



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

---

UG-EFN-SOFTWARE-v10.0

August 2022

[www.efinixinc.com](http://www.efinixinc.com)



# Contents

Figures.....	v
Tables.....	vii
<b>Introduction.....</b>	<b>viii</b>
New in v2022.1.....	ix
Upgrading from Older Versions.....	ix
Where to Learn More.....	ix
<b>Hardware and Software Requirements.....</b>	<b>x</b>
<b>Chapter 1: Setting Up.....</b>	<b>12</b>
Efinity Quick Start.....	12
Setting General Tool Preferences.....	13
Auto-Load Place-and-Route Data.....	13
Efinity Main Window.....	14
<b>Chapter 2: Managing Projects.....</b>	<b>15</b>
Project Editor.....	15
Project Pane.....	17
Migrating a Project to another FPGA.....	17
Using VHDL Libraries.....	18
<b>Chapter 3: Running the Tool Flow.....</b>	<b>20</b>
Run the Flow with the Dashboard Controls.....	20
Run the Flow from the Command Line.....	21
About Efinity® Synthesis.....	21
Netlist Pane.....	22
Viewing Messages and Logs.....	22
Result Pane.....	23
Viewing Place-and-Route Results.....	24
<b>Chapter 4: Using the IP Manager.....</b>	<b>26</b>
Supported IP Cores.....	27
Supported IP Cores by Family.....	28
Using the IP Configuration Wizard.....	30
Generated Files.....	31
Instantiating IP in Your Project.....	31
Managing IP in Your Project.....	31
IP Settings File.....	33
Getting Updated IP.....	33
<b>Chapter 5: Constraining Logic and Assigning Pins.....</b>	<b>34</b>
About the Interface Designer.....	35
Get Oriented.....	35
Using the Resource Assigner.....	38
Resource View.....	38
Importing and Exporting Assignments.....	39
Interface Scripting File.....	40
.csv File for GPIO Blocks.....	41
Scripting an Interface Design.....	41
Viewing the Package Pinout.....	43
Selecting a Pin.....	44
Browsing for Pins.....	45
Constraining Logic Manually (Beta).....	46

Tiles.....	46
Working with Primitives.....	48
Enabling Manual Assignments.....	49
Assignment Rules.....	50
Creating a Location Assignment File.....	50
<b>Chapter 6: Analyzing Timing.....</b>	<b>52</b>
<b>Chapter 7: Simulating.....</b>	<b>53</b>
Changing the Default Testbench Names.....	54
Simulate with the iVerilog Simulator.....	54
View Waveforms.....	55
Simulate with the ModelSim Simulator.....	55
Simulate with the NCSim Simulator.....	57
<b>Chapter 8: Debugging.....</b>	<b>58</b>
Profile Editor Perspective.....	59
Virtual I/O Debug Core.....	60
Logic Analyzer Debug Core.....	62
Debug Wizard.....	64
Debug Perspective: Logic Analyzer.....	66
Understanding Capture Control.....	67
Debug Perspective: Virtual I/O.....	68
Concurrent Debugging.....	68
Resource Usage.....	69
Disable the Debug Core.....	69
<b>Chapter 9: Configuring an FPGA.....</b>	<b>70</b>
FPGA Configuration Modes.....	70
Flash Programming Modes.....	70
About the Programmer GUI.....	72
Edit the SPI Active Clock.....	74
Generate a Bitstream (Programming) File.....	74
Working with Bitstreams.....	75
Edit the Bitstream Header.....	75
Bitstream Compression.....	75
Export to Raw Binary Format.....	75
Export to .svf Format.....	76
Convert to Intel Hex Format at the Command Line.....	76
Combine Bitstreams and Other Files.....	77
SPI Programming.....	77
Program a Single Image.....	77
Program Multiple Images (CBSEL).....	77
Program Multiple Images (Internal Reconfiguration).....	78
Program Multiple Images (Bitstream and Data).....	79
Program a Daisy Chain.....	79
JTAG Programming.....	80
JTAG Device IDs.....	80
Program a Single Image.....	80
Program Using a JTAG Chain.....	80
Program using a JTAG Bridge.....	82
JTAG Programming with FTDI Chip Hardware.....	82
FTDI Programming at the Command Line.....	82
Using the Command-Line Programmer.....	85
Project-Based Programming Options.....	86
Configuration Status Register.....	88
Verifying Configuration with the Programmer.....	90
Securing Titanium Bitstreams.....	90

Using the Efinity Bitstream Security Key Generator.....	92
Blowing Fuses with the SVF Player.....	94
Enabling Security for Your Project.....	96
Workflow for Using Security Features.....	96
Verifying Security Settings.....	98
<b>Chapter 10: Working with JTAG .svf Files.....</b>	<b>99</b>
Using the Efinity SVF Player.....	99
<b>Chapter 11: Working with Remote Hardware.....</b>	<b>101</b>
<b>Appendix: Installing USB Drivers.....</b>	<b>103</b>
Installing the Linux USB Driver.....	103
Installing the Windows USB Driver.....	104
<b>Appendix: Connecting Programming Hardware.....</b>	<b>105</b>
SPI Programming Connections.....	105
JTAG Programming Connections (Trion FPGAs).....	106
JTAG Programming Connections (Titanium FPGAs).....	108
<b>Appendix: Efinity Project Files.....</b>	<b>110</b>
Efinity Source Files for Version Control.....	110
Bitstream Generation.....	110
Debugger.....	111
Interface Designer.....	112
IP.....	115
Placement.....	116
Project.....	117
Routing.....	117
Synthesis.....	119
<b>Appendix: Shortcuts.....</b>	<b>120</b>
<b>Appendix: Icon List.....</b>	<b>121</b>
<b>Revision History.....</b>	<b>124</b>

# Figures

Figure 1: Design Flow Overview.....	viii
Figure 2: General Tool Preferences.....	13
Figure 3: Efinity Main Window.....	14
Figure 4: Dashboard Controls.....	20
Figure 5: Using the Netlist Pane.....	22
Figure 6: Using the Result Pane.....	23
Figure 7: Floorplan Editor.....	25
Figure 8: Project Pane > IP Folder Context-Sensitive Menu.....	32
Figure 9: Conceptual View of Interface Blocks.....	35
Figure 10: Interface Designer.....	36
Figure 11: Resource Assigner.....	37
Figure 12: Resource View.....	39
Figure 13: Package Planner.....	43
Figure 14: Selected Pin.....	44
Figure 15: Pin Quick View.....	44
Figure 16: Browsing for Pins.....	45
Figure 17: Tiles in the Floorplan Editor.....	47
Figure 18: Behavioral Simulation Example .do Macro.....	56
Figure 19: Post-Synthesis Simulation Example .do Macro.....	56
Figure 20: Debugger Profile Editor Perspective.....	59
Figure 21: Virtual I/O Core Block Diagram.....	60
Figure 22: Logic Analyzer Core Block Diagram.....	62
Figure 23: Debug Perspective GUI - Logic Analyzer.....	66
Figure 24: Virtual I/O Debugger.....	68
Figure 25: Flash Programming Board Setup.....	71
Figure 26: Programmer.....	72
Figure 27: Setting Programming Options (Trion).....	87
Figure 28: Setting Programming Options (Titanium).....	88
Figure 29: Bitstream Authentication.....	91
Figure 30: Bitstream Encryption.....	92
Figure 31: Disabling JTAG.....	92

Figure 32: Efinity Bitstream Security Key Generator.....	93
Figure 33: SVF Player.....	95
Figure 34: Advanced Device Configuration Status Security Signals.....	98
Figure 35: SVF Player.....	100
Figure 36: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Trion FPGAs).....	105
Figure 37: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Titanium FPGAs).....	105
Figure 38: Connect FT2232 Mini Module to JTAG Pins plus CRESET_N and SS_N.....	107
Figure 39: Connect FT4232 Mini Module to JTAG Pins plus CRESET_N and SS_N.....	107
Figure 40: Connect FT2232 Mini Module to JTAG Pins.....	108
Figure 41: Connect FT4232 Mini-Module to JTAG Pins.....	108
Figure 42: Connect FT2232 Mini-Module to JTAG Pins.....	109
Figure 43: Connect FT4232 Mini-Module to JTAG Pins.....	109

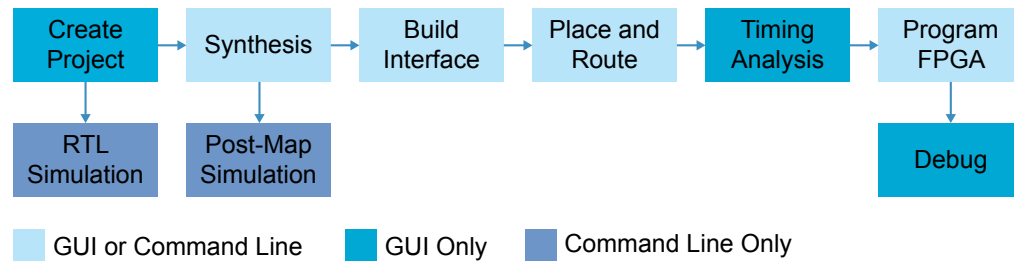
# Tables

Table 1: Titanium FPGAs Supported in Efinity Software v2022.1 Refer to the release notes on the web site for the latest support. Software patches often enable new device support.....	viii
Table 2: Trion FPGAs Supported in Efinity Software v2022.1 .....	viii
Table 3: Machine Memory Requirements.....	x
Table 4: Linux Operating System.....	xi
Table 5: Synthesis Project Settings.....	16
Table 6: Compilation Files and Reports.....	23
Table 7: Interface Designer Files.....	24
Table 8: IP Cores Supported in the IP Manager.....	27
Table 9: IP Cores Supported by Family.....	29
Table 10: End of Life IP Cores by Family.....	30
Table 11: Example GPIO .csv File.....	41
Table 12: FPGA Tile Types.....	46
Table 13: Mapping Trion Primitives to Tiles and Sub-Blocks.....	48
Table 14: Mapping Titanium Primitives to Tiles and Sub-Blocks.....	48
Table 15: Mapping Primitives to Tiles.....	49
Table 16: Virtual I/O Core Ports.....	60
Table 17: Logic Analyzer Core Ports.....	62
Table 18: FPGA Configuration Modes.....	70
Table 19: Flash Programming Modes.....	71
Table 20: Internal Oscillator Clock Settings.....	74
Table 21: Modes when Combining Images.....	77
Table 22: Titanium JTAG Device IDs.....	80
Table 23: Trion JTAG Device IDs.....	80
Table 24: Project-Specific Programming Options.....	86
Table 25: Configuration Status Register.....	89
Table 26: Monitoring with the Efinity Programmer.....	90
Table 27: Efinity Tools Used for Securing Bitstreams.....	91
Table 28: Project Options for Security.....	96
Table 29: USB Programming Connections.....	103
Table 30: Shortcuts.....	120
Table 31: Document Revision History.....	124

# Introduction

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

**Figure 1: Design Flow Overview**



**Table 1: Titanium FPGAs Supported in Efinity® Software v2022.1<sup>(1)</sup>**

FPGA	Package	Bitstream	Timing	Pinout
Ti35	F100S3F2, F225	✓	Preliminary	Final
Ti60	W64	✓	Preliminary	Final
	F100S3F2, F225	✓	Preliminary	Final
Ti90	M361, M484, F529	-	Preliminary	Preliminary
Ti120	M361, M484, F529	-	Preliminary	Preliminary
Ti180	M361, L484	✓	Preliminary	Preliminary
	M484, F529	-	Preliminary	Preliminary

**Table 2: Trion FPGAs Supported in Efinity® Software v2022.1**

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

<sup>(1)</sup> Refer to the release notes on the web site for the latest support. Software patches often enable new device support.



## New in v2022.1

The Efinity® software v2022.1 has the following new features and enhancements:

- Titanium memory and runtime improvements:
  - Uses 3.3x less memory
  - Place-and-route runtime is improved by about 2x.
  - Router runtime is between 5% and 25% better depending on the calculation used (arithmetic vs. geometric).
- Trion runtime improved by about 1.6x.
- Bitstream support for Titanium Ti180 FPGAs in the M361 package.
- New Package Planner tool lets you graphically select and browse the package pins.
- Concurrent Debug feature supports using multiple TAPs in one design.
- Fast active programming option to reduce the programming time.
- Location constraint feature (beta) that lets you specify the specific location (tile) of logic in your design.
- This release requires updated libraries for Windows and Linux. See [Hardware and Software Requirements](#) on page x.

## Upgrading from Older Versions

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 this version of Efinity software. If the old files are not compatible, the software will prompt you to re-compile.

## Where to Learn More

The Efinity® software includes documentation as PDF user guides and on-line HTML help. This documentation is provided with the software. You can also access the latest versions of PDF documentation in the Support Center:

- [Efinity Software User Guide](#)
- [Efinity Synthesis User Guide](#)
- [Efinity Timing Closure User Guide](#)
- [Efinity Software Installation User Guide](#)
- [Efinity Trion Tutorial](#)
- [Efinity Debugger Tutorial](#)
- [Titanium Interfaces User Guide](#)
- [Trion Interfaces User Guide](#)
- [Efinity Interface Designer Python API](#)
- [Quantum Trion Primitives User Guide](#)
- [Quantum Titanium Primitives User Guide](#)

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

- Computer with a minimum 32-bit or maximum 64-bit operating system.
  - A 32-bit Windows system is required for the Efinity standalone programmer.
  - A 64-bit Windows system is required for using the security tools in the Efinity standalone programmer.
- Machine memory requirements (when running Efinity design compilations):

**Table 3: Machine Memory Requirements**

Product	Model	Memory
Trion	T4, T8, T13, T20, T35	16 GB
	T55, T85, T120	32 GB
Titanium	Ti35, Ti60	16 GB
	Ti90, Ti120, Ti180	32 GB

- Your preferred text editor such as Notepad, gVim, Visual Studio

## Windows Requirements

- Windows 8.1 or later, 64 bit operating system
- Microsoft Visual C++ 2019 x64 runtime library (or latest version) redistributable  
<https://docs.microsoft.com/en-us/cpp/windows/redistributing-visual-cpp-files?view=msvc-170>





**Note:** If you want to use the stand-alone Programmer, you also need to install the x86 and x64 libraries (64-bit systems) or x86 library (32-bit systems).

- Zadig software to install USB drivers  
see **Installing the Windows USB Driver** on page 104
- Open-source Java 64-bit runtime environment; required for configuring the Sapphire RISC-V SoC and DMA Controller in the IP Manager; available from:
  - <https://www.java.com/en/download/manual.jsp> (Java 8)
  - <https://developers.redhat.com/products/openjdk/download> (OpenJDK 8 or 11)
  - <http://jdk.java.net/16/> (OpenJDK 16)

## Linux Requirements

Table 4: Linux Operating System

Operating System	Note
Ubuntu v18.04 or later	You may need to install some graphics related libraries before running the Efinity tools. An example list of libraries (not comprehensive) could be: <b>apt install libxcb-xinerama0 libxcb-icccm4 libxcb-image0 libxcb-keysyms1 libxcb-render-util0</b>
Red Hat Enterprise x86-64 v7.4 or later	<p>You may need to install some graphics related libraries before running the Efinity tools. An example list of libraries (not comprehensive) could be: <b>yum install libxkbcommon-x11 xcb-util-renderutil xcb-util-keysyms xcb-util-image xcb-util-wm xcb-util</b></p> <p> <b>Important:</b> If GUI is not working, set an environment variable, then launch the tool again using the following commands (applicable for both Ubuntu and Red Hat Enterprise Linux):</p> <pre>source/path/to/efinity/&lt;version&gt;/bin/setup.sh export QT_DEBUG_PLUGINS=1 efinity</pre> <p>The Qt verbose command line output can provide clues about which system libraries are missing and need to be installed via apt/yum.</p> <p> <b>Note:</b> Red Hat Enterprise x86-64 v9.0 has not yet been tested.</p>

- Linux X11 windowing system (for Efinity® GUI)
- Udev device manager for Efinix USB programming cable  
see [Installing the Linux USB Driver](#) on page 103

### 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.

For more information on the simulator: [iverilog.icarus.com/page](http://iverilog.icarus.com/page)

To download the simulator: [bleyer.org/icarus](http://bleyer.org/icarus)

To download the simulator source code: [github.com/steveicarus/iverilog](https://github.com/steveicarus/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.

Download and install the software from [gtkwave.sourceforge.net](http://gtkwave.sourceforge.net).



**Note:** Linux users can use the following commands:

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

# Setting Up

## Contents:

- **Efinity Quick Start**
- **Setting General Tool Preferences**
- **Efinity Main Window**

## 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.

### 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
```

### 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
```

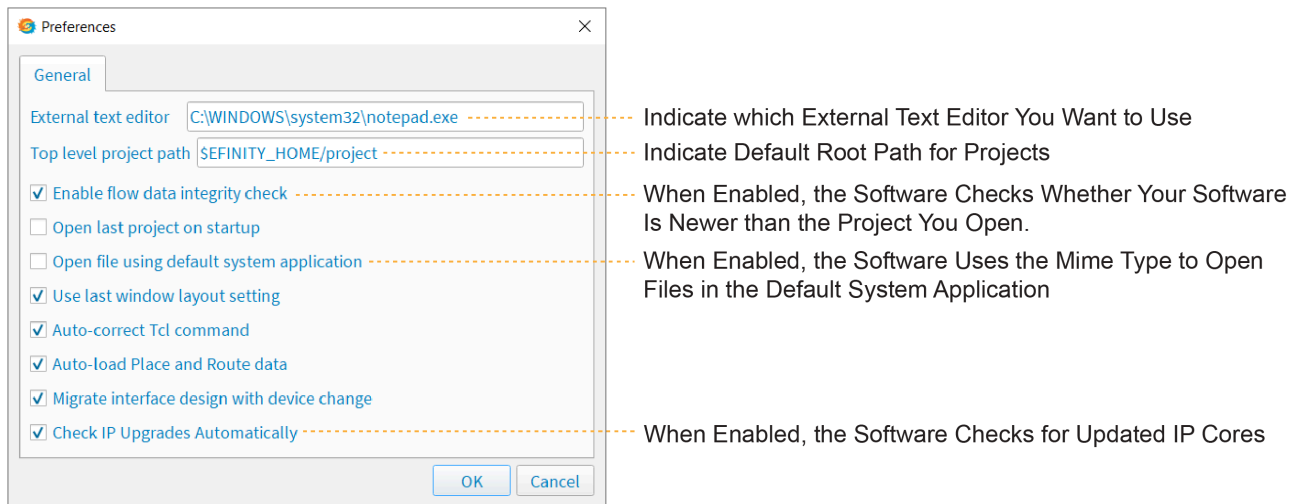
For command-line help:

