FPGAFlow Documentation

Electronic System Design Group, Technology

Jul 17, 2019

Contents

Electronic System Design Group

`FPGAFlow` is a python script to handle a complete FPGA design flow where the design files, libraries and supplementary tools are stored in a source code repository.

See *Installation* for installation notes.

CHAPTER 1

Adding Python Modules

When adding new python modules to: `$REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/Pipfile` the `pip requirements.txt` must also be updated to ensure https://fpgaflow.readthedocs.io/en/latest/index.html automatic build can operate correctly. This can be achieved by running:

```
cd $REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/
pipenv lock -r
```

and copying the output to: `$REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/requirements.txt`

1.1 License

1.1.1 Index:

Electronic System Design Group

1.1.1.1 Readme

`FPGAFlow` is a python script to handle a complete FPGA design flow where the design files, libraries and supplementary tools are stored in a source code repository.

See *Installation* for installation notes.

Adding Python Modules

When adding new python modules to: `$REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/Pipfile` the `pip requirements.txt` must also be updated to ensure https://fpgaflow.readthedocs.io/en/latest/index.html automatic build can operate correctly. This can be achieved by running:

```
cd $REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/
pipenv lock -r
```

and copying the output to: `$REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/requirements.txt`

License

Electronic System Design Group

1.1.1.2 Changelog

- : Workaround to prevent Sphinx from processing non-html files when parsing documentation generated using Doxygen.
- : Added argument to allow skipping of XML2VHDL generation. `--skip_xml2vhdl`
- #4:: Added `git` support for cloning dependencies from `git` repositories. Renamed `$REPO_USERNAME` `$SVN_USERNAME` to allow the script to differentiate between `git` and `svn` repositories. This requires an update to `~/.cshrc` before `FPGAFlow` will execute successfully.
- : Fixed path locations for documentation templates when executing `FPGAFlow` from a virtual environment.
- : Fixed running project without specifying a board.
- #3: Fixed issue where `--clean precompiled_sim` failed to clean the pre-compiled simulation libraries and `--clean fpga` failed to clean (and backup) the project build location.
- : Overrides the `vunit_args: num_threads:` value in the corresponding settings YAML file if `--open_project sim` is used. This is to prevent multiple Modelsim GUI instances from spawning when using this command line argument.
- : Tag replacement runs when setting `$LD_LIBRARY_PATH` from configuration YAML file.
- #1: Add Bamboo support to `FPGAFlow`.
- #2: Fixed issue where `build` in `$REPO_ROOT` or `${bamboo.REPO_ROOT}` prevents any source code from being added by `FPGAFlow`.
- : Fixed file paths to referenced files in packaged version.
- : Moved `xml2vhdl` usage from `sys.path` addition to using packaged version. This is now included in the `Pipfile`, assuming `xml2vhdl` is checked out under `$REPO_ROOT/tools/xml2vhdl`
- : Added support for searching for and adding HDL test-benches to corresponding libraries within the `VUnit` object.

- : projectflow.ProjectFlow Checks if test-bench simulation files exist matching the self.top_level_tb_entity[0] value from *Settings*.
- : *fpgavendor_iface.FpgaProject.create_quartus_project* adds top-level HDL test-bench to quartus project if it is successfully located on the file-system.
- : Added funcs.vunit_tb_cfg_encode to handle global encoding of VUnit test-bench configuration when running VUnit.
- : Updated *arguments* and vuint_iface.VUnitProject to handle additional arguments required by vunit_hdl>=0.3.9 which prevented VUnit from running test-cases.
- : Passing --open_gui sim and --clean all --clean sim arguments to VUnit.
- : Added generation of timestamp parameter for automatically inserting a Unix timestamp into the HDL top-level via a generic entry. This can be enabled/disabled from the board settings file using the: board_timestamp_enabled: <True|False> option.
- : Added generation of board_id parameter for automatically inserting a board_id into the HDL top-level via a generic entry. This can be configured from the board settings file using the: board_id: "00000000" option. Where the ID is a hex presentation of a 32 bit unsigned number.
- : *fpgavendor_iface.FpgaProject.create_quartus_project* now replaces all absolute paths with $env(REPO_ROOT), this means that the resulting project_gen.tcl has no references to the users working location and can be committed in the revision control system. When copying .qsf files line-by-line, blanks and commented-out lines are ignored, if the user wants comments to be copied the script will add any comment starting with more than one # comment character.
- : Added Argument parsing for --open_gui and --clean, see *Arguments*. These arguments depreciate options in the *Settings*. Converted --checkout_disabled to --checkout_enabled as the majority of user use cases are working on checked out locations.
- : qsys-generate now generates both for synthesis and simulation. When parsing the generated Modelsim generation script, the top-level simulation file is now added to the VUnit object to a HDL library matching the top-level HDL file for the IP. This matches the configuration as described in the corresponding generated .qip file.
- : Fixed issue hen parsing enabled vendor_ip in project where common wrapper file location was being to aggressive on determining wrapper file search path. Search locations are determined by: 1. wrapper file directory matches the IP name and 2. wrapper file directory matches the IP wrapper name (with _wrapper suffix stripped).
- : Fixed issue where *docflow.ProjectDoc.generate_automodule* added duplicate entries for modules found in the */build* and */dist* paths.
- : Fixed duplication of project name when building projects if board name and project name match.
- : Modified build path creation. If the build path is in boards the script will no longer duplicate vendor and tool_version in the constructed build path, as they already exist for the board being targeted.
- : Added MANIFEST.in to include all files required by FPGAflow to PyPi.
- : Restructured layout of FPGAFlow to allow for use of setuptools. Added *setup* which can be executed using: python setup.py sdist bdist_wheel from: $REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow-esdg
- : Fixed type in *vunit_iface* which caused default language to always be used when compiling standard libraries for Modelsim by invoking quartus.
- : Fixed issue with --checkout_disabled and --dev argument usage evaluating against the str "False" rather than the boolean False which prevented SVN checkouts.
- : Fixed issue where duplicate vendor_ip wrapper files were added to VUnit.

- : Fixed issue where missing `vendor_ip` locations caused the script to crash. Now they raise a critical warning.
- : Fixed issue where double slashes: `//` in `$REPO_ROOT` causes `VUnit` to fail to get sourcefile objects.
- : Added *fpgavendor_iface* module
- : Added provision to compile standard libraries for Modelsim by invoking `vivado`.
- : Added `glbl` support for `vivado` based simulations.
- : Added provision to compile standard libraries for Modelsim by invoking `quartus`. Converted *fpgavendor_iface.run_quartus_sh* and *fpgavendor_iface.run_vivado* from positional arguments to keyword arguments.
- : Added `projectflow.ProjectFlow.generate_project_name`.
- : Added support for `boards`. Boards are now handled in the same way as project dependencies.
- : Added support for mapping top-level directory names to other values. *projectmanager.ProjectDependency.top_dir_lookup*. Added documentation: *Directory Layouts*
- : Added `vendor_ip_handler` to process `vendor_ip` design files.
- : Added `board_handler` to process `board` design files, which overwrites project settings to target specific hardware.
- : Updated `vendor_ip` support. Changed method to add IP files and Wrapper Files.
- : Removed `vendor_ip_handler`. This is now handled by `projectflow` and *projectmanager*
- : Removed `board_handler`. This is now handled by `projectflow` and *projectmanager*
- : Added *fpgavendor_iface.FpgaProject.create_quartus_project* which creates a line-by-line `tcl` script to generate a `quartus` project file. HDL source files are determined from `VUnit` object which has all found `hdl` files in the project. If a top-level board file fails to include the project's top-level file the script will add the top-level project file and dependencies, but until a a project to board mechanism is developed the project to board wrapper will need to be manually added.
- : Added constraint file processing to `projectflow.ProjectFlow`.
- : Generated `project_gen.tcl` is now generated in the root project build path. Generates quartus project, and uses `rebuild_project: <True|False>` settings value to determine if existing projects should be backed up.
- : Added *Known Issues* to document problems and solutions to problems executing the script.
- : Added *fpgavendor_iface.run_ip_setup_simulation* to process `quartus` project file for extracting `vendor_ip` simulations files.
- : Added *fpgavendor_iface.run_ip_generate* to generate `quartus` vendor_ip.
- : Added `vunit_iface.compile_project_ip` to compile vendor_ip simulation files from enabled vendor_ip.
- : Added `--open_project` argument, which opens generated project in GUI.
- : Parses vendor_ip simulation compile order, derived from quartus project, and adds libraries and HDL to VUnit object.
- : Fixed issue where relative paths in *docflow* were incorrectly relative to the calling location when generating documentation for other modules.
- : Fixed bug where vendor specific HDL source code was not filtered when adding to `VUnit`.
- : Added `documentation: header:` support for modifying the common header used in generated `sphinx` documentation

- : Expanded automodule exclude term to include `template` instead of `templates`.
- : Added `XML2VHDL` dependency checking in `projectflow` to ensure that required VHDL libraries are included before proceeding.
- : Added `required_vhdl_libs` option for `XML2VHDL` VHDL library checking in settings file.
- : Added `projectflow.Xml2Vhdl` to create object for executing `xml2vhdl` script.
- : Added complete set of attributes to replicate arguments to pass to `xml2vhdl` script.
- : Moves into `xml2vhdl` script directory to execute `xml2vhdl`, returns to `projectflow` working directory when complete.
- : Added support for adding *other* FPGA vendors to `exclude_terms` based on *Configuration* file. See *supported_vendors:*.
- : Added support for getting `exclude_terms` from *Configuration* file. See *exclude_keywords:*.
- : Added `projectflow.ProjectFlow.search_paths_for_files` to recursively search paths taking into account `exclude_terms` and *EXCLUDE Files*
- : Added `vunit-hdl` for managing and processing HDL designs and simulations.
- : Added `projectflow.ProjectFlow.get_top_level_files` to find top-level HDL files in list of HDL files. If none are defined in *top_level:* the script will exit.
- : Added `arguments.Arguments._vunit_args` which adds `VUnit` Argument passing from *Settings* instead of command line. Added `args` as attribute to *`environmentsetup.ProjectEnvironment`* for use in `projectflow`.
- : Moved `projectflow.Xml2Vhdl` to `xml2vhdl_iface` to tidy up source code.
- : Moved `projectflow.VUnitProject` to *`vunit_iface`* to tidy up source code.
- : Added documentation for *vunit_args:* to *Settings*.
- : Added `no_color` argument support for `VUnit`
- : Added `sim_init_script` `VUnit` support for running GUI mode. Added `--headless` argument processing to override GUI options.
- : Fixed Release number in Documentation
- : Now using `action="store_true"` for *`arguments`* which will set optional arguments to `True` if present, `False` otherwise.
- : Added support for `bitbucket` username retrieved from system environment variable.
- : Uses `urlparse` to process URL construction for remote repository paths.
- : *`projectmanager`* now processes both `git` and `subversion` repository locations.
- : Renamed `helper` `helpers` to prevent namespace conflicts when importing `xml2vhdl`. This is a temporary workaround.
- : Added `numpy` to python virtual environment.
- : Removed obsolete `errorhandling.py` helper module.
- : Added docstrings for helper modules.
- : Grouped helper modules to tidy up documentation.
- : Updated docstring documentation for *`projectmanager`*
- : Added description of `repository_config:` to the *Settings* document.

