Verification Continuum™ Verdi®
Python-Based NPI Netlist Model

Version V-2023.12-SP1, March 2024

SYNOPSYS®

Copyright and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Feedback

Contents

Preface

The Python Based NPI Netlist Model User Guide provides convenient APIs to access netlist.

Customer Support

For any online access to the self-help resources, you can refer to the documentation and searchable knowledge base available in SolvNetPlus.

To obtain support for your Verdi product, choose one of the following:

- Open a case through SolvNetPlus.

  Go to https://solvnetplus.synopsys.com/s/contactsupportand provide the requested information, including:

  - Product L1 as Verdi
  - Case Type

  Fill in the remaining fields according to your environment and issue.

- Send an e-mail message to verdi_support@synopsys.com.

Include product name (L1), sub-product name/technology (L2), and product version in your e-mail, so it can be routed correctly.

Your e-mail will be acknowledged by automatic reply and assigned a Case number along with Case reference ID in the subject (ref:_...:ref).

For any further communication on this Case via e-mail, send e-mail to verdi_support@synopsys.com and ensure to have the same Case ref ID in the subject header or else it will open duplicate cases.

- You can call for support at:

https://www.synopsys.com/support/global-support-centers.html

Note:

In general, we need to be able to reproduce the problem in order to fix it, so a simple model demonstrating the error is the most effective way for us to identify the bug. If that is not possible, then provide a detailed explanation of the problem along with complete error and corresponding code, if any/permissible.

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1

Introduction to Python Based NPI

Python-Based NPI APIs support six models. They are as follows:

- Netlist
- Waveform
- Text
- Coverage
- Language
- Waveform Writer

  Each model have their own APIs to let you be able to traverse data objects and obtain objects' properties like the existing C-Based or Tcl-Based NPI APIs.

  In this guide, the environment setting for using **Python-Based NPI APIs for Netlist** is demonstrated.

Packages and Modules

This section describes the following topics:

Packages

The Python-based NPI package name is "pynpi", and it is placed at `$VERDI_HOME/share/NPI/python`.

Modules

There are seven modules inside the "pynpi" package: npisys, lang, netlist, text, cov waveform and waveformw. The first module, npisys, is the system model for initialization, loading design and exit. The other modules represent language model, netlist model, text model, coverage model, wave model and waveform writer model respectively

Module Functions and Class Objects

This section describes the following topics:

L0 Module Functions

Every module provides some L0 (level 0) functions to let you get the class objects. These functions return a class object or a list of class objects, and they follow the specification of the existing L0 APIs provided in C or Tcl.

L1 Module Functions

Similar to L0 module functions, every module also provides some L1 (level 1) functions to let you get advanced information based on the results obtained by L0 module functions. These functions follows the specification of the existing L1 APIs provided in C or Tcl.

Class Objects

The class object is similar to the so-called handle in NPI C APIs. The most difference is that some basic L0 APIs in C and Tcl will become class method function. These L0 APIs are usually to get integer value, string value, 1-to-1 method to get a handle, and 1-to-many method to get handle iterator.

User Interface and Use Flow

This chapter describes the user interface and use flow for Python-Based NPI APIs.

Environment and Library Setting:

The python library setting flow of using Python-Based NPI APIs contains four parts:

1. Check your Python's version:

   Python-Based NPI APIs need the Python version greater than 3.6.0.

2. Environment setting for "VERDI_HOME" is required for Python-based NPI. Ensure that you set it up before running program.

3. Add python library path into your python code before loading Python-Based NPI by the following commands:

```
rel_lib_path = os.environ['VERDI_HOME'] + '/share/NPI/
python'
```

```
sys.path.append(os.path.abspath(rel_lib_path))
```

4. Import module “npisys” for using the function of NPI initialization and exit from pynpi package.

   `from` **pynpi** `import` **npisys**

5. Import the module you need from pynpi package. For example, if you want to use netlist model, you can import the module as follows:

   `from` **pynpi** `import`**netlist**

6. Note that initialization function `npisys.init()` must be called before writing your code by using any other modules. Besides,`npisys.end()` should be called after finishing your code. Following is a simple example to demonstrate how to use netlist model by Python-Based NPI APIs.

Python program to use NPI netlist model: (`demo.py`)

```
#!/global/freeware/Linux/2.X/python-3.6.0/bin/python
import sys, os
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import npisys
from pynpi import netlist
# Initialize NPI
if not npisys.init(sys.argv):
 print("Error: Fail to initialize NPI")
 assert 0
# Beginning of your code here --------------------
#
# Example code can be found in later chapters
#
# End of your code -------------------------------
# End NPI
npisys.end()
```

C shell script to setup environment and execute Python program on 64-bit machine: (`run_demo`)

```
#!/bin/csh -f
# Setup your $VERDI_HOME here
setenv VERDI_HOME [YOUR_VERDI_HOME_PATH]
# run the python program
# - Input arguments depend on your program design
# - If loading design is required, you can pass the options like
./demo.py -sv demo.v
```

To run the example, put the above files in the same directory and execute the C shell script `run_demo`.

./run_demo

2

Module npisys

Overview

Module npisys is for setting Python-based NPI. You must call `npisys.init()` before using any other NPI modules and call `npisys.end()` after using any other NPI modules.

L0 APIs

Following are the public L0 APIs for system module:

npisys.init(*pyArgvList*)

System initialization for Python-Based NPI.

Parameters: pyArgList (str list) – input argument list, for example, sys.argv

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example:

```
>>>npisys.init(sys.argv)
```

npisys.load_design(*pyArgvList*)

Load design for Python-Based NPI.

Parameters: pyArgList (str list) – input argument list. For example, sys.argv

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example:

```
>>>npisys.load_design(sys.argv)
```

Feedback

npisys.end()

Clean NPI-related settings and data.

Parameters: none

Returns: Return 1 if successful. Otherwise, return 0.

Return type: int

Example:

```
>>>npisys.end()
```

3

Python Based NPI Netlist Model

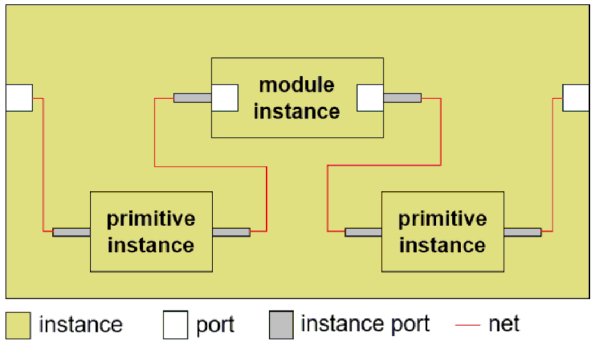
This section describes about the following topics:

Abstract

This module allows you to traverse designs from netlist perspective. The netlist is obtained from the Verdi inference database, which can be regarded as the results of technology-independent non-optimized synthesis results.

In netlist model, a design only consists of four kinds of objects: instance, port, instance port, and net.

Figure 1 Netlist Instance



Feedback

If you import design with symbol libraries, then there will be three other kinds of objects: library, cell, and cell pin. The connectivity relationship is provided along with Classes.

NPI Options

The NPI options and its description is given in the following table:

NPI Options	Description
-npi_nl_rtl_opt <rtl_configurations>	Specify the RTL Configurations. Default: “DetailRTL+DetailMux+GenBlock”.
-npi_nl_rtl_level <level>	Specify the RTL inference level (DetailLevel). Default: 100
-npi_nl_symlib <sym_libs> <sym_libpaths>	Specify the symbol library name and path. Multiple library names and paths can be separated by space or colon(:) character.

Symbol library name and path can also set by using the following environment:

```
>>> setenv NOVAS_LIBS demo_lib
>>> setenv NOVAS_LIBPATHS ./symlibs
```

Set the NPI options in the command as follows:

```
>>> ./example.py -dbdir simv.daidir -npi_nl_rtl_opt
DetailRTL+DetailMux+GenBlock
```

RTL Configurations

The NPI options and its description is given in the following table:

NPI Options	Description
DetailRTL	If set, instances are inferred in detailed view.
DetailMux	If set, unintended latches is extracted as MUX. This works only when DetailRTL is set.
RecogFSM	If set, FSM recognition applies to the inferred objects.
GenBlock	If set, the generate block at the architecture level is expanded. Therefore, the individual instances ise inferred.

Feedback

NPI Options	Description
InferenceLibCell	If set, cells with `celldefine/`endcelldesign, or imported with the -v or -y options, is recognized as modules. Works only when the symbol libraries does not exist.
RecogMem	If set, the memory block at the architecture level is expanded. Therefore, the individual instances are inferred. There is no effect when the block is at the RTL level.

Quick Start

This section describes the following chapters:

Environment and library setting

1. Add python library path using the following commands:

```
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/
python"
sys.path.append(os.path.abspath(rel_lib_path))
```

2. Import `npisys` to use the function of NPI initialization and exit.

   Import `netlist` to use the APIs of Waveform Model.

   `from pynpi import` **npisys**

   `from pynpi import` **netlist**

3. If there exists any error in `LD_LIBRARY_PATH`, add `$VERDI_HOME/share/NPI/lib/linux64` and

```
"$VERDI_HOME/platform/linux64/bin" to LD_LIBRARY_PATH:
os.environ['LD_LIBRARY_PATH'] =
os.environ['VERDI_HOME']+'/share/NPI/lib/
linux64:'+os.environ['VERDI_HOME']+'/platform/linux64/
bin:'+os.environ['LD_LIBRARY_PATH']
```

Example

- example.v

```
module top(clk, win, wout);
 input clk, win;
 output reg wout;
 reg w1;
 reg w2;
```

Feedback

```
 always@(posedge clk) begin
 w1<=win;
 end

 M inst(w1, w2);

 always@(posedge clk) begin
 wout<=w2;

 end
endmodule

module M (a, b);
 input a;
 output b;
 assign b = !a;
endmodule
```

- example.py

```
import sys
import os
rel_lib_path = os.environ["VERDI_HOME"] + "/share/NPI/python"
sys.path.append(os.path.abspath(rel_lib_path))
from pynpi import npisys
from pynpi import netlist

def trace_rec(hdl, indent, cb, visited_set):
 if indent > 10:
 return

 for i in range(indent):
 print(' ', end = '')
 print(f'{hdl.type()}, {hdl.full_name()}')
 if hdl in visited_set:
 return
 visited_set.add(hdl)

 inst_hdl = hdl.scope_inst()
 cell_type = inst_hdl.cell_type()
if cell_type == 'npiNlFlipFlopCell':
 cb[inst_hdl]=hdl
 print(f'--- Hit register: {inst_hdl.full_name()}')
 return
 driver_list = hdl.driver_list()
 indent += 1
 for sub_hdl in driver_list:
 trace_rec(sub_hdl, indent, cb, visited_set)

def cb_func_net_obj(hdl, cb):
 print(f'Trace from: {hdl.info()}')
 visited_set = set()
```

Feedback

```
 driver_list = hdl.driver_list()
 indent = 1
 for sub_hdl in driver_list:
 trace_rec(sub_hdl, indent, cb, visited_set)
 print('')

# main
if __name__ == '__main__':
 if ((not npisys.init(sys.argv)) or (not
npisys.load_design(sys.argv))):
 print('Please load design!')
 orig_stdout = sys.stdout
 f = open('npiNlExample.log', 'w')
 sys.stdout = f

 register_dict = {}
 netlist.hier_tree_trv_register_cb(netlist.ObjectType.DECL_NET,
cb_func_net_obj, register_dict)
 netlist.hier_tree_trv()
 print(f'Total < register, pin > pair found:
{len(register_dict)}')
 print(f' {register_dict}')

 sys.stdout = orig_stdout
 res = npisys.end()
 f.close()
```

Execution

```
>>> example.py -sv example.v -npi_nl_rtl_opt
DetailRTL+DetailMux+GenBlock
```

Result

- npiNlExample.log

```
Trace from: npiNlDeclNet, top.clk
 npiNlPort, top.clk

Trace from: npiNlDeclNet, top.w1
 npiNlInstPort, top.top:Always0#Always0:7:9:Reg.ROH_w1

--- Hit register: top.top:Always0#Always0:7:9:Reg

Trace from: npiNlDeclNet, top.w2
 npiNlInstPort, top.inst.b
  npiNlPort, top.inst.b
   npiNlInstPort, top.inst.M:Always0#SigTap0:21:21:NotRedu.OL_b
    npiNlInstPort, top.inst.M:Always0#SigTap0:21:21:NotRedu.IH_a
     npiNlPort, top.inst.a
```