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

## 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.

### Auto-Load Place-and-Route Data

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

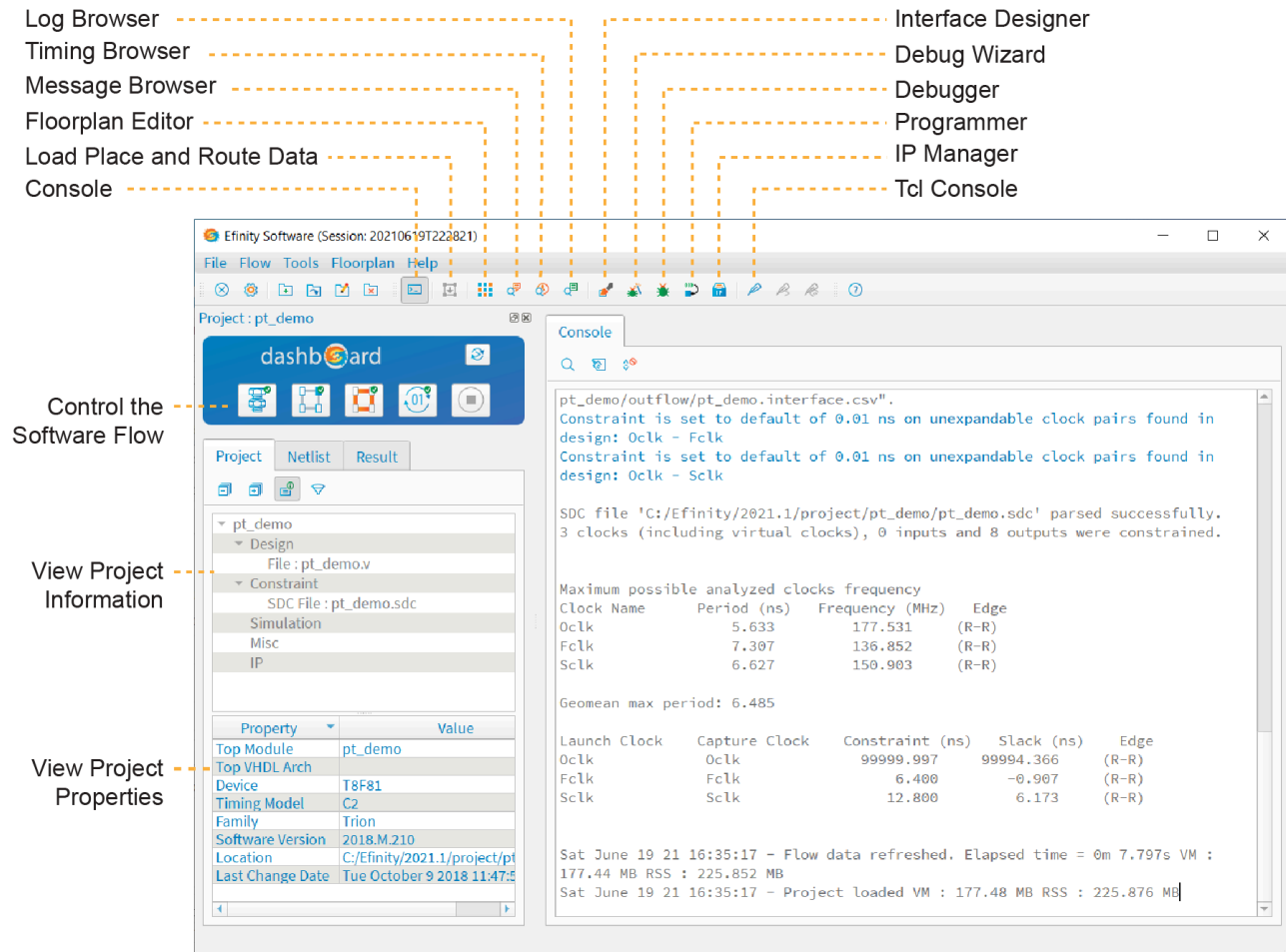
If you have disabled auto-loading, you cannot view place-and-route results in the Floorplan Editor or Timing Browser, or use the Tcl console. To enable these tools, click the **Load Place and Route Data** button in the main window.

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

# Efinity Main Window

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

**Figure 3: Efinity Main Window**



# Managing Projects

## Contents:

- [Project Editor](#)
- [Project Pane](#)
- [Migrating a Project to another FPGA](#)
- [Using VHDL Libraries](#)

## Project Editor

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

### Project Tab

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

### Design Tab

Use the Design tab to add design and constraint files to your project. You can also indicate the design's top-level module; if you do not specify one, the tool attempts to determine it automatically.

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 18 for details.



**Learn more:** For more information about language support, refer to the [Efinity Synthesis User Guide](#).

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

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

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



**Learn more:** Refer to [Analyzing Timing](#) on page 52 and the [Efinity Timing Closure User Guide](#) for more information on timing analysis.

## Synthesis Tab

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

**Table 5: Synthesis Project Settings**

Setting	Description
<b>Work Directory</b>	Specify a custom directory or use the default ( <b>work_syn</b> ).
<b>Generate post synthesis netlist</b>	Choose whether the software should create this netlist. Default: On
<b>Synthesis Options</b>	See "Synthesis Options" in the <a href="#">Efinity Synthesis User Guide</a> .
<b>Include Dir</b>	Specify directories to include in your project. If you use the IP Manager to add IP, the <b>ip/&lt;module&gt;</b> directory is listed here. The software searches these locations when you use include statements.
<b>Dynamic Parameter</b>	Use this area to add parameters and values that apply to the top-level module or entity in your project.
<b>Verilog `define Macro</b>	<p>Use this area to add `define macros to your project.</p> <p>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 <b>NAME</b> field and 1 in the <b>Value</b> field. Then if you want to include simulation only code, use this format:</p> <pre> `ifndef SYNTHESIS \$display(...) ... some other simulation directives ... `endif </pre> <p>You can also use the translate_on and translate_off directives to accomplish similar functionality.</p>



**Learn more:** Refer to the [Efinity Synthesis User Guide](#) for more information on these options.

## Place and Route Tab

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

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

Often, the default settings are the best choice, as these options will not help all designs.

- The placer effort level (`--placer_effort_level`) is a way to control how much runtime the placer uses when it tries to improve placement quality.
- The number of threads (`--max-threads`) controls how many thread that the placer can launch. The default setting (-1) means that the placer uses the maximum number of available processors.
- The `--seed` option introduces random noise in the placer. The seed is the value you set.





**Learn more:** Refer to "Place and Route Options" in the [Efinity Timing Closure User Guide](#) for more details on these options and how to optimize timing.

### Bitstream Generation Tab

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



**Learn more:** Refer to [Project-Based Programming Options](#) on page 86 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.



**Learn more:** Refer to [Debugging](#) on page 58 for more information.

## Project Pane

The Project pane, which is located under the Dashboard, shows all of the files in your design. Double-click a filename in the Project pane 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. In Efinity® v2020.2 and higher, you can also choose to show the file in it's containing folder.

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

You can right-click the folders in the Project pane 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 31 for more information on the IP context menus.

