically. However, the clock domain is left as **Undefined** for a pure combinational signal. In this case, you can edit the JSON file to specify the desired clock domain. This information is retained when the JSON file is regenerated in future synthesis runs.

Concurrent Debugging

In the Efinity software v2025.2 or higher, has an improved concurrent debugging mode (this improved mode does not require you to set up a hardware server). Concurrent debugging let you use the RISC-V IDE and multiple Efinity debuggers concurrently with a single FTDI interface.

When you enable concurrent debugging mode, OpenOCD is used as the JTAG back end. The Efinity software launches a daemon to manage the life cycle of OpenOCD processes automatically.

Note: The legacy concurrent debugger using the Efinity Hardware Server is no longer maintained as of the Efinity software v2025.2.

Concurrent debugging supports the following tools:

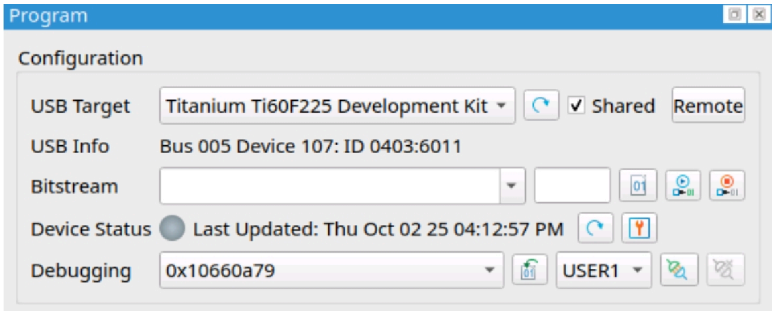
- Logic Analyzer
- Virtual I/O
- Transceiver Debugger
- RISC-V IDE
- Standalone Efinity Programmer

Concurrent debugging does not support the following tools and FPGAs:

- All Trion T4, T8, and T13 FPGAs
- Trion T20W80, T20Q100, T2Q144, T20F169, and T20F256 FPGAs
- Efinity SVF Player
- SPI Active programming mode

To use concurrent debugging:

1. Use the Debugger to create debug cores. Connect the debug cores and RISC-V SoCs to different JTAG User TAP interfaces (e.g., USER1 and USER2).
2. Compile the project.
3. Open Debugger windows for each JTAG User TAP interface. To open more than one Debugger window, choose **Tools > Open Debugger** or click the Debugger icon multiple times.
4. Turn on the **Shared** option in the Debugger or Transceiver Debugger window to enable concurrent debugging mode.



5. Start debugging.

Learn more: For more information on using RISC-V SoCs, see the Sapphire RV32 SoC User Guide.

Resource Usage

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 10

Debugging Transceivers

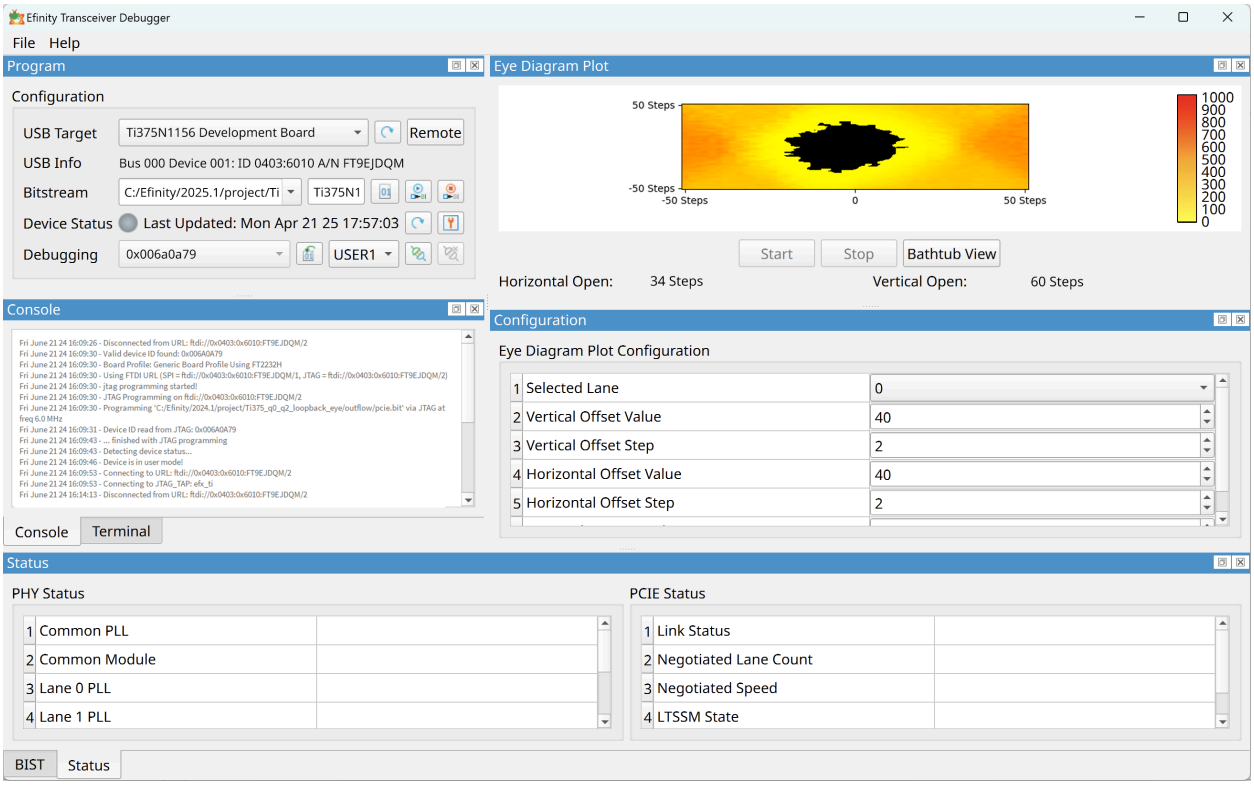
Contents:

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.

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. In v2025.1 and higher, the Transceiver Debugger supports BIST.

Learn more: Refer to the Efinity Transceiver Debugger Tutorial for information on how to use example designs provided with the Efinity software.

Figure 45: Efinity Transceiver Debugger



Launching the Transceiver Debugger

To open the Transceiver Debugger, choose **Tools > Open Transceiver Debugger**.

You can also 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.
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 > USERn** and connect the Transceiver Debugger. Choose the JTAG user TAP that connects to the the APB bridge. The loopback design provided with the Efinity software uses **USER2**.

   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.

   Important: When the PCIe link is in the polling compliance state, the Transceiver Debugger is not able to read the link width and speed. This behavior is expected.

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
Selected Lane	Choose the lane to use in the sys diagram plot, lane 0, 1, 2, or 3. Default: 0
Vertical Offset Value	Indicates how many vertical positions there are to the top and bottom 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.

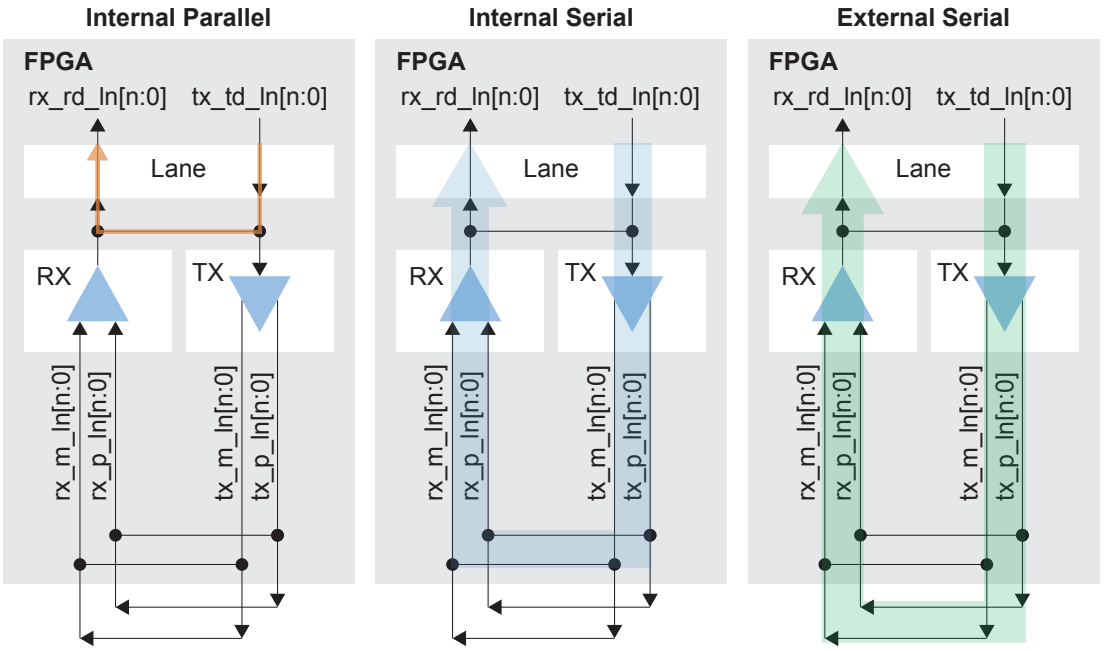
Setting	Description
	Default: 40
Horizontal Offset Step	Indicates how many horizontal positions to include in the same color pixel. For example, for an offset value of 40 and a step of 2, the software plots a total of 20 blocks on each side of the center line. A higher number results in less drawing time. Default: 2
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. Default: 7

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

Debugging with BIST

To use the BIST function, you need a loopback design, in which the transceiver transmits data and then receives it back. You can use internal parallel or serial loopbacks, or an external serial loopback. You access the BIST function through the transceiver's APB interface.

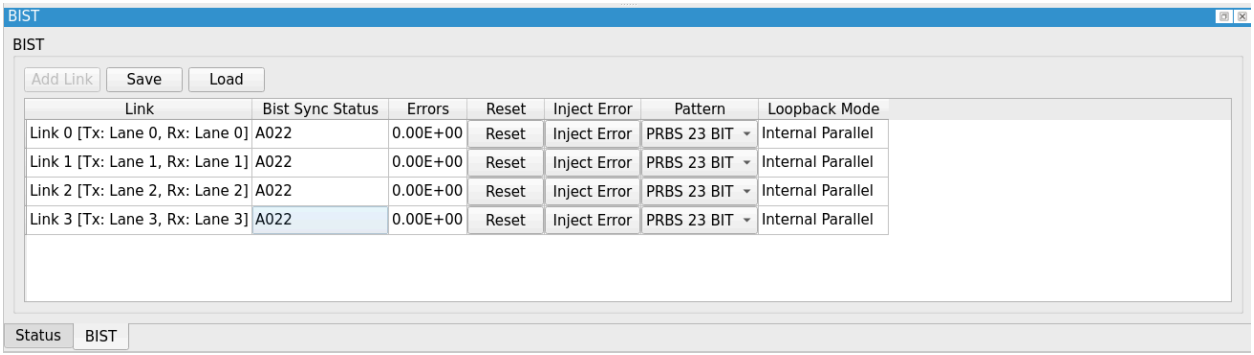
Figure 46: Transceiver BIST Loopback Types



Note: The Efinity includes a loopback project that you can use for testing with the transceivers BIST function.

In the Transceiver Debugger, click the **BIST** tab to set up the test.

Figure 47: Transceiver Debugger BIST Tab



- Click **Add Link** to add lanes. The **Link** column shows the list of links.
- **Bist Sync Status** shows a hex value indicating whether the link is running or has an error:
  - `A022`—BIST is running.
  - `A02A`, `A02E`—An error occurred. The value toggles between `A02A` and `A02E`.
  - `A020`—BIST is not running.
- **Errors** shows the current error count as an exponential sum.
- Click **Reset** to reset the link's error count to 0.
- Click **Inject Error** to force a 2-bit transmit error on the link.
- In the **Pattern** menu, choose the sequence type for testing. Currently the debugger only supports a pseudo-random binary sequence (PRBS).

Important: If the transceiver is in BIST mode, it can only be used for testing. Re-program the FPGA to use the transceiver for regular data transmission.

Sending Commands

The Transceiver Debugger's **Terminal** area lets you interact with the PCIe Controller's registers by sending commands through the APB interface.

- *Single Command*—Choose this option to send one command. Enter the command in the **Command** box and click **Send**.
- *Mass Read/Write*—To send multiple commands choose this option.
  - Click the input file button next to the **Input File** box and browse to select a text file of commands. The file should be a text file (**.txt**) with one command per line. for example:

```
write:0x204048,0x00000000
write:0x2041A8,0x00000A11
write:0x2041A8,0x00000A01
read:0x20F000,0x0000A022
read:0x2041AB,0x00000000
```

  - Optionally, you can write the output to a file as well as displaying it in the **Console**.
    1. Turn on **Output File**.
    2. Click the output file button and browse to a directory in which to save the output.
  - Click the play button to start running the commands.
  - Click the stop button to stop running commands.

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

Table 24: 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 SGMII 1G and 2.5G User Guide** ("Register Map")
- **Titanium PMA Direct User Guide** ("Register Map")

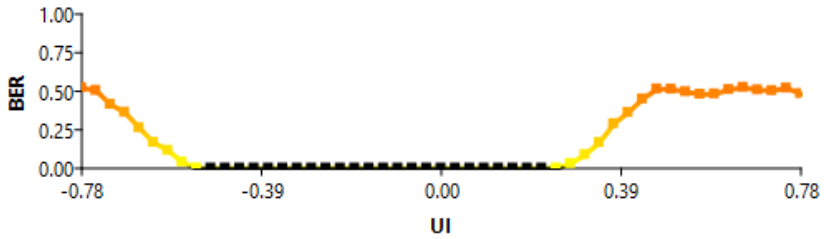
Interpreting the Results

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).

The bathtub view shows another representation of the link quality. The colors represent the same link quality levels as the eye view. Use the **Bathtub View** and **Eye View** buttons to switch between the views.

Figure 48: Bathtub View



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 11

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).

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 25: FPGA Configuration Modes

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

Flash Programming Modes

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

Important: If you are using the secutiry feature (Titanium or Topaz only), you can no longer use the JTAG bridge flash programming modes after you disable JTAG access. Refer to **Securing Titanium Bitstreams** on page 169 for details.

Table 26: Flash Programming Modes

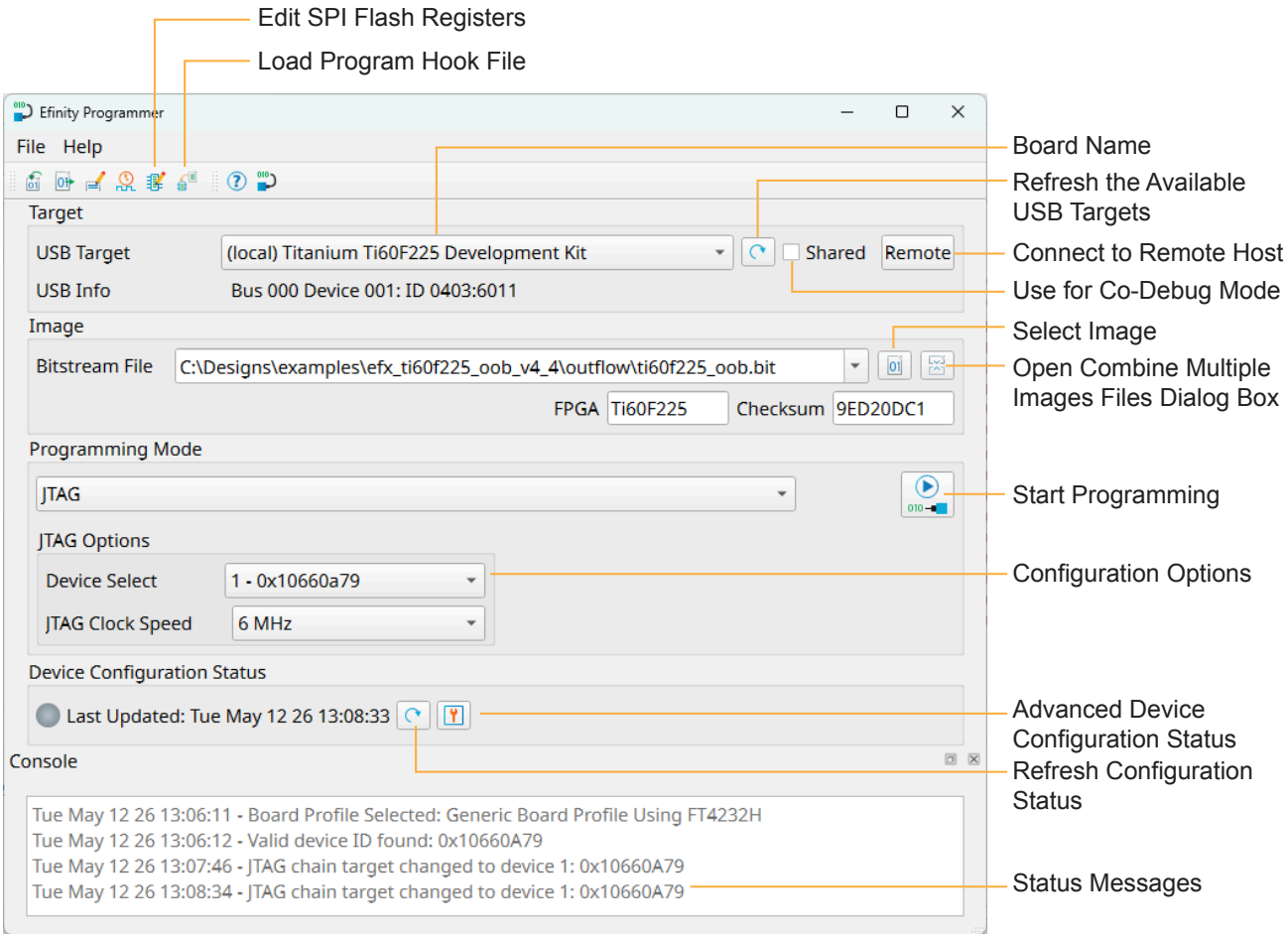
Mode	Description
SPI Active (serial/parallel)	Use the Efinity Programmer and a cable connected to a SPI header on the board.
SPI Active using JTAG Bridge	A improved version of the SPI Active using JTAG Bridge (Legacy) mode with a faster flash programming time.
SPI Active x8 using JTAG Bridge	A improved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

Learn more: Refer to **Program using a JTAG Bridge** on page 139 for more information.

About the Programmer GUI

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

Figure 49: Programmer



To use the Programmer:

1. Choose a local or remote target (the software indicates whether the connected hardware is local or remote).
2. (Optional) To use the Programmer at the same time as the Debugger, turn on the **Shared** option.
3. (Optional) Click the **Remote** button to connect to a board that is attached to a remote host. See Working with Remote Hardware on page 185.
4. Choose a bitstream file or use the drop-down arrow to select from a list of recently used bitstream files. 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>
```

5. 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	Starting Flash Address Flash Length Erase Before Programming Select Verify Method Device Select (the drop-down menu list box shows the order of the device in the JTAG chain and the JTAG ID) JTAG Clock Speed
SPI Active x8 using JTAG Bridge	All options from SPI Active using JTAG Bridge plus: Select Flash

6. 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 166 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.

Tip: In the Efinity software v2026.1 the Programmer has a new menu option **Use new SPI flash backend**, which is enabled by default. In v2026.1, the Programmer has new handling for flash profiles. In the rare case that your configuration setup does not work with the new handling, you can try turning off this option to use the legacy handling. This option will be removed in a future software version.

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 27: 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 163.

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

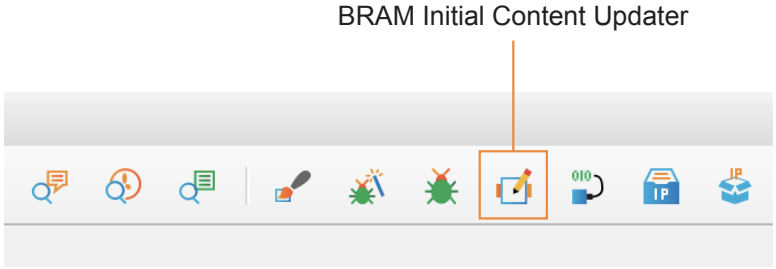
About the BRAM Initial Content Updater

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 50: Using the Netlist Pane



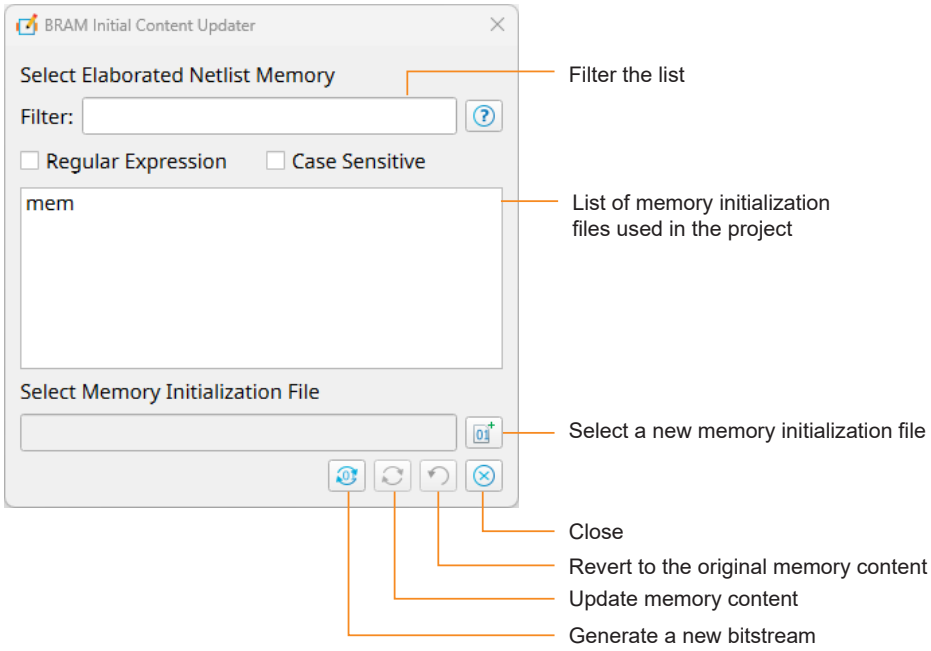
In the tool, you select the elaborated netlist memory that you want to update, not the post-map memory. Therefore, you do not need to know how synthesis decomposes and maps the memories to use this tool. Because this tool bypasses the full compilation flow, it does

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 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 51: 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 an Efinix development board you can use the Debugger example files provided with the Efinity software to experiment with the BRAM Initial Content Updater. By default, the example targets the Ti60 F225 Development Board. Refer to **Use Another Board** on page 127 for instructions on how to use a different board.

1. Connect the board to your computer.
2. Copy the `helloworld` project in the *<Efinity install path>*/**examples/helloworld-dbg** directory to your working directory.
3. Open the project.
4. Compile the project.
5. Configure the FPGA using the Programmer, JTAG mode, and the **.bit** file located in the project's **outflow** directory.
6. Open the BRAM Initial Content Updater.
7. Click the memory named **mem** to select it.
8. Click the Select Memory Initialization File button.
9. Select the **reverse.inithex** file in the **helloworld-dbg** directory.
10. Click **Open**.
11. Click the Update Memory Content button.
12. Click the Regenerate Bitstream button.
13. Close the BRAM Initial Content Updater.
14. 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 **v1.inithex** and **v2.inithex** files in the **helloworld-dbg** directory to update the bitstream to show other blinking patterns. Additionally, you can create your own **.hex** file to change the blinking pattern.

Use Another Board

Using another board is easy. First, change the device and family:

1. Choose **File > Edit Project**. The Project Editor dialog box opens.
2. Choose the device family in the **Family** drop-down list box.
3. Click **Select** next to **Device**. The **Device Selector** dialog box opens.
4. Choose the device on your board and click **OK**.

After you change the device, the software prompts you that the design has changed and you need to re-run the tool flow. Click **OK**. Then it displays the **Migrate Design** dialog box. You do not want to save the old interface:

1. Choose **Create New Design**.
2. Turn off the **Enable Design Backup** option.
3. Click **Next**.
4. The software gives a message that migration completed. Turn on the **Open Interface Design** option.
5. Click **Finish**. The Interface Designer opens.

Now you can import the correct interface design for your board:

1. Choose **File > Import Design**. The Import Design dialog box opens.
2. Choose **Interface Scripting File (.isf)** and click **Next**.
3. Click the file button to browse for the **.isf**.

4. Choose the **.isf** for your board.
5. Click **Finish**. The Interface Design loads the settings.
6. Save.

Run the BRAM Initial Content Updater from the Command Line

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

```
efx_bram_edit[.exe] [--help] --project PROJECT [--mem_info MEM_INFO] [--place PLACE]
    [-lbf LBF] [--output OUTPUT] [--family FAMILY] [--verbose] --memory MEMORY [--mode MODE]
```

Table 28: BRAM Initial Content Updater CLI Options

Option (long)	Option (short)	Input	Description
--help	-h	None	Show the help.
--project	-j	Filename	Specify the Efinity project file.
--mem_info	-i	Filename	Specify the memory file in protocol buffer format.
--place	-p	Filename	Specify a placement file.
--lbf	-l	Filename	Specify the Logical Bit File (**.lbf**).
--output	-o	Filename	Specify the name for the updated bitstream file.
--family	-f	Family name	Indicate the FPGA family, Trion®, Topaz, and Titanium.
--verbose	-v	Nonr	Print out verbose messages.
--memory	-b	Filename	Specify the logical memory you want to update and the memory initialization file. Use the format *<memory>,<initialization file>*
--mode	-m	update, read, revert	Indicate the mode for the update tool. `update`: Default. Use to update 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.

The following example command runs the BRAM Initial Content Updater on the **pt_demo** design:

Example: Using the BRAM Initial Content Updater

```
efx_bram_edit --project pt_demo.xml --memory mem,new_mem.hex
```

Working with Bitstreams

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

Edit the Bitstream Header

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

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

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

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

Bitstream Compression

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

Refer to AN 033: Configuring Titanium FPGAs for the bitstream sizes.

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

Export to Raw Binary Format

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

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

You can also convert the file to **.bin** at the command line as described in Convert to Intel Hex Format at the Command Line on page 130.

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

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

Table 29: export_bitstream.py Positional Arguments

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

Table 30: export_bitstream.py Options

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

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

Example: Converting Hex to Bin

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

Combine Bitstreams and Other Files

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

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

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

Table 31: Modes when Combining Images

Mode	Use For	Number of Images	Notes
Selectable Flash Image	Multi-image configuration	Up to 4	Use this mode if you want the CBSEL pins to control which image the FPGA loads. For this mode, you also need to choose **Image Type** > **External Controller Flash Image**. See Program Multiple Images (CBSEL) on page 133
	Internal reconfiguration	Up to 4	Use this mode if you want the internal reconfiguration pins to determine which image the FPGA loads. For this mode, you also need to choose **Image Type** > **Remote Update Flash Image**. See Program Multiple Images (Internal Reconfiguration) on page 134
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors [10].	See Program a Daisy Chain on page 135
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	See Program Multiple Images (Bitstream and Data) on page 135

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

[10] Efinity Programmer does not apply any constraints for combining multiple images. Efinix recommends that you run the IBIS simulation to check the signal integrity if you need to connect more than 4 devices in the same daisy chain.

Combine Bitstreams at the Command Line

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

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

Table 32: BRAM Initial Content Updater CLI Options

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

Example: Combining Bitstream Images

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

SPI Programming

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

Program a Single Image

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

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

Program Multiple Images (CBSEL)

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

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

Program Multiple Images (Internal Reconfiguration)

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

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

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

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

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

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

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

Program Multiple Images (Bitstream and Data)

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

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

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

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

Program a Daisy Chain

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

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

JTAG Programming

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

Trion Family JTAG Device IDs

The following table lists the Trion JTAG device IDs.

Table 33: Trion JTAG Device IDs

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

Titanium Family JTAG Device IDs

The following table lists the Titanium JTAG device IDs.

Table 34: 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
Ti125	All	0x006E0A79
Ti135	All	0x006C0A79
Ti165	All	0x006A1A79
Ti180	M484	0x00680A79
	J361, J484, J484D1, G400, G529	0x00690A79
	L484	0x00684A79
Ti240	All	0x006A2A79
Ti375	All	0x006A0A79

Topaz Family JTAG Device IDs

The following table lists the Topaz JTAG device IDs.

Table 35: Topaz JTAG Device IDs

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

Program a Single Image

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

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

Program Using a JTAG Chain

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

Trion FPGA example:

