skidl 0.0.29 documentation

This is a really, really bad API document. I'm working to make Sphinx generate a better one but, right now, the description on the homepage about how to use SKiDL is much better than this.

SKiDL API

Overview of How SKiDL Works

This is an *overview* of how SKiDL works.

skidl

skidl package

Subpackages

skidl.tools package

Submodules

skidl.tools.kicad module

Handler for reading Kicad libraries and generating netlists.

skidl.tools.kicad.**_load_sch_lib_**(*self*, *filename=None*, *lib_search_paths_=None*)

Load the parts from a KiCad schematic library file.

Parameters

filename – The name of the KiCad schematic library file.

skidl.tools.kicad.**_parse_lib_part_**(*self*, *get_name_only=False*)

Create a Part using a part definition from a KiCad schematic library.

This method was written based on the code from https://github.com/KiCad/kicad-library-utils/tree/master/schlib. It's covered by GPL3.

Parameters

- **part_defn** – A list of strings that define the part (usually read from a schematic library file). Can also be None.
- **get_name_only** – If true, scan the part definition until the name and aliases are found. The rest of the definition will be parsed if the part is actually used.

skidl.tools.kicad.**_gen_netlist_**(*self*)

skidl.tools.kicad.**_gen_netlist_comp_**(*self*)

skidl.tools.kicad.**_gen_netlist_net_**(*self*)

skidl.tools.kicad.**_gen_xml_**(*self*)

skidl.tools.kicad.**_gen_xml_comp_**(*self*)

skidl.tools.kicad.**_gen_xml_net_**(*self*)

skidl.tools.skidl module

Handler for reading SKiDL libraries and generating netlists.

skidl.tools.skidl.**_load_sch_lib_**(*self*, *filename=None*, *lib_search_paths_=None*)

Load the parts from a SKiDL schematic library file.

Parameters

SEARCH

Go

filename – The name of the SKiDL schematic library file.

skidl.tools.skidl.**_parse_lib_part_**(*self*, *get_name_only=False*)

Create a Part using a part definition from a SKiDL library.

skidl.tools.spice module

Handler for reading SPICE libraries.

skidl.tools.spice.**_load_sch_lib_**(*self*, *filename=None*, *lib_search_paths_=None*)

Load the .subckt I/O from a SPICE library file.

Parameters

- **filename** – The name of the SPICE library file.
- **lib_search_paths** – List of directories to search for the file.

skidl.tools.spice.**_parse_lib_part_**(*self*, *get_name_only=False*)

Create a Part using a part definition from a SPICE library.

skidl.tools.spice.**_mk_subckt_part**(*defn='XXX'*)

class skidl.tools.spice.**XspiceModel**(**args*, ***kwargs*)

Bases: `object`

Object to hold the parameters for an XSPICE model.

__init__(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

skidl.tools.spice.**_gen_netlist_**(*self*, ***kwargs*)

Return a PySpice Circuit generated from a SKiDL circuit.

Parameters

- **title** – String containing the title for the PySpice circuit.
- **libs** – String or list of strings containing the paths to directories containing SPICE models.

skidl.tools.spice.**node**(*net_or_pin*)

skidl.tools.spice.**_get_spice_ref**(*part*)

Return a SPICE reference ID for the part.

skidl.tools.spice.**_get_kwargs**(*part*, *kw*)

Return a dict of keyword arguments to PySpice element constructor.

skidl.tools.spice.**not_implemented**(*part*, *circuit*)

Unable to add a particular SPICE part to a circuit.

skidl.tools.spice.**add_part_to_circuit**(*part*, *circuit*)

Add a part to a PySpice Circuit object.

Parameters

- **part** – SKiDL Part object.
- **circuit** – PySpice Circuit object.

skidl.tools.spice.**_get_net_names**(*part*)

Return a list of net names attached to the pins of a part.

skidl.tools.spice.**add_subcircuit_to_circuit**(*part*, *circuit*)

Add a .SUBCKT part to a PySpice Circuit object.

Parameters

- **part** – SKiDL Part object.

- **circuit** – PySpice Circuit object.

`skidl.tools.spice.`**`add_xspice_to_circuit`**(*part*, *circuit*)

Add an XSPICE part to a PySpice Circuit object.

Parameters

- **part** – SKiDL Part object.
- **circuit** – PySpice Circuit object.

Module contents

This package contains the handler functions for various EDA tools.

Submodules

skidl.Alias module

Handles aliases for Circuit, Part, Pin, Net, Bus, Interface objects.

class `skidl.Alias.`**`Alias`**(**aliases*)

Bases: `set`

Multiple aliases can be added to another object to give it other names.

Parameters

aliases – A single string or a list of strings.

`__init__`(**aliases*)

Initialize self. See help(type(self)) for accurate signature.

`__iadd__`(**aliases*)

Add new aliases.

`__str__`()

Return the aliases as a delimited string.

`__eq__`(*other*)

Return true if both lists of aliases have at least one alias in common.

Parameters

other – The Alias object which self will be compared to.

`__hash__` = *None*

skidl.Bus module

Handles buses.

class `skidl.Bus.`**`Bus`**(*name*, **args*, ***attribs*)

Bases: `skidl.baseobj.SkidlBaseObject`

This class collects one or more nets into a group that can be indexed.

Parameters

- **name** – A string with the name of the bus.
- **args** – A list of ints, pins, nets, buses to attach to the net.

Keyword Arguments

attribs – A dictionary of attributes and values to attach to the Net object.

Example

```
n = Net()
led1 = Part("Device", 'LED')
b = Bus('B', 8, n, led1['K'])
```

classmethod **`get`**(*name*, *circuit=None*)

Get the bus with the given name from a circuit, or return None.

classmethod **fetch**(*name*, **args*, ***attribs*)

Get the bus with the given name from a circuit, or create it if not found.

__init__(*name*, **args*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

extend(**objects*)

Extend bus by appending objects to the end (MSB).

insert(*index*, **objects*)

Insert objects into bus starting at indexed position.

get_nets()

Return the list of nets contained in this bus.

get_pins()

It's an error to get the list of pins attached to all bus lines.

copy(*num_copies=None*, ***attribs*)

Make zero or more copies of this bus.

Parameters

num_copies – Number of copies to make of this bus.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Bus copies or a Bus if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a bus can be copied just by calling it like so:

```
b = Bus('A', 8)  # Create a bus.
b_copy = b(2)   # Get two copies of the bus.
```

You can also use the multiplication operator to make copies:

```
b = 10 * Bus('A', 8)  # Create an array of buses.
```

__call__(*num_copies=None*, ***attribs*)

Make zero or more copies of this bus.

Parameters

num_copies – Number of copies to make of this bus.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Bus copies or a Bus if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a bus can be copied just by calling it like so:

```
b = Bus('A', 8)  # Create a bus.
b_copy = b(2)   # Get two copies of the bus.
```

You can also use the multiplication operator to make copies:

```
b = 10 * Bus('A', 8)  # Create an array of buses.
```

`__mul__`(*num_copies*)

`__rmul__`(*num_copies*)

`__getitem__`(**ids*)

Return a bus made up of the nets at the given indices.

Parameters

ids – A list of indices of bus lines. These can be individual numbers, net names, nested lists, or slices.

Returns

A bus if the indices are valid, otherwise None.

`__setitem__`(*ids*, **pins_nets_buses*)

You can't assign to bus lines. You must use the += operator.

This method is a work-around that allows the use of the += for making connections to bus lines while prohibiting direct assignment. Python processes something like my_bus[7:0] += 8 * Pin() as follows:

```
1. Bus.__getitem__ is called with '7:0' as the index. This
   returns a NetPinList of eight nets from my_bus.
2. The NetPinList.__iadd__ method is passed the NetPinList and
   the thing to connect to the it (eight pins in this case). This
   method makes the actual connection to the part pin or pins. Then
   it creates an iadd_flag attribute in the object it returns.
3. Finally, Bus.__setitem__ is called. If the iadd_flag attribute
   is true in the passed argument, then __setitem__ was entered
   as part of processing the += operator. If there is no
   iadd_flag attribute, then __setitem__ was entered as a result
   of using a direct assignment, which is not allowed.
```

`__iter__`()

Return an iterator for stepping thru individual lines of the bus.