## 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.

## 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.

## 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**.
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;
```

# Running the Tool Flow

## Contents:

- **Run the Flow with the Dashboard Controls**
- **Run the Flow from the Command Line**
- **About Efinity Synthesis**
- **Netlist Pane**
- **Viewing Messages and Logs**
- **Result Pane**
- **Viewing Place-and-Route Results**

The Efinity software supports GUI and command line tool flows.

## Run the Flow with the Dashboard Controls

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

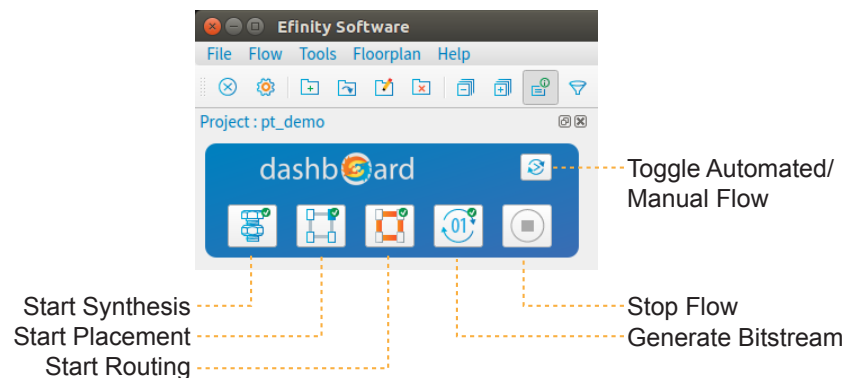


**Note:** The automated flow is on by default.

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

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

*Figure 4: 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

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 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<sup>®</sup> 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.

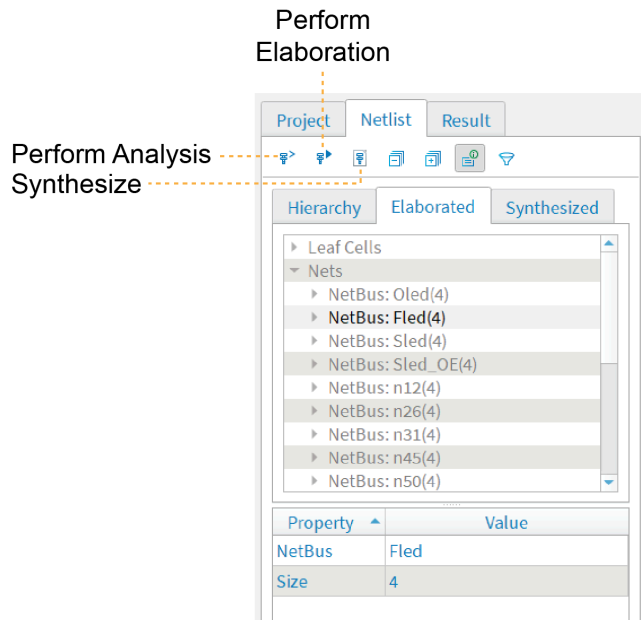
---

## Netlist Pane

The Netlist pane, 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 pane to open a context-sensitive menu with shortcut actions.

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

*Figure 5: Using the Netlist Pane*



## 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.

### 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 52 for more details.

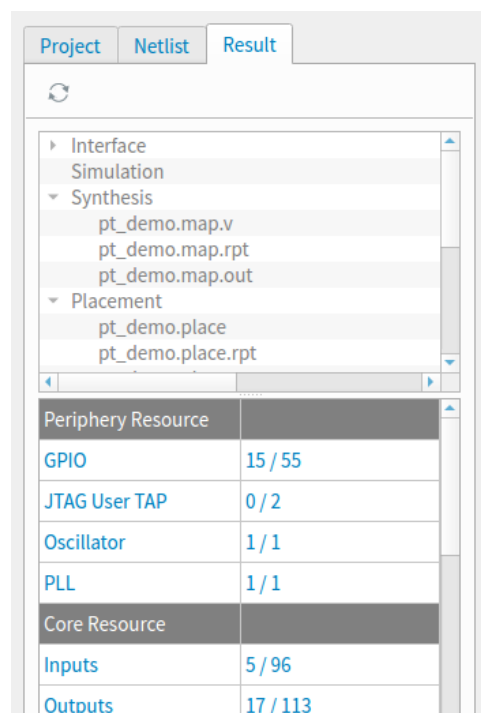
## Log Browser

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

## Result Pane

The Result pane, 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 pane to open a context-sensitive menu with shortcut actions.

*Figure 6: Using the Result Pane*



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

*Table 6: 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.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.

Category	File	Description
	<b>&lt;project&gt;.res.csv</b>	Provides the resource usage for all of the modules in the design.
<b>Placement</b>	<b>&lt;project&gt;.place</b>	Detailed placement report.
	<b>&lt;project&gt;.place.rpt</b>	Resource summary report.
	<b>&lt;project&gt;.place.out</b>	Messages output to the Console during placement.
<b>Routing</b>	<b>&lt;project&gt;.pnr.rpt</b>	Provides the resource summary for inputs, outputs, clocks, LEs, memory, and multipliers (Trion) or DSP Blocks (Titanium).
	<b>&lt;project&gt;.route.rpt</b>	Routing report.
	<b>&lt;project&gt;.timing.rpt</b>	Static timing analysis report.
	<b>&lt;project&gt;.route.out</b>	Messages output to the Console during routing.
<b>Bitstream</b>	<b>&lt;project&gt;.hex</b>	Use this file to program in SPI active or passive mode.
	<b>&lt;project&gt;.bit</b>	Use this file for JTAG programming.
	<b>&lt;project&gt;.pgm.out</b>	Messages output to the Console during bitstream generation.

**Table 7: Interface Designer Files**

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

File	Description
<b>&lt;project&gt;.interface.csv</b>	Constrains the FPGA design pins used in the interface between the core and the periphery.
<b>&lt;project&gt;.pt.rpt</b>	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.
<b>&lt;project&gt;.pinout.rpt</b>	Has the board design pinout with pin number, signal name, pin name, I/O bank, etc. in a nicely formatted text file format.
<b>&lt;project&gt;.pinout.csv</b>	Pinout report file formatted as <b>.csv</b> .
<b>&lt;project&gt;.pt_timing.rpt</b>	Timing report for the Trion® and Titanium interface logic.
<b>&lt;project&gt;.pt.sdc</b>	Template SDC file to constrain the FPGA design pins based on the interface configuration.
<b>&lt;project&gt;_or.ini</b>	Contains option register information the Programmer uses.
<b>&lt;project&gt;_template.v</b>	Template Verilog HDL file defining the FPGA design pins based on the interface configuration.

## 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.

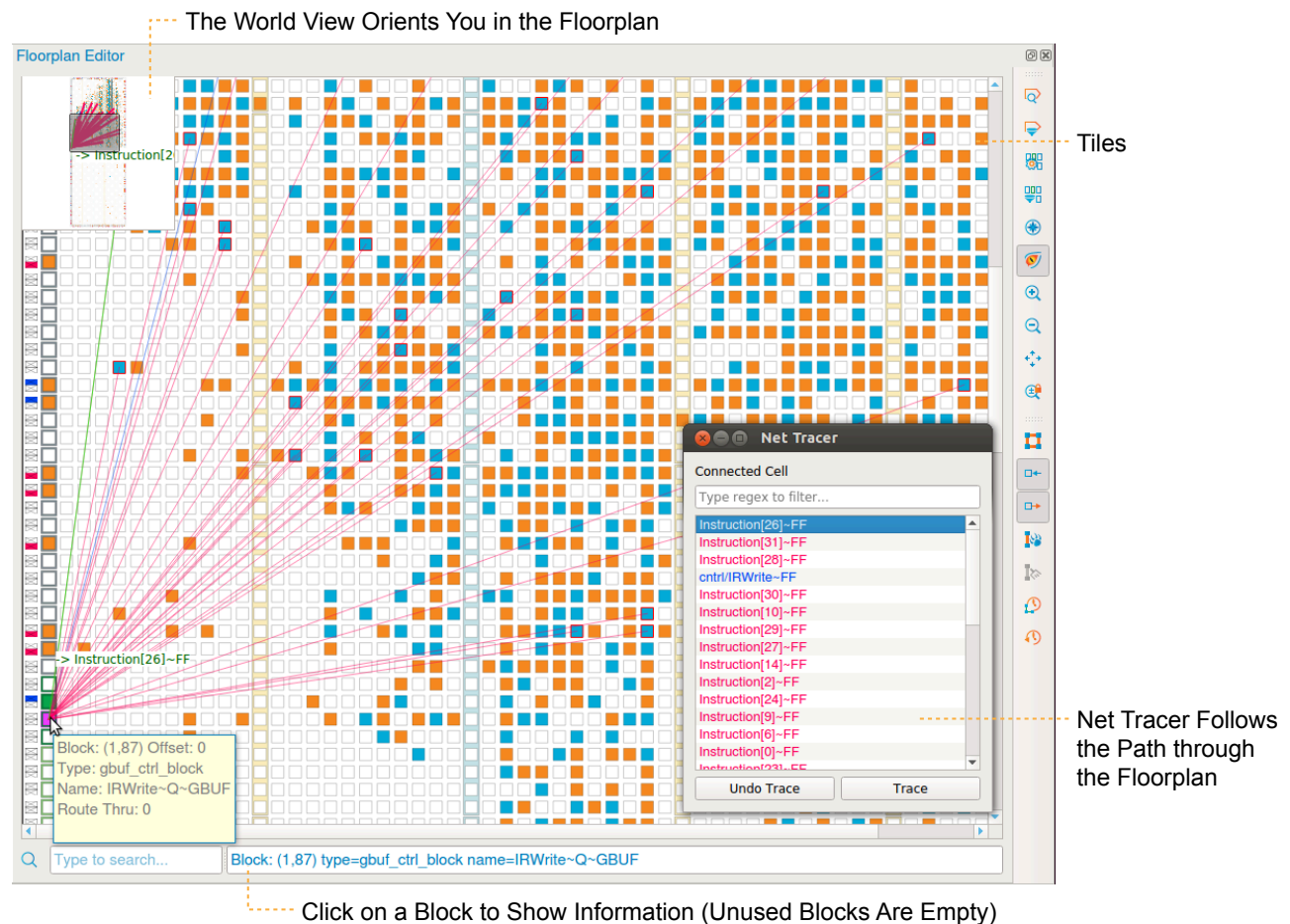


**Note:** Double-click on a file name to open it in the Efinity Code Editor.

- 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 7: Floorplan Editor**



**Note:** If you have disabled auto-loading, you cannot view place-and-route results in the Floorplan Editor or Timing Browser, or use the Tcl console. To enable these tools, click the **Load Place and Route Data** button in the main window. Refer to [Auto-Load Place-and-Route Data](#) on page 13 for details.



**Learn more:** Refer to the [Efinity Trion Tutorial](#) for more information on how to use the Floorplan Editor.

# Using the IP Manager

## Contents:

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

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



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

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

The IP Manager consists of:

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

## Supported IP Cores

The Efinity® software version 2020.2 and higher includes IP cores. These cores have been designed to work with the Efinity® version they are shipped with.



**Note:** Efinix sometimes also releases IP cores as downloads in the Support Center; those cores are not listed in this table.

*Table 8: IP Cores Supported in the IP Manager*

IP Core	IP Version			
	2020.2	2021.1	2021.2	2022.1
<b>Arithmetic</b>				
CORDIC			1.0 (In Efinity patch v2021.2.323.2.18)	1.0
Divider	1.0	1.2	2.0	2.2
Integer Square Root	1.1	1.2	1.2	1.3
<b>Bridges and Adaptors</b>				
ABP3 to AXI4 Lite Converter	1.0	1.1	1.1	1.2
AXI Interconnect				1.0
AXI4-Stream Switch				1.0
Direct Memory Access	-	1.0	1.1	1.2
<b>Ethernet</b>				
Triple Speed Ethernet MAC	1.3	2.0	2.2	2.3
<b>Memory</b>				
FIFO (Legacy)	2.3	2.5	-	-
FIFO	-	-	1.0	1.1
<b>Memory Controllers</b>				
ASMI SPI Flash Controller Core	1.1	2.0	2.0	2.1
DDR Hard Memory Controller-Calibration	2.0	2.0	2.0	-
DDR Hard Memory Controller-Calibration and Reset				1.0
DDR Hard Memory Controller-Reset	1.0	1.0	1.0	1.1
HyperRAM Controller	-	1.0	2.0	2.1

IP Core	IP Version			
	2020.2	2021.1	2021.2	2022.1
JTAG SPI Flash Loader	1.2	2.1	3.0	3.1
SDRAM Controller	1.4	1.4	1.5	1.6
SD Host Controller	–	1.0	1.0	1.1
MIPI				
MIPI CSI-2 RX Controller	–	1.0	2.0	2.1
		1.0	2.0	2.1
MIPI CSI-2 TX Controller				
MIPI 2.5G CSI-2 RX Controller				1.0
MIPI 2.5G CSI-2 TX Controller				1.0
MIPI DSI TX Controller	–	–	1.0	1.1
MIPI D-PHY RX Controller	–	1.0	2.0	2.1
		1.0	2.0	2.1
		1.0	2.0	2.1
		1.0	2.0	2.1
		1.0	2.0	2.1
		1.0	2.0	2.1
MIPI D-PHY TX Controller				
MIPI D-PHY BIDIR RX Controller				
MIPI D-PHY BIDIR TX Controller				
Processors and Peripherals				
Sapphire SoC	–	1.0	2.0	2.2
Jade SoC	2.00	2.2	2.4	_(2)
Opal SoC	2.00	2.2	2.4	_(2)
Ruby SoC	2.00	2.2	2.4	_(2)
Serial Interface Protocols				
I²C	2.0	3.1	3.1	3.2
UART	1.0	1.1	1.1	1.2

## 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.

<sup>(2)</sup> The Ruby, Jade, and Opal SoCs are end of life and not supported in the Efinity software v2022.1 and higher. Instead use the Sapphire SoC.

Table 9: IP Cores Supported by Family

IP Core	Trion		Titanium	
	Supported	Not Supported	Supported	Not Supported
<b>Arithmetic</b>				
CORDIC	All		All	
Divider	All		All	
Integer Square Root	All		All	
<b>Bridges and Adaptors</b>				
ABP3 to AXI4 Lite Converter	All		All	
AXI Interconnect	All		All	
AXI4-Stream Switch	All		All	
Direct Memory Access	All		All	
<b>Ethernet</b>				
Triple Speed Ethernet MAC	All			All
<b>Memory</b>				
FIFO (Legacy)	All		All	
FIFO	All		All	
<b>Memory Controllers</b>				
ASMI SPI Flash Controller Core	All		All	
DDR Hard Memory Controller-Calibration and Reset	T20, T35, T55, T85, T120	T4, T8, T13		All
DDR Hard Memory Controller-Reset	T20, T35, T55, T85, T120	T4, T8, T13		All
HyperRAM Controller	All		All	
JTAG SPI Flash Loader	All		All	
SDRAM Controller	All			All
SD Host Controller	All		All	
<b>MIPI</b>				
MIPI CSI-2 RX Controller MIPI CSI-2 TX Controller		All	All	
MIPI 2.5G CSI-2 RX Controller MIPI 2.5G CSI-2 TX Controller		All	Ti90, Ti120, Ti180	Ti35, Ti60
MIPI DSI TX Controller		All	All	
MIPI D-PHY RX Controller MIPI D-PHY TX Controller MIPI D-PHY BIDIR RX Controller MIPI D-PHY BIDIR TX Controller		All	All	
<b>Processors and Peripherals</b>				
Sapphire SoC	All except T4	T4	All	
<b>Serial Interface Protocols</b>				
I <sup>2</sup> C	All		All	

IP Core	Trion		Titanium	
	Supported	Not Supported	Supported	Not Supported
UART	All		All	

Table 10: End of Life IP Cores by Family

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

## 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>\_tpl.v**—Verilog HDL instantiation template.
- **<module name>\_tpl.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.

## Instantiating IP in Your Project

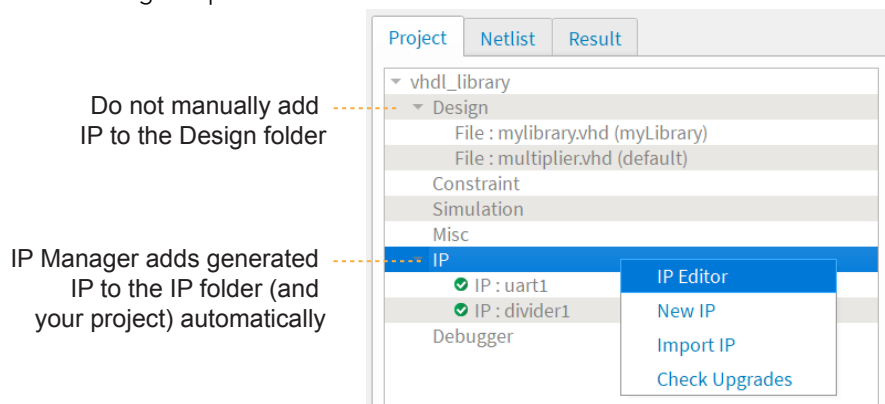
The IP Manager creates these template files in the *<project>/ip/<module name>* directory:

- *<module name>.v\_tpl.v* is the Verilog HDL module.
- *<module name>.v\_tpl.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 33 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 13.




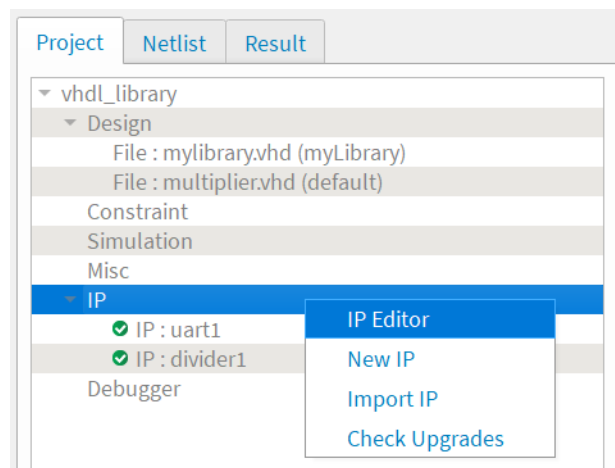
Icon	Meaning
	The IP core is up to date.
	There is an optional update to the IP core. Efinix recommends that you launch the IP Configuration wizard for the core to make any changes and then re-generate the deliverables.
	There is a required update to the IP core. You must launch the IP Configuration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

Figure 8: Project Pane > 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.

# Constraining Logic and Assigning Pins

## Contents:

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

The tools in the Efinity<sup>®</sup> 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<sup>®</sup> software uses the constraint file to constrain your design to the interface blocks.



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

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

The [Efinity Interface Designer Python API](#) describes how to create an interface design using scripting.

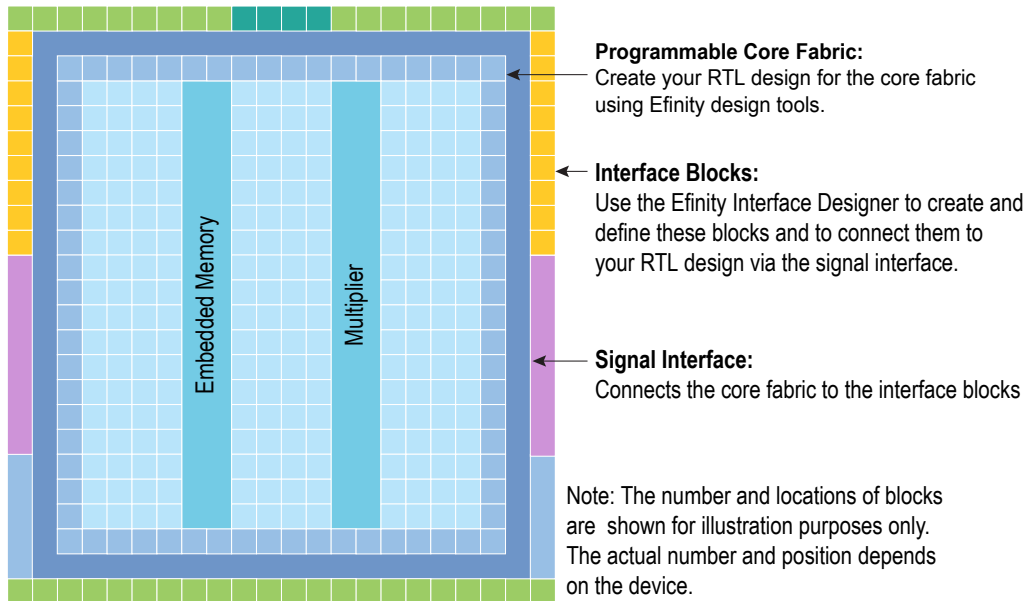
---

## About the Interface Designer

Trion® 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 9: Conceptual View of Interface Blocks**

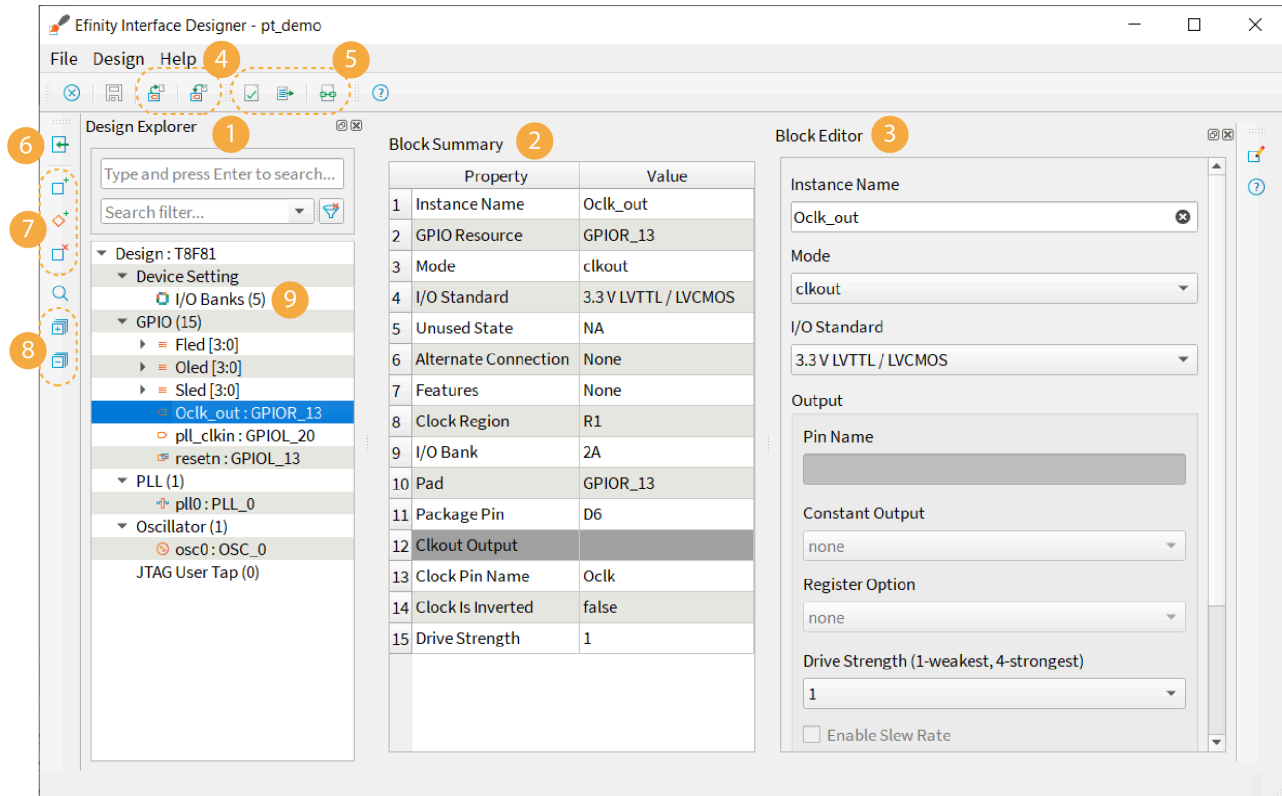


## Get Oriented

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 its 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 10: Interface Designer

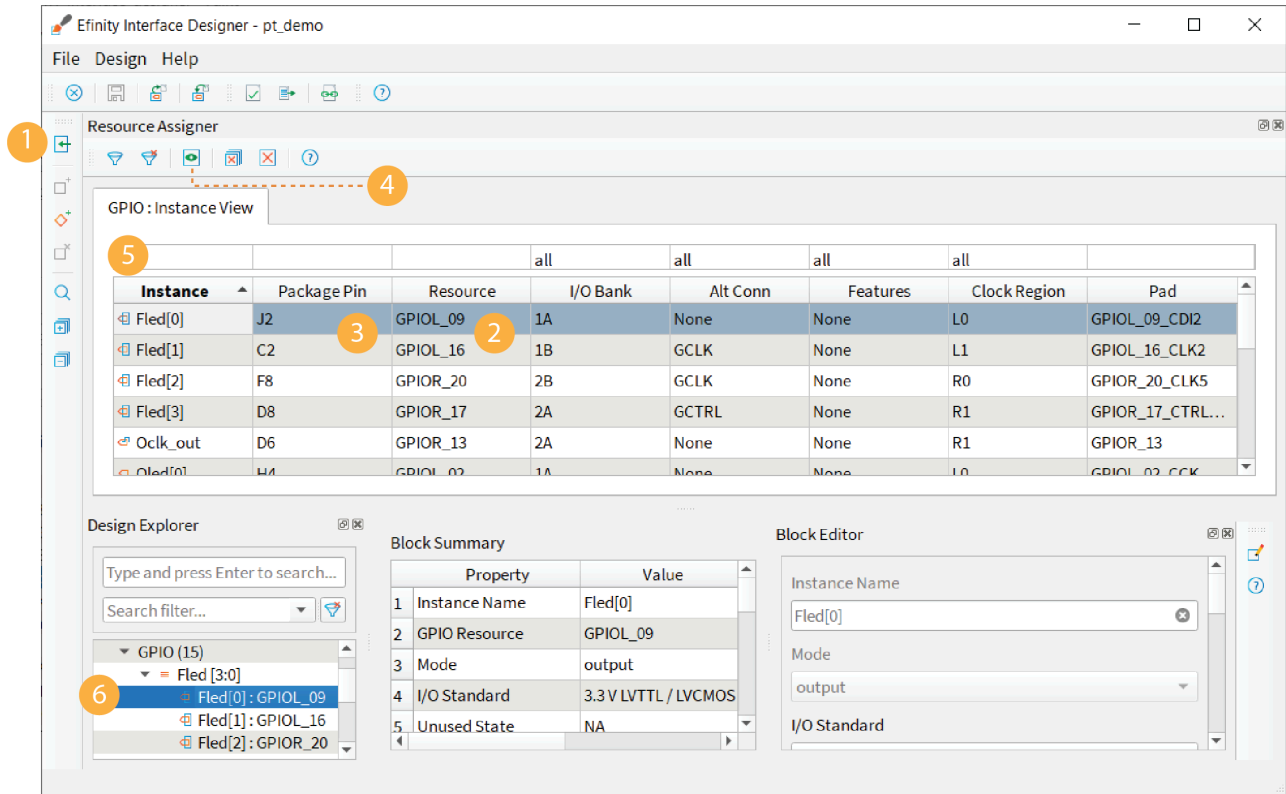


## Notes:

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.

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 11: 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.



**Important:** 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 [#unique\\_63/unique\\_63\\_Connect\\_42\\_d222e35](#)



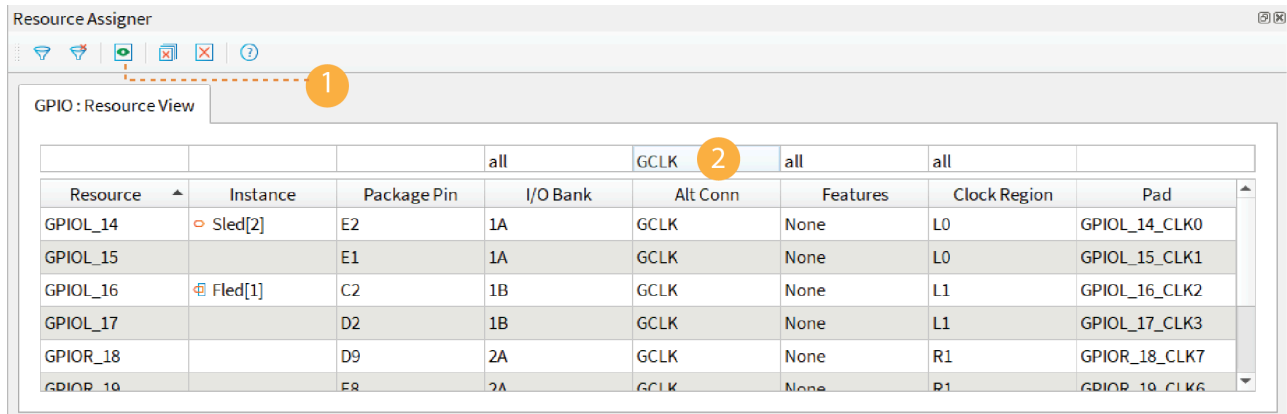
**Note:** Titanium: When using HSIO pins as GPIO, make sure to leave at least 1 pair of unassigned HSIO pins between any GPIO and HSIO pins in the same bank. This separation reduces noise. The Efinity software issues an error if you do not leave this separation.

## 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, **GCTRL**.

Figure 12: Resource View



## Importing and Exporting Assignments

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

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

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

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

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



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

## Interface Scripting File

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

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

### Example: Example Interface Scripting File

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

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

# 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_clkln","C3")
design.assign_pkg_pin("resetsn","F1")
```



## .csv File for GPIO Blocks

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

**Table 11: Example GPIO .csv File**

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

When working with the .csv file:

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



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

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

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

## Scripting an Interface Design

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.<sup>(4)</sup> 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

<sup>(3)</sup> Unassigned instances have a blank field for the Package Pin-Pad Name column.

<sup>(4)</sup> Source: [What Is Python? Executive Summary](#)

