Efinity® Python API

UG-EFN-PYAPI-v6.2
December 2023
www.efinixinc.com

Contents

Introduction

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.[1] 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 target your design to another board, or export the interface to perform analysis. This user guide describes how to use the API and provides a function reference.

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

About the API Functions

The API helps you manipulate interface blocks and their properties, and also lets you perform general actions on the design such as opening or creating designs and saving.

The Efinity® software does not update your interface design in real time when you run API Python scripts or issue commands at the Python console. If you have the Interface Designer open while running Python scripts, it can become out of sync with the files you are manipulating. Therefore, keep the Interface Designer closed while working on the command line and then open it when you are finished to see the result.

Blocks and Their Properties

Each interface block has a unique object ID as well as a name in string format. You define the name when you create the block. For a given block type, the block names must be unique. If you re-use the same name for multiple blocks, the API will fail with an error. So be sure to use unique names. You can reference blocks by name or object ID.

Note: Support for blocks in the **.isf** has rolled out over several Efinity® versions. Refer to **Block Types and Device Settings** on page 40 for details.

Properties and their values are strings. You can retrieve or set a property's value, or you can get data structures of properties and their values. Refer to **Data Types** on page 7 for details.

Note: Some block properties are deprecated and are scheduled for removal in a future version of Efinity software. Efinix recommends that you use the indicated replacement properties and migrate any existing designs that use the deprecated properties.

[1] Source: **What Is Python? Executive Summary**

Interface Scripting File

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

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

Example: Example Interface Scripting File

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

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

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

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

Working with Python

If you are already comfortable with other scripting languages such as Tcl, you only need minimal understanding of Python to use this API. The following sections give an overview of useful Python language characteristics.

Code Blocks

Unlike other languages that use a pair of open and close brackets {}, Python use indentation for code blocks. So formatting your code correctly is important.

This code:

```
# Declare a list of pin names
pin_names = ["resetn", "Oclk"]

# Iterate through all
for pin in pin_names:
    print("Pin name: " + pin)  # Inner code block is indented
```

Prints this result to the Python console:

```
Pin name: resetn
Pin name: Oclk
```

Variables

To create a variable, declare a name and assign a value to it:

```
# Declare a variable
design_name = "pt_demo"
```

Python determines variable type when you run the script. You do not to specify its type when you declare it.

Data Types

The API uses simple variables and sequence-based and map-based data structures.

Simple Variables

Simple variables are strings or numbers like integers and floating-point.

Sequences

Sequence-based structures are a collection of a variable's instances/objects. You access the members of the structure by indexing, and you can iterate (loop through) the structure. Python starts its index at zero. The API mostly uses an array-like type, which is called a `list` in Python.

This code:

```
# Declare a list of pin names
pin_names = ["resetn", "Oclk"]

# Get a single pin
clkpin_name = pin_names[1]
print("Clk pin: " + clkpin_name)

# Iterate through all
for pin in pin_names:
    print("Pin name: " + pin)
```

Prints this result in the Python console:

```
Clk pin: Oclk
Pin name: resetn
Pin name: Oclk
```

Maps

A map is a collection of key-value pairs. The API typically uses the dictionary type, which is called a `dict` in Python.

This code:

```
# Declare a dict of pin name and its information
pin_data = {
    "resetn":"Global control reset pin",
    "Oclk":"Output clock pin name"
}

# Get a single pin info
clk_pin_info = pin_data.get("Oclk", None)
print("Clk pin info: " + clk_pin_info)

# Iterate through all
for name, info in pin_data.items():
    print("Pin name: " + name)
    print("Pin info: " + info)
```

Prints this result to the Python console:

```
Clk pin info: Output clock pin name
Pin name: resetn
Pin info: Global control reset pin
Pin name: Oclk
Pin info: Output clock pin name
```

Displaying a Variable's Value

To output a variable's value, use the `print` statement. This code:

```
# Declare a variable
design_name = "pt_demo"

# Inspecting a single variable
print("Design: " + design_name)

# Another way of inspecting a single variable
print("Design: {}".format(design_name))

# Yet another way of inspecting a single variable
print(f"Design: {design_name}")
```

Prints this output to the Python console:

```
Design: pt_demo
Design: pt_demo
Design: pt_demo
```

If you want to print a map, the built-in Python function, `pprint()`, nicely prints out the content. This code:

```
# Declare a dict of pin name and its information
pin_data = {
    "resetn":"Global control reset pin",
    "Oclk":"Output clock pin name"
}

# Inspecting a dict
import pprint             # This can be done at the top as well
pprint.pprint(pin_data)
```

Prints this result to the Python console:

```
{'Oclk': 'Output clock pin name', 'resetn': 'Global control reset pin'}
```

Functions

You define function using `def <my_function> <parameters>`.

```
# Function declaration
def display_pin_data(pin_data=None):
    for name, info in pin_data.items():
        print("Pin name: " + name)
        print("Pin info: " + info)

# Calling function
pin_data = {
    "resetn":"Global control reset pin",
    "Oclk":"Output clock pin name"
}
display_pin_data(pin_data)
```

To make it easier to use your function, you can define default values for each parameter.

```
# Function declaration
def display_pin_names(pin_names_list=[]):
    if len(pin_names_list) == 0:
        print("Pin names list is empty")
    else:
        for pin in pin_names:
            print("Pin name: " + pin)

# Calling function
pin_names = ["resetn", "Oclk"]
display_pin_names(pin_names)  # All pin names are printed
display_pin_names() # Default parameter is empty list, no pin name printed
```

Handling Errors

If you have errors in your code (which we all know *never* happens), Python typically prints a message. If you enable the verbose option, Python provides more details about the error.

When running your script, you may want to handle errors when they happen. For example, you can print a message and stop the script execution. Refer to **Exceptions** on page 76 for a list of Efinity® API-specific exceptions.

The API typically uses Python exceptions to indicate errors during function execution.

Below are two examples that handle errors.

Example: Error Handling Case 1

```
def display_pin_names_with_exception(pin_names_list=[]):
    if len(pin_names_list) == 0:
        raise Exception("Pin name list is empty")
    else:
        for pin in pin_names:
            print("Pin name: " + pin)

# Calling function
try:
    display_pin_names_with_exception() # Will throw exception since pin name list is empty
except Exception as excp:
    print("Caught an exception: {}".format(excp))
```

Example: Error Handling Case 2

```
# Get access to API exceptions
import api_service.excp.design_excp as DE

# Generate constraints and reports, handles error by catching exception
try:
    design.generate(enable_bitstream=False)
except DE.PTDsgCheckException as excp:
    print("Design check fails : {} Msg={}".format(excp.get_msg_level(), excp.get_msg()))
    sys.exit(1)
except DE.PTDsgGenConstException as excp:
    print("Fail to generate constraint : {} Msg={}".format(excp.get_msg_level(),
 excp.get_msg()))
    sys.exit(1)
except DE.PTDsgGenReportException as excp:
    print("Fail to generate report : {} Msg={}".format(excp.get_msg_level(), excp.get_msg()))
    sys.exit(1)
```

Elements of a Python Script

This topic explains the basic elements you need in your Python script.

The first items to include are `import` commands to import the packages you want to use. In a typical API script, you would import these packages:

- *os*—Operating system package.
- *sys*—System package, for example to access your path.
- *pprint*—To "pretty print" data structures.

Next, set your paths to ensure that Python can find the Interface Designer API files.

```
pt_home = os.environ['EFXPT_HOME']
sys.path.append(pt_home + "/bin")
```

Then, import the specific API that you want to use with the `from <name> import <class>` command. For example, to import the `DesignAPI` class from the `api_service.design` module, use this code:

```
from api_service.design import DesignAPI
```

Note: Refer to API Classes on page 15 for a list of available classes and modules.

You can set the `is_verbose` option to `True` to enable detailed message printing in the API. Of course, you can also write your own messages with print statements in your script.

Finally, you include the API commands to manipulate your interface design. The following code shows a complete script, build_ptdemo.py. This script is provided in the *<Efinity® install path>*/**project/pt_demo/script** directory.

```
# Get access to useful python package
import os
import sys
import pprint

# Tell python where to get Interface Designer's API package
pt_home = os.environ['EFXPT_HOME']
sys.path.append(pt_home + "/bin")

from api_service.design import DesignAPI  # Get access to design database API
from api_service.device import DeviceAPI  # Get access to device database API
import api_service.excp.design_excp as APIExcp  # Get access to API exception

is_verbose = True  # Set to True to see detail messages from API engine
design = DesignAPI(is_verbose)
device = DeviceAPI(is_verbose)

# Create empty design
device_name = "T8F81"  # Matches Device name from Efinity's Project Editor
project_name = "pt_demo"
output_dir = "output"  # New pt_demo periphery design will be generated in this directory
design.create(project_name, device_name, output_dir)

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

# Configure property
design.set_property("Fled", "OUT_REG", "REG")  # Set output to be registered
design.set_property("Fled", "OUT_CLK_PIN", "Fclk")  # Set output clock pin name
design.set_property("Oclk_out", "OUT_CLK_PIN", "Oclk")  # Set output clock pin name

# Pin assignment
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("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("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("resetn", "F1")
design.assign_pkg_pin("Oclk_out", "D6")
design.assign_pkg_pin("pll_clkin", "C3")

# Check design, generate constraints and reports
design.generate(enable_bitstream=False)

# Save the configured periphery design
design.save()
```

How to Run a Script

Before you run Python scripts from the command line, you need to set up your environment. Efinix provides Windows and Linux scripts to make it easy to set up the environment.

In Linux, you can run scripts from any directory after setting up the environment.

In Windows, you set up the environment and then use a helper batch file to run the script from any directory.

Linux

```
> source bin/setup.sh        // set up environment
> python3 my_script.py       // run script
```

Windows

```
> bin\setup.bat                          // set up environment
> efx_run_pt_script.bat my_script.py     // use helper file to run script
```

Issuing Commands in the Python Console

In addition to scripts, you can use the API commands in a Python 3 console. To run in a console, you need to import packages, set up paths, and import the API before using API commands. To open a Python 3 console:

Open Console in Linux

```
> source bin/setup.sh                  // set up environment
> python3                              // open console
```

Open Console in Windows

```
> bin\setup.bat                       // set up environment
> efx_run_pt_script.bat               // use helper file to open console
```

Import Packages, Set up Paths, and Import API

```
> import os                                  // import operating system package
> import sys                                 // import system package
> pt_home = os.environ['EFXPT_HOME']         // set environment
> sys.path.append(pt_home + "/bin")          // set path
> from api_service.design import DesignAPI // import the API
> design = DesignAPI(is_verbose=True)        // turn on verbose messages
```

Use API Commands

```
> design.load("./pt_demo.peri.xml")
> rstn = design.get_gpio("rstn")
```

Getting Help

Each command in the Interface Designer Python API has help as docstrings. You can view help using the command `print(<api>.<function name>.__doc__)`. For example, this command:

```
print(design.get_gpio.__doc__)
```

Shows the help for the `get_gpio()` function:

```
Get GPIO block by its name

:param name: GPIO name
:return: GPIO object id. None if not found
```

Example Scripts

The Efinity® software includes example scripts to help you get starting with writing your own.

Table 1: Example Python Scripts

Location	Script	Description
project/pt_demo/script	**build_ptdemo.py**	Creates an interface for the pt_demo project.
	query_ptdemo.py	Shows how to get information from the existing pt_demo interface design.
	build_ptdemo_error_handling.py	Illustrates how to handle critical errors when building an interface design.
project/ example_scripts/ interface	**gpio_pin_assignment.py**	Shows how to use the API to import pin assignments from a **.csv** file.
	iobank_setting.py	Provides examples on how to set I/O bank voltages and view the settings.
	lvds_builder.py	Illustrates how to create various kinds of LVDS pins (Titanium only).
	mipi_dphy_design.py	Demonstrates how to configure a 4-lane MIPI D-PHY interface for the Ti60 FPGA.
	pll_auto_clock.py	Demonstrates how to use auto clock calculator function.
	pll_core_clock.py	Shows how to configure core clock source.
	pll_dyn_clock.py	Illustrates how to configure dynamic clock source.
	pll_ext_clock_gpio.py	Demonstrates how to configure external clock source.
	pll_ext_clock_lvds.py	Shows how to configure an external clock source using LVDS (Titanium only).
	pll_manual_clock.py	Shows how to configure the PLL clock manually.
project/ example_scripts/ipm	**example_fifo.py**	Demonstrates how to configure and generate a FIFO IP core using the API.

API Classes

The API consists of the following classes and modules.

Table 2: Efinity® Python API Classes and Modules

Class	Module	Use for	Description
APIExcep	api_service.excp.design_excp	Interface Designer	Contains exceptions for the DesignAPI functions
APIVersion	api_service.api_info	Interface Designer	Contains functions to get information about the API and it's versioning.
DesignAPI	api_service.design	Interface Designer	Includes the majority of API functions relating to the Interface Designer.
DeviceAPI	api_service.device	Interface Designer	Contains functions relating to the interface resources.
IPMDesignAPI	ipm_api_service.design	IP Manager	Includes the functions to create, configure, and generate IP cores.
ProjectXML	ipm_api_service.projectxml	IP Manager	Has functions for adding IP to your Efinity® project.

API Functions by Category

Design Management Functions

These functions control the overall interface design.

create() import_design() save() clean_output()
load() export_design() save_as()

Create Block Functions

You use these functions to create blocks. To simplify the process, there are custom functions for clocks, controls, inputs, outputs, etc.

Inputs, Outputs, and Bidirectional

create_input_gpio() create_output_gpio() create_inout_gpio()
create_open_drain_output_gpio() create_unused_gpio()

Clocks and Controls

create_input_clock_gpio() create_clockout_gpio() create_global_control_gpio()

VREF Pin

create_vref_gpio()

PLLs

create_pll_input_clock_gpio() create_pll_ext_fb_gpio()

MIPI

create_mipi_input_clock_gpio()

General Block

create_block()

Delete Block Functions

You use these functions to delete blocks.

delete_block() delete_gpio() clear_design()

Get Block Functions

You can get the name or object ID of a GPIO, bus, or other block. These functions return a string, a list, or an object ID.