- : Updated docstring documentation for *environmentsetup*
- : Added *simulation:* description to *Configuration* documentation page.
- : Added in depth descriptions for *Settings* documentation page.
- : Added *fpga:* descriptions in *Configuration* documentation.
- : Added LICENCE to each python module.
- : Alphabetical ordering of index and modules, based on filename not title.
- : Added `releases` support with this changelog file.
- : Updating documentation for *docflow*
- : Fixed external links in documentation, fixed readthedocs badge usage.
- : Added License link in README.
- : Completed docstring documentation for `sphinx` based functions in *docflow*
- : Completed docstring documentation for `doxygen` based functions in *docflow*
- : Started to update docstring documentation for *version* and *arguments*

Electronic System Design Group

1.1.1.3 Modules

helpers/arguments.py

BSD © 2018-2019 Science and Technology Facilities Council & contributors

`arguments.py` is a helpers module provided to define and configure the argument passing and parsing for the main modules.

class arguments.**Arguments**

Creates arguments for passing files and configuration data to the script.

A Class which configures and retrieves Arguments for use by a python script. Dictionaries retrieved from YAML files are added as attributes to this class with no sub-processing. Information including the filename of each YAML file parsed is also added as an attribute.

See *Arguments*

config
dict – the parsed dictionary of the `config.yml` passed using to script `--config`

settings
dict – the parsed dictionary of the `settings.yml` passed to script using `--settings`

checkout_enabled
bool – set `True` using optional `--checkout_enabled`, otherwise `False`

dev_flag
bool – set `True` using optional `--dev`, otherwise `False`

open_gui
str – Valid values: `'fpga'`, `'sim'`. Using optional `--open_gui`, otherwise `None`

clean
list of str – Valid values: `'all'`, `'fpga'`, `'ip'`, `'precompiled_sim'`, `'sim'`. Using optiona `clean`, otherwise: `None`

helpers/customlogging.py

`customlogging.py` is a helpers module provided to define and configure consistent logging. The logging configuration is handled via `../logging.yml` YAML file and uses `colorlog` for console output. A rolling system is used for outputting to files.

class customlogging.**LogLevelsConsts**
Logger level equivalent strings as constants, used to pass log level to helpers functions.

DEBUG
str – String matching DEBUG logger level.

INFO
str – String matching INFO logger level.

WARNING
str – String matching WARNING logger level.

CRITICAL
str – String matching CRITICAL logger level.

ERROR
str – String matching ERROR logger level.

customlogging.**config_logger**(*name*, *class_name=None*)
Allows a logger to be set up and/or configured in all modules/module classes

Parameters

- **name** (*str*) – Name of the logger.
- **class_name** (*str, optional*) – Name of the class to which the logger is associated. Default value: None

Returns A configured logger object.

Return type logging (obj)

customlogging.**constant**(*f*)
setters and getters for constants used throughout

Prevents setting of existing objects. Raises a TypeError.

Returns property

Raises `TypeError` – On setter.

customlogging.**errorexit**(*logr*)
Inserts a exit on error message in the logger output and performs a `sys.exit(-1)`

Parameters **logr** (*obj*) – A logger object to pass the exit on error message to.

Returns None

customlogging.**mand_missing**(*obj*, *field*)
Inserts a mandatory field missing error message in the logger output and calls `errorexit()`

Parameters

- **logr** (*obj*) – A logger object to pass the error message to.
- **field** (*str*) – field to report in the error message.

Returns None

customlogging.**path_missing**(*obj*, *name*, *path*)
Inserts a path missing error message in the logger output and calls errorexit()

Parameters

- **logr** (*obj*) – A logger object to pass the error message to.
- **name** (*str*) – Name of processing element which caused the error message.
- **path** (*str*) – Missing Path which caused the error message.

Returns None

customlogging.**sect_break**(*logr*, *level=<property object>*)
Inserts a section break in the logger output.

Parameters

- **logr** (*obj*) – A logger object to pass the section break to.
- **level** (*attr of LogLevelsConsts obj, optional*) – Logger level of the section break. Default value: LogLevelsConsts.INFO

Returns None

customlogging.**setup_logging**(*logr_settings='../logging.yml'*, *default_level=20*)
Setup logging configuration from a YAML file

Parameters

- **logr_settings** (*str, optional*) – Defines the relative location of the logging configuration file,
- **the root module. Default value** (*from*) – logging.yml
- **default_level** (*logging attr, optional*) – Sets the default logging level. Default value: logging.INFO

Returns None

docflow.py

docflow.py is a module provided to setup and manage the auto-documentation generation using doxygen and sphinx. Project specific values are extracted from the corresponding settings.yml YAML file and the projects structure is determined from the config.yml YAML file, each passed to the module via *arguments* (see *Arguments*).

Todo:

- Consider Moving doxygen and sphinx template values to config.yml YAML file for improved flexibility.
- Determine the name to assign to: breathe_project_name.

class docflow.**ProjectDoc**(***kwargs*)

An object describing the configuration and settings for the projects documentation generation.

Inherits: project.RepoFlow (obj): Inherited RepoFlow object. See *projectmanager.RepoFlow*.

Keyword Arguments ****include_dependencies** (*bool, optional*) – Process and include project dependencies in the generated documentation. Default value: `False`

generate_automodule(***kwargs*)

Generates `sphinx` reStructuredText auto-module directive

Generates the auto-module directive, in the form:

```
.. automodule:: module
   :members:
```

Keyword Arguments

- ****module** (*str, optional*) – The name of the module to auto-module. Default value: `False`
- ****indent_size** (*int, optional*) – Number of spaces in indent. Default value: `3`

Returns A line-by-line list of the complete `sphinx` reStructuredText auto-module directive

Return type list of str

generate_doxyfile(***kwargs*)

Generate the `doxygen .doxyfile` from project derived values and template.

Warning: `EXAMPLE_PATH` is set with fixed locations and should be modified to be defined in project `settings.yml` YAML file.

Keyword Arguments

- ****repo_root** (*str, optional*) – Full resolved path defined by `$REPO_ROOT` Default value: `''`
- ****name** (*str, optional*) – The name of the `.doxyfile` to generate. Default value: `''`
- ****doxy_input_paths** (*list of str, optional*) – List of paths to include as input sources for the generation of documentation. Default value: `list()`
- ****doxy_source_path** (*str, optional*) – Path to `doxygen` source directory which includes common documentation files (`*.md`) to include in generated documentation. Default value: `''`
- ****doxy_build_path** (*str, optional*) – Full path where to build documentation. Default value: `''`
- ****doxy_mainpage_name** (*str, optional*) – Name of the mainpage to use as `index.html` in generated HTML documentation. Default value: `'README.md'`
- ****doxy_template** (*str, optional*) – Full path of template file to use for generic `.doxyfile` configuration options. Default value: `''`
- ****doxy_extra_style_sheets** (*str, optional*) – Extra CSS style sheets used in the generation of HTML pages. Default value: `''`

- ****image_path** (*str, optional*) – Full path of images to include in generated documentation. Default value: `False`
- ****restricted_src** (*bool, optional*) – If set `True` stops source code from being published in the generated documentation. Default value: `True`
- ****project_title** (*str, optional*) – The project title used in the generated documentation. Default value: `''`
- ****project_version** (*str, optional*) – The project version number, in the form: `YYYY.NN` used in the generated documentation. Default value: `''`
- ****project_logo** (*bool, optional*) – Full path to the logo to use in the generated documentation. Default value: `False`
- ****project_language** (*str, optional*) – The language of the project being generated, used to optimise the output of the generated documentation. Default value: `''`
- ****doxypypy_script** (*str, optional*) – Full path of `doxypypy.py` for parsing `python` code in `doxygen`. `sphinx` is preferred over this option. Default value: `''`
- ****dot_path** (*str, optional*) – Full path to `dot` for using `graphviz` to generate diagrams in generated documentation. Default value: `''`

Returns

tuple containing:

- str: Full path of the generated `doxyfile`
- str: Full path of the generated `xml` to be used by `breathe` to bridge `doxygen` and `sphinx` generated documentation.

or: int: `-1` on failure.

Return type tuple

generate_doxymainpage(***kwargs*)

Generates the `doxygen` Main Page to act as link between `doxygen` and `sphinx` documentation

Parses `generated_html_path` looking for HTML files where the filename ends with `htmlpagefilter` and added the file to the generated mainpage, along with the pages title extracted from the HTML page.