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



---

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

---



---

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

---

## Viewing the Package Pinout

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

Figure 13: Package Planner

**Package Planner - pt\_demo**

File View Help

Package View

256-ball FBGA Pinout Diagram  
Device Name: T20F256

Top View

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

A B C D E F G H J K L M N P R T

View Configuration

Report Pin information

☐ Display I/O bank group

☐ Display PLL Reference Clock

☐ Display PLL External Feedback

☐ Display Global Clock

☐ Display Global Control

Turn these options on to highlight them on the pinout image

Click to View Details for a Selected Pin

Toggle Pin Configuration

Toggle World View

Zoom In

Zoom Out

Fit Pinout to the Window

Show Package Bottom

Rotate Left

Rotate Right

Reset the Orientation

Toggle the Legend

Toggle the Pin Browser

Open Help

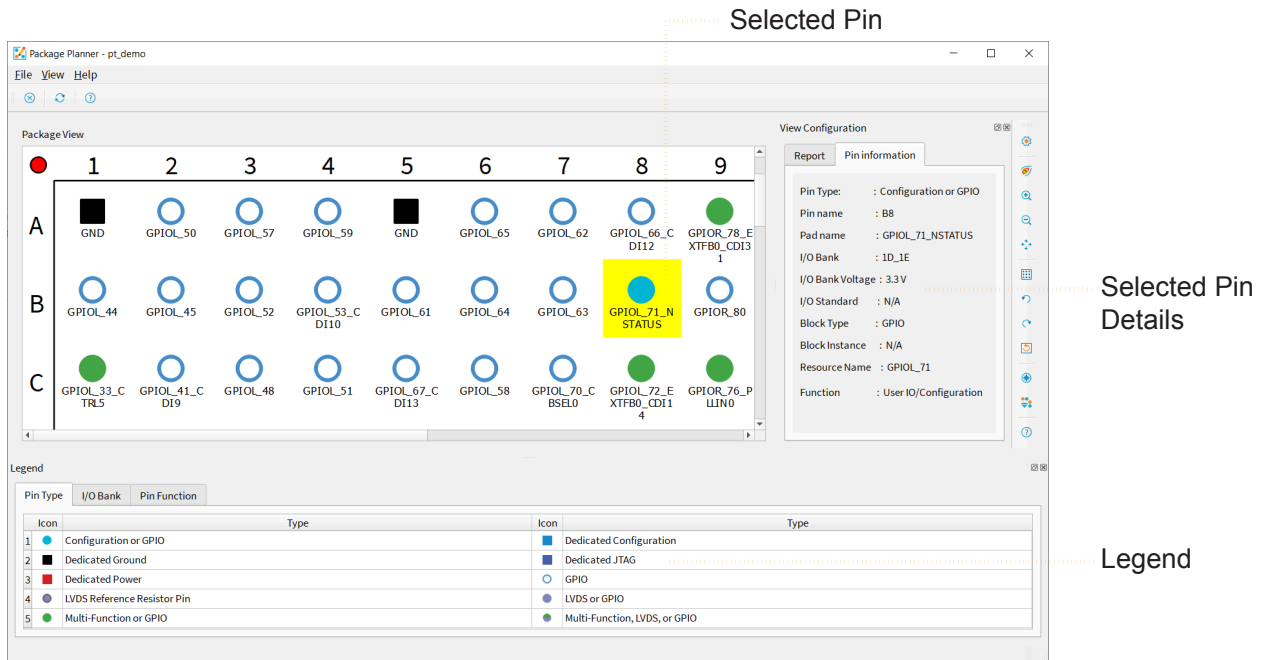
Refresh the Pinout

Indicates Pin 1

## Selecting a Pin

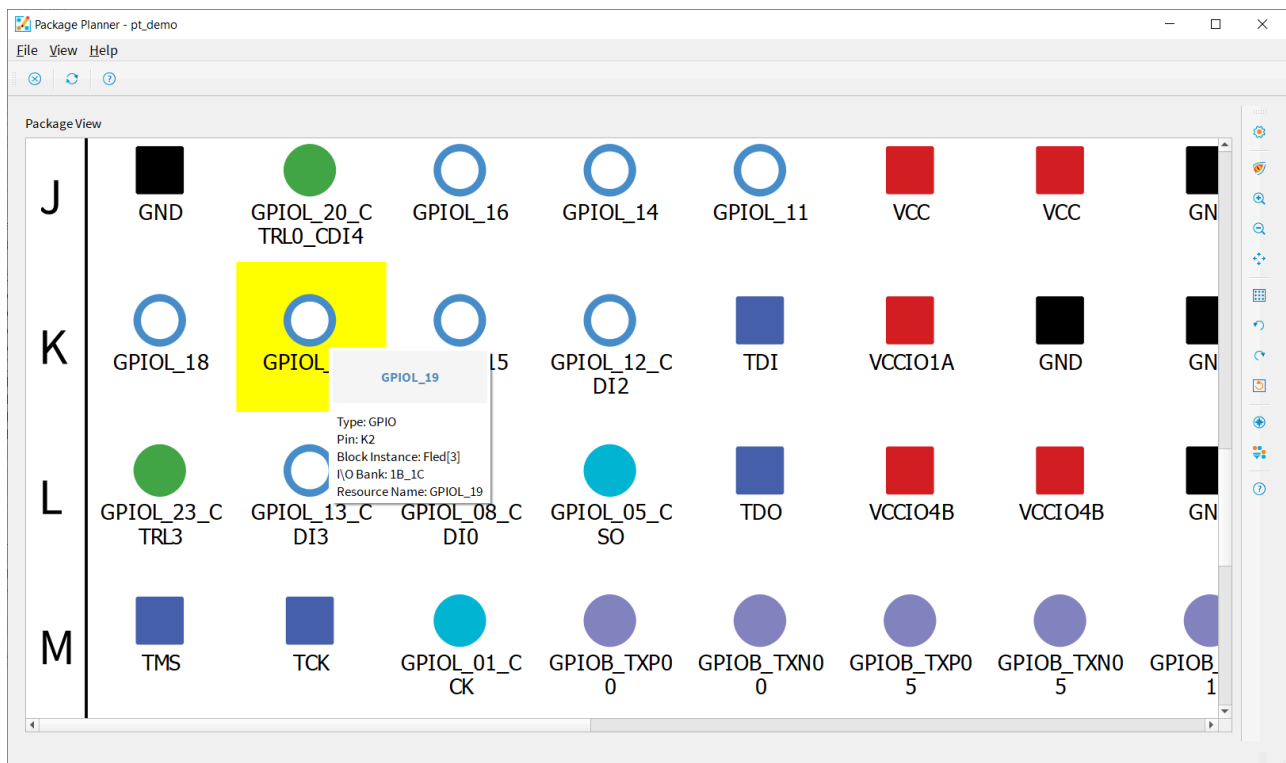
Click a pin in the pinout to highlight it. The **Pin Information** tab opens to show the details about the selected pin. Open the Legend to view the meaning of the pins' color coding.

Figure 14: Selected Pin



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

Figure 15: Pin Quick View



## Browsing for Pins

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

Figure 16: Browsing for Pins

Selected Pad Name

Filter Pins

Pin Search

Pin Name	Pad Name	Pin Type	I/O Bank	I/O Bank Voltage	I/O Standard	Block Type	Block Instance	Resource Name	Function
A2	GPIO_50	GPIO	1D_1E	3.3 V		GPIO		GPIO_50	User IO
A3	GPIO_57	GPIO	1D_1E	3.3 V		GPIO		GPIO_57	User IO
A4	GPIO_59	GPIO	1D_1E	3.3 V		GPIO		GPIO_59	User IO
A6	GPIO_65	GPIO	1D_1E	3.3 V		GPIO		GPIO_65	User IO
A7	GPIO_62	GPIO	1D_1E	3.3 V		GPIO		GPIO_62	User IO
A8	GPIO_66_C011	GPIO	1D_1E	3.3 V		GPIO		GPIO_66	User IO

Package View

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	GND	GPIO_50	GPIO_57	GPIO_59	GND	GPIO_65	GPIO_62	GPIO_66_C011	GPIO_78_E012	GPIO_79_E013	GPIO_84	GND	GPIO_95_C026	GPIO_90	GPIO_94_C027
B	GPIO_44	GPIO_45	GPIO_52	GPIO_53_C010	GPIO_61	GPIO_64	GPIO_63	GPIO_71_N014	GPIO_80	GPIO_82	GPIO_83	GPIO_93	GPIO_97	GPIO_92	GPIO_98
C	GPIO_33_C015	GPIO_41_C016	GPIO_48	GPIO_51	GPIO_67_C013	GPIO_68	GPIO_70_C014	GPIO_72_E015	GPIO_76_P016	GPIO_81_M017	GPIO_86	GPIO_85	GPIO_99	GPIO_102	GPIO_112_C023

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



**Important:** This feature is beta in the Efinity software v2022.1.

### Tiles

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

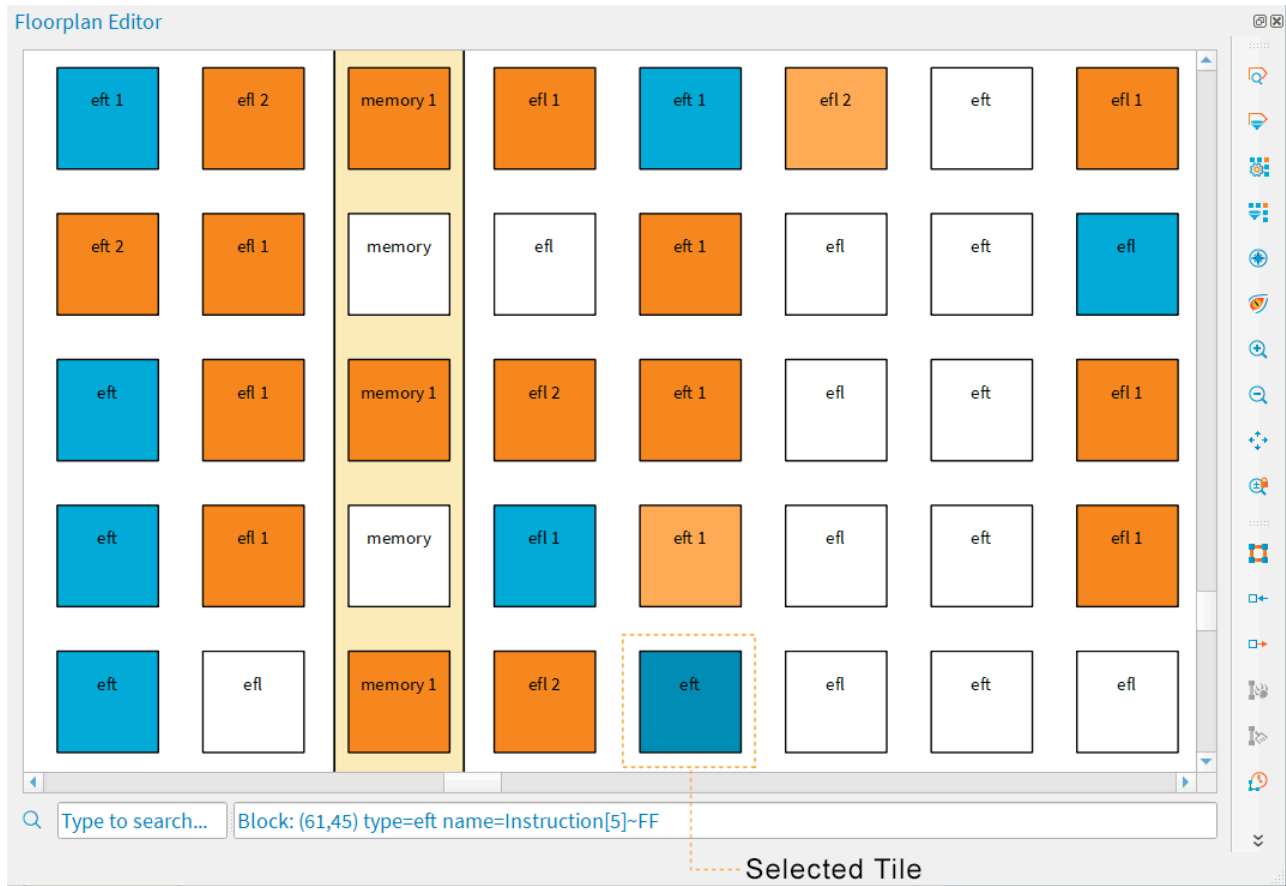
*Table 12: FPGA Tile Types*

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

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

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

Figure 17: Tiles in the Floorplan Editor



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

## Working with Primitives

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

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

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

Tile	Sub-Block			
	0	1	2	3
EFT	EFX_LUT4 EFX_ADD EFX_COMB4	-	EFX_FF	-
EFL	EFX_LUT4 EFX_ADD EFX_COMB4	-	EFX_FF	-
RAM	EFX_RAM_5K EFX_DPRAM_5K	Reserved	-	-
MULT	EFX_MULT	-	-	-

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

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



The following table shows another view of the same mappings.

**Table 15: Mapping Primitives to Tiles**

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

## Finding Primitive Cell Names

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

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

## Enabling Manual Assignments

Because manual assignments are beta in the Efinity software v2022.1, 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 primitive cells *can* be packed together, you can assign them to the same location. The sub-block index must be unique for each primitive cell in a location. For example, if you assign four EFX\_DSP12 primitives to the same tile, they must each have a different sub-block.
- The software does not pack manually assigned cells with unassigned cells. For example, if you place a EFX\_DSP12 into a DSP tile at sub-block 0 and do not assign any other sub-blocks, the software will not pack any other DSP logic into that tile, leaving sub-blocks 1, 2, and 3 empty. Similarly, only assigning flipflops (which use sub-block 2) uses more overall resources because sub-block 0 is left empty.



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

### Flipflops

- An EFX\_FF can be packed alone or with its driver cell (EFX\_LUT4, EFX\_SRL8, EFX\_ADD, or EFX\_COMB4).
- An EFX\_FF can only be packed with an EFX\_SRL8 if they share CE and CLK inputs and if the EFX\_FF does not have an inverted input.
- An EFX\_FF cannot be packed if it has an inverted input connected to a multi-fanout net.

### RAM, Multiplier, and DSP

- EFX\_MULT, EFX\_DSP48, and all RAM primitives cannot share a tile with any other cells.
- Two EFX\_DSP24 primitives or up to four EFX\_DSP12 primitives **not** connected by CASCIN/CASCOUT signals can be packed together and share a location.

### Chains

EFX\_DSP48, EFX\_DSP24, EFX\_DSP12, EFX\_ADD, and EFX\_SRL8 can form chains. If one cell in the chain is assigned a location, every other cell in the chain must also be assigned a location, in the correct order.

## Creating a Location Assignment File

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

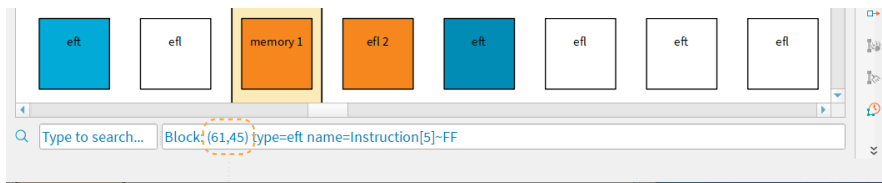
```
<block name>    <x>    <y>    <subblk>
```

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

You must include all data for each assignment.

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

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



*x,y coordinates for the selected tile*

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

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

```
dump_placeloc = on
```



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

### Example: LUT and Flipflop

The example packs an EFX\_FF with its driver, LUT\_A, an EFX\_LUT4.

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

### Example: SRL8 Chain

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

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

### Example: Parallel Cascaded DSP Block

This example assigns locations to every EFX\_DSP24 across two chains. There can be two EFX\_DSP24 cells per DSP tile.

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

# 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 Editor shows the locations of the paths and cells in the fabric.
- The Tcl Command 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](#).

---

# Simulating

## Contents:

- [Changing the Default Testbench Names](#)
- [Simulate with the iVerilog Simulator](#)
- [Simulate with the ModelSim Simulator](#)
- [Simulate with the NCSim Simulator](#)

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. 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.



**Note:** The Efinix primitive models are located in the directory `<installation directory>/sim_models/verilog`

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
```

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
```

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

## Changing the Default Testbench Names

The simulation flow assumes that:

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

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

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

### Example: Changing Default Names

Linux:

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

Windows:

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



**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.



**Note:** You can download the free Icarus Verilog (iVerilog) simulator from [iverilog.icarus.com](http://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 (`<project name>.rtl.simlog` and `<project name>.map.simlog`) and error messages (`<project name>.log`) in your project's **outflow** directory.

## View Waveforms

To use GTKWave to view a waveform:

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

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

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

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

## Simulate with the ModelSim Simulator

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



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

For example, the commands to simulate are:

### Example: Simulate with ModelSim

Linux:

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

Windows:

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

The simulator responds with

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

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

## Simulate with the ModelSim GUI

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

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

4. Add the `vsim -t ps <work space>.<test bench module name>` command to start simulation in the ps timeframe.
5. Add the `run <number>us` command to generate a waveform up to `<number> μs`.
6. Run the ModelSim software in the Transcript console.
7. Change to the project root directory.
8. Use the `do` command to execute the macro (`do <name>.do`).
9. Add signals to the waveform in the **Objects** tab.
10. View the simulation result in the **Wave** tab.

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

**Figure 18: Behavioral 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 lus
```

**Figure 19: 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 lus
```



## Simulate with the NCSim Simulator

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



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

For example, the commands to simulate are:

### Example: Simulate with NCSim

Linux:

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

Windows:

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

The simulator responds with

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

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

# Debugging

## Contents:

- [Profile Editor Perspective](#)
- [Debug Wizard](#)
- [Debug Perspective: Logic Analyzer](#)
- [Debug Perspective: Virtual I/O](#)
- [Concurrent Debugging](#)
- [Resource Usage](#)
- [Disable the Debug Core](#)

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

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

Debugging involves the following general steps:

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

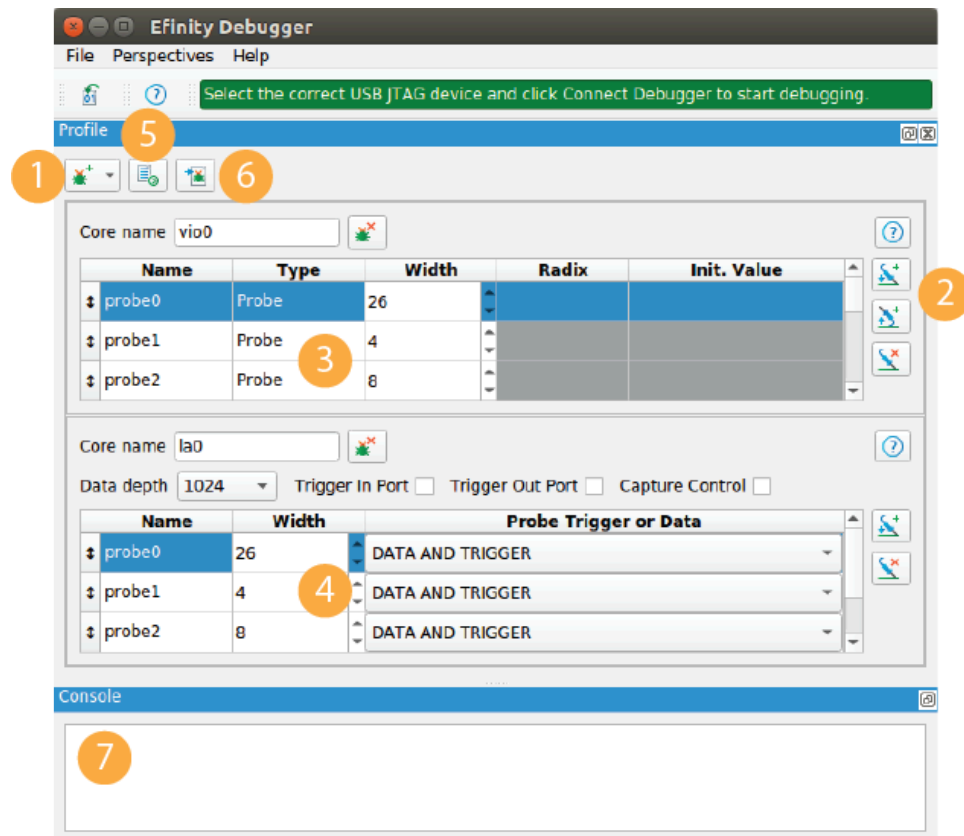


**Note:** The minimum operating frequency of the debug cores is 2 times the JTAG TCK frequency.

## Profile Editor Perspective

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

Figure 20: 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 21: Virtual I/O Core Block Diagram

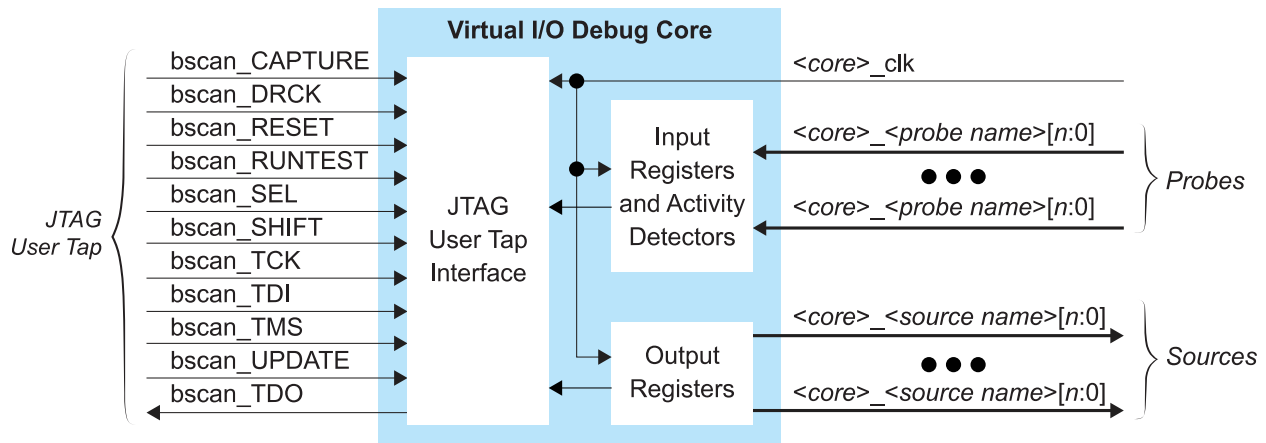


Table 16: Virtual I/O Core Ports

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

## Adding a Virtual I/O Core

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

---

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

---

8. Add a JTAG User Tap block in the Interface Designer. Choose **JTAG\_USER1** as the **JTAG Resource**.



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

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

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

## Logic Analyzer Debug Core

You use the Logic Analyzer core (1a) to monitor the signals in your design. You can capture connected wire or register values over a specified time period. 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 22: Logic Analyzer Core Block Diagram

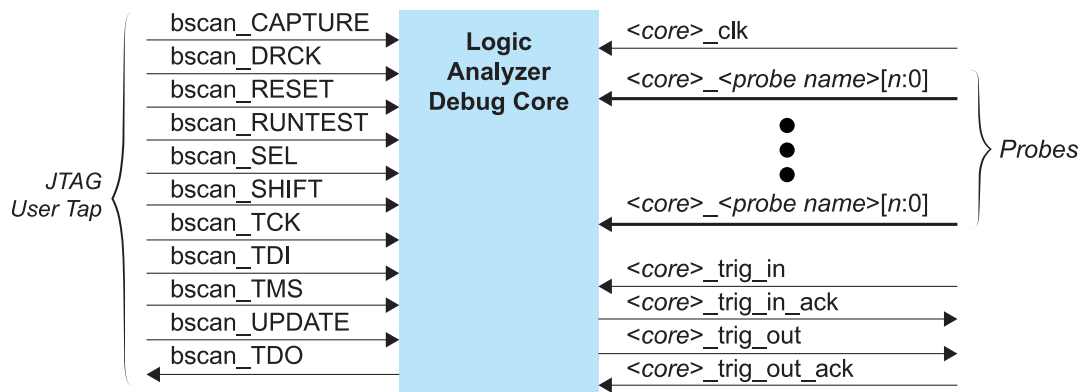


Table 17: Logic Analyzer Core Ports

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.
bscan_RESET	Input	Reset output for the TAP controller.