By Name	As List	By Object ID
get_all_block_name()	get_all_gpio()	get_block()
get_all_gpio_name()	get_block_type()	get_bus()
	get_bus_gpio()	get_gpio()

Block Property Functions

These functions retrieve properties for a block.

get_property()	get_all_property()	set_property()

Bus Functions

The API includes two bus-specific functions, and other functions also support buses. For example, to create an input bus, output bus, or inout bus, you simply specify the MSB and LSB in addition to the name. The API also supports the same bus properties as the GUI, refer to GPIO Property Reference on page 41 for details.

get_bus()	get_bus_gpio()	export_design()
create_input_gpio()	create_output_gpio()	create_inout_gpio()
delete_gpio()	get_all_gpio()	get_all_gpio_name()

Resource Functions

These functions help you manage the FPGA resources.

assign_resource()	get_block_resource()	get_block_resource_name()
get_resource()	get_gpio_resource()	get_gpio_resource_name()
is_resource_used()		

Package Pin Functions

You use these functions to manage package pin assignments.

assign_pkg_pin()	get_pkg_pin()	is_pkg_pin_used()

Clock Functions

These functions help you configure and build your PLL.

auto_calc_pll_clock()
calc_pll_clock()
gen_pll_ref_clock()
trace_ref_clock()

Device Settings

These functions are for device settings such as I/O bank information and the FPGA properties.

get_all_iobank_name()
get_device_property()
get_iobank_voltage()
set_device_property()
set_iobank_voltage()
get_all_global_mux_name()
get_regional_buffer_info()
get_global_dynamic_mux_input_info()
reset_device_settings()

Preset Settings

You use this these functions to get or set the DDR presets. DDR presets are supported in the Trion family only.

get_all_preset_info()
get_preset()
set_preset()

Constraint and Report Functions

You use this function to generate a constraint file and reports.

generate()

Design Check Functions

You use these functions to check the design or get a list of errors.

check_design()
get_design_check_issue()

API Information Functions

The API information functions retrieve versioning details.

- The major number indicates significant changes and new features. The scripts written for a previous version are not compatible with the new API.
- The minor number indicates significant new features. Scripts written for a previous version are compatible with the new API.
- The revision number indicates bug fixes and minor enhancements.

get_current_version() get_current_version_info() get_version_info()

get_all_version() get_supported_block()

IP Manager Functions

You can create IP core instances with the API. For an example script, refer to *<Efinity path>*/project/example_scripts/ipm/example_fifo.py.

add_ip() create_ip() config_ip() get_ip_list()

generate_ip() is_ip_exists_ip() save() validate_ip()

API Functions: Interface Designer

This section provides an alphabetical list of API functions used for the Interface Designer.

assign_pkg_pin()

Usage	`assign_pkg_pin(inst, pin_name)`
Parameters	`inst`: The block's object ID or instance name. `pin_name`: GPIO pin name.
Import	DesignAPI
Description	Assign the specified GPIO to the resource associated with the specified package pin.

assign_resource()

Usage	`assign_resource(inst, res_name, block_type)`
Parameters	`inst`: The block's object ID or instance name. `res_name`: Resource name, enclose in double quotes. `block_type`: Type of block. If `inst` is an instance name, indicate the block type. The default is `GPIO`.
Import	DesignAPI
Description	Assign a resource to a GPIO or other block.

auto_calc_pll_clock()

Usage	`auto_calc_pll_clock(inst, target_freq=None, apply_optimal=True, result_len)`
Parameters	`inst`: A PLL instance block object id or an instance name. `target_freq`: A map of output frequency parameters in MHz. The parameters depend on the PLL you are using. If you do not include a parameter, the software uses the default setting.

PLL V3	PLL V2	PLL V1
CLKOUTn_FREQ CLKOUTn_PHASE CLKOUTn_DYNPHASE_EN where n is 0, 1, 2, 3, or 4	CLKOUTn_FREQ CLKOUTn_PHASE where n is 0, 1, or 2	CLKOUTn_FREQ where n is 0, 1, or 2

`result_len`: Number of results. `0` returns all results. The default is `None` which returns 5 results.

`apply_optimal`: When True, apply optimal result counters to PLL, when False, does not.

Returns	A list of PLL setting options, a list of property maps.
Import	DesignAPI
Description	Given desired output clock frequency in MHz (and/or phase in degree), this function sets the required counters and output divider. See PLL Property Reference on page 73.

calc_pll_clock()

Usage	`calc_pll_clock(inst)`
Parameters	`inst`: A PLL instance block object id or an instance name.
Returns	Calculated frequencies property map.
Import	DesignAPI
Description	Calculates the VCO frequency, PLL frequency, and all enabled output clock frequencies. You should configure the M, N, and O counters and output dividers before using this function. See **PLL Property Reference** on page 73.

check_design()

Usage	`check_design()`
Parameters	None.
Returns	Boolean `True` (if passes) or `False` (if fails).
Import	DesignAPI
Description	Performs a design rule check. For Python to print errors, set `is_verbose()` to `True`. See **Elements of a Python Script** on page 11.

clear_design()

Usage	`clear_design(block_types)`
Parameters	`block_type`: The block type strings, enclose in double quotes. Refer to **Block Types and Device Settings** on page 40 for the supported blocks. Default is `None` which removes all the instances in the design.
Import	DesignAPI
Description	By default, all the instances of specific block_types will be removed; otherwise, all the instances in the design will be removed.

clean_output()

Usage	`clean_output()`
Parameters	None.
Import	DesignAPI
Description	Cleans the generated output files in the project's **outflow** directory.

create()

Usage	`create(design_name, device_name, path, auto_save, overwrite)`
Parameters	`design_name`: Efinity project name, enclose in double quotes. `device_name`: Name and package for the target FPGA, enclose in double quotes. For example, T20F256. This must be the same name you specified for the project in the Efinity Project Editor. `path`: Location to store the design file, enclose in double quotes. `auto_save`: Boolean `True` or `False` (default). If set to `True`, the design writes the file at the specified location. `overwrite`: Boolean `True` or `False` (default). If `auto_save` is `True`, overwrites the file if it exists at the specified location.
Import	DesignAPI
Description	Create a new interface design. After you create the design, call the `save()` function to save the file to disk.

create_block()

Usage	`create_block(name, block_type, **kwargs`[2]`)`
Parameters	`name`: Name of the block, enclose in double quotes. `block_type`: The block type, enclose in double quotes. Refer to **Block Types and Device Settings** on page 40 for the supported blocks. *MIPI Lane TX and RX:* `mode`: Defines a clock lane (`CLOCK_LANE`) or data lane (`DATA_LANE`). If you do not specify the parameter, the software defaults to `DATA_LANE`. `conn_type`: For the MIPI Lane RX, indicate whether you want to connect to the global network (GCLK) or regional network (RCLK). If you do not specify the parameter, the software defaults to `GCLK`. *LVDS:* `tx_mode`: For LVDS_TX blocks and LVDS_BIDIR blocks, specify whether the block is data (DATA) or clock out (CLKOUT). If you do not specify the parameter, the software defaults to `DATA`. `rx_conn_type`: For LVDS_RX blocks and LVDS_BIDIR blocks, specify the connection type for the LVDS RX. `NORMAL` (default)), `GCLK`, `RCLK`, `PLL_CLKIN`, `PLL_EXTFB`.
Returns	The object ID of the new block.
Import	DesignAPI
Description	Create a new block instance of the specified type. Do not use `create_block()` for GPIO, instead use one of the `create_<type>_gpio()` blocks.

create_clockout_gpio()

Usage	`create_clockout_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create an output clock GPIO (`clkout`).

[2] **kwargs are parameters that are used for specific block types.

create_global_control_gpio()

Usage	`create_global_control_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create a global control GPIO with connection type `GCTRL`.

create_inout_gpio()

Usage	`create_inout_gpio(name, msb, lsb)`
Parameters	`name`: GPIO name, enclose in double quotes. `msb`: Integer. If creating a bus, specify the MSB. The default is `None`. `lsb`: Integer. If creating a bus, specify the LSB. The default is `None`.
Returns	The object ID of the GPIO or bus.
Import	DesignAPI
Description	Create an inout GPIO. To create a bus, specify values for the MSB and LSB bits.

create_input_clock_gpio()

Usage	`create_input_clock_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create an input clock GPIO with connection type `GCLK`.

create_input_gpio()

Usage	`create_input_gpio(name, msb, lsb)`
Parameters	`name`: GPIO name, enclose in double quotes. `msb`: Integer. If creating a bus, specify the MSB. The default is `None`. `lsb`: Integer. If creating a bus, specify the LSB. The default is `None`.
Returns	The object ID of the GPIO or bus.
Import	DesignAPI
Description	Create an input GPIO. To create a bus, specify values for the MSB and LSB bits.

create_mipi_input_clock_gpio()

Usage	`create_mipi_input_clock_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create MIPI input clock GPIO (`mipi_clkin`) with connection type `MIPI_CLKIN`. This input clock is used with the Trion hard MIPI blocks.

create_open_drain_output_gpio()

Usage	`create_open_drain_output_gpio(name)`
Parameters	`name`: GPIO name, enclose in double quotes.
Returns	The object ID of the GPIO.
Import	DesignAPI
Description	Create an inout GPIO in open drain output configuration; the output is grounded. Constant output is set to 0 (GND).

create_output_gpio()

Usage	`create_output_gpio(name, msb, lsb)`
Parameters	`name`: GPIO name, enclose in double quotes. `msb`: Integer. If creating a bus, specify the MSB. The default is `None`. `lsb`: Integer. If creating a bus, specify the LSB. The default is `None`.
Returns	The object ID of the GPIO or bus.
Import	DesignAPI
Description	Create an output GPIO. To create a bus, specify values for the MSB and LSB bits.

create_pll_ext_fb_gpio()

Usage	`create_pll_ext_fb_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create a PLL external feedback input GPIO (`pll_extfb_conn`) with connection type `PLL_EXTFB`.

create_pll_input_clock_gpio()

Usage	`create_pll_input_clock_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create an input clock for the PLL (`pll_clkin`) with connection type `PLL_CLKIN`.

create_regional_input_clock_gpio()

Usage	`create_regional_input_clock_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create a regional input clock GPIO with connection type `RCLK`.

create_unused_gpio()

Usage	`create_unused_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	The GPIO object ID.
Import	DesignAPI
Description	Create an unused GPIO.

create_vref_gpio()

Usage	`create_vref_gpio(name)`
Parameters	`name`: The GPIO name, enclose in double quotes.
Returns	A GPIO object id.
Import	DesignAPI
Description	Create a VREF GPIO.

delete_block()

Usage	`delete_block(inst, block_type)`
Parameters	`inst`: Block object ID or name. If you are deleting the block by name, enclose the name in double quotes. `block_type`: The block type; enclose in double quotes. the default is `JTAG`.
Import	DesignAPI
Description	Delete the specified block instance. if you use a name for `inst`, also specify the block type. For GPIO, use `delete_gpio()`.

delete_gpio()

Usage	`delete_gpio(inst)`
Parameters	`inst`: GPIO or bus object ID or name. If you are deleting the GPIO by name, enclose the name in double quotes. The API searches for GPIO first.
Import	DesignAPI
Description	Delete a GPIO block or GPIO bus block.

export_design()

Usage	`export_design(block_type, block_inst, isf_file, export_all_pin)`
Parameters	`block_type`: Optional. A comma-separated list of block types, enclose names in double quotes. The default is `None`. `block_inst`: Optional. A comma-separated list of instance names, enclose names in double quotes. For GPIO, you can specify a list of bus instances. The default is `None`. `isf_file`: Optional. User-specified path and filename. If you do not specify a name, the script creates the file <*design*>**.isf** in the project directory. `export_all_pin`: Optional. Export all pin names, including default auto-generated names. The default is `False`.
Import	DesignAPI
Description	Exports the interface design settings for the specified block type(s) or block instance(s) to an **.isf** file in the project directory. If you specify both `block_type` and `block_inst`, the block type takes precedence.

generate()

Usage	`generate(enable_bitstream, outdir)`
Parameters	`enable_bitsteam`: Boolean `True` or `False`. If `True`, generate the periphery bitstream file. `outdir`: Specify an optional output directory with path. The default, `None`, saves files in the project's **outflow** directory. Enclose in double quotes.
Import	DesignAPI
Description	Performs design checks and generates constraint and report files. Optionally generate a periphery bitstream file.

gen_pll_ref_clock()

Usage	`gen_pll_ref_clock(inst, **kwargs`[3]`)`
Parameters	`inst`: PLL block `pll_res`: PLL resource name, will overwrite existing resource if it exists. `refclk_name`: GPIO instance name for reference clock.

Parameter	PLL	Description
`refclk_res`	V1	GPIO resource name for reference clock.
`refclk_name`	V1	GPIO instance name for reference clock.
`refclk_src`	V2, V3	Reference clock source type, CORE, EXTERNAL or DYNAMIC.
`ext_refclk_no`	V2, V3	External clock source number (V2: 0 or 1) (V3: 0, 1, 2[4]) for external clocks only.
`ext_refclk_type`	V3	Block name of the clock source. Options are GPIO (default), LVDS_RX, and LVDS_BIDIR.

Import	DesignAPI
Description	Generate a PLL reference clock. Note: the software automatically detects the GPIO resource. Refer to **Example Scripts** on page 14 for examples. See **PLL Property Reference** on page 73 for PLL property list..

get_all_block_name()

Usage	`get_all_block_name(block_type)`
Parameters	`block_type`: The block type; enclose in double quotes. the default is `GPIO`.
Returns	A list of the block names.
Import	DesignAPI
Description	Gets the names of all block instances for the specified block type.

(3) **kwargs are parameters that are used for specific block types.
(4) Some PLLs have 2 external clock sources and some have 3. Refer to the PLL chapter in the Titanium Interfaces User Guide for a listing of PLL resources by FPGA and package.

get_all_global_mux_name()

Usage	`get_all_global_mux_name()`
Parameters	None.
Returns	A list of the clock multiplexer names.
Import	DesignAPI
Description	Gets the names of all clock multiplexer, Bottom, Left, Top, and Right.

get_all_gpio()

Usage	`get_all_gpio()`
Parameters	None.
Returns	A list of GPIO or an empty list.
Import	DesignAPI
Description	Gets a list of all GPIO and buses used.

get_all_gpio_name()

Usage	`get_all_gpio_name()`
Parameters	None.
Returns	A list of GPIO and bus names.
Import	DesignAPI
Description	Gets the names of all GPIO and buses used.

get_all_iobank_name()

Usage	`get_all_iobank_name()`
Parameters	None.
Returns	A list of I/O banks.
Import	DesignAPI
Description	Get the names of all the I/O banks.

get_all_preset_info()

Usage	`get_all_preset_info(name)`
Parameters	`name`: The DDR instance name. Enclose instance names in double quotes.
Returns	Returns all available DDR preset of the provided DDR instance name.
Import	DesignAPI
Description	Get all preset options according to the provided DDR instance name.

get_all_property()

Usage	`get_all_property(block_type, object_id)`
Parameters	`block_type`: Type of block; enclose in double quotes. The default is `None`. `object_id`: Block object ID. The default is `None`.
Returns	A dict of property names to values.
Import	DesignAPI
Description	Displays all of the properties and values for the specified block type. If you also specify an `object_id`, this command only shows the properties for the block type and object. For example: `block_type="JTAG"`, displays all possible properties. `block_type="GPIO"`, displays all possible properties for all modes. `block_type="GPIO", obj_id=input_gpio`, displays all properties for input GPIO.

get_all_version()

Usage	`get_all_version()`
Parameters	None.
Returns	A list of version information objects.
Import	APIVersion
Description	Get detailed release information for all existing API versions.

get_block_type()

Usage	`get_block_type()`
Parameters	None.
Returns	A list of block type names.
Import	DesignAPI
Description	Gets a list of the available block types. See **Block Types and Device Settings** on page 40.

get_block()