`is_movable`()

Return true if the bus is movable to another circuit.

A bus is movable if all the nets in it are movable.

`connect`(**pins_nets_buses*)

Return the bus after connecting one or more nets, pins, or buses.

Parameters

pins_nets_buses – One or more Pin, Net or Bus objects or lists/tuples of them.

Returns

The updated bus with the new connections.

Notes

You can connect nets or pins to a bus like so:

```
p = Pin()        # Create a pin.
n = Net()        # Create a net.
b = Bus('B', 2)  # Create a two-wire bus.
b += p,n         # Connect pin and net to B[0] and B[1].
```

`__iadd__`(**pins_nets_buses*)

Return the bus after connecting one or more nets, pins, or buses.

Parameters

pins_nets_buses – One or more Pin, Net or Bus objects or lists/tuples of them.

Returns

The updated bus with the new connections.

Notes

You can connect nets or pins to a bus like so:

```
p = Pin()       # Create a pin.
n = Net()       # Create a net.
b = Bus('B', 2) # Create a two-wire bus.
b += p,n        # Connect pin and net to B[0] and B[1].
```

name

Get, set and delete the name of the bus.

When setting the bus name, if another bus with the same name is found, the name for this bus is adjusted to make it unique.

__str__()

Return a list of the nets in this bus as a string.

__repr__()

Return a list of the nets in this bus as a string.

__len__()

Return the number of nets in this bus.

width

Return width of a Bus, which is the same as using the len() operator.

__bool__()

Any valid Bus is True

__nonzero__()

Any valid Bus is True

skidl.Circuit module

Handles complete circuits made of parts and nets.

class skidl.Circuit.**Circuit**(***kwargs*)

Bases: skidl.baseobj.SkidlBaseObject

Class object that holds the entire netlist of parts and nets.

parts

List of all the schematic parts as Part objects.

nets

List of all the schematic nets as Net objects.

buses

List of all the buses as Bus objects.

hierarchy

A '.'-separated concatenation of the names of nested SubCircuits at the current time it is read.

level

The current level in the schematic hierarchy.

context

Stack of contexts for each level in the hierarchy.

erc_list = *[<function dflt_circuit_erc>]*

__init__(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

reset(*init=False*)

Clear any circuitry and cached part libraries and start over.

`mini_reset`(*init=False*)

Clear any circuitry but don't erase any loaded part libraries.

`add_parts`(**parts*)

Add some Part objects to the circuit.

`rmv_parts`(**parts*)

Remove some Part objects from the circuit.

`add_nets`(**nets*)

Add some Net objects to the circuit. Assign a net name if necessary.

`rmv_nets`(**nets*)

Remove some Net objects from the circuit.

`add_buses`(**buses*)

Add some Bus objects to the circuit. Assign a bus name if necessary.

`rmv_buses`(**buses*)

Remove some buses from the circuit.

`add_stuff`(**stuff*)

Add Parts, Nets, Buses, and Interfaces to the circuit.

`rmv_stuff`(**stuff*)

Remove Parts, Nets, Buses, and Interfaces from the circuit.

`__iadd__`(**stuff*)

Add Parts, Nets, Buses, and Interfaces to the circuit.

`__isub__`(**stuff*)

Remove Parts, Nets, Buses, and Interfaces from the circuit.

`get_nets`()

Get all the distinct nets for the circuit.

`ERC`(**args*, ***kwargs*)

Run class-wide and local ERC functions on this circuit.

`_merge_net_names`()

Select a single name for each multi-segment net.

`generate_netlist`(***kwargs*)

Return a netlist and also write it to a file/stream.

Parameters

- **file** – Either a file object that can be written to, or a string containing a file name, or None.
- **tool** – The EDA tool the netlist will be generated for.
- **do_backup** – If true, create a library with all the parts in the circuit.

Returns

A netlist.

`generate_xml`(*file_=None*, *tool=None*)

Return netlist as an XML string and also write it to a file/stream.

Parameters

file – Either a file object that can be written to, or a string containing a file name, or None.

Returns

A string containing the netlist.

generate_graph(*file_=None, engine='neato', rankdir='LR', part_shape='rectangle', net_shape='point', splines=None, show_values=True, show_anon=False, split_nets=['GND'], split_parts_ref=[]*)

Returns a graphviz graph as graphviz object and can also write it to a file/stream. When used in ipython the graphviz object will drawn as an SVG in the output.

See https://graphviz.readthedocs.io/en/stable/ and http://graphviz.org/doc/info/attrs.html

Parameters

- **file** – A string containing a file name, or None.
- **engine** – See graphviz documentation
- **rankdir** – See graphviz documentation
- **part_shape** – Shape of the part nodes
- **net_shape** – Shape of the net nodes
- **splines** – Style for the edges, try 'ortho' for a schematic like feel
- **show_values** – Show values as external labels on part nodes
- **show_anon** – Show anonymous net names
- **split_nets** – splits up the plot for the given list of net names
- **split_parts_ref** – splits up the plot for all pins for the given list of part refs

Returns

graphviz.Digraph

backup_parts(*file_=None*)

Saves parts in circuit as a SKiDL library in a file.

Parameters

file – Either a file object that can be written to, or a string containing a file name, or None. If None, a standard library file will be used.

Returns

Nothing.

_gen_netlist_kicad()

_gen_netlist_spice(***kwargs*)

Return a PySpice Circuit generated from a SKiDL circuit.

Parameters

- **title** – String containing the title for the PySpice circuit.
- **libs** – String or list of strings containing the paths to directories containing SPICE models.

_gen_xml_kicad()

skidl.Circuit.**SubCircuit**(*f*)

A @SubCircuit decorator is used to create hierarchical circuits.

Parameters

f – The function containing SKiDL statements that represents a subcircuit.

skidl.Circuit.**subcircuit**(*f*)

A @SubCircuit decorator is used to create hierarchical circuits.

Parameters

f – The function containing SKiDL statements that represents a subcircuit.

skidl.Interface module

Handles interfaces for subsystems with complicated I/O.

class `skidl.Interface.`**`Interface`**(***kwargs*)

Bases: `object`

An Interface bundles a group of nets/buses into a single entity with each net/bus becoming an attribute.

`__init__`(***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

skidl.Net module

Handles nets.

class `skidl.Net.`**`Traversal`**(*nets*, *pins*)

Bases: `tuple`

Create new instance of Traversal(nets, pins)

`__getnewargs__`()

Return self as a plain tuple. Used by copy and pickle.

static **`__new__`**(*_cls*, *nets*, *pins*)

Create new instance of Traversal(nets, pins)

`__repr__`()

Return a nicely formatted representation string

`__slots__` = *()*

`_asdict`()

Return a new OrderedDict which maps field names to their values.

`_field_defaults` = *{}*

`_fields` = *('nets', 'pins')*

`_fields_defaults` = *{}*

classmethod **`_make`**(*iterable*)

Make a new Traversal object from a sequence or iterable

`_replace`(***kwds*)

Return a new Traversal object replacing specified fields with new values

`nets`

Alias for field number 0

`pins`

Alias for field number 1

class `skidl.Net.`**`Net`**(*name=None*, *circuit=None*, **pins_nets_buses*, ***attribs*)

Bases: `skidl.baseobj.SkidlBaseObject`

Lists of connected pins are stored as nets using this class.

Parameters

- **name** – A string with the name of the net. If None or '', then a unique net name will be assigned.
- **circuit** – The Circuit object this net belongs to.
- ***pins_nets_buses** – One or more Pin, Net, or Bus objects or lists/tuples of them to be connected to this net.

Keyword Arguments

attribs – A dictionary of attributes and values to attach to the Net object.

`erc_list` = *[<function dflt_net_erc>]*

classmethod `get`(*name*, *circuit=None*)

Get the net with the given name from a circuit, or return None.

classmethod `fetch`(*name*, **args*, ***attribs*)

Get the net with the given name from a circuit, or create it if not found.

`__init__`(*name=None*, *circuit=None*, **pins_nets_buses*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

`_traverse`()

Return all the nets and pins attached to this net, including itself.

`get_pins`()

Return a list of pins attached to this net.

`get_nets`()

Return a list of nets attached to this net, including this net.