Keyword Arguments

- ****doxy_mainpage_dict** (*dict, optional*) – Dictionary of mainpage settings defining the structure and configuration of the mainpage to generate. Default value: `dict()`
- ****doxy_source_path** (*str, optional*) – Full path to the `doxygen` source path. Default value: `None`
- ****generated_html_path** (*str, optional*) – Full path to the generated `doxygen` generated HTML. Default value: `None`
- ****doxy_mainpage_title** (*str, optional*) – Tile of the generated mainpage which is referenced in the generated documentation. Default value: `'Doxygen Index`
- ****htmlpagefilter** (*str, optional*) – The suffix of the HTML pages to add to the generated mainpage (excluding the file extension). Default value: `'page`

Returns Name of the generated main page

Return type str

generate_include(***kwargs*)
Generates sphinx reStructuredText include directive

Generates the include directive, in the form:

```
.. include:: <include_ref>
```

Keyword Arguments

- ****include_ref*** (*str, optional*) – The name of the reference to include. Default value: ''
- ****indent_size*** (*int, optional*) – Number of spaces in indent. Default value: 3

Returns A line-by-line list of the complete sphinx reStructuredText include directive

Return type list of str

generate_index_heading(***kwargs*)
Generates reStructuredText headings matching sphinx recommended formatting.

see: http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

Keyword Arguments

- ****heading_text*** (*str, optional*) – The heading string. Default value: False
- ****heading_char*** (*str, optional*) – A single character representing the reStructuredText heading level. Default value: #

  Valid values: #, *, =, –, ^, "

Returns A line-by-line list of the complete reStructuredText heading

Return type list of str

generate_indices_and_tables(***kwargs*)
Generates sphinx reStructuredText Indices and Tables List

Generates the TOC Tree, in the form:

```
==================
Indices and Tables:
==================

* :ref:`genindex`
* :ref:`modindex`
* :ref:`search`
```

Keyword Arguments

- ****indent_size*** (*int, optional*) – Number of spaces in indent. Default value: 3
- ****language*** (*str, optional*) – Language to determine if module index is included. Included if set to: python. Excluded if set to anything else. Default value: python

Returns A line-by-line list of the complete sphinx reStructuredText Indices and Tables

Return type list of str

generate_sphinx_index_file(***kwargs*)
Generates sphinx reStructuredText Index File

Searches for `.rst` and `.md` files in the `source_path` and adds them to the generated index file.

If a file matching the `index_name` is found it is **not** added to the generated index file.

If a html directory is found, using the `/html/` string, it is added **once** to the generated index file.

Keyword Arguments

- ****source_code_path** (*str, optional*) – Top-Level Path of Source-Code Location. Default value: `''`
- ****source_path** (*str, optional*) – Documentation source path to parse for valid files to add to index. Default value: `''`
- ****static_path** (*str, optional*) – The name of the `sphinx` Static Path. Default value: `_static`
- ****index_name** (*str, optional*) – The name of the index file to generate. Default value: `index.rst`
- ****readme_name** (*str, optional*) – The name of a valid README file. Could have `.rst` or `.md` file extension. Default value: `README`
- ****src_code_path_list** (*list of str, optional*) – List of source code paths to parse to add to Sphinx documentation. These paths are relative to `source_path`. Default value: `list()`
- ****max_depth** (*int, optional*) – The maximum depth of the toctree. Default value: `2`
- ****indent_size** (*int, optional*) – Number of spaces in indent. Default value: `3`
- ****language** (*str, optional*) – Language to determine if module index is included. Included if set to: `python`. Excluded if set to anything else. Default value: `python`
- ****header** (*str, optional*) – Common header file to use in generated documentation. Default value: `'_static/esdg_header.rst'`

Returns None

generate_sphinx_modules(***kwargs*)

Generates `sphinx` reStructuredText Module File

Searches for `.py` files in locations found in `src_code_path_list`, then passes that list to: *docflow.ProjectDoc.generate_automodule()*

Keyword Arguments

- ****source_code_path** (*str, optional*) – Top-Level Path of Source-Code Location. Default value: `''`
- ****source_path** (*str, optional*) – Documentation source path where the `<filename>.rst` should be generated. Default value: `''`
- ****filename** (*str, optional*) – The name of module file to generate. Default value: `modules.rst`
- ****src_code_path_list** (*list of str, optional*) – List of source code paths to parse to add to Sphinx documentation. Default value: `list()`
- ****indent_size** (*int, optional*) – Number of spaces in indent. Default value: `3`
- ****header** (*str, optional*) – Header file to reference at the top of the generated file. Default value: `_static/esdg_header.rst`

Returns None

generate_toctree(***kwargs*)

Generates sphinx reStructuredText Table-of-Contents Tree

Generates the TOC Tree, in the form:

```
==================
Index:
==================

..toctree::
  :maxdepth: 2

  srcs[0]
  srcs[1]
  srcs[n]
  Doxygen Index <_static/html/index.html#://>
```

Keyword Arguments

- ****max_depth** (*int, optional*) – The maximum depth of the toctree. Default value: 2
- ****indent_size** (*int, optional*) – Number of spaces in indent. Default value: 3
- ****srcs** (*list of str, optional*) – List of srcs to include in the toctree. Default value: list()
- ****reference_static_html** (*bool, optional*) – Allows the addition of a link to page external to sphinx. Default value: False
- ****static_index_text** (*str, optional*) – The human readable string for the reference_static_html link. Default value: Doxygen Index
- ****static_index_link** (*str, optional*) – The location of the reference_static_html link, relative to the sphinx top-level source path. Default value: _static/html/index.html

Returns A line-by-line list of the complete sphinx reStructuredText Table-of-Contents Tree

Return type list of str

load_settings_from_yaml(***kwargs*)

Loads a settings.yml YAML file and returns the 'documentation' key.

Keyword Arguments

- ****settings_root** (*str, optional*) – Path to the project's root settings directory. Default value: ''
- ****settings_filename** (*str, optional*) – Name of settings file to use. Default value: default_settings.yml

Returns None

preprocess_sphinxconf(***kwargs*)

Preprocessing for the sphinx conf.py File based on the current settings

Checks locations of directories and files exist and copies relevant files to documentation source directory prior to generating the configuration file.

Keyword Arguments

- ****source_path** (*str, optional*) – Documentation source path. Default value: ''
- ****src_code_path** (*str, optional*) – Top-level path of source code to add to sphinx documentation. This path will searched recursively, directories name: template will be added to a list to be used to add template files to the configuration. Default value: ''
- ****sphinx_source_path** (*str, optional*) – The path of sphinx source code to add in the generated documentation. Default value: ''
- ****sphinx_build_path** (*str, optional*) – The path where sphinx will generate the documentation. Default value: ''
- ****project_name** (*str, optional*) – The project name. Default value: ''
- ****project_version** (*str, optional*) – The projects documentation version. Default value: ''
- ****project_release** (*str, optional*) – The project release. Default value: ''
- ****project_author** (*str, optional*) – The project documentation's author. Default value: ''
- ****project_org** (*str, optional*) – The project's organisation. Default value: ''
- ****project_logo** (*str, optional*) – The name of the project logo to use in the project's documentation. Default value: False
- ****release_uri** (*str, optional*) – The URL for releases to determine the release version. Default value: False
- ****issue_uri** (*str, optional*) – The URL for releases to determine the issue reference. Default value: False
- ****releases_document_name** (*str or list of str, optional*) – The name of the Changelog used by releases. Default value: False
- ****style_sheets** (*str, optional*) – Additional Style-Sheets to use when generating HTML documentation. Default value: False
- ****breathe_project** (*str, optional*) – Breathe project to use when bridging doxygen generated documentation with sphinx. Default value: False
- ****toc_depth** (*int, optional*) – The depth of the toctree used in the generated documentation. Default value: 4
- ****image_path** (*str, optional*) – Path to directory of images to add to the generated sphinx documentation. Files in this directory will be copied to the static_path used by sphinx. Default value: False
- ****source_path** – Path to directory of source files to add to the generated sphinx documentation. Files in this directory will be copied to the static_path used by sphinx. Default value: False
- ****static_path** (*str, optional*) – Path to directory where source files will be copied for the generation of sphinx documentation. Default value: '_static'
- ****doxygen_html_path** (*str, optional*) – Path where doxygen has generated HTML documentation prior to generating the sphinx documentation. These pages will be copied to a 'html' directory in the sphinx static_path for inclusion in the generated sphinx documentation. Default value: None'

- ****sphinx_conf_template** (*str, optional*) – Relative Path, from `docflow.py`, to the `sphinx` configuration file template to append to the generated configuration file. Default value: `''`
- ****sphinx_make_template** (*str, optional*) – Relative Path, from `docflow.py`, to the `sphinx` `Makefile` file template to append to the generated configuration file. Default value: `''`
- ****language** (*str, optional*) – Language to determine if module index is included. Included if set to: `python`. Excluded if set to anything else. Default value: `python`
- ****header** (*str, optional*) – Header file to reference at the top of the generated file. Default value: `_static/esdg_header.rst`

Returns None

process_sphinx_conf_file(***kwargs*)

Generates `sphinx` `conf.py` File based on the current settings

Completes options from passed values and completes the remaining file by copying the contents from a template file.