Usage	`get_block(name, block_type)`
Parameters	`name`: Name of the block, enclose in double quotes. `block_type`: The block type, enclose in double quotes.
Returns	Block object ID.
Import	DesignAPI
Description	Get the object ID of the named block. For GPIO, use `get_gpio()`.

get_block_resource()

Usage	`get_block_resource(res_name, block_type)`
Parameters	`res_name`: Resource name; enclose in double quotes. `block_type`: Type of block, enclose in double quotes.
Returns	A dict of property name, value pairs; otherwise returns an empty dict.
Import	DeviceAPI
Description	Get the properties for the specified block resource. If you do not specify a resource name, this command gets information for all resources.

get_block_resource_features()

Usage	`get_block_resource_features(res_name, block_type)`
Parameters	`res_name`: Resource name; enclose in double quotes. `block_type`: MIPI_RX_LANE or MIPI_TX_LANE, enclose in double quotes.
Returns	A dict of property name, value pairs; otherwise returns an empty dict.
Import	DeviceAPI
Description	Get the features (MIPI group, mode, and function) for the specified block resource. If you do not specify a resource name, this command gets information for all resources.

get_block_resource_name()

Usage	`get_block_resource_name(block_type)`
Parameters	`block_type`: Type of block, enclose in double quotes.
Returns	A list of resource names.
Import	DeviceAPI
Description	Get a list of all available resources for the specified block type.

get_bus()

Usage	`get_bus(name)`
Parameters	`name`: GPIO bus name, enclose in double quotes.
Returns	If found, returns the GPIO bus object ID.
Import	DesignAPI
Description	Get the object ID of a GPIO bus block by name. To get a single GPIO block, use `get_gpio()`.

get_bus_gpio()

Usage	`get_bus_gpio(bus_object_id)`
Parameters	`bus_object_id`: Object ID of the GPIO bus.
Returns	A list of GPIO or an empty list.
Import	DesignAPI
Description	Get a list of the GPIO blocks in the specified bus.

get_current_version()

Usage	`get_current_version()`
Parameters	None.
Returns	Text string.
Import	APIVersion
Description	Get the release's full version number.

get_current_version_info()

Usage	`get_current_version_info()`
Parameters	None.
Returns	A version object.
Import	APIVersion
Description	Get detailed version information for the release.

get_design_check_issue()

Usage	`get_design_check_issue()`
Parameters	None.
Returns	A list of messages.
Import	DesignAPI
Description	Get a list of design check issues.

get_device_property()

Usage	`get_device_property(inst, prop_name, devset_type)`
Parameters	`inst`: The value; enclose in double quotes. `prop_name`: Name of the property, enclose in double quotes. `devset_type`: Specify the device setting type. For the Efinity software v2022.2, the "IOBANK", "SEU", "RU", "EXT_FLASH", and "CLKMUX" types are supported.
Returns	A value if assigned, otherwise, an empty string.
Import	DesignAPI
Description	Get the value of the property for the specified device setting block.

get_device_api()

Usage	`get_device_api()`
Parameters	None.
Returns	Returns DeviceAPI for current design.
Import	DeviceAPI
Description	Gets DeviceAPI for current design.

get_global_dynamic_mux_input_info()

Usage	`get_global_dynamic_mux_input_info(clkmux_name, dynamic_mux, clock_input, index=None)`
Parameters	`clkmux_name`: Clock multiplexer name, enclose in double quotes. `dynamic_mux`: Dynamic multiplexer number (0 or 7), enclose in double quotes. `clock_input`: Dynamic multiplexer input index (0, 1, 2, or 3), enclose in double quotes. `index`: The index associated with the options available at the selected input enclose in double quotes or None. • Ti35 and Ti60: 0, 1, 2, 3, 4 • Ti90, Ti120, and Ti180: 0, 1, 2, 3, 4, 5, 6
Returns	Returns a list of input resources.
Import	DesignAPI
Description	Get the list of input resources associated to a specific dynamic multiplexer input. Each dynamic multiplexer has 4 inputs with each input having multiple possible resource options to be selected. If `index` is empty, then all available resources associated to the clock_input is listed.

get_gpio()

Usage	`get_gpio(name)`
Parameters	`name`: GPIO name, enclose in double quotes.
Returns	If found, returns the GPIO object ID.
Import	DesignAPI
Description	Get the object ID of a GPIO block by name. To get a bus, use `get_bus()`.

get_gpio_resource()

Usage	`get_gpio_resource(resource_name)`
Parameters	`resource_name`: The GPIO resource name, enclose in double quotes.
Returns	A dict of property name, value pairs; otherwise returns an empty dict.
Import	DeviceAPI
Description	Get the properties for the specified GPIO resource. If you do not specify a resource name, this command returns information for all resources. The properties are: `NAME`: Resource name `INSTANCE`: Instance name, if any `ALT_CONN`: Alternate connection name, if any `CLK_REGION`: Clock region name `IO_BANK`: I/O bank name `PACKAGE_PIN`: Package pin name `PAD`: Pad name `FEATURES`: Special features of the resource, such as DDIO

get_gpio_resource_name()

Usage	`get_gpio_resource_name()`
Parameters	None.
Returns	A list of resource names.
Import	DeviceAPI
Description	Gets a list of all available GPIO resources, including LVDS pins used as GPIO.

get_iobank_voltage()

Usage	`get_iobank_voltage(bank_name)`
Parameters	`bank_name`: The I/O bank; enclose in double quotes.
Returns	The voltage if assigned.
Import	DesignAPI
Description	Get the voltage set for the specified I/O bank.

get_object_block_type()

Usage	`get_object_block_type(inst)`
Parameters	`inst`: Instance name or object ID; enclose in double quotes.
Returns	Returns the block type; otherwise returns an empty string.
Import	DesignAPI
Description	Get the block type for the specified block resource.

get_pkg_pin()

Usage	`get_pkg_pin(inst)`
Parameters	`inst`: GPIO object ID or instance name. Enclose instance names in double quotes.
Returns	If assigned, returns the pin name; otherwise an empty string.
Import	DesignAPI
Description	Get the package pin assigned to the GPIO instance.

get_preset()

Usage	`get_preset(name)`
Parameters	`name`: The DDR instance name. Enclose instance names in double quotes.
Returns	Returns current DDR preset ID and description.
Import	DesignAPI
Description	Get preset ID and preset description of current DDR instance.

get_property()

Usage	`get_property(inst, prop_name, block_type)`
Parameters	`inst`: The block's object ID or instance name. Enclose instance names in double quotes. `prop_name`: Name of the property or a comma-separated list of properties. Enclose names in double quotes. `block_type`: Type of block, enclose in double quotes. If `inst` is an instance name, indicate the block type. The default is `GPIO`.
Returns	A dict of property name to value.
Import	DesignAPI
Description	Display the value of the specified property for the instance and block type. You can specify one or more property names. See **GPIO Property Reference** on page 41

get_regional_buffer_info()

Usage	`get_regional_buffer_info(clkmux_name, regional_buf=None)`
Parameters	`clkmux_name`: Left, Right, Bottom, or Top; enclose in double quotes. `regional_buf`: Regional buffer number; enclose in double quotes or None.
Returns	Returns a list of regional buffer resources associated to the specified clock multiplexer; otherwise an empty string.
Import	DesignAPI
Description	Get the information associated with the regional buffer. If `regional_buf` is none or not specified, then it returns info for all regional buffers associated to the clock multiplexer.

get_resource()

Usage	`get_resource(inst, block_type)`
Parameters	`inst`: The block's object ID or instance name. Enclose instance names in double quotes. `block_type`: Indicates the block type, enclose in double quotes. The default is `GPIO`.
Returns	If assigned, returns a string with the name; otherwise an empty string.
Import	DesignAPI
Description	Get the name of the resource assigned to the instance of the specified block type.

get_supported_block()

Usage	`get_supported_block(ver_no=None)`
Parameters	`ver_no=None`: Version number or None; enclose in double quotes.
Returns	A list of block type names.
Import	APIVersion
Description	Get a list of all the supported block types for the specified version number. If `ver_no=None`, get a list of all the supported block types for the current version.

get_version_info()

Usage	`get_version_info(ver_no)`
Parameters	`ver_no`: Version number; enclose in double quotes.
Import	APIVersion
Description	Provide information for the specified version number.

import_design()

Usage	`import_design(design_isf_file, gen_issue_csv, path)`
Parameters	`design_isf_file`: Design filename with **.isf** extension, enclose in double quotes. `gen_issue_csv`: Boolean `True` or `False` (default). When `True`, the command exports any design issues to a **.csv** file. `path`: Specify an optional path to save the issues **.csv** file. If you do not specify a path, it is saved into the same location as the **.isf**.
Returns	Boolean `True` (if successful) or `False` (if errors).
Import	DesignAPI
Description	Imports the Interface Design from the specified **.isf**.

is_pkg_pin_used()

Usage	`is_pkg_pin_used(pin_name)`
Parameters	`pin_name`: Package pin name, enclose in double quotes.
Returns	Boolean `True` if used or `False` if not.
Import	DesignAPI
Description	Check if a package pin already has an assignment.

is_resource_used()

Usage	`is_resource_used(res_name, block_type)`
Parameters	`res_name`: Resource name, enclose in double quotes. `block_type`: Indicates the block type, enclose in double quotes. The default is `GPIO`.
Returns	Boolean `True` if used or `False` if not.
Import	DesignAPI
Description	Check if a resource already has an assignment.

load()

Usage	`load(design_file)`
Parameters	`design_file`: The design filename including the path, enclose in double quotes.
Import	DesignAPI
Description	Load the interface design from the specified file.

reset_device_settings()

Usage	`reset_device_settings(devset_types=None)`
Parameters	`devset_type`: Specify the device setting type string. For the Efinity software v2023.2, "IOBANK", "SEU", "RU", "EXT_FLASH", and "CLKMUX" types are supported. Default is "NONE" which resets all device settings to default.
Import	DesignAPI
Description	Reset `devset_type` device settings to the default. If `devset_type` is not set, all device settings are reset to default.

save()

Usage	`save()`
Parameters	None.
Import	DesignAPI
Description	Save the interface to the file specified with the `create()` command.

save_as()

Usage	`save_as(design_file, overwrite)`
Parameters	`design_file`: The design filename including the path, enclose in double quotes. `overwrite`: Boolean `True` or `False` (default). If `auto_save` is `True`, overwrites the file if it exists at the specified location.
Import	DesignAPI
Description	Save the interface design to a new file.

set_device_property()

Usage	`set_device_property(inst, prop_name, prop_value, devset_type)`
Parameters	`inst`: The device setting instance to set; enclose in double quotes. `prop_name`: Name of the property, enclose in double quotes. `prop_value`: The property value. enclose in double quotes. `devset_type`: Specify the device setting type. For the Efinity software v2022.2, "IOBANK", "SEU", "RU", "EXT_FLASH", and "CLKMUX" types are supported.
Import	DesignAPI
Description	Set the property of the specified device setting.

set_iobank_voltage()

Usage	`set_iobank_voltage(bank_name, voltage)`
Parameters	`bank_name`: The I/O bank; enclose in double quotes. `voltage`: The voltage to set; enclose in double quotes.
Import	DesignAPI
Description	Set the voltage of the specified I/O bank. You can set values for multiple banks by defining a map. Refer to the **iobank_setting.py** example script.

set_preset()

Usage	`set_preset(name, preset_id)`
Parameters	`name`: The DDR instance name. Enclose instance names in double quotes. `preset_id`: The DDR preset ID.
Import	DesignAPI
Description	Set preset ID and define which preset is used.

set_property()

Usage	`set_property(inst, prop_name, prop_value, block_type)`
Parameters	`inst`: The block's object ID or instance name, enclose instance names in double quotes. `prop_name`: Name of the property, enclose in double quotes. `prop_value`: The property value, enclose in double quotes. `block_type`: Type of block, enclose in double quotes. If `inst` is an instance name, indicate the block type. The default is `GPIO`.
Import	DesignAPI
Description	Set the value of the specified property for the specified block. If `inst` is an instance name, indicate the block type. See **GPIO Property Reference** on page 41 Do not use `set_property()` for resources or pins, instead use `assign_resource()` and `assign_pkg_pin()`, respectively.

trace_ref_clock()

Usage	`trace_ref_clock(inst, block_type="PLL")`
Parameters	`inst`: A PLL instance block object id or an instance name. `block_type`: Block type of the instance if it is specified by name.
Returns	A list of one or more clock source property maps.
Import	DesignAPI
Description	Trace the clock source for the target instance. If the instance is a PLL, the clock source is a GPIO instance (PLL V1 and V2) or a GPIO or LVDS instance (PLL V3). In v2021.2, only the PLL block is supported.

API Functions: IP Manager

This section provides an alphabetical list of API functions used for the IP Manager.

add_ip()

Usage	`add_ip(module_name)`
Parameters	`module_name`: The name of the IP module.
Import	ProjectXML
Description	Add the IP module to the project.

config_ip()

Usage	`config_ip(module_name, configs)`
Parameters	`module_name`: Module or instance name for the IP. Enclose in double quotes. `configs`: The IP parameters and values in dictionary format.
Import	IPMDesignAPI
Description	Configure the IP instance or module with the specified parameters.

create_ip()

Usage	`create_ip(module_name, vendor, library, name)`
Parameters	`module_name`: Module or instance name for the IP. Enclose in double quotes. `vendor`: IP vendor name (string). Enclose in double quotes. `library`: IP library name (string). Enclose in double quotes. `name`: IP name (string). Enclose in double quotes.
Returns	The vendor, library, name, and version in Vlnv format.
Import	IPMDesignAPI
Description	Create an instance of the IP core with default parameters. To parameterize the IP, use the `config_ip` command.

generate_ip()

Usage	`generate_ip(module_name)`
Parameters	`module_name`: Module or instance name for the IP. Enclose in double quotes.
Returns	Generation result message (success or error).
Import	IPMDesignAPI
Description	Generates the specified IP module or instance. The generated files are saved in the **ip** folder in the project directory.

get_ip_list()

Usage	`get_ip_list([vendor,] [library,] [name,] [device_name,] [family_name])`
Parameters	`vendor`: IP vendor name (string). Enclose in double quotes. `library`: IP library name (string). Enclose in double quotes. `name`: IP name (string). Enclose in double quotes. `device_name`: Target FPGA (string). Enclose in double quotes. `family_name`: Target FPGA family (Trion or Titanium). Enclose in double quotes.
Returns	A list of IP cores, optionally filtered, in VInv format.
Import	IPMDesignAPI
Description	Get list of installed IP cores available in the Efinity® software. You can filter the list using the parameters.

is_ip_exists()

Usage	`is_ip_exists(module_name)`
Parameters	`module_name`: IP module name, enclose in double quotes.
Returns	Boolean `True` if used or `False` if not.
Import	ProjectXML
Description	Check whether an IP module is in the project.

save()

Usage	`save()`
Parameters	None.
Import	ProjectXML
Description	Save the project XML file.

validate_ip()

Usage	`validate_ip(module_name)`
Parameters	`module_name`: Module or instance name for the IP. Enclose in double quotes.
Returns	True if the parameters are correctly configured. Error messages, if any, generated during validation.
Import	IPMDesignAPI
Description	Validates the configured parameters for the specified IP module or instance.

