`openlane.common` API

Common Utilities Module

A number of common utility functions and classes used throughout the codebase.

class `openlane.common.TclUtils`

Bases: `object`

A collection of useful Tcl utilities.

static `escape(s: str) → str`

RETURNS:

If the string can be parsed by Tcl as a single token, the string is returned verbatim.

Otherwise, the string is returned in double quotes, with any unsafe characters escaped with a backslash.

PARAMETERS:

s (*str*)

RETURN TYPE:

str

static `join(ss: Iterable[str]) → str`

PARAMETERS:

ss (*Iterable[str]*) – Input list

RETURNS:

The input list converted to a Tcl-compatible list where each element is either a single token or double-quoted (i.e. interpreted by Tcl as a single element.)

RETURN TYPE:

str

latest

`openlane.common.parse_metric_modifiers(metric_name: str) → Tuple[str,` `Mapping[str, str]]`

Parses a metric name into a base and modifiers as specified in the METRICS2.1 naming convention.

PARAMETERS:

metric_name (*str*) – The name of the metric as generated by a utility.

RETURNS:

A tuple of the base part as a string, then the modifiers as a key-value mapping.

RETURN TYPE:

Tuple[str, *Mapping*[str, str]]

```
openlane.common.aggregate_metrics(input: Mapping[str, Any],
    aggregator_by_metric: Mapping[str, Tuple[int | float | Decimal,
    Callable[[Iterable[int | float | Decimal]], int | float | Decimal]]
    | Metric] | None = None) → Dict[str, Any]
```

Takes a set of metrics generated according to the METRICS2.1 naming convention.

PARAMETERS:

- **input** (*Mapping[str, Any]*) – A mapping of strings to values of metrics.
- **aggregator_by_metric** (*Mapping[str, Tuple[int | float | Decimal, Callable[[Iterable[int | float | Decimal]], int | float | Decimal]] | Metric] | None*) – A mapping of metric names to either: - A tuple of the initial accumulator and reducer to aggregate the values from all modifier metrics - A `Metric` class

RETURNS:

A tuple of the base part as a string, then the modifiers as a key-value mapping.

RETURN TYPE:

Dict[str, *Any*]

```
class openlane.common.GenericDictEncoder(*, skipkeys=False,
    ensure_ascii=True, check_circular=True, allow_nan=True,
    sort_keys=False, indent=None, separators=None, default=None)
```

Bases: `JSONEncoder`

latest

A JSON encoder for `GenericDict` objects. Also handles some types not necessarily handled by the default JSON encoder, i.e., UserString, os.PathLike, Decimals, etcetera.

It is recommended to use `GenericDict.to_json()` unless you know what you're doing.

`default(o)`

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`class openlane.common.GenericDict(copying: Mapping[KT, VT] | None = None, /, overrides: Mapping[KT, VT] | None = None)`

Bases: `Mapping`[`KT`, `VT`]

latest

A dictionary with generic keys and values that is compatible with Python 3.8.

PARAMETERS:

- **copying** (*Mapping[KT, VT] | None*) – A base Mapping object to copy values from.
- **overrides** (*Mapping[KT, VT] | None*) – Another mapping object to override the value from *copying* with.

`pop(key: KT, /) → VT`

PARAMETERS:

key (*KT*) – The key to pop the value for.

RETURNS:

The value for key. Raises `IndexError` if the key does not exist. The key/value pair is then deleted from the dictionary.

RETURN TYPE:

VT

`copy() → T`

Convenience replacement for *object.__class__(object)*, which would create a copy of the `GenericDict` object.

RETURNS:

The copy

PARAMETERS:

self (*T*)

RETURN TYPE:

T

`to_raw_dict() → dict`

RETURNS:

A copy of the underlying Python built-in `dict` for this class.

RETURN TYPE:

dict

`get_encoder() → Type[GenericDictEncoder]`

latest

RETURNS:

A JSON encoder handling GenericDict objects.