Keyword Arguments

- ****source_path** (*str, optional*) – Documentation source path. Default value: `''`
- ****src_code_path_list** (*list of str, optional*) – List of source code paths to parse to add to `sphinx` documentation. Default value: `list()`
- ****conf_name** (*str, optional*) – The name of `sphinx` configuration file to generate. Default value: `conf.py`
- ****project_name** (*str, optional*) – The project name. Default value: `''`
- ****project_version** (*str, optional*) – The projects documentation version. Default value: `''`
- ****project_release** (*str, optional*) – The project release. Default value: `''`
- ****project_author** (*str, optional*) – The project documentation's author. Default value: `''`
- ****project_org** (*str, optional*) – The project's organisation. Default value: `''`
- ****project_logo** (*str, optional*) – The name of the project logo to use in the project's documentation. Default value: `False`
- ****release_uri** (*str, optional*) – The URL for releases to determine the release version. Default value: `False`
- ****issue_uri** (*str, optional*) – The URL for releases to determine the issue reference. Default value: `False`
- ****releases_document_name** (*str or list of str, optional*) – The name of the Changelog used by `releases`. Default value: `False`
- ****style_sheets** (*str, optional*) – Additional Style-Sheets to use when generating HTML documentation. Default value: `False`
- ****template_file** (*str, optional*) – The name of sphinx configuration template file. file extension. Default value: `''`
- ****breathe_project** (*str, optional*) – Breathe project to use when bridging doxygen generated documentation with sphinx. Default value: `False`

- ****toc_depth** (*int, optional*) – The depth of the toctree used in the generated documentation. Default value: `4`
- ****indent_size** (*int, optional*) – Number of spaces in indent. Default value: `4`

Returns None

process_sphinx_makefile(***kwargs*)
Processes a `sphinx` `Makefile` from a supplied template.

Keyword Arguments

- ****build_path** (*str, optional*) – absolute path of the `sphinx` build location. Default value: `''`
- ****make_file_template** (*list of str*) – line-by-line list of `Makefile` template contents. Default value: `''`
- ****makefile** (*str, optional*) – name of the makefile. Default value: `Makefile`

Returns None

run_doxygen(***kwargs*)
`doxygen` runner

Keyword Arguments

- ****doxyfile** (*str, optional*) – The full path of `doxygen .doxyfile` used to generate documentation. Default value: `''`
- ****doxy_bin** (*str, optional*) – The name of the bin to use to execute `doxygen` Default value: `'doxygen'`

Returns None

run_sphinx(***kwargs*)
`sphinx` runner

Keyword Arguments

- ****sphinx_source_path** (*str, optional*) – The path of `sphinx` source to generate documentation. Default value: `''`
- ****sphinx_build_path** (*str, optional*) – The path where `sphinx` will generate the documentation. Default value: `''`
- ****builder** (*str, optional*) – builder to invoke. Valid values: `html`. Default value: `html`
- ****use_make** (*bool ,optional*) – Use the *Makefile* to generate documentation instead of executing directly. Default value: `False`

Returns `0` on Success

Return type int

set_doxy_paths(***kwargs*)
Sets the `doxygen` source and build paths.

Note: The keyword argument `doxy_version` should be available on the host system and this value must be included in the *documentation: doxygen: support_tools:* within the configuration file (see *Configuration*).

Keyword Arguments

- **`**doc_root`** (*str, optional*) – Path to projects root `docs` directory. Default value: `''`
- **`**doxy_version`** (*str, optional*) – version number of `doxygen` - determines the build path. Default value: `''`
- **`**clean`** (*bool, optional*) – if set to `True`, removes all existing files from the build directory. Default value: `False`

Returns

tuple containing:

- str: Full path of `doxygen` source location.
- str: Full path of `doxygen` build location.

Return type tuple

`set_image_path`(***kwargs*)

Sets the full-path to the images directory to use in generated documentation.

Keyword Arguments

- **`**doc_root`** (*str, optional*) – Path to projects root `docs` directory. Default value: `''`
- **`**image_path`** (*str, optional*) – Relative Path, from `doc_root`, to the project's image directory. Default value: `''`

Returns

the full-path to the images directory if found on file-system.

or: bool: `False` if not found.

Return type str

`set_logo`(***kwargs*)

Sets the full-path to the logo to use in generated documentation.

Keyword Arguments

- **`**doc_root`** (*str, optional*) – Path to projects root `docs` directory. Default value: `''`
- **`**logo_path`** (*str, optional*) – Relative Path, from `doc_root`, to the project's logo directory. Default value: `''`
- **`**logo_name`** (*str, optional*) – Name of logo file to use. Default value: `''`

Returns

the full-path to the logo if found on file-system.

or: bool: `False` if not found.

Return type str

`set_source_path`(***kwargs*)

Sets the full-path to the source directory to use in generated documentation.

Keyword Arguments

- ****doc_root** (*str, optional*) – Path to projects root `docs` directory. Default value: `''`
- ****source_path** (*str, optional*) – Relative Path, from `doc_root`, to the project's source_path directory. Default value: `''`

Returns

the full-path to the source directory if found on file-system

or: bool: `False` if not found.

Return type str

set_sphinx_paths(***kwargs*)

Sets the `sphinx` source, build and static paths.

Note: The keyword argument `sphinx_version` should be available on the host system and this value must be included in the *documentation: sphinx: support_tools:* within the configuration file (see *Configuration*).

Keyword Arguments

- ****doc_root** (*str, optional*) – Path to projects root `docs` directory. Default value: `''`
- ****sphinx_version** (*str, optional*) – version number of `sphinx` to use - determines the build path. Default value: `''`
- ****sphinx_static** (*str, optional*) – `sphinx` static directory name. Where existing source files are copied to in order to be included in the generated documentation. Default value: `'_static'`
- ****clean** (*bool, optional*) – if set to `True`, removes all existing files from the build directory. Default value: `False`

Returns

tuple containing:

- str: Full path of `sphinx` source location.
- str: Full path of `sphinx` build location.
- str: Full path of `sphinx` static location.

Return type tuple

source_code_publication_msg(***kwargs*)

Sends a Critical Warning to the Console if Source Code Will be Published in the Documentation

Keyword Arguments

- ****name** (*str, optional*) – Name of project which is publishing source code in the generated documentation. Default value: `''`
- ****restricted_flag** (*bool, optional*) – Flag to determine if source-code is being published in the generated documentation. Default value: `True`

Returns None

environmentsetup.py

`environmentsetup.py` is a module provided to configure the host environment for each project, setting up environment variables which persist during the execution of the script which defined paths and license references for each tool using the the flow. Specific values are extracted from the corresponding `settings.yml` YAML file and the tool installation structure is determined from the `config.yml` YAML file, each passed to the module via *arguments* (see *Arguments*).

Todo:

- Add `breathe` configuration to *ProjectEnvironment.configure_documentation_tool()*

---

class environmentsetup.**ProjectEnvironment**(***kwargs*)

An object containing attributes defining the environment variables required for processing a project.

Specific values are extracted from the corresponding `settings.yml` YAML file and the tool installation structure is determined from the `config.yml` YAML file, each passed to the module via *arguments* (see *Arguments*).

add_environment_variable(***kwargs*)

Adds a system environment variable for the running script.

If the system environment variable already exists this will add the new value as the first value and append the old value using `':'` as a delimiter.

Keyword Arguments

- ****name** (*str*) – Name of the system environment variable to create/append. Default value: `None`
- ****value** (*str*) – Value to assign to the system environment variable. Default value: `None`
- ****overwrite** (*bool, optional*) – Default value: `False`

Returns None

check_if_supported(***kwargs*)

Checks if value in `settings.yml` is supported in `config.yml`

Checks value against a list of `supported_values` and exits with error if not found with reference to the full-path of the failing `settings.yml` YAML used.

Keyword Arguments

- ****value** (*str*) – Default value: `None`
- ****supported_values** (*list of str*) – Default value: `list()`
- ****settings_fileref** (*str*) – Full path, including filename, of settings file used. Default value: `None`

Returns None

configure_documentation_tool()

Configures and validates the system environment variables for running documentation tools.

Supports the following automatic documentation generation tools:

- `doxygen`

- `breathe`
- `sphinx`

Returns None

configure_fpga_vendor_tool()
Configures and validates the system environment variables for running the *FPGA* vendor tool.

Returns None

configure_license(***kwargs*)
Configures license environment variables.

Gets configuration from `config.yml` YAML file (see *Configuration*).

Keyword Arguments ****license_settings** (*dict*) – The license settings from configuration dictionary. Default value: `None`

Returns None

configure_mode(***kwargs*)
Configure tool execution mode.

Provides support for running the tools in 32 or 64 bit modes as well as referencing libraries not supported by the host operating system distribution.

Keyword Arguments

- ****mode** (*str*) – Default value: `None`
- ****mode_config** (*dict*) – Dictionary defining the mode of running the tool being configured. Default value: `None`
- ****mode_type** (*str*) – Default value: `sim`

Returns None

configure_modelsim_environment()
Configures and validates the system environment variables for running `modelsim`

Returns None

configure_paths(***kwargs*)
Configures system environment paths required for tool execution.

Keyword Arguments ****paths_config** (*dict*) – Dictionary defining the paths and values for the tool being configured. Default value: `None`

Returns None

get_envvar(***kwargs*)
Method to get system environment variable using `os.environ`

If the system environment variable does **not** exist *customlogging.errorexit()* is called.

Keyword Arguments ****variable_name** (*str*) – Name of the system environment variable to get from system. Default value: `None`

Returns The retrieved system environment variable.

Return type (str)

set_args_from_dict(***kwargs*)
Sets arguments from a dictionary.

Creates attributes from a dictionary based on `key` or `key[subkey]`, from the item or items in list found in `dictionary[key]` or `dictionary[key][subkey]`. If using: `from_envvars=True` the attribute will be sourced from the system environment variable.

Keyword Arguments

- **`**key`** (*`str`*) – Name of the dictionary key to process. Default value: `None`
- **`**subkey`** (*`str, optional`*) – Name of the dictionary sub-key to process. Default value: `None`
- **`**dictionary`** (*`dict`*) – Dictionary to extract `[key]` or `[key][subkey]` value from. Default value: `dict()`
- **`**from_envvars`** (*`bool, optional`*) – If `True` gets the value from the corresponding system environment variable. Default value: `False`

Returns None

`set_attr_from_path`(***kwargs*)

Sets an attribute: `name` containing a full-path, without file name.