Block Types and Device Settings

The following tables list the block types and device setings the Interface Designer Python API supports, organized by Efinity® release.

Table 3: Block Types

Block Description	Block Name	Trion		Titanium	
		2023.1	2023.2	2023.1	2023.2
Single GPIO.	GPIO	✓	✓	✓	✓
GPIO bus.	GPIO_BUS	✓	✓	✓	✓
JTAG User Tap block.	JTAG	✓	✓	✓	✓
LVDS block.	LVDS_RX LVDS_TX	✓	✓	✓	✓
	LVDS_BIDIR	-	-	✓	✓
Oscillator block.	OSC	✓	✓	✓	✓
Simple PLL block (PLL V1). Advanced PLL block (PLL V2).	PLL	✓	✓	-	-
PLL block (PLL V3).	PLL	-	-	✓	✓
MIPI RX or TX block.	MIPI	✓	✓	-	-
MIPI TX or RX Lane block.	MIPI_RX_LANE MIPI_TX_LANE	-	-	✓	✓
MIPI D-PHY TX or RX block.	MIPI_DPHY_TX MIPI_DPHY_RX	-	-	✓	✓
DDR DRAM block.	DDR	✓	✓	✓	✓
SPI Flash block.	SPI_FLASH	✓	✓	✓	✓
HyperRAM block.	HYPERRAM	-	-	✓	✓
PLL SSC block.	PLL_SSC	-	-	✓	✓

Table 4: Device Settings

Setting	Trion		Titanium	
	2023.1	2023.2	2023.1	2023.2
I/O Banks	✓	✓	✓	✓
Remote Update	✓	✓	✓	✓
SEU	-	-	✓	✓
Clock Mux	-	-	✓	✓
External Flash Controller	-	-	✓	✓

GPIO Property Reference

GPIO blocks, buses, and bus members have a variety of properties. Some properties can be changed, while for others, you need to delete the block and create a new one with the property you want.

GPIO Input Properties

You can change these properties for GPIO and buses.

You cannot change these properties for bus members.

API Name	GUI Name	Trion	Titanium	Values
DLY_ENA_PIN	Enable Pin Name		✓	Pin name
DLY_RST_PIN	Reset Pin Name		✓	Pin name
DLY_INC_PIN	Control Pin Name		✓	Pin name
IN_PIN	Input Pin Name	✓	✓	Pin name
IN_HI_PIN	Input Pin Name (HI)	✓	✓	Pin name
IN_LO_PIN	Input Pin Name (LO)	✓	✓	Pin name
IS_INCLK_INVERTED	Input Clock Inverted	✓	✓	0, 1
IN_CLK_PIN	Input Clock Pin Name	✓	✓	Pin name
IN_REG	Not applicable	✓		BYPASS, REG, DDIO, DDIO_RESYNC
			✓	BYPASS, REG, DDIO, DDIO_RESYNC, DDIO_RESYNC_PIPE, SERIAL
INDELAY	Static Delay Setting		✓	0 - 63
INDELAY_DYN_MODE	Enable Dynamic Delay		✓	0, 1
INFASTCLK_PIN	Serial Clock Pin Name		✓	Pin name
PULL_UP_ENA_PIN	Dynamic Pull Up Enable Pin Name		✓	Pin name
GBUF_PIN	Global Pin Name		✓	Pin name

GPIO Output and Output Enable Properties

You can change these properties for GPIO and buses. You cannot change these properties for bus members.

Table 5: GPIO Output and Output Enable Properties

API Name	GUI Name	Trion	Titanium	Values	Notes
OUT_PIN	Output Pin Name	✓	✓	Pin name	
OUT_HI_PIN	Output Pin Name (HI)	✓	✓	Pin name	
OUT_LO_PIN	Output Pin Name (LO)	✓	✓	Pin name	
IS_OUTCLK_INVERTED	Output Clock Inverted	✓	✓	0, 1	
OUT_CLK_PIN	Output Clock Pin Name	✓	✓	Pin name	
OUT_REG	Not applicable	✓		BYPASS, REG, INVREG, DDIO, DDIO_RESYNC	
			✓	BYPASS, REG, DDIO, DDIO_RESYNC, DDIO_RESYNC_PIPE, SERIAL	
OUTDELAY	Static Delay Setting		✓	0 - 15	
OUTFASTCLK_PIN	Serial Clock Pin Name		✓	Pin name	
OE_CLK_PIN_INV	–	✓	✓	0, 1	Deprecated in Efinity v2023.1. Use IS_OUTCLK_INVERTED.
OE_CLK_PIN	–	✓	✓	Pin name	Deprecated in Efinity v2023.1. Use OUT_CLK_PIN.
OEN_PIN	OEN Pin Name		✓	Pin name	
OE_PIN	OE Pin Name	✓	✓	Pin name	
OE_REG	Not applicable	✓	✓	BYPASS, REG	

Table 6: Deprecated GPIO Output and Output Enable Properties

API Name	Deprecated In	Replacement
OE_CLK_PIN_INV	2023.1	IS_OUTCLK_INVERTED
OE_CLK_PIN	2023.1	OUT_CLK_PIN

GPIO Bus Properties

All of these GPIO bus properties are applicable to the Trion and Titanium families.

API Name	GUI Name	Values	Notes
BUS_MODE	Bus Mode	INPUT, OUTPUT, INOUT, CLKOUT, NONE	You set the mode when you create a bus block.
BUS_LSB	Bus LSB		You can change the LSB.
BUS_MSB	Bus MSB		You can change the MSB.
BUS_NAME	Bus Name	Pin name	You specify the name when you create the bus.

GPIO General Properties

You specify the properties in the following table when you create the GPIO or bus member. You cannot change it. Instead, delete the block and create a new one.

Table 7: General Properties You Set when You Create a Block

API Name	GUI Name	Trion	Titanium	Values	Mode
CONN_TYPE[5]	Connection Type	✓		GCLK, GCTRL, PLL_CLKIN, MIPI_CLKIN	IN, INOUT, CLKIN
			✓	GCLK, RCLK, PLL_CLKIN, PLL_EXTFB, MIPI_CLKIN, VREF	IN, INOUT, CLKIN
NAME	Instance Name	✓	✓	Instance name	All
MODE	Mode	✓	✓	INPUT , OUTPUT, INOUT, CLKOUT, NONE	All

[5] For bus members, you can change the connection type.

Table 8: General Properties You Can Change

API Name	GUI Name	Trion	Titanium	Values	Mode
BUS_HOLD	Enable Bus Hold		✓	0, 1	IN, INOUT
CONST_OUTPUT[6]	Constant Output	✓	✓	NONE, 0, 1	OUT, INOUT
DRIVE_STRENGTH	Drive Strength	✓		1, 2, 3, 4	OUT, INOUT, CLKOUT
			✓	2, 4, 6, 8, 10, 12, 16	OUT, INOUT, CLKOUT
IO_STANDARD	I/O Standard	✓		3.3_V_LVTTL_/_LVCMOS 2.5_V_LVCMOS 1.8_V_LVCMOS	All
			✓	1.2_V_Differential_HSTL 1.2_V_Differential_SSTL 1.2_V_HSTL 1.2_V_LVCMOS 1.2_V_SSTL 1.5_V_Differential_HSTL 1.5_V_Differential_SSTL 1.5_V_HSTL 1.5_V_LVCMOS 1.5_V_SSTL 1.8_V_Differential_HSTL 1.8_V_Differential_SSTL 1.8_V_HSTL 1.8_V_LVCMOS 1.8_V_SSTL 2.5_V_LVCMOS 3.0_V_LVCMOS 3.0_V_LVTTL 3.3_V_LVCMOS 3.3_V_LVTTL	All
PULL_OPTION	Pull Option	✓		NONE WEAK_PULLUP WEAK_PULLDOWN	IN, INOUT, CLKIN
			✓	NONE WEAK_PULLUP WEAK_PULLDOWN DYNAMIC	IN, INOUT, CLKIN
RESOURCE	GPIO Resource	✓	✓	Resource name	All
SCHMITT_TRIGGER	Enable Schmitt Trigger	✓	✓	0,1	IN, INOUT, CLKIN
SLEW_RATE	Enable Slew Rate	✓	✓	0,1	OUT, INOUT
UNUSED_STATE[7]	Unused State	✓	✓	INPUT_WITH_WEAK_PULLUP INPUT_WITH_WEAK_PULLDOWN	IN

[6] You can change this property for GPIO and bus members.
[7] This property only applies to GPIO.

I/O Bank Property Reference

The I/O bank device setting has the following properties.

Table 9: I/O Bank Properties

API Name	GUI Name	Trion	Titanium	Values
DYNAMIC_VOLTAGE	Enable Dynamic Voltage		✓	0, 1
MODE_SEL_PIN	Mode Select Pin Name		✓	Pin name
NAME	I/O Bank	✓	✓	Bank name (see data sheet for bank names)
VOLTAGE	I/O Voltage	✓	✓	1.2, 1.5, 1.8, 2.5, 3.0, 3.3 (see data sheet for supported voltages by bank)

JTAG Property Reference

The JTAG User Tap block has the following properties.

Table 10: JTAG Properties

API Name	GUI Name	Values
CAPTURE	Input Pin - Capture Pin Name	Pin name
DRCK	Input Pin - Gated Test Clock Pin Name	Pin name
RESET	Input Pin - Reset Pin Name	Pin name
RUNTEST	Input Pin - Run Test Pin Name	Pin name
SEL	Input Pin - User Instruction Active Pin Name	Pin name
SHIFT	Input Pin - Shift Pin Name	Pin name
TCK	Input Pin - Test Clock Pin Name	Pin name
TDI	Input Pin - Test Data Pin Name	Pin name
TDO	Output Pin - Test Data Pin Name	Pin name
TMS	Input Pin - Test Mode Select Pin Name	Pin name
UPDATE	Input Pin - Update Pin Name	Pin name

LVDS Property Reference

Table 11: LVDS General Properties (All Modes)

API Name	GUI Name	Trion	Titanium	Values
NAME	Instance Name	✓	✓	Instance name
RESOURCE	LVDS Resource	✓	✓	Resource name

Table 12: LVDS TX Properties (LVDS_TX and LVDS_BIDIR Modes)

API Name	GUI Name	Trion	Titanium	Values
TX_DELAY	Static Mode Delay Setting		✓	0 - 63
TX_DIFF_TYPE	Output Differential Type		✓	LVDS, SUBLVDS, CUSTOM
TX_EN_SER	Enable Serialization	✓	✓	0, 1
TX_FASTCLK_PIN	Serial Clock Pin Name	✓	✓	Pin name
TX_HALF_RATE	Enable Half Rate Serialization		✓	0, 1
TX_MODE	Mode	✓	✓	DATA, CLKOUT
TX_OE_PIN	Output Enable Pin Name	✓	✓	Pin name
TX_OUT_PIN	Output Pin/Bus Name	✓	✓	Pin name
TX_OUTPUT_LOAD	Output Load	✓		3, 5, 7, 10
TX_PRE_EMP	Output Pre-Emphasis		✓	LOW, MEDIUM_LOW, MEDIUM_HIGH, HIGH
TX_REDUCED_SWING	Reduced VOD Swing	✓		0, 1
TX_RST_PIN	Reset Pin Name		✓	Pin name
TX_SER	Serialization Width	✓		2, 3, 4, 5, 6, 7, 8
			✓	1, 2, 3, 4, 5, 6, 7, 8, 10
TX_SLOWCLK_DIV	Parallel Clock Division	✓		1, 2
TX_SLOWCLK_PIN	Parallel Clock Pin Name	✓	✓	Pin name
TX_VOD	Output Differential, VOD		✓	LARGE, TYPICAL, SMALL

Table 13: LVDS RX Properties (LVDS_RX and LVDS_BIDIR Modes)

API Name	GUI Name	Trion	Titanium	Values
GBUF	Global Pin Name[8]		✓	Pin name
RX_CONN_TYPE	Connection Type	✓		NORMAL, PLL_CLKIN
			✓	NORMAL, GCLK, RCLK, PLL_CLKIN, PLL_EXTFB
RX_DBG_PIN	DPA Debug Bus Name		✓	Pin name
RX_DELAY	Static Mode Delay Setting		✓	0 - 63
	Static Delay Setting	✓		0 - 63
RX_DELAY_MODE	Delay Mode		✓	STATIC, DYNAMIC, DPA
RX_DESER	Serialization Width	✓		2, 3, 4, 5, 6, 7, 8
			✓	1, 2, 3, 4, 5, 6, 7, 8, 10
RX_DLY_ENA_PIN	Dynamic Enable Pin Name		✓	Pin name
RX_DLY_INC_PIN	Dynamic Delay Control Pin Name		✓	Pin name
RX_DLY_RST_PIN	Dynamic Reset Delay Pin Name		✓	Pin name
RX_EN_DELAY	Enable Delay Setting	✓		0, 1
RX_EN_DESER	Enable Deserialization	✓	✓	0, 1
RX_ENA_PIN	Enable Pin Name		✓	Pin name
RX_FASTCLK_PIN	Serial Clock Pin Name	✓	✓	Pin name
RX_FIFO	Enable Clock Crossing FIFO		✓	0, 1
RX_FIFO_EMPTY_PIN	FIFO Empty Pin Name		✓	Pin name
RX_FIFO_RD_PIN	Enable FIFO Read Pin Name		✓	Pin name
RX_FIFOCLK_PIN	FIFO Clock Pin Name		✓	Pin name
RX_HALF_RATE	Enable Half Rate Serialization		✓	0, 1
RX_IN_PIN	Input Pin/Bus Name	✓	✓	Pin name
RX_LOCK_PIN	DPA Lock Pin Name		✓	Pin name
RX_RST_PIN	Reset Pin Name		✓	Pin name
RX_SLOWCLK_PIN	Parallel Clock Pin Name	✓	✓	Pin name
RX_TERM	Termination		✓	ON, OFF, DYNAMIC
	On-Die LVDS Termination	✓		0, 1
RX_TERM_PIN	Termination Pin Name		✓	Pin name
RX_VOC_DRIVER	Enable Output Common Mode Driver		✓	0, 1

[8] GUI is available in CLKMUX under Regional Buffers tab.