Feedback

```
      npiNlInstPort, top.inst.a
       npiNlInstPort, top.top:Always0#Always0:7:9:Reg.ROH_w1
--- Hit register: top.top:Always0#Always0:7:9:Reg

Trace from: npiNlDeclNet, top.win
  npiNlPort, top.win

Trace from: npiNlDeclNet, top.wout
  npiNlInstPort, top.top:Always1#Always1:13:15:Reg.ROH_wout
--- Hit register: top.top:Always1#Always1:13:15:Reg

Trace from: npiNlDeclNet, top.inst.a
  npiNlPort, top.inst.a
   npiNlInstPort, top.inst.a
    npiNlInstPort, top.top:Always0#Always0:7:9:Reg.ROH_w1
--- Hit register: top.top:Always0#Always0:7:9:Reg

Trace from: npiNlDeclNet, top.inst.b
  npiNlInstPort, top.inst.M:Always0#SigTap0:21:21:NotRedu.OL_b
   npiNlInstPort, top.inst.M:Always0#SigTap0:21:21:NotRedu.IH_a
    npiNlPort, top.inst.a
     npiNlInstPort, top.inst.a
      npiNlInstPort, top.top:Always0#Always0:7:9:Reg.ROH_w1
--- Hit register: top.top:Always0#Always0:7:9:Reg

Total < register, pin > pair found: 2
  {InstHdl('npiNlInst', 'top.top:Always0#Always0:7:9:Reg'):
    PinHdl('npiNlInstPort',
'top.top:Always0#Always0:7:9:Reg.ROH_w1'),
    InstHdl('npiNlInst', 'top.top:Always1#Always1:13:15:Reg'):
     PinHdl('npiNlInstPort',
'top.top:Always1#Always1:13:15:Reg.ROH_wout')}
```

Demo Case for Documents

For all document of functions, netlist uses the following design and library file as example:

Design

demo.v

```
`timescale 1 ns / 1ns
typedef logic [7:0] ubyte;

interface pram_intf;
 logic VMA;
 logic R_W;
 ubyte addr;
 wire [7:0] data;
 wire w1;
```

Feedback

```
 modport master(output VMA, R_W, addr, inout data);
 modport slave (input VMA, R_W, addr, inout data);
endinterface

module top(clk, win, b, c, a);
 input clk, win, b, c;
 reg reg1, reg2, reg3;
 output a;
 wire signed [3:0] wout, wa, wb, wc;
 wire [0:3] wd;
 wire [1:3] we;
 wire [2:1] wf;
 wire [1:0] a, b;
 wire [6:0] c;
 wire [2:0] d;
 wire e, f, g, h, i, w, x, y, z;
 assign c = {2'b01, 3'b011, a};
 assign d = 3'bx01;
 assign wc = {a ,b};
 assign we = wf;
 assign wd = wc;
 M m1(wout, wa, wb, wa);
 demo_mux m2();
 pram_intf m3();
 demo_func m4();
 level_shifter m5(e, f, g, h);
 ret_cell m6(i, g, h);
 udp_mux2 mux(w, x, y, z);
 assign a = reg1 & b;
 assign a = reg2;
 assign a = reg3 | c;
 always@(posedge clk) begin
 reg1<=win;
 reg2<=win;

reg3<=win;
 end
 demo_ext_ref m7();
endmodule

module M (out, a, b, c);
 output reg [3:0] out, c;
 input [3:0] a, b;
 wire signed [3:0] a, b;
 wire clk;
 always @(posedge clk) begin
 out = a | b;
 end
 assign c = a;
endmodule

module demo_func();
```

Feedback

```
 wire a, b, c, d;
 andd i_andd(a, b, c, d);
endmodule

module demo_mux;
 reg sel, r1, r2, r3;

 wire w1 = (sel) ? r1: r2; // npiNlMuxCell

 always @(sel) begin // npiNlMuxCell
 if (sel) r3 = r1;
 else r3 = r2;
 end
endmodule

primitive udp_mux2 (out, in0, in1, sel);
 output out;
 input in0, in1, sel;

 table
// in0 in1 sel : out
  1 ? 0 : 1 ;
  0 ? 0 : 0 ;
  ? 1 1 : 1 ;
  ? 0 1 : 0 ;
  0 0 x : 0 ;
  1 1 x : 1 ;
  endtable
endprimitive // udp_mux2

module demo_ext_ref ();
 assign top.a = 2'b1;
endmodule
```

Library

demo_lib.lib:

```
library(test_func) {
  cell(andd) {
   area:1;
    pin(A) { direction : input ; }
    pin (B) { direction : input ; }
    pin (C) { direction : input ; }
    pin (D) { direction : output ;
              function : "A*B*C" ;
              x_function : "!B * C" ;
              three_state : "B * !C" ; }
 }
 cell(level_shifter) {
      is_level_shifter : "true" ;
      level_shifter_type : HL;
```

```
     pad_cell : "true";
 pg_pin(VDD) {
    direction : input;
    voltage_name : VDD;
    pg_type : primary_power;
    std_cell_main_rail : true;
    pg_function : "VSS";
    switch_function : "VDD + VSS";
}
pg_pin(VSS) {
  direction : output;
  voltage_name : VSS;
  pg_type : primary_ground;
}
area:1;
pin (A) {
  direction : input;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  input_voltage_range ( 0.7, 0.9);
}
pin (B) {
  direction : output;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  function : "A";
  power_down_function : "!VDD + VSS";
  is_pad : "true" ;
  output_voltage_range (1.1, 1.3)
 }
}
cell(ret_cell) {
  retention_cell : my_ret_cell;
pg_pin(VDD) {
 voltage_name : VDD;
 pg_type : primary_power;
}
pg_pin(VSS) {
  voltage_name : VSS;
  pg_type : primary_ground;
}
pin (RETN) {
  direction : input;
  related_power_pin : VDD;
  related_ground_pin : VSS;
  always_on : true;
  retention_pin (save_restore, "1");
  }
 }
}
```

To compile demo_lib.lib++:

```
>>> mkdir symlib
>>> syn2SymDB -o demo_lib demo_lib.lib
>>> mv demo_lib.lib++ ./symlib
```

To run an test.py with compiled line and design:

```
>>> test.py -sv demo.v -npi_nl_rtl_opt DetailRTL+DetailMux+GenBlock
-npi_nl_symlib demo_lib ./symlib
```

or,

```
>>> setenv NOVAS_LIBS demo_lib
>>> setenv NOVAS_LIB_PATHS ./symlib
>>> test.py -sv demo.v -npi_nl_rtl_opt DetailRTL+DetailMux+GenBlock
```

Properties and Enum

All objects have the following common properties:

1. type: It shows the type of the netlist handle.

   A InstHdl object's type is ObjectType.INST.

   A PinHdl object's type can be [ObjectType.PORT, ObjectType INSTPORT] based on its role, or [ObjectType.PSEUDO_PORT, ObjectType.PSEUDO_INSTPORT] when it's part-selected.

   A NetHdl object's type can be [ObjectType.DECL_NET, ObjecObjectType.CONCAT_NET, ObjectType.SLICE_NET] based on its declaration in design, or ObjectType.PSEUDO_NETwhen it's part-selected.

   A LibHdl object's type is ObjectType.LIBObjectType.LIB.

   A CellHdl object's type is ObjectType.CELL.

   A CellPinHdl object's type is ObjectType.CELLPIN.

2. **name** and **full_name**: the short name and full hierarchical name for the handle.
3. **info**: the infomation string of a InstHdl in style "type, full_name, src_info." And "type, full_name" for handle of other classes.

A property of handle might return an integer or a string. If the property supports both integer and string, the option get_enum is used to control the output format.

For example, property lang_type defined by the following syntax:

```
def lang_type(self, get_enum=False)
```

Feedback

An exception is the property type, it returns an IntEnum of class netlist. ObjectType when get_enum set True.

All the property strings and integer are listed in: Property, Strings, and Integers.

Classes

All classes with properties and relationships between objects are listed below:

ObjectType

class netlist. ObjectType

Object type of netlist handles. Used in return value of type of each handle class and argument in hier_tree_trv().

INST = *1*

PORT = *2*

INSTPORT = *3*

DECL_NET = *4*

CONCAT_NET= *5*

SLICE_NET = *6*

PSEUDO_PORT = *8*

PSEUDO_INSTPORT = *9*

PSEUDO_NET = *10*

LIB = *11*

CELL = *12*

CELLPIN = *13*

FuncType

*class*netlist. FuncType

Function type used in register_cb()..

SIG_TO_SIG = 0

Apply to – sig_to_sig_conn_list() aand NetHdl.to_sig_conn_list().

FAN_IN = *1*

Apply to – NetHdl.fan_in_reg_list()

FAN_OUT = *2*

Apply to – NetHdl.fan_out_reg_list()

ValueFormat

class netlist.ValueFormat.

Value string format used in NetHdl.value().

BIN = *0*

OCT = *1*

HEX = *2*

DEC = *3*

Property, Strings, and Integers

For InstHdl.lang_type(), PinHdl.lang_type(), and NetHdl.lang_type():

string	integer
npiNlSystemVerilog	1
npiNlVHDL	2
npiNlSPICE	3

For InstHdl.cell_type(), and CellHdl.cell_type():

string	integer	comment
npiNlNegCell	1	
npiNlNotCell	2	
npiNlAndCel	3	
npiNlOrCell	4	
npiNlXorCell	5	

Feedback

string	integer	comment
npiNlNandCell	6	
npiNlNorCell	7	
npiNlXNorCell	8	
npiNlAndReduCell	9	
npiNlOrReduCell	10	
npiNlXorReduCell	11	
npiNlNandReduCell	12	
npiNlNorReduCell	13	
npiNlXNorReduCell	14	
npiNlAdderCell	15	
npiNlMulCell	16	
npiNlSubCell	17	
npiNlDivCell	18	
npiNlModCell	19	
npiNlEqCompCell	20	
npiNlNotEqCompCell	21	
npiNlGreateCompCell	22	
npiNlGreateEqCompCell	23	
npiNlLessCompCell	24	
npiNlLessEqCompCell	25	
npiNlShiftLeftCell	26	
npiNlShiftRightCell	27	
npiNlSlaCell	28	
npiNlSraCell	29	
npiNlRolCell	30	

Feedback

string	integer	comment
npiNlRorCell	31	
npiNlRemCell	32	
npiNlAbsCell	33	
npiNlExpCell	34	
npiNlLogAndCell	35	
npiNlLogOrCell	36	
npiNlBufCell	37	
npiNlNonSynCell	38	
npiNlCounterCell	39	
npiNlComboCell	40	
npiNlMacroCell	41	
npiNlTriBufCell	42	
npiNlTriCell	43	
npiNlFlipFlopComboCell	44	Belongs to register
npiNlFlipFlopCell	45	Belongs to register
npiNlLatchComboCell	46	Belongs to register
npiNlLatchCell	47	Belongs to register
npiNlMosCell	48	
npiNlExternalRamCell	49	Belongs to register
npiNlMuxCell	50	
npiNlAssignCell	51	
npiNlOpCell	52	
npiNlFSMCell	53	Belongs to register
npiNlInitCell	54	
npiNlEQCell	55	

Feedback

string	integer	comment
npiNlInfLatchCell	56	
npiNlTranCell	57	
npiNlGateClkCell	58	
npiNlEncoderCell	59	
npiNlAOICell	60	
npiNlOAICell	61	
npiNlUnsignedCell	62	
npiNlMinusCell	63	
npiNlBusKeeperCell	64	
npiNlBiDirectionCell	65	
npiNlMinimumCell	66	
npiNlMaximumCell	67	
npiNlCondConvCell	68	
npiNlMatchingEqCell	69	
npiNlMatchingNotEqCell	70	
npiNlMatchingGtCell	71	
npiNlMatchingGtEqCell	72	
npiNlMatchingLsCell	73	
npiNlMatchingLsEqCell	74	
npiNlNMosCell	75	
npiNlPMosCell	76	
npiNlDiodeCell	77	
npiNlNPNCell	78	
npiNlPNPCell	79	
npiNlNJFCell	80	

Feedback

string	integer	comment
npiNlPJFCell	81	
npiNlResCell	82	
npiNlCapCell	83	
npiNlIndCell	84	
npiNlVoltageSource	85	
npiNlCurrentSource	86	
npiNlVCVSource	87	
npiNlVCCSource	88	
npiNlCCVSource	89	
npiNlCCCSource	90	
npiNlInstrumentedCell	91	
npiNlClockBlockCell	92	
npiNlModuleCell	999	Module Instance

For InstHdl.inst_type():

string	integer
npiNlHierInst	1
npiNlRTLInst	2
npiNlSymbolLibInst	3
npiNlFSMInst	4
npiNlUDPInst	5

For InstHdl.power_cell_type(), and CellHdl.power_cell_type():

string	integer
npiNlPowerUndefCell	1

string	integer
npiNlPowerISOCell	2
npiNlPowerRETCell	3
npiNlPowerSwitchCell	4
npiNlPowerAlwaysOnCell	5
npiNlPowerSwitchCell	6
npiNlPowerSRSNCell	7
npiNlPowerSPACell	8
npiNlPowerBMuxCell	9
npiNlPowerRPTRCell	10
npiNlPowerPADCell	11