Keyword Arguments

- **`**name`** (*`str`*) – Name of the attribute to set. Default value: `None`
- **`**path`** (*`str`*) – Full path, excluding any filenames, of reference to attribute. Default value: `` `None ``
- **`**create_if_missing`** (*`bool, optional`*) – If `True` Creates the directory structure on the file-system, using `os.makedirs()`. If the directory structure does **not** exist on the file-system. Default value: `False`
- **`**force_attr`** (*`bool, optional`*) – If `True` Creates the attribute even if the directory structure does **not** exist on the file-system. Default value: `False`

Returns None

`set_envattr`(***kwargs*)

Checks to see if the attribute already exists before setting.

Keyword Arguments **`**env`** (*`str`*) – Name of the system environment variable to check. Default value: `None`

Returns None

`set_path_environ`(***kwargs*)

Sets system environment `$path` variable.

This performs the equivalent of running:

```
setenv env path
```

If `$env` already exists it will append to the existing value:

```
setenv env path:${env}
```

Keyword Arguments

- **`**env`** (*`str`*) – Name of the system environment variable name. Default value: `None`
- **`**path`** (*`str`*) – System environment variable value. Default value: `None`
- **`**delimiter`** (*`str, optional`*) – Delimiter to use between multiple environment variable values Default value: `':'`

- ****overwrite** (*bool, optional*) – If `True` overwrite existing value instead of appending. Default value: `False`

Returns None

setattr_from_envvar(***kwargs*)
Sets attribute from system environment variable.

Gets the value from corresponding *$envs'* and sets attribute matching the system environment variable name with the value retrieved.

Keyword Arguments ****envs** (*str*) – Name of the system environment variable to process. Default value: `None`

Returns None

fpgavendor_iface.py

BSD © 2018-2019 Science and Technology Facilities Council & contributors

`fpgavendor_iface.py` is a module to handle the generation of FPGA projects and interfacing with *vunit_iface*, to include the compilation of FPGA vendor simulation modules based on tool version and FPGA device family.

class fpgavendor_iface.**FpgaProject**(***kwargs*)

add_ip(***kwargs*)

Keyword Arguments

- ****vunit_obj** (*obj*) – A `vunit_iface.VunitProject` object. Default value: `self.vu`
- ****project_file** (*str*) – Absolute path to project file.
- ****rebuild_project** (*bool, optional*) – If `True` rebuild existing project. Default value: `False`

Returns None

backup_project(***kwargs*)
Backup project using timestamp.

Appends time stamp to the project directory in the form: `_%Y%m%d%H%M%S` and uses `shutil.copytree()` to backup the project. Then cleans the original directory (using `shutil.rmtree()` and remaking the location using: `os.makedirs()`)

Keyword Arguments ****project_path** (*str*) – Absolute path to project root.

Returns None

create_quartus_project(***kwargs*)
Creates a `quartus` project using supplied settings.

Copy constraints from `.qsf` file line-by-line directly, ignoring blank and commented lines, into project file replacing any instance of `$REPO_PATH` with `$env(REPO_ROOT)`. Leaves the original constraints file untouched, which should have the same effect as having sourced the constraints file.

Keyword Arguments

- ****project_name** (*str, optional*) – Name of project file to generate. Default value: `'project'`

- ****build_path** (*str*) – Absolute path to build location for generated project
- ****target_language** (*str, optional*) – Target default language for the generated project. Default value: 'VHDL_2008'
- ****top_level_entity** (*str*) – Top-level HDL entity name.
- ****hdl_files** (*list of tuples*) – containing HDL libraries and absolute path to HDL file.
- ****tb_hdl_file** (*list of tuples*) – containing HDL libraries and absolute path to Test Bench HDL file.
- ****ip_files** (*list of str*) – List of absolute paths fof IP files to add to project.
- ****constraint_files** (*list of str*) – List of absolute paths of constraints to add to project.

Returns None

create_vivado_project(***kwargs*)
Creates a vivado project using supplied settings.

Keyword Arguments

- ****project_name** (*str, optional*) – Name of project file to generate. Default value: 'project'
- ****build_path** (*str*) – Absolute path to build location for generated project
- ****target_language** (*str, optional*) – Target default language for the generated project. Default value: 'VHDL'
- ****top_level_entity** (*str*) – Top-level HDL entity name.
- ****hdl_files_dict** (*dict*) – Dictionary of HDL libraries (keys) and list of absolute HDL files.
- ****ip_files** (*list of str*) – List of absolute paths fof IP files to add to project.
- ****constraint_files** (*list of str*) – List of absolute paths of constraints to add to project.

Returns None

get_project_file()
Gets the absolute path of the project file.

Returns Absolute path of project file. None if not found.

Return type str

fpgavendor_iface.**generate_timestamp**()
Generates a Unix 32 Bit Time Stamp from the systems time as an integer value.

Returns Timestamp as an integer for conversion to a VHDL *unsigned* std_logic_vector(31 downto 0).

Return type str

fpgavendor_iface.**resolve_tool_version**(***kwargs*)
Resolves the tool_version based on vendor

Keyword Arguments

- **tool_version** (*str*) – Version of tool to resolve.

- **vendor** (*str, optional*) – Vendor name. Valid values: `'altera'`, `'xilinx'`. Default `'xilinx'`

Returns The resolved `tool_version` when successful, otherwise `False`

Return type str

`fpgavendor_iface.`**run_ip_generate**(***kwargs*)

Executes the Quartus `qsys-generate`

Executes the `qsys-generate` to generate Quartus IP.

Keyword Arguments

- ****exe** (*str, optional*) – Executable to run. Default value: `'qsys-generate'`
- ****ip_file** (*str*) – Absolute path to IP file to generate.
- ****output_path** (*str*) – Absolute path where output files are generated.
- ****synth** (*bool, optional*) – generate for synthesis if `True` otherwise generate for simulation. Default value: `True`
- ****synth_language** (*str, optional*) – HDL language used to generate synthesis files. Valid values: `'VHDL'` or `'VERILOG'`. Default value: `'VERILOG'`
- ****sim_language** (*str, optional*) – HDL language used to generate simulation files. Valid values: `'VHDL'` or `'VERILOG'`. Default value: `'VHDL'`
- ****family** (*str, optional*) – FPGA Family. Default value: `False`
- ****device** (*str, optional*) – FPGA Device. Default value: `False`
- ****clean** (*bool, optional*) – Cleans IP Path Prior to Generating IP. Default value: `False`

Returns None

`fpgavendor_iface.`**run_ip_setup_simulation**(***kwargs*)

Executes the Quartus `run-ip-setup-simulation`

Executes the `run-ip-setup-simulation` to gather simulation setup commands for all Vendor IP libraries used in current design.

Keyword Arguments

- ****exe** (*str, optional*) – Executable to run. Default value: `'ip-setup-simulation'`
- ****project_file** (*str*) – Absolute path to project file to process.
- ****output_path** (*str*) – Absolute path where output files are generated.

Returns None

`fpgavendor_iface.`**run_quartus_sh**(***kwargs*)

Runs `quartus_sh` to execute compilation for standard HDL simulation libraries.

Keyword Arguments

- ****quartus_bin** (*str, optional*) – Name of executable to run quartus. Default value: `'quartus_sh'`
- ****script_mode** (*bool, optional*) – Run `quartus_sh` in script mode. Default value: `False`
- ** –

- ****tool** (*str*) – Name of the tool to run. Should be derived from: `simulator_class = SIMULATOR_FACTORY.select_simulator()` `simname = simulator_class.name`
- ****language** (*str, optional*) – Valid options: `'verilog'` or `'vhdl'`. Default value: `'vhdl'`
- ****modelsim_path** (*str*) – Absolute path to the version of Modelsim to use to compile standard HDL libraries.
- ****output_path** (*str*) – Absolute path where to compile standard HDL libraries.
- ****tcl_file_name** (*str*) – Absolute path to the `tcl` script used by `vivado`
- ****tcl_args** (*str, optional*) – Additional positional arguments needed to execute `tcl_file_name` Default value: `None`
- ****gui** (*bool, optional*) – Runs Quartus in GUI Mode. Default value: `False`
- ****project_file** (*str*) – Absolute path to project file.

Returns None

fpgavendor_iface.**run_vivado**(***kwargs*)

Runs `vivado` in specified mode.

When running in `'gui'` mode, the tcl_file keyword argument is the project file to open.

Note: The `shell=True` in `check_call` is important in windows where `vivado` is just a bat file.

Keyword Arguments

- ****vivado_bin** (*str, optional*) – Name of executable to run vivado. Default value: `'vivado'`
- ****mode** (*str, optional*) – Name of the mode to run. Valid values: `'batch'`, `'tcl'` or `'gui'`. Default value: `'batch'`
- ****tcl_file_name** (*str*) – Absolute path to the `tcl` script used by `vivado` when running in `'tcl'` mode, and the project file to open when running in `'gui'` mode.
- ****tcl_args** (*str, optional*) – Additional positional arguments needed to execute `tcl_file_name` Default value: `None`

Returns None

helpers/funcs.py

BSD © 2018-2019 Science and Technology Facilities Council & contributors

`funcs.py` is a helpers module to provide functions commonly used throughout.

class funcs.**Const**

Allows attributes to be treated like constants where they can be set one time only.

Raises ConstError if attribute already exists.

exception ConstError

funcs.**clean_path**(***kwargs*)

Cleans given path, by removing and recreating empty path.

Keyword Arguments ****path** (*str*) – Absolute path to clean.

Returns None

funcs.**copy_files_from_dir**(*src_path*, *dst_path*)
Copy files from source directory to destination directory.

Uses `shutil.copy2` to copy files recursively from one directory to another directory.

Parameters

- **src_path** (*str*) – Full-path of the source directory.
- **dst_path** (*str*) – Full-path of the destination directory.

Returns None

funcs.**get_kwarg**(*arg*, *kwargs*, *default*)
Finds arg in kwargs and returns the value. If arg is not found return the provided default value.