`is_attached`(*pin_net_bus*)

Return true if the pin, net or bus is attached to this one.

`is_movable`()

Return true if the net is movable to another circuit.

A net is movable if it’s not part of a Circuit or if there are no pins attached to it.

`copy`(*num_copies=None*, *circuit=None*, ***attribs*)

Make zero or more copies of this net.

Parameters

num_copies – Number of copies to make of this net.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Net copies or a Net if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a net can be copied just by calling it like so:

```
n = Net('A')     # Create a net.
n_copy = n()     # Copy the net.
```

You can also use the multiplication operator to make copies:

```
n = 10 * Net('A')  # Create an array of nets.
```

`__call__`(*num_copies=None*, *circuit=None*, ***attribs*)

Make zero or more copies of this net.

Parameters

num_copies – Number of copies to make of this net.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Net copies or a Net if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a net can be copied just by calling it like so:

```
n = Net('A')    # Create a net.
n_copy = n()    # Copy the net.
```

You can also use the multiplication operator to make copies:

```
n = 10 * Net('A')  # Create an array of nets.
```

__mul__(*num_copies*)

__rmul__(*num_copies*)

__getitem__(**ids*)

Return the net if the indices resolve to a single index of 0.

Parameters

ids – A list of indices. These can be individual numbers, net names, nested lists, or slices.

Returns

The net, otherwise None or raises an Exception.

__setitem__(*ids*, **pins_nets_buses*)

You can't assign to Nets. You must use the += operator.

This method is a work-around that allows the use of the += for making connections to nets while prohibiting direct assignment. Python processes something like net[0] += Pin() as follows:

```
1. Net.__getitem__ is called with '0' as the index. This
   returns a single Net.
2. The Net.__iadd__ method is passed the net and
   the thing to connect to it (a Pin in this case). This
   method makes the actual connection to the pin. Then
   it creates an iadd_flag attribute in the object it returns.
3. Finally, Net.__setitem__ is called. If the iadd_flag attribute
   is true in the passed argument, then __setitem__ was entered
   as part of processing the += operator. If there is no
   iadd_flag attribute, then __setitem__ was entered as a result
   of using a direct assignment, which is not allowed.
```

__iter__()

Return an iterator for stepping through the net.

is_implicit()

Return true if the net name is implicit.

connect(**pins_nets_buses*)

Return the net after connecting other pins, nets, and buses to it.

Parameters

***pins_nets_buses** – One or more Pin, Net, or Bus objects or lists/tuples of them to be connected to this net.

Returns

The updated net with the new connections.

Notes

Connections to nets can also be made using the += operator like so:

```
atmega = Part('atmel', 'ATMEGA16U2')
net = Net()
net += atmega[1]  # Connects pin 1 of chip to the net.
```

__iadd__(**pins_nets_buses*)

Return the net after connecting other pins, nets, and buses to it.

Parameters

***pins_nets_buses** – One or more Pin, Net, or Bus objects or lists/tuples of them to be connected to this net.

Returns

The updated net with the new connections.

Notes

Connections to nets can also be made using the += operator like so:

```
atmega = Part('atmel', 'ATMEGA16U2')
net = Net()
net += atmega[1]  # Connects pin 1 of chip to the net.
```

disconnect(*pin*)

Remove the pin from this net but not any other nets it's attached to.

merge_names()

For multi-segment nets, select a common name for all the segments.

create_network()

Create a network from a single net.

__and__(*obj*)

Attach a net and another part/pin/net in serial.

__rand__(*obj*)

Attach a net and another part/pin/net in serial.

__or__(*obj*)

Attach a net and another part/pin/net in parallel.

__ror__(*obj*)

Attach a net and another part/pin/net in parallel.

generate_netlist_net(*tool=None*)

Generate the net information for inclusion in a netlist.

Parameters

tool – The format for the netlist file (e.g., KICAD).

generate_xml_net(*tool=None*)

Generate the net information for inclusion in an XML file.

Parameters

tool – The format for the XML file (e.g., KICAD).

ERC(**args*, ***kwargs*)

Run class-wide and local ERC functions on this net.

__str__()

Return a list of the pins on this net as a string.

__repr__()

Return a list of the pins on this net as a string.

__len__()

Return the number of pins attached to this net.

width

Return width of a Net, which is always 1.

`name`

Get, set and delete the name of this net.

When setting the net name, if another net with the same name is found, the name for this net is adjusted to make it unique.

`netclass`

Get, set and delete the net class assigned to this net.

If not net class is set, then reading the net class returns None.

You can’t overwrite the net class of a net once it’s set. You’ll have to delete it and then set it to a new value.

Also, assigning a net class of None will have no affect on the existing net class of a net.

`drive`

Get, set and delete the drive strength of this net.

The drive strength cannot be set to a value less than its current value. So as pins are added to a net, the drive strength reflects the maximum drive value of the pins currently on the net.

`valid`

`test_validity`()

`__bool__`()

Any valid Net is True

`__nonzero__`()

Any valid Net is True

`_gen_netlist_net_kicad`()

`_gen_xml_net_kicad`()

class `skidl.Net.`**`NCNet`**(*name=None*, *circuit=None*, **pins_nets_buses*, ***attribs*)

Bases: `skidl.Net.Net`

Lists of unconnected pins are stored using this Net subclass.

This is a netlist subclass used for storing lists of pins which are explicitly specified as not being connected. This means the ERC won’t flag these pins as floating, but no net connections for these pins will be placed in the netlist so there will actually be no connections to these pins in the physical circuit.

Parameters

- **name** – A string with the name of the net. If None or ‘’, then a unique net name will be assigned.
- ***pins_nets_buses** – One or more Pin, Net, or Bus objects or lists/tuples of them to be connected to this net.

Keyword Arguments

attribs – A dictionary of attributes and values to attach to the object.

`__init__`(*name=None*, *circuit=None*, **pins_nets_buses*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

`generate_netlist_net`(*tool=None*)

NO_CONNECT nets don’t generate anything for netlists.

`drive`

Get the drive strength of this net.

The drive strength is always NOCONNECT_DRIVE. It can’t be changed. The drive strength cannot be deleted.

skidl.NetPinList module

Specialized list for handling nets, pins, and buses.

class `skidl.NetPinList.`**`NetPinList`**

Bases: `list`

`__iadd__`(**nets_pins_buses*)

Implement self+=value.

`create_network`()

Create a network from a list of pins and nets.

`__and__`(*obj*)

Attach a NetPinList and another part/pin/net in serial.

`__rand__`(*obj*)

Attach a NetPinList and another part/pin/net in serial.

`__or__`(*obj*)

Attach a NetPinList and another part/pin/net in parallel.

`__ror__`(*obj*)

Attach a NetPinList and another part/pin/net in parallel.

`width`

Return width, which is the same as using the len() operator.

`aliases`

skidl.Network module

Object for for handling series and parallel networks of two-pin parts, nets, and pins.

class `skidl.Network.`**`Network`**(**objs*)

Bases: `list`

Create a Network object from a list of pins, nets, and parts.

`__init__`(**objs*)

Create a Network object from a list of pins, nets, and parts.

`__and__`(*obj*)

Combine two networks by placing them in series.

`__rand__`(*obj*)

Combine two networks by placing them in series. (Reverse-ordered operation.)

`__or__`(*obj*)

Combine two networks by placing them in parallel.

`create_network`()

Creating a network from a network just returns the original network.

skidl.Note module

Supports user-specified notes that can be attached to other SKiDL objects.

class `skidl.Note.`**`Note`**(**notes*)

Bases: `list`

Stores one or more strings as notes.

Create a note.

Parameters

notes – Either a string or an iterable containing multiple strings.

Returns

A Note object containing note strings.

`__init__`(**notes*)

Create a note.

Parameters

notes – Either a string or an iterable containing multiple strings.

Returns

A Note object containing note strings.

`__iadd__`(**notes*)

Add new notes to a Note object.

Parameters

notes – Either a string or an iterable containing multiple strings.

Returns

A Note object containing note strings.

`__str__`()

Return notes as a concatenated set of strings.