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

## Adding a Logic Analyzer Core Manually

1. Open the Debugger.
2. Choose **Perspectives > Profile Editor**.
3. Choose **Add Debug Core > Logic Analyzer**.
4. Specify the core name.
5. Select the data depth. This settings lets you control how much data is saved for the probes. The more data you save, the more on-chip memory is used.
6. Turn on Trigger In Port and/or Trigger Out Port to enable those signals.
7. Turn on Capture Control if you want to change the capture mode in the **Capture Setup** tab during debugging (see **Debug Perspective: Logic Analyzer** on page 66 for details). If you turn this option on, the Logic Analyzer uses more FPGA resources.
8. Add probes (outputs from your design to the JTAG interface).
  - a. Choose a width and specify the signal to which you want to connect the probe in your design.
  - b. Choose **Data and Trigger** (default) to save data and can trigger when to capture. Choose **Data Only** to save data.  
Choose **Trigger Only** to trigger when to capture data. Trigger only signals do not display in the resulting waveform.
9. Click **Generate Core RTL**. The Efinity® software saves the debug profile in your project directory as **debug\_profile.json**. The software also creates a debug template (**debug\_TEMPLATE.v**), which includes the module for the debug profile you created and **debug\_top.v**, which is the RTL logic for the debug core.
10. Add the **debug\_top.v** file to your project.

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

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



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

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

- a. Add all of the JTAG input and output pins to the project's top module. Refer to the JTAG User TAP block pin names in the Interfaces Design to get the pin list.
- b. Instantiate the debug core in the project's top module. You can copy the example code from the generated **debug\_TEMPLATE.v** file in the project folder.
- c. Connect the ntes that you want to monitor and drive the FPGA signals. You need to map the net (input, output, wire, register, and/or signal) to the port of the instantiated debug core (edb\_top\_inst).

13. Compile the design.

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



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

## Debug Wizard

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

### Using the Wizard

1. Launch the Debug Wizard.
2. Choose the buffer depth. The buffer uses on-chip RAM, therefore, a larger buffer uses more RAM.
3. Optionally enable capture control. Enabling this option lets you change the capture mode in the **Capture Setup** tab during debugging (see [Debug Perspective: Logic Analyzer](#) on page 66 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:

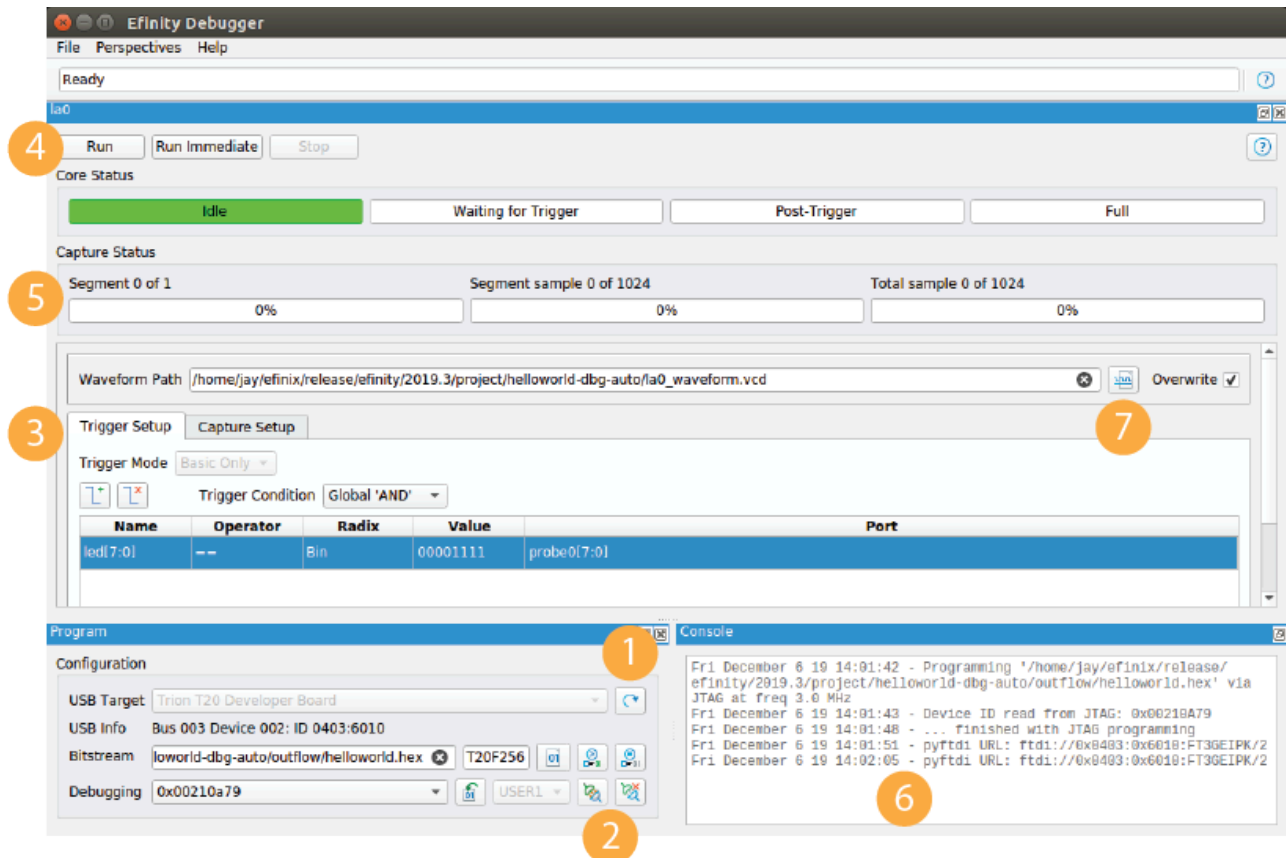
1. Open the Project Editor.
2. Click the **Debugger** tab.
3. Select your project's **debug\_profile.wizard.json** in the **Debug Profile** box.
4. Turn on **Debugger Auto Instantiation**.









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

## Debug Perspective: Logic Analyzer

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

Figure 23: Debug Perspective GUI - Logic Analyzer



- |   |                   |   |                      |   |                     |
|---|-------------------|---|----------------------|---|---------------------|
|  | Select Bitstream  |  | 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.
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.

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.



**Note:** Download and install the GTKWave software from [gtkwave.sourceforge.net](http://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.

## *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:

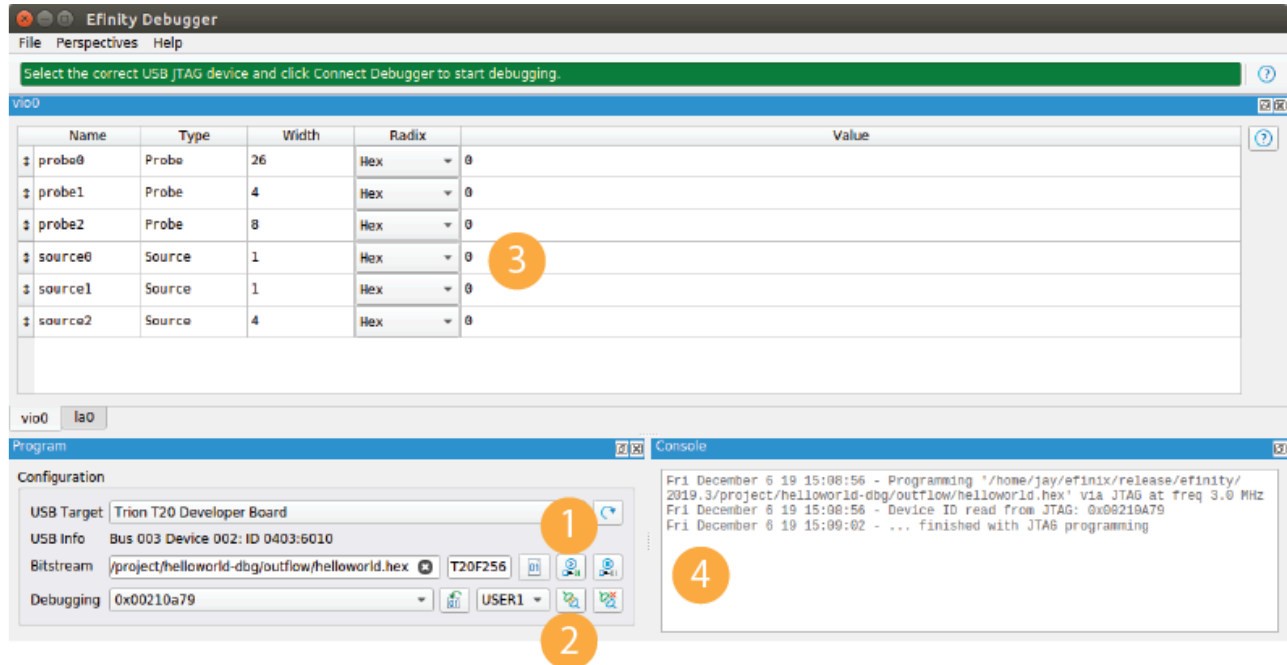
- **Always**—Stores a data sample during a given clock cycle regardless of any capture conditions you set.
- **Basic**—Only stores a data sample during a given clock cycle if the capture condition evaluates as true. Select this option to add nets and set capture conditions.

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.

## Debug Perspective: Virtual I/O

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

Figure 24: 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.

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.

## Concurrent Debugging

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

This feature is supported in the Efinity software v2022.1 and higher.

The concurrent debug feature requires you to connect to the board using the Efinity Hardware Server. You can use the same computer for the server and client. Launch the

server with the board connected to your computer and then connect to the server from the Debugger client using the same IP address.

See **Working with Remote Hardware** on page 101 for instructions on setting up a Hardware Server.

To open more than one Debugger window:

1. Open the first Debugger window using the Efinity GUI as you normally would.
2. Open the second Debugger window by executing the following command in a terminal or command prompt:

*Windows:*

```
<Efinity_path>\debugger\bin\efinity_dgb.bat
```

*Linux:*

```
<Efinity_path>/debugger/bin/efinity_dgb.sh
```



**Download:** Refer to the Efinity Debugger Tutorial for more information on using concurrent debug.

## Resource Usage

In Efinity version 2020.1 and higher, you can view the resources used by the debug cores in the Dashboard's Results pane in the Debugger table. The software reports:

- Whether auto-instantiation is turned on or off
- Whether the debug target is the elaborated or post-map netlist
- Number of flipflops used
- Number of adders used
- Number of LUTs used
- Number of memory blocks used



**Note:** The resource 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.

# Configuring an FPGA

## Contents:

- [FPGA Configuration Modes](#)
- [Flash Programming Modes](#)
- [About the Programmer GUI](#)
- [Generate a Bitstream \(Programming\) File](#)
- [Working with Bitstreams](#)
- [SPI Programming](#)
- [JTAG Programming](#)
- [Using the Command-Line Programmer](#)
- [Project-Based Programming Options](#)
- [Configuration Status Register](#)
- [Verifying Configuration with the Programmer](#)
- [Securing Titanium Bitstreams](#)

When you have finished running your design through the flow, you are ready to configure a device. You configure devices using the standalone GUI or command-line Programmer tool and a USB cable attached to your board. You can download the bitstream file into the device itself or into flash memory. Before you begin configuration, install the USB drivers for the programming cable (see [Appendix: Installing USB Drivers](#) on page 103).

## FPGA Configuration Modes

Trion<sup>®</sup> and Titanium FPGAs have dedicated configuration pins. You select the configuration mode by setting the appropriate condition on the input configuration pins. Trion<sup>®</sup> and Titanium FPGAs support the following configuration modes.

**Table 18: 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.

The Efinity software includes the JTAG SPI Flash Loader IP core that gives you full control over a SPI flash device and lets you perform actions comparable to an FTDI flash controller chip. With this IP core you can turn the FPGA into a flash programmer and use it to program the flash device.

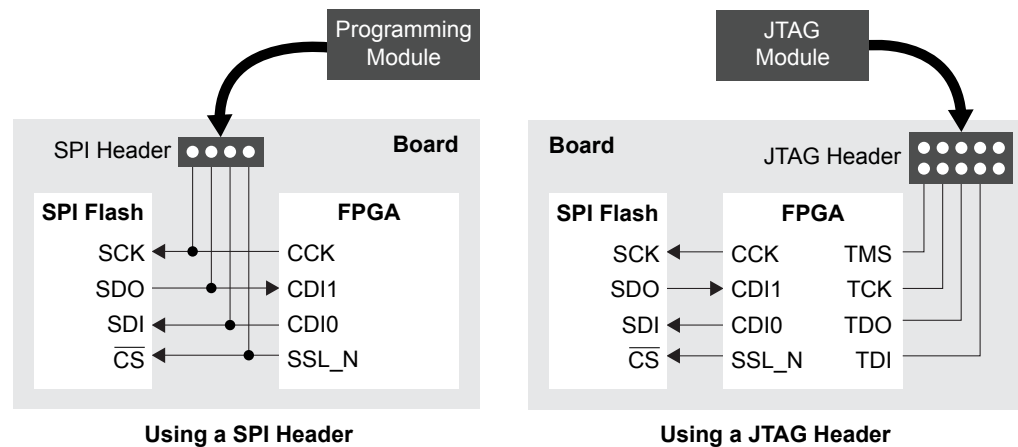


**Learn more:** Refer to the [JTAG SPI Flash Loader Core User Guide](#) for more information.

**Table 19: 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	Program a single flash device. First, program the FPGA with a design that turns it into a flash programmer. Then, program the flash.
SPI Active x8 using JTAG Bridge	Program two identical flash devices. First, program the FPGA with a dual flash programmer. Then, program the two flash devices. This mode allows you to load images more quickly, and is only supported for Titanium FPGAs.

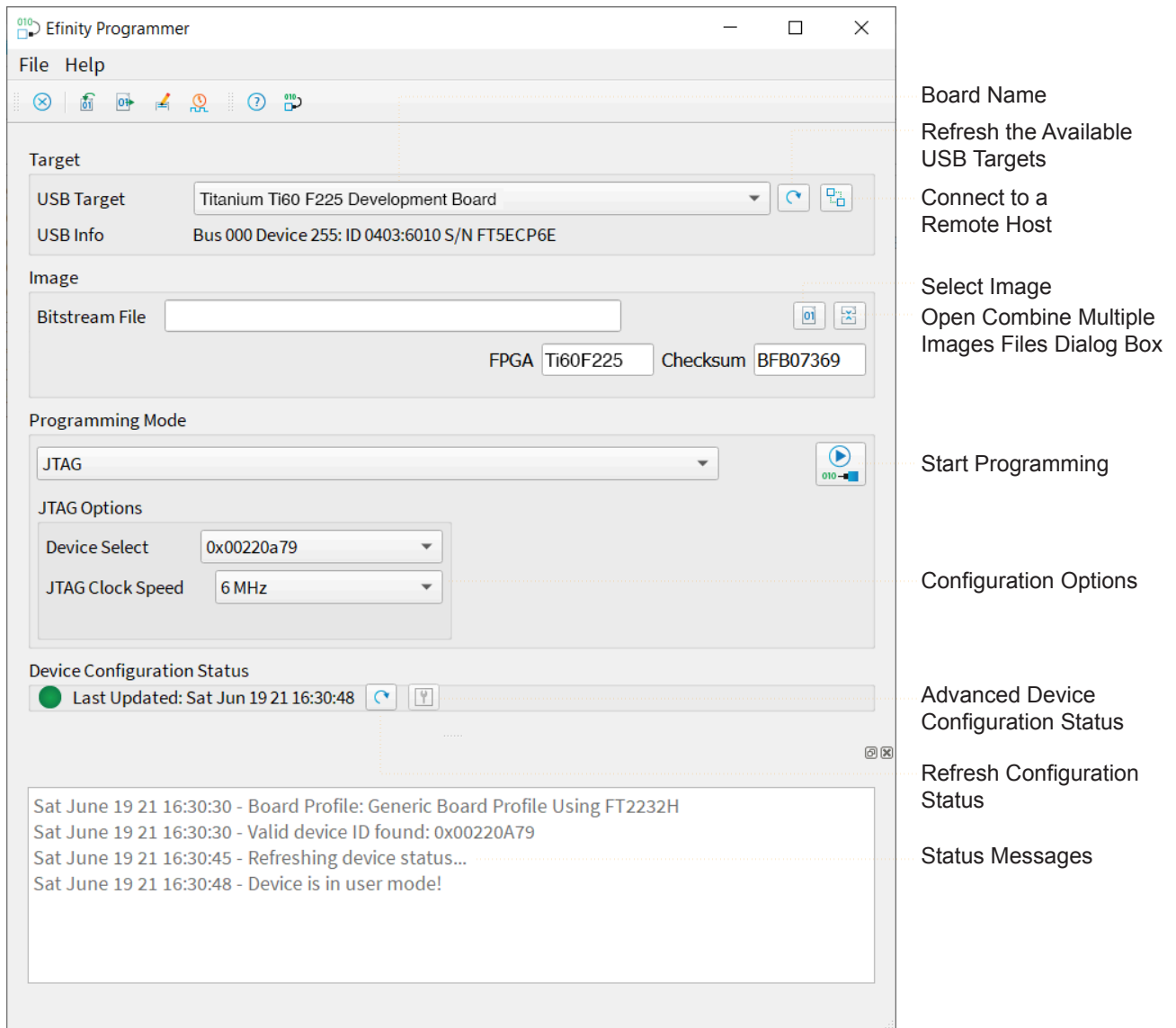
**Figure 25: Flash Programming Board Setup**



## About the Programmer GUI

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

Figure 26: Programmer



To use the Programmer:

1. Choose a target. Click the Edit Remote Host List button to connect to a board attached to a remote host. See [Working with Remote Hardware](#) on page 101.
2. Choose a bitstream file. Use a **.hex** file for SPI modes or a **.bit** file for JTAG mode. After you select a bitstream, the Programmer reads the bitstream and displays data in the **FPGA** and **Checksum** fields. The checksum excludes the pre-header and ignores whether



characters are uppercase or lowercase; therefore, it is a checksum of the bitstream content, not a file checksum.

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

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

3. Choose the programming mode and then select options.

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

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

The Programmer has status information that gives you diagnostics:

- The FPGA or flash device's configuration status displays in the Device Configuration Status area. Click the Refresh button to refresh the status and display messages in the console.
- Use the Advanced Device Configuration Status button to get diagnostics that can be helpful when debugging why configuration is failing. Refer to [Configuration Status Register](#) on page 88 for more information.



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

## Edit the SPI Active Clock

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

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

**Table 20: 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 86.



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

## 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 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 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 76.