For PinHdl.port_type(), and CellPinHdl.port_type():

string	integer
npiNlDataPort	1
npiNlClockPort	2
npiNlSyncSetPort	3
npiNlSyncResetPort	4
npiNlAsyncSetPort	5
npiNlAsyncResetPort	6
npiNlControlPort	7
npiNlTriEnablePort	8
npiNlComboOutputPort	9
npiNlTriOutputPort	10
npiNlLatchedPort	11
npiNlRegisteredPort	12

Feedback

string	integer
npiNlFSMControlPort	13
npiNlFSMOutputPort	14
npiNlNotifyPort	15
npiNlPrimaryPowerPort	16
npiNlBackupPowerPort	17
npiNlInternlPowerPort	18
npiNlPrimaryGroundPort	19
npiNlBackupGroundPort	20
npiNlInternlGroundPort	21
npiNlIsoEnbPort	22
npiNlIsoDataPort	23
npiNlLvsEnbPort	24
npiNlLvsDataPort	25
npiNlSwtPort	26
npiNlInferLatchPort	27
npiNlPunchPort	28

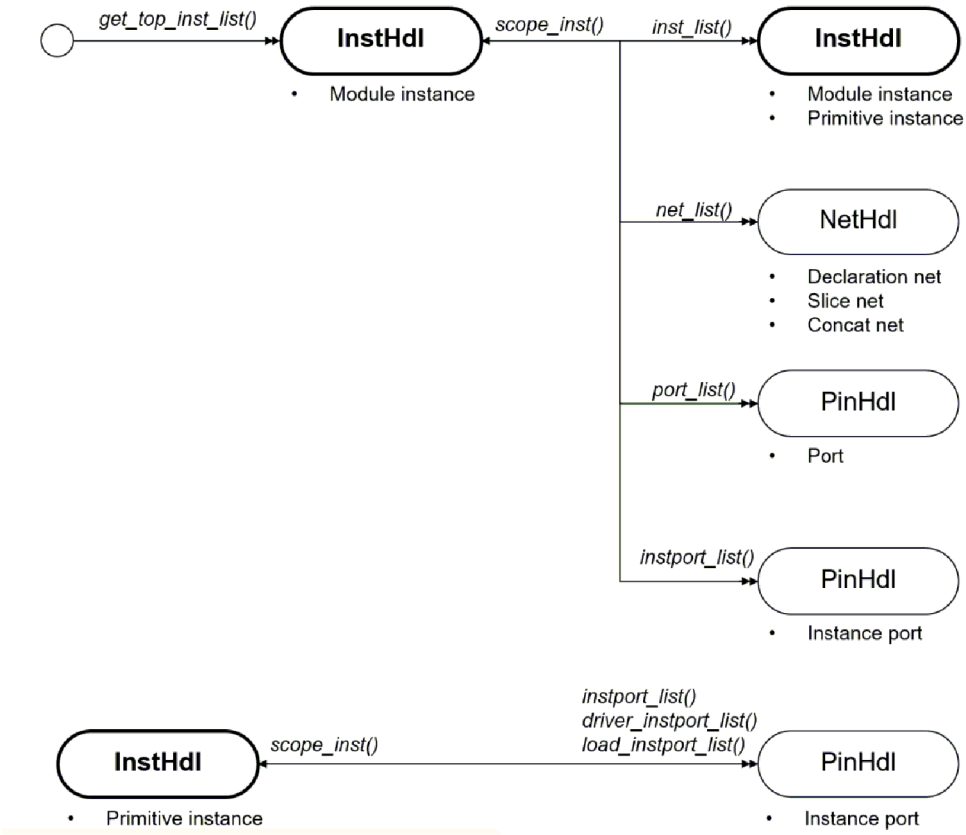
For PinHdl.port_type(), and CellPinHdl.direction()

string	integer
npiNlInput	1
npiNlOutput	2
npiNlInout	3

Feedback

InstHdl

Figure 2 Object diagram



class netlist. InstHdl(*cps_obj*)

InstHdl includes two types: module instance and primitive instance.

- **Module Instance**:

  A module reference in the design that serves as a container for sub instance, port, instance port, and net objects.

- **Primitive Instance**:

  Denotes the minimum design unit. It is used to represent low-level logic gates and switches with specific cell functions. For example, npiNlAndCell, npiNlFSMCell, and npiNlBufCell.

  You can query a InstHdl by public functions, either by *get_inst()*, or traverse by *get_top_inst_list()*get_top_inst_list(), and *InstHdl.inst_list()*.

1-1 Method

scope_inst()	Get upper scope instance handle.

1-m Method

inst_list()	Get internal instance handles defined in/inferences from the instance in list.
net_list()	Get net handles defined in/inferences from the instance in list.
port_list()	Get port handles defined in the instance in list.
instport_list()	Get instport handles defined in the instance in list.
driver_instport_list()	Get input/inout instport handles defined in the primitive instance in list.
load_instport_list()	Get output/inout instport handles defined in the primitive instance in list.

General Properties

type(get_enum=False)	The type string/enum.
name()	The short name string.
full_name()	The full hierarchical name string.
def_name()	The definition name (Module name) of the instance
lang_type(get_enum=False)	The language type string/int.
cell_type(get_enum=False)	The cell type string/int.
inst_type(get_enum=False)	The instance type string/int.
src_info()	The multiple source information in style: { File1 : BeginLineNo1 : EndLineNo1} {File2 : BeginLineNo2 : EndLineNo2} …
info()	The infomation string in style: type, full_name, src_info.
file()	File path where the instance exists.

begin_line_no()	The begin line number in the file where the instance exists.
end_line_no()	The end line number in the file where the instance exists.
is_interface()	Whether the instance is inferred from a SystemVerilog interface.
is_modport()	Whether the instance is inferred from a SystemVerilog modport.
is_intf_connector()	Whether the instance is inferred to connect the modport net and the whole interface port.
is_intf_net()	Whether the instance is inferred from a net of SystemVerilog interface.

Properties of VHDL design

entity_name()	The entity name of the VHDL instance.
arch_name()	The architecture name of VHDL instances.

Properties of library cell

power_cell_type(get_enum=False)	The power cell type string/int inferred from library.
is_memory_cell()	Whether the instance's corresponding cell has memory() definition in library.
is_pad_cell()	Whether the instance's corresponding cell has set the attribute pad_cell true.

scope_inst()

Get upper scope instance handle.

Parameters: None.

Returns: The handle of upper scope.

Return type: InstHdl

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").scope_inst()}')
InstHdl('npiNlInst', 'top')
```

Feedback

inst_list()

Get internal instance handles defined in/inferenced from the instance in list.

Only module instance has sub instances inside.

Parameters: None.

Returns: The list of internal sub instance handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").inst_list()}')
[InstHdl('npiNlInst', 'top.m1.M:Always0#SigTap0:58:58:Assignment'),
InstHdl('npiNlInst', 'top.m1.M:Always1#Always0:55:57:Or'),
InstHdl('npiNlInst', 'top.m1.M:Always2#Always0:55:57:Reg')]
```

net_list()

Get net handles defined in/inferenced from the instance in list.

Note:

Only *module instance* has nets inside.

Parameters: None.

Returns: The list of internal net handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").net_list()}')
[NetHdl('npiNlDeclNet', 'top.m1.GEN0_out[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.a[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.b[3:0]'), NetHdl('npiNlDeclNet',
'top.m1.c[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.clk'), NetHdl('npiNlDeclNet',
'top.m1.out[3:0]')]
```

port_list()

Get port handles defined in the instance in list.

Note:

Only module instance has ports inside.

Parameters: None.

Returns: The list of internal port handles.

Feedback

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").port_list()}')
[PinHdl('npiNlPort', 'top.m1.a[3:0]'), PinHdl('npiNlPort',
'top.m1.b[3:0]'),
PinHdl('npiNlPort', 'top.m1.c[3:0]'), PinHdl('npiNlPort',
'top.m1.out[3:0]')]
```

instport_list()

Get instport handles defined in the instance in list.

Parameters: None.

Returns: The list of instport handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").instport_list()}')
[PinHdl('npiNlInstPort', 'top.m1.a[3:0]'), PinHdl('npiNlInstPort',
'top.m1.b[3:0]'),
PinHdl('npiNlInstPort', 'top.m1.c[3:0]'), PinHdl('npiNlInstPort',
'top.m1.out[3:0]')
```

driver_instport_list()

Get input/inout instport handles defined in the primitive instance in list.

Note:

Only primitive instance has driver instports defined.

Parameters: None.

Returns: The list of driver instport handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
driver_instport_list()}')
[PinHdl('npiNlInstPort',
'top.top:Always7#SigOp7:41:41:Or.IH_c[6:0]'),
PinHdl('npiNlInstPort', 'top.top:Always7#SigOp7:41:41:Or.IH_reg3')]
```

load_instport_list()

Get output/inout instport handles defined in the primitive instance in list.

Note:

Only primitive instance has load instports defined.

Parameters: None.

Returns: The list of load instport handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
load_instport_list()}')
[PinHdl('npiNlInstPort',
'top.top:Always7#SigOp7:41:41:Or.OH_a[1:0]')]
```

type(*get_enum=False*)

The type string/enum.

Parameters: get_enum (bool) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type: str/ ObjectType

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").type()}')
npiNlInst
>>> print(f'{netlist.get_inst("top.m1").type(True)}')
1
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").name()}')
m1
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns: The string of handle hierarchical name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").full_name()}')
top.m1
```

def_name()

The definition name (also known as, Module name) of the instance.

Parameters: None.

Returns: The definition name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").def_name()}')
M
```

lang_type(*get_enum=False*)

The language type string/int.For example, npiNlSystemVerilog, npiNlVHDL, npiNlSPICE.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of language type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").lang_type()}')
npiNlSystemVerilog

>>> print(f'{netlist.get_inst("top.m1").lang_type(True)}')
1
```

cell_type(*get_enum=False*)

The cell type string/int.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of cell type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").cell_type()}')
npiNlModuleCell

>>> print(f'{netlist.get_inst("top.m1").cell_type(True)}')
999

>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
cell_type()}')
npiNlOrCell

>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
cell_type(True)}')
4
```

inst_type(get_enum=False)

The instance type string/int. For example, npiNlHierInst, npiNlRTLInst, npiNlSymbolLibInst, npiNlFSMInst, npiNlUDPInst.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of instance type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").inst_type()}')
npiNlHierInst

>>> print(f'{netlist.get_inst("top.m1").inst_type(True)}')
1

>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
inst_type()}')
npiNlRTLInst

>>> print(f'{netlist.get_inst("top.top:Always7#SigOp7:41:41:Or").
inst_type(True)}')
2

>>> print(f'{netlist.get_inst("top.m4.i_andd").inst_type()}')
npiNlSymbolLibInst
>>> print(f'{netlist.get_inst("top.m4.i_andd").inst_type(True)}')
3
```

power_cell_type(*get_enum=False*)

The power cell type string/int inferred from library.

Note:

Only the instance objects inferred from library has meaningful power cell type.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of power cell type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_inst("top.m5").power_cell_type()}')
npiNlPowerLVSCell

>>> print(f'{netlist.get_inst("top.m5").power_cell_type(True)}')
4
```

src_info()

The multiple source information in style: { File1 : BeginLineNo1 : EndLineNo1} {File2 : BeginLineNo2 : EndLineNo2} …

Parameters: None.

Returns: The source infomation of the instance.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").src_info()}')
{demo.v : 33 : 33}
```

info()

The information string in style: type, full_name, src_info.

Parameters: None.

Returns: The string of handle’s information.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").info()}')
npiNlInst, top.m1, {m.v : 14 : 14}
```

file()

File path where the instance exists.

Parameters: None.

Returns: The file path of the instance.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").file()}')
m.v
```

begin_line_no()

The begin line number in the file where the instance exists.

Parameters: None.

Returns: The begin line number of the instance.

Return type: int

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").begin_line_no()}')
33
```

end_line_no()

The end line number in the file where the instance exists.

Parameters: None.

Returns: The end line number of the instance.

Return type: int

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").end_line_no()}')
33
```

is_interface()

Whether the instance is inferred from a SystemVerilog interface.

Parameters: None.

Returns: True if it is interface.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").is_insterface()}')
False
```

```
>>> print(f'{netlist.get_inst("top.m3").is_insterface()}')
True
```

is_modport()

Whether the instance is inferred from a SystemVerilog modport.

Parameters: None.

Returns: True if it is an interface modport.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m3.master").is_modport()}')
True
```

is_intf_connector()

Whether the instance is inferred to connect the modport net and the whole interface port.

Parameters: None.