Parameters

- **arg** (*str*) – Name of argument to extract
- **kwargs** (*dict*) – Dictionary of keys, values to find and return `arg` value from.
- **default** – The value to assign if `arg` is not a key in `kwargs`

Returns `default`

funcs.**readfile_as_list**(*full_path*)
Takes a full-path to file and returns the contents as a list.

Parameters **full_path** (*str*) – Full path, including file name, of the file to read from.

Returns Line-by-line list of file contents.

Return type (list)

funcs.**replace_tag_with_attr_value**(*obj*, *tag*)
Replaces a `<TAG>` with a corresponding attribute value.

Looks for `<TAG>` (enclosed in **<>**) in tag string passed to the function, and if found returns a string with the tag replaced by the value as `obj.<TAG>` attribute.

If the attribute exists but is not populated with a value an "empty" string `''` is returned. This allows all <TAG> options in the layout configuration to be optional.

Parameters

- **obj** (*obj*) – Object to assign replaced `<TAG>` as attribute.
- **tag** (*str*) – string to check for sub-strings enclosed in <>.

Returns An empty string if corresponding attribute for the `<TAG>` is **not** found; or the corresponding attribute if found; or the original `<TAG>` if anything else.

Return type (str)

funcs.**set_kwargs**(*obj*, *kwargs*)
Sets attributes to specified existing object.

Parameters

- **obj** (*str*) – name of the object to assign attribute(s).
- **kwargs** (*dict*) – dictionary of key, values, where key is the attribute and value is the vaule to assign

Returns None

funcs.**strip_tag**(*tag*)
Removes the < and > from tag and returns the lowercase string enclosed.

Parameters **tag** (*str*) – UPPERCASE tag enclosed in < and >

Returns lowercase TAG with < and > stripped.

Return type (str)

funcs.**validate_file**(*f*, *touch=False*)
Checks the given file exists on the file-system. If touch is True, create the file if it is missing.

Parameters

- **f** (*str*) – Full path, including file name, of the file to validate.
- **touch** (*bool, optional*) – If set creates an empty file if not found on the file-system. Default value: False

Returns True on success, False otherwise.

funcs.**writefile_as_list**(*full_path*, *contents*)
Takes a list and writes contents line-by-line to file.

Parameters

- **full_path** (*str*) – Full path, including file name, of the file to write.
- **contents** (*list*) – Line-by-line list of contents to write.

Returns None

projectmanager.py

BSD © 2018-2019 Science and Technology Facilities Council & contributors

projectmanager.py is a module provided to setup, validate and execute source-code retrieval from externally hosted repositories. Project specific values are extracted from the corresponding settings.yml YAML file and the projects structure is determined from the config.yml YAML file, each passed to the module via *arguments* (see *Arguments*).

class projectmanager.**ProjectDependency**(*repo_root*, *name*, *category*, *url*, *local_path*, *remote_path*, *remote_rev*, *required_vhdl_libs=[]*, *vunit_default=False*, *top_level=False*, *board_name=None*, *board_settings=None*, *family=None*, *device=None*, ***kwargs*)
Processes project dependencies.

Takes a dictionary and converts keys to attributes before parsing configuration to construct all attributes required to describe a Dependency Object

Parameters

- **repo_root** (*str*) – The resolved $REPO_ROOT system environment variable. Required by: *projectmanager.ProjectDependency.layout_to_dir* and *projectmanager.ProjectDependency.prune_path_list*
- **name** (*str*) – Name of dependency
- **category** (*str*) – The dependency's category
- **url** (*str*) – Root URL to remote repository location for the dependency

- **local_path** (*str*) – Local path of the *working copy* for the dependency
- **remote_path** (*str*) – Remote path, from the `url` for the dependency
- **remote_rev** (*str*) – Specific repository revision for the dependency
- **required_vhdl_libs** (*list of str, optional*) – List of required HDL libraries required by the dependency. Default value: `list()`
- **vunit_default** (*bool, optional*) – VUnit dependency flag. Default value: `False`
- **top_level** (*bool, optional*) – Project Top-level dependency flag. Default value: `False`
- **board** (*str, optional*) – Target board for dependency. Default value: `None`
- **board_settings** (*str, optional*) – Settings file name for target board for dependency. Default value: `None`
- **family** (*str, optional*) – Target FPGA Family for HDL dependency. Default value: `None`
- **device** (*str, optional*) – Target FPGA Device for HDL dependency. Default value: `None`

Keyword Arguments

- ****repo_config_dict** (*dict*) – The layout dictionary which defines the projects layout within the repository and working copy on the local file-system. Required by *projectmanager.ProjectDependency.layout_to_dir* Default value: `dict()`
- ****lib_append_categories** (*list of str*) – Default value: `list()`
- ****bitbucket_username** (*str, optional*) – Default value: `None`
- ****tag_replacement_dict** (*dict*) – Dictionary of required tag replacement fields required by *projectmanager.ProjectDependency.layout_tag_attr_replace*. This is variable based on `category` Default value: `dict()`
- ****retag_layout** (*bool, optional*) – `True`

get_lib_from_name(***kwargs*)

Determines the `VHDL` library name from project name.

Determines the `VHDL` library name from the project name, ignoring if suffix already exists. the suffix is only appended if the `category` exists in the `lib_append_categories` list

Keyword Arguments

- ****name** (*str*) – The project name being processed. Default value: `''`
- ****category** (*str*) – The `category` of the project being processed. Default value: `''`
- ****lib_append_categories** (*list of str, optional*) – List of categories to append `lib_suffix` from `config.yml` configuration file (see *categories:*). If the `category` is **not** in this list the suffix will **not** be appended. Default value: `list()`.
- ****lib_suffix** (*str, optional*) – The library suffix to append. Default value: `'_lib'`

Returns The name of the `VHDL` library

Return type (str)

layout_tag_attr_replace(***kwargs*)
Replaces <TAG> with corresponding attribute

Looks for <TAG> (enclosed in <>) in tag string passed to the function, and if found returns a string with the tag replaced by the value in the self.<TAG> attribute.

If the attribute exists but is not populated with a value an "empty" string '' is returned. This allows all <TAG> options in the layout configuration to be optional. Although self.name should always have a value as it is also used to determine filenames in the directory structure.

Keyword Arguments ****tag** (*str*) – string to check for sub-strings enclosed in <>.

Returns An empty string if corresponding attribute for the <TAG> is **not** found; or the corresponding attribute if found; or the original <TAG> if anything else.

Return type (str)

layout_to_dir(***kwargs*)
Takes a single top-level and creates the directory and file structure as a list.

This layout structure is expected to be defined in the layout_dict keyword passed to the module.

Uses top_refs list to look for a reference to indicate the top reference string, which is used to populate self._top_dirs, this can then be used to create attributes for top_level reference directories.

Keyword Arguments

- ****layout_dict** (*dict*) – Default value: dict()
- ****repo_root** (*str*) – Default value: None
- ****top_refs** (*list of str*) – Default value: list()
- ****replace_tags** (*bool*) – Default value: True

Returns None

prune_path(***kwargs*)
Prunes path down towards root_path.

Takes a path and prunes that path until prune_to is found, if root_path is reached it means that prune_to was not found in the path.

Returns path up to and including prune_to, or False if not found.

Keyword Arguments

- ****path** (*str*) – Path to prune. Default value: None
- ****prune_to** (*str*) – Prune down to this path. Default value: None
- ****root_path** (*str*) – If root_path is reached the prune fails. Default value: None

Returns The found pruned path. The *bool:* False is returned if root_path is reached without finding the prune criteria.

Return type (str)

prune_path_list(***kwargs*)
Prunes list of paths based on keyword matching.

Keyword Arguments

- ****path_list** (*list of str*) – List of paths to prune. Default value: None
- ****keyword** (*str*) – Keyword which **must** exist in path_list item for prune. Default value: ''

- ****root_path** (*str*) – If `root_path` is reached the prune fails. Default value: `None`

Returns A list of pruned paths.

Return type (list of str)

retag_path(***kwargs*)
Re-tags path(s).

Keyword Arguments

- ****term** (*str*) – Term to retag.
- ****tag** (*str*) – Tag value to replace term with.
- ****paths** (*str or list of str*) – absolute path(s) to retag. This assumes the absolute path is in the form: `$REPO_ROOT/<TERM>/xx/xx/xx/<TERM>` where the second term will be replaced with `tag`.
- ****repo_root** (*str*) – The actual path of the system environment variable `$REPO_ROOT`

Returns List of retagged absolute paths. Empty `list()` if no repo_root supplied.

Return type list of str

static set_top_refs(*category*, *name*, *tool_version*)

Parameters

- **category** (*str*) – category to determine the top-level reference to return.
- **name** (*str*) – returned if `category` is *not* `boards` or `vendor_ip`
- **tool_version** (*str*) – returned if `category` *is* `boards` or `vendor_ip`

Returns list of str

static top_dir_lookup(***kwargs*)
Maps supported top-level directory names to known values for specified usage throughout

Keyword Arguments ****dir_name** (*str*) – Name of directory name to attempt to map.

Returns Mapped name if matched. `dir_name` if *not* matched.

Return type str

class projectmanager.**RepoFlow**(***kwargs*)
Parses the project settings file for repository_config key and processes locations found under the key. This provides enough information to construct the local working_copy path using `repo_root` inherited from `projectenvironment.ProjectEnvironment` and the URL of the remote repository.

Inherits: projectenvironment.ProjectEnvironment (obj): Inherited RepoFlow object. See `projectenvironment.projectenvironment`.

Username is assumed to be the same for each repository top-level URL (using `self.svn_username`, `self.bitbucket_username` and/or `self.git_username` inherited from `projectenvironment.ProjectEnvironment`)

construct_local_path(***kwargs*)
Constructs the local path for the local working-copy.

Constructed by joining `root` and `path`.