## Export to .svf Format

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

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



**Note:** For more information on using this bitstream format, refer to [Working with JTAG .svf Files](#) on page 99.

You can also convert the file to **.hex** at the command line as described in [Convert to Intel Hex Format at the Command Line](#) on page 76.

## Convert to Intel Hex Format at the Command Line

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

```
export_bitstream.py [-h] [--family <Trion or Titanium>] [--idcode IDCODE] [--freq FREQ]
  [--sdr_size SDR SIZE] [--tir_length TIR LENGTH] [--hir_length HIR LENGTH]
  [--tdr_length TDR LENGTH] [--hdr_length HDR LENGTH] [--enter_user_mode <on or off>]
  <format> <input filename> <output filename>
```

Where *<format>* is:

- hex\_to\_bin
- hex\_to\_intelhex
- bin\_to\_hex
- intelhex\_to\_hex
- hex\_to\_svf

For example:

```
C:\Efinity\2021.1\bin\setup.bat
python3 C:\Efinity\2021.1\pgm\bin\efx_pgm\export_bitstream.py hex_to_bin new_project.hex
test2.bin
```

## Combine Bitstreams and Other Files

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

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

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

*Table 21: Modes when Combining Images*

Mode	Use For	Number of Images	Refer to
Selectable Flash Image	Multi-image configuration	Up to 4	<a href="#">Program Multiple Images (CBSEL)</a> on page 77
	Internal reconfiguration	Up to 4	<a href="#">Program Multiple Images (Internal Reconfiguration)</a> on page 78
Daisy Chain	Daisy chains	Any number of JTAG devices including those from other vendors	<a href="#">Program a Daisy Chain</a> on page 79
Generic Image Combination	A bitstream and other files	One bitstream and any number of other files	<a href="#">Program Multiple Images (Bitstream and Data)</a> on page 79

## SPI Programming

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

### Program a Single Image

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

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

### Program Multiple Images (CBSEL)

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

1. Click the **Combine Multiple Images** button.
2. Choose **Mode > Selectable Flash Image**.
3. Enter the output file name.
4. Choose the output file location. The default is the project's **outflow** directory.
5. Choose **External Flash Image**.
6. Click in the table row corresponding to the position for which you want to add an image.

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



**Note:** For more information on programming multiple images, refer to [Example Design: Configuring a Trion Development Board with Multiple Images](#) on the Downloads page in the Support center.

## *Program Multiple Images (Internal Reconfiguration)*

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

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



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

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



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

## Program Multiple Images (Bitstream and Data)

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

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



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

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

## Program a Daisy Chain

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

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

## JTAG Programming

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

### JTAG Device IDs

*Table 22: Titanium JTAG Device IDs*

FPGA	Package	JTAG Device ID
Ti35	All	0x10661A79
Ti60ES	All	0x00360A79
Ti60	All	0x10660A79
Ti90	M361, M484, F529	0x00681A79
	L484	0x00688A79
Ti120	M361, M484, F529	0x00682A79
	L484	0x0068CA79
Ti180	M361, M484, F529	0x00680A79
	L484	0x00684A79

*Table 23: Trion JTAG Device IDs*

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

### Program a Single Image

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

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

### Program Using a JTAG Chain

You can program an FPGA that is part of a JTAG chain. The chain can include Trion<sup>®</sup> 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 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 ICODE instruction



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



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

To program using a JTAG chain:

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

## Program using a JTAG Bridge

Programming with a JTAG bridge is a 2-step process: first you configure the FPGA to turn it into a JTAG programmer (**.bit**) and second you use the FPGA to program the flash device with the bitstream (**.hex**).

With the Efinity software v2021.2.323.2.18 or higher, the Programmer has an option to auto-configure the FPGA with the **.bit** file so that you can complete both steps with a single button push.

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** mode.
4. Turn on the **Auto configure JTAG Bridge Image** option.
5. Click the **Select Image File** button to browse for the **.bit** file to configure the FPGA. The Programmer stores the file location so you do not have to specify it the next time you program.
6. Click **Start Program**. The Programmer first configures the FPGA and then programs the flash device.



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

## JTAG Programming with FTDI Chip Hardware

These instructions describe how to program Trion<sup>®</sup> and Titanium FPGAs using the FTDI Chip FT232H and FT4232H Mini Modules. Efinix<sup>®</sup> has tested the hardware for use with Trion<sup>®</sup> 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 and programming failures.

1. Open the Efinity<sup>®</sup> software.
2. Open the Efinity<sup>®</sup> 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 FT232H 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<sup>®</sup> includes a script, **ftdi\_program.py**, which you can use for command-line programming with FTDI modules. The command is in the format:

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

`<mode>` is the programming mode (active, passive, jtag, jtag\_chain, jtag\_bridge, or jtag\_bridge\_x8<sup>(5)</sup>)



**Note:** To use the `jtag_bridge` or `jtag_bridge_x8` modes, you must have already configured the FPGA with the JTAG SPI flash loader. Refer to the [JTAG SPI Flash Loader Core User Guide](#) for more information.



**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.

`<url>` is in the format:

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

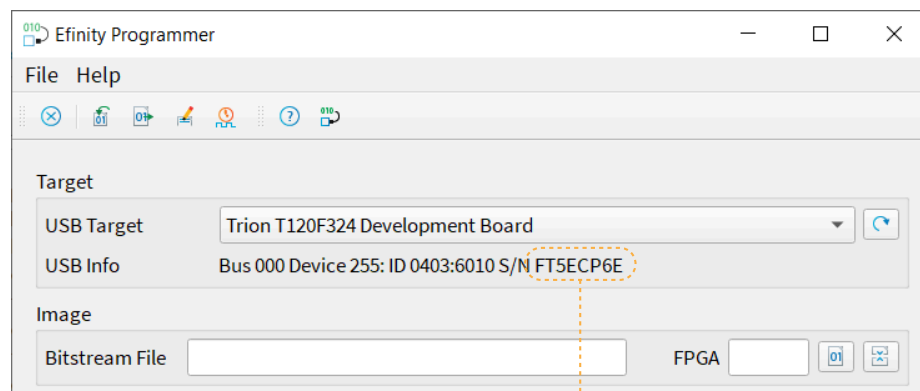
where:

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

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

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

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



The string after S/N is the FTDI serial number

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

## Linux Examples

To program in Linux:

<sup>(5)</sup> The `jtag_bridge_x8` mode is only supported in some Titanium FPGAs. Refer to the data sheet for the modes your FPGA supports.

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

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

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

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

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

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

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

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

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

## Windows Examples

To program in Windows:

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

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

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

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

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

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

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

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

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

## Using the Command-Line Programmer

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

### Example: Command-Line Programmer

Linux:

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

Windows:

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

(Optional) Use these options:

- `--pgm_opts mode` specifies the configuration mode. The available modes are:
  - `active`—SPI active configuration
  - `passive`—SPI passive configuration
  - `jtag`—JTAG programming
  - `jtag_bridge`—SPI active using JTAG bridge mode
  - `jtag_bridge_x8`—SPI active x8 using JTAG bridge mode (used with 2 flash devices)<sup>(6)</sup>

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

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

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

<sup>(6)</sup> Used with 2 flash devices. Only supported in some Titanium 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 24: Project-Specific Programming Options**

Option	Notes
Active/Passive	Your choice of active or passive affects the pinout and determines which choices are available for the programming mode.
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: 0xFFFFFFFF
Clock Source	For Titanium 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 <b>Internal Oscillator</b> .
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 FPGAs, choose whether the configuration clock should sample on the rising or falling edge. The default is <b>Rising</b> . For Trion FPGAs, this option is always <b>Rising</b> .
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	When you turn this option on, the control block issues 4-byte addresses when it configures the FPGA. This option is not supported for Ti35 and Ti60 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. <b>on:</b> The bitstream has initialized memory. <b>off:</b> The bitstream does not have initialized memory. <b>smart:</b> For the Trion family, this option has the same effect as <b>on</b> . For the Titanium 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 only) When turned on (default), the software compresses the bitstream. If you choose <b>Bitstream Encryption</b> or <b>Bitstream Authentication</b> , this option is turned off and disabled because you cannot compress a bitstream and use the security features simultaneously.
Bitstream Encryption	On: The software generates an encrypted bitstream. You also need to specify the <b>.bin</b> file in the <b>FPGA Key Data File</b> box. Off: (default) The software generates a plaintext bitstream.
Bitstream Authentication	On: The software generates a signed bitstream. You also need to specify the <b>.bin</b> file in the <b>FPGA Key Data File</b> box and the RSA private key ( <b>.pem</b> ) file in the <b>RSA Private Key</b> box. Off: (default) The software generates an unsigned bitstream.
FPGA Key Data File	Specify the location and name of the <b>.bin</b> file you generated with the Efinity Bitstream Security Key Generator.
RSA Private Key	Specify the location and name of the RSA private key file ( <b>.pem</b> ).

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

Figure 27: Setting Programming Options (Trion)

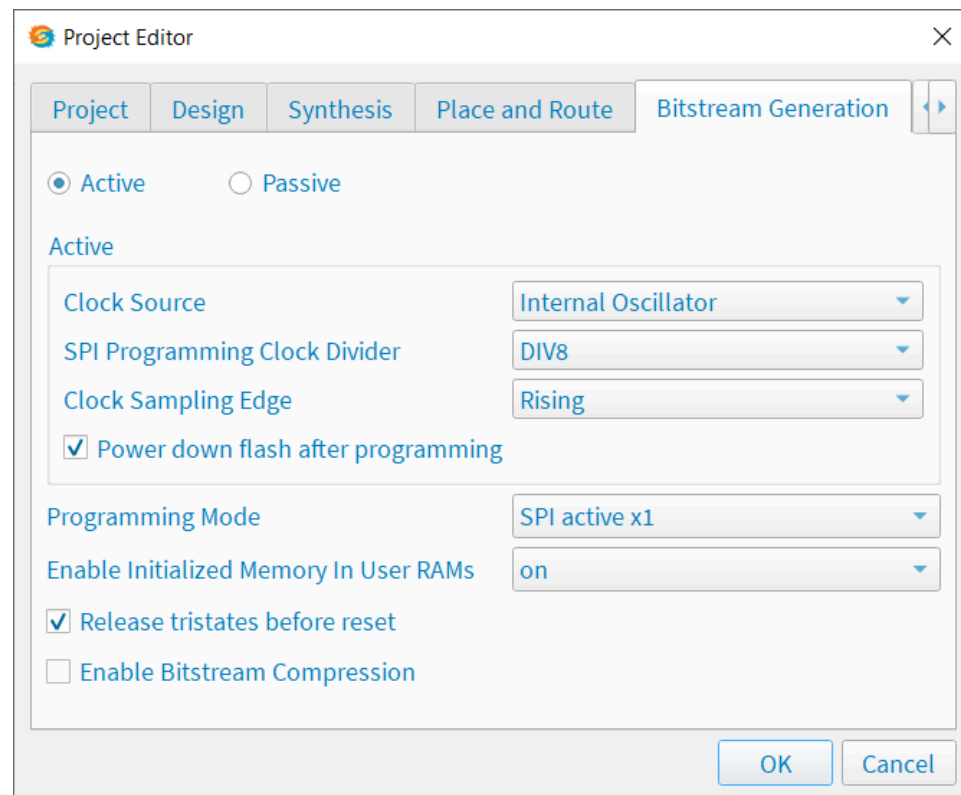


Figure 28: Setting Programming Options (Titanium)

The screenshot shows the 'Project Editor' window with the 'Bitstream Generation' tab selected. The 'Active' radio button is chosen. Under the 'Active' section, the 'Clock Source' is set to 'Internal Oscillator', 'SPI Programming Clock Divider' is 'DIV8', and 'Clock Sampling Edge' is 'Rising'. The checkbox 'Power down flash after programming' is checked. The 'Programming Mode' is set to 'SPI active x1' and 'Enable Initialized Memory In User RAMs' is set to 'on'. The checkboxes 'Release tristates before reset' and 'Enable Bitstream Compression' are both checked. Under the 'Bitstream Security' section, 'Bitstream Encryption' and 'Bitstream Authentication' are unchecked. There are input fields for 'FPGA KeyData File' and 'RSA Private Key', each with a file selection icon to its right. 'OK' and 'Cancel' buttons are at the bottom right.



**Learn more:** 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 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 25: Configuration Status Register**

Name	Description
IN_USER <sup>(7)</sup>	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.
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. 0: Indicates that the FPGA received a bitstream with the wrong device ID or a CRC error is detected. NSTATUS can also go low if the bitstream does not have the correct signature. 1: During configuration, indicates that the FPGA is in configuration mode.
CRC32_ERROR_CORE	0: No CRC errors were detected in the core configuration bits. 1: One or more CRC errors were detected in the core configuration bits.
RMUPD_ERROR	0: No errors occurred during remote update. 1: An error occurred during remote update configuration. Has the same value as the remote update error status signal sent to the core fabric.
CONFIG_END	0: Configuration is not complete. 1: Configuration completed (whether successful or not).
SYNC_PAT_FOUND	0: Indicates that the FPGA is not receiving the expected synchronization pattern at start of the bitstream. Check for board or power issues. 1: Indicates that the FPGA detected a synchronization pattern at start of the bitstream, and the clock and data connections to the FPGA are acceptable. Any configuration problems are likely digital or logical in nature.
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 <sup>(7)</sup>	For an encrypted bitstream: 0: Encryption 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.
RSA_PASS <sup>(7)</sup>	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.
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.

<sup>(7)</sup> This bit is not supported in Ti60ES FPGAs.

Name	Description
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:

- Use the Efinity Programmer to check whether the FPGA is in user mode. In the Programmer, click the **Device Configuration Status > Refresh Device Configuration Status** button. The console displays the FPGA status.
- For SPI active modes, if the **Programmer > SPI Active Options > Verify After Programming** option is turned on (which is the default), the Programmer confirms that the flash is programmed correctly. This option can help find errors such as a flash image that is corrupted during a write, an improperly skipped erase step, etc.



**Note:** Generally, Efinix recommends that you keep the **Verify After Programming** option turned on. However, if you are using the FPGA's built-in CRC checking (enabled by default) with NSTATUS monitoring to verify configuration, you can use that method as a way of verifying the flash (that is, if the FPGA goes into user mode, the flash write is verified).

You can also use a microcontroller or LEDs to verify configuration. Refer to "Monitoring with a Microcontroller or LEDs" in [AN 006: Configuring Trion FPGAs](#) or [AN 033: Configuring Titanium FPGAs](#).

### Monitoring with the Efinity Programmer

With the board connected to your computer, you can monitor the FPGA's status with the Programmer. The following table describes the values for CDONE and NSTATUS and their meaning.

*Table 26: Monitoring with the Efinity Programmer*

CDONE	NSTATUS	Programmer Message	Description
0	0 or 1	Device is not in user mode	Configuration failed. If NSTATUS is 0, it indicates that the FPGA received a bitstream with the wrong device ID or a CRC error is detected.
1	1	Device is not in user mode	Bit stream transmission completed, however, it is corrupted with undetectable errors that is not wrong device ID or CRC error.
1	1	Device is in user mode	<p>The FPGA is functioning correctly according to the user design.</p> <p>The FPGA is not functioning as expected. The most likely reason is that the FPGA is programmed with a bitstream that was targeted for a different configuration mode or width.</p>

## Securing Titanium Bitstreams




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

- *Encryption*—Encrypt your bitstream using an AES-256 key.
- *Authentication*—Sign your bitstream with an RSA-4096 private key.

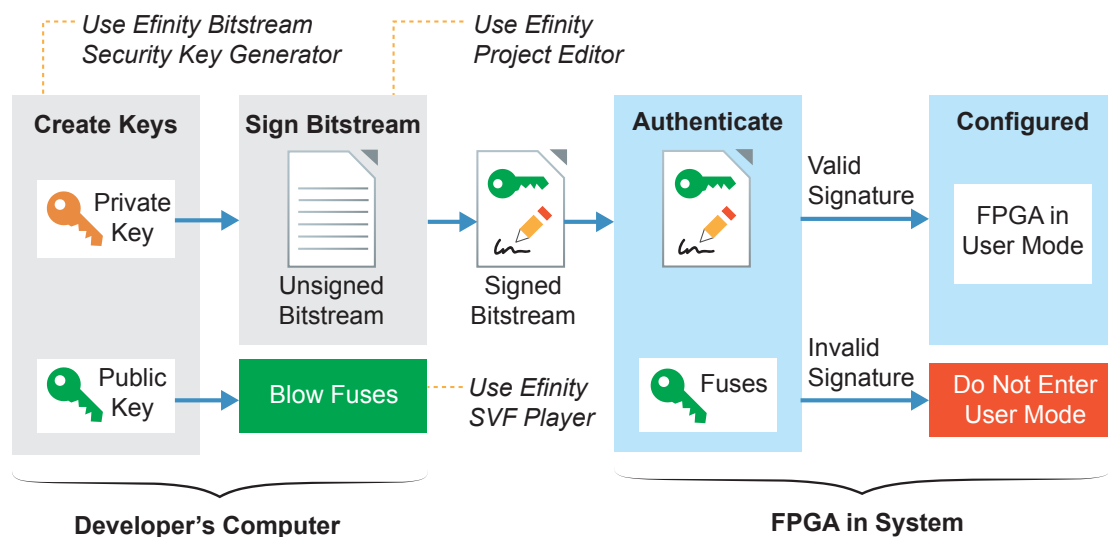
- *Disable JTAG*—Disable all JTAG instructions except for `IDCODE`, `DEVICE_STATUS`, and `BYPASS`.

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

**Table 27: 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 bitstream signed with the correct private key. After you blow fuses with an AES-256 key, the FPGA only accepts a plaintext bitstream or a bitstream signed with the correct key. Program the JTAG fuse to disable JTAG function. After you blow the JTAG fuse, you cannot use any JTAG command except <code>IDCODE</code> , <code>DEVICE_STATUS</code> , and <code>BYPASS</code> .
 Project Editor	Turn on bitstream encryption and/or authentication, and specify the <b>.bin</b> file created by the Bitstream Security Key Generator. Turn on bitstream authentication and specify the private key ( <b>.pem</b> ) file to sign the bitstream.