RETURN TYPE:

Type[*GenericDictEncoder*]

`keys()`

RETURNS:

A set-like object providing a view of the keys of the GenericDict object.

`values()`

RETURNS:

A set-like object providing a view of the values of the GenericDict object.

`items() → ItemsView[KT, VT]`

RETURNS:

A set-like object providing a view of the GenericDict object as (key, value) tuples.

RETURN TYPE:

ItemsView[*KT*, *VT*]

`dumps(**kwargs) → str`

PARAMETERS:

kwargs – Passed to `json.dumps`.

RETURNS:

A JSON string representing the the GenericDict object.

RETURN TYPE:

str

`check(key: KT, /) → Tuple[KT | None, VT | None]`

latest

Checks if a key exists and returns a tuple in the form `(key, value)`.

PARAMETERS:

key (*KT*) – The key in question

RETURNS:

If the key does not exist, the value of `key` will be `None` and so will `value`. If the key exists, `key` will be the key being checked for existence and `value` will be the value assigned to said key in the GenericDict object.

Do note `None` is a valid value for some keys, so simply checking if the second element `is not None` is insufficient to check whether a key exists.

RETURN TYPE:

Tuple[*KT* | None, *VT* | None]

`update(incoming: Mapping[KT, VT])`

A convenience function to update multiple values in the GenericDict object at the same time. :param incoming: The values to update

PARAMETERS:

incoming (*Mapping[KT, VT]*)

`update_reorder(incoming: Mapping[KT, VT])`

A convenience function to update multiple values in the GenericDict object at the same time. Pre-existing keys are deleted first so the values in incoming are emplaced at the end of the dictionary.

PARAMETERS:

incoming (*Mapping[KT, VT]*) – The values to update

`class openlane.common.GenericImmutableDict(copying: Mapping[KT, VT] | None = None, /, *args, **kwargs)`

Bases: `GenericDict`[`KT`, `VT`]

PARAMETERS:

copying (*Mapping[KT, VT] | None*)

latest

`openlane.common.copy_recursive(input, translator: ~typing.Callable = <function idem>)`

Copies any arbitrarily-deep nested structure of Mappings and/or Sequences.

PARAMETERS:

- **input** – The input nested structure
- **translator** (*Callable*) –

  Before an object is appended, this function will be called to process the value.

  By default, `idem()` is called.

RETURNS:

The copy.

All sequences will become built-in `list`(s) and all mappings will become built-in `dict`(s).

`openlane.common.idem(obj: T, *args, **kwargs) → T`

RETURNS:

the parameter `obj` unchanged. Useful for some lambdas.

PARAMETERS:

obj (*T*)

RETURN TYPE:

T

`openlane.common.get_openlane_root() → str`

Returns the root OpenLane folder, i.e., the folder containing the `__init__.py`.

RETURN TYPE:

str

`openlane.common.get_opdks_rev() → str`

Gets the Open_PDKs revision confirmed compatible with this version of OpenLane.

RETURN TYPE:

str

`openlane.common.slugify(value: str, lower: bool`

latest

PARAMETERS:

- **value** (*str*) – Input string
- **lower** (*bool*)

RETURNS:

The input string converted to lower case, with all characters except alphanumerics, underscores and hyphens removed, and spaces and dots converted into hyphens.

Leading and trailing whitespace is stripped.

RETURN TYPE:

str

`openlane.common.protected(method)`

A decorator to indicate protected methods.

It dynamically adds a statement to the effect in the docstring as well as setting an attribute, `protected`, to `True`, but has no other effects.

PARAMETERS:

f – Method to mark as protected

`openlane.common.final(f)`

A decorator to indicate final methods and final classes.

Use this decorator to indicate to type checkers that the decorated method cannot be overridden, and decorated class cannot be subclassed. For example:

```
class Base:
    @final
    def done(self) -> None:
        ...
class Sub(Base):
    def done(self) -> None:  # Error reported by type checker
          ...

@final
class Leaf:
    ...
class Other(Leaf):  # Error reported by type
    ...
```

latest

There is no runtime checking of these properties.

