



Efinity[®] Command-Line Interface User Guide

UG-EFN-CLI-v1.0
November 2025
www.efinixinc.com



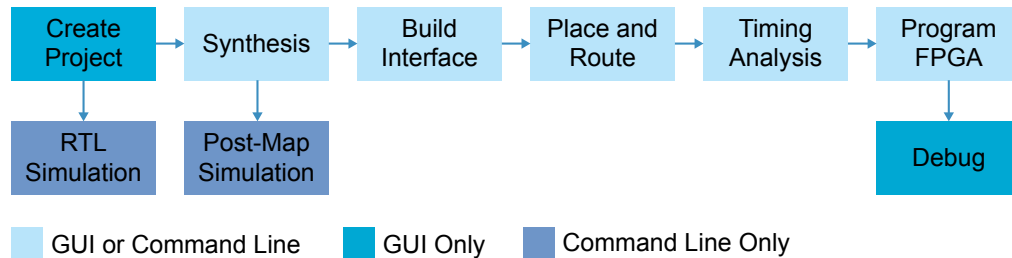
Contents

Introduction.....	3
Using Efinity Command-Line Scripts.....	3
Efinity Quick Start.....	4
Run the Flow from the Command Line.....	5
efx_run Script.....	5
--flow Option.....	6
Place and Route Multi-Run Script.....	7
Perform Static Timing Analysis at the Command Line.....	8
Simulating.....	10
Simulation Models.....	11
Changing the Default Testbench Names.....	12
Simulate with the iVerilog Simulator.....	13
View Waveforms.....	13
Simulate with the ModelSim Simulator.....	14
Simulate with the NCSim Simulator.....	15
Simulate with the Aldec Active HDL or Riviera-PRO Simulator.....	16
Configure FPGAs at the Command Line.....	17
Launch the Debugger from the Command Line.....	17
Using the Command-Line Programmer.....	17
FTDI Programming at the Command Line.....	18
Use the SVF Player at the Command Line.....	21
Identifying FTDI URLs.....	22
Run the BRAM Initial Content Updater from the Command Line.....	23
Using the Block RAM Resource Estimator.....	24
Manage Bitstream Files at the Command Line.....	25
Generate Bitstream Checksums at the Command Line.....	25
Convert to Intel Hex Format at the Command Line.....	25
Combine Bitstreams at the Command Line.....	26
Encrypt or Sign Bitstreams from the Command Line.....	27
Using the Efinity Database Export Utility (Beta).....	28
Export JTAG Operations at the Command Line.....	28
Appendix: Additional Flow Options.....	31
Synthesis Options.....	31
Place and Route Options.....	33
Programming Options.....	34
Where to Learn More.....	37
Revision History.....	38

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 interface to perform simulation and automate the flow using scripts.

Figure 1: Design Flow Overview



This document describes scripts that can be used to run the flow, configure FPGAs, and perform various other operations with the Efinity software at the command-line interface.



Note: Within the Efinity source files, you may find additional internal scripts that are not referenced in this document. Efinix developers recommend against using command-line scripts that are not covered by official documentation.

Using Efinity Command-Line Scripts

This topic overviews the format used by Efinity command-line scripts.

Script input at the command line includes the script name followed by positional arguments and/or options. Multiple arguments or options may be used depending on the script.

In each script topic, the script is listed with all arguments and options, including those that are not required. Optional inputs are listed in brackets. When an option is followed by the option name in capital letters, it takes an additional input. For example:

```
efx_run.py project [--flow FLOW]
```

efx_run.py is the script name, **project** is a positional argument, **--flow** is an option, and **FLOW** is the option input.

The following example command runs the full flow (**--flow full**) using the **efx_run.py** script on the **helloworld** project:

Example: Command Line Input

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

Efinity Quick Start

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



Warning: Do not use non-ASCII characters in the Efinity project paths.

Windows

Set up your environment and PATH:

```
bin\setup.bat
```

Launch the Efinity GUI from the command line:

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

Run Efinity from the command line:

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

For command-line help:

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

Linux

Set up your environment and PATH:

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

Launch the Efinity GUI from the command line:

```
efinity
```

Run Efinity from the command line:

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

For command-line help:

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

Run the Flow from the Command Line

The following topics overview the software flow at the command line.

efx_run Script

You can run the complete software flow from the command line using the **efx_run.py** Python 3 script.

```
efx_run.py design [--help] [--prj PRJ] [--flow FLOW] [--tcl_script TCL SCRIPT]
[-v EXTRA_FILES [EXTRA_FILES ...] | --flist FLIST] [--tb TB EXTRA_FILES] [--tb_top TB_TOP]
[--dir SOURCE_DIR] [--map_opts MAP_OPTS [MAP_OPTS ...]] [--pt_opts PT_OPTS [PT_OPTS ...]]
[--pnr_opts VPR_OPTS [VPR_OPTS ...]] [--pgm_opts PGM_OPTS [PGM_OPTS ...]] [--modelsim]
[--aldec] [--ncsim] [--res_compare] [--output_dir OUTPUT_DIR] [--work_dir WORK_DIR]
[--timeout TIMEOUT] [--un_flow] [--physical_syn_input PHYSICAL_SYN_INPUT]
```



Note: You can use **efx_run.bat** to run the **efx_run.py** script in the Windows Command Prompt.