```
<?xml version="1.0"?>

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

Titanium Topaz FPGA example:

```
<?xml version="1.0"?>

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

where:

- `chip_num` is the device order starting from position 1.
- `id_code` is the hexadecimal JEDEC device ID (all lowercase letters)
- `ir_width` is the width of the instruction register in bits
- `istr_code` is the binary IDCODE instruction

Note: To create a Test Data In (TDI) connection, use `chip_num="1"` as the first device.

Note: For Trion FPGAs, use 1100 as the `istr_code`.

Note: For Titanium Topaz FPGAs, use 11000 as the `istr_code`.

To program using a JTAG chain:

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

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

Program using a JTAG Bridge

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

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

Note: The JTAG bridge modes were changed in the Efinity software v2025.1. If you are using an older version of software and want to use the SPI Active using JTAG Bridge (Legacy) mode, refer to **Appendix: Program using a JTAG Bridge (Legacy)** on page 283.

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

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

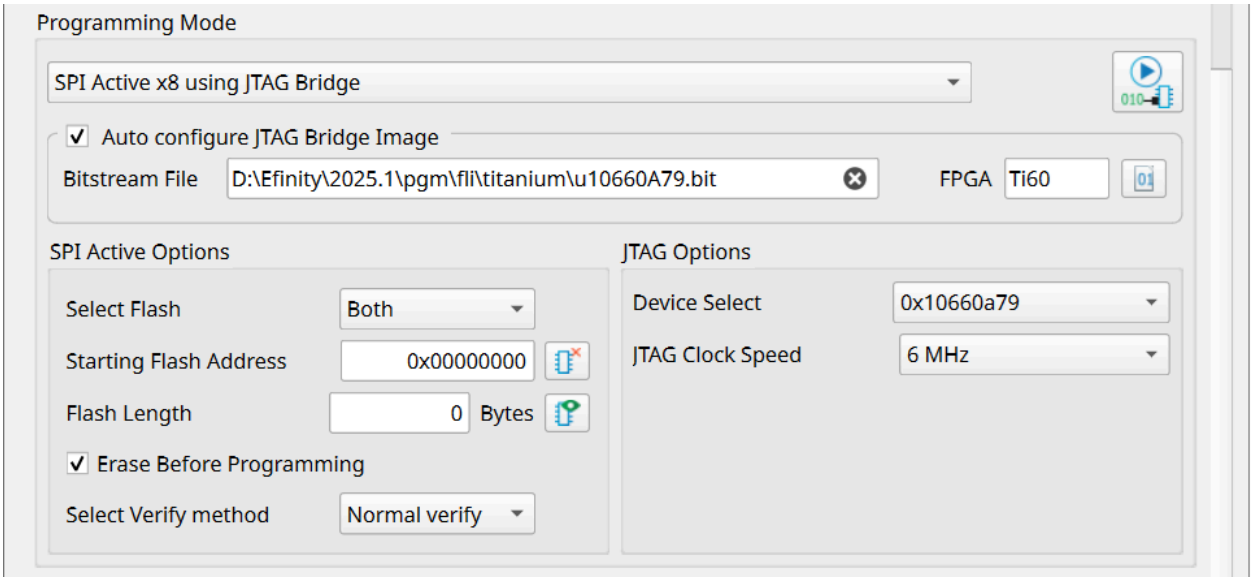
The Titanium and Topaz **.bit** files include a custom JTAG USERCODE in the bitstream:

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

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

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

Figure 52: SPI Active Using JTAG Bridge Options



To program using a JTAG bridge:

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

4. Turn on the **Auto configure JTAG Bridge Image** option.

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

5. Specify your own **.bit** file.

   a) In the **Programming Mode** box, click **Select Image File**.

   b) The **Open Image File** dialog box opens to a directory of available pre-built **.bit** files.

   Choose the file for your FPGA (Trion), or browse to find your own **.bit** file.

   The Programmer remembers which file you specify and uses it automatically the next time you run the Programmer.

6. Choose the **SPI Active Options** and **JTAG Options**.

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

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

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

Refer to **Using the Efinity Bitstream Security Key Generator** on page 172 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 program the flash with JTAG Bridge by connecting to a soft JTAG tap IP and four GPIOs. Refer to:

- **Blowing Fuses with the SVF Player** on page 174
- **JTAG Command Support with Security Enabled** on page 176

JTAG Programming with FTDI Chip Hardware

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

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

1. Open the Efinity® software.
2. Open the Efinity® Programmer.
3. Click the Select Bitstream Image button.
4. Browse to your image and click **OK**.
5. Choose one of the following in the **USB Target** drop-down list:
   - **Dual RS232 HS** for FT2232H Mini Module
   - **FT4232H_MM** for FT4232H Mini Module
6. Choose **JTAG** from the **Programming Mode** drop-down list.
7. Click **Start Program**.

FTDI Programming at the Command Line

The Efinity software includes a Python script you can use for programming FTDI modules at the command line. The following command is for Windows. For Linux, use **ftdi_pgm.sh**.

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

Table 36: ftdi_pgm.py Positional Arguments

Argument	Description
input_file	HEX file generated from `efx_pgm`.

Table 37: ftdi_pgm.py Options

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

Option (Long)	Option (Short)	Input	Description
--mode	-m	passive, active, jtag, jtag_chain, erase_flash, read_flash, jtag_bridge, jtag_bridge_x8	Programming mode. See the **Efinity Programmer User Guide**. In Efinity software versions prior to v2025.1, the `jtag_bridge` and `jtag_bridge_new` options were named `jtag_bridge_new` and `jtag_bridge_x8_new`, respectively. (11) 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 <*installation directory*>/pgm/fli/<*family*>. Refer to the **JTAG SPI Flash Loader Core User Guide** for information on using the legacy flash loader.
--output_file	-o	Filename	Output file used for `read_flash` mode.
--url	-u	URL	FTDI URL (see **Identifying FTDI URLs** on page 145).
--aurl	-a	URL	Alternative URL (*Deprecated*).
--xml	-x	Filename	XML file for JTAG programming.
--num	-n	Number	Chip target number for JTAG chain programming.
--board_profile	-b	Generic Board Profile Using FT232, Digilent JTAG-HS3, FireAnt Development Board, Generic Board Profile Using FT2232H, ISX Programming Cable, Titanium Ti180J484 Dev Board, Titanium Ti180M484 Development Kit, Generic Board Profile Using FT4232, JinChen Programming Cable, TJ180A484S Development Kit, Xyloni Development Board, Generic Board Profile Using FT4234HA	Name of the board profile used.
--address	N/A	Hex number	Starting flash address for flash read and write operations.
--num_bytes	N/A	Number	Number of bytes to erase or read. For modes `erase` and `read` only.
--burst_size	N/A	Number	Individual read or write burst size in multiples of 256 bytes. For legacy JTAG bridge modes only (`jtag_bridge` and `jtag_bridge_x8`).
--jtag_bridge_mode	N/A	Erase, write, erase_and_write, read, all, all_no_erase	JTAG bridge programming mode.

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

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

Linux Examples

To program in Linux:

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

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

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

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

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

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

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

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

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

Windows Examples

To program in Windows:

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

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

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

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

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

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

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

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

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

Identifying FTDI URLs

Certain Efinity® scripts contain the `--url` and `--aurl` options, which require the input of an FTDI URL.

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.

The FTDI URL is in the format:

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

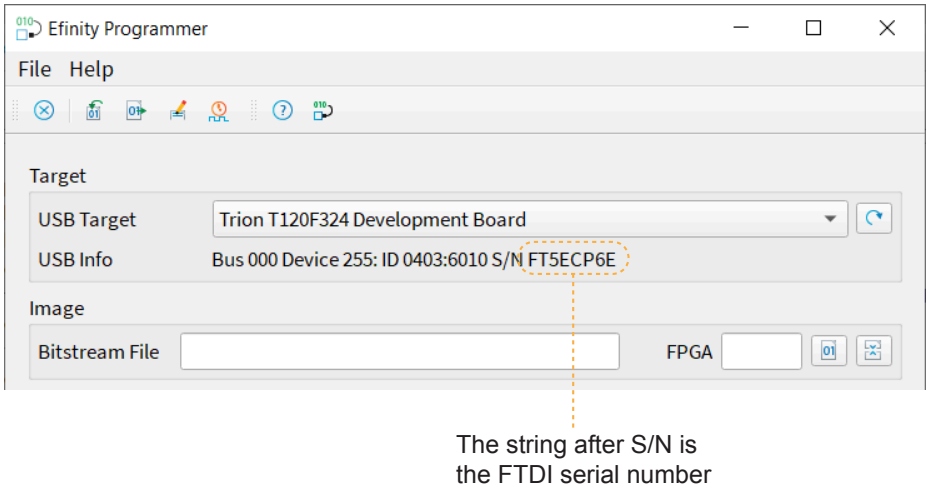
where:

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

<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 Titanium Ti375C529 Development Board Titanium Ti375N1156 Development Board
2232h	Titanium Ti180J484 Development Board
2232h	Topaz Tz170J484 Development Board

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

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



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

Programmer Messages

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

Error Messages

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

Message	ERROR: Incompatible file extension for programming mode, please use .hex file for SPI Active x8 using JTAG Bridge programming
Reason	SPI Active x8 using JTAG Bridge chosen as programming mode but **.bit** file specified in bitstream file.
To fix	Specify correct the **.hex** bitstream file.

Message	ERROR: Check board is plugged in, and then click on "Refresh USB Targets"
Reason	Board disconnected or powered off during programming.
To fix	Reconnect the board and click the refresh button.

Message	Image file not found
Reason	Bitstream file not loaded.
To fix	Load the bitstream file.

Message	Failure to configure was detected
Reason	Programmer internally failed to enter configuration mode.
To fix	Program again.

Message	Unable to configure from flash device.
Reason	JTAG state failed to enter USER mode.
To fix	Reprogram the bitstream.

Message	Unable to determine status of device.
Reason	1. JTAG programming in unknown state; potentially hardware issue. 2. Different width chosen to program the bitstream file. For example, the bitstream is x1 width but SPI Active x8 is chosen.
To fix	1. Try to reprogram the bitstream. If using C323HM cable, check the connectivity. 2. Choose the correct width or run the bitgen again.

Message	ERROR: Flash verify unsuccessful... mismatch found
Reason	1.Programmed flash does not match with the selected bitstream. 2. The board does not support the selected verify method.
To fix	1. Reprogram the bitstream. 2. Select "Normal verify."

Message	ERROR: Unable to verify JTAG interface, cannot determine configuration status
Reason	JTAG mode used to program the board, but JTAG interface 1 is unstable or the JTAG connection using C2323HM is incorrect or disconnected.
To fix	Check the driver for the JTAG interface or check the wire connection.

Message	ERROR: Unknown error trying to read flash device, aborting. Aborting flash programming FtdiProgram error: could not get flash device
Reason	Attempted to program the board via SPI Active or SPI Passive while interface 0 (SPI) is disabled.
To fix	Check the driver for the SPI interface. If it is unsupported (using C232HM), then it is not possible to program with SPI.

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

Message	ERROR: Cannot edit SPI Active clock settings for .bit file, please use the .hex file
Reason	Opened Edit SPI Active Clock on a **.bit** file.
To fix	Edit SPI Active Clock only works for **.hex** files.
Message	ERROR: Input file has been corrupted, unable to determine target device
Reason	Opened Edit SPI Active Clock on a corrupted **.hex** file.
To fix	Replace the corrupted **.hex** file to a valid one.
Message	ERROR: Unable to read input image file, file maybe have been corrupted
Reason	Opened Edit SPI Active Clock on a corrupted **.hex** file.
To fix	Replace the corrupted **.hex** file to a valid one.
Message	ERROR: Device code for JTAG Bridge image CANNOT be Unknown. Please ensure you are using correct bitstream file
Reason	1. Programmer is unable to detect the device code. 2. Connectivity issue. 3. USB driver missing for the JTAG interface. 4. Programmer is unable to detect non-Efinix device.
To fix	Ensure there is a valid device ID at Device Select.
Message	ERROR: Export feature only works with Efinity bitstreams
Reason	Export failed because the bitstream is not originally from Efinity.
To fix	Bitstream exports not originally from Efinity are not supported.
Message	ERROR: Cannot detect JTAG chain setup. Please import JTAG chain file
Reason	Programmer is unable to detect non-Efinix device.
To fix	Import the JCF.
Message	Calculated IR width is invalid. Please import JTAG chain file
Reason	A board in the chain is powered off.
To fix	Power on the board and click the refresh button.
Message	Total IR width of the previous JCF does not match actual total IR width. Please import JTAG chain file again.
Reason	Programmer is unable to detect non-Efinix device and cannot auto-detect IR length of the board in the JTAG chain.
To fix	Import the JCF.
Message	ERROR: The Programmer cannot detect the FPGA in the JTAG chain. Check the JTAG cable or header for connectivity issues
Reason	Incorrect connection of the chain or improperly connected wire.
To fix	Check the wire connectivity and check the IDCODE through the SVF Player.
Message	ERROR: Invalid ASCII character detected in header, cannot display header text
Reason	Bitstream header is incorrectly formatted.
To fix	Run the bitstream again to generate a new bitstream.

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

Info Messages

Message Debug Profile: <*file path*> is modified.

Reason Notifies the user that the currently loaded debug profile has been modified on disk since it was last loaded.

Message Please reload it.

Reason Tells the user to reload the modified debug profile to apply the latest changes.

Message Disconnected active debugger connections.

Reason The active JTAG/debugger connections were closed before reloading the profile.

Message Loaded Debug Profile from file: <*file path*>

Reason Confirms a debug profile was successfully loaded from the specified JSON file path.

Message Generated profile file = <*file path*>

Reason Reports the path of the debug JSON file that was just written to disk after the user clicked **Generate**.

Message Connecting to URL: <*URL*>

Reason Reports the FTDI/USB URL being used to open the JTAG session for the debugger connection.

Message Disconnected from JTAG session: <*session ID*>

Reason Confirms that the JTAG session was successfully closed after the user clicked **Disconnect**.

Message Settings from previous JTAG chain file loaded automatically.

Reason Indicates that device order and IR widths were resolved from a chached JCF without user intervention.

Message Generated report file <*file path*>

Reason Confirms that an image-combine report file was written to the given path after the user combined images.

Message Generated Multi Image output file: <FILE_PATH>
Device: <*device name*>
Image type: <*image type*>
CBSEL[1:0] Flash Address Length Input Image File
2'b00 <FINAL_ADDR> <FINAL_ADDR> <INPUT_FILE_PATH>
2'b01 <FINAL_ADDR> <FINAL_ADDR> <INPUT_FILE_PATH>
2'b10 <FINAL_ADDR> <FINAL_ADDR> <INPUT_FILE_PATH>
2'b11 <FINAL_ADDR> <FINAL_ADDR> <INPUT_FILE_PATH>

Reason Summary table after a Multi-Image output file is generated, showing the output file path; device; image type; and the flash address, length, and source file for each CBSEL[1:0] slot.

Message WARNING: Automatically changing lead device SPI clock to DIV4/20 MHz

Reason Warns that the lead device's SPI clock was automatically capped at DIV4 (20 MHz) to ensure compatability during multi-device daisy-chain image generation.

Message Generated Daisy Chain output file: <FILE_PATH>
Device Input Image File
<DEVICE_NAME_1> <FILE_PATH_1>
<DEVICE_NAME_2> <FILE_PATH_2>
<DEVICE_NAME_3> <FILE_PATH_3>

Reason Summary table printed after a daisy chain output file is generated, listing each device and its corresponding input image file.

Message Generated FPGA Soc Image output file: <FILE_PATH>
Flash Address Length Input Image File

Reason Summary table printed after an FPGA-SoC combined image is generated, showing each region's flash address, length, and source file.
<FLASH_START_ADDRESS_1> <DATA_LENGTH_1> <INPUT_FILE_PATH_1>
<FLASH_START_ADDRESS_2> <DATA_LENGTH_2> <INPUT_FILE_PATH_2>
<FLASH_START_ADDRESS_3> <DATA_LENGTH_3> <INPUT_FILE_PATH_3>

Message Successfully generate PEM File: <*file path*>

Reason Confirms that a PEM-format key file was successfully generated for a Titanium device.

Message Writing JTAG Disabling to separate SVF Command File...

Reason Notifies the user that the JTAG-disable commands are being written to a separate SVF file because "JTAG Disabling" is set to "ON" or "DISABLE_EFUSE_ONLY" and the "Use Separate SVF" option is checked.

Message Successfully generate key/svf files.

Reason Final confirmation that all key and SVF command files were successfully generated for the Titanium device.

Message Reading PEM File: <*file path*>

Reason Reports the PEM file path being read after the user selected an RSA PEM file.

Message Successfully read PEM File: <*file path*>

Reason Confirms that the specified RSA PEM file was parsed and imported successfully.

Message Imported JTAG chain file: <*file path*>

Reason Confirms that the JTAG chain file (JCF/XML) was successfully imported for use in the programming, debugging or Titanium key-generation flow.

Message Erasing entire flash...

Reason Indicates that a full-chip flash erase has started on the T8F81 development board.

Message ... finished erasing flash

Reason Confirms that the full-chip flash erase completed successfully on the T8F81 development board.

Message Writing '<*file path*>' to flash memory...

Reason Reports the start of writing the bitstream/image file to the SPI flash on the T8F81 development board.

Message ... finished passive programming

Reason Confirms that SPI Passive (direct-to-FPGA) programming on the T8F81 development board completed successfully.

Message Successfully modified header for image file <*file path*>

Reason Confirms that the bitstream header was patched and the modified image was written to the destination file.

Message JTAG chain target changed to device <DEVICE_INDEX>: <JTAG_IDCODE>

Reason Reports that the active JTAG chain target was switched to a different device index and shows the new device's IDCODE.

Message Valid device ID found: <JTAG_IDCODE>

Reason Confirms that the JTAG scan returned a recognised Efinix IDCODE for the currently selected device.

Message Device is in user mode!

Reason Confirms that the FPGA device has entered user mode.

Message Device configuration finished, ready to enter user mode upon release of CDONE

Reason Indicates the FPGA finished configuration and is waiting for the CDONE pin to be released before entering user mode.

Message SPI Active settings opened!

Reason Confirms that the SPI Active clock-settings dialog has been opened and current settings have been read from the bitstream.

Message No settings have been changed yet

Reason Informs the user that they attempted to generate a new bitstream without having modified any SPI Active clock settings.

Message Generating bitstream, may take a while...

Reason Notifies the user that the bitstream is being regenerated with the updated SPI Active clock settings and warns it may take time.

Message Done, bitstream overwritten with new settings!

Reason Confirms that the bitstream file was successfully overwritten with the new SPI Active clock configuration.

Message SPI Active settings closed, new settings have been applied

Reason Confirms that the SPI Active clock-settings dialog was closed and the modified settings were saved.

Message SPI Active settings closed, no changes have been made

Reason Confirms that the SPI Active clock-settings dialog was closed without any settings being changed.

Message Non-AN35 Device Detected: Disabling NegEdge sampling and extra cycles

Reason Informs the user that the loaded bitstream targets a non-AN35 device, so negative-edge sampling and extra clock cycle options are automatically disabled.

Message Successfully generated *<file path>*

Reason Confirms that a bitstream file was successfully exported and saved to the given path.

Message Jtag2Apb version: *<version>*

Reason Reports the version code read from the Jtag2Apb bridge when connecting to the transceiver debugger. Used to verify RTL compatibility.

Message Using cached file "*<file name>*" on server.

Reason Informs the user that the file already exists on the remote hardware server, so re-uploading is skipped.

Message Uploading file "*<file path>*" to server...

Reason Reports that a local file is being transferred to the remote hardware server because no matching cached copy was found.

Message Downloading the file from server to "*<file path>*"...

Reason Reports that a result file is being downloaded from the remote hardware server to the local output path.

Message Successfully wrote *<file path>*

Reason Confirms that an Intel HEX or a raw HEX export of the bitstream was successfully written to the specified file.

Message Successfully wrote *<file path>* in *<seconds>*

Reason Confirms that the Intel HEX file was exported to a raw HEX file and reports the time taken for the export operation.

Message Warning: GPIOs are not set for *<BUS>*-bus due to driver not available.

Reason Warns that GPIO pins for the specified FTDI bus (SPI/JTAG) could not be configured because the FTDI driver is unavailable on this system.

Message Found matching Flash profile from Efinix Flash JSON Database.

Reason Confirms that the detected SPI flash device matched a known entry in the Efinix flash profile database, so vendor-specific settings will be used.

Message Warning: Detected general exception while checking 4byte address mode flag - *<error message>*

Reason Warns that an unexpected exception occurred when reading the flash's 4-byte address mode status register. The check is skipped and programming continues.

Message Enable 4-byte address mode = True

Reason Confirms that 4-byte (32-bit) address mode has been enabled on the SPI flash to support devices larger than 128 Mbit.

Message Disable 4-byte address mode = True

Reason Confirms that 4-byte address mode has been disabled (reverted to 3-byte/24-bit addressing) on the SPI flash before programming.

Message Overwrite Address mode to 3-bytes

Reason Reports that the flash address mode is being forced back to 3-byte mode. For Micon MT25Q SPI flash only.

Message Finished read in <*seconds*>

Reason Reports the elapsed time for a flash read operation to complete.

Message Restore Non-volative configuration register.

Reason Reports that the flash's non-volatile configuration register is being restored to its original value after programming. For Micon MT25Q SPI flash only.

Message Exit 4-byte address mode

Reason Reports that the flash is being taken out of 4-byte address mode as part of post-programming cleanup.

Message Unlock all sectors for write.

Reason Reports that all write-protected sectors of the SPI flash are being unlocked by clearing the status register before an erase or write operation.

Message Erasing <*size*> from flash @ <*frequency*> (may take a while...)

Reason Reports the start of a flash erase, showing the region size and start address. Warns the operation may take significant time.

Message Finished erase in <*seconds*>

Reason Reports the elapsed time for the flash erase operation.

Message Restore Write-protect status register.

Reason Reports that the flash's status register is being restored to its original state after an erase/write operation. The status register might be cleared to unlock sectors.

Message SPI Active Programming on <*URL*>

Reason Reports the start of SPI Active programming and the FTDI URL being used to communicate with the flash.