`openlane.common.mkdirp(path: str | PathLike)`

Attempts to create a directory and all of its parents.

Does not fail if the directory already exists, however, it does fail if it is unable to create any of the components and/or if the path already exists as a file.

PARAMETERS:

path (*str* | *PathLike*) – A filesystem path for the directory

`class openlane.common.zip_first(a: Iterable, b: Iterable, fillvalue: Any)`

Bases: `object`

Works like `zip_longest` if ｜a｜ > ｜b｜ and `zip` if ｜a｜ <= ｜b｜.

PARAMETERS:

- **a** (*Iterable*)
- **b** (*Iterable*)
- **fillvalue** (*Any*)

`openlane.common.format_elapsed_time(elapsed_seconds: SupportsFloat) → str`

PARAMETERS:

elapsed_seconds (*SupportsFloat*) – Total time elapsed in seconds

RETURNS:

A string in the format `{hours}:{minutes}:{seconds}:{milliseconds}`

RETURN TYPE:

str

`class openlane.common.Filter(filters: Iterable[str])`

Bases: `object`

latest

Encapsulates commonly used wildcard-based filtering functions into an object.

PARAMETERS:

filters (*Iterable[str]*) –

A list of a wildcards supporting the fnmatch spec.

The wildcards will be split into an “allow” and “deny” list based on whether the filter is prefixed with a `!`.

`get_matching_wildcards(input: str) → Generator[str, Any, None]`

PARAMETERS:

input (*str*) – An input to match wildcards against.

RETURNS:

An iterable object for *all* wildcards in the allow list accepting `input`, and *all* wildcards in the deny list rejecting `input`.

RETURN TYPE:

Generator[str, *Any*, None]

`match(input: str) → bool`

PARAMETERS:

input (*str*) – An input string to either accept or reject

RETURNS:

A boolean indicating whether the input: * Has matched at least one wildcard in the allow list * Has matched exactly 0 inputs in the deny list

RETURN TYPE:

bool

`filter(inputs: Iterable[str]) → Generator[str, Any, None]`

PARAMETERS:

inputs (*Iterable[str]*) – A series of inputs to filter according to the wildcards.

RETURNS:

An iterable object of any values in `inpu` latest
least one wildcard in the allow list * Have matched exactly 0 inputs in the deny list

RETURN TYPE:

Generator[str, *Any*, None]

`openlane.common.get_latest_file(in_path: str | PathLike, filename: str) → Path | None`

PARAMETERS:

- **in_path** (*str* | *PathLike*) – A directory to search in
- **filename** (*str*) – The final filename

RETURNS:

The latest file matching the parameters, by modification time

RETURN TYPE:

Path | None

`openlane.common.process_list_file(from_file: str | PathLike) → List[str]`

Convenience function to process text files in a `.gitignore`-style format, i.e., those where the lines may be:

- A list element
- A comment prefixed with `#`
- Blank

PARAMETERS:

from_file (*str* | *PathLike*) – The input text file.

RETURNS:

A list of the strings listed in the file, ignoring lines prefixed with a `#` and empty lines.

RETURN TYPE:

List[str]

`class openlane.common.Path(seq)`

Bases: `UserString`, `PathLike`

A Path type for OpenLane configuration variables.

latest

Basically just a string.

`exists() → bool`

A convenience method calling `os.path.exists()`

RETURN TYPE:

bool

`validate(message_on_err: str = '')`

Raises an error if the path does not exist.

PARAMETERS:

message_on_err (*str*)

`class openlane.common.ScopedFile(*, contents='')`

Bases: `Path`

Creates a temporary file that remains valid while this variable is in scope, and is deleted upon deconstruction.

The object itself is a string pointing to that file path.

PARAMETERS:

contents – The contents of the temporary file to create.

`class openlane.common.Toolbox(tmp_dir: str)`

Bases: `object`

An assisting object shared by a Flow and all its constituent Steps.