Returns

A string made up of the concatenated notes in the object joined by newlines.

skidl.Part module

Handles parts.

class `skidl.Part.`**`UnitValue`**

Bases: `object`

class `skidl.Part.`**`PinNumberSearch`**(*part*)

Bases: `object`

A class for restricting part pin indexing to only pin numbers while ignoring pin names.

`__init__`(*part*)

Initialize self. See help(type(self)) for accurate signature.

`get_pins`(**pin_ids*, ***criteria*)

`__getitem__`(**pin_ids*, ***criteria*)

`__setitem__`(*ids*, **pins_nets_buses*)

class `skidl.Part.`**`PinNameSearch`**(*part*)

Bases: `object`

A class for restricting part pin indexing to only pin names while ignoring pin numbers.

`__init__`(*part*)

Initialize self. See help(type(self)) for accurate signature.

`get_pins`(**pin_ids*, ***criteria*)

`__getitem__`(**pin_ids*, ***criteria*)

`__setitem__`(*ids*, **pins_nets_buses*)

class `skidl.Part.`**`Part`**(*lib=None*, *name=None*, *dest='NETLIST'*, *tool=None*, *connections=None*, *part_defn=None*, *circuit=None*, ***attribs*)

Bases: `skidl.baseobj.SkidlBaseObject`

A class for storing a definition of a schematic part.

`ref`

String storing the reference of a part within a schematic (e.g., ‘R5’).

`value`

String storing the part value (e.g., ‘3K3’).

`footprint`

String storing the PCB footprint associated with a part (e.g., SOIC-8).

`pins`

List of Pin objects for this part.

Parameters

- **lib** – Either a SchLib object or a schematic part library file name.
- **name** – A string with name of the part to find in the library, or to assign to the part defined by the part definition.
- **dest** – String that indicates where the part is destined for (e.g., LIBRARY).
- **tool** – The format for the library file or part definition (e.g., KICAD).
- **connections** – A dictionary with part pin names/numbers as keys and the nets to which they will be connected as values. For example: { ‘IN-‘:a_in, ‘IN+’:gnd, ‘1’:AMPED_OUTPUT, ‘14’:vcc, ‘7’:gnd }
- **part_defn** – A list of strings that define the part (usually read from a schematic library file).
- **circuit** – The Circuit object this Part belongs to.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the part. For example, manf_num=’LM4808MP-8’ would create an attribute named ‘manf_num’ for the part and assign it the value ‘LM4808MP-8’.

Raises

- *** Exception if the part library and definition are both missing.** –
- *** Exception if an unknown file format is requested.** –

`erc_list` = *[<function dflt_part_erc>]*

`__init__`(*lib=None*, *name=None*, *dest='NETLIST'*, *tool=None*, *connections=None*, *part_defn=None*, *circuit=None*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

`add_xspice_io`(*io*)

Add XSPICE I/O to the pins of a part.

classmethod `get`(*text*, *circuit=None*)

Get the part with the given text from a circuit, or return None.

Parameters

text – A text string that will be searched for in the list of parts.

Keyword Arguments

circuit – The circuit whose parts will be searched. If set to None, then the parts in the default_circuit will be searched.

Returns

A list of parts that match the text string with either their reference, name, alias, or their description.

`_find_min_max_pins`()

Return the minimum and maximum pin numbers for the part.

`parse`(*get_name_only=False*)

Create a part from its stored part definition.

Parameters

get_name_only – When true, just get the name and aliases for the part. Leave the rest unparsed.

associate_pins()

Make sure all the pins in a part have valid references to the part.

copy(*num_copies=None*, *dest='NETLIST'*, *circuit=None*, *io=None*, ***attribs*)

Make zero or more copies of this part while maintaining all pin/net connections.

Parameters

- **num_copies** – Number of copies to make of this part.
- **dest** – Indicates where the copy is destined for (e.g., NETLIST).
- **circuit** – The circuit this part should be added to.
- **io** – XSPICE I/O names.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Part copies or a single Part if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a part can be copied just by calling it like so:

```
res = Part("Device",'R')    # Get a resistor.
res_copy = res(value='1K')  # Copy the resistor and set resistance value.
```

You can also use the multiplication operator to make copies:

```
cap = Part("Device", 'C')   # Get a capacitor
caps = 10 * cap             # Make an array with 10 copies of it.
```

__call__(*num_copies=None*, *dest='NETLIST'*, *circuit=None*, *io=None*, ***attribs*)

Make zero or more copies of this part while maintaining all pin/net connections.

Parameters

- **num_copies** – Number of copies to make of this part.
- **dest** – Indicates where the copy is destined for (e.g., NETLIST).
- **circuit** – The circuit this part should be added to.
- **io** – XSPICE I/O names.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the copy.

Returns

A list of Part copies or a single Part if num_copies==1.

Raises

Exception if the requested number of copies is a non-integer or negative. –

Notes

An instance of a part can be copied just by calling it like so:

```
res = Part("Device",'R')    # Get a resistor.
res_copy = res(value='1K')  # Copy the resistor and set resistance value.
```

You can also use the multiplication operator to make copies:

```
cap = Part("Device", 'C')   # Get a capacitor
caps = 10 * cap             # Make an array with 10 copies of it.
```

__mul__(*num_copies*)

__rmul__(*num_copies*)

add_pins(**pins*)

Add one or more pins to a part.

__iadd__(**pins*)

Add one or more pins to a part.

get_pins(**pin_ids*, ***criteria*)

Return list of part pins selected by pin numbers or names.

Parameters

pin_ids – A list of strings containing pin names, numbers, regular expressions, slices, lists or tuples. If empty, then it will select all pins.

Keyword Arguments

criteria – Key/value pairs that specify attribute values the pins must have in order to be selected.

Returns

A list of pins matching the given IDs and satisfying all the criteria, or just a single Pin object if only a single match was found. Or None if no match was found.

Notes

Pins can be selected from a part by using brackets like so:

```
atmega = Part('atmel', 'ATMEGA16U2')
net = Net()
atmega[1] += net  # Connects pin 1 of chip to the net.
net += atmega['RESET']  # Connects reset pin to the net.
```

__getitem__(**pin_ids*, ***criteria*)

Return list of part pins selected by pin numbers or names.

Parameters

pin_ids – A list of strings containing pin names, numbers, regular expressions, slices, lists or tuples. If empty, then it will select all pins.

Keyword Arguments

criteria – Key/value pairs that specify attribute values the pins must have in order to be selected.

Returns

A list of pins matching the given IDs and satisfying all the criteria, or just a single Pin object if only a single match was found. Or None if no match was found.

Notes

Pins can be selected from a part by using brackets like so:

```
atmega = Part('atmel', 'ATMEGA16U2')
net = Net()
atmega[1] += net  # Connects pin 1 of chip to the net.
net += atmega['RESET']  # Connects reset pin to the net.
```

__setitem__(*ids*, **pins_nets_buses*)

You can't assign to the pins of parts. You must use the += operator.

This method is a work-around that allows the use of the += for making connections to pins while prohibiting direct assignment. Python processes something like my_part['GND'] += gnd as follows:

```
1. Part.__getitem__ is called with 'GND' as the index. This
   returns a single Pin or a NetPinList.
2. The Pin.__iadd__ or NetPinList.__iadd__ method is passed
   the thing to connect to the pin (gnd in this case). This method
   makes the actual connection to the part pin or pins. Then it
   creates an iadd_flag attribute in the object it returns.
3. Finally, Part.__setitem__ is called. If the iadd_flag attribute
   is true in the passed argument, then __setitem__ was entered
   as part of processing the += operator. If there is no
   iadd_flag attribute, then __setitem__ was entered as a result
   of using a direct assignment, which is not allowed.
```

__iter__()

Return an iterator for stepping thru individual pins of the part.

is_connected()

Return T/F depending upon whether a part is connected in a netlist.

If a part has pins but none of them are connected to nets, then this method will return False. Otherwise, it will return True even if the part has no pins (which can be the case for mechanical parts, silkscreen logos, or other non-electrical schematic elements).

is_movable()

Return T/F if the part can be moved from one circuit into another.

This method returns true if:

1. the part is not in a circuit, or
2. the part has pins but none of them are connected to nets, or
3. the part has no pins (which can be the case for mechanical parts, silkscreen logos, or other non-electrical schematic elements).

set_pin_alias(*alias*, **pin_ids*, ***criteria*)

Set the alias for a part pin.