Message Warning: Detected general exception while processing bitstream header - <*error message*>

Reason Warns that an unexpected error occurred when parsing the bitstream header. The hex file provided may not be a bitstream file. Programming may proceed with default settings.

Message Warning: Detected general exception '<*error message*>' while enabling QE bit

Reason Warns that an unexpected error occurred when attempting to determine the data width of the bitstream file; quad mode of the SPI flash will not be enabled.

Message Finished write in <*seconds*>

Reason Reports the elapsed time for the flash write (programming) operation.

Message Flash verify successful

Reason Confirms that the data read back from flash after programming matches the expected image; the bitstream was written correctly.

Message "SPI active" programming...done

Reason Confirms that the full SPI Active programming sequence (erase, write, optional verify) was completed successfully.

Message SPI passive programming on <SPI_URL>

Reason Reports the start of SPI Passive (direct-to-FPGA) programming and the FTDI URL being used.

Message <SPI_URL>

Reason Prints the resolved FTDI URL for the current SPI Passive programming target.

Message Finished SPI direct program

Reason Confirms that direct SPI (passive) programming of the FPGA is complete.

Message jtag programming started!

Reason Reports that the JTAG programming sequence has started.

Message JTAG Chain Programming on <JTAG_URL>, target: <DEVICE_INDEX>

Reason Reports the FTDI URL and target device index being used for JTAG chain programming when multiple devices are in the chain.

Message JTAG Programming on <JTAG_URL>

Reason Reports the FTDI URL being used for single-device JTAG programming.

Message Warning: <*bus*>-bus could not be opened due to driver not available.

Reason Warns that the specified FTDI GPIO bus could not be opened because the required driver is not installed or accessible.

Message Device ID read from JTAG: <JTAG_IDCODE>

Reason Reports the 32-bit IDCODE value read from the JTAG TAP during JTAG programming initialisation.

Message ... finished with JTAG programming

Reason Confirms that JTAG programming of the FPGA is complete.

Message We will only support same flash type for x8 Programming!

Reason Informs the user that x8-width JTAG2SPI programming requires all flash devices to be of the same type.

Message Unrecognized Flash device. Will use Generic Flash profile. Please contact support if you face any problem.

Reason Warns that the connected SPI flash was not found in the database; a generic fallback profile will be used. Advises the user to contact support if issues arise.

Message Flash device: <*flash name*> @ JTAG freq <*frequency*> MHz

Reason Reports the identified SPI flash device name and the JTAG clock frequency being used for JTAG-to-SPI programming.

Message Connecting to JTAG_TAP: <*TAP type*>

Reason Reports the specific JTAG TAP type being targeted when setting up the JTAG engine for JTAG-to-SPI programming.

Message Erasing <*size*> from flash address @ <*frequency*> (may take a while...)

Reason Reports the start of an erase during JTAG-to-SPI programming, showing the region size and start address.

Message the flash doesn't support Dual mode. Fallback to x1 mode

Reason Warns that the connected flash does not support x2 (Dual SPI) mode. Programming falls back to x1 single-bit mode.

Message the flash doesn't support Quad mode. Fallback to x1 mode

Reason Warns that the connected flash does not support x4 (Quad SPI) mode. Programming falls back to x1 single-bit mode.

Message JTAG2SPI dual flash programming...done

Reason Confirms that JTAG-to-SPI programming in dual-flash (x8) mode completed successfully.

Message JTAG2SPI programming...done

Reason Confirms that JTAG-to-SPI flash programming in single-flash mode completed successfully.

Message sync pattern check: found <*received sync pattern*> but expected <*expected sync pattern*>... assuming read data is not a programmed bitstream

Reason Reports that the sync pattern read back from flash does not match the expected value. The region is treated as unprogrammed (blank or corrupted).

Message WARNING: expected x8 spi width bitstream, found <*data width*> instead

Reason Warns that the user started flash programming in dual-flash (x8) mode but the data width in the bitstream header is not 8, which may indicate a configuration mismatch.

Message Flash device: <*flash name*> @ SPI freq <*frequency*> MHz

Reason Reports the identified SPI flash device name and the SPI clock frequency being used for SPI Active programming.

Message Erasing full chip (may take a while...)

Reason Reports the start of a full-chip erase. Warns the user the operation may take a significant amount of time.

Message Falling back to x1 mode

Reason Warns that the requested multi-bit SPI mode (x2/x4) is unavailable and programming is falling back to x1 (single-bit) mode. For new SPI backend path.

Message Dual flash programming...done

Reason Confirms that flash programming in dual-flash (x8) mode completed successfully. For new SPI backend path.

Message Flash programming...done

Reason Confirms that flash programming in single-flash mode completed successfully. For new SPI backend path.

Message WARNING: Unable to confirm x1 width for JTAG programming, issues may be encountered...

Reason Warns that the width check could not verify x1 mode for the selected bitstream; programming will proceed but may fail.

Shown when "Allow .bit/.hex files for all configuration modes" is enabled and the target device is Trion family.

Message Warning: Non-bitstream file detected.

Reason Warns that the selected image file does not appear to be a valid Efinix bitstream.

Shown when the "Auto configure JTAG Bridge Image" option is turned on.

Message Auto programming JTAG Bridge image <*file path*>...

Reason Reports that the JTAG Bridge image is being auto-programmed.

Message Advanced device status feature disabled, unable to verify JTAG connection with this device

Reason Informs the user that the advanced device status table is disabled because the JTAG interface is not available.

Message Custom JTAG clock speed set to <*JTAG frequency*> Hz

Reason Confirms that the user-entered custom JTAG clock frequency has been applied for subsequent JTAG operations.

Message Auto configure JTAG Bridge Image = <*file path*>

Reason Reports that the JTAG Bridge image file will be used for JTAG Bridge programming.

Message Starting to program JTAG Bridge Image

Reason Indicates the beginning of JTAG Bridge image programming.

Message Executing file <*hook file path*>

Reason Reports that the pre-programming or post-programming SPI script file is being executed before or after the main programming operation begins.

Message Programming operation completed.

Reason Confirms that the entire programming sequence, including any pre- or postscripts, finished without a fatal error.

Message Force stop programming

Reason Records a timestamped entry that the user manually stopped the programming operation in progress.

Message Board Profile Selected: <*profile name*>

Reason Confirms the name of the board profile that was automatically selected for the current USB target.

Message No Board Profile Selected

Reason Informs the user that no matching board hardware profile was found for the current USB target. Default pin settings will be used.

Message Switching to SPI mode because JTAG is not connected…

Reason Informs the user that JTAG connectivity could not be verified for the selected USB device, so the programming mode is automatically switched to SPI Active.

Message Imported JTAG chain file: <*file path*>

Reason Confirms that a JTAG Chain File (JCF) was successfully imported.

Message Please select JTAG programming mode to update device chain list before refreshing status

Reason Instructs the user to switch to a JTAG programming mode first if a JTAG Chain File (JCF) is imported so the device chain list is populated before attempting a status refresh.

Message Refresh status feature disabled, unable to verify JTAG connection with this device

Reason Informs the user that the status-refresh stopped because the JTAG interface selected device cannot be verified.

Message Refreshing device status...

Reason Indicates that a JTAG device-status read is in progress after the user clicks **Refresh**.

Message Closing connection to <*server address*>...

Reason Reports that the connection to a remote hardware server is being closed after the user deselects it from the remote host list.

Message No remote host configured.

Reason Informs the user that no remote hardware server URL was provided. The tool will fall back to local mode.

Message Note: Fast verify requires SPI x4 width connections from flash memory to FPGA

Reason Notes that the fast verify feature requires a quad-width (x4) SPI connection between the flash and FPGA.

Message Migrated hw_tools.ini to hw_tools.json

Reason Informs the user that the legacy INI-format hardware tools config file was automatically converted to the new JSON format.

Message Programming Device <*device index*>

Reason Informs the user of the target JTAG chain device index being used at the start of JTAG programming. The first target has an index of 1.

Message revert to local backend

Reason Reports that the hardware backend has been reset to local mode after the remote server connection was lost.

Message No programming cable detected, cannot program

Reason Reports that programming could not proceed because no USB device was found.

Message Image file <*file path*> not found

Reason Reports that the image file specified for programming does not exist on the file system.

Message do not know how to program for <*URL*> based usb controller

Reason Reports that the detected USB controller type is not FTDI or Atmel and is therefore unsupported by the programmer.

Message Using FTDI URL (SPI = <*SPI_URL*>, JTAG = <*JTAG_URL*>)

Reason Reports the two FTDI channel URLs (SPI bus and JTAG bus) being used for the current programming session.

Message Check board is plugged in, and then click on "Refresh USB Targets"

Reason A USB-related error has occurred. Advises the user to check the board connection and refresh the USB target list.

Message Sent command <*address*>. Read <*data length*> byte(s) from SPI Flash

Reason Logs the SPI flash register read command sent and the raw bytes received in the Send SPI Flash Commands dialog.

Message Updated advanced device status register readings

Reason Confirms that the advanced JTAG status register values were successfully read and the status panel has been refreshed. For Titanium devices only.

Message Warning: an unexpected <*error message*> occurred when claiming driver interface <*interface number*>

Reason Warns that an unexpected error occurred when claiming FTDI USB driver interface index *i* when resolving USB devices.

Message Warning: FTDI Channel Index collided, Issues with driver installation and may cause error in programming.

Reason Warns that two FTDI channels resolved to the same index, indicating a driver installation problem that may cause programming failures.

Message ... processed line <*current line number*> of <*total line number*> ...

Reason Periodic progress update emitted by a timer during SVF playback, showing the current line number against the total number of SVF commands.

Message Starting <*file path*> SVF execution...

Reason Reports that SVF playback has started for the specified SVF file.

Message Finished executing <*file path*>

Reason Confirms that all SVF commands in the file were processed or execution was stopped by the user.

Message Last execution line: <*last line number*> (EOF)

Reason Reports the last SVF command line that was executed. "EOF" (end of file) is shown if the entire file was processed.

Message Closing JTAG interface...

Reason Reports that the JTAG session is being closed at the end of SVF execution, or after an error or stop.

Message Finish Execution!

Reason Final confirmation banner displayed in the GUI after SVF execution completes.

Using the Command-Line Programmer

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

Example: Command-Line Programmer

Linux:

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

Windows:

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

Options

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

Table 38: --pgm_opts Modes

Mode	Description
active	SPI Active configuration.
passive	SPI Passive configuration.
jtag	JTAG programming. See the **Efinity Programmer User Guide** for more information about programming with the JTAG interface.
jtag_bridge	SPI Active using JTAG bridge mode.
jtag-bridge_x8	SPI Active x8 using JTAG bridge mode (used with two flash devices). [12]

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

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

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

Note: See **Programming Options** for more programming options.

[12] 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 39: 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 **Falling**. 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 53: Setting Programming Options (Trion)

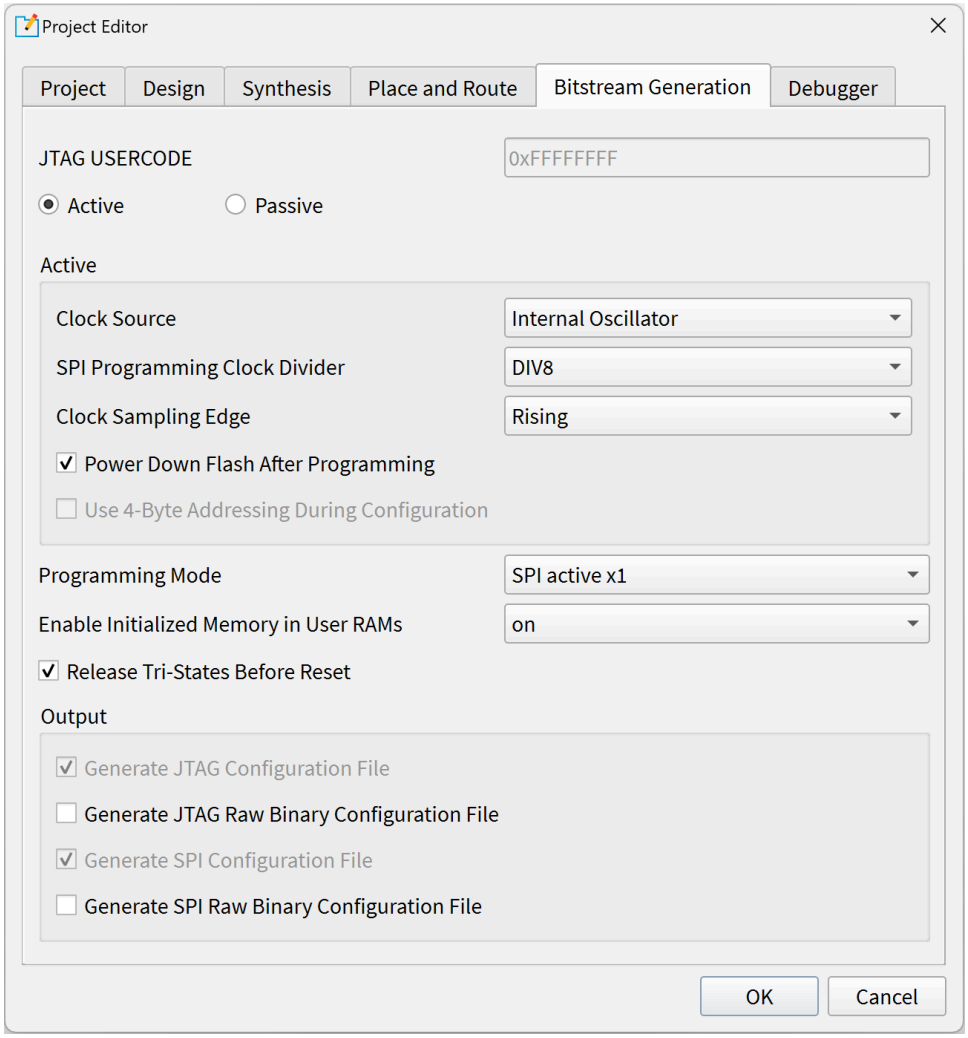
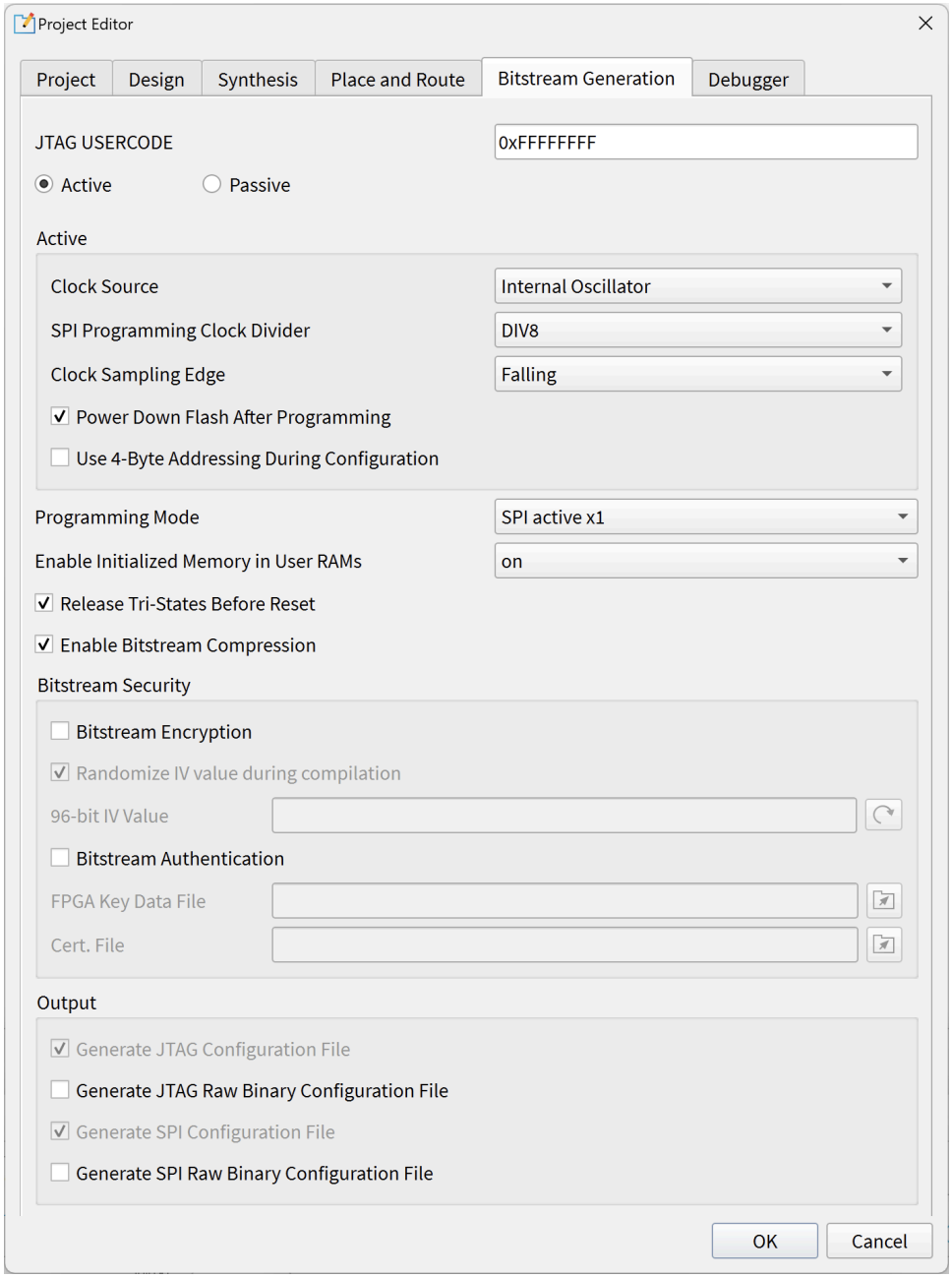


Figure 54: Setting Programming Options (Titanium Topaz)



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 40: 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: Configuration 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 in the golden image during remote update configuration.
CONFIG_END	0: Configuration is not complete. 1: Configuration completed (whether successful or not).
SYNC_PAT_FOUND	0: Indicates that the FPGA is not receiving the expected synchronization pattern at start of the bitstream. Check for board or power issues. 1: Indicates that the FPGA detected a synchronization pattern at start of the bitstream., and the clock and data connections to the FPGA are acceptable. Any configuration problems are likely digital or logical in nature. After successful configuration the status will return to 0.
SEU_ERROR	0: No SEU detection errors were found. 1: An SEU detection error was found when reading back the SEU CRAM. Has the same value as the SEU detection error status signal to the core fabric.
CRC32_ERROR_PERIPH	0: No CRC errors were detected in the interface configuration bits. 1: One or more CRC errors were detected in the interface configuration bits.
AES256_PASS	For an encrypted bitstream: 0: Decryption failed. The encryption keys used in to program the fuses may not match the ones used to encrypt the bitstream 1: The encrypted bitstream was decrypted successfully. If the bitstream is not encrypted, this register is always a 1. **Note:** This bit is not supported in Ti60ES FPGAs.
RSA_PASS	When using RSA authentication: 0: The signature check failed. The RSA keys used to program the fuses may not match the ones used to sign the bitstream in the Efinity project. 1: The bitstream signature was verified successfully If RSA authentication is not used, this register is always a 1. **Note:** This bit is not supported in Ti60ES FPGAs.

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

Verifying Configuration with the Programmer

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

Securing Titanium Bitstreams

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

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

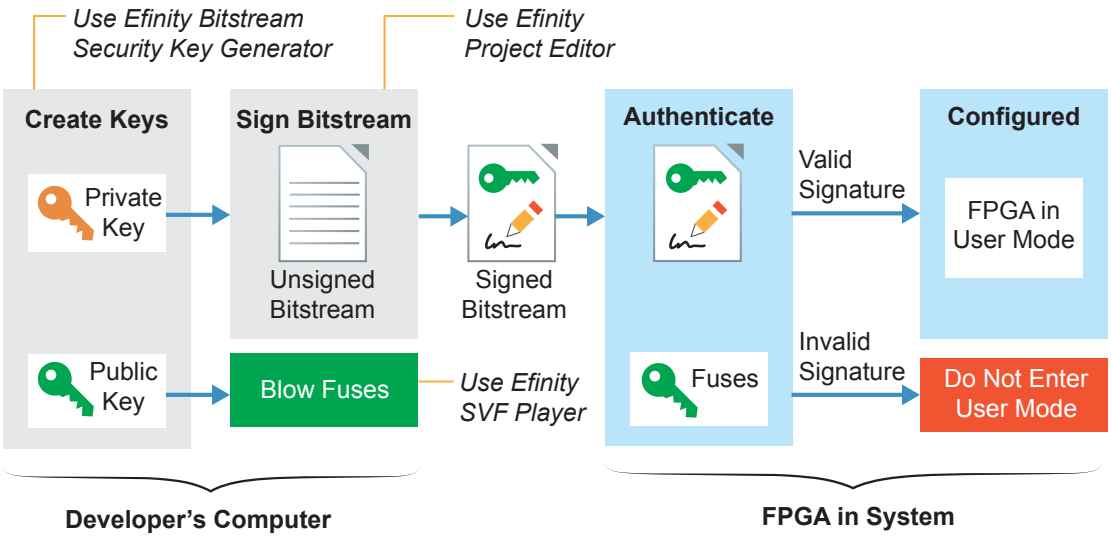
Note: Refer to JTAG Command Support with Security Enabled on page 176 for details on the JTAG disabling modes and which commands they support.

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

Table 41: Efinity Tools Used for Securing Bitstreams

Tool	Used for
 Bitstream Security Key Generator	Create or specify an AES-256 key. Create or specify an RSA-4096 private key. Specify whether to disable JTAG.
 SVF Player	Program the fuses in the Titanium FPGA with the AES-256 key and/or RSA certificate data. After you blow the fuses with an RSA key, the FPGA only accepts a bistream signed with the correct private key. After you blow fuses with an AES-256 key, the FPGA only accepts a plaintext bitstream or a bitstream signed with the correct key. Program the JTAG fuse to disable JTAG function.
 Project Editor	Turn on bitstream encryption and/or authentication, and specify the **.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 55: 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 56: Bitstream Encryption

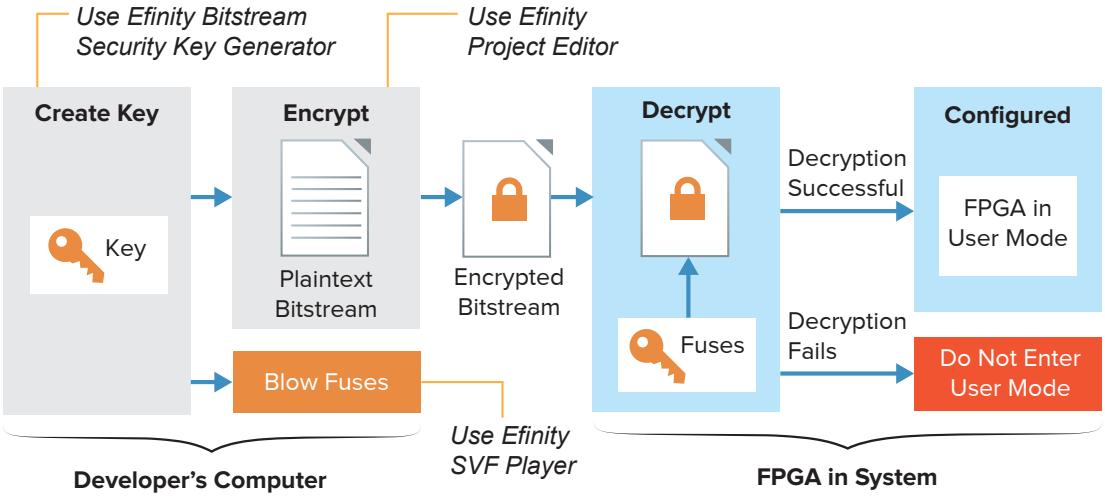
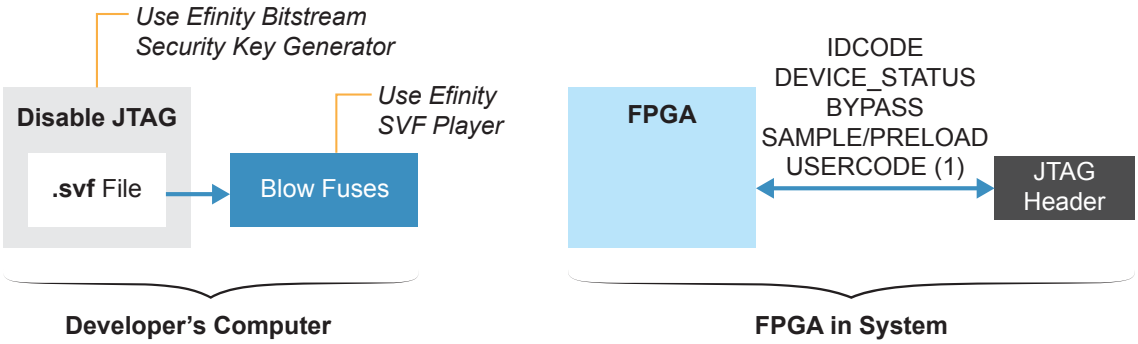


Figure 57: Disabling JTAG



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

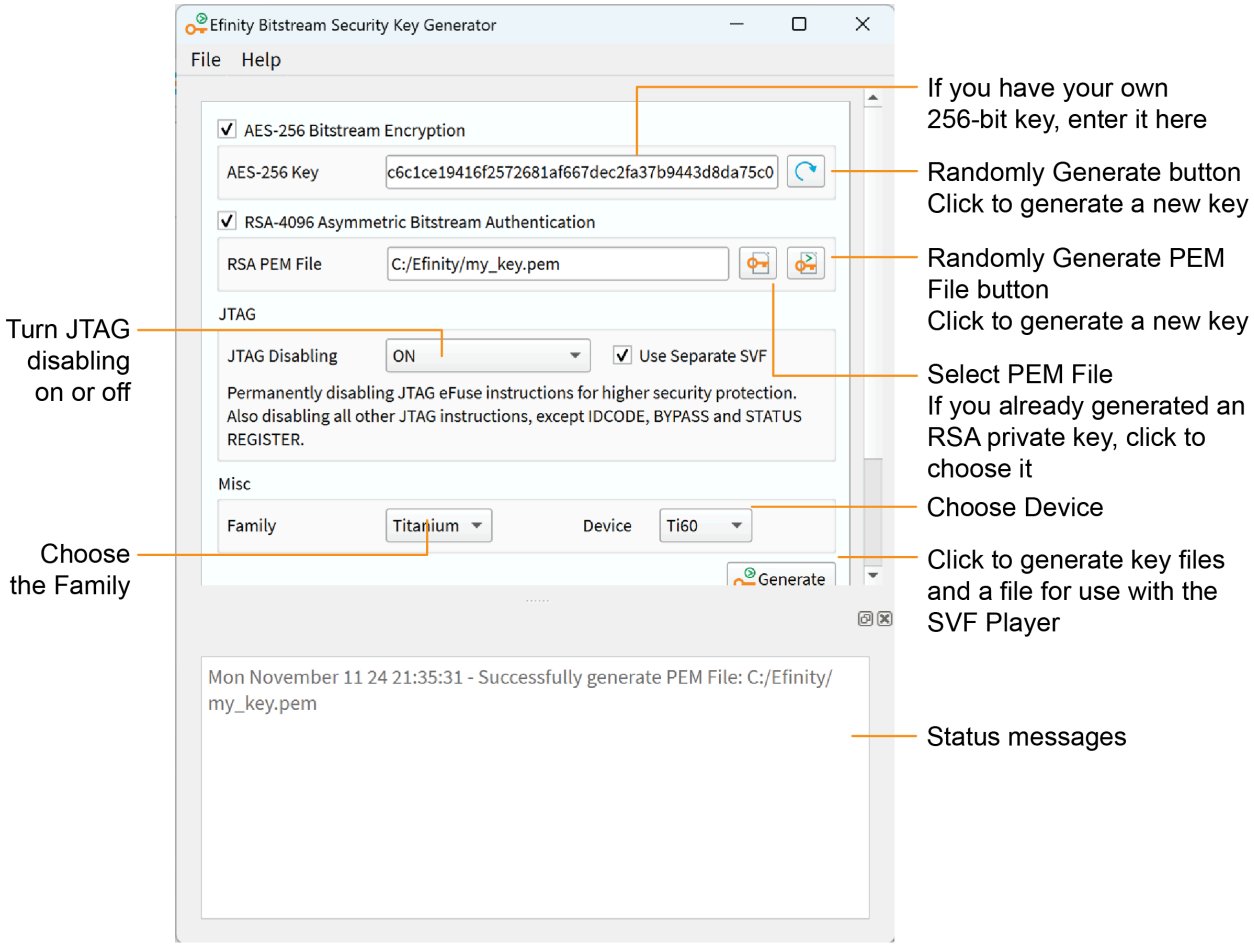
The following sections describe how to use each of these tools to enable security features.

Using the Efinity Bitstream Security Key Generator

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

Note: You can use the Efinity Bitstream Security Key Generator iteratively. That is, you can first use encryption and later add in RSA authentication, and even later disable JTAG commands. Refer to Workflow for Using Security Features on page 178 for more information.

Figure 58: 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 for those used to get device information.
DISABLE_EFUSE_ONLY	Permanently disables the JTAG efuse instructions only. Other JTAG instructions are not affected, for example, you can still perform debugging.

> **Note:** See JTAG Command Support with Security Enabled on page 176 for details.

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

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

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

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

The tool creates the following files:

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

Note: 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 178). 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.

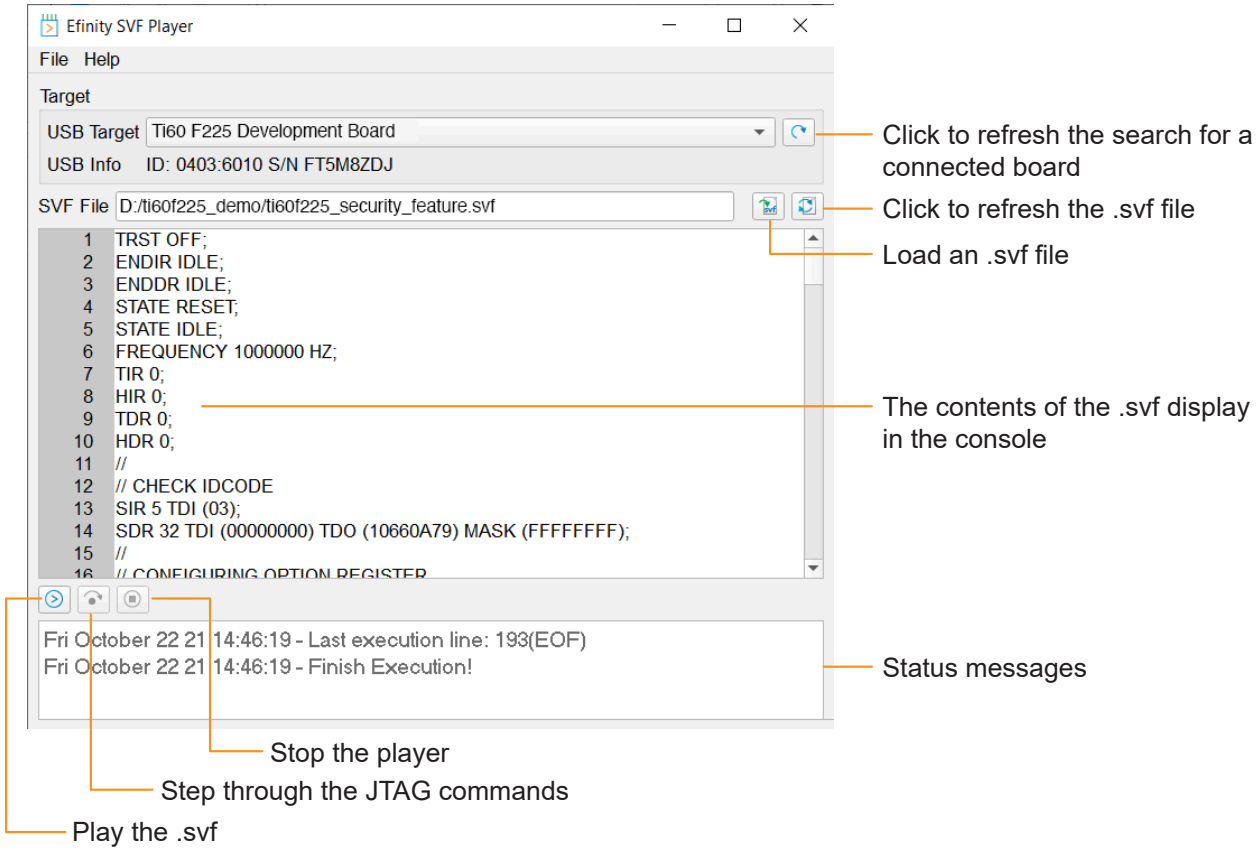
The **.svf** used for blowing fuses performs a variety of JTAG commands.

- 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 59: SVF Player



To blow fuses with the SVF Player:

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

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

3. Click the Play button to play the **.svf** file. It takes a very short amount of time to blow fuses.
4. Toggle `CRESET_N` or power cycle your board for the new fuse settings to take effect.

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

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

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

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 42: 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**).

JTAG Command Support with Security Enabled

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

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

The following table shows the commands supported for each mode.

Table 43: Allowed JTAG Commands with Security Enabled

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

Refer to the following topics for details:

- **Securing Titanium Bitstreams** on page 169
- **Using the Efinity Bitstream Security Key Generator** on page 172

Encrypt or Sign Bitstreams from the Command Line

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

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

Table 44: AddSecurityTitanium.py Positional Arguments

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

Table 45: AddSecurityTitanium.py Options

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

Example: Sign and Encrypt a File

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

Workflow for Using Security Features

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

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

Blowing Fuses Iteratively

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

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

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

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

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

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

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

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

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

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

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

Securing Bitstreams Iteratively

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

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

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

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

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

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

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

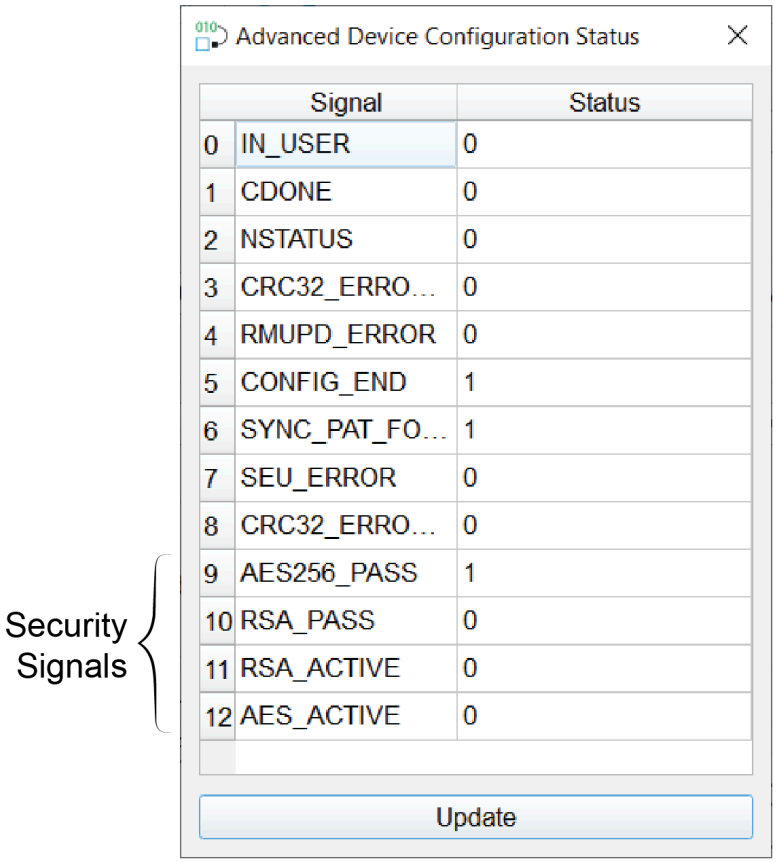
Verifying Security Settings

You may want to verify that your Titanium FPGA is correctly using the security features that you enabled. You can use the **Advanced Device Configuration Status** dialog box (Programmer) to view the security status signals. See Configuration Status Register on page 166 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 60: 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 12

Working with JTAG .svf Files

Contents:

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 129.

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:

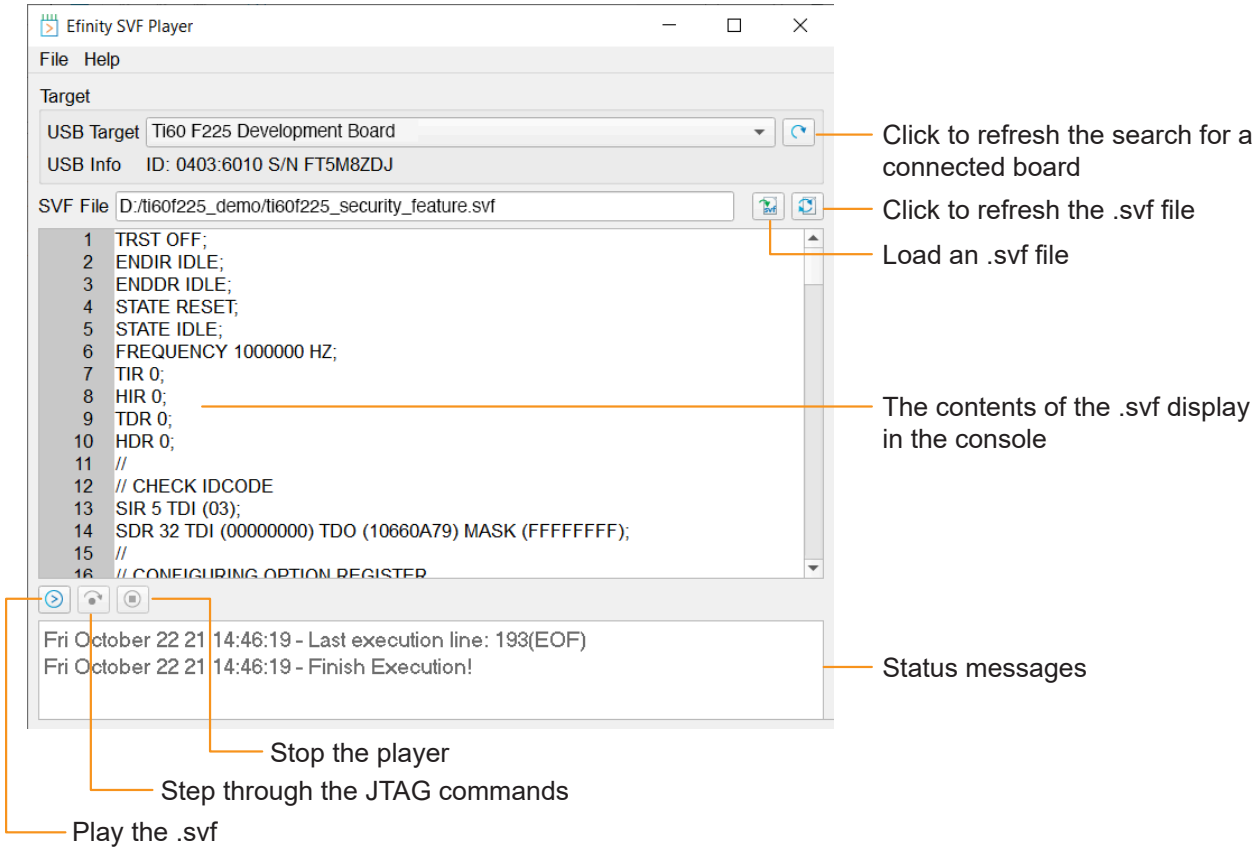
- Trion T20 in F324 and F400 packages
- Trion T35 in any package
- Trion T55, T85, and T120 in any package
- All Titanium and Topaz FPGAs in any package

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

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

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

Figure 61: SVF Player



To use the SVF Player:

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

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

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

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

Export JTAG Operations at the Command Line

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

When using the **export_bitstream_flashloaderv3.py** script, you must connect to the FPGA board using a download cable.