**Figure 29: 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 30: Bitstream Encryption

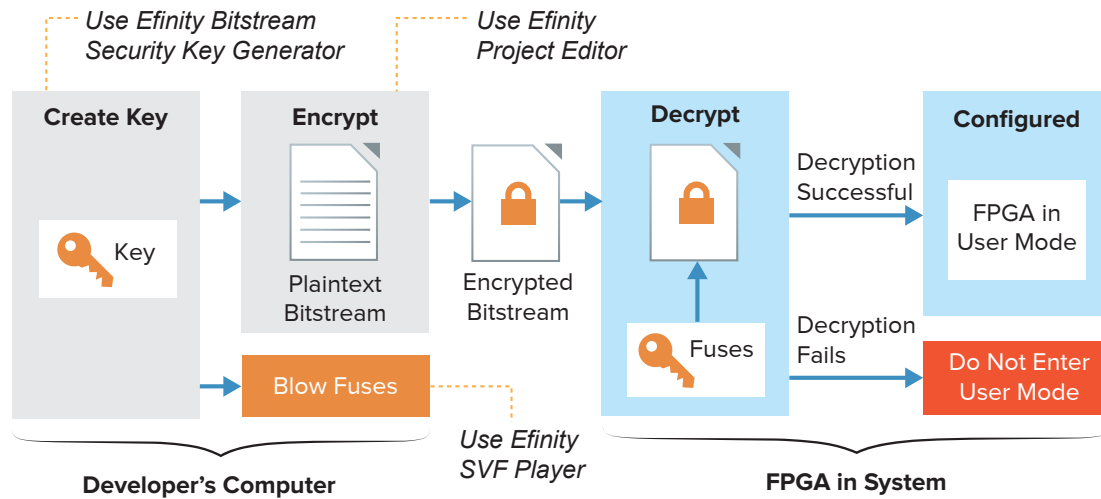
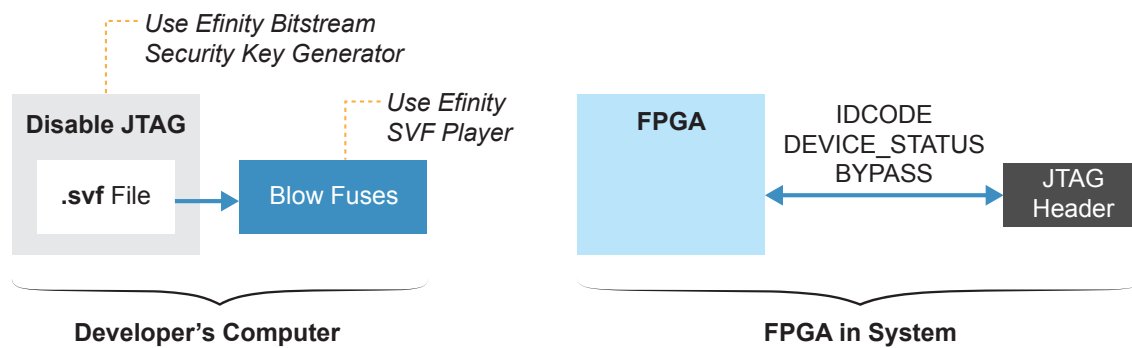


Figure 31: Disabling JTAG



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

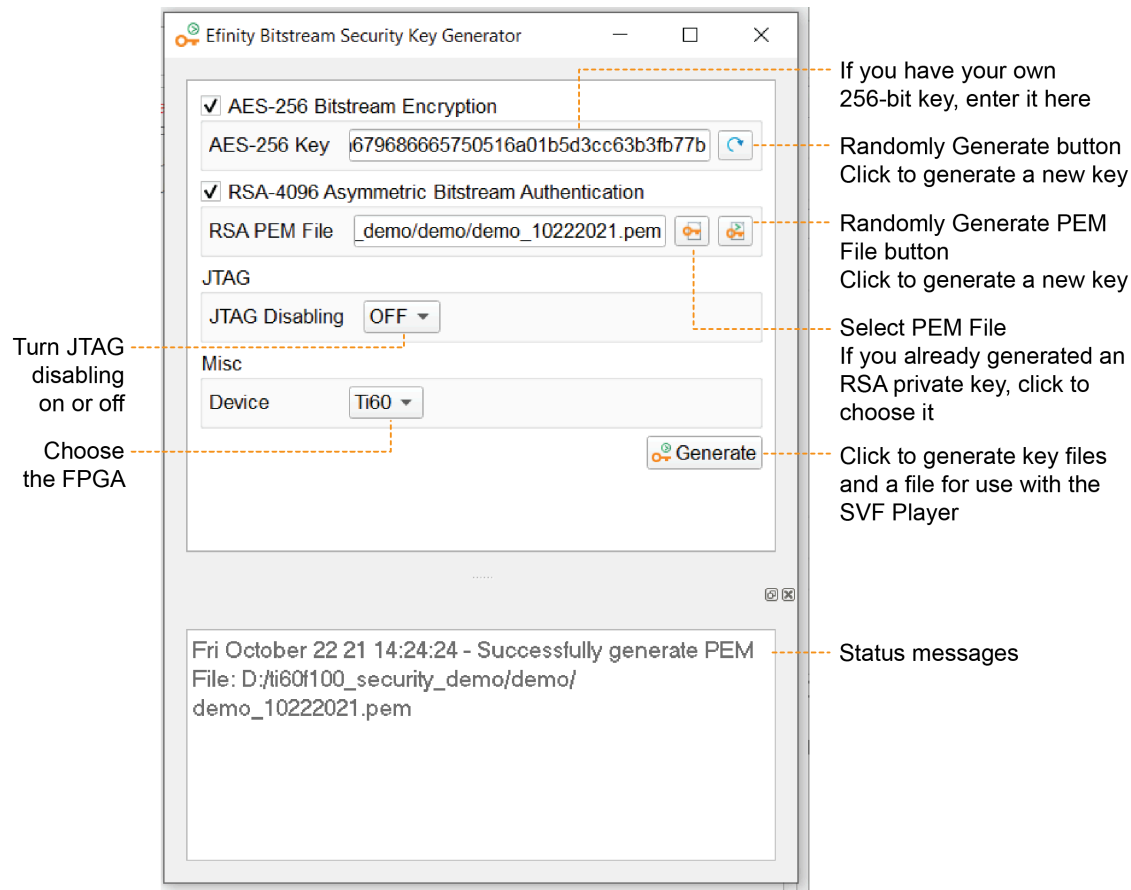
### Using the Efinity Bitstream Security Key Generator

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



**Note:** You can use the Efinity Bitstream Security Key Generator iteratively. That is, you can first use encryption and later add in RSA authentication, and even later disable JTAG commands. Refer to [Workflow for Using Security Features](#) on page 96 for more information.

Figure 32: 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** for **JTAG Disabling**. Otherwise, leave it set to **OFF**.



**Important:** Do not 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.

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 96). You need to copy it from the **AES-256 Key** box and save it into a text file.

## Blowing Fuses with the SVF Player

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

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

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

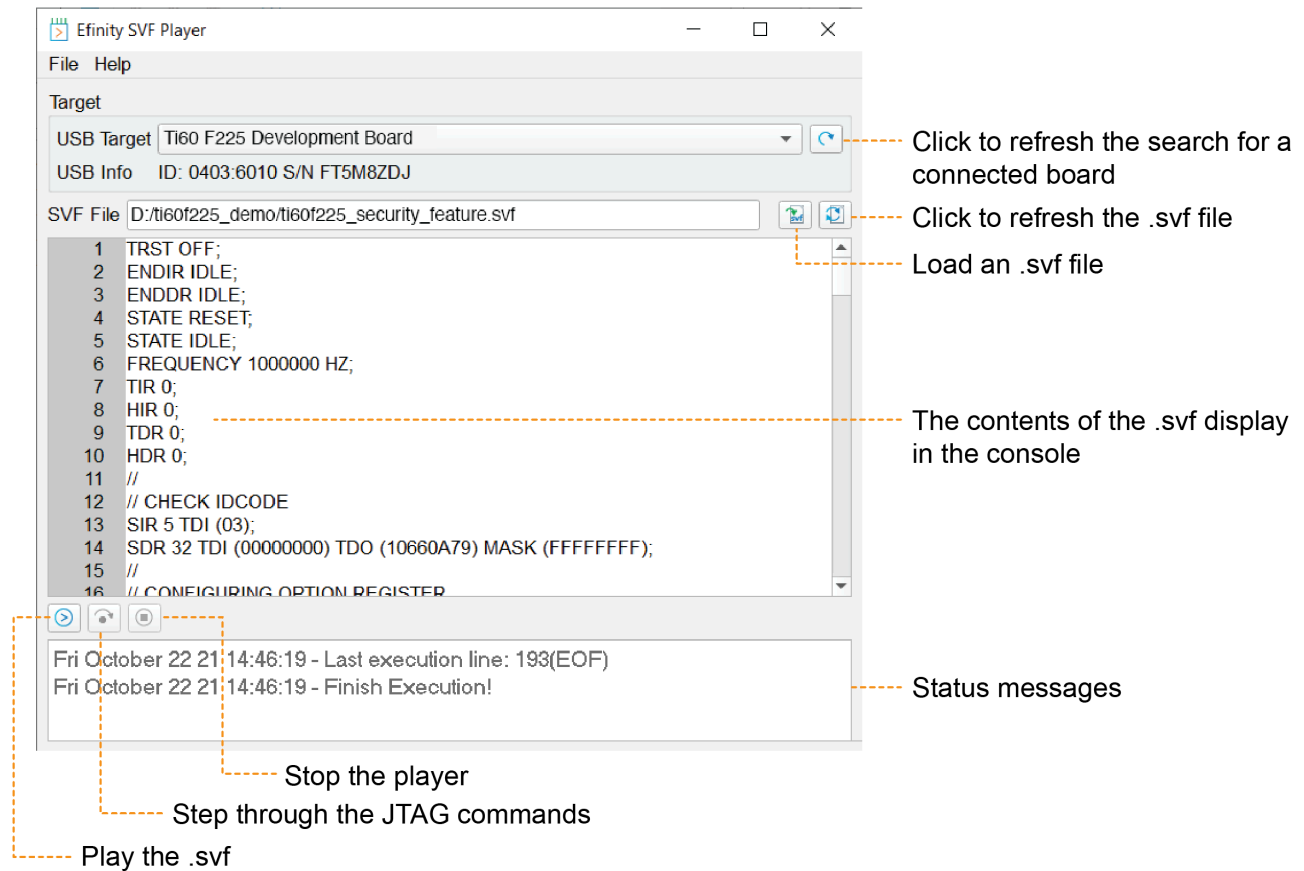
- It checks the FPGA's **IDCODE** and compares it to the **.svf** to ensure that the player is targeting the correct FPGA.
- For AES encryption, the key is sent in eight 32-bit words, followed by a validation step.
- For RSA authentication, the key is sent in twelve 32-bit words, followed by a validation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

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




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

Figure 33: 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 28: Project Options for Security**

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

## 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 88 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 34: Advanced Device Configuration Status Security Signals**

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

Security Signals {

Update

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

# Working with JTAG .svf Files

## Contents:

- **Using the Efinity SVF Player**

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

The Efinity Programmer can convert a bitstream file to **.svf** so that you can use third-party JTAG hardware to program an Efinix FPGA. Refer to **Export to .svf Format** on page 76.

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

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

## Using the Efinity SVF Player

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

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

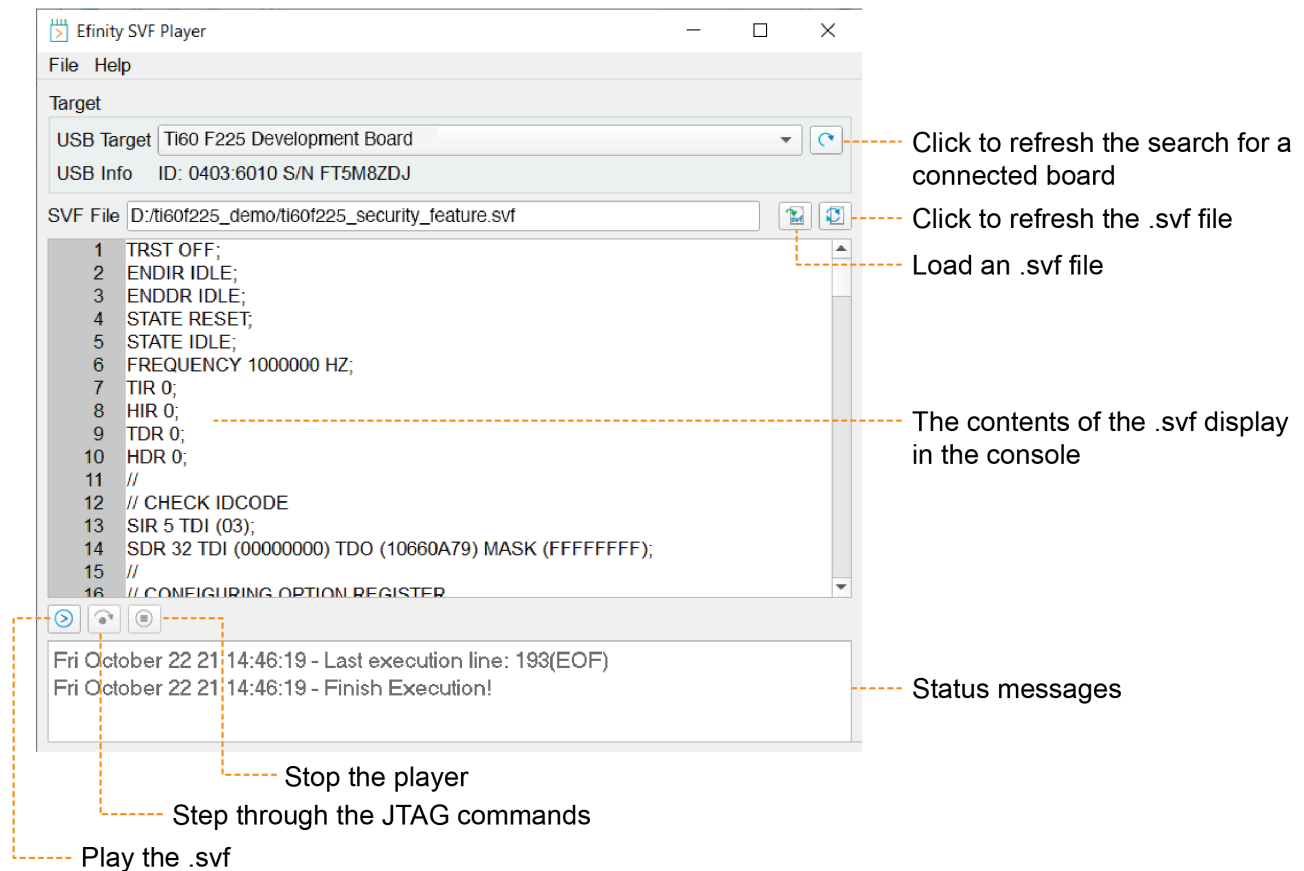
- Trion T20 in F324 and F400 packages
- Trion T35 in any package
- Trion T55, T85, and T120 in any package
- All Titanium FPGAs in any package

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 35: 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.

# Working with Remote Hardware

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



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

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

## Start the Efinity Hardware Server

You start the Efinity Hardware Server using the **efinity\_hw\_server.py** command-line tool.

```
efinity_hw_server.py [-h] [-a <address>] [-p <port>]
```

Where:

- **-h** shows help.
- **<address>** is the server address; if you do not specify an address, the Efinity Hardware Server defaults to 0.0.0.0 (that is, all IPv4 addresses on the local machine).
- **<port>** is the server port number; if you do not specify a port, the Efinity Hardware Server defaults to 8080.

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

### Windows:

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

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

### Linux:

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

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

## Stop the Efinity Hardware Server

In the terminal or Command Shell, enter **Ctrl+C** to stop the server.

## Connect the Board to the Server

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

For your own board, use a JTAG Mini-Module or JTAG cable to connect the board to the server. When you connect to the remote host from your computer, the module or cable name appears in the Programmer's **USB Target** list. (Refer to **JTAG Programming with FTDI Chip Hardware** on page 82.)

### Connect to a Remote Host

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

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

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

# Appendix: Installing USB Drivers

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

Efinix development boards have FTDI chips (FT232H, FT2232H, or FT4232H) to communicate with the USB port and other interfaces such as SPI, JTAG, or UART.



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

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



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

**Table 29: USB Programming Connections**

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



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

## Installing the Linux USB Driver

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

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

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



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

## Installing the Windows USB Driver

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



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

To install the driver:

1. Connect the board to your computer with the appropriate cable and power it up.
2. Run the Zadig software.



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

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



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

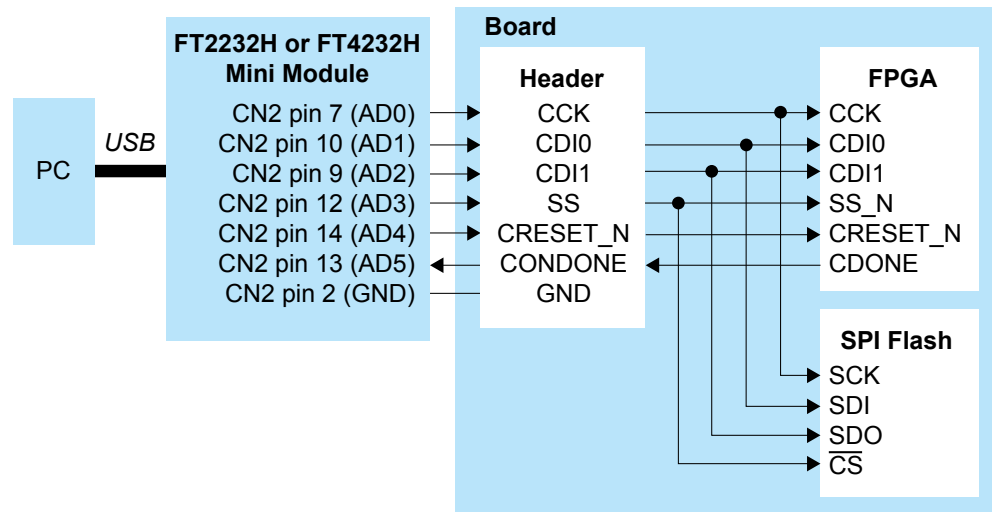


# Appendix: Connecting Programming Hardware

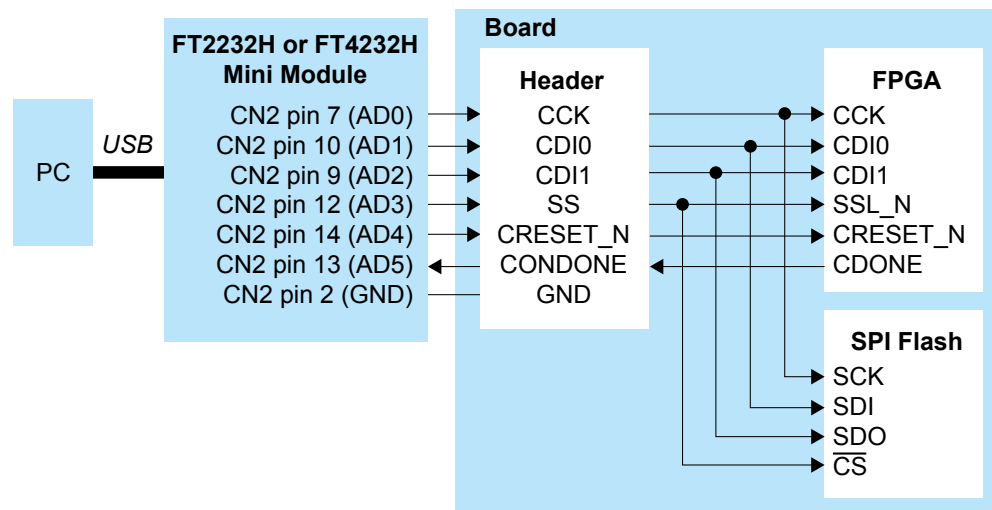
## SPI Programming Connections

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

*Figure 36: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Trion FPGAs)*



*Figure 37: SPI Flash Programming with FTDI FT2232H and FT4232H Mini Module Connections (Titanium FPGAs)*