DDR Property Reference

Trion DDR Properties

Table 14: DDR General Properties

API Name	GUI Name	Values
CLK_NAME	Clock Instance	Instance name
CLK_PIN	Clock Pin Name	PLL output clock 0 name
CLK_RESOURCE	Clock Resource	PLL resource
MEMORY_TYPE	Memory Type	DDR3, LPDDR2, LPDDR3
NAME	Instance Name	Instance Name
RESOURCE	DDR Resource	None, DDR_0

Table 15: DDR Configuration Properties

API Name	GUI Name	Values		
		DDR3	LPDDR3	LPDDR2
ADV_DENSITY_EN	Enable Advanced Density Setting	0, 1	0, 1	0, 1
COLUMN_WIDTH	Column Width	0 - 99	0 - 99	0 - 99
DQ_WIDTH	DQ Width	x16	x16	x16
MEMORY_SPEED	Speed Grade	800D, 800E, 1066E, 1066F, 1066G	800, 1066	400, 533, 667, 800, 1066
MEMORY_WIDTH	Width	x8, x16	x16	x16
MEMORY_DENSITY	Density	1G, 2G, 4G, 8G	4G, 8G	256M, 512M, 1G, 2G, 4G
PRESET	Preset[9]	153, 155, 157, 159, 161, 163, 165, 167, 169, 171	200	175, 177, 179, 181, 183
ROW_WIDTH	Row Width	0 - 99	0 - 99	0 - 99

Table 16: DDR Advanced Options (FPGA Settings) Properties

API Name	GUI Name	Values		
		DDR3	LPDDR3	LPDDR2
FPGA_INPUT	FPGA Input Termination (Ohm)	20, 30, 40, 60, 120	120, 240, OFF	120, OFF
FPGA_OUTPUT	FPGA Output Termination (Ohm)	34, 40	34, 40, 48, 60, 80	34, 40, 48, 60, 80, 120

[9] Use **get_preset()** command to obtain preset names.

Table 17: DDR Advanced Options (Memory Mode Register Settings) Properties

API Name	GUI Name	Values		
		DDR3	LPDDR3	LPDDR2
ASR	Auto Self-Refresh	Manual, Auto		
BL	Burst Length	8		8
CL	CAS Latency	5-14		
CWL	CAS Write Latency	5-12		
DQ_ODT	DQ Termination		Disable, RZQ/1, RZQ/2, RZQ/4	
MOUTPUT	Output Termination	RZQ/6, RZQ/7		
OUT_DRIVE_STR	Output Drive Strength (Ohm)		34.3, 34.3 pull-down / 40 pull-up, 34.3 pull-down / 48 pull-up, 40, 40 pull-down / 48 pull-up, 48	34.3, 40, 48, 60, 80, 120
PRECHARGE_PD	DLL Precharge Power Down	On, Off		
READ_BURST_TYPE	Read Burst Type	Interleaved, Sequential		Interleaved, Sequential
RL/WL	Read/Write Latency		RL=3/WL=1, RL=6/WL=3, RL=8/WL=4, RL=9/WL=5	RL=3/WL=1, RL=4/WL=2, RL=5/WL=2, RL=6/WL=3, RL=7/WL=4, RL=8/WL=4
RTT_WR	Memory Dynamic ODT	Off, RZQ/2, RZQ/4		
RTT_NOM	Input Termination	Off, RZQ/2, RZQ/4, RZQ/6, RZQ/8, RZQ/12		
SRT	Self-Refresh Temperature	Normal, Extended		

Table 18: DDR Advanced Options (Memory Timing Settings) Properties

API Name	GUI Name	Values
TFAW	tFAW, Four Bank Active Window (ns)	20.0 - 100.0
TRAS	tRAS, Active To Precharge Command Period (ns)	20.0 - 100.0
TRC	tRC, Active To Active Or REF Command Period (ns)	20.0 - 100.0
TRCD	tRCD, Active To Read Or Write Delay (ns)	2.0 - 50.0
TREFI	tREFI, Average Periodic Refresh Interval (ns)	2.0 - 30.0
TRFC	tRFC, Refresh to Active Or Refresh to Refresh Delay (ns)	90.0 - 960.0
TRP	tRP, PRecharfe Cpmmand Period (ns)	8.0 - 50.0
TRRD	tRRD, Active to Active Command Period (ns)	2.0 - 50.0
TRTP	tRTP, Internal Read To Precharge Delay (ns)	2.0 - 50.0
TWTR	tWTR, Internal Write to Read Command Delay (ns)	2.0 - 100.0

Table 19: DDR Advanced Options (Controller Settings) Properties

API Name	GUI Name	Values
CONTROL_MAP	Controller To Memory Address Mapping	ROW-COL_HIGH-BANK-COL_LOW, ROW-BANK-COL, BANK-ROW-COL
CONTROL_REFRESH_EN	Enable Self-Refresh Controls	No, Yes
POWER_DOWN_EN	Enable Auto Power Down	Off, Active, Precharge

Table 20: DDR Advanced Options (Gate Delay Tuning Settings) Properties

API Name	GUI Name	Values
GCOARSE_DELAY	Gate Coarse Delay Tuning	0 - 5
GDELAY_OVERRIDE_EN	Enable Gate Delay Override	0, 1
GFINE_DELAY	Gate Fine Delay Tuning	0 - 255

Table 21: DDR Control Properties

API Name	GUI Name	Values
CONTROL_TYPE	Type	Disable, Calibration, User Reset, Reset and Calibration
RESET_PIN	Master Reset Pin Name	Pin name
SCL_IN_PIN	SCL Input Pin Name	Pin name
SDA_IN_PIN	SDA Input Pin Name	Pin name
SDA_OUT_PIN	SDA Output Pin Name	Pin name
SEQ_RESET_PIN	Sequencer Reset Pin Name	Pin name
SEQ_START_PIN	Sequencer Start Pin Name	Pin name

Table 22: DDR AXI Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_CLK_INPUT_PIN	AXI Clock Input Pin Name	Pin name
AXI*n*_CLK_INVERT_EN	Invert AXI Clock Input	0, 1
AXI*n*_AID_BUS	Address ID [7:0] Bus Name	Pin name
AXI*n*_AREADY_PIN	Address Ready Pin Name	Pin name
AXI*n*_AVALID_PIN	Address Valid Pin Name	Pin name
AXI*n*_ABUS	Address [31:0] Bus Name	Pin name
AXI*n*_ABURST_LEN_BUS	Burst Length [7:0] Bus Name	Pin name
AXI*n*_ABURST_SIZE_BUS	Burst Size [2:0] Bus Name	Pin name
AXI*n*_ABURST_BUS	Burst Type [1:0] Bus Name	Pin name
AXI*n*_ALOCK_BUS	Lock Type [1:0] Bus Name	Pin name
AXI*n*_AOP_TYPE_PIN	Operation Type Pin Name	Pin name
AXI*n*_BID_BUS	Response ID [7:0] Bus Name	Pin name
AXI*n*_BREADY_PIN	Response Ready Pin Name	Pin name
AXI*n*_BVALID_PIN	Write Response Valid Pin Name	Pin name
AXI*n*_RDATA_BUS	Read Data Bus Name	Pin name
AXI*n*_RID_BUS	Read ID Bus Name	Pin name
AXI*n*_RLAST_PIN	Read Last Pin Name	Pin name
AXI*n*_RREADY_PIN	Read Ready Pin Name	Pin name
AXI*n*_RRESP_BUS	Read Response Bus Name	Pin name
AXI*n*_RVALID_PIN	Read Valid Pin Name	Pin name
AXI*n*_WDATA_BUS	Write Data Bus Name	Pin name
AXI*n*_WID_BUS	Write ID Bus Name	Pin name
AXI*n*_WLAST_PIN	Write Last Pin Name	Pin name
AXI*n*_WREADY_PIN	Write Ready Pin Name	Pin name
AXI*n*_WSTRB_BUS	Write Strobes Bus Name	Pin name
AXI*n*_WVALID_PIN	Write Valid Pin Name	Pin name
TARGET*n*_EN	Enable Target n	0, 1

Titanium DDR Properties

Table 23: DDR General Properties

API Name	GUI Name	Values
CLK_NAME	Clock Instance	Instance name
CLK_PIN	Clock Pin Name	PLL output clock 0 name
CLK_RESOURCE	Clock Resource	PLL resource
MEMORY_TYPE	Memory Type	LPDDR4, LPDDR4x
NAME	Instance Name	Instance Name
RESOURCE	DDR Resource	None, DDR_0

Table 24: DDR Configuration Properties

API Name	GUI Name	Values	
		LPDDR4	LPDDR4x
DQ_WIDTH	DQ Width	x16, x32	x16, x32
MEMORY_DENSITY	Density	2G, 3G 4G, 6G, 8G, 12G, 16G	2G, 3G 4G, 6G, 8G, 12G, 16G
PHYSICAL_RANK	Physical Rank	1, 2	1, 2

Table 25: DDR Advanced Options (FPGA Settings) Properties

API Name	GUI Name	Values	
		LPDDR4	LPDDR4x
VREF_RANGE	VREF Range	Range 0, Range 1	Range 0, Range 1
VREF_SETTING	VREF Settings (% of VDDQ)	**Range 0:** 5.40 - 38.42 **Range 1:** 11.90 - 48.22	**Range 0:** 11.60 - 49.70 **Range 1:** 21.20 - 59.30
DQ_PD_DRV_STRENGTH	DQ Pull-Down Drive Strength (Unit: Ohm)	34.3, 40, 48, 60, 80, 120, 240	34.3, 40, 48, 60, 80, 120, 240
DQ_PD_ODT	DQ Pull-Down ODT (Unit: Ohm)	34.3, 40, 48, 60, 80, 120, 240, Hi-Z	34.3, 40, 48, 60, 80, 120, 240, Hi-Z
DQ_PU_DRV_STRENGTH	DQ Pull-Up Drive Strength (Unit: Ohm)	34.3, 40, 48, 60, 80, 120, 240	34.3, 40, 48, 60, 80, 120, 240
DQ_PU_ODT	DQ Pull-Up ODT (Unit: Ohm)	34.3, 40, 48, 60, 80, 120, 240, Hi-Z	34.3, 40, 48, 60, 80, 120, 240, Hi-Z

Table 26: DDR Advanced Options (Memory Mode Register Settings) Properties

API Name	GUI Name	Values
BL	Burst Length	BL = 16 Sequential, BL = 16 or 32 Sequential, BL = 32 Sequential
CA_ODT_CS0	CA Bus Receiver On-Die Termination for CS0	Disable, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
CA_ODT_CS1	CA Bus Receiver On-Die Termination for CS1	Disable, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
DBI_READ_EN	Enable DBI Read	0: Disable read data bus inversion 1:Enable read data bus inversion
DBI_WRITE_EN	Enable DBI Write	0: Disable write data bus inversion 1:Enable write data bus inversion
DQ_ODT_CS0	DQ Bus Receiver On-Die Termination for CS0	Disable, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
DQ_ODT_CS1	DQ Bus Receiver On-Die Termination for CS1	Disable, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
PDDS_CS0	Pull-Down Drive Strength for CS0	RFU, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
PDDS_CS1	Pull-Down Drive Strength for CS1	RFU, RZQ/1, RZQ/2, RZQ/3, RZQ/4, RZQ/5, RZQ/6
CA_VREF_RANGE	CA VREF Setting Range Selection	RANGE [0], RANGE [1]
CA_VREF_SETTING	CA VREF Settings (% of VDD2)	**RANGE [0]:** 10 - 30 (step: 0.4) **RANGE [1]:** 22 - 42 (step: 0.4)
DQ_VREF_RANGE	DQ VREF Setting Range Selection	RANGE [0], RANGE [1]
DQ_VREF_SETTING	DQ VREF Settings (% of VDDQ)	**RANGE [0]:** 10 - 30 (step: 0.4) **RANGE [1]:** 22 - 42 (step: 0.4)
CK_ODTE_CS0	CK ODT CS0 Enabled for Non-terminating Rank	Override Disabled, Override Enabled
CK_ODTE_CS1	CK ODT CS1 Enabled for Non-terminating Rank	Override Disabled, Override Enabled
CS_ODTE_CS0	CS ODT CS0 Enabled for Non-terminating Rank	Override Disabled, Override Enabled
CS_ODTE_CS1	CS ODT CS1 Enabled for Non-terminating Rank	Override Disabled, Override Enabled
CA_ODTD_CS0	CA ODT CS0 Termination Disable	Obeys ODT_CA Bond Pad, Disabled
CA_ODTD_CS1	CA ODT CS1 Termination Disable	Obeys ODT_CA Bond Pad, Disabled

Table 27: DDR Advanced Options (Memory Timing Settings) Properties

API Name	GUI Name	Values
TFAW	tFAW, Four Bank Active Window (ns)	40.0 - 100.0
TRAS	tRAS, Active To Precharge Command Period (ns)	42.0 - 100.0
TRCD	tRCD, Active To Read Or Write Delay (ns)	18.0 - 100.0
TRRD	tRRD, Active to Active Command Period (ns)	10.0 - 100.0
TRTP	tRTP, Internal Read To Precharge Delay (ns)	7.5 - 100.0
TWTR	tWTR, Internal Write to Read Command Delay (ns)	10.0 - 60.0
TCCD	tCCD, CAS-to-CAS Delay	Integer 8 - 31
TCCDMW	tCCDMW, CAS-to-CAS Delay Masked Write	Integer 32 - 63
TPPD	tPPD, Precharge to Precharge Delay (cycles)	Integer 4 - 7
TRPAB	tRPab, Row Precharge Time (All Banks) (ns)	21.0 - 100.0
TRPPB	tRPpb, Row Precharge Time (Single Bank) (ns)	18.0 - 100.0
TSR	tSR, Minimum Self Refresh Time (ns)	15.0 - 100.0
TWR	tWR, Write Recovery Time (ns)	18.0 - 60.0

Table 28: DDR Config Controller Properties

API Name	GUI Name	Values
CFG_DONE_PIN	Config Controller Done Pin	Pin name
CFG_RESET_PIN	Config Controller Reset Pin	Pin name
CFG_SEL_PIN	Config Controller Select Pin	Pin name
CFG_START_PIN	Config Controller Start Pin	Pin name

Table 29: DDR Controller Status Properties

API Name	GUI Name	Values
CTRL_CLK_PIN	Controler Status Clock Pin	Pin name
CTRL_CLK_INVERT_EN	Invert Controller Status Clock	Pin name
CTRL_INT_PIN	Controller Detects Interrupt Pin Name	Pin name
CTRL_MEM_RST_VALID_PIN	Controller Has Reseted Pin Name	Pin name
CTRL_REFRESH_PIN	Controller Refresh Command Pin Name	Pin name
CTRL_CKE_PIN	Delayed CKE From Controller	Pin name
CTRL_BUSY_PIN	Controller Busy PIn Name	Pin name
CTRL_CMD_Q_ALMOST_FULL_PIN	Command Queue Full Pin Name	Pin name
CTRL_DP_IDLE_PIN	Data Path Idle Pin Name	Pin name
CTRL_PORT_BUSY_PIN	Port Busy Reading Data Pin Name	Pin name

Table 30: DDR AXI Properties

n is 0 or 1

API Name	GUI Name	Values
TARGET*n*_EN	Enable Target n	0, 1
AXI*n*_DATA_WIDTH	Data Width n	512
AXI*n*_CLK_INPUT_PIN	AXI Clock Input Pin Name	Pin name
AXIn_CLK_INVERT_EN	Invert AXI Clock Input	0, 1
AXIn_ARSTN_PIN	AXI Reset Pin Name	Pin name