```
export_bitstream_flashloaderv3.py [--help] --jtag_bridge_file JTAG_BRIDGE_FILE [--freq FREQ]
    [--start_address START_ADDRESS] [--target TARGET] [--chip_num CHIP_NUM]
    [--jcf JCF] [--url URL] [--verify_method VERIFY_METHOD] input_file output_file
```

Table 46: export_bitstream_flashloaderv3.py Positional Arguments

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

Table 47: export_bitstream_flashloaderv3.py Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--jtag_bridge_file	N/A	File path	The file path of the JTAG-to-SPI Bridge bitstream.
--freq	N/A	Number	JTAG frequency. Default: 6e6
--start_address	N/A	Hex number	Starting flash address for flash read and write operations, excluding 0x.
--target	N/A	lower, upper, both	Target flash. `both` is used for the x8 mode.
--chip_num	N/A	Number	1-based device position in the JTAG chain. Default: 1
--jcf	N/A	File path	JTAG chain file in XML format. Not needed if there is only one device.
--url	-u	URL	FTDI URL (see Identifying FTDI URLs on page 145).
--verify_method	N/A	none, onchipx1, onchipx2, onchipx4	The method used to verify the downloaded bitstream. Default: onchipx2 (On-chip hash calculation with SPI x2 mode)

Example: Export JTAG Bridge Operations to an SVF File

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

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

Run the export_bitstream_flashloaderv3 Python Script

To create an SVF file:

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

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

To use the created SVF file:

1. Download the JTAG bridge into the FPGA.
2. Execute the created SVF file to program the SPI flash with your project bitstream.

Chapter 13

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.

Start the Efinity Hardware Server

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

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

Table 48: efinity_hw_server.py Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--addr	-a	Address	Server address. If you do not specify an address, the Efinity Hardware Server defaults to 0.0.0.0 (that is, all IPv4 addresses on the local machine).
--cache-dir	N/A	Directory path	Directory to store cache files. If not specified, the server uses your operating system's temporary directory.
--legacy	N/A	None	Run the legacy hardware server for the remote engine. The hardware server in the Efinity software v2026.1 is not backwards compatible. Therefore, to use a Programmer or Debugger from a previous version with the v2026.1 hardware, you must use this option. However, a v2026.1 Programmer or Debugger cannot connect to the hardware server in legacy mode.
--port	-p	Number	Server port number. If you do not specify a port, the Efinity Hardware Server defaults to 8080.
--session-timeout	N/A	Number of seconds	Seconds without a /ping before a session is cleaned up (default: 60)

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 Prompt, enter Ctrl+C to stop the server.

Connect the Board to the Server

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

For your own board, use a JTAG Mini-Module or JTAG cable to connect the board to the server. When you connect to the remote host from your computer, the module or cable name appears in the Programmer's **USB Target** list. (Refer to **JTAG Programming with FTDI Chip Hardware** on page 141.)

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 Remote 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.

Chapter 14

Appendix: Efinity Messages

Contents:

The Efinity software generates messages during compilation. The following topics describe these mesasges and what they mean or how to fix them (if needed).

For Interface Designer messages, refer to the "Design Check" topics in the interfaces user guides:

- Trion Interfaces User Guide
- Titanium Interfaces User Guide
- Topaz Interfaces User Guide

Note: Including all messages in the documentation is a work in progress; Efinix will continue to update these topics and add more messages in upcoming releases.

Synthesis Messages

The software may issue the following messages during synthesis.

EFX-0002 (ERROR)

Message	Illegal command line.
To fix	Check the command line help for usage.

EFX-0003 (ERROR)

Message	Unknown Command Line Option : '<*option*>'.
To fix	Check the command line help for usage.

EFX-0004 (ERROR)

Message	Invalid value for option '<*option*>' in command line.
To fix	Check the command line help for usage.

EFX-0005 (ERROR)

Message	Invalid syntax: '<*cmd*>'.
To fix	Check the command line help for usage.

EFX-0006 (WARNING)

Message	Command line option: '<*option*>' is deprecated and will be ignored.
To fix	Option has no effect or is deprecated, refer to the Efinity Synthesis User Guide for expected behaviour.

EFX-0007 (WARNING)

Message	Option value for '<*option*>' is out of range and will be ignored.
To fix	Check the command line help for usage.

EFX-0008 (WARNING)

Message	Top module not specified. Using module '<*top*>' as top module for the design.
To fix	Specify the top module/entity name in the project settings (Project Editor, Design tab, Top Module/Entity box).

EFX-0010 (VERIFIC_ERROR)

Message	*<name> <file: line number>*
To fix	Please check and correct the RTL according to the error message.

EFX-0011 (VERIFIC_WARNING)

Message	*<name> <file: line number>*
To fix	Please check and correct the RTL according to the error message.

EFX-0013 (WARNING)

Message	Mode '<*mode*>' is not supported, default to 'speed'.
To fix	Check the command line help for usage.

EFX-0015 (ERROR)

Message	Input source file is not specified.
To fix	Check command line or project settings to include design files.

EFX-0016 (WARNING)

Message	Input source file '<*filename*>' does not exist and will be skipped.
To fix	Check command line or project settings for the correct design file name and path.

EFX-0017 (WARNING)

Message	Include dir '<*directory*>' does not exist and will be skipped.
To fix	Check command line or project settings for the correct include path.

EFX-0018 (WARNING)

Message	--f file '<*filename*>' does not exist and will be skipped.
To fix	Check command line or project settings for the correct -f file name and path.

EFX-0019 (ERROR)

Message	Analysis of source file '<*file*>' failed. Please check previous error messages.
To fix	Please check and correct the RTL according to the error message.

EFX-0020 (ERROR)

Message	Elaboration of design failed. Please check previous error messages.
To fix	Please check and correct the RTL according to the error message.

EFX-0021 (ERROR)

Message	Elaboration of module '<*module*>' failed. Please check previous error messages.
To fix	Please check and correct the RTL according to the error message.

EFX-0022 (WARNING)

Message	Top module is not specified. Top level parameters will be ignored.
To fix	Check command line or project settings for correct top module and parameter specifications.

EFX-0023 (WARNING)

Message	Top level parameters list must have an even number of elements. (Ignoring parameters)
To fix	Check command line or project settings to ensure you specified the correct parameter names/values.

EFX-0024 (ERROR)

Message	Cannot instantiate unknown module '<*instance name*>'. <*file: line number*>
To fix	Ensure that the module's source code is included in your design files.

EFX-0025 (ERROR)

Message	Illegal instantiation of empty module <*instance name*>. EFX_PRAGMA__IS_USER_DECLARED_BLACK_BOX pragma should be set for empty modules. <*file: line number*>
To fix	Check whether an empty module is intended; if so, use a pragma on the module.

EFX-0026 (ERROR)

Message	Black box instantiation '<*instance name*>' with inout ports is not supported. <*file: line number*>
To fix	Synthesis does not support a black box with an 'inout' port.

EFX-0027 (WARNING)

Message	Specified map library '<*library*>' already exists in lib containers, please specify new library (skipping mapping).
To fix	Map library with the same name is already loaded. Please use a different map library.

EFX-0029 (WARNING)

Message	Settings_file '<*filename*>' does not exist and will be ignored.
To fix	Check command line or project settings for the correct settings file name and path.

EFX-0030 (ERROR)

Message	Inferred latch '<*instance name*>' is not supported. <*file: line number*>
To fix	Check your RTL to fix the unsupported latch type in the design.

EFX-0032 (WARNING)

Message	primitive '<*instance name*>' has high impedance (1'bz) input, rewiring to GND. <*file: line number*>
To fix	Check whether the unconnected signal or unspecified input to the instance is intended.

EFX-0033 (WARNING)

Message	Project XML file specified. Command line option --f files are ignored.
To fix	The file list in project settings takes precedence over command line --f files.

EFX-0034 (WARNING)

Message	Project XML file specified. Command line --v source files are ignored.
To fix	The file list in project settings takes precedence over command line --v files.

EFX-0035 (INTERNAL_ERROR)

Message	Component '<*instance name*>' (prim_mux) is not fully connected. <*file: line number*>
To fix	Consult Efinix Support for INTERNAL ERROR messages.

EFX-0037 (ERROR)

Message	Cell '<*cell*>' has compiled netlist, but it does not have matching ports. <*errmsg*>
To fix	Please check and correct the RTL according to the error message.

EFX-0039 (INTERNAL_ERROR)

Message	Fail to parse elaborated netlist to find clock domains for MARK_DEBUG signal(s). Debugger profile will not be generated.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0100 (INTERNAL_ERROR)

Message	I/O mapping failed for <*direction*> port : '<*port name*>'
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0101 (INTERNAL_ERROR)

Message	I/O mapping failed (<*description*>).
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0102 (INTERNAL_ERROR)

Message	I/O port pre-synthesis failed.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0103 (INTERNAL_ERROR)

Message	Failed to create netlist for logic synthesis.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0104 (INTERNAL_ERROR)

Message	Logic synthesis netlist checking failed.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0107 (INTERNAL_ERROR)

Message	Failed to decompose wide multiplier '<*instance name*>' (blasting to logic). <*file: line number*>
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0109 (ERROR)

Message	Incompatible architecture setting for multipliers.
To fix	The selected options are incompatible, refer to the Efinity Synthesis User Guide for the expected behaviour.

EFX-0110 (ERROR)

Message	Multiplier '<*instance name*>' illegal instantiation <*family*> fabric does not support <*multiplier size*> multiplier. <*file: line number*>
To fix	Check your RTL and fix any unsupported or incorrect primitives.

EFX-0111 (ERROR)

Message	Incompatible architecture setting for RAM blocks.
To fix	The selection options are incompatible, refer to the Efinity Synthesis User Guide for the expected behaviour.

EFX-0112 (WARNING)

Message	Write mode '<*mode name*>' for memory block '<*instance name*>' is not supported in this architecture. READ_UNKNOWN mode will be used. <*file: line number*>
To fix	The targeted device only supports READ_UNKNOWN mode in memory.

EFX-0113 (ERROR)

Message	Memory block '<*instance name*>' has WRITE_MODE='WRITE_FIRST', but read and write widths do not match. <*file: line number*>
To fix	The targeted device only supports WRITE_FIRST mode if the read and write widths match. Change the write mode or use matching read/write widths.

EFX-0114 (ERROR)

Message	Memory block '<*instance name*>' has WRITE_MODE='WRITE_FIRST', but read and write address ports are not driven by the same signals. <*file: line number*>
To fix	The targeted device only supports WRITE_FIRST mode if the read and write address ports are driven by the same signals. Change the write mode or use matching read/write address.

EFX-0115 (WARNING)

Message	Memory block '<*instance name*>' has async read and write ports. WRITE_MODE will be set to \"READ_UNKNOWN\". <*file: line number*>
To fix	If the read and write clocks are different, the target device's memory only supports READ_UNKNOWN mode.

EFX-0116 (ERROR)

Message	Memory block '<*instance name*>' has illegal combination of read/write widths: r=<*rmode*>,w=<*wmode*>. <*file: line number*>
To fix	Read and write data widths of the memory instance are not available in this device.

EFX-0118 (ERROR)

Message	Memory block '<*instance name*>' has WRITE_MODE set to <*mode*> but has async read and write ports. Async read/write ports only support WRITE_MODE=\"READ_UNKNOWN\". <*file: line number*>
To fix	If the read and write clocks are different, the target device's memory only supports READ_UNKNOWN mode.

EFX-0120 (WARNING)

Message	Memory block '<*instance name*>' has mis-matched read/write address. WRITE_MODE cannot be set to \"WRITE_FIRST\". It will be set to \"READ_UNKNOWN\" instead. <*file: line number*>
To fix	The targeted device only supports WRITE_FIRST mode if the read and write address widths match. Change the write mode or use matching read/write address widths.

EFX-0200 (WARNING)

Message	Removing redundant signal : <*net name*>. <*file: line number*>
To fix	Check your RTL to confirm the unused signal is intended.

EFX-0201 (WARNING)

Message	Re-wiring to GND non-driven net/port '<*net name*>' in module '<*modName*>'. <*file: line number*>
To fix	Check your RTL to confirm the undriven signal is intended.

EFX-0202 (ERROR)

Message	Net '<*net name*>' has multiple drivers. <*file: line number*>
To fix	Check your RTL to fix the multiple driver signal.

EFX-0203 (WARNING)

Message	Instance <*instance name*> has unconnected port <*port name*>. <*instance, file: line number*>
To fix	Check your RTL to confirm the undriven port is intended.

EFX-0204 (ERROR)

Message	Top design has no ports.
To fix	Check your RTL to make sure top level ports are declared.

EFX-0205 (ERROR)

Message	Primitive instance '*<instance name>*' net '*<net name>*' is not driven. *<net, file: line number>*
To fix	Check your RTL to make sure required ports are connected.

EFX-0206 (ERROR)

Message	Instance '*<instance name>*' of type '*<cell name>*' is unsupported. *<instance, file: line number>*
To fix	The target device does not support primitives of this type. Check your RTL to make sure it uses the correct primitive.

EFX-0207 (WARNING)

Message	'*<instance name>*' port '*<port name>*' is wired to wide vector, synthesis will keep only LSB connection. *<instance, file: line number>*
To fix	Check your RTL to make sure the trimming is intended.

EFX-0209 (ERROR)

Message	Primitive '*<instance name>*' is not supported. Please check your command line settings. *<file: line number>*
To fix	Fabric option is not set to support instantiated primitives, refer to the Efinity Synthesis User Guide for expected behaviour.

EFX-0210 (ERROR)

Message	No top design.
To fix	No top design module/entity available. Check your RTL and project settings.

EFX-0211 (WARNING)

Message	Top design has no output port.
To fix	Check your RTL to make sure top level output ports are declared.

EFX-0250 (ERROR)

Message	Primary output net '*<net name>*' is not driven by EFX_IOBUF.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0251 (ERROR)

Message	Primary input port '*<port name>*' must connect to the core only through an IOBUF or a GBUFCE.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0252 (ERROR)

Message	Primary input port '<*port name*>' is driven by IOBUF, but not connected to the 'IO' pin.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0253 (ERROR)

Message	Primary input port '<*port name*>' is connected to an IOBUF, but is not connected to the core.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0254 (ERROR)

Message	The primary output port '<*output*>' is not wired to a net.
To fix	Check RTL to make sure all primary input/output ports are connected.

EFX-0255 (ERROR)

Message	The primary input port '<*input*>' should not be driven by output '<*instancePort*>' of module/instance '<*instanceName*>'.
To fix	Primary input port is driven by module output. Check your RTL to make sure connections are legal.

EFX-0256 (WARNING)

Message	The primary output port '<*port name*>' wire '<*net name*>' is not driven.
To fix	Primary output port is not driven. Check your RTL to make sure connections are legal.

EFX-0257 (INTERNAL_ERROR)

Message	Number of ports do not match in pre/post map netlists (<*pre-map number*> vs <*post-map number*>).
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0258 (INTERNAL_ERROR)

Message	Number of port-buses do not match in pre/post map netlists (<*pre-map number*> vs <*post-map number*>).
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0259 (INTERNAL_ERROR)

Message	Primary ports mismatch in pre/post map netlists.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0260 (INTERNAL_ERROR)

Message	Primary port-buses mismatch in pre/post map netlists.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0261 (INTERNAL_ERROR)

Message	Primary connected ports : (<*number*>) and IOBUFS (<*number*>) count comparison failure: number of connected primary ports must be greater or equal to the number of IOBUFs (note some primary ports are clock ports and are connected to GBUFCE).
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0262 (ERROR)

Message	Primary port nets mismatch in pre/post map netlists.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0263 (INTERNAL_ERROR)

Message	Primary port '<*port name*>' has illegal net name '<*net name*>'. Net name should match port name.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0264 (INTERNAL_ERROR)

Message	Primary port '<*port name*>' net has multiple port connections.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0265 (ERROR)

Message	Bi-directional port '<*net name*>' is not supported in this device. Check synthesis options.
To fix	Bi-directional port is not supported. Check your RTL for correct use of ports.

EFX-0267 (ERROR)

Message	Interface block instance '<*instance name*>' pin '<*pin name*>' is not connected to dedicated top-level port.
To fix	The interface block pin should be connected to top level port. Check your RTL for proper connection.

EFX-0268 (WARNING)

Message	Interface block instance '<*instance name*>' pin '<*pin name*>' is connection to dangling net '<*net name*>' and the connection to core will be removed.
To fix	The interface block pin connection is not used. Check your RTL to see if this is intended.

EFX-0298 (WARNING)

Message	Instantiated primitive '<*instance name*>' has unused pin '<*pin name*>' based on mode. Pin connection will be removed. <*file: line number*>
To fix	Primitive has unused connection. Check your RTL to correct the issue.

EFX-0300 (WARNING)

Message	Instantiated LUT '<*instance name*>' has LUTMASK '<*lutMask*>'. It does not depend on connected input '<*port name*>'. Connection will be removed. <*file: line number*>
To fix	Primitive LUT may have redundant input based on LUTMASK. Check your RTL to see if this is intended.

EFX-0301 (ERROR)

Message	Instantiated LUT '<*instance name*>' has LUTMASK '<*lutMask*>'. It depends on unconnected input '<*port name*>'. <*file: line number*>
To fix	Primitive LUT is missing required input based on LUTMASK. Check and fix your RTL.

EFX-0302 (ERROR)

Message	Instantiated LUT '<*instance name*>' has LUTMASK '(<*lutMask*>)'. It does not fit into a 4-input LUT. <*file: line number*>
To fix	Primitive LUT LUTMASK requires more than 4 inputs. Check and fix your RTL.

EFX-0305 (ERROR)

Message	Instantiated LUT '<*instance name*>' has duplicated inputs at ports : '<*port 1 name*>' and '<*port 2 name*>'.
To fix	LUT should not have duplicated inputs. If the primitive is instantiated, check and fix the your RTL.

EFX-0350 (ERROR)

Message	EFX_ADD '<*instance name*>' CI pin can only be driven by CO pin of another EFX_ADD.
To fix	EFX_ADD primitive CI can only be driven by CO. If the primitive is instantiated, check and fix the your RTL.

EFX-0351 (ERROR)

Message	EFX_ADD '<*instance name*>' CO pin should only drive CI pin.
To fix	EFX_ADD primitive CO can only drive one CI. If the primitive is instantiated, check and fix your RTL.

EFX-0352 (ERROR)

Message	EFX_ADD '<*instance name*>' CO net drives multiple loads.
To fix	EFX_ADD primitive CO can only drive one CI. If the primitive is instantiated, check and fix your RTL.

EFX-0354 (ERROR)

Message	Instantiated EFX ADD '<*instance name*>' port '<*port name*>' (net '<*net name*>') has illegal CI/CO connection.
To fix	EFX_ADD primitive CI/CO has an illegal connection. If the primitive is instantiated, check and fix your RTL.

EFX-0375 (ERROR)

Message	GBUFCE instance '<*instance name*>' has illegal connections.
To fix	EFX_GBUFCE primitive has an illegal connection. If the primitive is instantiated, check and fix your RTL.

EFX-0376 (ERROR)

Message	Clock pin '<*clock pin name*>' of instance '<*instance name*>' is not driven.
To fix	Clock pin of the primitive is not driven. If the primitive is instantiated, check and fix your RTL.

EFX-0377 (ERROR)

Message	Clock port '<*port name*>' of instance '<*instance name*>' is constant. <*instance, file: line number*>
To fix	Clock pin of the primitive is tied to constant. If the primitive is instantiated, check and fix your RTL.

EFX-0378 (WARNING)

Message	Clock-enable port '<*port name*>' of instantiated FF '<*instance name*>' is disabled.
To fix	Clock enable port of the primitive is always disabled. Check your RTL to see if this is intended.

EFX-0379 (WARNING)

Message	Clock-enable port '<*port name*>' of instance '<*instance name*>' is tied to VCC, but polarity is active-low.
To fix	Clock enable port of the primitive is always disabled. Check your RTL to see if this is intended.

EFX-0382 (ERROR)

Message	'I' pin of GBUF/GBUFCE instance '<*instance name*>' should only be connected to primary input port.
To fix	EFX_GBUF primitive has an illegal connection. If the primitive is instantiated, check and fix your RTL.

EFX-0383 (ERROR)

Message	Clock pin '<*port name*>' of instance '<*instance name*>' is not connected. <*instance, file: line number*>
To fix	Clock pin of the primitive is not driven. If the primitive is instantiated, check and fix your RTL.

EFX-0384 (WARNING)

Message	Cannot determine clock domain for MARK_DEBUG signal '<*sigName*>', skipping it. <*net, file: line number*>
To fix	Cannot determine clock domain, choose a different MARK_DEBUG signal in your RTL.

EFX-0385 (ERROR)

Message	Cascaded clock buffers is not supported. Clock buffer '<*driverName*>' is driving another clock buffer '<*loadName*>'. <*net, file: line number*>
To fix	Clock buffer primitives have illegal connection. If the primitives are instantiated, check and fix your RTL.

EFX-0400 (ERROR)

Message	Illegal connection of IOBUF '<*instance name*>' O net to '<*instance name*>' instance.
To fix	The EFX_IOBUF port is not connected correctly. If the primitive is instantiated, check and fix your RTL.

EFX-0410 (ERROR)

Message	Signal driving output port '<*port name*>' is tristated, which is not supported. <*net, file: line number*>
To fix	A tri-state signal driving an output port is not supported. If the primitive is instantiated, check and fix your RTL.

EFX-0449 (ERROR)

Message	Instance '<*instance name*>' non-clock pin '<*pin name*>' is driven by GBUFCE '<*name*>'.
To fix	EFX_GBUFCE can only drive clock pins. If the primitive is instantiated, check and fix your RTL.

EFX-0471 (ERROR)

Message	Memory block '<*name*>' has illegal mode <*mode*> '<*value*>'. <*file: line number*>
To fix	Memory has an illegal parameter. If the primitive is instantiated, check and fix your RTL.

EFX-0473 (ERROR)

Message	<*cell name*> '<*instance name*>' is not supported in <*family*> device. <*instance, file: line number*>
To fix	RAM primitive is not supported in the device. Check and fix your RTL.

EFX-0500 (INTERNAL_ERROR)

Message	library model has unexpected port '<*library name*>'.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0501 (ERROR)

Message	For '<*cell name*>' number of parameters should be : '<*numParams*>'.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0505 (ERROR)

Message	Instance '<*instance name*>' library port '<*port name*>' is missing. <*instance, file: line number*>
To fix	Instance has an unconnected port. If the primitive is instantiated, check and fix your RTL.

EFX-0506 (WARNING)

Message	Set/reset port '<*port name*>' of instance '<*instance name*>' is always enabled.
To fix	Instance set/reset pin is always enabled. Check your RTL if it is intended.

EFX-0508 (ERROR)

Message	Instantiated primitive '<*instance name*>' has illegal parameter '<*name*>'. <*instance, file: line number*>
To fix	Instance has an illegal parameter. If the primitive is instantiated, check and fix your RTL.

EFX-0509 (INTERNAL_ERROR)

Message	'<*instance name*>' instance parameter '<*name*>' value <*value*> differs from attribute value <*value*>
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0510 (ERROR)

Message	'<*instance name*>' instance has missing parameter '<*name*>' when compared with attribute.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0511 (ERROR)

Message	Path '<*path name*>' does not exist.
To fix	Path for file operation is invalid. Check command line and project settings.

EFX-0512 (ERROR)

Message	Pre-synthesis failure is encountered in flow.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0513 (ERROR)

Message	Fail to create path '<*path name*>'.
To fix	TODO

EFX-0650 (ERROR)

Message	Input port '<*port name*>' of '<*cell name*>' instance '<*instance name*>' should not be connected to constant.
To fix	Pin of the primitive should not be tied to constant. If the primitive is instantiated, check and fix your RTL.

EFX-0651 (WARNING)

Message	Instance '<*instance name*>' port '<*port name*>' is always disabled. <*instance, file: line number*>
To fix	Pin of the primitive is disabled. Check your RTL to see if this is intended.

EFX-0652 (ERROR)

Message	'<*cell name*>' instance '<*instance name*>' port '<*port name*>' is not disabled but related register is not used. <*instance, file: line number*>
To fix	Primitive pin should be disabled based on configuration. If the primitive is instantiated, check and fix your RTL.

EFX-0654 (WARNING)

Message	*** COMBINATIONAL LOOP detected! Signals contributing to the loop are: <*net names*> <*net, file: line number*>
To fix	Efinix does not recommend using combinational loops. Check your RTL to see if this is intended.

EFX-0655 (WARNING)

Message	Mapping into logic memory block '<*RAM net name*>' (<*size*> bits). <*file: line number*>
To fix	Memory will be implemented in logic. Check your RTL to see if this is intended.

EFX-0656 (WARNING)

Message	Optimizing into logic zero initialized read-only memory block '<*RAM net name*>'. <*file: line number*>
To fix	If ROM is initialized with zeros, it will be optimized. Check your RTL to see if this is intended. Note that RAMs with a disabled write-enable or write-clock will also be treated as ROMs.

EFX-0657 (WARNING)

Message	Mapping into logic memory block '<*RAM net name*>' (<*size*> bits) <*file: line number*> because <*reason*>
To fix	Memory will be implemented in logic. Check your RTL to see if this is intended.

EFX-0658 (ERROR)

Message	Memory '<*instance name*>' has disconnected port '<*port name*>'. <*instance, file: line number*>
To fix	TODO

EFX-0659 (ERROR)

Message	ROM port '<*port name*>' is unconnected.
To fix	Memory instance has an unconnected port. If the primitive is instantiated, check and fix your RTL.

EFX-0662 (INTERNAL_ERROR)

Message	Fail to bit blast RAM net '<*RAM net name*>'. <*file: line number*>
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0666 (WARNING)

Message	Mapping into logic multiplier '<*instance name*>'. <*instance, file: line number*>
To fix	Multiply operator will be implemented in logic. Check your RTL to see if this is intended.

EFX-0672 (ERROR)

Message	Memory '<*net name*>' size <*net size*> exceeds maximum bit-blast threshold <*threshold*>. Please check option --max-bit-blast-mem-size. <*net, file: line number*>
To fix	Memory is too large for bit-blasting. Check your RTL if intended or increase the memory blasting threshold.

EFX-0674 (INTERNAL_ERROR)

Message	Logic synthesis failed.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0675 (INTERNAL_ERROR)

Message	Post synthesis netlist creation failed.
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0676 (WARNING)

Message	Include path is not specified for RAM net '<*net name*>' hex init file <*file*> path is not resolving. <*net, file: line number*>
To fix	You should specify the memory initialization file search path in the include directory setting. Check your project settings.

EFX-0678 (WARNING)

Message	Failed to read memory block '<*net name*>' hex init file '<*file*>'. <*net, file: line number*>
To fix	Fail to read memory initialization file. Check the file content or location for correctness.

EFX-0679 (INTERNAL_ERROR)

Message	Memory '<*RAM net name*>' pre-synthesis failure. <*file: line number*>
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0680 (ERROR)

Message	Dual-port memory '<*RAM net name*>' has incompatible control signal, RE ('<*read enable name*>') should also control WE ('<*write enable name*>'). <*file: line number*>
To fix	The dual-port memory signal RE should also control WE. See the Quantum Primitives User Guide for details.

EFX-0681 (INTERNAL_ERROR)

Message	Memory '<*RAM net name*>' is invalid for inference but cannot be bit-blasted. <*net, file: line number*>
To fix	Consult Efinix Support on INTERNAL ERROR messages.

EFX-0683 (WARNING)

Message	RAM net '<*RAM net name*>' has incompatible control or address signals. <*net, file: line number*>
To fix	The memory has incompatible control or address signals. Check your RTL to see if this is intended.

EFX-0684 (WARNING)

Message	RAM net '<*RAM net name*>' cannot be implemented in block RAM or block ROM. <*file: line number*>
To fix	The memory cannot be implemented in BRAM blocks and will be blasted to logic. Check your RTL and the Efinity Synthesis User Guide for memory inference examples.

EFX-0687 (WARNING)

Message	'syn_preserve' or 'syn_keep' attribute on net '<*net1*>' is transferred to primary I/O net '<*net2*>'.
To fix	A 'syn_preserve' attribute on the net is transferred to a primary I/O port.

EFX-0689 (WARNING)

Message	SRL8 chain started with '<*startBit*>' is too long, but cannot be broken up.
To fix	The software cannot break up a long shift register chain, which may cause place and route issues.

EFX-0691 (WARNING)

Message	Carry skip insertion threshold (<*cskip_t*>) is too short, expect minimum <*min_cskip_t*>.
To fix	The carry skip insertion threshold setting is invalid. Check the Efinity Synthesis User Guide for detail.

EFX-0702 (INFO)

Message	Formal equivalence check of spec(pre-map) '<*specNetlistName*>' and impl(post-map) '<*implNetlistName*>' netlists PASS status.
To fix	TODO

EFX-0809 (ERROR)

Message	The periphery cell <*cell name*> is not supported on the current device.
To fix	The periphery primitive is not supported in this device. Check your RTL and fix it.

Post-Synthesis Check Messages

The software may issue the following messages during post-synthesis checking.

EFX-9001 (ERROR)

Message	Output Port <*name*> is unconnected and not driven.
To fix	Check the top level port connections.

EFX-9002 (ERROR)

Message	Port <*name*> is directly connected to <*number*> other top-level ports.
To fix	Check the top level port connections.

EFX-9003 (ERROR)

Message	Output port <*name*> is connected to <*name*>/<*name*>, but only Input or Inout ports can be connected to an EFX_GBUFCE I port.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9004 (ERROR)

Message	Port <*name*> is connected to <*name*>/<*name*>, but must be connected to an EFX_IOBUF IO port or an EFX_GBUFCE I port.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9005 (ERROR)

Message	Port <*name*> is connected to <*number*> EFX_GBUFCE instances.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9006 (ERROR)

Message	Port <*name*> is connected to <*number*> EFX_IOBUF instances.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9007 (ERROR)

Message	Unexpected constant net <*name*>.
To fix	Consult Efinix Support on this ERROR message.

EFX-9008 (ERROR)

Message	Post-synthesis netlist is not legal because it has 0 total nets.
To fix	Consult Efinix Support on this ERROR message.

EFX-9009 (ERROR)