Table 1: efx_run.py Positional Arguments

Argument	Description
design	Efinity project XML file name.

Table 2: efx_run.py Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--prj	N/A	None	Utilize alternate project XML file.
--flow	-f	map, interface, pnr, pgm, compile, program, rtlsim, mapsim, pnrsim, full, ptsimrtl, ptsimfc, sta_tclsh, setup_efxlib	Tool flow. Default: compile See --flow Option on page 6 for detailed input descriptions.
--tcl_script	-t	Filename	Optional Tcl script to be used with the <code>sta_tclsh</code> flow. A Tcl interactive flow will be started if omitted.
N/A	-v	Filename	Additional HDL files to map.
--flist	N/A	File path	Path to text file listing additional HDL source files.
--tb	N/A	Filename	Testbench HDL simulation files. For simulation only.
--tb_top	N/A	Module name	Top-level testbench module name. For simulation only.
--dir	N/A	File path	Search for path to source files.
--map_opts	N/A	Command options	Pass option string to efx_map directly. See Synthesis Options on page 31 for detailed options.

Option (Long)	Option (Short)	Input	Description
--pt_opts	N/A	Command options	Pass option string to the Interface Designer directly.
--pnr_opts	N/A	Command options	Pass option string to efx_pnr directly. See Place and Route Options on page 33 for detailed options.
--pgm_opts	N/A	Command options	Pass option string to efx_pgm directly. See Programming Options on page 34 for detailed options.
--modelsim	-m	None	Use the ModelSim or QuestaSim simulators.
--aldec	-N/A	None	Use the Aldec Active-HDL or Riviera-PRO simulators.
--ncsim	N/A	None	Use the NCSim simulator.
--output_dir	N/A	File path	Output directory.
--work_dir	N/A	File path	Work directory.
--timeout	N/A	Number	Cumulative timeout for all stages in seconds.
--un_flow	N/A	None	Flag for unified netlist flow.
--physical_syn_input	N/A	File path	Path to physical synthesis request file.

--flow Option

The **efx_run.py** script is used to run the flow at the command line. You can use **--flow** option to direct the software through set actions in the flow.

Table 3: Compilation Commands

Command	Description
--flow compile	Default. Performs synthesis, place and route, and generates the bitstream hex file.
--flow map	Performs synthesis.
--flow pnr	Performs place and route.
--flow full	Runs the full flow, including RTL and post-synthesis simulation.
--flow interface	Generates the interface constraint files.
--flow sta_tclsh	Runs timing analysis. If the --tcl_script option is passed, the timing analysis commands in the script are processed and the flow completes. If no script is specified, enters an interactive Tcl console. See Perform Static Timing Analysis at the Command Line on page 8 for details.

Table 4: Simulation Commands

Command	Description
--flow rtlsim	Performs RTL simulation on the design's source files.
--flow mapsim	Performs simulation with the post-synthesis netlist file.

Command	Description
--flow ptsimrtl	Simulate the interface after generating interface constraints with the unified netlist option (--un_flow).
--flow ptsimfc	Perform full-chip simulation after generating interface constraints with the unified netlist option (--un_flow).
--flow setup_efxlib	Creates the initial elaborated library of Efinix primitives for use in VHDL simulation.

Table 5: Programming Commands

Command	Description
--flow pgm	Creates the bitstream hex file used to configure the device.
--flow program	Programs the target device. See Using the Command-Line Programmer on page 17 for details.

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

Example: Run the Complete Flow from the Command Line

Linux:

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

Windows:

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

Place and Route Multi-Run Script

The **efx_run_pnr_sweep.py** Python 3 script can be used to run place and route multiple times in succession.

```
efx_run_pnr_sweep.py [--help] [--force] [--timingSumReport [TIMINGSUMREPORT]]
  project_XML {sweep_seeds, sweep_opt_levels} [--start_seed [START_SEED]]
  [--num_seeds [NUM_SEEDS]] | --end_seed [END_SEED]]
```



Note: You can use **efx_run_pnr_sweep.bat** to run the **efx_run_pnr_sweep.py** script in the Windows Command Prompt.



Learn more: Refer to Place and Route Options in the [Efinity Timing Closure User Guide](#) for more information on when to use this script.

Table 6: efx_run_pnr_sweep.py Positional Arguments

Argument	Description
project XML	The Efinity project XML file.
sweep_seeds	Enable seeds sweeping. Only one sweep algorithm can be chosen between <code>sweep_seeds</code> and <code>sweep_opt_levels</code> for each run. Default: sweep 10 seeds (0-9)
sweep_opt_levels	Enable the optimization levels sweeping. Only one sweep algorithm can be chosen between <code>sweep_seeds</code> and <code>sweep_opt_levels</code> for each run.

Table 7: *efx_run_pnr_sweep.py* Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--force	-f	None	Always run the complete compile flow. Default: false.
--timingSumReport	N/A	Filename	Specify the timing summary report file. Default: timing.sum.rpt .
--start_seed	N/A	Number	The seed start value. For use with the <code>sweep_seeds</code> algorithm. Default: 0
--num_seeds	N/A	Number	Number of seeds that will sweep. For use with the <code>sweep_seeds</code> algorithm. Default: 10
--end_seed	N/A	Number	The seed end value. For use with the <code>sweep_seeds</code> algorithm. Default: -1

Perform Static Timing Analysis at the Command Line

You can use the Tcl Console at the command line to run timing analysis on a fully compiled design. (If you have not compiled yet, the software issues an error message.)