Table 31: DDR AXI (Read Address Channel) Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_ARID_BUS	Address ID [5:0] Bus Name	Pin name
AXI*n*_ARREADY_PIN	Address Ready Pin Name	Pin name
AXI*n*_ARVALID_PIN	Address Valid Pin Name	Pin name
AXI*n*_ARLEN_BUS	Burst Length [7:0] Bus Name	Pin name
AXI*n*_ARSIZE_BUS	Burst Size [2:0] Bus Name	Pin name
AXI*n*_ARBURST_BUS	Burst Type [1:0] Bus Name	Pin name
AXI*n*_ARLOCK_PIN	Lock Type Bus Name	Pin name
AXI*n*_ARQOS_PIN	QoS Identifier for Read Transaction Pin Name	Pin name
AXI*n*_ARADDR_BUS	Read Address [32:0] Bus Name	Pin name
AXI*n*_ARAPCMD_PIN	Read Auto-Precharge Pin Name	Pin name

Table 32: DDR AXI (Write Address Channel) Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_AWID_BUS	Address ID [5:0] Bus Name	Pin name
AXI*n*_AWREADY_PIN	Address Ready Pin Name	Pin name
AXI*n*_AWVALID_PIN	Address Valid Pin Name	Pin name
AXI*n*_AWLEN_BUS	Burst Length [7:0] Bus Name	Pin name
AXI*n*_AWSIZE_BUS	Burst Size [2:0] Bus Name	Pin name
AXI*n*_AWBURST_BUS	Burst Type [1:0] Bus Name	Pin name
AXI*n*_AWLOCK_PIN	Lock Type Bus Name	Pin name
AXI*n*_AWCACHE_BUS	Memory Type [3:0] Bus Name	Pin name
AXI*n*_AWQOS_PIN	QoS Identifier for Write Transaction Pin Name	Pin name
AXI*n*_AWADDR_BUS	Write Address [32:0] Bus Name	Pin name
AXI*n*_AWALLSTRB_PIN	Write All Strobes Asserted Pin Name	Pin name
AXI*n*_AWAPCMD_PIN	Write Auto-Precharge Pin Name	Pin name
AXI*n*_AWCOBUF_PIN	Write Coherent Bufferable Selection Pin Name	Pin name

Table 33: DDR AXI (Write Response Channel) Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_BID_BUS	Response ID [7:0] Bus Name	Pin name
AXI*n*_BREADY_PIN	Response Ready Pin Name	Pin name
AXI*n*_BVALID_PIN	Write Response Valid Pin Name	Pin name
AXI*n*_BRESP_BUS	Write Response [1:0] Bus Name	Pin name

Table 34: DDR AXI (Read Data Channel) Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_RDATA_BUS	Read Data Bus Name	Pin name
AXI*n*_RID_BUS	Read ID Bus Name	Pin name
AXI*n*_RLAST_PIN	Read Last Pin Name	Pin name
AXI*n*_RREADY_PIN	Read Ready Pin Name	Pin name
AXI*n*_RRESP_BUS	Read Response Bus Name	Pin name
AXI*n*_RVALID_PIN	Read Valid Pin Name	Pin name

Table 35: DDR AXI (Write Data Channel) Properties

n is 0 or 1

API Name	GUI Name	Values
AXI*n*_WDATA_BUS	Write Data Bus Name	Pin name
AXI*n*_WLAST_PIN	Write Last Pin Name	Pin name
AXI*n*_WREADY_PIN	Write Ready Pin Name	Pin name
AXI*n*_WSTRB_BUS	Write Strobes Bus Name	Pin name
AXI*n*_WVALID_PIN	Write Valid Pin Name	Pin name

Table 36: Pin Swizzling Properties

API Name	GUI Name	Values
PIN_SWIZZLE_EN	Enable Package Pin Swapping	0, 1
PIN_SWIZZLE_DQM0	DQ/DM Pin Swizzle Group0	STRING: "DQ[0],DQ[1],DQ[2],DQ[3],DQ[4],DQ[5],DQ[6],DQ[7],DM[0]"[10]
PIN_SWIZZLE_DQM1	DQ/DM Pin Swizzle Group1	STRING: "DQ[8],DQ[9],DQ[10],DQ[11],DQ[12],DQ[13],DQ[14],DQ[15],DM[1]"[10]
PIN_SWIZZLE_DQM2	DQ/DM Pin Swizzle Group2	STRING: "DQ[16],DQ[17],DQ[18],DQ[19],DQ[20],DQ[21],DQ[22],DQ[23],DM[2]"[10]
PIN_SWIZZLE_DQM3	DQ/DM Pin Swizzle Group3	STRING: "DQ[24],DQ[25],DQ[26],DQ[27],DQ[28],DQ[29],DQ[30],DQ[31],DM[3]"[10]
PIN_SWIZZLE_CA	Address Pin Swizzle	STRING:"CA[0],CA[1],CA[2],CA[3],CA[4],CA[5]"[10]

[10] The sequence of items in the strings determines the swizzling of pin. Left most item is the smallest index and the right most is the biggest.

Remote Update Property Reference

The remote update block has the following properties.

Table 37: Remote Update Properties

API Name	GUI Name	Values
CBSEL_PIN	Image Selector [1:0] Bus Name	Pin name
CLK_PIN	Clock Pin Name	Pin name
CONFIG_PIN	Configuration Control Pin Name	Pin name
ENA_PIN	Image Selector Capture Pin Name	Pin name
ERROR_PIN	Error Status Pin Name	Pin name
IN_USER_PIN	In User Pin Name	Pin name
INVERT_CLK_EN	Invert Clock	0, 1
RECONFIG_EN	Enable Internal Reconfiguration Interface	0, 1

SEU Property Reference

The single-event upset (SEU) properties are applicable to the Titanium family only. The Trion family does not support SEU.

Table 38: SEU Properties

API Name	GUI Name	Values
CONFIG_PIN	Reconfiguration Pin Name	Pin name
DONE_PIN	SEU Done Detection Pin Name	Pin name
ENA_DETECT	Enable SEU Detection	0, 1
ERROR_PIN	Error Status Pin Name	Pin name
INJECT_ERROR_PIN	Error Injection Pin Name	Pin name
MODE	Mode	AUTO, MANUAL
RST_PIN	Error Reset Pin Name	Pin name
START_PIN	SEU Start Detection Pin Name	Pin name
WAIT_INTERVAL	Wait Interval	0 - 0xffffff (0.0 - 1677721.5 µs)

Clock Multiplexer Property Reference

The clock multiplexer properties are applicable to the Titanium family only.

Table 39: Clock multiplexer Properties

Use **get_all_global_mux_name()** on page 28 to get the instance name.

API Name	GUI Name	Values
DYN_MUX0_EN	Enable Dynamic Mux 0	0, 1
DYN_MUX7_EN	Enable Dynamic Mux 7	0, 1
CORE0_PIN	Core Clock 0 Pin Name	Pin name
CORE1_PIN	Core Clock 1 Pin Name	Pin name
CORE2_PIN	Core Clock 2 Pin Name	Pin name
CORE3_PIN	Core Clock 3 Pin Name	Pin name
DYN_MUX0_SEL_PIN	Dynamic Mux 0 - Dynamic Clock Mux Select Bus Name	Pin name
DYN_MUX0_OUT_PIN	Dynamic Mux 0 - Dynamic Clock Pin Name	Pin name
DYN_MUX7_SEL_PIN	Dynamic Mux 7 - Dynamic Clock Mux Select Bus Name	Pin name
DYN_MUX7_OUT_PIN	Dynamic Mux 7 - Dynamic Clock Pin Name	Pin name
DYN_MUX0_IN0_TO_CORE_EN	Dynamic Mux 0 - Dynamic Clock Input 0 - Independently Connect To Core	0, 1
DYN_MUX0_IN1_TO_CORE_EN	Dynamic Mux 0 - Dynamic Clock Input 1 - Independently Connect To Core	0, 1
DYN_MUX0_IN2_TO_CORE_EN	Dynamic Mux 0 - Dynamic Clock Input 2 - Independently Connect To Core	0, 1
DYN_MUX0_IN3_TO_CORE_EN	Dynamic Mux 0 - Dynamic Clock Input 3 - Independently Connect To Core	0, 1
DYN_MUX7_IN0_TO_CORE_EN	Dynamic Mux 7 - Dynamic Clock Input 0 - Independently Connect To Core	0, 1
DYN_MUX7_IN1_TO_CORE_EN	Dynamic Mux 7 - Dynamic Clock Input 1 - Independently Connect To Core	0, 1
DYN_MUX7_IN2_TO_CORE_EN	Dynamic Mux 7 - Dynamic Clock Input 2 - Independently Connect To Core	0, 1
DYN_MUX7_IN3_TO_CORE_EN	Dynamic Mux 7 - Dynamic Clock Input 3 - Independently Connect To Core	0, 1
DYN_MUX0_IN0	Dynamic Mux 0 - Dynamic Clock Input 0	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX0_IN1	Dynamic Mux 0 - Dynamic Clock Input 1	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX0_IN2	Dynamic Mux 0 - Dynamic Clock Input 2	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX0_IN3	Dynamic Mux 0 - Dynamic Clock Input 3	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX7_IN0	Dynamic Mux 7 - Dynamic Clock Input 0	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX7_IN1	Dynamic Mux 7 - Dynamic Clock Input 1	0, 1, 2, 3, 4, 5, 6[11]

[11] Use **get_global_dynamic_mux_input_info()** on page 32 for information about the available input.

API Name	GUI Name	Values
DYN_MUX7_IN2	Dynamic Mux 7 - Dynamic Clock Input 2	0, 1, 2, 3, 4, 5, 6[11]
DYN_MUX7_IN3	Dynamic Mux 7 - Dynamic Clock Input 3	0, 1, 2, 3, 4, 5, 6[11]

HyperRAM Property Reference

All of these HyperRAM block properties are applicable to Titanium FPGAs in F100S3F2 packages only.

Table 40: HyperRAM Properties

API Name	GUI Name	Values
CK_N_HI_PIN	Differential Clock Pin Name (N HI)	Pin name
CK_N_LO_PIN	Differential Clock Pin Name (N LO)	Pin name
CK_P_HI_PIN	Differential Clock Pin Name (P HI)	Pin name
CK_P_LO_PIN	Differential Clock Pin Name (P LO)	Pin name
CLK_PIN	HyperRAM Controller Clock Pin Name	Pin name
CLK90_PIN	90 Degree Phase-Shifted Clock Pin Name	Pin name
CLKCAL_PIN	Calibration Clock Pin Name	Pin name
CS_N_PIN	Active-Low HyperRAM Chip Select Pin Name	Pin name
DQ_IN_HI_PIN	DQ Input [15:0] Bus Name (HI)	Pin name
DQ_IN_LO_PIN	DQ Input [15:0] Bus Name (LO)	Pin name
DQ_OE_PIN	DQ Output Enable [15:0] Bus Name	Pin name
DQ_OUT_HI_PIN	DQ Output [15:0] Bus Name (HI)	Pin name
DQ_OUT_LO_PIN	DQ Output [15:0] Bus Name (LO)	Pin name
NAME	Instance Name	Instance Name
RESOURCE	HyperRAM Resource	Pin name
RST_N_PIN	Active-Low HyperRAM Reset Pin Name	Pin name
RWDS_IN_HI_PIN	Read/Write Data Strobe Input [1:0] Bus Name (HI)	Pin name
RWDS_IN_LO_PIN	Read/Write Data Strobe Input [1:0] Bus Name (LO)	Pin name
RWDS_OE_PIN	Read/Write Data Strobe Output Enable [1:0] Bus Name	Pin name
RWDS_OUT_HI_PIN	Read/Write Data Strobe Output [1:0] Bus Name (HI)	Pin name
RWDS_OUT_LO_PIN	Read/Write Data Strobe Output [1:0] Bus Name (LO)	Pin name

SPI Flash Property Reference

All of these SPI flash block properties are applicable to Titanium FPGAs in F100S3F2 packages and Trion FPGAs in QFP100F3 packages only.

Table 41: SPI Flash Properties

API Name	GUI Name	Values
CLK_PIN	Clock Pin Name	Pin name
CS_N_OE_PIN	Flash Chip Select (Active-Low) Output Enable Pin Name	Pin name
CS_N_OUT_PIN	Flash Chip Select (Active-Low) Pin Name	Pin name
HOLD_N_IN_PIN	Hold (Active-Low) Input Pin Name	Pin name
HOLD_N_OE_PIN	Hold (Active-Low) Output Enable Pin Name	Pin name
HOLD_N_OUT_PIN	Hold (Active-Low) Pin Name	Pin name
MISO_IN_PIN	Data Input From Flash Pin Name	Pin name
MISO_OE_PIN	Data Input From Flash Output Enable Pin Name	Pin name
MISO_OUT_PIN	Data Input From Flash Output Pin Name	Pin name
MOSI_IN_PIN	Data Output To Flash Input Pin Name	Pin name
MOSI_OUT_PIN	Data Output To Flash Pin Name	Pin name
MOSI_OE_PIN	Data Output To Flash Output Enable Pin Name	Pin name
MULT_CTRL_EN	Enable Multiple Controller	0, 1
NAME	Instance Name	Instance Name
RESOURCE	SPI Flash Resource	SPI_FLASH0
REG_EN	Enable Register Interface	0, 1
RW_WIDTH	Read/Write Width	x1, x2, x4
SCLK_OE_PIN	Clock Output to Flash Output Enable Pin Name	Pin name
SCLK_OUT_PIN	Clock Output to Flash Pin Name	Pin name
WP_N_OE_PIN	Write Protect (Active-Low) Output Enable Pin Name	Pin name
WP_N_IN_PIN	Write Protect (Active-Low) Input Pin Name	Pin name
WP_N_OUT_PIN	Write Protect (Active-Low) Pin Name	Pin name

Table 42: Deprecated SPI Flash Properties

API Name	Deprecated In	Replacement
CS_N_PIN	2023.1	CS_N_OUT_PIN
ENA_OE	2023.1	MULT_CTRL_EN
HOLD_N_PIN	2023.1	HOLD_N_OUT_PIN
MISO_PIN	2023.1	MISO_IN_PIN
MOSI_PIN	2023.1	MOSI_OUT_PIN
SCLK_PIN	2023.1	SCLK_OUT_PIN
WP_N_PIN	2023.1	WP_N_OUT_PIN

External Flash Controller Property Reference

The external flash controller properties are applicable to Titanium FPGAs in F100S3F2 packages only.

Table 43: External Flash Control Properties

API Name	GUI Name	Values
EXT_FLASH_CTRL_EN	Enable external flash controller access to flash memory	0, 1

MIPI Property Reference (Trion)

All of these MIPI block properties are applicable to the Trion family only.