Parameters

- **alias** – The alias for the pin.
- **pin_ids** – A list of strings containing pin names, numbers, regular expressions, slices, lists or tuples.

Keyword Arguments

criteria – Key/value pairs that specify attribute values the pin must have in order to be selected.

Returns

Nothing.

make_unit(*label*, **pin_ids*, ***criteria*)

Create a PartUnit from a set of pins in a Part object.

Parts can be organized into smaller pieces called PartUnits. A PartUnit acts like a Part but contains only a subset of the pins of the Part.

Parameters

- **label** – The label used to identify the PartUnit.
- **pin_ids** – A list of strings containing pin names, numbers, regular expressions, slices, lists or tuples.

Keyword Arguments

criteria – Key/value pairs that specify attribute values the pin must have in order to be selected.

Returns

The PartUnit.

create_network()

Create a network from the pins of a part.

`__and__`(*obj*)

Attach a part and another part/pin/net in serial.

`__rand__`(*obj*)

Attach a part and another part/pin/net in serial.

`__or__`(*obj*)

Attach a part and another part/pin/net in parallel.

`__ror__`(*obj*)

Attach a part and another part/pin/net in parallel.

`_get_fields`()

Return a list of component field names.

`generate_netlist_component`(*tool=None*)

Generate the part information for inclusion in a netlist.

Parameters

tool – The format for the netlist file (e.g., KICAD).

`generate_xml_component`(*tool=None*)

Generate the part information for inclusion in an XML file.

Parameters

tool – The format for the XML file (e.g., KICAD).

`ERC`(**args*, ***kwargs*)

Run class-wide and local ERC functions on this part.

`erc_desc`()

Create description of part for ERC and other error reporting.

`__str__`()

Return a description of the pins on this part as a string.

`__repr__`()

Return a description of the pins on this part as a string.

`export`()

Return a string to recreate a Part object.

`ref`

Get, set and delete the part reference.

When setting the part reference, if another part with the same reference is found, the reference for this part is adjusted to make it unique.

`value`

Get, set and delete the part value.

`foot`

Get, set and delete the part footprint.

`__bool__`()

Any valid Part is True

`__nonzero__`()

Any valid Part is True

`__len__`()

Return the number of pins in this part.

`_gen_netlist_comp_kicad`()

_gen_xml_comp_kicad()

_parse_lib_part_kicad(*get_name_only=False*)

Create a Part using a part definition from a KiCad schematic library.

This method was written based on the code from https://github.com/KiCad/kicad-library-utils/tree/master/schlib. It's covered by GPL3.

Parameters

- **part_defn** – A list of strings that define the part (usually read from a schematic library file). Can also be None.
- **get_name_only** – If true, scan the part definition until the name and aliases are found. The rest of the definition will be parsed if the part is actually used.

_parse_lib_part_skidl(*get_name_only=False*)

Create a Part using a part definition from a SKiDL library.

_parse_lib_part_spice(*get_name_only=False*)

Create a Part using a part definition from a SPICE library.

class skidl.Part.**SkidlPart**(*lib=None*, *name=None*, *dest='TEMPLATE'*, *tool='skidl'*, *connections=None*, ***attribs*)

Bases: skidl.Part.Part

A class for storing a SKiDL definition of a schematic part. It's identical to its Part superclass except:

- The tool defaults to SKIDL.
- The destination defaults to TEMPLATE so that it's easier to start

  a part and then add pins to it without it being added to the netlist.

SKIDL = *'skidl'*

TEMPLATE = *'TEMPLATE'*

__init__(*lib=None*, *name=None*, *dest='TEMPLATE'*, *tool='skidl'*, *connections=None*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

class skidl.Part.**PartUnit**(*part*, **pin_ids*, ***criteria*)

Bases: skidl.Part.Part

Create a PartUnit from a set of pins in a Part object.

Parts can be organized into smaller pieces called PartUnits. A PartUnit acts like a Part but contains only a subset of the pins of the Part.

Parameters

- **part** – This is the parent Part whose pins the PartUnit is built from.
- **pin_ids** – A list of strings containing pin names, numbers, regular expressions, slices, lists or tuples. If empty, it will match *every* pin of the part.

Keyword Arguments

criteria – Key/value pairs that specify attribute values the pin must have in order to be selected.

Examples

This will return unit 1 from a part:

```
lm358 = Part('linear','lm358')
lm358a = PartUnit(lm358, unit=1)
```

Or you can specify the pins directly:

```
lm358a = PartUnit(lm358, 1, 2, 3)
```

`__init__`(*part*, **pin_ids*, ***criteria*)

Initialize self. See help(type(self)) for accurate signature.

`add_pins_from_parent`(**pin_ids*, ***criteria*)

Add selected pins from the parent to the part unit.

skidl.Pin module

Handles part pins.

class `skidl.Pin.Pin`(***attribs*)

Bases: `skidl.baseobj.SkidlBaseObject`

A class for storing data about pins for a part.

Parameters

attribs – Key/value pairs of attributes to add to the library.

`nets`

The electrical nets this pin is connected to (can be >1).

`part`

Link to the Part object this pin belongs to.

`func`

Pin function such as PinType.types.INPUT.

`do_erc`

When false, the pin is not checked for ERC violations.

class `types`

Bases: `enum.IntEnum`

An enumeration.

`BIDIR` = *3*

`INPUT` = *1*

`NOCONNECT` = *13*

`OPENCOLL` = *9*

`OPENEMIT` = *10*

`OUTPUT` = *2*

`PASSIVE` = *5*

`PULLDN` = *12*

`PULLUP` = *11*

`PWRIN` = *7*

`PWROUT` = *8*

`TRISTATE` = *4*

`UNSPEC` = *6*

classmethod `add_type`(**pin_types*)

Add new pin type identifiers to the list of pin types.

Parameters

pin_types – Strings identifying zero or more pin types.

class `drives`

Bases: `enum.IntEnum`

An enumeration.

NOCONNECT = *1*

NONE = *2*

ONESIDE = *5*

PASSIVE = *3*

POWER = *8*

PULLUPDN = *4*

PUSHPULL = *7*

TRISTATE = *6*

pin_info = *{<types.INPUT: 1>: {'drive': <drives.NONE: 2>, 'func_str': 'INPUT', 'function': 'INPUT', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.PASSIVE: 3>}, <types.OUTPUT: 2>: {'drive': <drives.PUSHPULL: 7>, 'func_str': 'OUTPUT', 'function': 'OUTPUT', 'max_rcv': <drives.PASSIVE: 3>, 'min_rcv': <drives.NONE: 2>}, <types.BIDIR: 3>: {'drive': <drives.TRISTATE: 6>, 'func_str': 'BIDIR', 'function': 'BIDIRECTIONAL', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.NONE: 2>}, <types.TRISTATE: 4>: {'drive': <drives.TRISTATE: 6>, 'func_str': 'TRISTATE', 'function': 'TRISTATE', 'max_rcv': <drives.TRISTATE: 6>, 'min_rcv': <drives.NONE: 2>}, <types.PASSIVE: 5>: {'drive': <drives.PASSIVE: 3>, 'func_str': 'PASSIVE', 'function': 'PASSIVE', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.NONE: 2>}, <types.UNSPEC: 6>: {'drive': <drives.NONE: 2>, 'func_str': 'UNSPEC', 'function': 'UNSPECIFIED', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.NONE: 2>}, <types.PWRIN: 7>: {'drive': <drives.NONE: 2>, 'func_str': 'PWRIN', 'function': 'POWER-IN', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.POWER: 8>}, <types.PWROUT: 8>: {'drive': <drives.POWER: 8>, 'func_str': 'PWROUT', 'function': 'POWER-OUT', 'max_rcv': <drives.PASSIVE: 3>, 'min_rcv': <drives.NONE: 2>}, <types.OPENCOLL: 9>: {'drive': <drives.ONESIDE: 5>, 'func_str': 'OPENCOLL', 'function': 'OPEN-COLLECTOR', 'max_rcv': <drives.TRISTATE: 6>, 'min_rcv': <drives.NONE: 2>}, <types.OPENEMIT: 10>: {'drive': <drives.ONESIDE: 5>, 'func_str': 'OPENEMIT', 'function': 'OPEN-EMITTER', 'max_rcv': <drives.TRISTATE: 6>, 'min_rcv': <drives.NONE: 2>}, <types.PULLUP: 11>: {'drive': <drives.PULLUPDN: 4>, 'func_str': 'PULLUP', 'function': 'PULLUP', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.NONE: 2>}, <types.PULLDN: 12>: {'drive': <drives.PULLUPDN: 4>, 'func_str': 'PULLDN', 'function': 'PULLDN', 'max_rcv': <drives.POWER: 8>, 'min_rcv': <drives.NONE: 2>}, <types.NOCONNECT: 13>: {'drive': <drives.NOCONNECT: 1>, 'func_str': 'NOCONNECT', 'function': 'NO-CONNECT', 'max_rcv': <drives.NOCONNECT: 1>, 'min_rcv': <drives.NOCONNECT: 1>}}*

__init__(***attribs*)

Initialize self. See help(type(self)) for accurate signature.

copy(*num_copies=None*, ***attribs*)

Return copy or list of copies of a pin including any net connection.