Parameters

- ****root** (*str*) – The top-level root. Usually derived from the `$REPO_ROOT` environment variable. Default value: `None`
- ****path** (*str*) – The path, derived from *subversion:*, but with the `subpath:` removed. This allows the switching between `trunk`, `branches` and `tags` to be managed by the repository client. Default value: `None`

Returns Full path of the local working-copy path.

Return type (str)

construct_remote_path(***kwargs*)

Constructs the remote path in the externally hosted repository.

Determines if the URL is `git` and/or `bitbucket` from `urlparse.netloc` Uses `urlparse` to construct URL, where `username` is constructed manually at the front of `urlparse.netloc`

Uses `posixpath` to construct path portion of URL by joining `urlparse.path`, `path` and `subpath`.

Parameters

- ****url** (*str*) – The top-level root URL. See *subversion:*. Default value: `None`
- ****path** (*str*) – The path. See *subversion:*. Default value: `None`
- ****subpath** (*str*) – The sub-path. See *subversion:*. Default value: `None`

Returns

- (str): Full path of the remote URL.
- (str): Username for remote URL.
- (str): Repo type `<'subversion' or 'git'>`. Determined from URL.

Return type tuple containing

get_category_from_path(***kwargs*)

Gets `category:` from `path`.

The first directory in the path string represents the category.

Keyword Arguments

- ****path** (*str*) – Default value: `''`
- ****category_mapping** (*dict, optional*) – Category mapping (see *categories:*). Default value: `dict()`.
- ****supported_categories** (*list of str, optional*) – List of supported categories from `config.yml` configuration file (see *categories:*). Default value: `list()`.

Returns the `category` extracted from `path`.

Return type (str)

get_contrib_lib_from_path(***kwargs*)

Gets `contrib_lib:` from `path`.

Assumes (i know!) that `contrib_lib` exists between the `category` and `name`. For a valid `contrib_lib` to exist the number of elements in `path` should be = 3.

Keyword Arguments

- ****name** (*str*) – The project name being processed. Default value: `''`
- ****path** (*str*) – The path of the project on the file-system. Default value: `''`

- ****category_mapping** (*dict, optional*) – Category mapping (see *categories:*). Default value: `dict()`.

Returns the `contrib_lib` extracted from `path`.

Return type (str)

process_repository_config(***kwargs*)

Process repository configuration for current project

Keyword Arguments ****repository_config** (*dict*) – Dictionary of Repository Configuration for Repository Type extracted from `settings.yml` YAML file.

Returns A list of processed *ProjectDependency* objects.

Return type (list of obj)

class projectmanager.**VendorIpDependency**(***kwargs*)

Vendor IP is an Vendor IP object containing methods and attributes required for processing `vendor_ip`

Keyword Arguments

- ****repo_root** (*str*) – Absolute path to the system environment `$REPO_ROOT`.
- ****master_vendor_ip** (*obj*) – A project dependency object for the *master* `vendor_ip` dependency for the project.
- ****vendor** (*str*) – FPGA Vendor. Used to determine `vendor_ip` locations. Valid values: `'xilinx'` and `altera'`. Default value: `'xilinx'`
- ****tool_version** (*str*) – FPGA Tool Version. Used to determine `vendor_ip` locations.
- ****family** (*str*) – FPGA Family. Used to determine `vendor_ip` locations.
- ****device** (*str*) – FPGA Device. Used to determine `vendor_ip` locations.
- ****ip_dict** (*dict*) –
- ****settings_file** (*str*) – Absolute path to settings YAML file, used in logs.

resolved_master_tags(***kwargs*)

Keyword Arguments

- ****paths** (*str, or list of str*) – Absolute path to resolve tags
- ****vendor** (*str*) – FPGA Vendor. Used to determine `vendor_ip` locations. Valid values: `'xilinx'` and `altera'`. Default value: `'xilinx'`
- ****tool_version** (*str*) – FPGA Tool Version. Used to determine `vendor_ip` locations.
- ****family** (*str*) – FPGA Family. Used to determine `vendor_ip` locations.
- ****device** (*str*) – FPGA Device. Used to determine `vendor_ip` locations.

Returns List of absolute paths with tags resolved.

Return type list of str

setup.py

helpers/version.py

version.py is a helpers module provided to handle public version numbering which follows the PEP440 guidelines.

Note: The __version__ uses the *version.Version* (defined in this module), therefore is used towards the end of this module.

---

class version.**Version**(*major*, *minor*, *micro*, *prerelease=''*, *svn_rev=''*, *disable_svn_logging=False*)

Creates a Class with methods to construct and fetch PEP440 Public Version Numbering

Creates an initial PEP440 Public Version Identifier from the supplied parameters after validating the prerelease parameter,

Parameters

- **major** (*int*) – Representing the Major Version Number (changes affecting execution method)
- **minor** (*int*) – Representing the Minor Version Number (bugfixes)
- **micro** (*int*) – Representing the Micro Version Number (correction of typos/release notes)
- **prerelease** (*int, optional*) – Addition Pre-Release Indicator. Valid values: a, b, rc, where: a = alpha; b = beta; and rc = release candidate
- **svn_rev** (*str, optional*) – SVN Revision Number
- **disable_svn_logging** (*bool*) – Do not include SVN output in logs. Default value: False

get_version()

Method to get the string representation of the version number.

Returns String of Version Number

set_version(*major*, *minor*, *micro*, *pre*)

Method to set the string representation of a Version Number.

Parameters

- **major** – Integer Value Representing the Major Version Number (changes affecting execution method)
- **minor** – Integer Value Representing the Minor Version Number (bugfixes)
- **micro** – Integer Value Representing the Micro Version Number (correction of typos/release notes)
- **pre** – Optional Addition Pre-Release Indicator... a = alpha b = beta rc = release candidate

Returns string

version.**about**(*modname*, *ver*, *revision*)

Method to Create an About String, to display Method Name and Version/Rev

Parameters

- **modname** (*str*) – Module Name
- **ver** (*str*) – Version Number, in the form: MAJOR.MINOR.MICRO
- **revision** (*str*) – SVN Revision Number

Returns

Return type str

vunit_iface.py

BSD © 2018-2019 Science and Technology Facilities Council & contributors

`vunit_iface.py` is a module to handle interfacing and utilising external `VUnit` module.

See VUnit Documentation.

class vunit_iface.**VUnitProject**(***kwargs*)

VUnit Project is a VUnit object used to configure and execute simulation test-cases.

Keyword Arguments

- ****simulation_path** (*str*) – Absolute simulation root path for the VUnit project.
- ****sim_pre_compiled_libs** (*str*) – Absolute path to the location for precompiled simulation libraries.
- ****fpga_vendor** (*str*) – FPGA Vendor. Used to determine build and precompiled library locations. Valid values: `'xilinx'` and `altera'`. Default value: `'xilinx'`
- ****fpga_toolversion** (*str*) – FPGA Tool Version. Used to determine build and precompiled library locations.
- ****sim_version** (*str*) – Simulation Version. Used to determine build and precompiled library locations.
- ****hdl_lib_file_dict** (*dict, optional*) – A dictionary containing library keys and absolute paths to HDL files for each library. Default value: `dict()`
- ****tb_hdl_lib_file_dict** (*dict, optional*) – A dictionary containing library keys and absolute paths to test-bench HDL files for each library. Default value: `dict()`
- ****args** (*obj, optional*) – `argparse` object containing arguments passed to script. If one is not supplied it will be generated here.
- ****use_osvvm** (*bool, optional*) – Use OSVVM Components. Default value `False`
- ****use_verification_components** (*bool, optional*) – Use `VUnit` Verification Components. Default value `False`
- ****settings_file** (*str*) – Absolute path to settings YAML file, used in logs.
- ****clean_precompiled_libs** (*bool, optional*) – If `True` clean precompiled standard simulation libraries for the current project. Default value: `False`
- ****msim_64bit** (*bool, optional*) – If `False` uses -32bit flag to execute Modelsim.
- ****hdl_sim_libs** (*dict*) – Dictionary of Vendor Standard HDL Libraries to add to VUnit.
- ****modelsim_installdir** (*str, optional*) – Absolute path to the Modelsim install directory. Used when compiling `altera` standard libraries.
- ****repo_root** (*str*) – Absolute path to the resolved system environment variable `$REPO_ROOT`.

vu

obj – The VUnit object.

add_hdl_files(***kwargs*)
Adds HDL Files to corresponding library.

Keyword Arguments ****hdl_lib_file_dict** (*dict*) – A dictionary containing library keys and absolute paths to files for each library.

Returns None

add_standard_libraries(***kwargs*)
Adds Standard Simulation Libraries to VUnit

This method is derived from : VUnit supplied example

Keyword Arguments

- ****vunit_obj** (*obj, optional*) – A VUnit object. Default value: `self.vu`
- ****standard_library_path** (*str, optional*) – Absolute path to the location for precompiled simulation libraries. Default value: `self.sim_pre_compiled_libs_path`
- ****hdl_sim_libs** (*dict*) – Dictionary of Vendor Standard HDL Libraries to add to VUnit.

Returns `True` when successful, `False` when unsuccessful.