Returns: True if it is an interface connector.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m3.pram_intf_master_connect").
is_intf_connector()}')
True
```

is_intf_net()

Whether the instance is inferred from a net of SystemVerilog interface.

Parameters: None.

Returns: True if it is an interface net.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m3.w1").is_intf_net()}')
True
```

entity_name()

The entity name of the VHDL instance.

Note:

This property is for VHDL instance only.

Parameters: None.

Returns: The entity name of the instance.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").entity_name()}')
None
```

arch_name()

The architecture name of VHDL instances.

Note:

This property is for VHDL instance only.

Parameters: None.

Returns: The arch name of the instance.

Return type: str

Examples:

```
>>> print(f'{netlist.get_inst("top.m1").arch_name()}')
None
```

is_memory_cell()

Whether the instance's corresponding cell has memory() definition in library.

Note:

This property is for library cell instance only.

Parameters: None.

Returns: True if it is a memory cell.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m5").is_memory_cell()}')
False
```

Feedback

is_pad_cell()

Whether the instance's corresponding cell has set the attribute pad_cell true.

Note:

This property is for library cell instance only.

Parameters: None.

Returns: True if it is a pad cell.

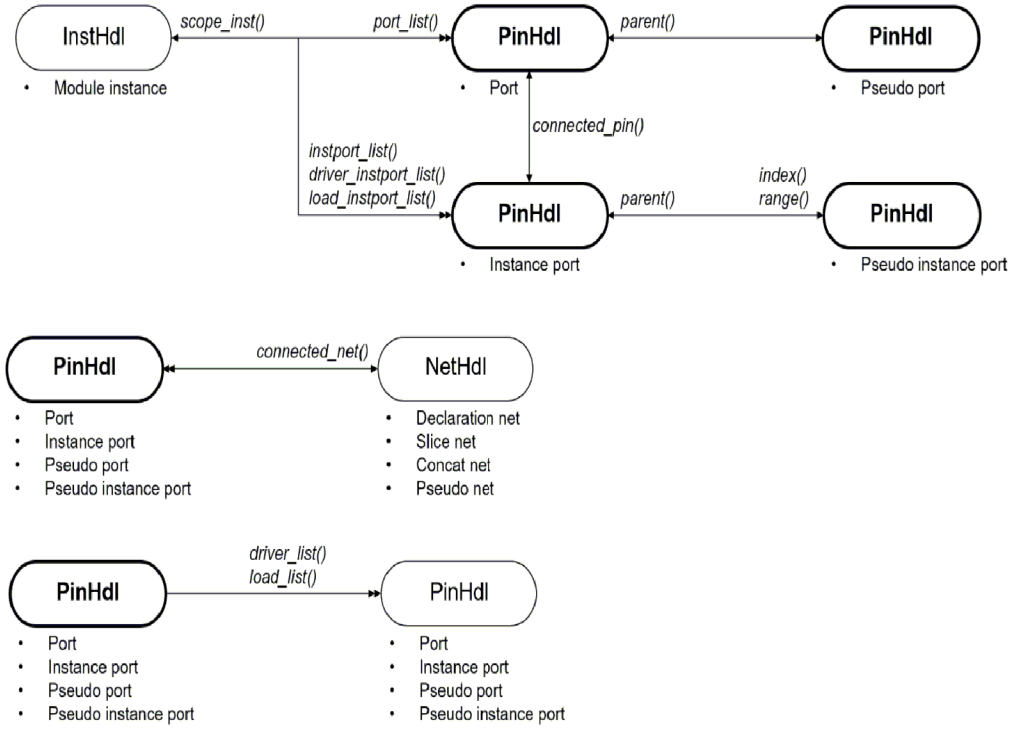
Return type: bool

Examples:

```
>>> print(f'{netlist.get_inst("top.m5").is_pad_cell()}')
True
```

PinHdl

Figure 3 Object diagram



Feedback

*class netlist.*PinHdl(*cps_obj*)

PinHdl includes two types: port and instance port.

Port:

A port object represents the port defined in a module instance. It is used to communicate between instance port and net objects belonging to the module instance.

Instance Port:

An instance port object represents the port instance along with the instance object. It is used to communicate between instance and net objects connected outside the instance.

You can query a PinHdl by public functions, either by get_port(), get_instport(), or traverse by InstHdl.port_list(), InstHdl.instport_list().

1-1 Method

scope_inst()	Get upper scope instance handle.
parent()	Get parent port/instport handle from which the pseudo port/instport is created.
index(index)	Get pseudo port/instport of specified index.
range(left, right)	Get pseudo port/instport of specified range index.
connected_pin()	Get the connected port/instport handle.
connected_net()	Get the connected net handle.

1-m Method

driver_list()	Get its driver port/instport handles in list.
load_list()	Get its load port/instport handles in list.

General Properties

type(get_enum=False)	The type string/enum.
name()	The short name string.
full_name()	The full hierarchical name string.
info()	The information string in style: type, full_name.

Feedback

lang_type(get_enum=False)	The language type string/int.
port_type(get_enum=False)	The functional type string/int.
direction(get_enum=False)	The direction string/int.
left()	The left index of port/instport.
right()	The right index of port/instport.

Property for port

is_intf_type_port()	Whether the port object is an interface port.

Properties of instport

port_state(get_enum=False)	The port active state string/int.
port_order()	The order of the instport in its instance.
cond_annot()	The conditional value of the instport of Mux/Latch cell.
table_lookup(input_value)	Lookup on the output according to the input value specified in the UDP table.
func()	The function attribute of instport defined in the library.
x_func()	The x_function attribute of instport defined in the library.
three_state_func()	The three_state attribute of instport defined in the library.
is_pg_pin()	Whether the instport is a pg_pin.
is_std_cell_main_rail_pin()	Whether the std_cell_main_rail attribute is set.
voltage_name()	The string of its specified voltage_name.
pg_func()	The string of a pg_pin’s specified pg_function.
switch_func()	The string of a pg_pin’s specified switch_function.
is_pad()	Whether the is_pad attribute is set.

related_power()	The string of its specified related_power_pin.
related_ground()	The string of its specified related_ground_pin.
power_down_func()	The string of its specified power_down_function.
input_voltage_rng_max()	The string of max_value specified in input_voltage_range(min_value, max_value) of an input pin.
input_voltage_rng_min()	The string of min_value specified in input_voltage_range(min_value, max_value) of an input pin.
output_voltage_rng_max()	The string of max_value specified in output_voltage_range(min_value, max_value) of an output pin.
output_voltage_rng_min()	The string of min_value specified in output_voltage_range(min_value, max_value) of an output pin.
ret_pin_class_name()	The pin_class of the retention_pin(pin_class, disable_value).
is_always_on()	Whether the always_on attribute is set.

scope_inst()

Get upper scope instance handle.

Note:

Only for PinHdl of type() = npiNlPort/npiNlInstPort.

Parameters: None.

Returns: The handle of upper scope.

Return type: InstHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a[3:0]").scope_inst()}')
InstHdl('npiNlInst', 'top.m1')

>>> print(f'{netlist.get_instport("top.m1.a[3:0]").scope_inst()}')
InstHdl('npiNlInst', 'top.m1')
```

parent()

Get parent port/instport handle from which the pseudo port/instport is created.

Note:

Only for PinHdl of type() = npiNlPseudoPort/npiNlPseudoInstPort.

Parameters: None.

Returns: The parent handle.

Return type:PinHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a[1]").parent()}')
PinHdl('npiNlPort', 'top.m1.a[3:0]')

>>> print(f'{netlist.get_instport("top.m1.a[1]").parent()}')
PinHdl('npiNlInstPort', 'top.m1.a[3:0]')
```

index(index)

Get pseudo port/instport of specified index.

Parameters: **index** (int) – Index number to query from the referenced handle.

Returns: pseudo handle based on the index number of the referenced handle

Return type: PinHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a").index(1)}')
PinHdl('npiNlPseudoPort', 'top.m1.a[3:0]#[1]')

>>> print(f'{netlist.get_instport("top.m1.a").index(1)}')
PinHdl('npiNlPseudoInstPort', 'top.m1.a[3:0]#[1]')
```

range(*left, right*)

Get pseudo port/instport of specified range index.

Parameters:

left (int) – The left bound index to query from the referenced handle.

right (int) – The right bound index to query from the referenced handle.

Returns: pseudo handle based on the index range of the referenced handle

Return type: PinHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a").range(2,1)}')
PinHdl('npiNlPseudoPort', 'top.m1.a[3:0]#[2:1]')

>>> print(f'{netlist.get_instport("top.m1.a").range(2,1)}')
PinHdl('npiNlPseudoInstPort', 'top.m1.a[3:0]#[2:1]')
```

connected_pin()

Get the connected port/instport handle.

Parameters: None.

Returns: The connected PinHdl.

Return type: PinHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a[3:0]").connected_pin()}')
PinHdl('npiNlInstPort', 'top.m1.a[3:0]')

>>> print(f'{netlist.get_instport("top.m1.a[3:0]").connected_pin()}')
PinHdl('npiNlPort', 'top.m1.a[3:0]')
```

connected_net()

Get the connected net handle.

Parameters: None.

Returns: The connected NetHdl.

Return type:NetHdl

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a[3:0]").connected_net()}')
NetHdl('npiNlDeclNet', 'top.m1.a[3:0]')

>>> print(f'{netlist.get_instport("top.m1.a[3:0]").connected_net()}')
NetHdl('npiNlDeclNet', 'top.wa[3:0]')
```

driver_list()

Get its driver port/instport handles in list.

Parameters: None.

Returns: The driver list of PinHdl is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a").driver_list()}')
[PinHdl('npiNlInstPort', 'top.m1.a[3:0]')]

>>> print(f'{netlist.get_instport("top.m1.a").driver_list()}')
[PinHdl('npiNlInstPort', 'top.m1.c[3:0]')]
```

Feedback

load_list()

Get its load port/instport handles in list.

Parameters: None.

Returns: The load list of PinHdl is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a").load_list()}')
[PinHdl('npiNlInstPort',
'top.m1.M:Always0#SigTap0:58:58:Assignment.IH_a[3:0]'),
PinHdl('npiNlInstPort',
'top.m1.M:Always1#Always0:55:57:Or.IH_a[3:0]')

>>> print(f'{netlist.get_instport("top.m1.a").load_list()}')
[PinHdl('npiNlPort', 'top.m1.a[3:0]')]
```

type(*get_enum=False*)

The type string/enum.

Parameters: **get_enum** (*bool*) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type: str/ ObjectType

Examples:

```
>>> print(f'{netlist.get_port("top.a").type()}')
npiNlPort

>>> print(f'{netlist.get_port("top.a").type(True)}')
2

>>> print(f'{netlist.get_instport("top.a").type()}')
npiNlInstPort

>>> print(f'{netlist.get_instport("top.a").type(True)}')
3
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_port("top.a").name()}')
a[1:0]

>>> print(f'{netlist.get_instport("top.a").name()}')
a[1:0]
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns: The string of handle hierarchical name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_port("top.a").full_name()}')
top.a[1:0]

>>> print(f'{netlist.get_instport("top.a").full_name()}')
top.a[1:0]
```

info()

The information string in style: type, full_name.

Parameters: None.

Returns: The string of handle's information.

Return type: str

Examples:

```
>>> print(f'{netlist.get_port("top.a").info()}')
npiNlPort, top.a[1:0]

>>> print(f'{netlist.get_instport("top.a").info()}')
npiNlInstPort, top.a[1:0]
```

lang_type(*get_enum=False*)

The language type string/int. For example, npiNISystemVerilog, npiNIVHDL, npiNISPICE.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of language type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_port("top.a").lang_type()}')
npiNlSystemVerilog

>>> print(f'{netlist.get_port("top.a").lang_type(True)}')
1
```

port_type(*get_enum=False*)

The functional type string/int. For example, npiNlDataPort, npiNlClockPort, npiNlControlPort, npiNlTriEnablePort, npiNlRegisteredPort.

Parameters: **get_enum** (bool) – Determine if return in integer. Default: False.

Returns: The string/int of port type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_port("top.a").port_type()}')
npiNlModulePort

>>> print(f'{netlist.get_port("top.a").port_type(True)}')
999

>>> print(f'{netlist.get_instport("top.m1.M:Always0#SigTap0:58:58:
Assignment.IH_a").port_type()}')
npiNlDataPort

>>> print(f'{netlist.get_instport("top.m1.M:Always0#SigTap0:58:58:A
ssignment.IH_a").port_type(True)}')
```

direction(get_enum=False)

The direction string/int. For example, npiNlInput, npiNlOutput, or npiNlInout.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of port direction.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_port("top.a").direction()}')
npiNlOutput

>>> print(f'{netlist.get_port("top.a").direction(True)}')
2
```

size()

The size of port/instport.

Parameters: None.

Returns: The size of handle is returned.

Return type: int

Examples:

```
>>> print(f'{netlist.get_port("top.a").size()}')
2
```

left()

The left index of port/instport.

Parameters: None.

Returns: The left index of handle is returned.

Return type: int

Examples:

```
>>> print(f'{netlist.get_port("top.a").left()}')
1
```

right()

The right index of port/instport.

Parameters: None.

Returns: The right index of handle is returned.

Return type: int

Examples:

```
>>> print(f'{netlist.get_port("top.a").right()}')
0
```

is_intf_type_port()

Weather the port object is an interface port.

Note:

Only for PinHdl of type() = npiNlPort/npiNlPseudoPort.

Parameters: None.

Returns: if the port object is an interface port, return True.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_port("top.a").is_intf_type_port()}')
False

>>> print(f'{netlist.get_port("top.m3.master").is_intf_type_port()}')
True
```

port_state(*get_enum=False*)

The port active state string/int. For example, npiNlHighActive, npiNlLowActive, npiNlRisingActive, and npiNlFallingActive.