The toolbox may create artifacts that are cached to avoid constant re-creation between steps.

PARAMETERS:

tmp_dir (*str*)

`aggregate_metrics(input: Dict[str, Any], aggregator_by_metric: Dict[str, Tuple[Any, Callable[[Iterable], Any]]]) → Dict[str, Any]`

Deprecated since version 2.0.0b1: Use ‘aggregate_metrics’ from ‘openlane.common’

PARAMETERS:

- **input** (*Dict[str, Any]*)
- **aggregator_by_metric** (*Dict[str, Tuple[Any, Callable[[Iterable], Any]]]*)

RETURN TYPE:

Dict[str, *Any*]

`filter_views(config: Mapping[str, Any], views_by_corner: Mapping[str, Path | Iterable[Path]], timing_corner: str | None = None) → List[Path]`

Given a mapping from (wildcards of) corner names to views, this function enumerates all views matching either the default timing corner or an explicitly-provided override.

PARAMETERS:

- **config** (*Mapping[str, Any]*) – The configuration. Used solely to extract the default corner.
- **views_by_corner** (*Mapping[str, Path | Iterable[Path]]*) – The mapping from (wild cards) of corner names to views.
- **corner** – An explicit override for the default corner. Must be a fully qualified IPVT corner.
- **timing_corner** (*str | None*)

RETURNS:

The created list

RETURN TYPE:

List[Path]

```
get_macro_views(config: Mapping[str, Any], view: DesignFormat,
    timing_corner: str | None = None, unless_exist: None |
    DesignFormat | Sequence[DesignFormat] = None) → List[Path]
```

For `Config` objects (or similar Mappings) that have Macro information, this function gets all Macro views matching a certain `DesignFormat` for

latest

either the default timing corner or an explicitly-provided override.

PARAMETERS:

- **config** (*Mapping[str, Any]*) – The configuration.
- **view** (*DesignFormat*) – The design format to return views of.
- **timing_corner** (*str* | *None*) – An explicit override for the default corner set by the configuration.
- **corner** – An explicit override for the default corner. Must be a fully qualified IPVT corner.
- **unless_exist** (*None* | *DesignFormat* | *Sequence[DesignFormat]*) –

  If a Macro also has a view for these `DesignFormat`s, do not return a result for the requested `DesignFormat`.

  Useful for if you want to return say, Netlists if reliable LIB files do not exist.

RETURNS:

A list of the Macro views matched by the process described above.

RETURN TYPE:

List[*Path*]

```
get_timing_files_categorized(config: Mapping[str, Any],
    timing_corner: str | None = None, prioritize_nl: bool = False) →
    Tuple[str, List[Path], List[Path], List[Tuple[str, Path]]]
```

Read the Docs latest

Returns the lib files for a given configuration and timing corner.

PARAMETERS:

- **config** (*Mapping[str, Any]*) – A configuration object or a similar mapping.
- **timing_corner** (*str | None*) –

  A fully qualified IPVT corner to get SCL libs for.

  If not specified, the value for `DEFAULT_CORNER` from the SCL will be used.
- **prioritize_nl** (*bool*) –

  Do not return lib files for macros that have gate-Level Netlists and SPEF views.

  If set to `false`, only lib files are returned.

RETURNS:

A tuple of: * The name of the timing corner * A list of lib files * A list of netlists * A list of tuples of instances and SPEFs

RETURN TYPE:

Tuple[str, *List*[*Path*], *List*[*Path*], *List*[*Tuple*[str, *Path*]]]

```
get_timing_files(config: Mapping[str, Any], timing_corner: str | None
    = None, prioritize_nl: bool = False) → Tuple[str, List[str]]
```

latest

Returns the lib files for a given configuration and timing corner.

PARAMETERS:

- **config** (*Mapping[str, Any]*) – A configuration object or a similar mapping.
- **timing_corner** (*str* | *None*) –

  A fully qualified IPVT corner to get SCL libs for.

  If not specified, the value for `DEFAULT_CORNER` from the SCL will be used.