Message	Post-synthesis netlist has only vcc and gnd nets because all logic is eliminated. Legal, but will cause router to crash.
To fix	The design logic may have been optimized away by synthesis. Check your design for unintended connections or unused outputs.

EFX-9010 (ERROR)

Message	Post-synthesis netlist is not legal because it has <*number*> GND nets.
To fix	Consult Efinix Support on this ERROR message.

EFX-9011 (ERROR)

Message	Post-synthesis netlist is not legal because it has <*number*> VCC nets.
To fix	Consult Efinix Support on this ERROR message.

EFX-9012 (ERROR)

Message	Net <*name*> is not driven.
To fix	Consult Efinix Support on this ERROR message.

EFX-9013 (ERROR)

Message	Net <*name*> has multiple drivers.
To fix	Consult Efinix Support on this ERROR message.

EFX-9014 (ERROR)

Message	Net <*name*> is driving multiple clock buffers. <*name*>
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9015 (ERROR)

Message	Clock net <*name*> drives clock ports and clock buffer <*name*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9017 (ERROR)

Message	<*name*> is missing port <*name*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9018 (ERROR)

Message	<*name*> is missing portbus <*name*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9019 (ERROR)

Message	<*name*> portbus of <*name*> instance <*name*> is size <*number*> but <*number*> expected.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9020 (ERROR)

Message	<*name*> port of <*name*> instance <*name*> is disconnected.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9021 (ERROR)

Message	<*name*> port of <*name*> instance <*name*> is constant.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9022 (ERROR)

Message	<*name*> port of <*name*> instance <*name*> is not driven by an EFX_GBUFCE/O port.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9024 (ERROR)

Message	<*name*> instance <*name*> has different read/write address bit <*number*> in WRITE_FIRST WRITE_MODE.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9025 (ERROR)

Message	<*name*> instance <*name*> <*name*> port is disabled, but <*name*> is not.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9026 (ERROR)

Message	<*name*> instance <*name*> <*name*> port is not disabled in a disabled mode.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9027 (ERROR)

Message	<*name*> instance <*name*> input <*name*> is active, but unused. Check parameter settings.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9028 (ERROR)

Message	<*name*> instance <*name*> has different read/write address enable in WRITE_FIRST WRITE_MODE.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9029 (ERROR)

Message	<*name*> instance <*name*> with WRITE_MODE = <*name*> has different read/write clock nets/ polarity.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9030 (ERROR)

Message	<*name*> parameter of <*name*> instance <*name*> is not valid.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9031 (ERROR)

Message	<*name*> instance <*name*> <*name*> read width <*number*> is not compatible with write width <*number*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9032 (ERROR)

Message	<*name*> instance <*name*> Port A read/write widths <*number*>/<*number*> are not compatible with Port B <*number*>/<*number*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9033 (ERROR)

Message	<*name*> instance <*name*> both CLKA port and CLKB ports are disabled.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9034 (ERROR)

Message	<*name*> instance <*name*> WE[1] port is not disabled, but WRITE_WIDTH is <*number*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9035 (ERROR)

Message	<*name*> instance <*name*> WRITE_MODE is WRITE_FIRST but READ_WIDTH is <*number*> and WRITE_WIDTH is <*number*>.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9036 (ERROR)

Message	<*name*> instance <*name*> WRITE_MODE is WRITE_FIRST but WCLKE is not permanently enabled.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9037 (ERROR)

Message	<*name*> instance <*name*> has parameter SR_SYNC_PRIORITY=<*number*>, but the device only supports a value of <*number*>.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9038 (ERROR)

Message	<*name*> instance <*name*> CLK port is not disabled but no registers are enabled.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9039 (ERROR)

Message	<*name*> instance <*name*> SHIFT_ENA port is not disabled, but SHIFTER parameter is 0.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9040 (ERROR)

Message	<*name*> instance <*name*> <*name*> must be enabled when <*name*>.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9041 (ERROR)

Message	<*name*> instance <*name*> <*name*> must be <*name*> when <*name*>.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9042 (ERROR)

Message	<*name*> instance <*name*> MODE '<*name*>' is illegal.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9043 (ERROR)

Message	<*name*> instance <*name*> W_REG may not be enabled when W_SEL is P.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9044 (ERROR)

Message	<*name*> instance <*name*> CASCIN[<*number*>] port must be connected to GND or CASCOUT.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9045 (ERROR)

Message	<*name*> instance <*name*> CASCOUT[<*number*>] port must be disconnected or connected to CASCIN.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9046 (ERROR)

Message	<*name*> instance <*name*> CASCIN port must be connected to GND or CASCOUT.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9047 (ERROR)

Message	<*name*> instance <*name*> CASCOUT port must be disconnected or connected to CASCIN.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9048 (ERROR)

Message	<*name*> instance <*name*> W_SEL cannot be P if CASCOUT_SEL is W.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9049 (ERROR)

Message	<*name*> instance <*name*> CI port must be connected to GND or CO.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9050 (ERROR)

Message	<*name*> instance <*name*> CO port must be disconnected or connected to CI.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9051 (ERROR)

Message	<*name*> instance <*name*> <*name*> port must not be connected in <*name*> mode.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9052 (ERROR)

Message	<*name*> instance <*name*> unexpected MODE '<*name*>'.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9053 (ERROR)

Message	<*name*> instance <*name*> CO port must be connected to only 1 CIN.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9054 (ERROR)

Message	<*name*> instance <*name*> is driven by net <*name*> more than once.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9055 (ERROR)

Message	<*name*> instance <*name*> LUTMASK %04X is too large to fit into a 4-input LUT.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9056 (ERROR)

Message	<*name*> instance <*name*> LUTMASK %04X does not depend on connected input <*number*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9057 (ERROR)

Message	<*name*> instance <*name*> LUTMASK %04X does depends on unconnected input <*number*>.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9058 (ERROR)

Message	<*name*> instance <*name*> Q7 port must be disconnected or connected to only an SRL8 D port.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9059 (ERROR)

Message	I port of <*name*> instance <*name*> is constant.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-9060 (ERROR)

Message	Clock net <*name*> is driving the <*name*> port of block <*name*>, it must only drive clock ports.
To fix	The GBUFCE output cannot drive a primitive's non-clock port. However, the input can drive it. Use a different signal or drive the non-clock port with the same value as the input to the GBUFCE (which uses regular routing).

EFX-9062 (ERROR)

Message	<*name*> instance <*name*> input OP is active, but unused (W_SEL is not set to X). Check parameter settings.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-9064 (ERROR)

Message	Clock net <*name*> is driving control signal, but the 'CE' of the clock buffer is not constant '1'.
To fix	Clock buffer used to drive control signal must have its 'CE' pin tied high to avoid glitches.

EFX-8001 (WARNING)

Message	<*name*> port of <*name*> instance <*name*> is permanently enabled
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-8002 (WARNING)

Message	<*name*> port of <*name*> instance <*name*> is permanently disabled
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-8003 (WARNING)

Message	Input/inout port <*name*> is unconnected. Place and route may remove this unconnected port.
To fix	Check if port is intentionally unconnected.

EFX-8004 (WARNING)

Message	<*name*> port of <*name*> instance <*name*> is connected in a disabled mode.
To fix	If the instance is an instantiated primitive, please check the port connections.

EFX-8005 (WARNING)

Message	<*name*> instance <*name*> is using deprecated WIDTH parameter <*number*>.
To fix	If the instance is an instantiated primitive, please check the correct parameter value and range.

EFX-8009 (WARNING)

Message	Net <*name*> is dangling.
To fix	The net may be dangling due to 'syn-keep' or 'syn-preserve' property.

Device Database Messages

The software may issue the following messages when reading the device database.

ArchDBMissingMLatchModelSignal (Error)

Message	Cannot found signal {} in the mlatch model in architecture file
Description	Cannot found a signal in the mlatch model in architecture file.

ArchModelCheckWarningCount (Warning)

Message	Found {} warnings when checking timing parameters
Description	Timing parameter checks have completed. The number of warnings identified has been displayed.

ArchModelClockPortNoAssociatedPort (Warning)

Message	Model {}: Clock port {} does not have any associated port (Expected: At least 1) in mode {}
Description	A clock port in an Arch model does not have any associated port.

ArchModelInputPortUnexpectedAssociatedClockCount (Warning)

Message	Model {}: Input port {} has {} clock associated (Expected: 1) in mode {}
Description	An input port in an Arch model has an incorrect number of associated clocks.

ArchModelMissingAnnotatedClockPortInput (Warning)

Message	Annotated clock port {} cannot be found in input of model {}
Description	An annotated clock port is missing in an Arch model input.

ArchModelMissingAnnotatedClockPortInputLine (Warning)

Message	Annotated clock port {} cannot be found in input of model {} (line: {})
Description	An annotated clock port is missing in an Arch model input.

ArchModelMissingAnnotatedPortInputLine (Warning)

Message	Annotated port {} cannot be found in input of model {} (line: {})
Description	An annotated port is missing in an Arch model input.

ArchModelMissingAnnotatedPortOutput (Warning)

Message	Annotated port {} cannot be found in output of model {}
Description	An annotated port is missing in an Arch model output.

ArchModelMissingAnnotations (Warning)

Message	Model {} with pb_type blif_model {} doesn't have annotations
Description	An Arch model doesn't have annotations.

ArchModelOutputPortUnexpectedAssociatedClockCount (Warning)

Message	Model {}: Output port {} has {} clock associated (Expected: 1) in mode {}
Description	An output port in an Arch model has an incorrect number of associated clocks.

CellDelayOverrideLoading (Info)

Message	Loading CellDelayOverride information from '{}'
Description	Successfully read in the CellDelayOverride file.

CellSiteInstCellNotFound (Error)

Message	Cell instance '{}' in CellSiteInstance file not found in netlist
Description	A cell in the CellSiteInst file was not found in the netlist.

CellSiteInstInvalidSiteInst (Error)

Message	SiteInst '{}' for cell instance '{}' in the CellSiteInstance file is not valid
Description	A cell in the CellSiteInst file was assigned to an invalid SiteInst.

CellSiteInstLoading (Info)

Message	Loading CellSiteInst information from '{}'
Description	Successfully read in the CellSiteInst file.

CellSiteInstMultipleCell (Error)

Message	Cell instance '{}' has multiple entries in the CellSiteInstance file
Description	A cell in the CellSiteInst file was found more than once.

ConnectionProfileJsonDoesNotExist (Error)

Message	File {} does not exist
Description	Connection profile JSON file does not exist

ConnectionProfileParseFailed (Error)

Message	Failed to parse connection profile {}
Description	Failed to parse connection profile

ConnectionProfileParseFailedFromString (Error)

Message	Failed to parse connection profile
Description	Failed to parse connection profile

CoreInterfaceConstraintsFileChanged (Error)

Message	The contents of the current interface constraints file does not match the configuration used during earlier compilation stages. Please rerun the compilation flow from the beginning
Description	The contents of the interface constraints file does not match the configuration used during earlier compilation stages. Rerun the compilation flow from the beginning

CoreInterfaceConstraintsFileClockSourceRegistrationFailed (Error)

Message	Failed to register clock soure "{}"
Description	The Efinity software has failed to register a clock source.

CoreInterfaceConstraintsFileDuplicatePinName (Error)

Message	Duplicate pin name found in core interface constraints file: {}
Description	The Efinity software has identified duplicate pin names in a core interface constraints file.

CoreInterfaceConstraintsFileIllegalConstraint (Error)

Message	Illegal {} constraint found in core interface constraints file: {}
Description	The Efinity software has identified an illegal constraint in a core interface constraints file.

CoreInterfaceConstraintsFileIllegalIOLocation (Error)

Message	Illegal IO location found in core interface constraints file: {}
Description	The Efinity software has identified an illegal IO location in a core interface constraints file.

CoreInterfaceConstraintsFileIllegalPinIdx (Error)

Message	Illegal pin index found in core interface constraints file: {}
Description	The Efinity software has identified an illegal pin index in a core interface constraints file.

CoreInterfaceConstraintsFileIllegalType (Error)

Message	Illegal type found in core interface constraints file: {}
Description	The Efinity software has identified an illegal type in a core interface constraints file.

CoreInterfaceConstraintsFileInconsistentClockConstraints (Error)

Message	Inconsistent clock constraints found in core interface constraints file: {}
Description	The Efinity software has identified inconsistent clock constraints in a core interface constraints file.

CoreInterfaceConstraintsFileLoadFailed (Error)

Message	Failed to read core interface constraints file "{}". {} issues identified
Description	The Efinity software has failed to read a core interface constraints file.

CoreInterfaceConstraintsFileLoadSuccess (Info)

Message	Successfully read core interface constraints file "{}"
Description	The Efinity software has successfully read and processed a core interface constraints file.

CoreInterfaceConstraintsFileMissingClockDefinition (Info)

Message	No defintion found for clock "{}". Assuming clock is internally generated
Description	The Efinity software has not found a definition for the referenced clock.

CoreInterfaceConstraintsFileMultiplePinConstraint (Error)

Message	Pin is constrained more than once in core interface constraints file: {}
Description	The Efinity software has identified multiple constraints on the same pin in a core interface constraints file.

CoreInterfaceConstraintsFileNotFound (Error)

Message	Core interface constraints file "{}" cannot be located
Description	The Efinity software is unable to locate a core interface constraints file.

CoreInterfaceConstraintsFilePinTypeOffsetMismatch (Error)

Message	Pin type does not match offset in core interface constraints file: {}
Description	The Efinity software has identified an pin type/offset mismatch in a core interface constraints file.

CoreInterfaceConstraintsFileReading (Info)

Message	Reading core interface constraints file "{}"
Description	The Efinity software is reading a core interface constraints file.

CoreInterfaceConstraintsFileUnknownConstraint (Error)

Message	Unknown constraint found in core interface constraints file: {}
Description	The Efinity software does not recognize a constraint in a core interface constraints file.

CoreInterfaceConstraintsMissingClockNet (Error)

Message	No clock net found for clock {}
Description	No clock net found for given clock

DatabasDeviceNoAutoGrid (Error)

Message	Database::LoadDevice() called when called when Auto sized grid
Description	Internal flow control error. PnR database does not support an automatic grid definition.

DatabaseAlreadySetup (Error)

Message	Database::SetUp() called when database already setup
Description	Internal flow control error. PnR database attempted to be setup twice.

DatabaseBadClusterData (Error)

Message	Cluster data mismatch detected during load placement
Description	Logical blocks found without cluster information after loading placement data.

DatabaseClockViolations (Error)

Message	{} clock violations are detected during load placement
Description	Clock regions found with too many clocks after loading placement data.

DatabaseClusteredNetlistAlreadyLoaded (Error)

Message	Cannot load clustered netlist: netlist is already loaded
Description	Internal flow control error. PnR database attempted to load the clustered netlist twice.

DatabaseClusteredNetlistDoesntExist (Error)

Message	Cannot load clustered netlist: netlist does not exist
Description	Clustered netlist file not found.

DatabaseClusteredNetlistLoaded (Error)

Message	Cannot erase clustered netlist: netlist must first be unloaded
Description	Internal flow control error. PnR database attempted to erase the clustered netlist while it was loaded.

DatabaseClusteredNetlistNotLoaded (Error)

Message	Cannot unload clustered netlist: netlist is not loaded
Description	Internal flow control error. PnR database attempted to unload the clustered netlist when it was not loaded.

DatabaseDeviceAlreadyLoaded (Error)

Message	Database::LoadDevice() called when device already loaded
Description	Internal flow control error. PnR database attempted to be loaded the device twice.

DatabaseDeviceNotLoaded (Error)

Message	{} called when device not loaded
Description	Internal flow control error. PnR database function called without the device being loaded.

DatabaseGlobalNetworkPlacementViolations (Error)

Message	Error(s) exist in global network placement
Description	Violations found in global network data after loading placement data.

DatabaseNetlistAlreadyLoaded (Error)

Message	Database::LoadLogicalNetlist() called when logical netlist already loaded
Description	Internal flow control error. PnR database attempted to load the logical netlist twice.

DatabaseNetlistDoesntExist (Error)

Message	The specified logical netlist file '{}' does not exist
Description	Logical netlist file not found.

DatabaseNetlistLoaded (Error)

Message	{} called when logical netlist loaded
Description	Internal flow control error. PnR database function called while the logical netlist is loaded

DatabaseNetlistNotLoaded (Error)

Message	{} called when logical netlist not loaded
Description	Internal flow control error. PnR database function called without the logical netlist being loaded.

DatabaseNotSetup (Error)

Message	{} called when database not setup
Description	Internal flow control error. PnR database function called without the database setup.

DatabasePlacementAlreadyLoaded (Error)

Message	Database::LoadPlacement() called when placement already loaded
Description	Internal flow control error. PnR database attempted to load the placement data twice.

DatabasePlacementChecks (Info)

Message	Run checks after load placement
Description	Performing checks to confirm placement data is consistent with the device.

DatabasePlacementDoesntExist (Error)

Message	Database::LoadPlacement() called when place file does not exist
Description	Placement file not found.

DatabasePlacementLoaded (Error)

Message	Database::ErasePlacement() called when placement loaded
Description	Internal flow control error. PnR database attempted to erase the placement data while it was still loaded.

DatabasePlacementNotLoaded (Error)

Message	Database::UnloadPlacement() called when placement not loaded
Description	Internal flow control error. PnR database attempted to unload the placement data when it was not loaded.

DatabasePlacementViolations (Error)

Message	Found {} error(s) in placement
Description	Violations found after loading placement data.

DatabaseReadRouterLookaheadFailed (Error)

Message	Failed to read router lookahead for device {} from file {}
Description	Router lookahead file could not be read.

DatabaseReadRouterLookaheadPassed (Info)

Message	Read router lookahead for device {} from file {}
Description	Router lookahead file successfully read.

DatabaseRoutingDoesNotExist (Error)

Message	{} called when route file does not exist
Description	PnR routing data file not found.

DatabaseRoutingEraseTrouteTrace (Info)

Main Message	Deleting any pre-existing troute trace files
Action	Removing routing trace data from a previous compile.
Sub-Message	Removing troute trace file '{}'
Description	Troute trace file that was removed.

DatabaseRoutingGraphAlreadyLoaded (Error)

Message	{} called when routing graph already loaded
Description	Internal flow control error. PnR database attempted to load the routing graph twice.

DatabaseRoutingGraphNotLoaded (Error)

Message	{} called when routing graph not loaded
Description	Internal flow control error. PnR database attempted to call function while the routing graph was not loaded.

DatabaseRoutingGraphStats (Info)

Message	Generated {} RR nodes and {} RR edges
Description	Information about the size of the routing graph.

DatabaseRoutingNotLoaded (Error)

Message	{} called when routing not loaded
Description	Internal flow control error. PnR database attempted to call function while the routing data was not loaded.

DatabaseRoutingUpdateTraceSwitchDataFailed (Error)

Message	Failure in UpdateTraceSwitchData, called by {}
Description	Internal flow control error. Errors detected while updating switch data based on the routing file.

DatabaseSynchronizers (Info)

Main Message	Found {} synchronizers as follows:
Action	Listing the clock-domain-crossing synchronizer chains found in the netlist.
Sub-Message	{}:
Description	Synchronizer register name.
Sub-Message	Synchronizer {}:
Description	Synchronizer chain ID.

DesignContainsClkDrivenSinks (Error)

Main Message	The design contains clock input ports that are driven by a constant
Action	The design contains clock input ports that are driven by constant(s). This is not a supported configuraton.
Sub-Message	Clock input port "{}" is driven by constant net "{}"
Description	Information about the violation.

DeviceMissingIOPadInfo (Error)

Message	Missing IO pad info at ({} {}, {})
Description	Missing IO pad info

DeviceMissingIOPadType (Error)

Message	Missing IO pad type at ({} {}, {})
Description	Missing IO pad type

DeviceUnusedIOPads (Info)

Message	This device has {} unused io pads using {} package
Description	This device has unused io pads

EfxParseDumpParamsFail (Error)

Message	Unable to dump parameters to {}
Description	The dump parameters operation failed.

EfxParseDumpParamsPass (Info)

Message	Dumped parameters to {}
Description	The dump parameters operation succeded.

FinalAnalysisFanoutRecommendationFileWriteFailed (Error)

Message	Write to {} failed
Description	The write operation failed.

GridDimensionsOutOfRange (Error)

Message	nx and ny must be less than 32767, since the router uses shorts (16-bit) to store coordinates
Description	nx and ny must be less than 32767

IOPlacementConstraintsFileWriteFailed (Error)

Message	Failed to write IO placement constraints file to "{}"
Description	The Efinity software is unable to write an IO placement constraints file.

IOPlacementConstraintsFileWriting (Info)

Message	Writing IO placement constraints to "{}"
Description	The Efinity software is writing an IO placement constraints file.

IncompatableCellSiteInst (Error)

Message	Cell instance '{}' of type '{}' cannot be placed on SiteInst '{}' as indicated in the CellSiteInst file
Description	The cell type is not compatible with the Site selected for it.

IncompatiblePartitionInterface (Error)

Main Message	Found interface incompatiblities between top design and imported partition "{}"
Action	Found interface incompatiblities between top design and imported partition.
Sub-Message	Input interface "{}" is missing from the top design
Description	Missing interface.
Sub-Message	Input interface "{}" is missing from the partition
Description	Missing interface.
Sub-Message	Output interface "{}" is missing from the top design
Description	Missing interface.
Sub-Message	Output interface "{}" is missing from the partition
Description	Missing interface.

LbfWriteSuccessful (Info)

Message	Successfully written updated LBF file "{}"
Description	Successfully written updated LBF file

LookaheadIncorrectFileExtension (Warning)

Message	Router lookahead filename {} does not have .zst file extension; format is incorrect!
Description	Router lookahead does not have .zst file extension

LookaheadReadCorrupted (Warning)

Message	Pre-computed router lookahead file '{}' appears to be corrupted
Description	Pre-computed router lookahead file appears to be corrupted

LookaheadReadFailed (Warning)

Message	Could not read pre-computed router lookahead file '{}'
Description	Could not read pre-computed router lookahead file

LookaheadReadOutOfDate (Error)

Message	Pre-computed router lookahead file '{}' has out of date version
Description	Pre-computed router lookahead file has out of date version

LookaheadWriteFailed (Warning)

Message	Could not write lookahead data to file '{}'; continuing with in-memory data
Description	Could not write lookahead data to file

LookaheadWriteReadbackSuccessful (Info)

Message	Successfully performed lookahead data file readback checks
Description	Successfully performed lookahead data file readback checks

MifFailedToRestore (Error)

Message	Could not restore "{}" back to the initial settings
Description	Failed to restore logical memory to the initial settings

MifFailedToUpdate (Error)

Message	Could not update "{}" from init file "{}" : {}
Description	Failed to update logical memory from memory information file

MifMissingMemoryData (Error)

Message	Could not find memory "{}" in saved memory information
Description	Could not find memory information in saved memory information file

MifUpdateSuccessful (Info)

Message	Updated initial memory contents for "{}"
Description	Successfully updated initial memory contents

NetistLogicalBlockClockCounts (Info)

Message	block {} i ({}) count_clocks={} logical_block_clock_count[i]={}
Description	The number of clocks for a given block has been displayed

NetistLogicalBlockInputCounts (Info)

Message	block {} i ({}) count_inputs={} logical_block_input_count[i]={}
Description	The number of inputs for a given block has been displayed

NetistLogicalBlockOutputUnconnectedClockPin (Error)

Message	Logical Block #{} with output {} has a clock pin unconnected ({})
Description	Logical block has unconnected clock pin

NetlistActivityFileBlifFileMismatch (Error)

Message	Net {} found in activity file, but it does not exist in the .blif file
Description	Net found in activity file, but it does not exist in the .blif file

NetlistActivityFileMissingSignal (Error)

Message	Activity file does not contain signal {}
Description	Activity file does not contain signa

NetlistAutoIobufGeneration (Info)

Message	auto_iobuf_generation
Description	auto_iobuf_generation

NetlistBlockNoOutput (Warning)

Message	Block contains no output
Description	Block contains no output

NetlistBlocksSweptThisPass (Info)

Message	Pass {}: Swept away {} blocks with no fanout
Description	The number of dangling blocks removed in this optimization pass.

NetlistClockNetConnectingNetMismatch (Error)

Message	Clock net for block {} #{} is net {} #{} but connecting net is {} #{}
Description	Mismatch between clock net and connecting net to a block

NetlistConstGenerator (Info)

Message	Net is a constant generator: {}
Description	Net is a constant generator

NetlistConstGeneratorCount (Info)

Message	Found {} constant generator nets
Description	The number of constant genertor nets (VCC/GND) found in the netlist

NetlistConstGeneratorLUT0 (Warning)

Message	Net '{}' is driven by a 0LUT (const generator) '{}' that is neither VCC nor GND
Description	LUT primitive with 0 inputs was found in the netlist.

NetlistCreateGND (Info)

Message	Creating GND const gen for DSP tie-offs
Description	Creating a GND cell to use for tying off unused DSP inputs.

NetlistCreateVCC (Info)

Message	Creating VCC block
Description	Creating a VCC cell to use for tying off unused cell inputs.

NetlistFileBlocksAndNetsInfo (Info)

Message	Total blocks: {}, total nets: {}
Description	Information about netlist clocks and nets has been displayed.

NetlistFileInOutInfo (Info)

Message	Primary inputs: {}, primary outputs: {}
Description	Information about netlist input and outputs has been displayed.

NetlistFileInfo (Info)

Message	Input netlist file: '{}'
Description	Information about a netlist file has been displayed.

NetlistFileLutLatchSubcktsInfo (Info)

Message	LUTs: {}, latches: {}, subckts: {}
Description	Information about netlist LUTs, latches and subckts has been displayed.

NetlistFileStdAbsorbableLatchInfo (Info)

Message	# standard absorbable latches: {}
Description	Information about netlist standard absorbable latches has been displayed.

NetlistFileUnexpectedEndLines (Error)

Message	Found {} .end lines; expected {}
Description	Unexpected netlist file format

NetlistFileUnexpectedInputLines (Error)

Message	Found {} .inputs lines; expected {}
Description	Unexpected netlist file format

NetlistFileUnexpectedModelLines (Error)

Message	Found {} .model lines; expected {}
Description	Unexpected netlist file format

NetlistFileUnexpectedOutputLines (Error)

Message	Found {} .outputs lines; expected {}
Description	Unexpected netlist file format

NetlistFileVPackNetBadControlNetConnection (Error)

Message	Netlist inconsistency: Block #{}({}) control port {} is connected to net #{}({}), but is not a control net
Description	The block control port is connected to net, but the net is not a control net.

NetlistFileVPackNetBadControlPortConnection (Error)

Message	Netlist inconsistency: Block #{}({}) port {} is connected to global control net #{}({}), but is not a control port
Description	The block port connected to a global control net is not a control port.

NetlistFileVPackNetBrokenConnection (Error)

Message	Netlist inconsistency: Block #{}({}) is connected to net #{}({}), but connecting net is #{}({})
Description	A net in the netlist is connected to a block pin, but the block pin connects to a different net.

NetlistFileVPackNetBrokenConnection2 (Error)

Message	Netlist inconsistency: Block #{}({}) is connected to net #{}({}), but the net does not have a connection to the block
Description	A block pin connects to a net, but that net does not connect back to the block pin.

NetlistFileVPackNetDanglingInvertedInput (Error)

Message	Netlist inconsistency: Block #{}({}) input {}[{}] is unconnected, but inverted
Description	A block port is unconnected and marked as inverted.

NetlistFileVPackNetInvalidNumDrivers (Error)

Message	vpack_net {} has {} signals driving it
Description	VPack net has invalid number of drivers

NetlistFileVPackNetMissingConnection (Error)

Message	Netlist inconsistency: vpack_net #{}({}) is missing connection {}
Description	A net in the netlist has allocated a connection, but no block is connected to it.

NetlistFileVPackNetNoFanout (Warning)

Message	vpack_net {} has no fanout
Description	VPack net has no fanout

NetlistFileVPackNetReservedName (Error)

Message	vpack_net #{} has the reserved name {}
Description	VPack net has reserved name

NetlistFoundClockNetNotDirectlyDrivenByInputPad (Info)

Message	Found clock net not driven directly by input pad ({})
Description	Found clock net not driven directly by input pad

NetlistFoundInternalGeneratedClock (Info)

Message	Found internal generated clock at net {}: {} driven by {}: {}
Description	Found internal generated clock

NetlistGlobalControlOutputNetMismatch (Error)

Message	Netlist inconsistency: Block #{}({}) global control output {} drives non-global control net #{}({})