Table 44: MIPI (Base) Properties

API Name	GUI Name	RX	TX	Values
NAME	Instance Name	✓	✓	Instance name
RESOURCE	MIPI Resource	✓	✓	Resource name
PHY_FREQ	PHY Frequency (MHz)		✓	80.00 - 1500.00
REFCLK_FREQ	Reference Clock Frequency (MHz)		✓	6.00, 12.00, 19.20, 25.00, 26.00, 27.00, 38.40, 52.00
REFCLK_RES	Reference Clock Resource		✓	Resource name
REFCLK_INST	Reference Clock Instance		✓	Instance name
CONT_PHY_CLK_EN	Enable Continuous PHY Clocking		✓	0, 1

Table 45: MIPI (Control) Properties

API Name	GUI Name	RX	TX	Values
ESC_CLK_PIN	Escape Clock Pin Name		✓	Pin name
ESC_CLK_INVERTED_EN	Invert Escape Clock		✓	0, 1
PIXEL_CLK_PIN	Pixel Clock Pin Name	✓	✓	Pin name
PIXEL_CLK_INVERTED_EN	Invert Pixel Clock Pin Name	✓	✓	0, 1
DPHY_RSTN_PIN	DPHY Reset Pin Name	✓	✓	Pin name
RSTN_PIN	CSI-2 Reset Pin Name	✓	✓	Pin name
LANES_PIN	Lanes Bus Name	✓	✓	Pin name
CAL_CLK_PIN	DPHY Calibration Clock Pin Name	✓		Pin name
CAL_CLK_INVERTED_EN	Invert DPHY Calibration Clock	✓		0, 1
VC_ENA_PIN	Virtual Channel Enable Bus Name	✓		Pin name

Table 46: MIPI (Video ULPS) Properties

API Name	GUI Name	RX	TX	Values
ULPS_CLK_ENTER_PIN	Clock Lane Enter Pin Name		✓	Pin name
ULPS_CLK_EXIT_PIN	Clock Lane Exit Pin Name		✓	Pin name
ULPS_ENTER_PIN	Data Lane Enter [3:0] Bus Name		✓	Pin name
ULPS_EXIT_PIN	Data Lane Exit [3:0] Bus Name		✓	Pin name

Table 47: MIPI (Video) Properties

API Name	GUI Name	RX	TX	Values
FRAME_MODE_PIN	Frame Mode Pin Name		✓	Pin name
HRES_PIN	Horizontal Resolution [15:0] Bus Name		✓	Pin name
HSYNC_PIN	VC Horizontal Sync Pin Name	✓	✓	Pin name
VSYNC_PIN	VC Vertical Sync Pin Name	✓	✓	Pin name
VALID_PIN	Valid Pixel Data Pin Name	✓	✓	Pin name
TYPE_PIN	Video Data Type [5:0] Bus Name	✓	✓	Pin name
DATA_PIN	Video Data [63:0] Bus Name	✓	✓	Pin name
VC_PIN	Virtual Channel [1:0] Bus Name	✓	✓	Pin name
CNT_PIN	Valid Pixel Count Bus Name	✓		Pin name

Table 48: MIPI (Lane Mapping) Properties

API Name	GUI Name	RX	TX	Values
TXD0_LANE	TXD0		✓	clk, data0, data1, data2, data3
TXD1_LANE	TXD1		✓	clk, data0, data1, data2, data3
TXD2_LANE	TXD2		✓	clk, data0, data1, data2, data3
TXD3_LANE	TXD3		✓	clk, data0, data1, data2, data3
TXD4_LANE	TXD4		✓	clk, data0, data1, data2, data3
RXD0_LANE	RXD0	✓		clk, data0, data1, data2, data3
RXD1_LANE	RXD1	✓		clk, data0, data1, data2, data3
RXD2_LANE	RXD2	✓		clk, data0, data1, data2, data3
RXD3_LANE	RXD3	✓		clk, data0, data1, data2, data3
RXD4_LANE	RXD4	✓		clk, data0, data1, data2, data3
RXD0_PN_SWAP	RXD0 - Swap P&N Pin	✓		0, 1
RXD1_PN_SWAP	RXD1 - Swap P&N Pin	✓		0, 1
RXD2_PN_SWAP	RXD2 - Swap P&N Pin	✓		0, 1
RXD3_PN_SWAP	RXD3 - Swap P&N Pin	✓		0, 1
RXD4_PN_SWAP	RXD4 - Swap P&N Pin	✓		0, 1

Table 49: MIPI (Timing) Properties

API Name	GUI Name	RX	TX	Values
TCLK_POST	Clock Timer : T-CLK-POST (ns)		✓	113 - 1375
TCLK_TRAIL	Clock Timer : T-CLK-TRAIL (ns)		✓	16 - 175
TCLK_PREPARE	Clock Timer : T-CLK-PREPARE (ns)		✓	48 - 112
TCLK_ZERO	Clock Timer : T-CLK-ZERO (ns)		✓	304 - 687
ESC_CLK_FREQ	Clock Timer : Escape Clock Freq (MHz)		✓	11.00 - 20.00
TCLK_PRE	Clock Timer : T-CLK-PRE (ns)		✓	250 - 1000
THS_PREPARE	Data Timer : T-HS-PREPARE (ns)		✓	58 - 187
THS_ZERO	Data Timer : T-HS-ZERO (ns)		✓	142 - 875
THS_TRAIL	Data Timer : T-HS-TRAIL (ns)		✓	10 - 1562
CAL_CLK_FREQ	Calibration Clock Freq (MHz)	✓		80 - 120
TCLK_SETTLE	Clock Timer : T-CLK-SETTLE (ns)	✓		33 - 3237
THS_SETTLE	Data Timer : T-HS-SETTLE (ns)	✓		33 - 3237

Table 50: MIPI (Status) Properties

API Name	GUI Name	RX	TX	Values
STATUS_EN	Enable Status	✓		0, 1
CLEAR_PIN	Clear Pin Name	✓		Pin name
ERROR_PIN	Error [17:0] Bus Name	✓		Pin name
ULPS_CLK_PIN	UPLS Clock Pin Name	✓		Pin name
ULPS_PIN	UPLS [3:0] Bus Name	✓		Pin name

MIPI D-PHY Property Reference (Titanium)

All of these MIPI D-PHY block properties are applicable to the Titanium family only.

Table 51: MIPI D-PHY General Properties (All Modes)

x is RX or TX.

API Name	GUI Name	Values
DATA_WIDTH	Width of the data bus	8, 16
ENABLE_TURNAROUND	Enable Turn-around Feature in Data Lane 0	0, 1
NAME	Instance Name	Instance Name
NUM_DATA_LANES	Number of data lanes	1, 2, 4
PHY_LANE_0	Physical Lane 0 Map to Logical Lane	clk, data0, data1, data2, data3, unused
PHY_LANE_1	Physical Lane 1 Map to Logical Lane	clk, data0, data1, data2, data3, unused
PHY_LANE_2	Physical Lane 2 Map to Logical Lane	clk, data0, data1, data2, data3, unused
PHY_LANE_3	Physical Lane 3 Map to Logical Lane	clk, data0, data1, data2, data3, unused
PHY_LANE_4	Physical Lane 4 Map to Logical Lane	clk, data0, data1, data2, data3, unused
PHY_LANE_0_PN_SWAP	Swap physical lane P/N on physical lane 0	0, 1
PHY_LANE_1_PN_SWAP	Swap physical lane P/N on physical lane 1	0, 1
PHY_LANE_2_PN_SWAP	Swap physical lane P/N on physical lane 2	0, 1
PHY_LANE_3_PN_SWAP	Swap physical lane P/N on physical lane 3	0, 1
PHY_LANE_4_PN_SWAP	Swap physical lane P/N on physical lane 4	0, 1
RESOURCE	MIPI Resource	MIPI*x*0, MIPI*x*1,MIPI*x*2, MIPI*x*3,

Table 52: MIPI D-PHY RX (Control, Status and Clock) Properties

API Name	GUI Name	Values
BUF_WORD_CLKOUT_HS_NAME	Global Pin Name[12]	Pin name
CFG_CLK_FREQ	Configuration Clock Frequency (MHz)	80 to 120
CFG_CLK_PIN	Configuration Clock Pin Name	Pin name
CFG_CLK_INVERT_EN	Invert Configuration Clock	0, 1
DATA_RATE	Data rate in Mbps	80 to 2500
LP_CLK_PIN	LP Clock State Pin Name	Pin name
RESET_N_PIN	Active Low Reset Pin Name	Pin name
RST0_N_PIN	Active Low Async FIFO Reset0 Pin Name	Pin name
RX_CLK_ACTIVE_HS_PIN	Receiver Clock Active Pin Name	Pin name
RX_ULPS_CLK_NOT_PIN	Active Low ULP State Clock Lane Pin Name	Pin name
RX_ULPS_ACTIVE_CLK_NOT_PIN	Active Low ULP State Active Pin Name	Pin name
STOPSTATE_CLK_PIN	Stop State Clock Lane Pin Name	Pin name
TX_CLK_ESC_PIN	Escape Mode Transmit Clock Pin Name	Pin name
TX_CLK_ESC_INVERT_EN	Invert Escape Mode Transmit Clock	0, 1
WORD_CLKOUT_HS_CONN_TYPE	HS Receive Byte/Word Clock Pin Connection Type	gclk, rclk
WORD_CLKOUT_HS_PIN	HS Receive Byte/Word Clock Pin Name	Pin name

[12] GUI is available in CLKMUX under Regional Buffers tab.

Table 53: MIPI D-PHY RX (Data Lanes) Properties

n is 0, 1, 2, or 3

API Name	GUI Name	Values
BUF_RX_CLK_ESC_NAME	Global Pin Name(12)	Pin name
DIRECTION_PIN	Transmit/Receive Direction Pin Name	Pin name
ERR_ESC_LAN*n*_PIN	Escape Entry Error Pin Name	Pin name
ERR_CONTROL_LAN*n*_PIN	Control Error Pin Name	Pin name
ERR_SOT_SYNC_HS_LAN*n*_PIN	State-of-Transmission (SOT) Sync Error Pin Name	Pin name
ERR_SOT_HS_LAN*n*_PIN	State-of-Transmission (SOT) Error Pin Name	Pin name
ERR_SYNC_ESC_PIN	LP Data Transmit Synchronization Error Pin Name	Pin name
ERR_CONTENTION_LP0_PIN	LP0 Contention Error Pin Name	Pin name
ERR_CONTENTION_LP1_PIN	LP1 Contention Error Pin Name	Pin name
FORCE_RX_MODE_PIN	Force Receive Mode/Wait For Stop Pin Name	Pin name
RX_CLK_ESC_CONN_TYPE	Escape Mode Receive Clock Pin Connection Type	normal, rclk
RX_CLK_ESC_LAN*n*_PIN	Escape Mode Receive Clock Pin Name	Pin name
RX_ULPS_ESC_LAN*n*_PIN	Escape Mode ULP Pin Name	Pin name
RX_ULPS_ACTIVE_NOT_LAN*n*_PIN	Active Low ULP State Data Lane Pin Name	Pin name
RX_ACTIVE_HS_LAN*n*_PIN	HS Reception Active Pin Name	Pin name
RX_VALID_HS_LAN*n*_PIN	HS Data Receive Valid Pin Name	Pin name
RX_SYNC_HS_LAN*n*_PIN	HS Receiver Sync Observed Pin Name	Pin name
RX_SKEW_CAL_HS_LAN*n*_PIN	HS Receiver Skew Calibration Pin Name	Pin name
RX_DATA_HS_LAN*n*_PIN	HS Receive Data Bus Name	Pin name
RX_LPDT_ESC_PIN	Escape Mode LP Data Pin Name	Pin name
RX_VALID_ESC_PIN	Escape Mode Data Valid Pin Name	Pin name
RX_DATA_ESC_PIN	Escape Mode Data 7:0 Bus Name	Pin name
RX_TRIGGER_ESC_PIN	Escape Mode Receive Trigger 3:0 Bus Name	Pin name
STOPSTATE_LAN*n*_PIN	Data Lane In Stop State Pin Name	Pin name
TURN_REQUEST_PIN	Turnaround Request Pin Name	Pin name
TX_REQUEST_ESC_PIN	Escape Mode Transmit Request Pin Name	Pin name
TX_LPDT_ESC_PIN	Escape LP Data Transmit Mode Pin Name	Pin name
TX_VALID_ESC_PIN	Escape Mode Transmit Data Valid Pin Name	Pin name
TX_READY_ESC_PIN	Escape Mode Transmit Ready Pin Name	Pin name
TX_ULPS_ESC_PIN	Escape ULP Transmit Mode Pin Name	Pin name
TX_ULPS_EXIT_PIN	Transmit ULP Exit Sequence Pin Name	Pin name
TX_TRIGGER_ESC_PIN	Escape Mode Transmit Trigger 3:0 Bus Name	Pin name
TX_DATA_ESC_PIN	Escape Mode Transmit Data 7:0 Bus Name	Pin name

Table 54: MIPI D-PHY TX (Control, Status and Clock) Properties

API Name	GUI Name	Values
BUF_RX_CLK_ESC_NAME	Global Pin Name(12)	Pin name
BUF_WORD_CLKOUT_HS_NAME	Global Pin Name(12)	Pin name
ENABLE_SSC	Enable Spread Spectrum Clock (SSC)	0, 1
PHY_BANDWIDTH	Phy Bandwidth in Mbps	80 to 2500, Multiple of 10
PLL_SSC_AMP	SSC Amplitude for MIPI Internal PLL (PPM)	2500 - 4999
PLL_SSC_AMP_INIT	SSC Initial Amplitude for MIPI Internal PLL (PPM)	2501 - 5000
PLL_SSC_PERIOD	SSC Frequency for MIPI Internal PLL (KHz)	30 - 33
PLL_SSC_EN_PIN	PLL SSC Enable Pin Name	Pin name
PLL_UNLOCK_PIN	PLL Unlock State Pin Name	Pin name
REF_CLK_FREQUENCY	Reference Clock Frequency	12.0, 19.2, 25.0, 26.0, 27.0, 38.4, 52.0
REF_CLK_SELECT	Reference Clock Source Type	gpio , pll , core
RESET_N_PIN	Active Low Reset Pin Name	Pin name
RX_CLK_ESC_CONN_TYPE	Escape Mode Receive Clock Connection Type	normal, rclk
RX_CLK_ESC_PIN	Escape Mode Receive Clock Pin Name	Pin name
STOPSTATE_CLK_PIN	Stop State Clock Lane Pin Name	Pin name
TX_CLK_ESC_PIN	Escape Mode Transmit Clock Pin Name	Pin name
TX_CLK_ESC_INVERT_EN	Invert Escape Mode Transmit Clock	0, 1
TX_REQUEST_HS_PIN	HS Clock Request Pin Name	Pin name
TX_ULPS_CLK_PIN	ULP State Clock Lane Pin Name	Pin name
TX_ULPS_EXIT_PIN	ULP Exit Pin Name	Pin name
TX_ULPS_ACTIVE_CLK_NOT_PIN	Active Low ULP State Active Pin Name	Pin name
WORD_CLKOUT_HS_PIN	HS Transmit Byte/Word Clock Pin Name	Pin name
WORD_CLKOUT_HS_CONN_TYPE	HS Transmit Byte/Word Clock Connection Type	gclk, rclk