To enter the interactive Tcl Console, use the `sta_tclsh` flow option with the **efx_run.py** Python 3 script. The terminal or command prompts displays an interactive Tcl Console similar to `tclsh`. You can enter Tcl commands and expressions for evaluation.

You can also run the Tcl Console in batch mode. In this mode you also specify a script name. The Tcl Console runs the script and exists after executing it.

Table 8: *Command-Line TCL Console Options*

Option (Long)	Option (Short)	Input	Description
--flow	-f	sta_tclsh	Tool flow. See --flow Option on page 6 for alternative flow options.
--tcl_script	-t	Filename	TCL script.

Example: Command-Line TCL Console

Linux:

```
efx_run.py --flow sta_tclsh helloworld.xml
```

Windows:

```
efx_run.bat --flow sta_tclsh helloworld.xml
```

Example: Command-Line TCL Console

Linux:

```
efx_run.py --flow sta_tclsh --tcl_script example_report.tcl
```

Windows:

```
efx_run.bat --flow sta_tclsh --tcl_script example_report.tcl
```

Simulating

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



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

Simulation involves the following steps:

1. Perform behavioral RTL simulation to ensure that the RTL design matches your testbench functionality. You can include multiple Verilog HDL design files. Use the `--flow rtlstim` flag.
2. Run the mapper to synthesize your design using the `--flow map` flag. The software creates the `<project name>.map.v` file in the **outflow** directory, which you use for post-synthesis simulation.
3. Perform post-map simulation using the top-level testbench and the `.map.v` file using the `--flow mapsim` flag.
4. After generating the interface constraints with the unified netlist option you can simulate the interface logic:
 - a. Simulate the core and portion of the interface design defined in the source RTL using the `--flow ptsimrtl` flag.
 - b. Simulate the core design and all of the interface logic defined in the source RTL or the Interface Designer using the `--flow ptsimfc` flag. This requires a new testbench targeted at the full chip module called `<project>~chip`.

The following example shows the commands for these three steps:

Example: Simulating at the Command Line

Linux:

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

Windows:

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

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

Simulation Models

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

Table 9: Core Primitive Simulation Models

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_ADD	Simple Full Adder	✓	✓	✓	efx_add.v
EFX_COMB4	Simple 4-Input LUT ROM plus Simple Adder		✓	✓	efx_comb4.v
EFX_DPRAM5K	5 Kbit True-Dual-Port RAM Block	✓			efx_dpram5k.v
EFX_DPRAM10	10 Kbit True-Dual-Port RAM Block		✓	✓	efx_dpram10.v
EFX_DSP12	Quad-Mode 4 x 4 DSP Block		✓	✓	efx_dsp12.v
EFX_DSP24	Dual-Mode 8 x 8 DSP Block		✓	✓	efx_dsp24.v
EFX_DSP48	Full Function DSP Block		✓	✓	efx_dsp48.v
EFX_FF	D Flip-flop with Clock Enable and Set/Reset Pin	✓	✓	✓	efx_ff.v
EFX_GBUFCE	Global Clock Buffer	✓	✓	✓	efx_gbufce.v
EFX_LUT4	Simple 4-Input LUT ROM	✓	✓	✓	efx_lut4.v
EFX_MULT	18 x 18 Multiplier	✓			efx_mult.v
EFX_RAM_5K	5 Kbit RAM Block	✓			efx_ram_5k.v
EFX_RAM10	10 Kbit RAM Block		✓	✓	efx_ram10.v
EFX_SRL8	8-Bit Shift Register		✓	✓	efx_srl8.v

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

Table 10: Interface Primitive Simulation Models

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

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_CLKOUT	Clock Output Buffer	✓	✓	✓	EFX_CLKOUT.v
EFX_FPLL_V1	Fractional PLL		✓	✓	EFX_FPLL_V1.v
EFX_GPIO_V1	Basic GPIO	✓			EFX_GPIO_V1.v
EFX_GPIO_V2	GPIO with Double Data I/O Function	✓			EFX_GPIO_V2.v
EFX_GPIO_V3	HVIO and HSIO Used as Single-Ended GPIO		✓	✓	EFX_GPIO_V3.v
EFX_GPIO_DIFF_V3	HVIO and HSIO Used as Differential GPIO		✓	✓	EFX_GPIO_DIFF_V3.v
EFX_IBUF	Single-Ended Input Buffer	✓	✓	✓	EFX_IBUF.v
EFX_IDDIO	Input Double Data I/O Register	✓	✓	✓	EFX_IDDIO.v