Parameters

num_copies – Number of copies to make of pin.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the pin.

Notes

An instance of a pin can be copied just by calling it like so:

```
p = Pin()     # Create a pin.
p_copy = p()  # This is a copy of the pin.
```

__call__(*num_copies=None*, ***attribs*)

Return copy or list of copies of a pin including any net connection.

Parameters

num_copies – Number of copies to make of pin.

Keyword Arguments

attribs – Name/value pairs for setting attributes for the pin.

Notes

An instance of a pin can be copied just by calling it like so:

```
p = Pin()      # Create a pin.
p_copy = p()   # This is a copy of the pin.
```

__mul__(*num_copies*)

__rmul__(*num_copies*)

__getitem__(**ids*)

Return the pin if the indices resolve to a single index of 0.

Parameters

ids – A list of indices. These can be individual numbers, net names, nested lists, or slices.

Returns

The pin, otherwise None or raises an Exception.

__setitem__(*ids*, **pins_nets_buses*)

You can't assign to Pins. You must use the += operator.

This method is a work-around that allows the use of the += for making connections to pins while prohibiting direct assignment. Python processes something like net[0] += Net() as follows:

```
1. Pin.__getitem__ is called with '0' as the index. This
   returns a single Pin.
2. The Pin.__iadd__ method is passed the pin and
   the thing to connect to it (a Net in this case). This
   method makes the actual connection to the net. Then
   it creates an iadd_flag attribute in the object it returns.
3. Finally, Pin.__setitem__ is called. If the iadd_flag attribute
   is true in the passed argument, then __setitem__ was entered
   as part of processing the += operator. If there is no
   iadd_flag attribute, then __setitem__ was entered as a result
   of using a direct assignment, which is not allowed.
```

__iter__()

Return an iterator for stepping through the pin.

is_connected()

Return true if a pin is connected to a net (but not a no-connect net).

is_attached(*pin_net_bus*)

Return true if this pin is attached to the given pin, net or bus.

connect(**pins_nets_buses*)

Return the pin after connecting it to one or more nets or pins.

Parameters

pins_nets_buses – One or more Pin, Net or Bus objects or lists/tuples of them.

Returns

The updated pin with the new connections.

Notes

You can connect nets or pins to a pin like so:

```
p = Pin()      # Create a pin.
n = Net()      # Create a net.
p += net       # Connect the net to the pin.
```

`__iadd__`(**pins_nets_buses*)

Return the pin after connecting it to one or more nets or pins.

Parameters

pins_nets_buses – One or more Pin, Net or Bus objects or lists/tuples of them.

Returns

The updated pin with the new connections.

Notes

You can connect nets or pins to a pin like so:

```
p = Pin()      # Create a pin.
n = Net()      # Create a net.
p += net       # Connect the net to the pin.
```

`disconnect`()

Disconnect this pin from all nets.

`get_nets`()

Return a list containing the Net objects connected to this pin.

`get_pins`()

Return a list containing this pin.

`create_network`()

Create a network from a single pin.

`__and__`(*obj*)

Attach a pin and another part/pin/net in serial.

`__rand__`(*obj*)

Attach a pin and another part/pin/net in serial.

`__or__`(*obj*)

Attach a pin and another part/pin/net in parallel.

`__ror__`(*obj*)

Attach a pin and another part/pin/net in parallel.

`chk_conflict`(*other_pin*)

Check for electrical rule conflicts between this pin and another.

`erc_desc`()

Return a string describing this pin for ERC.

`get_pin_info`()

`__str__`()

Return a description of this pin as a string.

`__repr__`()

Return a description of this pin as a string.

`export`()

Return a string to recreate a Pin object.

`net`

Return one of the nets the pin is connected to.

`width`

Return width of a Pin, which is always 1.

`drive`

Get, set and delete the drive strength of this pin.

`__bool__`()

Any valid Pin is True.

`__nonzero__`()

Any valid Pin is True.

`BIDIR` = *3*

`INPUT` = *1*

`NOCONNECT` = *13*

`OPENCOLL` = *9*

`OPENEMIT` = *10*

`OUTPUT` = *2*

`PASSIVE` = *5*

`PULLDN` = *12*

`PULLUP` = *11*

`PWRIN` = *7*

`PWROUT` = *8*

`TRISTATE` = *4*

`UNSPEC` = *6*

class `skidl.Pin.`**`PhantomPin`**(***attribs*)

Bases: `skidl.Pin.Pin`

A pin type that exists solely to tie two pinless nets together. It will not participate in generating any netlists.

`__init__`(***attribs*)

Initialize self. See help(type(self)) for accurate signature.

class `skidl.Pin.`**`PinList`**(*num*, *name*, *part*)

Bases: `list`

A list of Pin objects that's meant to look something like a Pin to a Part. This is used for vector I/O of XSPICE parts.

`__init__`(*num*, *name*, *part*)

Initialize self. See help(type(self)) for accurate signature.

`__getitem__`(*i*)

Get a Pin from the list. Add Pin objects to the list if they don't exist.

`copy`()

Return a copy of a PinList for use when a Part is copied.

`disconnect`()

Disconnect all the pins in the list.

skidl.SchLib module

Handles schematic libraries for various ECAD tools.

class `skidl.SchLib.`**`SchLib`**(*filename=None*, *tool=None*, ***attribs*)

Bases: `object`

A class for storing parts from a schematic component library file.

`filename`

The name of the file from which the parts were read.

`parts`

The list of parts (composed of Part objects).

Parameters

- **filename** – The name of the library file.
- **tool** – The format of the library file (e.g., KICAD).

Keyword Arguments

attribs – Key/value pairs of attributes to add to the library.

Load the parts from a library file.

`_cache` = *{}*

`__init__`(*filename=None*, *tool=None*, ***attribs*)

Load the parts from a library file.

classmethod `reset`()

Clear the cache of processed library files.

`add_parts`(**parts*)

Add one or more parts to a library.

`__iadd__`(**parts*)

Add one or more parts to a library.

`get_parts`(*use_backup_lib=True*, ***criteria*)

Return parts from a library that match *all* the given criteria.

Keyword Arguments

criteria – One or more keyword-argument pairs. The keyword specifies the attribute name while the argument contains the desired value of the attribute.

Returns

A single Part or a list of Parts that match all the criteria.

`get_part_by_name`(*name*, *allow_multiples=False*, *silent=False*, *get_name_only=False*)

Return a Part with the given name or alias from the part list.

Parameters

- **name** – The part name or alias to search for in the library.
- **allow_multiples** – If true, return a list of parts matching the name. If false, return only the first matching part and issue a warning if there were more than one.
- **silent** – If true, don't issue errors or warnings.

Returns

A single Part or a list of Parts that match all the criteria.

`__getitem__`(*name*, *allow_multiples=False*, *silent=False*, *get_name_only=False*)

Return a Part with the given name or alias from the part list.

Parameters

- **name** – The part name or alias to search for in the library.
- **allow_multiples** – If true, return a list of parts matching the name. If false, return only the first matching part and issue a warning if there were more than one.
- **silent** – If true, don't issue errors or warnings.