Note:

Only for PinHdl of type() = npiNlInstPort/npiNlPseudoInstPort and the inst_type() of its scope InstHdl = npiNlRTLInst/npiNlSymbolLibInst/ npiNlFSMInst.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of port state.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_instport("top.m1.M:Always0#SigTap0:58:58:
Assignment.IH_a").port_state()}')
npiNlHighActive

>>> print(f'{netlist.get_instport("top.m1.M:Always0#SigTap0:58:58:
Assignment.IH_a").port_state(True)}')
1
```

port_order()

The order of the instport in its instance.

Note:

Only for PinHdl of type() = npiNlInstPort/npiNlPseudoInstPort. And the InstHdl.inst_type() of its scope InstHdl = npiNlRTLInst when DetailRTL option set. Or the InstHdl.cell_type() = npiNlComcoCell and InstHdl.inst_type() = npiNlUDPInst of its scope InstHdl.

Parameters: None.

Returns: The order of instport.

Feedback

Return type: int

Examples:

```
>>> print(f'{netlist.get_instport("top.m1.M:Always1#Always0:55:57:
Or.IH_a").port_order()}')
0

>>> print(f'{netlist.get_instport("top.m1.M:Always1#Always0:55:57:
Or.IH_b").port_order()}')
```

cond_annot()

The conditional value of the instport of Mux/Latch cell.

Note:

Only for PinHdl of type() = npiNlInstPort/npiNlPseudoInstPort, and port_type() = npiNlDataPort. And the InstHdl.inst_type() = npiNlRTLInst and the InstHdl.cell_type()= npiNlMuxCell/npiNlInfLatchCell of its scope InstHdl.

Parameters: None.

Returns: The conditional annotation.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m2.demo_mux:
Always0#SigOp0:69:69:Mux.IH_r1").cond_annot()}')
1'b1

>>> print(f'{netlist.get_instport("top.m2.demo_mux:
Always0#SigOp0:69:69:Mux.IH_r2").cond_annot()}')
1'b0
```

table_lookup(*input_value*)

Lookup on the output according to the input value specified in the UDP table.

Note:

Only for PinHdl of type() = npiNlInstPort/npiNlPseudoInstPort, and direction() = npiNlOutput. And the InstHdl.inst_type() = npiNlUDPInst of its scope InstHdl. In addition, the z values passed to UDP inputs shall be treated the same as x values.

Parameters: input_value (str) – The input value [1, 0, x, z] of UDP table.

Returns: The output value string. If the input value is invalid, return None.

Return type: str

Examples:

```
>>> hdl = netlist.get_instport('top.mux.out')
>>> in_data = ['00x', '11x', '1x0', 'x10', 'xxx', '1?0']
>>> for data in in_data:
 print(f'in = {data}, out = {hdl.table_lookup(data)}')
in = 00x, out = 0
in = 11x, out = 1
in = 1x0, out = 1
in = x10, out = x
in = xxx, out = x
in = 1?0, out = None
```

func()

The function attribute of instport defined in the library.

Note:

Only for PinHdl of type() = npiNlInstPort, and port_type() = npiNlComboOutputPort/npiNlTriOutputPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The function string.

Return type: str.

Examples:

```
>>> print(f'{netlist.get_instport("top.m4.i_andd.D").func()}')
(A&B&C)
```

x_func()

The x_function attribute of instport defined in the library.

Note:

Only for PinHdl of type() = npiNlInstPort, and port_type() = npiNlComboOutputPort/npiNlTriOutputPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns:The x_function string.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m4.i_andd.D").x_func()}')
(!(B)&C)
```

Feedback

three_state_func()

The three_state attribute of instport defined in the library.

Note:

Only for PinHdl of type() = npiNlInstPort, and port_type() = npiNlComboOutputPort/npiNlTriOutputPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The three_state string.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m4.i_andd.D").
three_state_func()}')
(B&!(C))
```

is_pg_pin()

Whether the instport is a pg_pin.

Note:

Only for PinHdl of type()= npiNlInstPort, and port_type() = npiNlPrimaryPowerPort/npiNlBackupPowerPort/npiNlInternlPowerPort/ npiNlPrimaryGroundPort/npiNlBackupGroundPort/npiNlInternlGroundPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns:True, if it is pg pin.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.VDD").
is_pg_pin()}')
True
```

is_std_cell_main_rail_pin()

Whether the std_cell_main_rail attribute is set.

Note:

Only for PinHdl of type() = npiNlInstPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: True if the std_cell_main_rail attribute is set.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.VDD").
is_std_cell_main_rail_pin()}')
True
```

voltage_name()

The string of its specified voltage_name.

Note:

Only for PinHdl of type() = npiNlInstPort, and port_type() = npiNlPrimaryPowerPort/npiNlBackupPowerPort/npiNlInternlPowerPort/ npiNlPrimaryGroundPort/npiNlBackupGroundPort/npiNlInternlGroundPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The voltage_name of a cell pg_pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.VDD").voltage_name()}')
VDD
```

pg_func()

The string of a pg_pin's specified pg_function.

Note:

Only for PinHdl of type() = npiNlInstPort, and is_pg_pin() = True. And InstHdl.inst_type()= npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The pg_function of a cell pg_pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.VDD").pg_func()}')
VSS
```

Feedback

switch_func()

The string of a pg_pin's specified switch_function.

Note:

Only for PinHdl of type() = npiNlInstPort, and is_pg_pin() = True. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The switch_function of a cell pg_pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.VDD").switch_func()}')
VDD + VSS
```

is_pad()

Whether the is_pad attribute is set.

Note:

Only for PinHdl of type()= npiNlInstPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: True, if it is pad.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.B").is_pad()}')
True
```

related_power()

The string of its specified related_power_pin.

Note:

Note: Only for PinHdl of type()= npiNlInstPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The related_power_pin of a cell pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.A").related_power()}')
VDD
```

related_ground()

The string of its specified related_ground_pin.

Note:

Only for PinHdl of type() = npiNlInstPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The related_ground_pin of a cell pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.A").related_ground()}')
VSS
```

power_down_func()

The string of its specified power_down_function.

Note:

Note: Only for PinHdl of type() = npiNlInstPort, and direction() = npiNlOutput. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The power_down_function of a cell pin.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.B").power_down_func()}')
!VDD + VSS
```

input_voltage_rng_max()

The string of max_value specified in input_voltage_range(min_value, max_value) of an input pin.

Note:

: Only for PinHdl of type() = npiNlInstPort, and direction() = npiNlInput. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None

Returns: The input voltage range max value.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.A").
input_voltage_rng_max()}')
0.9
```

input_voltage_rng_min()

The string of min_value specified in input_voltage_range(min_value, max_value) of an input pin.

Note:

Only for PinHdl of type() = npiNlInstPort, and direction() = npiNlInput. And InstHdl.inst_type()= npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The input voltage range min value.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.A").
input_voltage_rng_min()}')
0.7
```

output_voltage_rng_max()

The string of max_value specified in output_voltage_range(min_value, max_value) of an output pin.

Note:

Only for PinHdl of type() = npiNlInstPort, and direction() = npiNlOutput. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The output voltage range max value.

Return type: str

Feedback

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.B").
output_voltage_rng_max()}')
1.3
```

output_voltage_rng_min()

The string of min_value specified in output_voltage_range(min_value, max_value) of an output pin.

Note:

Only for PinHdl of type() = npiNlInstPort, and direction() = npiNlOutput. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: The output voltage range min value.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m5.B").
output_voltage_rng_min()}')
1.1
```

ret_pin_class_name()

The pin_class of the retention_pin(pin_class, disable_value).

Note:

Only for npiNlInstPort inferred from npiNlCell: retention_cell.

Parameters: None.

Returns: the pin class name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_instport("top.m6.RETN").
ret_pin_class_name()}')
save_restore
```

is_always_on()

Whether the always_on attribute is set.

Feedback

Note:

Only for PinHdl of type() = npiNlInstPort. And InstHdl.inst_type() = npiNlSymbolLibInst of its scope InstHdl.

Parameters: None.

Returns: True if it is always on.

Return type: bool

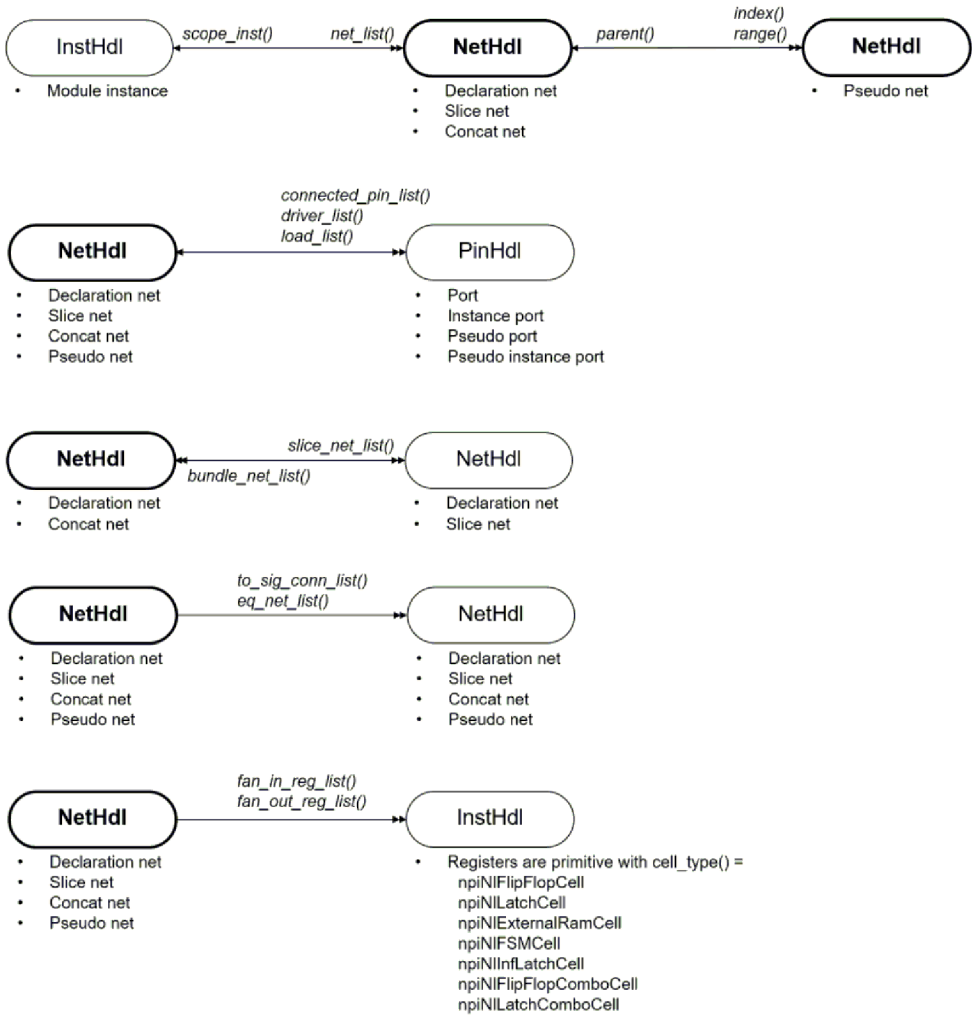
Examples:

```
>>> print(f'{netlist.get_instport("top.m6.RETN").is_always_on()}')
True
```

Feedback

NetHdl

Figure 4 Object diagram



class netlist.NetHdl(cps_obj)

A net object represents physical connections between PinHdl objects.

NetHdl includes three types: decl net, slice net, and concat net.

Decl net:

Represents the full bus of a declared net.

Feedback

Slice net:

Represents a variable slice in the source code.

Concat net:

An object concatenated with decl net and slice net objects.

Following is an example:

```
module top;
 wire [3:0] a, b;
 wire [5:0] c = {a[2:0],b[1:0]};
endmodule
```

Where a[3:0] and b[3:0] are decl nets; a[2:0] and b[1:0] are slice nets; And a[2:0],b[1:0] is a concat net.