Primitive	Description	Trion	Titanium	Topaz	Filename
EFX_IOREG	Single-Ended Bi-Directional Register	✓	✓	✓	EFX_IOREG.v
EFX_IO_BUF	Single-Ended Bi-Directional Buffer	✓	✓	✓	EFX_IO_BUF.v
EFX_IREG	Single-Ended Input Register	✓	✓	✓	EFX_IREG.v
EFX_JTAG_CTRL	JTAG Interface	✓	✓	✓	EFX_JTAG_CTRL.v
EFX_JTAG_V1	JTAG User TAP Interface	✓	✓	✓	EFX_JTAG_V1.v
EFX_OBUF	Single-Ended Output Buffer	✓	✓	✓	EFX_OBUF.v
EFX_ODDIO	Output Double Data I/O Register	✓	✓	✓	EFX_ODDIO.v
EFX_OREG	Single-Ended Output Register	✓	✓	✓	EFX_OREG.v
EFX_OSC_V1	Oscillator	✓			EFX_OSC_V1.v
EFX_OSC_V3	Oscillator		✓	✓	EFX_OSC_V3.v
EFX_PLL_V1	Simple PLL	✓			EFX_PLL_V1.v
EFX_PLL_V2	Advanced PLL	✓			EFX_PLL_V2.v
EFX_PLL_V3	Full-Featured PLL		✓	✓	EFX_PLL_V3.v
EFX_LVDS_RX_V1	LVDS Reciever	✓			EFX_LVDS_RX_V1.v
EFX_LVDS_RX_V2	LVDS Receiver		✓	✓	EFX_LVDS_RX_V2.v
EFX_LVDS_TX_V1	LVDS Transmitter	✓			EFX_LVDS_TX_V1.v
EFX_LVDS_TX_V2	LVDS Transmitter		✓	✓	EFX_LVDS_TX_V2.v
EFX_LVDS_BIDIR_V2	LVDS Transmitter/Receiever		✓	✓	EFX_LVDS_BIDIR_V2.v

Changing the Default Testbench Names

The simulation flow assumes that:

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

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

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

Example: Changing Default Names

Linux:

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

Windows:

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



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

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

Simulate with the iVerilog Simulator

By default, the Efinity® software calls the iVerilog simulator. Use the `--flow rtlsim|mapsim` flag. This free simulator has limitations: it cannot decrypt code (such as IP from the IP Manager), and it does not support SystemVerilog.



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



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

For example, the commands to simulate are:

Example: Simulate with iVerilog

Linux:

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

Windows:

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

The simulator responds with

- PASS if the simulation is successful.
- a Python exception warning if the simulation is unsuccessful.

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

View Waveforms

To use GTKWave to view a waveform:

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

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

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

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

Simulate with the ModelSim Simulator

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



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

For example, the commands to simulate are:

Example: Simulate with ModelSim

Linux:

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

Windows:

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

The simulator responds with

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

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

Simulate with the ModelSim GUI

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

1. Create a new macro file `<project name>.do` in your project directory.
2. For behavioral simulation, define your workspace and include your source code.
3. For post-synthesis simulation, define your workspace, include the post-mapping synthesis file, and include the simulation models for Efinix primitives.
4. Add the `vsim -t ps <work space>.<test bench module name>` command to start simulation in the ps timeframe.
5. Add the `run <number>us` command to generate a waveform up to `<number> μs`.
6. Run the ModelSim software in the Transcript console.
7. Change to the project root directory.
8. Use the `do` command to execute the macro (`do <name>.do`).
9. Add signals to the waveform in the **Objects** tab.
10. View the simulation result in the **Wave** tab.

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

Figure 2: 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 3: 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 (`<flow>.rtl.simlog` and `<flow>.map.simlog`) and error messages (`<flow>.log`) in your project's **outflow** directory.

Simulate with the Aldec Active HDL or Riviera-PRO Simulator

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



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

For example, the commands to simulate are:

Example: Simulate with Aldec Active HDL or Riviera-PRO

Linux:

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

Windows:

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



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

Linux:

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

Windows:

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

The simulator responds with

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

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

Configure FPGAs at the Command Line

The following topics overview FPGA configuration at the command line.

Launch the Debugger from the Command Line

You can quickly launch the Efinity Debugger GUI from the command line using the **efinity_dbg.sh** script.

```
efinity_dbg.sh [--help] [--project_home PROJECT_HOME] [--cls {DbgGui}]
               [--profile_file PROFILE_FILE] [--no-save] [--device DEVICE] [--family FAMILY]
```



Note: Use **efinity_dbg.bat** to run the **efinity_dbg.sh** script in the Windows Command Prompt.

Table 11: *efinity_dbg.sh* Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--project_home	N/A	File path	Path of the project directory when starting the Debugger GUI from the Efinity GUI. In standalone mode, the debug profile JSON will be saved in the current directory.
--cls	N/A	DbgGui	Class for main GUI to launch. Default: DbgGui
--profile_file	N/A	Filename	Debug profile JSON file.
--no_save	N/A	None	Do not save the debug profile when quit.
--device	N/A	Device name	Device name as per the Efinity project.
--family	N/A	Family name	Device family.

Using the Command-Line Programmer

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

Example: Command-Line Programmer

Linux:

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

Windows:

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

Options

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

Table 12: --pgm_opts Modes

Mode	Description
active	SPI Active configuration.
passive	SPI Passive configuration.
jtag	JTAG programming. See the Efinity Programmer User Guide for more information about programming with the JTAG interface.
jtag_bridge	SPI Active using JTAG bridge mode.
jtag-bridge_x8	SPI Active x8 using JTAG bridge mode (used with two flash devices). ⁽¹⁾

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

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

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



Note: See [Programming Options](#) on page 34 for more programming options.