Description	A block with an output that must drive a global control line is connected to a non-global control net.

NetlistGlobalNetMismatch (Error)

Message	Netlist inconsistency: Block #{}({})non-global output {} drives global net #{}({})
Description	A block with an output that cannot drive a global control line is connected to a global net.

NetlistGlobalOutputNetMismatch (Error)

Message	Netlist inconsistency: Block #{}({}) global output {} drives non-global net #{}({})
Description	A block with an output that must drive a global line is connected to a non-global net.

NetlistGlobalPromotion (Info)

Message	Net {} promoted to global due to interface constraints
Description	A normal net in the core netlist is promoted to global because it is used as an interface clock in the design periphy.

NetlistIOInpadHasClock (Error)

Message	IO inpad #{} with output {} has clock
Description	IO inpad has a clock

NetlistIOInpadNumInputPins (Error)

Message	IO inpad logical_block #{} name {} of type {} has {} input pins
Description	IO inpad has an invalid number of input pins

NetlistIOInpadNumOutputPins (Error)

Message	IO inpad logical_block #{} name {} of type {} has {} output pins
Description	IO inpad has an invalid number of output pins

NetlistIOOutpadHasClock (Error)

Message io outpad #{} with name {} has clock

Description IO outpad has a clock

NetlistIOOutpadNumInputPins (Error)

Message io outpad logical_block #{} name {} of type {} has {} input pins

Description IO outpad has an invalid number of input pins

NetlistIOOutpadNumOutputPins (Error)

Message io outpad logical_block #{} name {} of type {} has {} output pins

Description IO outpad has an invalid number of output pins

NetlistIllegalConstGenerator (Error)

Message Internal Error: Non VCC/GND net '{}' marked as constant generator

Description Only VCC & GND nets may be marked as constant generators.

NetlistInitParseTimeElapsed (Info)

Message Loop for doall : {}, init_parse took {} seconds

Description The time elapsed for netlist parsing has been displayed.

NetlistInitReadTimeElapsed (Info)

Message Loop for doall : {} took {} seconds

Description The time elapsed for netlist reading has been displayed.

NetlistInputNetConnectingNetMismatch (Error)

Message Input net for block {} #{} is net {} #{} but connecting net is {} #{}

Description Mismatch between input net and connecting net to a block

NetlistInterfaceClkoutPinAlreadyDefined (Warning)

Message Interface clock output pin {} is already defined

Description The auto-generated name for an interface clock output collided with an existing name.

NetlistInterfaceClkoutPinCreated (Info)

Message Creating interface CLKOUT {}

Description A clock out pin is added to the core netlist to allow it to route to the periphery.

NetlistInterfaceClockBufferCreated (Info)

Message Creating interface clock buffer {}

Description A EFX_GBUFCE block is created for a normal net being promoted to global.

NetlistInterfaceClockNotFound (Error)

Message	Interface clock {} is not found
Description	An interface clock used in the design periphy, but the clock cannot be found in the core netlist.

NetlistInterfaceClockPinCreated (Info)

Message	Creating interface clock pin {}
Description	An interface clock pin is defined in the design periphy, but not the core netlist has been created.

NetlistInterfaceClockPinNotInput (Warning)

Message	Interface clock pin {} is not an input pin
Description	An interface clock pin is defined in the design periphy, but the cell in the core netlist is not an input.

NetlistLatchIncorrectNumParameters (Error)

Message	.latch does not have 5 parameters
Description	.latch does not have 5 parameters

NetlistLatchUnexpectedNumClockPins (Error)

Message	Latch #{} with output {} has {} clock pin(s), expected one (CLK)
Description	Unexpected number of clock pins on latch

NetlistLatchUnexpectedNumInputPins (Error)

Message	Latch #{} with output {} has {} input pin(s), expected one (D)
Description	Unexpected number of input pins on latch

NetlistLatchUnexpectedNumOutputPins (Error)

Message	Latch #{} with output {} has {} output pin(s), expected one (Q)
Description	Unexpected number of output pins on latch

NetlistLogicalBlockMultipleOutputPins (Warning)

Message	Logical_block #{} name {} of model {} has {} output pins instead of 1
Description	Logical block has more than one output pins

NetlistLogicalBlockNoFanout (Warning)

Message	logical_block {} #{} has no fanout
Description	A logical_block has no fanout

NetlistLogicalBlockNoFanoutOrFanin (Warning)

Message	logical_block {} #{} has no fanout or fanin
Description	A logical_block has no fanout or fanin

NetlistLogicalBlockUnknownType (Error)

Message	Unknown type for logical_block #{} {}
Description	Unknown type for logical_block

NetlistMissingDriver (Error)

Message	missing driver for net {}
Description	The specified net is is missing a driver

NetlistMultipleNetDrivers (Error)

Message	Number of drivers for net #{} ({}) has {} drivers
Description	Multiple drivers for net

NetlistMultipleSubcktSignalInst (Error)

Message	Two instances of {} subckt signal found in subckt {}
Description	Two instances of a subckt signal found in a subckt

NetlistNetsSweptThisPass (Info)

Message	Pass {}: Swept away {} nets with no fanout
Description	The number of dangling nets removed in this optimization pass.

NetlistNumClockNets (Info)

Message	circuit has {} clock net ({} of {} are internal clock); {} io_pads
Description	circuit has clock net

NetlistNumSweptNets (Info)

Message	Swept away {} nets with no fanout
Description	Nets with no fanout has been removed

NetlistNumUnconnectedBlocks (Info)

Message	{} unconnected blocks in input netlist
Description	Number of unconnected blocks in netlist

NetlistOutputNetConnectingNetMismatch (Error)

Message	Output net for block {} #{} is net {} #{} but connecting net is {} #{}
Description	Mismatch between output net and connecting net to a block

NetlistRemapRamLineFailed (Error)

Message	Failed moving net from {} to {} on {}, net is disconnected
Description	An error occured while trying to prepare the BRAM block for Place & Route.

NetlistRemovedLUTBuffers (Info)

Message	Removed {} LUT buffers
Description	Removed LUT buffers

NetlistRemovingBlock (Info)

Message	Removing block
Description	Removing block

NetlistRemovingInput (Info)

Message	Removing input
Description	Removing input

NetlistSweptNodes (Info)

Message	Sweept away {} nodes
Description	Swept away nodes

NetlistTotalSwept (Info)

Message	Swept away {} nets and {} blocks in total
Description	The total number of dangling nets and blocks removed.

NetlistUnsupportedCell (Error)

Message	Cell type "{}" is not supported in the current device
Description	The design netlist contains a primitive cell type that is not supported in the selected device.

NetlistUnsupportedCellCount (Error)

Message	Found {} unsupported cell type(s) in the post-synthesis netlist
Description	The total number of unsupported cell types in the design netlist.

NetlistVPackNetIsClockDriver (Info)

Message	vpack_net {} {} is a clock_net driver {} {} (model={})
Description	vpack_net is a clock_net drive

NetlistVprUnableToSweepHangingNode (Warning)

Message	Sweep hanging nodes in your logic synthesis tool because VPR can not do this yet
Description	Sweep hanging nodes in your logic synthesis tool because VPR can not do this yet

P2CReadFailed (Error)

Message	Failed to read partition to cells database "{}"
Description	Encountered an error while reading the partition to cells database file.

P2CWriteFailed (Error)

Message	Could not write to file: "{}"
Description	Encountered an error while writing the partition to cells database file.

PadNameDuplicate (Error)

Message	Pad name {} has defined {} times in the boundary file. Each pad must have a unique name
Description	Duplicate pad name

ParameterListNotEquivalentToCombination (Error)

Message	Parameter list in min_model file not equivalent to list in combination {}
Description	Parameter list in min_model file not equivalent to list in combination id

ParameterListNotEquivalentToReference (Error)

Message	Parameter list in combination id: {} not equivalent to reference combination {}
Description	Parameter list not equivalent to reference combination

ParameterNameMissingInMinModel (Error)

Message	Parameter name {} missing in min_model
Description	Parameter name is missing in min_model

ParamterNameMissingInCombID (Error)

Message	Parameter name {} missing in combination id: {}
Description	Parameter name is missing in combination id

PhysicalSynthesisJsonFileInvalid (Error)

Message	The Json File: {} is not valid and cannot be read
Description	The Json file is not valid and cannot be read.

PhysicalSynthesisJsonFileParseFailed (Error)

Message	Failed to parse Json File: "{}"
Description	Failed to parse Json File

PhysicalSynthesisJsonFilePathNotExist (Error)

Message	The Json File: {} does not exist
Description	The Json file does not exist.

PinNameDuplicate (Error)

Message	Package Pin name {} has defined {} times in the package file. Each pad must map to a unique pin name
Description	Duplicate pin name

PortSizeNetSizeMismatch (Error)

Message	Port size ({}:{}) and Net size ({}:{}) do not agree
Description	Port size and net size mismatch found

RoutabilityStatusReadFailed (Error)

Main Message	Failed to read routability metrics database '{}'
Action	The Efinity software has failed to read a routability metrics database
Sub-Message	Reason: {}
Description	Failure reason

RoutabilityStatusReadInvalidFileMagic (Error)

Message	The routability metrics database '{}' appears to be corrupted
Description	The Efinity software has encountered a corrupted routability metrics database

RoutabilityStatusReadInvalidSchemaVersion (Error)

Message	The routability metrics database '{}' appears to be from an older version of the Efinity software
Description	The Efinity software has encountered a corrupted routability metrics database

RoutabilityStatusRenameFailed (Error)

Main Message	Failed to rename routability metrics database from '{}' to {}
Action	The Efinity software has failed to rename a routability metrics database
Sub-Message	Reason: {}
Description	Failure reason

RoutabilityStatusWriteFailed (Error)

Main Message	Failed to write routability metrics database to '{}'
Action	The Efinity software has failed to write a routability metrics database
Sub-Message	Reason: {}
Description	Failure reason

SynthesizedCircuitStatistics (Info)

Main Message	Synthesized circuit stats:
Action	Synthesized circuit stats
Sub-Message	{} of type {}
Description	Model statistics
Sub-Message	{} LUTs of size {}
Description	LUT statistics

TimingReportWriteFailed (Warning)

Message	Failed to write timing report dump to '{}'
Description	Failed to write timing report dump

VdbDanglingInputFromClkBuffer (Error)

Message	Unexpected Dangling Input from CLK BUFFER, Synthesis Error
Description	Unexpected Dangling Input from CLK BUFFER

VdbDanglingOutputFromClkBuffer (Error)

Message	Unexpected Dangling Output from CLK BUFFER, Synthesis Error
Description	Unexpected Dangling Output from CLK BUFFER

VdbDecryptionEnvelopeParseFailed (Error)

Message	Failed to parse decryption envelope information
Description	Failed to parse decryption envelope information

VdbDigestMismatch (Error)

Message	Mis-matched digest content. Decryption failed
Description	Mis-matched digest content. Decryption failed

VdbFailedToDecryptKey (Error)

Message	Failed to decrypt key
Description	Failed to decrypt key

VdbFileDoesNotExist (Error)

Message	File {} does not exist
Description	VDB file does not exist

VdbFileRead (Info)

Message	Successfully read in VDB netlist file {}
Description	The design netlist was successfully read in from the VDB file on disk.

VdbFileReadFailed (Error)

Message	Failed to read in VDB netlist file {}
Description	The design netlist was not able to be created from the VDB file on disk.

VdbLicenseCheckoutFailed (Error)

Message	Failed to checkout valid license
Description	Failed to checkout valid license

VdbNoTopModule (Error)

Message	Cannot find top module in netlist: {}
Description	Cannot find top module in netlist

VdbP1735Version12PragmaVersionUnsupported (Error)

Message	Only support P1735 Version 1 or 2
Description	Only support P1735 Version 1 or 2

VdbP1735Version1PragmaVersionUnsupported (Error)

Message	Only P1735 Version 1 is supported
Description	Only P1735 Version 1 is supported

VdbP1735VersionPragmaUnspecified (Error)

Message	P1735 Version pragma must be specified
Description	P1735 Version pragma must be specified

VdbParseNetlistUnsupportedType (Error)

Message	Unsupported type for ParseNetlist
Description	Unsupported type for ParseNetlist

VdbToolControlCheckoutFailed (Error)

Message	Failed to verify tool control
Description	Failed to verify tool control

VdbUnsupportedVersionKey (Error)

Message	Version {} encryption cannot use {} key
Description	Incorrect key used for encryption version

XdbArchFileDoesNotExist (Error)

Message	Achitecture model "{}" cannot be found
Description	Achitecture model cannot be found

XdbArchLoadFailed (Error)

Message	Failed to load achitecture model file "{}"
Description	Failed to load achitecture model file

XdbDeviceLoadFailed (Error)

Message	Failed to load device database "{}"
Description	Failed to load device database

XdbFamilyLoadFailed (Error)

Message	Failed to load family database "{}"
Description	Failed to load family database

XdbTimingFileDoesNotExist (Error)

Message	Timing model "{}" cannot be found
Description	Timing model cannot be found

XdbTimingLoadFailed (Error)

Message	Failed to load timing model file "{}"
Description	Failed to load timing model file

Flow Messages

The software may issue the following messages during flow execution.

ConfFileMsgLimitOverriden (Warning)

Message	Message suppression option for {} is overriden by "{}": {}
Description	Message suppression option for the given message ID has been overriden by another option.

ConfFileUnexpectedValueType (Warning)

Main Message	Unexpected value type for key "{}" in message configuration file "{}"
Action	Unexpected value type for given key in the specified message configuration file.
Sub-Message	Given={}, Expected={}
Description	Context

ConnectionProfileOpenFailed (Error)

Message	Failed to open connection profile {}
Description	Failed to open connection profile

DanglingPartitionInterface (Warning)

Main Message	The design contains dangling partition interfaces. Partitions with undriven input interfaces and/or output interfaces with no fanout may be difficult to route during import compile. Consider tying off dangling interfaces to ensure routability during import compile
Action	The design contains dangling partition interfaces. Partitions with undriven input interfaces and/or output interfaces with no fanout may be difficult to route during import compile. Consider tying off dangling interfaces to ensure routability during import compile.
Sub-Message	Output interface "{}" in partition "{}" has no fanout
Description	Dangling output partition interface.
Sub-Message	Input interface "{}" in partition "{}" is undriven
Description	Dangling input partition interface.

DeprecatedOption (Warning)

Message	Use of efx_pnr option "{}" is deprecated
Description	Deprecated efx_pnr option.

EfinityHomeUnset (Error)

Message	The environment variable EFINITY_HOME is not set. Please run "{}" before launching Efinity executables and scripts
Description	The environment variable EFINITY_HOME is not set.

EfxSSLInitFailed (Error)

Message	Failed to initialise the SSL encryption library: {}
Description	The Efinity software has failed to initialise the SSL encryption library.

EfxStaFailedToCreateWorkDir (Error)

Message	Failed to create temporary work directory "{}". Please ensure there is sufficient disk space and filesystem permissions
Description	The Efinity software has failed to create a temporary work directory.

EfxStaFailedToWriteWorkFile (Error)

Message	Failed to write temporary work file "{}". Please ensure there is sufficient disk space and filesystem permissions
Description	The Efinity software has failed to write a temporary work file.

EfxStaLoadDatabasesFailed (Error)

Message	Failed to load the post-route design databases. Please ensure the design has been fully compiled before running the Efinity Static Timing Analysis Tool
Description	The Efinity software has failed to load the post-route design databases.

EfxStaMissingAction (Error)

Message	At least one operation must be specified. Supported operations include: {}
Description	At least one operation must be specified

EnteringTclInteractiveShell (Info)

Message	Entering Tcl interactive shell
Description	Efinity is entering the Tcl interactive shell

EvaluatingTclScript (Info)

Message	Evaluating user specified Tcl script "{}"
Description	Efinity is evaluating the specified Tcl script

ExcessUtilizationXLR (Error)

Message	Not enough physical locations for XLR blocks in current design. See placement report file "{}" for details
Description	Not enough physical locations for XLR blocks in current design

ExitingTclInteractiveShell (Info)

Message	Exiting Tcl interactive shell
Description	Efinity is exiting the Tcl interactive shell

FailedToFindInstanceInNetPath (Error)

Message	Cannot find inst '{}' in net '{}' connect path
Description	Failed to find instance in net path

FailedToFindNetlistItem (Error)

Message	failed to find {} {}
Description	Failed to find netlist item

FailedToFindNetlistItemWithCommand (Error)

Message	Cannot find {} "{}". command: "{}"
Description	Cannot find netlist item

FailedToFindPort (Error)

Message	Cannot find port {} with width {}
Description	Cannot find port

FailedToOpenInputFile (Error)

Message	Failed to open input file "{}" for reading
Description	The Efinity software has failed to open an input file.

FailedToReadMaplibFile (Error)

Message	Failed to read maplib {}
Description	Failed to read maplib file

FailedToReadMsgConfFile (Error)

Main Message	Failed to read message configuration TOML file "{}"
Action	Failed to read message configuration TOML file.
Sub-Message	Error: {}
Description	Failure reason
Sub-Message	at {}
Description	Source location

FlowBitstreamSupportNotAvailable (Warning)

Message	Bitstream support for this device not available. No bitstream file(s) will be generated
Description	Bitstream support for this device not available

FlowDeleteXmlReportFileFailed (Warning)

Message	Fail to delete xml report file {}. Error : {}
Description	The Efinity software has failed to delete an xml report file.

FlowDumpPrimitiveTimingAfterRouting (Info)

Message	Dump primitive timing after routing
Description	Dump primitive timing after routing

FlowFailedToOpenLogFile (Warning)

Message	Failed to open log file "{}" for writing: {}. File logging will be disabled for this run
Description	The Efinity software has failed to open a log file. File logging will be disabled for this run.

FlowIgnoreAutoTimingTradeoff (Warning)

Message	Ignoring auto_timing_tradeoff since timing_tradeoff is explicitly set to: {}
Description	Ignoring auto_timing_tradeoff since timing_tradeoff is explicitly set

FlowLeadRouteLoading (Info)

Message	Since load_route option is applied, VPR loads existing .place and .route files
Description	Since load_route option is applied, VPR loads existing .place and .route files

FlowMappedClbDimensions (Info)

Message	The circuit will be mapped into a {} x {} array of clbs
Description	The mapped clb dimension has been displayed

FlowPreparingXmlReportFile (Info)

Message	Prepare xml report data, file {}
Description	The Efinity software is preparing an xml report file

FlowReportHighResourceUtilization (Info)

Message	High resource utilization found. Software will optimize placement for routability
Description	High resource utilization found. Software will optimize placement for routability

FlowReportReplaced (Warning)

Message	{} will be removed and replaced with data in the current run
Description	A report file will be removed and replaced in the current run.

FlowReportTimingTradeoffExceedLimits (Info)

Message	This run will use the nearest allowable timing_tradeoff value {}
Description	This run will use the nearest allowable timing_tradeoff value

FlowReportUtilization (Info)

Message	The architecture has {} {}. Total {} utilization : {} %
Description	The device utilization hsa been displayed.

FlowReportUtilizationExceedLimits (Info)

Message	{} utilization {} % for architecture's device count {} exceeds the allowable limits
Description	EFL+EFT+EFM utilization exceeds the allowable limits

FlowReportUtilizationIndex (Info)

Message	utilization_index : {} and arch_size_index : {}
Description	Utilization indexes information has been displayed

FlowReportUtilizationIndexExceedLimits (Info)

Message	New utilization_index : {} and arch_size_index : {} just to get this run going
Description	Utilization indexes to get this run going

FlowReportUtilizationNearlyExceedLimits (Info)

Message	High resource utilization found. Software will optimize placement for routability
Description	High resource utilization found. Software will optimize placement for routability

FlowReportUtilizationNearlyExceedLimitsDetailed (Info)

Message	The {} utilization {} % for architecture's device count {} nearly exceeds the allowable limits. Still proceeding with placement
Description	The EFL+EFT+EFM utilization almost exceeds the allowable limits

FlowReportWriteFailed (Error)

Message	Failed to write to report file "{}"
Description	Failed to write to report file

FlowResettingTimingTradeoff (Info)

Message	Resetting timing_tradeoff since auto_timing_tradeoff on is set:
Description	Resetting timing_tradeoff since auto_timing_tradeoff on is set

FlowResettingTimingTradeoffNewValue (Info)

Message	{} {}: The timing_tradeoff was {}. New timing_tradeoff is {}
Description	The new timing_tradeoff value has been displayed

FlowVerboseOptionEnaabled (Warning)

Message	Verbose option is used. More messages would appear and the run would likely be slower
Description	Verbose option is used. More messages would appear and the run would likely be slower

FlowWriteXmlReportFileFailed (Warning)

Message	Fail to write xml report file {}. Exception : {}
Description	The Efinity software has failed to write an XML report file.

FlowWritingXmlReportFile (Info)

Message	Write xml report file {}
Description	The Efinity software is writing an XML report file

FlowXmlReportAddingCoreResourceGroup (Info)

Message	Add xml report group : Core Resource
Description	Add Core Resource xml report group

GroupOptions (Info)

Main Message	{} Options:
Action	Group options.
Sub-Message	{} : {}
Description	Option description.

HelpUsage (Info)

Message	Usage: {} [Options ...]
Description	Help text.

INIPnROptionIgnored (Warning)

Message	Ignored place and route setting "{}" from the settings file; This option may not be specified in the context of a settings file
Description	A setting has been ignored.

ImportPartitionEmptyFilename (Error)

Message	A Partition Archive (.part) file has not been specified for the Import Partition "{}"
Description	A Partition Archive (.part) file has not been specified for an Import Partition

ImportPartitionIncompatibleDevice (Error)

Message	The imported partition "{}" was generated for device {} {} which is incompatible with the current device selection {} {}
Description	An imported partition has incompatible device settings.

ImportRoutePartitionIncompatibleDevice (Error)

Message	The ImportRoute partition "{}" was generated for device {} {} which is incompatible with the current device selection {} {}
Description	An ImportRoute partition has incompatible device settings.

InvalidCommandLineOption (Error)

Message	Invalid command line option(s) : {}
Description	The command line option(s) provided to the Efinity software is invalid

InvalidInstanceNames (Error)

Main Message	The given design contains names with reserved characters. {} are reserved characters
Action	The given design contains names with reserved characters.
Sub-Message	Name "{}" contains reserved character(s)
Description	Invalid name.

InvalidOperatingConditionOption (Warning)

Message	Specified Operating Condition ({}) is invalid. Will set to the default model ({})
Description	The specified Operating Condition is not a valid option for the current device.

InvalidOptimizationLevel (Warning)

Main Message	Ignoring option "optimization_level={}" as it is not a valid optimization level
Action	The specified --optimization_level option is not a valid value.
Sub-Message	Supported options: {}
Description	List of supported --optimization_level options.

InvalidPartCellPlaceCnst (Error)

Main Message	Invalid placement location constraint(s) have been specified on cells contained within partitions
Action	Invalid placement location constraint(s) have been specified on cells contained within partitions.
Sub-Message	Cell "{}" in partition "{}" has been assigned to location ({}, {}), which is outside the partition bounding box {{ {} }}
Description	Invalid placement constraint.
Sub-Message	Interface cell "{}" in partition "{}" has been assigned to location ({}, {}), which is not on the perimeter of the partition bounding box {{ {} }}
Description	Invalid placement constraint.
Sub-Message	Non-Exclusive Cell "{}" has been assigned to a location ({}, {}) inside an exclusive partition {} with bounding box {{ {} }}
Description	Invalid placement constraint.

InvalidPartitionDefinitionAttribute (Error)

Message	Unsupported attribute "{}" specified for {} partition "{}" in partition definition file "{}":{}
Description	An unsupported attribute has been specified for a given partition.

MaplibFileNotDefined (Error)

Message	Maplib file not defined
Description	Maplib file not defined

MaxThreads (Info)

Message	Maximum concurrency has been set to {}
Description	The Efinity software will perform compilation with the specifed concurrency level.

NoConnectionProfileSpecified (Error)

Message	No connection profile specified
Description	No connection profile specified

NoInputSourceFile (Error)

Message	No source files defined, cannot perform Analysis
Description	No source files defined, cannot perform Analysis

NoInputSourceFileAnalyzed (Error)

Message	No source files analyzed, cannot perform Elaboration
Description	No source files analyzed, cannot perform Elaboration

NoModuleOrDesignUnitFound (Error)

Message	No module or design unit {} found
Description	No module or design unit found

NoRootDesignNameSpecified (Error)

Message	No root design name specified
Description	No root design name specified

NoRootSpecifiedWithRootArchSet (Error)

Message	No root specified with root-arch set
Description	No root specified with root-arch set

NoRootSpecifiedWithTopParametersSet (Error)

Message	No root specified with top-parameters set
Description	No root specified with top-parameters set

NoVdbFileSpecified (Error)

Message	No VDB file specified
Description	No VDB file specified

OptimizeCongEnabled (Info)

Message	Optimizing for congestion improvement
Description	Optimizing for congestion improvement.

PartitionBBoxAdjacent (Error)

Message	The bounding box specified for partition "{}" is adjacent to that of partition "{}"
Description	Two adjacent bounding boxes have been detected in a partition definition file.

PartitionBBoxAlignment (Error)

Message	y_min and/or y_max of the bounding box in "{}" are not aligned to the clock network
Description	y_min and/or y_max of a bounding box are not aligned to the clock network

PartitionBBoxCoordOutsideOfChip (Error)

Message	The bounding box specified for partition "{}" has coordinates which fall outside of the chip dimensions. The bounding box coordinates are x: [{}, {}), y: [{}, {}). The chip dimensions are x: [0, {}), y: [0, {})
Description	Bounding box coordinates detected which lie outside of the chip dimensions.

PartitionBBoxIllegalCoord (Error)

Message	The bounding box specified for partition "{}" has coordinates which do not meet the conditions x_min < x_max and y_min < y_max. The actual coordinates are x_min: x: [{}, {}), y: [{}, {})
Description	Bounding box coordinates detected which do not meet the basic conditions: x_min < x_max and y_min < y_max

PartitionBBoxIllegalHeight (Error)

Message	The bounding box specified for partition "{}" has an illegal height. Partition bounding boxes must have a minimum height of 80, and has to be a multiple of 80
Description	A bounding box with an illegal height has been detected in a partition definition file.

PartitionBBoxOverlap (Error)

Message	The bounding box specified for partition "{}" overlaps that of partition "{}"
Description	Two overlapping bounding boxes have been detected in a partition definition file.

PartitionBBoxSideOnNonXLRColumnErr (Error)

Message	The {} side of the bounding box for partition "{}" sits on a {} column
Description	Either the x_min or x_max-1 value of the bounding box for a given partition lines up with a non-XLR column. Please adjust the bounding box for this partition.

PartitionBBoxSideOnNonXLRColumnWarn (Warning)

Message	The {} side of the bounding box for partition "{}" sits on a {} column. This may reduce the number of available sites for partition interfaces, and may negatively affect the routability of the design
Description	Either the x_min or x_max-1 value of the bounding box for a given partition lines up with a non-XLR column. This may reduce the number of available sites for partition interfaces, and may negatively affect the routability of the design. We recommend adjusting the bounding box for this partition.

PartitionBBoxSideOnNonXLRRow (Error)

Message	The {} side of the bounding box for partition "{}" sits on a {} row
Description	Either the y_min or y_max-1 value of the bounding box for a given partition lines up with a non-XLR row. Please adjust the bounding box for this partition.

PartitionBBoxWidth (Error)

Message	The bounding box specified for partition "{}" must have width of at least {}
Description	Bounding box with insufficient width detected

PartitionClockInterfaceDangling (Error)

Message	Clock interface "{}" on partition "{}" is not driven
Description	A partition clock interface is not driven. All partition clock interfaces must be driven.

PartitionClockInterfaceLocationConstrained (Error)

Message	Clock interface "{}" on partition "{}" has been assigned a location constraint
Description	A partition clock interface has been assigned a location constraint. Location constraints may not be specied on clock interfaces.

PartitionConfigStale (Error)

Message	The partition configuration specified in the partition definition file does not match the configuration used to compile the current design through the earlier compilation stages
Description	The partition configuration specified in the partition definition file does not match the configuration used to compile the current design through the earlier compilation stages. Please rerun the compilation flow from the beginning to resolve this inconsistency.

PartitionDeclarationFileUsedDuringExport (Error)

Message	A partition declaration file "{}" is specified during a team-based-design export compile. Specify a partition definition file with partition bounding boxes instead
Description	A partition declaration file is specified during a team-based-design export compile. A partition definition file with partition bounding boxes is required during the team-based-design export stage.

PartitionDefinitionBBoxesIgnoredDuringImport (Warning)

Message	Partition bounding boxes specified in partition definition file "{}" are ignored during the team-based-design import compile
Description	Partition bounding boxes specified in partition definition file are ignored during the team-based-design import compile.

PartitionHierPathMapping (Info)

Message	Partition "{}" mapped to design hierarchy path "{}"
Description	The specified partition has been matched to a design hierarchy path.

PartitionHierarchyNotFound (Error)

Message	Partition "{}" assigned to unknown hierarchy path "{}"
Description	A partition has been assigned to a hierarchy path that does not exist in the design.

PartitionMissingIfaceCells (Error)

Message	Partition "{}" does not contain any interface cell. This is likely because the Synthesis stage was not performed with the same partition declaration file
Description	A partition does not contain any interface cell. This is likely because the Synthesis stage was not performed with the same partition declaration file. Rerun the Synthesis stage with the --partitions option to resolve this issue.