## JTAG Programming Connections (Trion FPGAs)

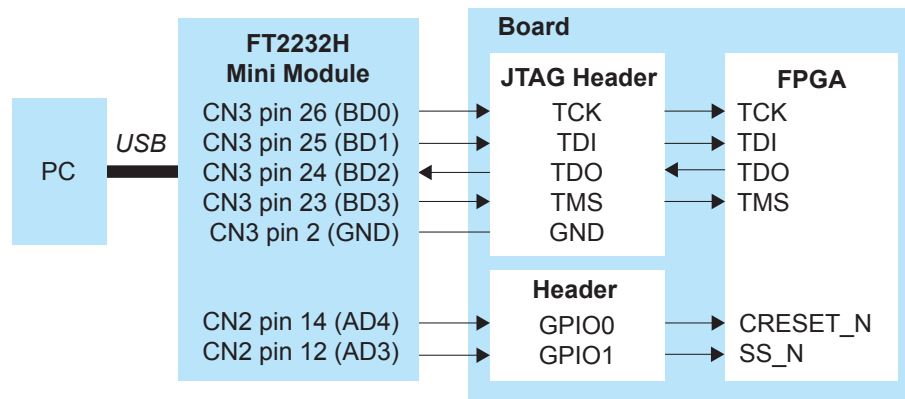
### Connecting a JTAG Cable

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

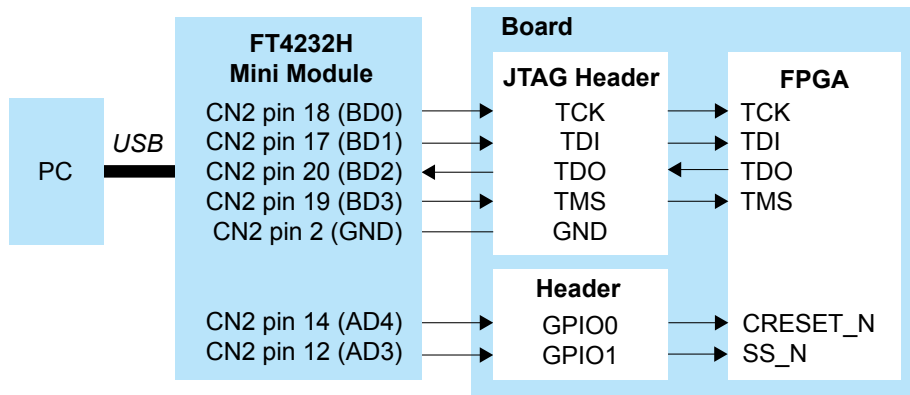
## Connecting a JTAG Mini Module

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

**Figure 38: Connect FT2232H Mini Module to JTAG Pins plus CRESET\_N and SS\_N**



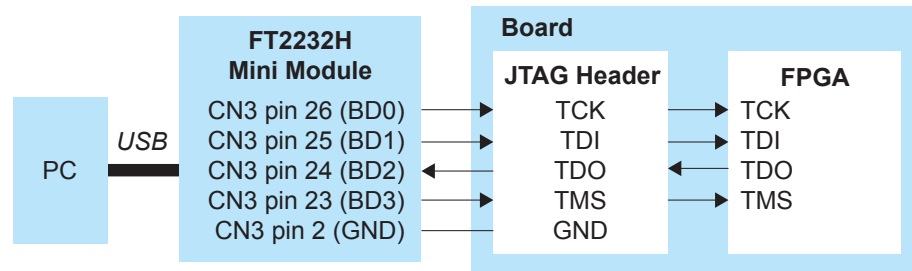
**Figure 39: Connect FT4232H Mini Module to JTAG Pins plus CRESET\_N and SS\_N**



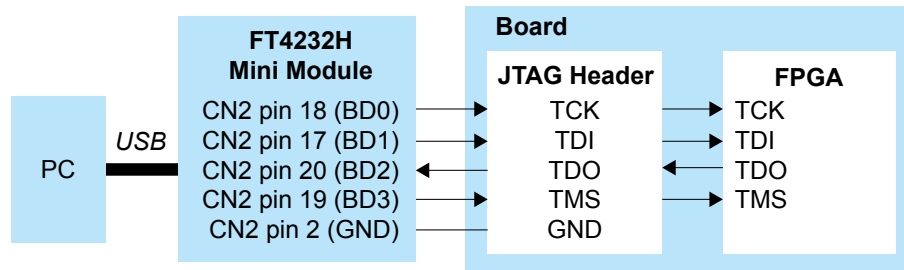
**Note:** This figure uses the CRESET\_N and SS\_N pins in addition to the standard JTAG pins. However, this setup is only needed for normal JTAG programming. You can use the standard 4 JTAG pins and any cable for programming with SPI Active using JTAG Bridge mode or for other JTAG functions.

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

*Figure 40: Connect FT2232H Mini Module to JTAG Pins*



*Figure 41: Connect FT4232H Mini-Module to JTAG Pins*



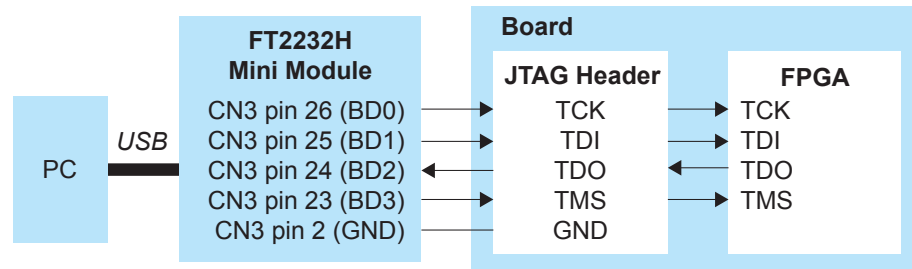
## JTAG Programming Connections (Titanium FPGAs)

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

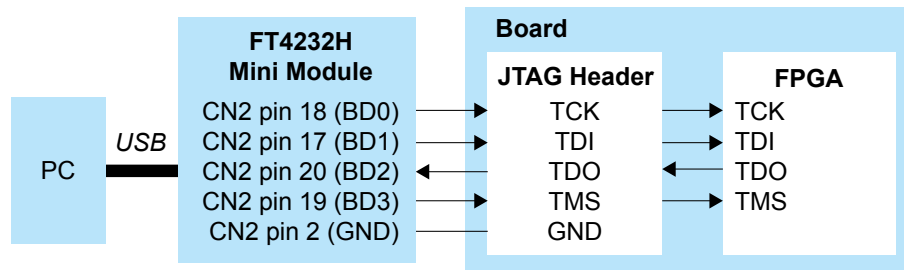
## Connecting a JTAG Mini Module

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

*Figure 42: Connect FT2232H Mini-Module to JTAG Pins*



*Figure 43: Connect FT4232H Mini-Module to JTAG Pins*



# Appendix: Efinity Project Files

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

## Efinity Source Files for Version Control

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

- **<project>.xml**
- **<project>.peri.xml**
- **<project>.sdc**
- **debug\_profile.wizard.json**
- **dbg\_top.v**
- **<module>.v**
- **settings.json**

## Bitstream Generation

*<project>.hex*

<b>In GUI</b>	Result pane > Bitstream menu
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the bitstream generation step
<b>Design source?</b>	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*

<b>In GUI</b>	Result pane > Bitstream menu
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the bitstream generation step
<b>Design source?</b>	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*

<b>In GUI</b>	Result pane > Bitstream menu
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the bitstream generation step
<b>Design source?</b>	No

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

## Debugger

*debug\_profile.wizard.json*

<b>In GUI</b>	Project Editor > Debugging tab
<b>In file system</b>	<i>&lt;project&gt;</i>
<b>Created by</b>	Efinity Debug Wizard
<b>Design source?</b>	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 64.

*dbg\_top.v*

<b>In GUI</b>	Result pane > Debugger menu
<b>In file system</b>	<i>&lt;project&gt;/outflow/work_dbg</i>
<b>Created by</b>	Efinity Debug Wizard
<b>Design source?</b>	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*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/outflow/work_dbg</i>
<b>Created by</b>	Efinity Debugger
<b>Design source?</b>	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 60 or **Logic Analyzer Debug Core** on page 62 for more information.

## Interface Designer

### *<project>.peri.xml*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;</i>
<b>Created by</b>	Interface Designer when you create an interface
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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`

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<code>&lt;project&gt;/outflow</code>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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`

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<code>&lt;project&gt;/outflow</code>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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`

<b>In GUI</b>	<b>Result pane &gt; Interfaces menu</b>
<b>In file system</b>	<code>&lt;project&gt;/outflow</code>
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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 constraints:

- *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

<b>In GUI</b>	Result pane > Interfaces menu
<b>In file system</b>	<project>/outflow
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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

<b>In GUI</b>	Result pane > Interfaces menu
<b>In file system</b>	<project>/outflow
<b>Created by</b>	Interface Designer when generating constraints
<b>Design source?</b>	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.

## IP

### *<module>.define*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/ip/&lt;module&gt;</i>
<b>Created by</b>	IP Configuration wizard
<b>Design source?</b>	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*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/ip/&lt;module&gt;</i>
<b>Created by</b>	IP Configuration wizard
<b>Design source?</b>	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 33 for instructions on using this file to create another instance of an IP core.

### *<module>\_tpl.v*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/ip/&lt;module&gt;</i>
<b>Created by</b>	IP Configuration wizard
<b>Design source?</b>	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>\_tpl.vhd*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/ip/&lt;module&gt;</i>
<b>Created by</b>	IP Configuration wizard
<b>Design source?</b>	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*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;/ip/&lt;module&gt;</i>
<b>Created by</b>	IP Configuration wizard
<b>Design source?</b>	Yes

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

## Placement

*<project>.place*

<b>In GUI</b>	<b>Result pane &gt; Placement menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the placement step
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Placement menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the placement step
<b>Design source?</b>	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).

*<project>.place.out*

<b>In GUI</b>	<b>Result pane &gt; Placement menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the placement step
<b>Design source?</b>	No

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

## Project

### *<project>.sdc*

<b>In GUI</b>	<b>Project pane &gt; Constraint menu</b>
<b>In file system</b>	<i>&lt;project&gt;</i>
<b>Created by</b>	User defined
<b>Design source?</b>	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 the [Efinity Timing Closure User Guide](#) for the full details.

### *<project>.xml*

<b>In GUI</b>	-
<b>In file system</b>	<i>&lt;project&gt;</i>
<b>Created by</b>	Efinity software when you create a project
<b>Design source?</b>	Yes

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

## Routing

### *<project>.pnr.rpt*

<b>In GUI</b>	<b>Result pane &gt; Routing menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the routing step
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Routing menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the routing step
<b>Design source?</b>	No

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

`<project>.route.out`

<b>In GUI</b>	Result pane > Routing menu
<b>In file system</b>	<code>&lt;project&gt;/outflow</code>
<b>Created by</b>	Efinity software during the route step
<b>Design source?</b>	No

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



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

`<project>.timing.rpt`

<b>In GUI</b>	Result pane > Routing menu
<b>In file system</b>	<code>&lt;project&gt;/outflow</code>
<b>Created by</b>	Efinity software during the route step
<b>Design source?</b>	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](#).

## Synthesis

### *<project>.map.v*

<b>In GUI</b>	<b>Result pane &gt; Synthesis menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the synthesis step
<b>Design source?</b>	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.rpt*

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

The Efinity software creates this file during the synthesis step. This file contains all of the reporting for the synthesis step. It give the top-level 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*

<b>In GUI</b>	<b>Result pane &gt; Synthesis menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow</i>
<b>Created by</b>	Efinity software during the synthesis step
<b>Design source?</b>	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*

<b>In GUI</b>	<b>Result pane &gt; Synthesis menu</b>
<b>In file system</b>	<i>&lt;project&gt;/outflow/work_syn</i>
<b>Created by</b>	Efinity software during the synthesis step
<b>Design source?</b>	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 tabular view of the modules and the resources used. You can filter the list using the filter field at the top.
- The **Text** tab shows a plain text view of the same data.

# Appendix: Shortcuts

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

**Table 30: Shortcuts**

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



# Appendix: Icon List

## Dashboard Icons

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

Toggle Automated Flow

## General Icons

- Filter
- Search
- General Tool Preferences
- Help
- Exit

## Property Icons

- Expand All
- Collapse All
- Toggle Properties Panel

## Log Icons

- Message Browser
- Log Browser

## Project Icons



















- Open Project
- New Project
- Close project
- Edit Project
- Choose Project Directory
- Remove a File from a Project
- Add a file to a Project
- Import Design & Constraint Files

- Save
- Results are out of sync with settings









## Flow Icons

- Run Complete Flow
- Synthesize
- Place
- Route
- Place & Route
- Stop
- Generate Bitstream File

















## Floorplan Icons

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












## Timing & Console Icons

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











## Interface Designer Icons

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




















## Tools Icons

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

## Programmer Icons

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

## Debugger Icons

 Debugger	 Add Probe	 Add Net
 Debug Wizard	 Add Source	 Remove Net
 Add DebugCore	 Remove Probe or Source	 Active High
 Remove DebugCore	 Import DebugProfile	 Active Low
 Connect Debugger	 Refresh USBTarget	 Toggle High
 Disconnect Debugger	 Select Waveform File	 Toggle Low
	 Advanced Device Configuration Status	

# Revision History

*Table 31: Document Revision History*

Date	Version	Description
August 2022	10.0	<p>Added new project-based programming option for 4-byte addressing.</p> <p>Updated the available options for the <b>Project Editor &gt; Place and Route</b> tab. (DOC-889)</p> <p>Clarified the instructions for instantiating debug cores. (DOC-883)</p> <p>Clarified that when using internal reconfiguration you must use <b>Programmer &gt; Combine Multiple Image Files &gt; Image Type &gt; Internal Flash Image</b> option. (DOC-874)</p> <p>Added topic on verifying configuration with the Programmer.</p> <p>When editing the bitstream header, do not remove any auto-generated data or the Programmer may not recognize the bitstream.</p> <p>Removed support for C232HM-DDHSL-0 cable. (DOC-860)</p> <p>Added a topic on the concurrent debug feature.</p> <p>Updated supported IP cores.</p> <p>Updated Installing USB Drivers topics.</p> <p>Updated supported IP cores.</p>
June 2022	9.2	<p>Pointed to new sourceforge location for GTKWave download. (DOC-797)</p>
April 2022	9.1	<p>Added Program using a JTAG Bridge topic.</p> <p>Added topic on combining a bitstream and other data into a single file for programming.</p> <p>Re-organized topics about working with bitstreams.</p> <p>Moved topics on installing USB drivers and connecting programming hardware to the appendix.</p> <p>The minimum operating frequency of the debug cores is 2 times the JTAG TCK frequency. (DOC-754)</p> <p>Added CORDIC core to the list of supported IP (included with Efinity patch v2021.2.323.2.18).</p>
December 2021	9.0	<p>Added Efinity Hardware Server documentation. (DOC-598)</p> <p>Added support for FTDI FT4232H Mini Module. (DOC-597)</p> <p>Added the JTAG USERCODE option to the Project-Based Programming Options topic.</p> <p>With the Efinity software v2021.2 and higher, you <b>must</b> use <b>.hex</b> for SPI and <b>.bit</b> for JTAG. (DOC-638)</p> <p>When importing an IP configuration <b>.json</b> file, specify the module name in the IP Configuration wizard. (DOC-611)</p> <p>Updated machine memory requirements (RAM).</p> <p>You may need to re-compile when upgrading from an older version.</p> <p>Added appendix of project file definitions.</p>
November 2021	8.2	<p>Added instructions on using the Titanium bitstream security features.</p> <p>Added instructions for using the Efinity SVF Player.</p> <p>Described how to export a bitstream to serial vector format (<b>.svf</b>).</p> <p>When using the stand-alone Programmer on 64-bit Windows, install both the x86 and x64 libraries. (DOC-576)</p> <p>Added instructions for importing IP cores. (DOC-584)</p>

Date	Version	Description
October 2021	8.1	<p>Added topic on flash programming modes.</p> <p>Added topic on the Titanium configuration status registers. (DOC-487)</p> <p>Added note that FTDI Chip FT2232H Mini Module supports 3.3 V I/O voltage only. (DOC-495)</p> <p>Added description of command to convert bitstream files from <b>.hex</b> to <b>.bin</b> to Exporting to Raw Binary Format topic. (DOC-527)</p> <p>JRE required for running the DMA Controller in the IP Manager. (DOC-549)</p> <p>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)</p>
June 2021	8.0	<p>Added support for Titanium family.</p> <p>Supported Ubuntu version is v18.04 or higher. v16.04 is end of life. (DOC-433)</p> <p>Added the Java runtime environment as a software requirement for configuring the Sapphire SoC in the IP Manager.</p> <p>Described more detail on the Enable Initialized Memory in User RAMs option in the <b>Project Editor &gt; Bitstream Generation</b> tab. (DOC-458)</p> <p>Added table of IP cores supported by family.</p> <p>Updated the FTDI command-line programming topic. Added the command-line programmer configuration mode options. (DOC-430)</p>
January 2021	7.1	Corrected JTAG chain file code example. (DOC-368)
December 2020	7.0	<p>Added a new chapter on using the IP Manager.</p> <p>Added instructions on using VHDL libraries.</p> <p>Explained how to resize the Project, Netlist, and Result panes.</p> <p>Described the context-sensitive menus in the Project, Netlist, and Result panes.</p> <p>Added requirement to install the Microsoft Visual C++ 2015 x86 runtime library for the standalone Programmer. (DOC-315)</p> <p>Updated instructions for performing JTAG programming at the command line. (DOC-323)</p> <p>Corrected JTAG Mini Module pin names for T4, T8, T13, T20BGA256, and T20BGA169 connection setup.</p> <p>Clarified Undefined clock domain signals in the Debug Wizard.</p> <p>Added table of files shown in the Result pane. (DOC-277)</p> <p>Interface scripting file now supports PLL.</p>
November 2020	6.1	<p>Updated instructions on installing Windows USB drivers.</p> <p>Added FTDI cable and module connection for T20BGA400.</p> <p>Added JTAG device IDs for T20BGA324 and T20BGA400.</p> <p>Removed the FTDI2232 from About USB Drivers topic making the description applicable to other FTDI chips.</p> <p>Corrected the command for using <code>--pgm_opts</code> with the command-line programmer.</p>

Date	Version	Description
June 2020	6.0	<p>Updated for v2020.1 release.</p> <p>Windows 7, Red Hat v6, and CentOS v6 no longer supported.</p> <p>Removed the chapters on SDC constraints and Tcl commands. This content is now in the Efinity Timing Closure User Guide.</p> <p>Added a topic on Efinity synthesis.</p> <p>Added a topic on project migration.</p> <p>Updated Programmer content to reflect new GUI and features.</p> <p>Consolidated and updated content on installing USB drivers for boards, C232HM-DDHSL-0 cable and FTDI FT2232H module.</p> <p>Added support for FTDI FT2232H module for JTAG programming.</p> <p>Added a topic on the various ways to view messages and logs.</p> <p>Added topic on the Interface Designer/s Resource View.</p> <p>Added a topic on using an API for scripting an interface design.</p> <p>Added topic on Interface Scripting File (.isf).</p>
December 2019	5.0	<p>Updated for v2019.3 release.</p> <p>Added chapter on using the Debugger.</p> <p>Added explanation that 2 unassigned pairs of LVDS pins should be located between and GPIO and LVDS pins in the same bank.</p>
August 2019	4.5	<p>Updated for v2019.2 release.</p> <p>Added information on enhanced Resource Assigner.</p> <p>Added information on JTAG programming.</p> <p>Added command-line instructions for using the Windows <b>efx_run.bat</b> file.</p>
April 2019	4.4	<p>Updated for v2019.1 release.</p> <p>Added information on new project manager capabilities.</p> <p>Updated <code>set_false_path</code> usage.</p>
January 2019	4.3	<p>Updated for v2018.4 release.</p> <p>Added more information on simulation and waveform viewing.</p> <p>Added instructions for installing Windows USB driver.</p> <p>Updated Programming information.</p>
October 2018	4.2	<p>Added a note pointing to AN 006: Configuring Trion FPGAs for more information about using multiple images and daisy chaining for configuration.</p> <p>Added Python 3 to the software requirements list as an option. For Windows, if you do not have a full version of Python, the <b>.py</b> extension may not be correctly associated with Python.</p>
June 2018	4.1	<p>Removed Python requirement; as of this release, Python is included with the software.</p> <p>Added the requirement that Windows users install the Microsoft Visual C++ 2015 x64 runtime library.</p>
April 2018	4.0	<p>Updated for v2018.0 release.</p> <p>Added "Constraining Logic and Assigning Pins" topic, which replaces section on fine-tuning your design.</p> <p>Updated information on device configuration.</p>
November 2017	3.1	Minor updates.
May 2017	3.0	<p>Updated for v2017.0 release.</p> <p>Described new Floorplan Editor tools.</p> <p>Updated SDC constraint information.</p>

Date	Version	Description
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.