You can query a NetHdl by public functions or traversed by get_net() or traversed by InstHdl.net_list().

1-1 Method

scope_inst()	Get upper scope instance handle.
parent()	Get parent net handle from which the pseudo net is created.
index(index)	Get pseudo net of specified index.
range(left, right)	Get pseudo net of specified range index.

1-m Method

connected_pin_list()	Get the connected port/instport handles in list.
driver_list()	Get its driver port/instport handles in list.
load_list()	Get its load port/instport handles in list.
fan_in_reg_list(stop_at_pin=False, report_primary_port=False, top_scope_name=None)	Trace fan-in cone and find all fan-in register instances.

fan_out_reg_list(stop_at_pin=False, report_primary_port=False, top_scope_name=None)	Trace fan-out cone and find all fan-out register instances.
to_sig_conn_list(to_hdl, assign_cell=False)	Find one of a connective path between two NetHdl objects.
slice_net_list()	Get sub member nets from npiNlDeclNet or npiNlScliceNet from npiNlConcatNet, or get npiNlScliceNet from npiNlDeclNet.
bundle_net_list()	Get the npiNlConcatNet from npiNlDeclNet or npiNlSliceNet, or get npiNlDeclNet from npiNlSliceNet.
eq_net_list(assign_cell=False)	Get equivelent nets.

General Properties

type(get_enum=False)	Get upper scope instance handle.
name()	The short name string.
full_name()	The full hierarchical name string.
info()	The infomation string in style: type, full_name.
lang_type(get_enum=False)	The language type string/int.
size()	The size of net.
left()	The left index of net.
right()	The right index of net.
is_signed()	Whether the net object is a signed signal.
is_generated()	Whether net object is generated by nSchema.
is_instrumented()	Whether the net object is generated with an instrumented cell.
is_generated()	Whether net object is generated by nSchema.
is_instrumented()	Whether the net object is generated with an instrumented cell.

Feedback

is_external_reference()	Whether net object is an external reference.
is_literal()	Whether the handle is inferred from a literal or a supply net.
value(value_format=<ValueFormat.BIN: 0>)	The value string of net in specified format.
actual_name_list(full_name=True)	List all actual names for a concat net or part of concat net.

scope_inst()

Get upper scope instance handle.

Note:

Only for NetHdl of type() = npiNlDeclNet/npiNlSliceNet/npiNlConcatNet.

Parameters: None.

Returns: The handle of upper scope.

Return type: InstHdl

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a[3:0]").scope_inst()}')
InstHdl('npiNlInst', 'top.m1')
```

parent()

Get parent port/instport handle from which the pseudo port/instport is created.

Note:

Only for NetHdl of type()= npiNlPseudoNet.

Parameters: None.

Returns: The parent handle.

Return type: NetHdl

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a[1]").parent()}')
NetHdl('npiNlDeclNet', 'top.m1.a[3:0]')
```

index(*index*)

Get pseudo port/instport of specified index.

Parameters: index (int) – Index number to query from the referenced handle.

Returns: pseudo handle based on the index number of the referenced handle

Return type: NetHdl

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").index(1)}')
NetHdl('npiNlPseudoNet', 'top.m1.a[3:0]#[1]')
```

range(*left, right*)

Get pseudo net of specified range index.

Parameters:

- **left** (int)– The left bound index to query from the referenced handle.
- **right** (int) – The right bound index to query from the referenced handle.

Returns: pseudo handle based on the index range of the referenced handle

Return type: NetHdl

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").range(2,1)}')
NetHdl('npiNlPseudoNet', 'top.m1.a[3:0]#[2:1]')
```

connected_pin_list()

Get the connected port/instport handles in list.

Parameters: None.

Returns: A list of connected PinHdl is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").connected_pin_list()}')
[PinHdl('npiNlInstPort',
'top.m1.M:Always0#SigTap0:58:58:Assignment.IH_a[3:0]'),
PinHdl('npiNlInstPort',
'top.m1.M:Always1#Always0:55:57:Or.IH_a[3:0]'),
PinHdl('npiNlPort', 'top.m1.a[3:0]')]
```

driver_list()

Get its driver port/instport handles in list.

Parameters: None.

Returns: The driver list of PinHdl is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").driver_list()}')
[PinHdl('npiNlPort', 'top.m1.a[3:0]')]
```

load_list()

Get its load port/instport handles in list.

Parameters: None.

Returns: The load list of PinHdl is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").load_list()}')
[PinHdl('npiNlInstPort',
'top.m1.M:Always0#SigTap0:58:58:Assignment.IH_a[3:0]'),
PinHdl('npiNlInstPort',
'top.m1.M:Always1#Always0:55:57:Or.IH_a[3:0]')]
```

fan_in_reg_list(*stop_at_pin=False, report_primary_port=False, top_scope_name=None*)

Trace fan-in cone and find all fan-in register instances.

Parameters:

- **stop_at_pin** (bool) – True: it will return the register pin instead of register instance. Default: False.
- **report_primary_port** (bool) – True: it will also collect the primary port when hit the scope defined on top_scope_name. Default: False.
- **top_scope_name** (str) – When report_primary_port is True, top_scope_name specify the hierarchical module instance name as boundary which traverse will not cross. Default: None, the boundary will be the top scopes in the design.

Returns: A list of the fan-in register instances/pins is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.out").fan_in_reg_list()}')
[InstHdl('npiNlInst', 'top.m1.M:Always2#Always0:55:57:Reg')]
```

```
>>> print(f'{netlist.get_net("top.m1.out]").fan_in_reg_list(True)}')
[PinHdl('npiNlInstPort',
'top.m1.M:Always2#Always0:55:57:Reg.ROH_out[3:0]')]

>>> print(f'{netlist.get_net("top.m1.a]").fan_in_reg_list(False,
True, "top.m1")}')
[PinHdl('npiNlPort', 'top.m1.a[3:0]')]
```

fan_out_reg_list(*stop_at_pin=False, report_primary_port=False, top_scope_name=None*)

Trace fan-out cone and find all fan-out register instances.

Parameters:

- **stop_at_pin** (bool) – True: it will return the register pin instead of register instance. Default: False.
- **report_primary_port** (bool) – True: it will also collect the primary port when hit the scope defined on top_scope_name. Default: False.
- **top_scope_name** (str) – When report_primary_port is True, top_scope_name specify the hierarchical module instance name as boundary which traverse will not cross. Default: None, the boundary will be the top scopes in the design.

Returns: A list of the fan-out register instances/pins is returned.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.clk").fan_out_reg_list()}')
[InstHdl('npiNlInst', 'top.m1.M:Always2#Always0:55:57:Reg')]

>>> print(f'{netlist.get_net("top.m1.clk").fan_out_reg_list(True)}')
[PinHdl('npiNlInstPort',
'top.m1.M:Always2#Always0:55:57:Reg.CKR_clk')]

>>> print(f'{netlist.get_net("top.m1.out").fan_out_reg_list(False,
True, "top.m1")}')
[PinHdl('npiNlPort', 'top.m1.out[3:0]')]
```

to_sig_conn_list(*to_hdl, assign_cell=False*)

Find one of a connective path between two NetHdl objects.

This function is the same as sig_to_sig_conn_list().

Feedback

Parameters:

- **to_hdl** (NetHdl) – The handle of the destination net.
- **assign_cell** (bool) – Specify whether or not an npiNlAssignCell is treated as a primitive cell. Default: False.

Returns: Returns a list of the NetHdl objects along the path found.

Return type: list

Examples:

```
>>> hdl1 = netlist.get_net('top.wa')
>>> hdl2 = netlist.get_net('top.wout')
>>> print(f'{hdl1.to_sig_conn_list(hdl2)}')
[NetHdl('npiNlDeclNet', 'top.wa[3:0]'), NetHdl('npiNlDeclNet',
'top.m1.a[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.GEN0_out[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.out[3:0]'),
NetHdl('npiNlDeclNet', 'top.wout[3:0]')]
```

slice_net_list()

Get sub member nets from npiNlDeclNet or npiNlSliceNet from npiNlConcatNet, or get npiNlSliceNet from npiNlDeclNet.

Parameters: None.

Returns: Returns a list of slice net handles..

Return type: list

Examples:

```
>>> hdl = netlist.get_net("top.2'b01,3'b011,a[1:0]")
>>> print(f'{hdl.slice_net_list()}')
[NetHdl('npiNlDeclNet', 'top.2'b01'),
NetHdl('npiNlDeclNet', 'top.3'b011'),
NetHdl('npiNlDeclNet', 'top.a[1:0]')]
```

bundle_net_list()

Get the npiNlConcatNet from npiNlDeclNet or npiNlSliceNet, or get npiNlDeclNet from npiNlSliceNet.

Parameters: None.

Returns: Returns a list of bundle net handles.

Return type: list

Examples:

```
>>> hdl = netlist.get_net("top.2'b01")
>>> print(f'{hdl.bundle_net_list()}')
[NetHdl('npiNlConcatNet', 'top.2'b01,3'b011,a[1:0]')]
```

eq_net_list(*assign_cell=False*)

Get equivalent nets.

Parameters: **assign_cell** (bool)

- **True**: An npiNlAssignCell will be treated as a primitive cell.
- **False**: An npiNlAssignCell will be passed through when traversing.
- **Default**: False.

Returns: Returns a list of equivalent net handles.

Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").eq_net_list()}')
[NetHdl('npiNlDeclNet', 'top.m1.c[3:0]'), NetHdl('npiNlDeclNet',
'top.wa[3:0]')]
>>> print(f'{netlist.get_net("top.b[1:0]").eq_net_list()}')
[NetHdl('npiNlPseudoNet', 'top.a[1:0],b[1:0]#[2:3]'),
NetHdl('npiNlPseudoNet', 'top.wc[3:0]#[1:0]'),
NetHdl('npiNlPseudoNet', 'top.wd[0:3]#[2:3]')]
```

type(*get_enum=False*)

The type string/enum.

Parameters: get_enum (bool) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type: str/ ObjectType

Examples:

```
>>> print(f'{netlist.get_net("top.a").type()}')
npiNlDeclNet
>>> print(f'{netlist.get_net("top.a").type(True)}')
4
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_net("top.a").name()}')
a[1:0]
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns: The string of handle hierarchical name.

Return type: str

Examples:

```
>>> print(f'{netlist.get_net("top.a").full_name()}')
top.a[1:0]
```

info()

The information string in style: type, full_name.

Parameters:None.

Returns:The string of handle's information.

Return type: str

Examples:

```
>>> print(f'{netlist.get_net("top.a").info()}')
npiNlDeclNet, top.a[1:0]
```

lang_type(*get_enum=False*)

The language type string/int. For example, npiNlSystemVerilog, npiNlVHDL, npiNlSPICE.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of language type.

Return type: str/int

Examples:

```
>>> print(f'{netlist.get_net("top.a").lang_type()}')
npiNlSystemVerilog
```

```
>>> print(f'{netlist.get_net("top.a").lang_type(True)}')
1
```

size()

The size of net.

Parameters: None.

Returns: The size of handle is returned.

Return type:int

Examples:

```
>>> print(f'{netlist.get_net("top.a").size()}')
2
```

left()

The left index of net.

Parameters: None.

Returns: The left index of handle is returned.

Return type: int

Examples:

```
>>> print(f'{netlist.get_net("top.a").left()}')
```

right()

The right index of net.

Parameters: None.

Returns: The right index of handle is returned.

Return type: int

Examples:

```
>>> print(f'{netlist.get_net("top.a").right()}')
0
```

is_signed()

Whether the net object is a signed signal.

Parameters: None.

Returns: True if it is signed.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_net("top.a").is_signed()}')
False

>>> print(f'{netlist.get_net("top.wa").is_signed()}')
True
```

is_generated()

Whether net object is generated by nSchema.

Parameters: None.

Returns: True if it is generated.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").is_generated()}')
False

>>> print(f'{netlist.get_net("top.m1.GEN0_out").is_generated()}')
True
```

is_instrumented()

Whether the net object is generated with an instrumented cell.

Parameters:None.

Returns: True if it is instrumented.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_net("top.m1.a").is_instrumented()}')
False
```

is_external_reference()

Whether net object is an external reference.

Parameters: None.

Returns:True if it is an external reference.

Return type: bool

Examples:

```
>>> print(f'{netlist.get_net("top.a").is_external_reference()}')
False

>>> print(f'{netlist.get_net("top.m7.top.a").
is_external_reference()}')
True
```

is_literal()

Whether the handle is inferred from a literal or a supply net.

Parameters: None.

Returns: True if it is inferred from a literal or a supply net.

Return type: bool

Examples:

```
>>> hdl = netlist.get_net("top.2'b01,3'b011,a[1:0]#[1:4]")
>>> print(f'{hdl.is_literal()}')
True
```

value(*value_format=<ValueFormat.BIN: 0>*)

The value string of net in specified format.