Returns

A single Part or a list of Parts that match all the criteria.

`__str__`()

Return a list of the part names in this library as a string.

`__repr__`()

Return a list of the part names in this library as a string.

`export`(*libname*, *file_=None*, *tool=None*)

Export a library into a file.

Parameters

- **libname** – A string containing the name of the library.
- **file** – The file the library will be exported to. It can either be a file object or a string or None. If None, the file will be the same as the library name with the library suffix appended.
- **tool** – The CAD tool library format to be used. Currently, this can only be SKIDL.

`__len__`()

Return number of parts in library.

`_load_sch_lib_kicad`(*filename=None*, *lib_search_paths_=None*)

Load the parts from a KiCad schematic library file.

Parameters

filename – The name of the KiCad schematic library file.

`_load_sch_lib_skidl`(*filename=None*, *lib_search_paths_=None*)

Load the parts from a SKiDL schematic library file.

Parameters

filename – The name of the SKiDL schematic library file.

`_load_sch_lib_spice`(*filename=None*, *lib_search_paths_=None*)

Load the .subckt I/O from a SPICE library file.

Parameters

- **filename** – The name of the SPICE library file.
- **lib_search_paths** – List of directories to search for the file.

skidl.baseobj module

Base object for Circuit, Interface, Part, Net, Bus, Pin objects.

class `skidl.baseobj.`**`SkidlBaseObject`**

Bases: `object`

`__init__`()

Initialize self. See help(type(self)) for accurate signature.

`aliases`

`notes`

skidl.defines module

Definitions used everywhere.

`skidl.defines.`**`set_net_bus_prefixes`**(*net*, *bus*)

skidl.erc module

ERC functions for Circuit, Part, Pin, Net, Bus, Interface objects.

`skidl.erc.`**`dflt_circuit_erc`**(*circuit*)

Do an electrical rules check on a circuit.

`skidl.erc.`**`dflt_part_erc`**(*part*)

Do an electrical rules check on a part in the schematic.

`skidl.erc.`**`dflt_net_erc`**(*net*)

Do electrical rules check on a net in the schematic.

skidl.namedlist module

List of named objects for Interface, Part, Net, Bus, Pin objects. These lists are actually dictionaries so an object with a given name can be accessed very quickly without searching through all the objects.

class `skidl.namedlist.`**`NamedList`**(**args*, ***kwargs*)

Bases: `dict`

`__init__`(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`add`(*item*)

`sub`(*item*)

`__iadd__`(*item*)

`__isub__`(*item*)

skidl.netclass module

Class for PCBNEW net classes.

class `skidl.netclass.`**`NetClass`**(*name*, ***attribs*)

Bases: `object`

`__init__`(*name*, ***attribs*)

Initialize self. See help(type(self)) for accurate signature.

skidl.netlist_to_skidl module

Convert a netlist into an equivalent SKiDL program.

`skidl.netlist_to_skidl.`**`netlist_to_skidl`**(*netlist_src*)

skidl.netlist_to_skidl_main module

Command-line program to convert a netlist into an equivalent SKiDL program.

`skidl.netlist_to_skidl_main.`**`main`**()

skidl.part_query module

Functions for finding/displaying parts and footprints.

`skidl.part_query.`**`parse_search_terms`**(*terms*)

Return a regular expression for a sequence of search terms.

Substitute a zero-width lookahead assertion (?=) for each term. Thus, the "abc def" would become "(?=.*(abc))(?=.*(def))" and would match any string containing both "abc" and "def". Or "abc (def|ghi)" would become "(?=.*(abc))((?=.*(def|ghi))" and would match any string containing "abc" and "def" or "ghi". Quoted terms can be used for phrases containing whitespace.

`skidl.part_query.`**`search_parts_iter`**(*terms*, *tool=None*)

Return a list of (lib, part) sequences that match a regex term.

`skidl.part_query.`**`search_parts`**(*terms*, *tool=None*)

Print a list of parts with the regex terms within their name, alias, description or keywords.

`skidl.part_query.`**`show_part`**(*lib*, *part_name*, *tool=None*)

Print the I/O pins for a given part in a library.

Parameters

- **lib** – Either a SchLib object or the name of a library.
- **part_name** – The name of the part in the library.
- **tool** – The ECAD tool format for the library.

Returns

A Part object.

class `skidl.part_query.`**`FootprintCache`**(**args*, ***kwargs*)

Bases: `dict`

Dict for storing footprints from all directories.

`__init__`(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`reset`()

`load`(*fp_tbl_filename*)

Load cache with footprints from libraries in fp-lib-table file.

`skidl.part_query.`**`search_footprints_iter`**(*terms*, *tool=None*)

Return a list of (lib, footprint) sequences that match a regex term.

`skidl.part_query.`**`search_footprints`**(*terms*, *tool=None*)

Print a list of footprints with the regex term within their description/tags.

`skidl.part_query.`**`show_footprint`**(*lib*, *module_name*, *tool=None*)

Print the pads for a given module in a library.

Parameters

- **lib** – The name of a library.
- **module_name** – The name of the footprint in the library.
- **tool** – The ECAD tool format for the library.

Returns

A Part object.

`skidl.part_query.`**`search`**(*terms*, *tool=None*)

Print a list of parts with the regex terms within their name, alias, description or keywords.

`skidl.part_query.`**`show`**(*lib*, *part_name*, *tool=None*)

Print the I/O pins for a given part in a library.

Parameters

- **lib** – Either a SchLib object or the name of a library.
- **part_name** – The name of the part in the library.
- **tool** – The ECAD tool format for the library.

Returns

A Part object.

skidl.pckg_info module

skidl.py_2_3 module

Some definitions to make stuff work with both Python 2 & 3.

skidl.pyspice module

Import this file to reconfigure SKiDL for doing SPICE simulations.

skidl.skidl module

class `skidl.skidl.`**`SkidlCfg`**(**dirs*)

Bases: `dict`

Class for holding SKiDL configuration.

`CFG_FILE_NAME` = *'.skidlcfg'*

`__init__`(**dirs*)

Initialize self. See help(type(self)) for accurate signature.

`load`(**dirs*)

Load SKiDL configuration from JSON files in given dirs.

`store`(*dir='.'*)

Store SKiDL configuration as JSON in directory as .skidlcfg file.

`skidl.skidl.`**`get_kicad_lib_tbl_dir`**()

Get the path to where the global fp-lib-table file is found.

`skidl.skidl.`**`invalidate_footprint_cache`**(*self*, *k*, *v*)

`skidl.skidl.`**`set_default_tool`**(*tool*)

Set the ECAD tool that will be used by default.

`skidl.skidl.`**`get_default_tool`**()

`skidl.skidl.`**`set_query_backup_lib`**(*val*)

Set the boolean that controls searching for the backup library.

`skidl.skidl.`**`get_query_backup_lib`**()

`skidl.skidl.`**`set_backup_lib`**(*lib*)

Set the backup library.

`skidl.skidl.`**`get_backup_lib`**()

skidl.utilities module

Utility functions used by the rest of SKiDL.

`skidl.utilities.`**`norecurse`**(*f*)

Decorator that keeps a function from recursively calling itself.

Parameters

f (*function*) –

`skidl.utilities.`**`natural_sort_key`**(*s*, *_nsre=re.compile('([0-9]+)')*)

Sorting function for pin numbers or names.

class `skidl.utilities.`**`CountCalls`**(*func*)

Bases: `object`

Decorator for counting the number of times a function is called.

This is used for counting errors and warnings passed to logging functions, making it easy to track if and how many errors/warnings were issued.

`__init__`(*func*)

Initialize self. See help(type(self)) for accurate signature.

`__call__`(**args*, ***kwargs*)

Call self as a function.

`reset`()

class `skidl.utilities.`**`TriggerDict`**(**args*, ***kwargs*)

Bases: `dict`

This dict triggers a function when one of its entries changes.

`__init__`(**args*, ***kwargs*)

Initialize self. See help(type(self)) for accurate signature.

`__setitem__`(*k*, *v*)

Set self[key] to value.