Table 55: MIPI D-PHY TX (Data Lanes) Properties

n is 0, 1, 2, or 3

API Name	GUI Name	Values
DIRECTION_PIN	Transmit/Receive Direction Pin Name	Pin name
ERR_ESC_PIN	Escape Entry Error Pin Name	Pin name
ERR_CONTROL_PIN	Control Error Pin Name	Pin name
ERR_CONTENTION_LP0_PIN	LP0 Contention Error Pin Name	Pin name
ERR_CONTENTION_LP1_PIN	LP1 Contention Error Pin Name	Pin name
ERR_SYNC_ESC_PIN	LP Data Transmit Synchronization Error Pin Name	Pin name
FORCE_RX_MODE_PIN	Force Receive Mode/Wait For Stop Pin Name	Pin name
RX_DATA_ESC_PIN	Escape Mode Receive Data [7:0] Bus Name	Pin name
RX_LPDT_ESC_PIN	Escape LP Data Receive Mode Pin Name	Pin name
RX_TRIGGER_ESC_PIN	Escape Mode Receive Trigger [3:0] Bus Name	Pin name
RX_ULPS_ESC_PIN	Escape ULP Receive Mode Pin Name	Pin name
RX_VALID_ESC_PIN	Escape Mode Receive Data Valid Pin Name	Pin name
STOPSTATE_LAN*n*_PIN	Stop State Data Lane Pin Name	Pin name
TURN_REQUEST_PIN	Turnaround Request Pin Name	Pin name
TX_REQUEST_ESC_LAN*n*_PIN	Escape Mode Transmit Request Pin Name	Pin name
TX_REQUEST_HS_LAN*n*_PIN	HS Transmit Request and Data Valid Pin Name	Pin name
TX_SKEW_CAL_HS_LAN*n*_PIN	HS Skew Calibration Pin Name	Pin name
TX_READY_HS_LAN*n*_PIN	HS Transmit Ready Pin Name	Pin name
TX_ULPS_ESC_LAN*n*_PIN	ULP Escape Mode State Pin Name	Pin name
TX_ULPS_EXIT_LAN*n*_PIN	ULP Exit Sequence Pin Name	Pin name
TX_ULPS_ACTIVE_NOT_LAN*n*_PIN	Active Low ULP State Data Lane Pin Name	Pin name
TX_DATA_HS_LAN*n*_PIN	HS Transmit Data Bus Name	Pin name
TX_WORD_VALID_HS_LAN*n*_PIN	HS High Byte Valid Pin Name	Pin name
TX_LPDT_ESC_PIN	Escape Mode LP Data Pin Name	Pin name
TX_VALID_ESC_PIN	Escape Mode Data Valid Pin Name	Pin name
TX_READY_ESC_PIN	Escape Mode Ready Pin Name	Pin name
TX_TRIGGER_ESC_PIN	Escape Mode Transmit Trigger [3:0] Bus Name	Pin name
TX_DATA_ESC_PIN	Escape Mode Data [7:0] Bus Name	Pin name

MIPI Lane Property Reference (Titanium)

All of these MIPI D-PHY block properties are applicable to the Titanium family only.

Table 56: MIPI Lane RX Properties

API Name	GUI Name	RX	TX	Values
CLKOUT_PIN	Byte Clock (core) Pin Name	✓		Pin name
CONN_TYPE	Connection Type	✓		GCLK, RCLK
DELAY	Static Mode Output/Input Delay	✓	✓	0 - 63
DELAY_MODE	Delay Mode	✓		STATIC, DYNAMIC
DLY_ENA_PIN	Dynamic Delay Enable Pin Name	✓		Pin name
DLY_INC_PIN	Dynamic Delay Control Pin Name	✓		Pin name
DLY_RST_PIN	Delay Reset Pin Name	✓		Pin name
FASTCLK_PIN	Serial Clock Pin Name		✓	Pin name
FIFO	Enable Clock Crossing FIFO	✓		0, 1
FIFO_EMPTY_PIN	FIFO Empty Pin Name	✓		Pin name
FIFO_RD_PIN	Enable FIFO Read Pin Name	✓		Pin name
GBUF	Global Pin Name[13]	✓		Pin name
HS_ENA_PIN	High-Speed Differential Enable Pin Name	✓		Pin name
HS_IN_PIN	High-Speed Input Pin Name	✓		Pin name
HS_OE_PIN	High-Speed Output Enable Pin Name		✓	Pin name
HS_OUT_PIN	High-Speed Output Pin Name		✓	Pin name
HS_TERM_PIN	High-Speed Termination Enable Pin Name	✓		Pin name
LP_N_IN_PIN	Low-Power (N) Input Pin Name	✓	✓	Pin name
LP_N_OE_PIN	Low-Power (N) Output Enable Pin Name		✓	Pin name
LP_N_OUT_PIN	Low-Power (N) Output Pin Name	✓	✓	Pin name
LP_P_IN_PIN	Low-Power (P) Input Pin Name	✓	✓	Pin name
LP_P_OE_PIN	Low-Power (P) Output Enable Pin Name		✓	Pin name
LP_P_OUT_PIN	Low-Power (P) Output Pin Name	✓	✓	Pin name
MODE	Mode	✓	✓	DATA_LANE, CLOCK_LANE
NAME	Instance Name	✓	✓	Instance name
RESOURCE	MIPI Resource	✓	✓	Pin name
REVERSIBLE	Enable LP Reverse Communication	✓	✓	0, 1
RST_PIN	Reset Pin Name	✓	✓	Pin name
SLOWCLK_PIN	Parallel Clock Pin Name		✓	Pin name

[13] GUI is available in CLKMUX under Regional Buffers tab.

OSC Property Reference

The OSC block has the following properties:

Table 57: OSC Properties

API Name	GUI Name	Trion	Titanium	Values
CLKOUT_PIN	Clock Pin Name	✓	✓	Pin name
ENA_PIN[14]	Clock Pin Name		✓	Pin name
FREQ	Frequency		✓	10, 20, 40, or 80
NAME	Instance Name	✓	✓	Pin name
RESOURCE	Oscillator Resource	✓	✓	OSC_0

[14] Not supported in Ti60ES FPGAs.

PLL Property Reference

Trion FPGAs have two types of PLLs, a simple one and a more advanced one. For the Python API, these PLLs are referenced as V1 (simple PLL) and V2 (advanced PLL).

In the Python API, the Titanium PLL is referenced as V3.

Table 58: PLL Properties

Trion FPGAs: *n* is 0, 1, or 2

Titanium FPGAs: *n* is 0, 1, 2, 3, or 4

API Name	GUI Name	V1	V2	V3	Values
CLKOUT*n*_EN	Enable Output Clock *n*	✓			0, 1
CLKOUT*n*_DIV	Output Clock *n* Divider	✓			2, 4, 8, 16, 32, 64, 128, 256
			✓		1 - 256
				✓	1 - 128
CLKOUT*n*_FREQ	Output Clock *n* Frequency	✓	✓	✓	Calculated value in MHz
CLKOUT*n*_PIN	Output Clock *n* Pin Name	✓			Pin name
CLKOUT*n*_PHASE	Phase Shift (Degree)		✓		0, 45, 90, 135, 180, 270
				✓	Calculated phase shift
CLKOUT*n*_DYNPHASE_EN	Output Clock *n* Enable Dynamic Phase			✓	0, 1
CLKOUT*n*_PHASE_SETTING	Output Clock *n* Phase Shift Setting			✓	0 - 7
CLKOUT*n*_CONN_TYPE	Output Clock *n* Connection Type (Note: For Ti90, Ti120, and Ti180 output clocks 3 and 4 only)			✓	GCLK, RCLK
CLKOUT*n*_CLKMUX_BUF_PIN	Output Clock *n* Buffered Pin Name (Note: For Ti90, Ti120, and Ti180 output clocks 3 and 4 only)			✓	Pin name
CLKOUT*n*_INVERT_EN	Output Clock Inversion - Output Clock *n*			✓	0, 1
CORE_CLK_PIN CORE_CLK1_PIN	Clock Source - Dynamic Clock - Core Clock [0 |1] Name		✓	✓	Clock name
DYN_CLK_SEL_PIN[15]	Clock Source - Dynamic Clock - Clock Selector Name		✓	✓	Clock name
EXT_CLK	Clock Source - External Clock		✓		EXT_CLK0, EXT_CLK1
				✓	EXT_CLK0, EXT_CLK1, EXT_CLK2
FEEDBACK_CLK	Use as feedback		✓		CLK0, CLK1, CLK2
				✓	CLK0, CLK1, CLK2, CLK3, CLK4

[15] If the reference clock source is CORE or DYNAMIC, set this name.

API Name	GUI Name	V1	V2	V3	Values
FEEDBACK_MODE	Feedback Mode		✓		INTERNAL, LOCAL, CORE
				✓	CORE, LOCAL, EXTERNAL
LOCKED_PIN	Locked Pin Name	✓	✓		Pin name
M	Multiplier (M)	✓	✓		1 - 255
N	Pre Divider (N)	✓	✓		1 - 15
NAME	Instance Name	✓			Instance name
O	Post Divider (O)	✓	✓		1, 2, 4, 8
PHASE_SHIFT_ENA_PIN	Phase Shift Enable Pin Name			✓	Pin name
PHASE_SHIFT_SEL_PIN	Phase Shift Select Pin Name			✓	Pin name
PHASE_SHIFT_PIN	Phase Shift Pin Name			✓	Pin name
PLL_FREQ	PLL Frequency	✓			Calculated value in MHz
REFCLK_FREQ	Reference Clock Frequency (MHz)	✓			10.0 - 50.0
			✓		10 - 330[16]
				✓	16 - 800[17]
REFCLK_SOURCE	Clock Source		✓	✓	EXTERNAL, CORE, DYNAMIC
RSTN_PIN	Reset Pin Name	✓			Pin name
RESOURCE	PLL Resource	✓			Resource name
VCO_FREQ	VCO Frequency	✓			Calculated value in MHz

Table 59: Deprecated PLL Properties

API Name	Deprecated In	Replacement
OUTPUT_CLOCKS_INVERTED	2023.1	CLKOUTn_INVERT_EN

[16] The allowed frequency range depends on the clock source you choose. Refer to the data sheet for the Trion FPGA for the PLL timing characteristics.

[17] The allowed frequency range depends on the clock source you choose. Refer to the data sheet for the Titanium FPGA for the PLL timing characteristics.

PLL SSC Property Reference (Titanium)

All of these PLL SSC block properties are applicable to the Titanium family only.

Table 60: PLL SSC Properties

API Name	GUI Name	Values
CLKOUT_FREQUENCY	Clockout Frequency (MHz)	5.0 - 312.5
ENABLE_SSC	Enable Spread Spectrum Clock (SSC)	0, 1
PLL_CLKOUT_CONN_TYPE	PLL Clock Out Connection Type	gclk, rclk
PLL_CLKOUT_PIN	PLL Clock Out Pin Name	Pin name
PLL_SSC_AMP	SSC Amplitude (PPM)	2500 - 4999
PLL_SSC_AMP_INIT	SSC Initial Amplitude (PPM)	2501 - 5000
PLL_SSC_EN_PIN	PLL SSC Enable Pin Name	Pin name
PLL_SSC_PERIOD	SSC Frequency (KHz)	30 - 33
PLL_UNLOCK_PIN	PLL Unlock State Pin Name	Pin name
REF_CLK_FREQUENCY	Reference Clock Frequency	12.0, 19.2, 25.0, 26.0, 27.0, 38.4, 52.0
REF_CLK_SELECT	Reference Clock Source Type	gpio , pll , core
RESET_N_PIN	Active Low Reset Pin Name	Pin name
RESOURCE	PLL SSC Resource	MIPI_TX0, MIPI_TX1, MIPI_TX2, MIPI_TX3

Exceptions

Table 61: Interface Designer Exceptions

Exception	Description
PTBlkCreateException	Exception when creating a block.
PTBlkDeleteException	Exception when deleting a block.
PTBlkEditException	Exception when editing a block property.
PTBlkReadException	Exception when querying for a block property.
PTDsgCheckException	Exception when running the design check.
PTDsgCreateException	Exception when creating a design.
PTDsgExportException	Exception when exporting design.
PTDsgGenConstException	Exception when generating design constraint files.
PTDsgGenReportException	Exception when generating the design report file.
PTDsgImportException	Exception when importing the design.
PTDsgLoadException	Exception when loading a design from a file.
PTDsgSaveException	Exception when saving a design.
PTIllegalValueException	Exception when setting a value that is illegal.
PTNameUsedException	Name collision exception, the name is used already.
PTPropReadOnlyException	Exception when setting value to a read-only property.
PTRefClockException	Reference clock related exception.
PTResInvalidException	Invalid resource exception.
PTResUsedException	Resource collision exception, the resource is used already.

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.

Revision History

Table 62: Revision History

Date	Version	Description
December 2023	6.2	Added result_len argument to auto_calc_pll_clock(). (DOC-1537) Added reset_device_settings() and clear design API functions. (DOC-1558)
October 2023	6.1	Added Pin Swizzling property for DDR. (DOC-1470)
June 2023	6.0	Removed GPIO's OE_CLK_PIN_INV and OE_CLK_PIN properties. (DOC-1248) Added DBI_READ_EN and DBI_WRITE_EN properties for Titanium DDR block. (DOC-1276) Added Output Clock Inversion property which allows the inversion of output clock individually. (DOC-941) Updated SPI Flash block properties and added support for Trion FPGAs in QFP100F3 packages. (DOC-1297) Added PLL SSC block and updated MIPI DPHY PLL SSC properties. (DOC-1297) Added note about deprecated properties.
April 2023	5.1	Updated generate() function description. (DOC-1143) Updated Titanium DDR block AXI Width property. (DOC-1209) Updated auto_calc_pll_clock() function for PLL V3. (DOC-1193)
December 2022	5.0	Added support for Trion LVDS and MIPI, and added support for Titanium DDR, CLKMUX and external flash control. Added clock multiplexer specific API commands.
August 2022	4.0	Updated for Efinity® v2022.1 Removed support for GCTRL in Titanium and removed support for PLL_EXTFB in Trion. (DOC-849) Added more get and set functions. Added remote update, SEU, HyperRAM, SPI flash, MIPI DPHY property references. Updated LVDS, PLL property references.
December 2021	3.1	Added API functions and property settings for I/O banks. Updated PLL properties for Titanium Ti90, Ti120, and Ti180 FPGAs. Corrected the API functions for obtaining version information. The top-level class is APIVersion. (DOC-521)
June 2021	3.0	Added support for Titanium family. Added IP Manager functions. (DOC-431) Added table of exceptions. Added table of API packages. Renamed document as Efinity Python API.

Date	Version	Description
December 2020	2.0	Updated for Efinity® v2020.2 Added functions for Simple PLL, Advanced PLL, and Oscillator. Added PLL properties table. Updated the block types supported in v2020.2 Clarified usage for create_block() and set_property(). (DOC-279) Added IN_REG, OUT_REG, and OE_REG GPIO properties. (DOC-279) Added the create_vref_gpio() function.
June 2020	1.0	Initial release.