Parameters: value_format (ValueFormat) – The format for the output value string. Default: ValueFormat.BIN

Returns: The string representing the value. If no value, return None.

Return type: str

Examples:

```
>>> hdl = netlist.get_net("top.2'b01,3'b011,a[1:0]#[1:4]")
>>> print(f'{hdl.value()}')
1011
>>> print(f'{hdl.value(netlist.ValueFormat.HEX)}')
b
>>> print(f'{hdl.value(netlist.ValueFormat.OCT)}')
13
>>> print(f'{hdl.value(netlist.ValueFormat.DEC)}')
11
```

actual_name_list(*full_name=True*)

List all actual names for a concat net or part of concat net.

Parameters: full_name (bool) – Specifying whether or not the output string is a full hierarchical name. Default: True.

Returns: A list of actual name strings.

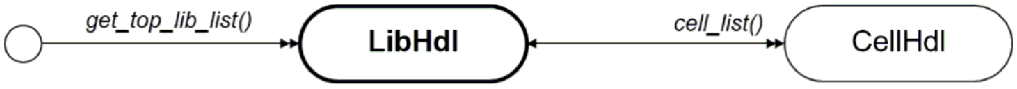
Return type: list

Examples:

```
>>> print(f'{netlist.get_net("top.a[1:0],b[1:0]#[1:2]").
actual_name_list()}')
['top.a[0]', 'top.b[1]']
```

LibHdl

Object diagram:



class netlist.LibHdl(cps_obj)

Library handle, serves as a container for cell objects.

If you import design with symbol libraries, netlist module can create corresponding cells and cellpins based on the cell definitions.

You can traverse LibHdl using get_top_lib_list().

1-m Method

cell_list()	Get internal cell handle list.

General Properties

type(get_enum=False)	The type string/enum.
name()	The short name string.
full_name()	The full hierarchical name string.
info()	The information string in style: type, full_name.
path()	The path of the library.

cell_list()

Get internal cell handle list.

Parameters: None.

Returns: The list of internal cell handles.

Return type: list

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> print(f'{mycell}')
[CellHdl('npiNlCell', 'demo_lib.andd'),
CellHdl('npiNlCell', 'demo_lib.level_shifter'),
CellHdl('npiNlCell', 'demo_lib.ret_cell')]
```

type(*get_enum=False*)

The type string/enum.

Parameters: get_enum (bool) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type: str/ ObjectType

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> print(f'{mylib[0].type()}')
npiNlLib
>>> print(f'{mylib[0].type(True)}')
11
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> print(f'{mylib[0].name()}')
demo_lib
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns:The string of handle hierarchical name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> print(f'{mylib[0].full_name()}')
demo_lib
```

info()

The information string in style: type, full_name.

Parameters: None.

Returns: The string of handle's information.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> print(f'{mylib[0].info()}')
npiNlLib, demo_lib
```

path()

The path of the library.

Parameters: None.

Returns: The path of library.

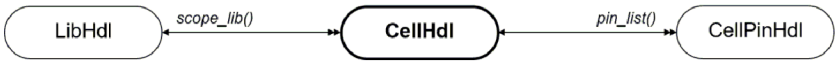
Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> print(f'{mylib[0].path()}')
<pwd>/symlib
```

CellHdl

Object diagram:



class netlist.CellHdl(cps_obj)

Cell handle, which contains the information of cell pins and cell functions.

You can query a CellHdl traversed by LibHdl.cell_list()

1-1 Method

scope_lib()	Get upper scope library handle.

1-m Method

pin_list()	Get internal pin handle list.

General Properties

type(get_enum=False)	The type string/enum.
name()	The short name string.
full_name()	The full hierarchical name string.
info()	The infomation string in style: type, full_name.
cell_type(get_enum=False)	The cell type string/int.
power_cell_type(get_enum=False)	The power cell type string/int inferred from library.
is_memory()	Whether the cell has memory() definition in library
is_pad_cell()	Whether the cell has set the attribute pad_cell true.

scope_lib()

Get upper scope library handle.

Parameters: None.

Returns: The handle of upper scope.

Return type: LibHdl

Examples :

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> print(f'{mycell[0].scope_lib()}')
LibHdl('npiNlLib', 'demo_lib')
```

pin_list()

Get internal pin handle list.

Parameters: None.

Returns: The list of internal pin handles.

Return type: list

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].pin_list()}')
[CellPinHdl('npiNlCellPin', 'demo_lib.andd.A'),
CellPinHdl('npiNlCellPin', 'demo_lib.andd.B'),
CellPinHdl('npiNlCellPin', 'demo_lib.andd.C'),
CellPinHdl('npiNlCellPin', 'demo_lib.andd.D')]
```

type(*get_enum=False*)

The type string/enum.

Parameters: get_enum (bool) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type : str/ ObjectType

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].type()}')
npiNlCell
>>> print(f'{mycell[0].type(True)}')
12
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples :

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].name()}')
andd
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns: The string of handle hierarchical name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].full_name()}')
demo_lib.andd
```

info()

The infomation string in style: type, full_name.

Parameters: None.

Returns: The string of handle's information.

Return type:str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].info()}')
npiNlCell, demo_lib.andd
```

cell_type(*get_enum=False*)

The cell type string/int.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns:The string/int of cell type.

Return type: str/int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[0].cell_type()}')
npiNlMacroCell
```

power_cell_type(*get_enum=False*)

The power cell type string/int inferred from library.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of power cell type.

Return type: str/int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[2].power_cell_type()}')
npiNlPowerRETCell
```

is_memory()

Whether the cell has memory() definition in library.

Parameters: None.

Returns: True if it is a memory cell.

Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[1].is_memory_cell()}')
False
```

is_pad_cell()

Whether the cell has set the attribute pad_cell true.

Parameters: None.

Returns: True if it is a pad cell.

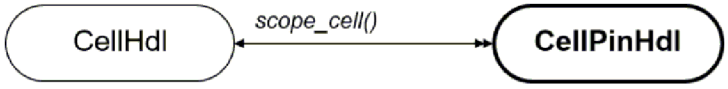
Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> print(f'{mycell[1].is_pad_cell()}')
True
```

CellPinHdl

Object diagram:



class netlist.**CellPinHdl**(*cps_obj*)

A cell pin is the pin defined in liberary cell.

You can query a CellHdl traversed byCellHdl.pin_list() .

1-m Method

scope_cell()	Get upper scope cell handle.

General Properties

type(get_enum=False)	The type string/enum.
name()	The short name string.
full_name()	The full hierarchical name string.
info()	The information string in style: type, full_name.
port_type(get_enum=False)	The path of the library.
direction(get_enum=False)	The direction string/int.
size()	The size of cell pin.
left()	The left index of cell pin.
right()	The right index of cell pin.
func()	The function attribute defined in the library.

x_func()	The x_function attribute of instport defined in the library.
three_state_func()	The three_state attribute of instport defined in the library.
is_pg_pin()	Whether the cell pin is a pg_pin.
is_std_cell_main_rail_pin()	Whether the std_cell_main_rail attribute is set.
voltage_name()	The string of its specified voltage_name.
pg_func()	The string of a pg_pin's specified pg_function.
switch_func()	The string of a pg_pin's specified switch_function.
is_pad()	Whether the is_pad attribute is set.
related_power()	The string of its specified related_power_pin.
related_ground()	The string of its specified related_ground_pin.
power_down_func()	The string of its specified power_down_function.
input_voltage_rng_max()	The string of max_value specified in input_voltage_range(min_value, max_value) of an input pin.
input_voltage_rng_min()	The string of min_value specified in input_voltage_range(min_value, max_value) of an input pin.
output_voltage_rng_max()	The string of max_value specified in output_voltage_range(min_value, max_value) of an output pin.
output_voltage_rng_min()	The string of min_value specified in output_voltage_range(min_value, max_value) of an output pin.
ret_pin_class_name()	The pin_class of the retention_pin(pin_class, disable_value).
is_always_on()	Whether the always_on attribute is set.

scope_cell()

Get upper scope cell handle.

Parameters: None.

Returns: The handle of upper scope.

Return type: CellHdl

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].scope_cell()}')
CellHdl('npiNlCell', 'demo_lib.andd')
```

type(*get_enum=False*)

The type string/enum.

Parameters: get_enum (bool) – Determine if return in enum ObjectType. Default: False.

Returns: The string/enum of handle type

Return type: str/ObjectType

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].type()}')
npiNlCellPin
>>> print(f'{mycellpin[0].type(True)}')
13
```

name()

The short name string.

Parameters: None.

Returns: The string of handle name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].name()}')
A
```

full_name()

The full hierarchical name string.

Parameters: None.

Returns: The string of handle hierarchical name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].full_name()}')
demo_lib.andd.A
```

info()

The information string in style: type, full_name.

Parameters: None.

Returns: The string of handle's information.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_listi()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].info()}')
npiNlCellPin, demo_lib.andd.A
```

port_type(*get_enum=False*)

The port functional type string/int. For example, npiNlDataPort, npiNlClockPort, npiNlControlPort, npiNlTriEnablePort, npiNlRegisteredPort.

Parameters: **get_enum** (bool) – Determine if return in integer. Default: False.

Returns: The string/int of port type.

Return type: str/int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].port_type()}')
npiNlDataPort
```

direction(*get_enum=False*)

The direction string/int. For example, npiNlInput, npiNlOutput, or npiNlInout.

Parameters: get_enum (bool) – Determine if return in integer. Default: False.

Returns: The string/int of port direction.

Return type: str/int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].direction()}')
npiNlInput
```

size()

The size of cell pin.

Parameters: None.

Returns: The size of handle is returned.

Return type: int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].size()}')
1
```

left()

The left index of cell pin.

Parameters: None.

Returns: The left index of handle is returned.

Return type: int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].left()}')
0
```

right()

The right index of cell pin.

Parameters: None.

Returns: The right index of handle is returned.

Return type: int

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[0].right()}')
0
```

func()

The function attribute defined in the library.

Parameters: None.

Returns: The function string.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[3].func()}')
(A&B&C)
```

x_func()

The x_function attribute of instport defined in the library.

Parameters: None.

Returns: The x_function string.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[3].x_func()}')
(!(B)&C)
```

three_state_func()

The three_state attribute of instport defined in the library.

Parameters: None.

Returns: The three_state string.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[0].port_list()
>>> print(f'{mycellpin[3].three_state_func()}')
(B&!(C))
```

is_pg_pin()

Whether the cell pin is a pg_pin.

Parameters: None.

Returns: True, if it is pg pin.

Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[2].is_pg_pin()}')
True
```

is_std_cell_main_rail_pin()

Whether the std_cell_main_rail attribute is set.

Parameters: None.

Returns: True, if the std_cell_main_rail attribute is set.

Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[2].is_std_cell_main_rail_pin()}')
True
```

voltage_name()

The string of its specified voltage_name.

Parameters: None.

Returns: The voltage_name of a cell pg_pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[2].voltage_name()}')
VDD
```

pg_func()

The string of a pg_pin's specified pg_function.

Parameters: None.

Returns: The pg_function of a cell pg_pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[2].pg_func()}')
VSS
```

switch_func()

The string of a pg_pin's specified switch_function.

Parameters: None.

Returns: The switch_function of a cell pg_pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[2].switch_func()}')
VDD + VSS
```

is_pad()

Whether the is_pad attribute is set.

Parameters: None.

Returns: True if it is pad.

Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[1].is_pad()}')
True
```

related_power()

The string of its specified related_power_pin.

Parameters: None.

Returns: The related_power_pin of a cell pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[0].related_power()}')
VDD
```

related_ground()

The string of its specified related_ground_pin.

Parameters: None.

Returns: The related_ground_pin of a cell pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[0].related_power()}')
VSS
```

power_down_func()

The string of its specified power_down_function.

Parameters: None.

Returns: The power_down_function of a cell pin.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[1].power_down_func()}')
!VDD + VSS
```

input_voltage_rng_max()

The string of max_value specified in input_voltage_range(min_value, max_value) of an input pin.

Parameters: None.

Returns: The input voltage range max value.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[0].input_voltage_rng_max()}')
0.9
```

input_voltage_rng_min()

The string of min_value specified in input_voltage_range (min_value, max_value) of an input pin.

Parameters: None.

Returns: The input voltage range min value.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[0].input_voltage_rng_min()}')
0.7
```

output_voltage_rng_max()

The string of max_value specified in output_voltage_range(min_value, max_value) of an output pin.

Parameters: None.

Returns: The output voltage range max value.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[1].output_voltage_rng_max()}')
1.3
```

output_voltage_rng_min()

The string of min_value specified in output_voltage_range (min_value, max_value) of an output pin.

Parameters: None.

Returns: The output voltage range min value.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[1].port_list()
>>> print(f'{mycellpin[1].output_voltage_rng_min()}')
1.1
```

ret_pin_class_name()

The pin_class of the retention_pin (pin_class, disable_value).