PartitionNesting (Error)

Message	The hierarchy path specified for partition "{}" shares a common prefix with that of partition "{}"
Description	Hierarchy paths with common prefixes have been detected in a partition definition file.

PartitionOnOperator (Error)

Message	Partition "{}" has been assigned to an operator "{}". Partitions may not be assigned to operators
Description	A partition has been assigned to an operator. Partitions may not be assigned to operators.

PartitionOnPrimitive (Error)

Message	Partition "{}" has been assigned to a device primitive "{}". Partitions may not be assigned to primitives such as EFX_FF
Description	A partition has been assigned to a device primitive. Partitions may not be assigned to primitives such as EFX_FF.

PartitionOptionMismatchNonPartitionDesign (Error)

Message	The current design was not previously compiled with partitions. Please rerun the compilation flow from the beginning to introduce partitions into the design
Description	The current design was not previously compiled with partitions. Please rerun the compilation flow from the beginning to introduce partitions into the design.

PartitionOptionMismatchUnspecified (Error)

Message	The current design was compiled with partitions during the earlier compilation stages but the --partition option is not specified for the current compilation
Description	The current design was compiled with partitions during the earlier compilation stages but the --partition option is not specified for the current compilation. Provide a partition definition file via the --partition option

PhysicalSynthesisInstanceDoesNotExist (Warning)

Message	The instance: {} does not exist in the current netlist
Description	The instance in the physical-synthesis split_net command does not exist.

PhysicalSynthesisLoadGroupsNotArray (Warning)

Message	The load_groups field must be an array
Description	The load_groups field must be an array.

PhysicalSynthesisLoadsMissing (Error)

Message	There are missing loads for the split_net command for net: {}
Description	There are missing loads in the split_net command. All sinks of the given net must be specified in load_groups.

PhysicalSynthesisMissingInstancePortRefLoads (Error)

Message	The physical_synthesis "split_net" command is invalid. The net: {} is missing a load for portref: {} on instance: {}
Description	There are missing instance or portref loads in the split_net command. All sinks of the given net must be specified in load_groups.

PhysicalSynthesisMissingPortLoads (Error)

Message	The physical_synthesis "split_net" command is invalid. The net: {} is missing a load for top-port: {}
Description	There are missing top-port loads in the split_net command. All sinks of the given net must be specified in load_groups.

PhysicalSynthesisNetDoesNotExist (Warning)

Message	The net: {} does not exist in the current netlist
Description	The net in the physical-synthesis split_net command does not exist.

PhysicalSynthesisNoLoads (Error)

Message	There are no loads specified in the split_net command
Description	There are no loads specified in the split_net command.

PhysicalSynthesisPortDoesNotExist (Warning)

Message	The port: {} does not exist in the current netlist
Description	The port in the physical-synthesis split_net command does not exist.

PhysicalSynthesisPortRefDoesNotExist (Warning)

Message	The portref: {} on instance: {} does not exist in the current netlist
Description	The portref specified for the instance in the physical-synthesis split_net command does not exist.

PhysicalSynthesisSplitNetFail (Warning)

Message	The physical_synthesis "split_net" has failed for net: {}. Skipping this command
Description	Unable to complete split_net command. Skipping command and moving to next.

PhysicalSynthesisSplitNetSuccess (Info)

Message	Successfully processed the split_net command for net: {}
Description	Successfully processed the split_net command for the current net.

PhysicalSynthesisVerificationFail (Warning)

Message	Verification failed after split_net command. Skipping the split_net command for net: {}
Description	Verification failed after split_net command. Skipping this split_net command.

PrjXMLOptionOverriden (Info)

Message	The project XML setting "{}" has been overriden from the command line
Description	A project XML setting has been overriden from the command line.

PrjXMLPnROptionIgnored (Warning)

Message	Ignored place and route setting "{}" from the project XML file; This option may not be specified in the context of a project XML file
Description	A project XML setting has been ignored.

PrjXMLUnknownPnROption (Warning)

Message	Ignored unrecognised place and route setting "{}" from the project XML file
Description	An unrecognised project XML setting has been ignored.

ProjectDatabaseAlreadyExists (Error)

Message	Project database has already existed. Fail to create new project
Description	Project database has already existed. Fail to create new project

ProjectDatabaseDoesNotExist (Error)

Message	Project XML file "{}" does not exist
Description	Project XML file does not exist

ProjectDatabaseFoundDeprecatedTagName (Warning)

Message	Found a deprecated tag name {}. Skipping
Description	Found a deprecated tag name

ProjectDatabaseParseFailed (Error)

Message	Failed to parse the project XML file "{}". The file format is unexpected
Description	Failed to parse the project XML file

ProjectDatabaseReading (Info)

Message	Reading project database "{}"
Description	Reading project database

ProjectDatabaseUnexpectedTagName (Error)

Message	Unexpected tag name {}
Description	Found an unexpected tag name

ProjectXmlNotSpecified (Error)

Message	A project XML file must be specified
Description	A project XML file must be specified

RSSUsage (Info)

Main Message	{} resident set memory usage: begin : {} MB, end : {} MB, delta : {} MB
Action	Resident set memory usage statistics.
Sub-Message	{} peak resident set memory usage : {} MB
Description	Peak resident set memory usage

RunningFlowStage (Info)

Message	Running {}
Description	The Efinity software is starting a flow stage

StageDuration (Info)

Main Message	{} took {} seconds
Action	Stage duration.
Sub-Message	{} took {} seconds (approximately) in total CPU time
Description	CPU time.

SyncHeadDrivenByNonSeq (Warning)

Message	Synchronizer chain starting at cell {} is driven by a non-sequential source {}
Description	Synchronizer head is driven by a Non-Sequential source A "Sequential-to-Sequential" path is recommended for Clock Domain Crossing

SyncHeadDrivenFromNonOverlappingRegion (Warning)

Message	Synchronizer chain starting at cell {} is constrained to region {} while its driver, {}, is constrained to region {}. These regions do not overlap
Description	The Driver and Synchronizer Cells are constrained to non-overlapping regions. This could lead to skew related issues

TrionBeneficialSkewUnsupported (Warning)

Message	Trion devices do not support the --beneficial_skew option
Description	Trion devices do not support the --beneficial_skew option.

TrionFirstIterCSSNotSupported (Warning)

Message	Trion devices do not support the --router_first_iter_css option
Description	Trion devices do not support the --router_first_iter_css option.

UnassignedPartitionInterface (Warning)

Message	The design contains unassigned partition interfaces. Unassigned partition interfaces will be randomly placed. See the placement report for the list of unassigned partition interfaces
Description	The design contains unassigned partition interfaces. Unassigned partition interfaces will be randomly placed. See the placement report for the list of unassigned partition interfaces.

UnrecognisedPartitionDeclarationFileHeader (Error)

Message	The specified partition {} file "{}" is missing a version identifier
Description	The given partition declaration or definition file is missing a version identifier. Partition declaration or definition files must start with "# Pdefn v<major>.<minor>".

UnrecognisedPartitionHierPath (Error)

Main Message	The partition {} file "{}" contains hierarchy path(s) which do not match any known design hierarchy
Action	The given partition declaration or definition file contains hierarchy path(s) which do not match any known design hierarchy.
Sub-Message	Partition "{}" is mapped to unrecognised hierarchy path "{}"
Description	Invalid hierarchy path.

UnsupportedDevice (Error)

Main Message	Unsupported device {} for family {}. Current supported devices for the current family are:
Action	Unsupported device option.
Sub-Message	- {}
Description	Family name.

UnsupportedFamily (Error)

Main Message	Unsupported family {}. Current supported families are:
Action	Unsupported family option.
Sub-Message	- {}
Description	Family name.

UnsupportedPartitionDeclarationFileVersion (Error)

Message	Unsupported partition {} file version {}.{} specified in "{}"
Description	A partition declaration or definition file has a version identifier that is unsupported by the current version of the Efinity software.

UsingDefaultDevice (Info)

Message	Using default device {} for family {}
Description	Using the default device for family.

UsingDefaultFamily (Info)

Message	Using default family {}
Description	Using the default family.

UsingDefaultTimingModel (Info)

Message	Using default timing model {} for device {} in family {}
Description	Using the default timing model for the current device.

VDBOudatedTeamBasedDesignNotSupported (Error)

Message	The Synthesis Verific Database "{}" is from an earlier version of the Efinity software which is not compatible with the team-based-design flow
Description	The Synthesis Verific Database provided is from an earlier version of the Efinity software which is not compatible with the team-based-design flow. Rerun the Synthesis compilation stage to resolve this issue.

VDBOutdatedNoHierUtilReport (Warning)

Message	The Synthesis Verific Database "{}" is from an earlier version of the Efinity software. The Place and Router hierarchical utilisation report will not be generated
Description	The Synthesis Verific Database provided is from an earlier version of the Efinity software. The Place and Router hierarchical utilisation report will not be generated. Rerun the Synthesis compilation stage to resolve this issue.

VMUsage (Info)

Main Message	{} virtual memory usage: begin : {} MB, end : {} MB, delta : {} MB
Action	Virtual memory usage statistics.
Sub-Message	{} peak virtual memory usage : {} MB
Description	Peak virtual memory usage

VdbDoesNotExist (Error)

Message	File {} does not exist, Aborted
Description	Vdb file does not exist

VdbFileNotSpecified (Error)

Message	VDB File not specified, Aborted
Description	VDB File not specified

VdbFlowFailed (Error)

Message	Vdb Flow has encountered an error: {}
Description	Vdb Flow has encountered an error

VdbLoadFailed (Error)

Message	Failed to restore netlist from file "{}", Aborted
Description	Failed to restore netlist from file

VdbMerging (Info)

Message	Going to Merge VDB
Description	Going to Merge VDB

VdbPrepareToWrite (Info)

Message	Going to Write NetlistDB
Description	Going to Write NetlistDB

VdbWriting (Info)

Message	Write NetlistDB {}
Description	Writing NetlistDB

XmlSchemaFileDoesNotExist (Warning)

Message	XML schema file "{}" does not exist
Description	XML schema file does not exist

Packer Messages

The software may issue the following messages during packing.

AutoPinIOResourceExhaustion (Error)

Main Message	Number of unassigned IO pins has has exceed auto-assignment eligible IO resources:
Action	Number of unassigned IO pins has has exceed auto-assignment eligible IO resources.
Sub-Message	{} : {}/{} resources used
Description	Auto-assignment eligible IO resource utilisation

CellPackingStatistics (Info)

Message	Packed {} lut-->ffs and {} ff-->luts out of {} ({} were lonely)
Description	Cell packing statistics

ClockResourceExhaustion (Error)

Main Message	Clock buffer usage has exceed device resources:
Action	Clock buffer usage has exceed device resources.
Sub-Message	{} : {}/{} resources used
Description	Clock resource utilisation

ClusterPackingStatistics (Info)

Message	Packed into {} clusters with {}({}) inputs(outputs) per cluster for a total of {}({}) inputs(outputs)
Description	Cluster packing statistics

DSPPackingStatistics (Info)

Message	Packed {} fracturable DSPs into {} clusters
Description	DSP packing statistics

MacroExceedsRegionHeight (Error)

Message	Macro chain with cell {}, has a length ({}) that exceeds the height ({}) of the assigned Region {}
Description	The length of the macro assigned to a Region cannot exceed the height of the Region

MacroExceedsRegionHeightUserErrorMsg (Error)

Message	One or more Macro Chains exceed the height of their assigned region(s)
Description	A Macro Chain cannot be placed into a region that is shorter than the chain itself Increase the height of the affected regions

MacroExceedsRegionWidth (Error)

Message	Macro chain with cell {}, has a length ({}) that exceeds the width ({}) of the assigned Region {}
Description	The length of the macro assigned to a Region cannot exceed the width of the Region

NetlistStatistics (Info)

Message	Netlist has {} luts, {} ffs and {} adders
Description	Netlist statistics

XLRPackingStatistics (Info)

Message	Packed into {} xlrs with {}({}) inputs(outputs) per xlr for a total of {}({}) inputs(outputs)
Description	XLR packing statistics

Placer Messages

The software may issue the following messages during placement.

BranchConstraintViolations (Warning)

Message	Total {} constraint violations: {}
Description	Unable to find a placement that resolved all clock/reset legalities.

CellInMultipleRegions (Error)

Message	Cell {} is driven by two Regional Clocks, which is not supported by Efinity. Reassign one of the Regional Clocks as a Global clock using the Interface Designer
Description	Efinity does not support Cells driven by two or more Regional clocks Using the Interface Designer, reassign one of the Regional Clocks (rclk) as a Global clock (gclk)

CellOutsideBBox (Error)

Message	Cell {} at location ({}, {}) assigned to Partition {} is not placed inside the Bounding-box : [x_min: {}, x_max: {}), [y_min: {}, y_max: {})
Description	Region Constrained cell is placed outside the region

ClkSinkPlacementConstrainedOutsideClkRegion (Error)

Message	Cell "{}" which is driven by clock driver "{}" ({}, {}) is placement constrained to a location ({}, {}) outside the associated clock region xmin: {}, xmax: {}, ymin: {}, ymax: {}
Description	A clock driven cell has been placement constrained to a location outside the legal clock region Adjust placement constraints to satisfy the clock region requirements

FileIOError (Error)

Message	Failed to {} file "{}"
Description	Fatal I/O Error Ensure input files exist. Ensure write permissions and space to write files exist

IdentifiedIntersectionRegionMsg (Info)

Message	Identified Intersection Region: "{}", Type: {}, Exclusive: {}, with Bounding-box : [x_min: {}, x_max: {}) x [y_min: {}, y_max: {})
Description	Properties of Region being imported in QPlacer

IdentifiedRegionMsg (Info)

Message	Identified Region: "{}", Type: {}, Exclusive: {}, with Bounding-box : [x_min: {}, x_max: {}), [y_min: {}, y_max: {})
Description	Properties of Region being imported in QPlacer

IdentifiedValidOverlap (Info)

Message	Identified cells constrained to multiple regions; computing intersection regions
Description	One or more cells are constrained to multiple regions simultaneously, resulting in overlapping region assignments Intersection regions are computed to reconcile these constraints

InfeasibleLegalizationErrorMsg (Error)

Message	Unable to find a legal placement for the design
Description	Efinity is unable to find a legal placement solution for the Cell Review partition constraints and resource utilization

InfeasibleMacrosErrorMsg (Error)

Message	Macro chain with cell: {}, height: {}, and type: {} is infeasible
Description	Efinity is unable to find a legal placement solution for the Macro chain Review partition constraints and resource utilization

InfeasibleMacrosFinalErrorMsg (Error)

Message	Design contains {} infeasible Macro Chains
Description	Efinity is unable to find a legal placement solution due to infeasible Macro Chains such as Adder Chains, SRL8 Chains, and DSP chains Review partition constraints and resource utilization

InfeasibleNodeErrorMsg (Error)

Message	Unable to find a legal placement solution for the Cell: {} and type: {}
Description	Efinity is unable to find a legal placement solution for the Cell Review partition constraints and resource utilization

InterfaceNotOnBoundary (Error)

Message	Interface Cell {} at location ({}, {}) assigned to Partition {} is not placed on the boundary of Bounding-box : [x_min: {}, x_max: {}), [y_min: {}, y_max: {})
Description	Interface Cell is not placed on the boundary of the assigned region

InvalidFixedPlacement (Error)

Message	Overlapping fixed placement at location x: {} y: {} detected for cells: {} and {}
Description	Overlapping fixed cells Review and verify any locked down placement provided as input

KeepOutCSVAddError (Error)

Message	Error in adding keepout locations from csv file
Description	The Efinity software encountered errors in adding keepout locations from csv file

KeepOutCSVCoordinatesOutOfBounds (Error)

Message	X,Y coordinates {},{} specified in keepout CSV file "{}" at line {} is outside of the chip dimensions
Description	X,Y coordinates specified in keepout CSV file is out of bounds.

KeepOutCSVUnexpectedLineFormat (Error)

Message	Unexpected format in keepout CSV file: "{}" at line {}. Keepout CSV files should contain lines with the format <x->,<y->
Description	Unexpected format in keepout CSV file.

OverlappingCellRegionAssignmentNoAreaOverlapp (Error)

Message	Cells ({}) are assigned to non overlapping regions ({})
Description	The listed cells are assigned to more than one region, yet the corresponding regions do not overlap geometrically Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToExclusiveRegion (Error)

Message	Cell {} is assigned to multiple regions: {}, but two or more regions are marked exclusive
Description	A cell may belong to multiple regions only when all involved regions allow sharing Cell cannot be assigned to multiple exclusive region Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToMoreThanOnePartition (Error)

Message	Cell {} is assigned to more one Partitions: {}
Description	A cell may not belong to multiple Partition Regions Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToMoreThanThreeRegions (Error)

Message	Cell {} is assigned to more than three Regions: {}
Description	A cell may not belong to more than three Regions Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToMoreTwoClockRegion (Error)

Message	Cell {} is assigned to more two clock regions: {}
Description	A cell may not belong to more than two clock regions Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToRegionWithMoreThanOneRect (Error)

Message	Cell {} is assigned to multiple regions: {}, but at least one of these regions have multiple defining rectangles
Description	A cell may belong to multiple regions only when all involved regions are defined by a single rectangle. If any region in the assignment set is defined by multiple rectangles, then overlap is prohibited Adjust region assignments to satisfy overlap constraints

OverlappingCellRegionAssignmentToRegionsGeneral (Error)

Message	An error occurred while handling overlapping cell assignments to regions
Description	One or more cells assigned to multiple regions without satisfying the required overlap conditions Adjust region assignments to satisfy overlap constraints

PartitionKeepOutAddError (Error)

Message	Error in adding keepout locations for partitions
Description	The Efinity software encountered errors in adding keepout locations for partitions Ensure that the placer_keepout files are present for all partitions

PartitionKeepOutFileReadFailed (Error)

Message	Unable to read keepout file "{}"
Description	The Efinity software is unable to read a keepout file Ensure that the .placer_keepout files are present for all partitions

PartitionKeepOutOnFixed (Error)

Message	Cannot fix a cell at a Keepout Site ({}, {})
Description	A Fixed Cell cannot be assigned to a Keepout site Ensure that the fixed cells are not assigned to keepout site

ResourceCapacityViolations (Error)

Main Message	{} Region lacks sufficient resources for the assigned cells
Action	Region lacks sufficient resources for the assigned cells
Sub-Message	There are more {} cells than available {} sites ({} cells, {} sites)
Description	There are more cells than available sites

TileDeficit (Error)

Message	Detected {} {} signals in {} tile region ({} {}) =-> ({} {}). There is a limit of {}
Description	post-placement check fail

TileDeficitWarning (Warning)

Message	Detected {} {} signals in {} tile region ({} {}) =-> ({} {}). There is a limit of {}
Description	post-placement check fail downgraded to warning

UnassignedIOCell (Warning)

Message	IO cell "{}" has no assigned placement; it will be placed randomly
Description	An IO cell has no assigned placement.

ValidChainsLoneSink (Error)

Main Message	Found {} "{}" {} output driving two or more sinks:
Action	Invalid placement, will fail post-placement checks Check netlist connectivity and ensure correct connectivity
Sub-Message	Sink: {}
Description	Sink enumeration

ValidChainsPlacementMismatch (Error)

Message	Found {} "{}" at location ({}, {}) with {} output driving non-adjacent {} "{}" at location ({}, {})
Description	Invalid placement, will fail post-placement checks Check netlist connectivity and ensure correct connectivity. Ensure locked-down cells are placed in valid locations

ValidChainsTypeMismatch (Error)

Message	{} cell {}'s {} output drives {}, which is an {} cell not an {} cell
Description	Invalid placement, will fail post-placement checks Check netlist connectivity and ensure correct connectivity

Router Messages

The software may issue the following messages during routing.

EstimatedRoutabilityMetrics (Info)

Message	Routability Estimation
Description	Estimated routability metrics.

UsingGlobalRoutingWithTimingDrivenPlacement (Warning)

Message	Using global routing with timing-driven placement. This is allowed, but strange, and circuit speed will suffer
Description	Using global routing with timing-driven placement. This is allowed, but strange, and circuit speed will suffer.

SDC Constraint Messages

The software may issue the following messages when processing SDC constraints.

CantReadConstraintDB (Error)

Message	Failed to read constraint database from '{}': {}
Description	Problem reading the timing constraint database.

CantReadConstraintDB2 (Error)

Message	Failed to read constraint database from '{}'
Description	Problem reading the timing constraint database.

CantWriteNetlistDB (Error)

Message	Failed to write netlist database to '{}'
Description	Problem writing the timing netlist database.

CorruptConstraintDB (Error)

Message	The constraint database '{}' appears to be corrupted
Description	Problem reading the timing constraint database.

CpdAndSetupSlackInfo (Info)

Message	{} {} to {} {} cpd: {} setup_slack: {}
Description	cpd and setup slack information

GenerateCDCReportFailed (Error)

Message	GenerateReportCDCReport failed to generate CDC paths report
Description	GenerateReportCDCReport failed.

GenerateReportDelayPathsReportFailed (Error)

Message	GenerateReportDelayPaths failed to generate delay paths report
Description	GenerateReportDelayPaths failed.

GenerateTimingPathsReportFailed (Error)

Message	GenerateReportTimingPaths failed to generate timing paths report
Description	GenerateReportTimingPaths failed.

HoldInfo (Info)

Message	pass {} hold Tns {} after iteration of allocating negative slack delays
Description	Hold information

IncompatibleConstraintDB (Error)

Message	The constraint database '{}' appears to be from an incompatible version of the Efinity software
Description	The constraint database is from a version of Efinity that is not compatible with this one.

LBAAvgSlackAndDeltaInfo (Info)

Message	Iteration {} of LBA average slack {} average_slack_delta {}
Description	LBA average slack and delta information

LoadTimingFromFileNotFound (Error)

Message	LoadTimingFromFile called, but file {} not found
Description	Timing model file was not found.

MaxDeviceFreq (Info)

Message	Max device frequency is {} MHz
Description	The maximum clock frequency of the device limited to this value.

RefPinModified (Error)

Main Message	The Efinity timing engine has found incompatible modifications to IO delay constraints in your design. The -reference_pin parameter on set_input_delay and set_output_delay SDC constraints may not be modified between Place and Route stages. Consider reverting those changes or rerun the Place and Route flow from the beginning
Action	The Efinity timing engine has found incompatible modifications to IO delay constraints in your design. The -reference_pin parameter on set_input_delay and set_output_delay SDC constraints may not be modified between Place and Route stages Consider reverting those changes or rerun the Place and Route flow from the beginning
Sub-Message	Expected: {}: {}
Description	Expected IO delay constraint
Sub-Message	Actual: {}: {}
Description	Actual IO delay constraint

RoutingNotLoaded (Error)

Message	{} called when routing not loaded
Description	Internal flow control issue. Timing function called before routing loaded.

TclCommandError (Warning)

Message	Suppressed error: {}
Description	Tcl command failed.

TclFileDoesNotExist (Error)

Message	File '{}' does not exist
Description	Tcl file does not exist.

TclSetupFailed (Error)

Message	Unable to Setup Tcl Interpreter: {}
Description	Error encountered setting up the Tcl Interpreter.

TimingAlreadyLoaded (Error)

Message	{} called when timing already loaded
Description	Internal flow control issue. Timing function called before timing loaded.

TimingDataPrelim (Info)

Message	NOTE: The timing data is not final
Description	Timing model is preliminary.

TimingNotLoaded (Error)

Message	{} called when timing not loaded
Description	Internal flow control issue. Timing function called before timing loaded.

UnloadTimingFailed (Error)

Message	Failed to unload timing
Description	Failed to unload timing.

Timing Analysis Messages

The software may issue the following messages during timing analysis.

CannotDeriveGeneratedClockFromSpecifiedSourcePin (Error)

Message	Error in generated clock {} : cannot derive master clock from the specified source pin
Description	Cannot find generated clock from the specified source pin

CannotFindSpecifiedMasterClock (Warning)

Message	Clock {} is specified as the master clock for generated clock {} but it is not present in the fanin of the clock target. The fanin contains the following clocks: {}
Description	Cannot find specified master clock from timing netlist

CannotTraceBackToMasterClock (Warning)

Message	Could not trace back to master clock from target pin {}. Will assume zero latency on derived clock {}
Description	Timer cannot trace back to master clock from target pin

ClockPeriodIsTooLarge (Warning)

Message	The clock period of {} is too large. Set to maximum clock period (0.5ms)
Description	The maximum clock period for timer is 0.5ms

ClockPeriodIsTooSmall (Warning)

Message	The clock period of {} is too small. Set to minimum clock period (1ps)
Description	The minimum clock period for timer is 1ps

CombinationalLoopDetected (Warning)

Message	Timer cuts a combinational loop at the {}
Description	A combinational loop in the timing netlist is detected. Timer cuts the edge an arbitrary point

IgnoreThruClockPinInException (Warning)

Message	When specifying exceptions, the -through cannot contain clocks. Ignoring the -through clock spec for this exception
Description	When specifying exceptions, the -through cannot contain clocks. Ignoring the -through clock spec for this exception

MissingNetInQDelay (Warning)

Message	Missing net on cell: {} port: {} in precomputed delay
Description	Missing net in precomputed delay

TooManyExceptionsAreInputted (Warning)

Message	Too many exceptions are inputted. That would slow down timing analysis. Please check your sdc constraints
Description	Timer cannot handle too many user exceptions

Unified Timing Flow Messages

The software may issue the following messages when running the unified timing flow.

UnsupportedPTConnection (Error)

Message	Detects unsupported connection on {}. Check the warning messages in Interface Designer
Description	This connection is not supported in Unified Timing flow.

Chapter 15

Appendix: Efinity Tools

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

Table 49: 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 172
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 125
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 99
Debug Wizard	Provides an automated flow for adding a logic analyzer core to your design.	Debug Wizard on page 106
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 60
IP Packager	Use this tool to "package" design files for re-use in the IP Manager.	Packaging Design Files on page 32
IP Manager	Interactive wizard that helps you customize and generate Efinix IP cores.	Using the IP Manager on page 50
JTAG SVF Player	JTAG SVF player that sends JTAG commands to an Efinix FPGA.	Using the Efinity SVF Player on page 181
Log Message	Tool to sort and browse through all of the messages resulting from the compilation flow.	Viewing Messages and Logs on page 41
Message Browser	Shows synthesis-specific messages that result when you elaborate the netlist.	Viewing Messages and Logs on page 41
Netlist Viewer	Displays and analyzes your design's netlist, including all components and their connections (nodes and nets).	Netlist Viewer (Beta) on page 38
Package Planner	Provides a visual representation of the FPGA package pins.	Editing and Viewing the Package Pinout on page 74
Programmer	Select bitstream images and program the FPGA directly or the flash device on a board.	About the Programmer GUI on page 122
Efinity RISC-V Embedded Software IDE	Develop and debug software for the Sapphire SoC suite of RISC-V processors.	Efinity RISC-V Embedded Software IDE on page 49
Tcl Command Console	Enter Tcl commands to analyze and explore timing.	

Tool	Description	Read More
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 on page 115

Chapter 16

Reference: 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.

Chapter 17

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

Chapter 18

Bitstream Generation

Contents:

<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.

Chapter 19

Debugger

Contents:

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 106.

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 102 or Logic Analyzer Debug Core on page 104 for more information.

Chapter 20

Interface Designer

Contents:

<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.

Chapter 21

IP

Contents:

<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 57 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.

Chapter 22

Partitions

Contents:

<project>.<partition>.part

In GUI	-
In file system	<project>**/outflow**
Created by	Efinity software after compilation for an export design partition
Design source?	Yes

The partition output file (**.part**) defines the partition, its interface, and its placement and routing. The software creates a partition output file for each export partition at the end of compilation. Partitions are named <*project*>.<*partition*>**.part** and saved in your project's **outflow** directory.

Important: To use this file in a new project, move or copy it to your new project directory.

<project>.pdefn

In GUI	Enter in the **Project Editor** > **User Partitions** > **Partition Definitions** **File** box
In file system	Project directory
Created by	Efinity software when you create a design partition
Design source?	Yes

The partition definition file (**.pdefn**) contains the settings for a partition, which represents a region of the core logic and its associated design hierarchy. You use this file with a partition interface file (.pifce).

- The software ignores empty lines and lines starting with # (comments).
- Each row represents a partition; separate fields with a semicolon (;).
- Enclose the filename with quotation marks ("") if the path includes spaces.

The following code shows an example partition definition file.