FTDI Programming at the Command Line

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

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

Table 13: ftdi_pgm.py Positional Arguments

Argument	Description
input_file	HEX file generated from efx_pgm.

Table 14: ftdi_pgm.py Options

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

⁽¹⁾ Used with two flash devices. Only supported in some Titanium Topaz FPGAs. Refer to the data sheet for the modes your FPGA supports.

Option (Long)	Option (Short)	Input	Description
--mode	-m	passive, active, jtag, jtag_chain, erase_flash, read_flash, jtag_bridge, jtag_bridge_x8	<p>Programming mode.</p> <p>See the Efinity Programmer User Guide.</p> <p>In Efinity software versions prior to v2025.1, the <code>jtag_bridge</code> and <code>jtag_bridge_new</code> options were named <code>jtag_bridge_new</code> and <code>jtag_bridge_x8_new</code>, respectively. (2)</p> <p>To use the JTAG bridge modes, you must have already configured the with the JTAG SPI flash loader.</p> <p>The Efinity software v2023.2 and higher includes pre-built flash loader.bit files in <code><installation directory>/pgm/fli/<family></code>. Refer to the JTAG SPI Flash Loader Core User Guide for information on using the legacy flash loader.</p>
--output_file	-o	Filename	Output file used for <code>read_flash</code> mode.
--url	-u	URL	FTDI URL (see Identifying FTDI URLs on page 22).
--aurl	-a	URL	Alternative URL (<i>Deprecated</i>).
--xml	-x	Filename	XML file for JTAG programming.
--num	-n	Number	Chip target number for JTAG chain programming.
--board_profile	-b	Generic Board Profile Using FT232, Digilent JTAG-HS3, FireAnt Development Board, Generic Board Profile Using FT2232H, ISX Programming Cable, Titanium Ti180J484 Dev Board, Titanium Ti180M484 Development Kit, Generic Board Profile Using FT4232, JinChen Programming Cable, TJ180A484S Development Kit, Xyloni Development Board, Generic Board Profile Using FT4234HA	Name of the board profile used.
--address	N/A	Hex number	Starting flash address for flash read and write operations.
--num_bytes	N/A	Number	Number of bytes to erase or read. For modes <code>erase</code> and <code>read</code> only.
--burst_size	N/A	Number	Individual read or write burst size in multiples of 256 bytes. For legacy JTAG bridge modes only (<code>jtag_bridge</code> and <code>jtag_bridge_x8</code>).
--jtag_bridge_mode	N/A	Erase, write, erase_and_write, read, all, all_no_erase	JTAG bridge programming mode.

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

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

Linux Examples

To program in Linux:

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

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

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

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

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

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

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

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

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

Windows Examples

To program in Windows:

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

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

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

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

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\ftdi_program.py <filename>.bit
-m jtag --url ftdi://ftdi:4232h:FT5ECP6E/1 --aur1 ftdi://ftdi:4232h:FT5ECP6E/1
```

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

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

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

```
%EFINITY_HOME%\bin\python3
%EFINITY_HOME%\pgm\bin\ftdi_program.py <filename>.bit
-m jtag --url ftdi://ftdi:2232h:FT5ECP6E/1 --aur1 ftdi://ftdi:2232h:FT5ECP6E/1
```

Use the SVF Player at the Command Line

You can run the Efinity SVF Player from the command line with the **svf_player.py** script.

```
ftdi_pgm.py [--help] [--ftdi FTDI] [--board_profile BOARD_PROFILE] [input_file]
```

Table 15: *ftdi_pgm.py* Positional Arguments

Argument	Description
input_file	Input SVF File

Table 16: *ftdi_pgm.py* Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--ftdi	-f	URL	Manually specify FDTI URL (see Identifying FTDI URLs on page 22).
N/A	-b	Generic Board Profile Using FT232, Digilent JTAG-HS3, FireAnt Development Board, Generic Board Profile Using FT2232H, ISX Programming Cable, Titanium Ti180J484 Dev Board, Titanium Ti180M484 Development Kit, Generic Board Profile Using FT4232, JinChen Programming Cable, TJ180A484S Development Kit, Xyloni Development Board, Generic Board Profile Using FT4234HA	Name of the board profile used.

Example: Using the Efinity SVF Player