`skidl.utilities.`**`scriptinfo`**()

Returns a dictionary with information about the running top level Python script: ———————————————————————————— dir: directory containing script or compiled executable name: name of script or executable source: name of source code file ———————————————————————————— *name* and *source* are identical if and only if running interpreted code. When running code compiled by *py2exe* or *cx_freeze*, *source* contains the name of the originating Python script. If compiled by PyInstaller, *source* contains no meaningful information.

Downloaded from: http://code.activestate.com/recipes/579018-python-determine-name-and-directory-of-the-top-lev/

`skidl.utilities.`**`get_script_name`**()

Return the name of the top-level script.

`skidl.utilities.`**`create_logger`**(*title*, *log_msg_id=''*, *log_file_suffix='.log'*)

Create a logger, usually for run-time errors or ERC violations.

`skidl.utilities.`**`get_skidl_trace`**()

Return a string containing the source line trace where a SKiDL object was instantiated.

`skidl.utilities.`**`is_binary_file`**(*filename*)

Return true if a file contains binary (non-text) characters.

`skidl.utilities.`**`merge_dicts`**(*dct*, *merge_dct*)

Dict merge that recurses through both dicts and updates keys.

Parameters

- **dct** – The dict that will be updated.
- **merge_dct** – The dict whose values will be inserted into dct.

Returns

Nothing.

`skidl.utilities.`**`find_and_open_file`**(*filename*, *paths=None*, *ext=None*, *allow_failure=False*, *exclude_binary=False*, *descend=0*)

Search for a file in list of paths, open it and return file pointer and full file name.

Parameters

- **filename** – Base file name (e.g., "my_file").
- **paths** – List of paths to search for the file.
- **ext** – The extension for the file (e.g., "txt").
- **allow_failure** – If false, failure to find file raises and exception.
- **exclude_binary** – If true, skip files that contain binary data.
- **descend** – If 0, don't search lower-level directories. If positive, search that many levels down for the file. If negative, descend into subdirectories without limit.

`skidl.utilities.`**`add_unique_attr`**(*obj*, *name*, *value*, *check_dup=False*)

Create an attribute if the attribute name isn't already used.

`skidl.utilities.`**`num_to_chars`**(*num*)

Return a string like ‘AB’ when given a number like 28.

`skidl.utilities.`**`rmv_quotes`**(*s*)

Remove starting and ending quotes from a string.

`skidl.utilities.`**`add_quotes`**(*s*)

Return string with added quotes if it contains whitespace or parens.

`skidl.utilities.`**`to_list`**(*x*)

Return x if it is already a list, or return a list if x is a scalar.

`skidl.utilities.`**`cnvt_to_var_name`**(*s*)

Convert a string to a legal Python variable name and return it.

`skidl.utilities.`**`list_or_scalar`**(*lst*)

Return a list if passed a multi-element list, otherwise return a single scalar.

Parameters

lst – Either a list or a scalar.

Returns

- A list if passed a multi-element list.
- The list element if passed a single-element list.
- None if passed an empty list.
- A scalar if passed a scalar.

`skidl.utilities.`**`flatten`**(*nested_list*)

Return a flattened list of items from a nested list.

`skidl.utilities.`**`reset_get_unique_name`**()

Reset the heaps that store previously-assigned names.

`skidl.utilities.`**`get_unique_name`**(*lst*, *attrib*, *prefix*, *initial=None*)

Return a name that doesn’t collide with another in a list.

This subroutine is used to generate unique part references (e.g., “R12”) or unique net names (e.g., “N$5”).

Parameters

- **lst** – The list of objects containing names.
- **attrib** – The attribute in each object containing the name.
- **prefix** – The prefix attached to each name.
- **initial** – The initial setting of the name (can be None or empty string).

Returns

A string containing the unique name.

`skidl.utilities.`**`fullmatch`**(*regex*, *string*, *flags=0*)

Emulate python-3.4 re.fullmatch().

`skidl.utilities.`**`filter_list`**(*lst*, ***criteria*)

Return a list of objects whose attributes match a set of criteria.

Return a list of objects extracted from a list whose attributes match a set of criteria. The match is done using regular expressions. Example: filter_list(pins, name=’io[0-9]+’, direction=’bidir’) will return all the bidirectional pins of the component that have pin names starting with ‘io’ followed by a number (e.g., ‘IO45’).

If an attribute of the lst object is a list or tuple, each entry in the list/tuple will be checked for a match. Only one entry needs to match to consider the entire attribute a match. This feature is useful when searching for objects that contain a list of aliases, such as Part objects.

Parameters

lst – The list from which objects will be extracted.

Keywords Args:

criteria: Keyword-argument pairs. The keyword specifies the attribute

name while the argument contains the desired value of the attribute. Regardless of what type the argument is, it is always compared as if it was a string. The argument can also be a regular expression that must match the entire string created from the attribute of the list object.

Returns

A list of objects whose attributes match *all* the criteria.

skidl.utilities.**expand_indices**(*slice_min*, *slice_max*, **indices*)

Expand a list of indices into a list of integers and strings.

This function takes the indices used to select pins of parts and lines of buses and returns a flat list of numbers and strings. String and integer indices are put in the list unchanged, but slices are expanded into a list of integers before entering the final list.

Parameters

- **slice_min** – The minimum possible index.
- **slice_max** – The maximum possible index (used for slice indices).
- **indices** – A list of indices made up of numbers, slices, text strings.

Returns

A linear list of all the indices made up only of numbers and strings.

skidl.utilities.**explode**(*bus_str*)

Explode a bus into its separate lines.

This function takes a bus expression like "ADDR[0:3]" and returns "ADDR0,ADDR1,ADDR2,ADDR3". It also works if the order is reversed, e.g. "ADDR[3:0]" returns "ADDR3,ADDR2,ADDR1,ADDR0". If the input string is not a valid bus expression, then the string is returned in a one-element list.

Parameters

bus_str – A string containing a bus expression like "D[0:3]".

Returns

A list of bus lines like ['D0', 'D1', 'D2', 'D3'] or a one-element list with the original input string if it's not a valid bus expression.

skidl.utilities.**find_num_copies**(***attribs*)

Return the number of copies to make based on the number of attribute values.

Keyword Arguments

attribs – Dict of Keyword/Value pairs for setting object attributes. If the value is a scalar, then the number of copies is one. If the value is a list/tuple, the number of copies is the length of the list/tuple.

Returns

The length of the longest value in the dict of attributes.

Raises

- **Exception if there are two or more list/tuple values with different** –
- **lengths that are greater than 1. (All attribute values must be scalars** –
- **or lists/tuples of the same length.)** –

skidl.utilities.**opened**(*f_or_fn*, *mode*)

Yields an opened file or file-like object.

Parameters

- **file_or_filename** – Either an already opened file or file-like object, or a filename to open.

- **mode** – The mode to open the file in.

`skidl.utilities.`**`expand_buses`**(*pins_nets_buses*)

Take list of pins, nets, and buses and return a list of only pins and nets.

`skidl.utilities.`**`exec_function_list`**(*inst*, *list_name*, **args*, ***kwargs*)

Execute class-wide and local functions on a class instance.

Parameters

- **inst** – Instance of a class.
- **list_name** – String containing the attribute name of the list of class-wide and local functions.
- **kwargs** (*args,*) – Arbitary argument lists to pass to the functions that are executed. (All functions get the same arguments.)

`skidl.utilities.`**`add_to_function_list`**(*class_or_inst*, *list_name*, *func*)

Append a function to a function list of a class or class instance.

`skidl.utilities.`**`add_erc_function`**(*class_or_inst*, *func*)

Add an ERC function to a class or class instance.

`skidl.utilities.`**`log_and_raise`**(*logger_in*, *exc_class*, *message*)

Module contents

SKiDL: A Python-Based Schematic Design Language

This module extends Python with the ability to design electronic circuits. It provides classes for working with:

- Electronic parts (`Part`),
- Collections of part terminals (`Pin`) connected via wires (`Net`), and
- Groups of related nets (`Bus`).

Using these classes, you can concisely describe the interconnection of parts using a flat or hierarchical structure. A resulting Python script outputs a netlist that can be imported into a PCB layout tool or Spice simulator. The script can also check the resulting circuitry for electrical rule violations.

Indices and tables

- Index
- Module Index
- Search Page