```
# Pdefn v0.3
# <name> ; <hier path> ; <type> ;{ <xMin>, <xMax>, <yMin>, <yMax> }/<filename>; <exclusive>
Alu      ; alu          ; export ;{     13,     53,     242,     322 }             ; false
Register ; register     ; import ;           ../step1/outflow/r4000.Register.part; false
Cntrl    ; cntrl        ; region ;{     27,     137,     42,     122 }             ; false
```

<*project*>.pifce

In GUI	Enter in the **Project Editor** > **User Partitions** > **Partition Interface File** box
In file system	Project directory
Created by	Efinity software when you create a design partition
Design source?	Yes

The partition interface file (**.pifce**) describes the partition names, interface names, and location in (x, y) coordinates. You use this file with a **partition definition file (.pdfn)**.

- The software ignores empty lines and lines starting with # (comments).
- Each row prepresents a partition; separate fields with a semicolon (;).
- The interface name corresponds to a port name from the associated RTL module.
- Each partition bit must be specified individually.
- The Efinity placer automatically places unconstrained interface cells.

The following code shows an example partition interface file.

```
# partition name ; interface name ;   x ;   y
#-----------------------------------------------
  Cntrl          ; Op[2]          ;  55 ;  42
  Cntrl          ; Op[3]          ;  45 ; 121
  Cntrl          ; Op[1]          ; 136 ;  67
  Alu            ; PC[0]          ;  20 ; 133
  Alu            ; PC[1]          ;  20 ; 141
  Alu            ; PC[2]          ;  43 ; 281
```

Chapter 23

Placement

Contents:

<*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 has several sections:

- *Resource Summary*—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.
- *Module Resource Usage Distribution Estimates*—Lists resource usage at the cluster level for instantiations of modules in the user RTL.

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.

Chapter 24

Project

Contents:

<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 in the **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!

Chapter 25

Routing

Contents:

<*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 Guideor 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 several 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.

Chapter 26

Synthesis

Contents:

<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 gives the top-level entity, 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.

Chapter 27

Appendix: Shortcuts

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

Table 50: 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**.

Chapter 28

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

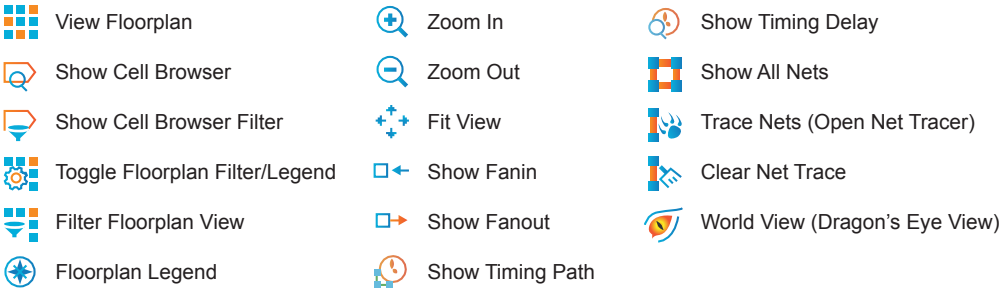
 Route

 Place & Route

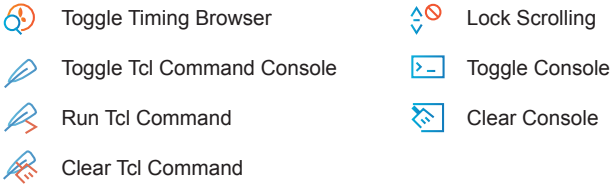
 Stop

 Generate Bitstream File

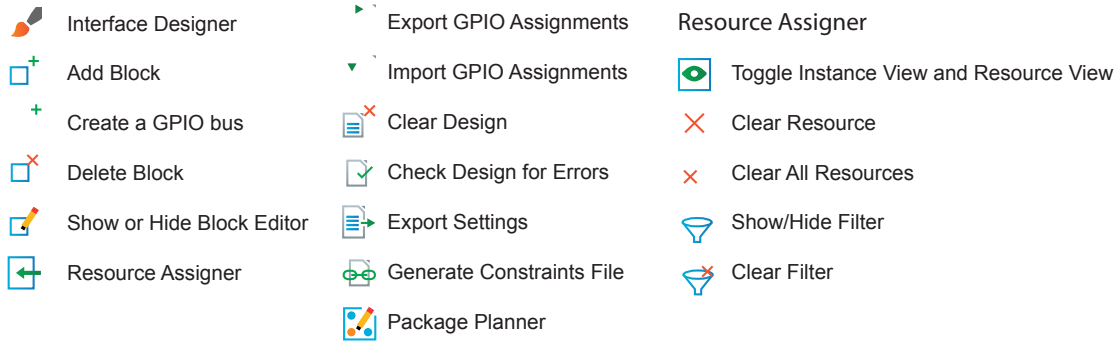
Floorplan Icons



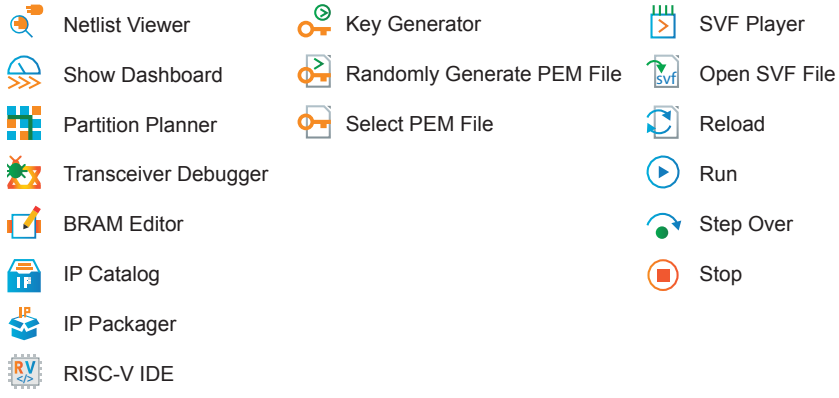
Timing & Console Icons



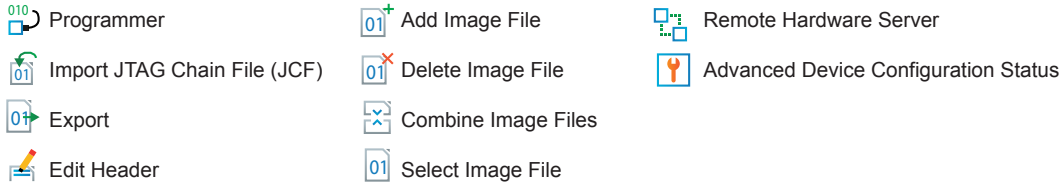
Interface Designer Icons



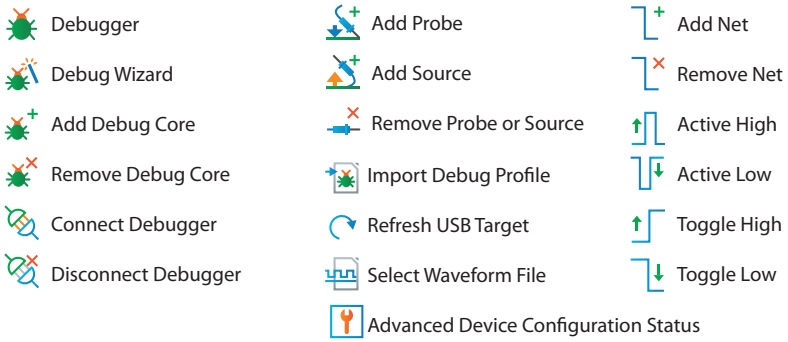
Tools Icons



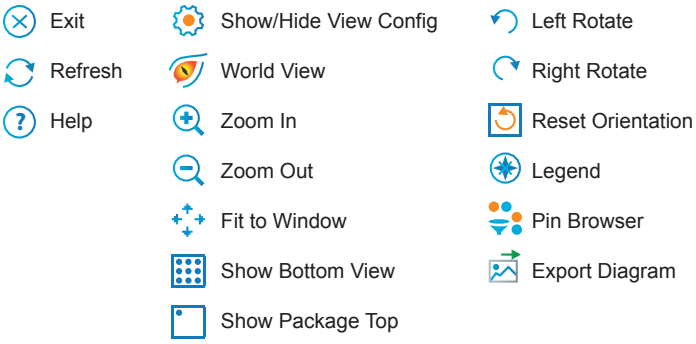
Programmer Icons



Debugger Icons



Package Planner Icons



Chapter 29

Appendix: Program using a JTAG Bridge (Legacy)

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

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

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

To program using a JTAG bridge:

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

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

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

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

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

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

Refer to Using the Efinity Bitstream Security Key Generator on page 172 for information on signing existing **.bit** files.

Chapter 30

Revision History

Table 51: Document Revision History

Date	Version	Description
June 2026	18.1	Ti125 F225 has bitstream support. Corrected typo in messages topics. (DOC-3103) Added PRBS Generator and Checker IP core to **Supported IP Cores by Family** on page 51.
May 2026	18.0	Updated for Efinity software v2026.1. Updated device support. Added new section on Partition Planner. (DOC-3012) Added Sapphire RV64 IP core and PMA 64b66b GearBox IP core to **Supported IP Cores by Family** on page 51. Updated Programmer GUI description for new options. (DOC-3068) Added info messages to **Programmer Messages** on page 146. (DOC-3066) Updated the options for **Working with Remote Hardware** on page 185. (DOC-3068) Added information about the **Enable Unified Netlist Flow** option in **Design Tab** on page 22. (DOC-3046) Updated messages.
March 2026	17.2	Added **Programmer Messages** on page 146. (DOC-2862) Bitstream support for Ti135 N576D2F4. Added MIPI 2.5G DSI TX, MIPI 2.5G DSI RX, and Divider_2 IP cores to **Supported IP Cores by Family** on page 51.
December 2025	17.1	For Titanium and Topaz FPGAs, the default setting for the **Project Editor** > **Bitstream Generation tab** > **Clock Sampling Edge** option is **Falling** (not **Rising**). (DOC-2804)

Date	Version	Description
November 2025	17.0	Updated for Efinity software v2025.2. Documented new Package Planner features. Described Python-JTAG co-debug feature. IP Manager has a new feature for out of context (OOC) flows. Updated debug profile filenames and steps for 2025.2 release. (DOC-2615) Added information about configuring external text editors to **Setting General Tool Preferences** on page 18. (DOC-2692) Added note about GUI mode option for Aldec Active HDL simulation. (DOC-2694) Added note about padding in combined bitstream images in **Combine Bitstreams and Other Files** on page 131. Added messages that the software may issue during compilation. (DOC-2611) Removed topics on installing USB drivers. Refer to the **Efinity Software Installation User Guide** instead. Updated memory recommendations for 2025.2 release. (DOC-2703) Updated default project file location. (DOC-2744) Added limitation for iVerilog simulator: cannot simulated encrypted code (IP) or SystemVerilog. (DOC-2636) Transceiver Debugger cannot determine the PCIe link width and speed when the PCIe link is in the polling compliance state (this behavior is expected). (DOC-2738) Updated instructions for using the BRAM Initial Content Updater example files. (DOC-2717) Added Module Resource Usage Distribution Estimates table description to the **.place.rpt** project file appendix. (DOC-2705) Updated content in **Concurrent Debugging** on page 113 to reflect new procedures. (DOC-2731) IP Manager generates additional files for out-of-context (OOC) synthesis. (DOC-2761)
September 2025	16.4	Updated for patch 2025.1.110.5.x. Updated device support.
August 2025	16.3	Updated for patch 2025.1.110.4.x. Updated Interface Primitive Simulation Models table.
July 2025	16.2	Updated for patch 2025.1.110.3.x. Updated device support.
June 2025	16.1	Updated for patch 2025.1.110.2.x. Updated device support.

Date	Version	Description
May 2025	16.0	Updated for Efinity software v2025.1. The JTAG SPI Flash Loader IP Core is removed from all families. It is replaced by the JTAG to SPI Flash IP core. (DOC-2285) Removed "Appendix: Connecting Programming Hardware." Refer to the configuration application notes for this information. SPI Active using JTAG Bridge (New) renamed as SPI Active using JTAG Bridge. Moved topic on SPI Active using JTAG Bridge (Legacy) to appendix.(DOC-2250) Added topic describing **Setting User and Project Directories** on page 19. (DOC-2465) Removed Auto-Load Place-and-Route topic. This feature is no longer needed with v2025.1. (DOC-2284) Updated machine memory requirements. (DOC-2286) Added "Using Mark Debug" subtopic in "Debugging Overview". (DOC-2344) Updated description of where you add the **.isf** to your project, see **Design Tab** on page 22. (DOC-2472) Added instructions for simulating with the Aldec Active HDL or Riviera-PRO simulators. (DOC-2463) Added description of the Transceiver Debugger's BIST function. (DOC-2469) You can now launch the Efinity RISC-V Embedded Software IDE from the Efinity GUI. (DOC-2430) Added topic on combining multiple image files at the command line. (DOC-2231) Generate Efinity Constraints Files button renamed as Generate Interface Output Files. (DOC-2296)
March 2025	15.4	Updated for patch 2024.2.294.4.15. Updated device support table for Tz100G400, Tz170G400, Ti180J484D1, and Ti135N484.
February 2025	15.3	Updated for patch 2024.2.294.3.14. Added support for Ti180 J484D1 packages. Added topic on resolving IP Manager issues. (DOC-2345)
January 2025	15.2	Updated for patch 2024.2.294.2.12. Added bitstream support for Tz200 and Tz325 FPGAs in C529 packages. Added bitstream support for Ti165 and Ti240 FPGAs in C529 packages. The HyperRam Controller IP Core no longer supports Trion FPGAs. (DOC-2312) Updated instructions for installing Linux USB drivers. (DOC-2279)
December 2024	15.1	Updated for patch 2024.2.294.1.19. Added N484 package for Ti85 and Ti135. Added C529 package for Tz200 and Tz325. Added bitstream support for Ti375 in N484 package and Tz110 and Tz170 in J361 and J484 packages.

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 14. (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 32. 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 frorm 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 13. (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.