Parameters: None.

Returns: the pin class name.

Return type: str

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[2].port_list()
>>> print(f'{mycellpin[0].ret_pin_class_name()}')
save_restore
```

is_always_on()

Whether the always_on attribute is set.

Feedback

Parameters: None.

Returns: True if it is always on.

Return type: bool

Examples:

```
>>> mylib = netlist.get_top_lib_list()
>>> mycell = mylib[0].cell_list()
>>> mycellpin = mycell[2].port_list()
>>> print(f'{mycellpin[0].is_always_on()}')
True
```

Public Functions

Get Handle by Full Hier Name

netlist.get_inst(name)	Get instance handle by full hierarchy name.
netlist.get_port(name)	Get port handle by full hierarchy name.
netlist.get_instport(name)	Get instport handle by full hierarchy name
netlist.get_net(name)	Get net handle by full hierarchy name.
netlist.get_actual_net(name)	Get net handle by full hierarchy actual name.

Get Handle List from Top Scope

netlist.get_top_inst_list()	Get top instances of the whole design.
netlist.get_top_lib_list()	Get top libraries of the whole desing design.

Connection

netlist.sig_to_sig_conn_list(from_hdl, to_hdl, assign_cell=False)	Find one of the connective path between two NetHdl objects.

Hierarchy Tree Trv Callback Function

Feedback

netlist.hier_tree_trv(scope_hier_name=None)	Traverse the hierarchy tree from the specified scope, and execute the callback function registered by user.
netlist.hier_tree_trv_register_cb(obj_type, cb_func, cb_data)	Register the callback function as it will create the association between user-defined callback function/data and hier_tree_trv().
netlist.hier_tree_trv_reset_cb()	Reset the callback function for hier_tree_trv().

Tracing Callback Function

netlist.register_cb(func_type, cb_func, cb_data)	Register callback function as it will create the association between user-defined callback function/data and specific connection funtions.
netlist.reset_cb()	Reset callback for connection functions.

Get Handle by Full Hier Name

netlist.get_inst(*name*)

Get instance handle by full hierarchy name.

Parameters: *name* (str) – The full hierarchical name.

Returns: The instance with specified name.

Return type: InstHdl

Examples:

```
>>> print(f'{netlist.get_inst("top.m1")}')
InstHdl('npiNlInst', 'top.m1')
```

netlist.get_port(*name*)

Get port handle by full hierarchy name.

Parameters: *name* (str) – The full hierarchical name.

Returns: The port with specified name.

Return type: PinHdl

Feedback

Examples:

```
>>> print(f'{netlist.get_port("top.m1.a")}')
PinHdl('npiNlPort', 'top.m1.a[3:0]')
```

netlist.get_instport(*name*)

Get instport handle by full hierarchy name.

Parameters: **name** (str) – The full hierarchical name.

Returns: The instport with specified name.

Return type: PinHdl

Examples:

```
>>> print(f'{netlist.get_instport("top.m1.a")}')
PinHdl('npiNlInstPort', 'top.m1.a[3:0]')
```

netlist.get_net(*name*)

Get net handle by full hierarchy name.

Parameters: name (str) – The full hierarchical name.

Returns: The net with specified name.

Return type: NetHdl

Examples:

```
>>> print(f'{netlist.get_net("top.wa")}')
NetHdl('npiNlDeclNet', 'top.wa[3:0]')
```

netlist.get_actual_net(*name*)

Get net handle by full hierarchy actual name.

Parameters: name (str) – The full hierarchical name.

Returns: The handle with specified name.

Return type: NetHdl

Examples:

```
>>> print(f'{netlist.get_actual_net("top.wa[1]")}'
NetHdl('npiNlPseudoNet', 'top.wa[3:0]#[1]')
```

Feedback

Get Handle List from Top Scope

netlist.get_top_inst_list()

Get top instances of the whole design.

Parameters: None.

Returns: list of top scope instances (InstHdl).

Return type: list

Examples:

```
>>> print(f'{netlist.get_top_inst_list()}')
[InstHdl('npiNlInst', 'top')]
```

netlist.get_top_lib_list()

Get top libraries of the whole design.

Parameters: None.

Returns: list of top libraries (InstHdl).

Return type: list

Examples:

```
>>> print(f'{netlist.get_top_lib_list()}')
[LibHdl('npiNlLib', 'demo_lib')]
```

Connection

netlist.sig_to_sig_conn_list(*from_hdl, to_hdl, assign_cell=False*)

Find one of the connective path between two NetHdl objects. This function is the same as NetHdl.to_sig_conn_list()

Parameters:

from_hdl(NetHdl) – the handle of the source net.

to_hdl (NetHdl) – the handle of the destination net.

assign_cell (bool) – specifying whether or not an npiNlAssignCell is treated as a primitive cell. Default: False.

Returns: A list of the NetHdls along with the path found.

Return type: list

Feedback

Examples:

```
>>> hdl1 = netlist.get_net('top.wa')
>>> hdl2 = netlist.get_net('top.wout')
>>> print(f'{netlist.sig_to_sig_conn_list(hdl1, hdl2)}')
[NetHdl('npiNlDeclNet', 'top.wa[3:0]'), NetHdl('npiNlDeclNet',
'top.m1.a[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.GEN0_out[3:0]'),
NetHdl('npiNlDeclNet', 'top.m1.out[3:0]'),
NetHdl('npiNlDeclNet', 'top.wout[3:0]')]
```

Hierarchy Tree Trv Callback Function

netlist.hier_tree_trv(*scope_hier_name=None*)

Traverse the hierarchy tree from the specified scope, and execute the callback function registered by user.

Parameters: scope_hier_name (str) – The full hier name of target scope. Default: None, it will traverse from top scopes.

Returns: Returns 1 if the corresponding scope can be found; otherwise, returns 0.

Return type: int

Examples:

```
>>> def cb_func_inst_obj(hdl, cb):
 print(f' INST: {hdl.info()}', file=f)
 cb.append(hdl)
>>> inst_list = []
>>> netlist.hier_tree_trv_register_cb(netlist.ObjectType.INST,
cb_func_inst_obj, inst_list)
>>> netlist.hier_tree_trv('top.m1')
INST: npiNlInst, top.m1, {demo.v : 32 : 32}
INST: npiNlInst, top.m1.M:Always0#SigTap0:58:58:Assignment, {demo.v :
58 : 58}
INST: npiNlInst, top.m1.M:Always1#Always0:55:57:Or, {demo.v : 56 :
56}
INST: npiNlInst, top.m1.M:Always2#Always0:55:57:Reg, {demo.v : 55 :
55} {demo.v : 56 : 56}
>>> print(f'{inst_list}')
[InstHdl('npiNlInst', 'top.m1'),
InstHdl('npiNlInst', 'top.m1.M:Always0#SigTap0:58:58:Assignment'),
InstHdl('npiNlInst', 'top.m1.M:Always1#Always0:55:57:Or'),
InstHdl('npiNlInst', 'top.m1.M:Always2#Always0:55:57:Reg')]
```

netlist.hier_tree_trv_register_cb(*obj_type, cb_func, cb_data*)

Register the callback function as it will create the association between user-defined callback function/data and hier_tree_trv().

Feedback

Parameters:

- **obj_type**(ObjectType) – Object types that can be iterated from scope. Which are [ObjectType.INST,ObjectType.PORT, ObjectType.INSTPORT, ObjectType.DECL_NET, ObjectType.CONCAT_NET, ObjectType.SLICE_NET
- **cb_func** (function) – The callback function.
- **cb_data** (object) – The callback data.

Returns

- Returns 1 if the input object type is valid to be registered;
- otherwise, returns 0.

Return type: int

Examples:

```
>>> def cb_func_inst_obj(hdl, cb):
 print(f' INST: {hdl.info()}', file=f)
 cb.append(hdl)
>>> inst_list = []
>>> netlist.hier_tree_trv_register_cb(netlist.ObjectType.INST,
cb_func_inst_obj, inst_list)
>>> netlist.hier_tree_trv('top.m1')
INST: npiNlInst, top.m1, {demo.v : 32 : 32}
INST: npiNlInst, top.m1.M:Always0#SigTap0:58:58:Assignment, {demo.v :
58 : 58}
INST: npiNlInst, top.m1.M:Always1#Always0:55:57:Or, {demo.v : 56 :
56}
INST: npiNlInst, top.m1.M:Always2#Always0:55:57:Reg, {demo.v : 55 :
55} {demo.v : 56 : 56}
>>> print(f'{inst_list}')
[InstHdl('npiNlInst', 'top.m1'),
InstHdl('npiNlInst', 'top.m1.M:Always0#SigTap0:58:58:Assignment'),
InstHdl('npiNlInst', 'top.m1.M:Always1#Always0:55:57:Or'),
InstHdl('npiNlInst', 'top.m1.M:Always2#Always0:55:57:Reg')]
```

netlist.hier_tree_trv_reset_cb()

Reset the callback function for hier_tree_trv(). It removes the association between user-defined callback function/data and hier_tree_trv().

Parameters: None.

Returns: None.

Examples:

```
>>> def cb_func_inst_obj(hdl, cb):
 print(f' INST: {hdl.info()}', file=f)
```

```
 cb.append(hdl)
>>> inst_list = []
>>> netlist.hier_tree_trv_register_cb(netlist.ObjectType.INST,
cb_func_inst_obj, inst_list)
>>> netlist.hier_tree_trv('top.m1')
INST: npiNlInst, top.m1, {demo.v : 32 : 32}
INST: npiNlInst, top.m1.M:Always0#SigTap0:58:58:Assignment, {demo.v :
58 : 58}
INST: npiNlInst, top.m1.M:Always1#Always0:55:57:Or, {demo.v : 56 :
56}
INST: npiNlInst, top.m1.M:Always2#Always0:55:57:Reg, {demo.v : 55 :
55} {demo.v : 56 : 56}
>>> netlist.hier_tree_trv_reset_cb()
>>> netlist.hier_tree_trv()
```

Tracing Callback Function

netlist.register_cb(*func_type, cb_func, cb_data*)

Register callback function as it creates the association between user-defined callback function/data and specific connection functions.

Parameters:

- **func_type** (FuncType) – Function type to be registered.
- **cb_func** (function) – The callback function will be called when each handle is traversed. If the function returns False, the tracing will be stopped on that handle.
- **cb_data** (object) – The callback data.

Returns: Returns 1 if the callback function is valid to be registered; otherwise, returns 0.

Return type: int

Examples:

This code registers a callback function to collect AND cells when tracing. Besides, it will stop trace while encounter the OR cell.

```
>>> def cb_func(hdl, cb):
 inst = hdl.scope_inst()
 cellType = inst.cell_type()
 if cellType == 'npiNlOrCell':
 return False
 if cellType == 'npiNlAndCell':
 cb.append(inst)
 return True
>>> inst_list = []
>>> netlist.register_cb(netlist.FuncType.FAN_IN, cb_func, inst_list)
>>> hdl = netlist.get_net('top.a')
```

```
>>> print(f'fan in with callback: {hdl.fan_in_reg_list()}')
fan in with callback: [InstHdl('npiNlInst',
'top.top:Always8#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always9#Always0:42:46:Reg')]
>>> print(f'{inst_list}')
[InstHdl('npiNlInst', 'top.top:Always5#SigOp5:39:39:And')]
>>> netlist.reset_cb()
>>> print(f'fan in without callback: {hdl.fan_in_reg_list()}')
fan in without callback: [InstHdl('npiNlInst',
'top.top:Always10#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always8#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always9#Always0:42:46:Reg')]
```

netlist.reset_cb()

Reset callback for connection functions.

Parameters: None.

Returns: None.

Examples:

This code registers a callback function to collect AND cells when tracing. Also, it will stop trace while encounter the OR cell.

```
>>> def cb_func(hdl, cb):
 inst = hdl.scope_inst()
 cellType = inst.cell_type()
 if cellType == 'npiNlOrCell':
 return False
 if cellType == 'npiNlAndCell':
 cb.append(inst)
 return True
>>> inst_list = []
>>> netlist.register_cb(netlist.FuncType.FAN_IN, cb_func, inst_list)
>>> hdl = netlist.get_net('top.a')
>>> print(f'fan in with callback: {hdl.fan_in_reg_list()}')
fan in with callback: [InstHdl('npiNlInst',
'top.top:Always8#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always9#Always0:42:46:Reg')]
>>> print(f'{inst_list}')
[InstHdl('npiNlInst', 'top.top:Always5#SigOp5:39:39:And')]
>>> netlist.reset_cb()
>>> print(f'fan in without callback: {hdl.fan_in_reg_list()}')
fan in without callback: [InstHdl('npiNlInst',
'top.top:Always10#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always8#Always0:42:46:Reg'),
InstHdl('npiNlInst', 'top.top:Always9#Always0:42:46:Reg')]
```