- **prioritize_nl** (*bool*) –

  Do not return lib files for macros that have gate-Level Netlists and SPEF views.

  If set to `false`, only lib files are returned.

RETURNS:

A tuple of:

- The name of the timing corner
- A heterogeneous list of files composed of: Lib files are returned as-is, Netlists are returned as-is, and SPEF files are returned in the format `{instance_name}@{spef_path}`.

  It is left up to the step or tool to process this list as they see fit.

RETURN TYPE:

Tuple[str, *List*[str]]

`remove_cells_from_lib(input_lib_files: FrozenSet[str], excluded_cells: FrozenSet[str]) → List[str]`

Creates a new lib file with some cells removed.

This function is memoized, i.e., results are cached for a specific set of inputs.

PARAMETERS:

- **input_lib_files** (*FrozenSet[str]*) – A *frozenset* of input lib files.
- **excluded_cells** (*FrozenSet[str]*) – A *frozenset* of wildcards of cells to remove from the files.

RETURNS:

A path to the lib file with the removed cells.

RETURN TYPE:

List[str]

`get_lib_voltage(input_lib: str) → Decimal | None`

Extract the voltage from the default operating conditions of a liberty file.

Returns `None` if and only if the `default_operating_conditions` key does not exist and the number of operating conditions enumerated is not exactly 1 (one).

PARAMETERS:

input_lib (*str*) – The lib file in question

RETURNS:

The voltage in question

RETURN TYPE:

Decimal | None

`class openlane.common.DRC(module: str, violations: Dict[str, Violation])`

Bases: `object`

latest

A primitive database representing DRC violations generated by a design.

PARAMETERS:

- **module** (*str*)
- **violations** (*Dict[str, Violation]*)

`classmethod from_magic(report: TextIOWrapper) → Tuple[DRC, int]`

Parses a report generated by Magic into a DRC object.

PARAMETERS:

report (*TextIOWrapper*) – A **text** input stream containing the report in question. You can pass the result of `open("drc.rpt")`, for example.

RETURNS:

A tuple of the DRC object and an int representing the number of DRC violations.

RETURN TYPE:

Tuple[*DRC*, int]

`dumps()`

RETURNS:

The DRC object as a JSON string.

`to_klayout_xml(out: BufferedIOBase)`

Converts the DRC object to a KLayout-compatible XML database.

PARAMETERS:

out (*BufferedIOBase*) – A **binary** output stream to the target XML file. You can pass the result of `open("drc.xml", "wb")`, for example.

`class openlane.common.Violation(rules: List[Tuple[str, str]], description: str, bounding_boxes: List[Tuple[decimal.Decimal, decimal.Decimal, decimal.Decimal, decimal.Decimal]] = <factory>)`

Bases: `object`

PARAMETERS:

- **rules** (*List[Tuple[str, str]]*)
- **description** (*str*)
- **bounding_boxes** (*List[Tuple[Decimal, Decimal, Decimal, Decimal]]*)

latest

`openlane.common.get_tpe() → ThreadPoolExecutor`

RETURNS:

OpenLane’s global `ThreadPoolExecutor`. This is used to run steps, so do not use them inside steps to avoid a deadlock.

RETURN TYPE:

ThreadPoolExecutor

`openlane.common.set_tpe(tpe: ThreadPoolExecutor)`

Allows replacing OpenLane’s global `ThreadPoolExecutor` with a customized one.

It will be used inside steps, so use different TPEs inside steps to avoid a deadlock.

PARAMETERS:

tpe (*ThreadPoolExecutor*) – The replacement ThreadPoolExecutor

`class openlane.common.RingBuffer(t: Type[VT], max: int)`

Bases: `Iterable`[`VT`]

A generic ring (circular) buffer that automatically pops the element at the head when full, and emplaces a new element in its place.

PARAMETERS:

- **t** (*Type[VT]*)
- **max** (*int*)

Submodule	Short Description
metrics	Metrics Module

latest