```
cd %EFINITY_HOME%\debugger\svf_player\bin\efx_svf
python3 svf_player.py ti60f225_security_feature.svf
```

Identifying FTDI URLs

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



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

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

The FTDI URL is in the format:

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

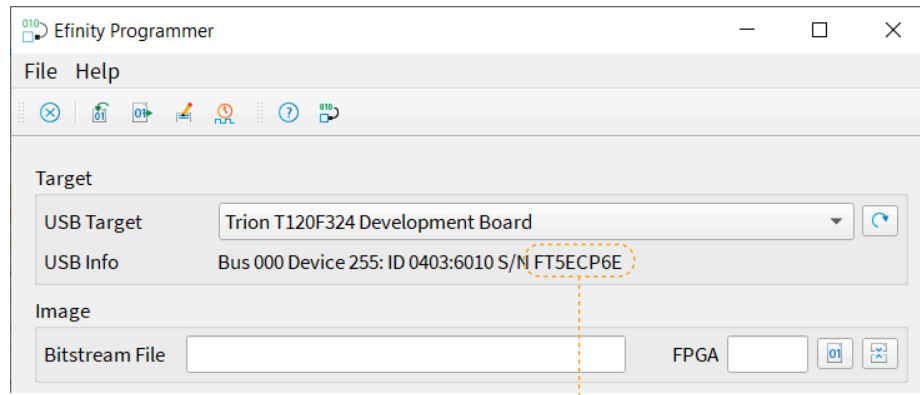
where:

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

<code><product></code>	Board
232h	Trion T8 Development Board
2232h	Trion T20 MIPI Development Board Trion T20 BGA256 Development Board Trion T120 BGA324 Development Board Trion T120 BGA576 Development Board
4232h	Xyloni Development Board
4232h	Titanium Ti60 BGA225 Development Board Titanium Ti375 Development Board Titanium Ti375 Development Board
2232h	Titanium Ti180J484 Development Board
2232h	Topaz Tz170J484 Development Board

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

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



The string after S/N is the FTDI serial number

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

Run the BRAM Initial Content Updater from the Command Line

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

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

Table 17: BRAM Initial Content Updater CLI Options

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

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

Example: Using the BRAM Initial Content Updater

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

Using the Block RAM Resource Estimator

You can access the Efnix Block RAM Resource Estimator from the command line interface by running **efx_map_ramest.exe**.

```
efx_map_ramest[.exe] [--help] --family FAMILY [--device DEVICE] [--be] [--mode MODE]
--size DEPTHxWIDTH [--size2 DEPTHxWIDTH]
```

Table 18: efx_run.py Options

Option (Long)	Option (Short)	Input	Description
--help	N/A	None	Show help.
--family	N/A	Family name	Specifies the family.
--device	N/A	Device name	Specifies the device.
--be	N/A	None	Specifies byte-enable mode. Only for Titanium and Topaz family FPGAs.
--mode	N/A	speed, area, power	Specifies the decomposition mode.
--size	N/A	<depth>x<width>	Memory size in depth by width.
--size2	N/A	<depth>x<width>	Second port memory size in depth by width. For TDP RAM.

The following example estimates the quantity of block RAMs required to implement 512x36 input memory on a Trion FPGA:

Example: Estimating Block RAMs

```
efx_map_ramest --family Trion --size 512x36
```

Manage Bitstream Files at the Command Line

The following topics describe command-line management of Efinity bitstream files.

Generate Bitstream Checksums at the Command Line

You can generate bitstream checksums at the command line using this command:

```
generate_checksum.py [--help] [--mode {sync_plus}] input_file
```

Table 19: export_bitstream.py Positional Arguments

Argument	Input	Description
input_file	Filename	Image file source.

Table 20: export_bitstream.py Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--mode	N/A	sync_plus	Checksum generator mode.

Example: Generating Checksums

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

Convert to Intel Hex Format at the Command Line

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

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

Table 21: export_bitstream.py Positional Arguments

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

Table 22: `export_bitstream.py` Options

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

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

Example: Converting Hex to Bin

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

Combine Bitstreams at the Command Line

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

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

Table 23: `BRAM Initial Content Updater CLI` Options

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

Option (long)	Option (short)	Input	Description
--outfile	-o	Filename	Output bitstream file.

Example: Combining Bitstream Images

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

Encrypt or Sign Bitstreams from the Command Line

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

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

Table 24: AddSecurityTitanium.py Positional Arguments

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

Table 25: AddSecurityTitanium.py Options

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

Option (Long)	Option (Short)	Input	Description
--keypair	-p	Key pair	RSA keypair PEM file (must match that used with GenKeyFileTitanium.py tool).
--passphrase	-x	Pass phrase	Passphrase associated with RSA private key, contained in RSA PEM key pair file. If the private key is passphrase-protected, then this option is required.
--public_key	N/A	Filename	RSA public key PEM file.

Example: Sign and Encrypt a File

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

Using the Efinity Database Export Utility (Beta)

You can use the Efinity Database Export Utility to export Efinity databases and ancillary files.

```
efx_export[.exe] [--help] --prj PRJ <operation> [<operation option>]
```



Important: The Efinity Database Export Utility is in beta in the Efinity software v2025.2. The tool currently only supports exporting input placement files (**.placeloc**).

Table 26: *efx_export.exe* Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--prj	N/A	Filename	Efinity XML project file name.

Table 27: *efx_export.exe* Supported Operations

Option (Long)	Option (Short)	Input	Description
--placeloc	N/A	Filename	Export the current cell placement locations in a format suitable as input for the --loc_assignment option on the efx_pnr tool.

Example: Export Input Placement Files

```
efx_export --prj helloworld.xml --placeloc foobar.placeloc
```

Export JTAG Operations at the Command Line

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

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

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

Table 28: export_bitstream_flashloaderv3.py Positional Arguments

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

Table 29: export_bitstream_flashloaderv3.py Options

Option (Long)	Option (Short)	Input	Description
--help	-h	None	Show help.
--jtag_bridge_file	N/A	File path	The file path of the JTAG-to-SPI Bridge bitstream.
--freq	N/A	Number	JTAG frequency. Default: 6e6
--start_address	N/A	Hex number	Starting flash address for flash read and write operations, excluding 0x.
--target	N/A	lower, upper, both	Target flash. both is used for the x8 mode.
--chip_num	N/A	Number	1-based device position in the JTAG chain. Default: 1
--jcf	N/A	File path	JTAG chain file in XML format. Not needed if there is only one device.
--url	-u	URL	FTDI URL (see Identifying FTDI URLs on page 22).
--verify_method	N/A	none, onchipx1, onchipx2, onchipx4	The method used to verify the downloaded bitstream. Default: onchipx2 (On-chip hash calculation with SPI x2 mode)

Example: Export JTAG Bridge Operations to an SVF File

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



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

Run the export_bitstream_flashloaderv3 Python Script

To create an SVF file:

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



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

To use the created SVF file:

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

Appendix: Additional Flow Options

The following topics overview additional options to be used with the **efx_run.py** script. These options can be found by running **.exe** applications included with the Efinity software with the `--help` option. You should not run the **.exe** applications on their own; their functions are included within the **efx_run.py** script.

Synthesis Options

You can find additional synthesis options for the **efx_run** script with the **efx_map** `--help` option.

```
efx_map[.exe] --help
```



Note: Do not run the **efx_map** application on its own. Use the synthesis options with the **efx_run** script and the `--map_opts` option.

Table 30: Synthesis Options

Option	Input	Description
<code>--help</code>	None	Show help.
<code>--project</code>	Filename	Specifies the project XML file name. If empty, the top-level module name is set as the project name.
<code>--project-xml</code>	Filename	Get the list of source files for synthesis from the project XML file. <code>--v</code> and <code>--f</code> options are ignored.
<code>--work-dir</code>	File path	Specifies the Efinity working directory path. If unspecified, the default work directory is the current directory.
<code>--output-dir</code>	File path	Specifies the Efinity output directory path. If unspecified, the default output directory is the current directory.
<code>--max_threads</code>	Number	Specifies the number of threads to use.
<code>--root</code>	Module name	Specifies the root module.
<code>--arch</code>	VHDL architecture name	Specifies the top-level VHDL architecture.
<code>--family</code>	Family name	Specifies the device family name.
<code>--device</code>	Device name	Specifies the device name.
<code>--v</code>	Filename	HDL source files.
<code>--l</code>	File path	Manage directories where <code>include <directive></code> will search (<code>--incdir</code> option in Verilog-XL, .hex memory initialization file path resolving).
<code>--f</code>	Filename	Verilog-XL option to specify a text file containing source file absolute names.
<code>--settings_file</code>	Filename	Specifies a settings file to be used instead of listing options into the command-line interface.

Option	Input	Description
--write-efx-verilog	Filename	Specifies an output Verilog HDL file for the mapped netlist.
--binary-db	Filename	Specifies a Verific binary database file (.vdb) to dump the mapped netlist.
--top-params	Parameters	List of top module parameters.
--verilog-macros	Verilog HDL macros	List of Verilog HDL macro definitions.
--max_mult	Number	Specifies the maximum number of DSP blocks to be inferred. Default: -1
--max_ram	Number	Specifies the maximum number of block RAMs to be inferred. Default: -1
--infer-clk-enable	1, 2, 3, 4	Infer flip-flock clock-enable signals from control logic. Default: 3
--gated-clk-to-ce	0, 1	Convert gated clock signals into flip-flop clock-enable signals. Default: 1 (enabled)
--infer-sync-set-reset	0, 1	Infer synchronous set/reset signals for flip-flops from control logic. Default: 1 (enabled)
--mode	speed, area, area2	Synthesis optimization mode. Default: speed
--fanout-limit	Number	Set high fanout limit. Default: 0
--seq_opt	0, 1	Run sequential synthesis. Default: 1 (enabled)
--message-verbosity	0, 1	Set message printout verbosity. Default: 0
--retiming	0, 1, 2	Perform register retiming. 0: disabled, 1: enabled (default), 2: advanced mode
--suppress_info_msgs	off, on	Suppress INFO messages from synthesis. Default: off
--suppress_warning_msgs	off, on	Suppress WARNING messages from synthesis. Default: off
--msg_suppression_list	Filename	File to suppress specific INFO/WARNING messages from synthesis by message IDs.

Example: Using map_opts with a Settings File

```
efx_run.py helloworld.xml --flow map --map_opts settings_file settings.txt
```

Place and Route Options

You can find additional place and route options for the **efx_run** script with the **efx_pnr --help** option.

```
efx_pnr[.exe] --help
```



Note: Do not run the **efx_pnr** application on its own. Use the place and route options with the **efx_run** script and the **--pnr_opts** option.

Table 31: Place and Route Primary Options

Option	Input	Description
--help	None	Show basic efx_pnr options.
--help-verbose	None	Show extended efx_pnr options.
--prj	Filename	Efinity project XML file name.
--family	Family name	Specifies the device family.
--device	Device name	Specifies the device name.
--circuit	Circuit name	Specifies the circuit name.
--settings_file	Filename	Specifies a settings file to be used instead of listing options into the command-line interface.
--verbose	None	Generate verbose details message output.
--route	None	Run router.
--place	None	Run placer.
--global_place	None	Run global placement.
--detail_place	None	Run detailed placement.
--pack	None	Run packer.

Table 32: Place and Route General Options

Option	Input	Description
--max_threads	Number	Maximum number of threads allowed.
--work_dir	File path	Specifies the Efinity working directory path.
--output_dir	File path	Specifies the Efinity output directory path.
--sdc_file	Filename	Timing constraints file.
--sync_file	Filename	Interface synchronization file (.csv).
--print_critical_path	Number	Number of critical paths to display.
--operating_conditions	Operating conditions name	Set operating conditions name to be used when performing timing analysis.
--suppress_info_msgs	off, on	Suppress INFO messages from place and route. Default: off
--suppress_warning_msgs	off, on	Suppress WARNING messages from place and route. Default: off

Option	Input	Description
--msg_suppression_list	Filename	File to suppress specific INFO/WARNING messages from place and route by message IDs.

Table 33: Placement Options

Option	Input	Description
--seed	Number	Initial placement seed value.

Table 34: Route Options

Option	Input	Description
--beneficial_skew	on, off	Enable clock-skew optimizations for Titanium family devices.
--max_router_iterations	Number	Maximum allowed router iterations.
--load_route	None	Load existing route file.

Table 35: Packing Options

Option	Input	Description
--physical_packing	None	Perform DSP cell packing based on early placement results. This may result in better quality of results at the cost of increased runtime during the packing stage. This option is in beta in the Efinity software version 2025.2.

Example: Using pnr_opts

```
efx_run.py helloworld.xml --flow pnr --pnr_opts verbose
```

Programming Options

You can find additional programming options for the **efx_run** script with the **efx_pgm --help** option.

```
efx_pgm[.exe] --help
```



Note: Do not run the **efx_pgm** application on its own. Use the place and route options with the **efx_run** script and the **--pgm_opts** option.

Table 36: Programmer Primary Options

Option	Input	Description
--help	None	Show help.
--project-xml	Filename	Specifies the project XML file name. If empty, the top-level module name is set as the project name.
--source	Filename	Input LBF/HEX file.
--periph	Filename	Input periphery configuration file (from periphery tool).

Option	Input	Description
--dest	Filename	Output HEX file.
--query_bitstreams	None	Program exits immediately. Exit code 0 indicates bitstreams enabled for given device, 1 if not.
--query_security_version	None	Program exits immediately. Prints value for security layer device version.

Table 37: Programmer General Options

Option	Input	Description
--family	Family name	Specifies the device family.
--device	Device name	Specifies the device name.
--displayed_device	Device name	Device name written to bitstream header.
--generate_header	on, off	Write a header to the bitstream file.
--timestamp	on, off	Write a timestamp to the bitstream file header.
--width	1, 2, 4, 8, 16, 32	Programming width.
--settings_file	Filename	Specifies a settings file to be used instead of listing options into the command-line interface.
--interface_designer_settings	Filename	Settings file generated from the Interface Designer.
--periph_override	Filename	CSV file to override Interface Designer LPF settings.
--mode	Mode	FPGA configuration mode.

Table 38: HEX Generation Options

Option	Input	Description
--oscillator_clock_divider	DIV8, DIV4, DIV2, DIV1	SPI Active programming clock divider.
--bitstream_compression	on, off	Enable compressed bitstream.
--optimize_bram_data	on, off	Optimize block RAM frames.
--spi_low_power_mode	on, off	Power down flash devices after programming.
--io_weak_pullup	on, off	Enable IO weak pullup during configuration.
--jtag_usercode	Number	JTAG USERCODE
--enable_roms	on, off, smart	Enable initialized memory in user RAMs.
--enable_crc_check	on, off	Enable CRC checking of bitstream during programming.
--enable_pcr_crc_check	on, off	Enable PCR CRC checking of bitstream during programming.
--enable_seu_crc_check	on, off	Enable SEU CRC checking of bitstream during user mode.
--enable_cbssel	on, off	Enable SPI Active image selection using CBSEL[1:0] pins.
--enable_remote_update	on, off	Enable remote update.
--remote_update_retries	Number	Number of retry attempts for remote update.
--flash_addr_2	Number	Flash byte address for multi-image selection #2.

Option	Input	Description
--flash_addr_3	Number	Flash byte address for multi-image selection #3.
--flash_addr_4	Number	Flash byte address for multi-image selection #4.
--enable_daisy_chain	on, off	Enable SPI Active/Passive daisy chain.
--daisy_chain_length	Number	Daisy chain total bit count.
--four_byte_addressing	on, off	Use 4 byte addressing during active mode configuration from SPI flash.
--active_capture_clk_edge	posedge, negedge	SPI Active sampling clock edge.
--enable_external_master_clock	on, off	Select external clock source for SPI Active configuration.
--seu_mode	automatic, manual	Select automatic or manual user mode SEU CRC checking.
--seu_wait_interval	Number	Wait interval between automatic mode SEU CRC checks.
--release_tri_then_reset	on, off	Release IO tri-states before reset upon entering user mode.
--activate_tri_then_reset	on, off	Activate tri-state IO pins before the chip resets during exit from user mode.
--osc_user_enable	on, off	Enable internal oscillator for user mode/core usage.
--osc_user_div_select	DIV8, DIV4, DIV2, DIV1	Select internal oscillator divider for user mode/core usage.

Example: Using Multiple `pgm_opts`

```
efx_run.py helloworld.xml --flow pnr --pgm_opts bitstream_compression=off width=4
```

Where to Learn More

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

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

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

Revision History

Table 39: Revision History

Date	Version	Description
November 2025	1.0	Initial release.