Return type bool

add_vivado_sim_dependencies(***kwargs*)
Add Vivado simulation dependencies to VUnit object

Keyword Arguments

- ****vivado_path** (*str*) – Absolute path to Vivado.
- ****src_paths** (*list of str, optional*) – List of absolute paths to search for dependencies relative to `vivado_path`. Default value: `['data/verilog/src']`
- ****src_files** (*list of str, optional*) – List of source file dependencies. Default value: `['glbl.v']`
- ****glbl** (*bool, optional*) – Include glbl dependency. Default value: `False`
- ****glbl_lib** (*str, optional*) – Library to add glbl to. Default value: `'xil_defaultlib'`
- ****vunit_obj** (*obj, optional*) – VUnit object. Default value: `self.vu`

Returns `True` if successful, `False` if unsuccessful

Return type bool

compile_project_ip(***kwargs*)
Compiles FPGA Project Specific IP Simulation Libraries

Keyword Arguments

- ****project_file** (*str*) – Absolute path to project file to parse.
- ****simulation_path** (*str, optional*) – Absolute path to the location for simulation path Default value: `self.simulation_path`
- ****project_ip_subpath** (*str, optional*) – Name of project_ip libraries subpath. Default value: `'project_ip'`
- ****msim_setup_script** (*str, optional*) – Filename of `.tcl` to compile Vendor IP simulation files in Modelsim. Default value: `'mentor/msim_setup.tcl'`

- ****top_level_lib** (*str, optional*) – Top-level library to add Vendor IP simulation file to in Modelsim, if `False` the library name will match the filename, as configured in the generated `.qip` file. Default value: `False`

Returns None

compile_standard_libraries(***kwargs*)
Compiles Standard Simulation Libraries

This method is derived from : VUnit supplied example

Keyword Arguments

- ****sim_pre_compiled_libs** (*str, optional*) – Absolute path to the location for precompiled simulation libraries. Default value: `self.sim_pre_compiled_libs_path`
- ****standard_lib_subpath** (*str, optional*) – Name of standard library subpath. Default value: `'standard'`
- ****fpga_vendor** (*str*) – FPGA Vendor. Valid values: `'xilinx'` or `'altera'`.
- ****settings_file** (*str*) – Absolute path to settings YAML file, used in logs.
- ****clean** (*bool, optional*) – If `True` cleans the standard library path. Default value: `False`
- ****msim_64bit** (*bool, optional*) – If `False` uses -32bit flag to execute Modelsim. Default value: `False`
- ****hdl_sim_libs** (*dict*) – Dictionary of Vendor Standard HDL Libraries to add to VUnit.
- ****modelsim_installdir** (*str, optional*) – Absolute path to the Modelsim install directory. Used when compiling `altera` standard libraries.

Returns `True` when successful, `False` when unsuccessful.

Return type bool

configure_gui(***kwargs*)
Pre-processing of additional simulation init file to use when GUI mode is enabled.

Keyword Arguments

- ****sim_init_script** (*str*) – Absolute path to initialisation file for GUI mode.
- ****repo_root** (*str*) – Replacement str for $REPO_ROOT.

Returns None

static is_verilog_header(*filename*)
Checks if filename indicates the file is a verilog header file.

Parameters **filename** (*str*) – Filename to check if it matches the criteria of a verilog header file.

Returns `True` if a verilog header file, otherwise `False`

Return type bool

read_compile_order(***kwargs*)
Read the vendor_ip simulation compile order file and filter out duplicate files

Keyword Arguments ****filename** (*str*) – Absolute path to compile order filename to process.

Returns list of str, set, list of str

Return type tuple containing

resolve_gui_init_script(***kwargs*)
Parses Simulation GUI Initialisation Script

There appears to be an issue with Modelsim directly using $env(REPO_ROOT) in scripts passed from `VUnit` to Modelsim, so this workaround creates a file with `REPO_ROOT` resolved and uses the created resolved file instead:

Keyword Arguments

- ****sim_init_script** (*str*) – Absolute path to initialisation file to resolve.
- ****repo_root** (*str*) – Replacement str for $REPO_ROOT.

Returns

Return type str

setup_simulation_options(***kwargs*)
Sets up the compile and sim options.

Keyword Arguments

- ****glbl** (*bool, optional*) – Include glbl dependency. Default value: `False`
- ****glbl_lib** (*str, optional*) – Library to add glbl to. Default value: `'xil_defaultlib'`

Returns None

vunit_iface.**tb_cfg_encode**(*tb_cfg*)
Encodes a VUnit test-case dictionary configuration to a string to pass to VUnit Test-Case

String is passed to VHDL test-bench file using the `encoded_tb_cfg` generic.

Parameters **tb_cfg** (*dict*) –

Returns str

Electronic System Design Group

1.1.1.4 Arguments

Arguments to pass to `FPGAFlow`.

Required

--config

The configuration YAML file describing:

- configuration of the host system
- source-code repository layout

See *Configuration* for description.

--settings

The Settings YAML file describing:

- Libraries and Tools Used by the current project
- Project Specific Settings

See *Settings* for description.

Optional

--checkout_enabled

Enables checkout of source code from repository. Defaults to: `False`.

To use:

```
--checkout_enabled
```

--headless

Headless Flag for Forcing GUI Based Options to `False`. Used to ensure Continuous Integration environments execute unimpeded by external user input. Defaults to: `False`.

```
--headless
```

--skip_xml2vhdl

Prevents the XML2VHDL Generation Script from being executed. Defaults to: `False`.

To use:

```
--skip_xml2vhdl
```

Warning: This Argument should be used with caution. It assumes existing generated `.vhd` exists for the project to compile. The resulting compilation **may** be out-of-date if changes to the memory-map have been made to any `.xml` files in the project.

--open_gui

Opens project in GUI. Overwritten by `--headless`, which will prevent any GUI from running.

fpga

Opens the project in either `quartus` or `vivado`.

sim

Opens the project in simulation for each enabled test-case.

```
--open_gui <fpga|sim>
```

--clean

Cleans Output Paths Generated by `FPGAFlow`.

all

Cleans **all** generated output paths excluding `'precompiled_sim'` which must be called explicitly.

fpga

Cleans the project for the current `vendor`, `tool_version` and `project_name`. Any existing FPGA Vendor project in the build path will be backed up, by appending a date/time stamp to the build folder and a new project will be generated by the script. If not used as an argument the script will use the existing project.

ip

Cleans the projects `vendor_ip` enabled IP generated paths for the current `vendor`, `tool_version`.

precompiled_sim

Cleans the pre-compiled libraries for the current `vendor`, `tool_version`, `sim_vendor` and `sim__version` for under the root path defined by `$SIM_PRE_COMPILED_LIBS_PATH`.

> **Warning:** This may fail if the user does not have the correct permissions to operate on this directory.

sim

Cleans the projects compiled simulation libraries.

```
--clean all
```

Electronic System Design Group

1.1.1.5 Configuration

The script is configured using a YAML file passed using the `--config` argument (see *Arguments*).

```
python ./docflow.py --config <CONFIG.YML>
```

The description of the YAML file is described below.

Tag Substitution

The configuration file operates using tag substitution with values set in the corresponding *Settings* file. Tags are all uppercase enclosed in <> characters.

```
<UPPERCASE_TAG>
```

where the UPPERCASE tag represents a key in the *Settings* which will be substituted when the script is executed. The *repository_required* fields in the configuration file should include all tag substitution keys.

Sections

`repository:`

The repository section configures the expected directory structure of each supported category for code stored in the externally hosted repository.

Listing 1: complete example of `config.yml` entry for `repository` section

```
repository:
  required:
    - name
    - contrib_lib
    - vendor
    - tool_version
    - sim_vendor
    - sim_version
  categories:
    - boards
    - library
    - tools
    - vendor_ip
  category_mapping:
    library: libraries
  lib_append_categories:
    - boards
    - library
  boards_layout:
    - boards:
      - <VENDOR>:
        - <NAME>:
          - <TOOL_VERSION>:
            - constraints:
            - docs:
              - common:
```

(continues on next page)

(continued from previous page)

```
            - logo:
          - doxygen:
            - <DOXY_VERSION>:
              - source:
          - sphinx:
            - <SPHINX_VERSION>:
              - build:
              - source:
                - _static:
                - _templates:
        - scripts:
          - python:
          - tcl:
        - settings:
          - <NAME>.yml
        - simulation:
          - testbenches:
            - EXCLUDE
        - src:
          - vhdl:
            - EXCLUDE
            - generated:
          - vhdl_packages:
          - xml:
library_layout:
  - libraries:
    - <CONTRIB_LIB>:
      - <NAME>:
          - constraints:
            - <VENDOR>:
              - pre_flow:
              - post_module:
              - post_flow:
              - pre_synth:
              - pre_opt:
          - docs:
            - common:
              - logo:
            - doxygen:
              - <DOXY_VERSION>:
                - source:
            - sphinx:
              - <SPHINX_VERSION>:
                - source:
                  - _static:
                  - _templates:
          - netlists:
          - scripts:
            - tcl:
            - python:
          - settings:
            - <NAME>.yml
          - simulation:
            - testbenches:
              - EXCLUDE
          - src:
            - vhdl:
```

(continues on next page)

(continued from previous page)

```
                - EXCLUDE
                - generated:
              - vhdl_packages:
              - xml:
  tools_layout:
    - tools:
      - <NAME>:
        - docs:
          - common:
                - logo:
          - doxygen:
            - <DOXY_VERSION>:
              - source:
          - sphinx:
            - <SPHINX_VERSION>:
              - build:
              - source:
                - _static:
                - _templates:
        - scripts:
          - python:
            - <NAME>-<ORG>:
              - <NAME>:
```

required:

This provides a list of required field from the *Settings* file required to execute the script correctly. The script expects each of the list items to be populated in the settings YAML file ready for *Tag Substitution*.

categories:

This provides a list of `categories` supported by the script. The layout of each repository item is defined by its `category` and the *<CATEGORY>_layout:* settings defined in the configuration YAML file. The list of `categories` is used by *Tag Substitution* to determine the *<CATEGORY>_layout:*.

category_mapping:

If the `category` is stored in the repository using a different reference `category_mapping` can be used to map the different category name to the `category` name.

lib_append_categories:

The convention employed by this script is to name libraries matching their `name` in the repository. This section allows `categories` to be listed where `_lib` should be appended to the `name`.

Note: Names which already have `_lib` included in their name in the repository will NOT have `_lib` re-appended.

<CATEGORY>_layout:

A layout should be defined for each `category` in the *categories:* list.

This defines the layout for all files stored in the corresponding repository location. *Tag Substitution* is used to fully construct the `<CATEGORY>_layout` and it is expected that all tags are listed in the *repository_required* section of the configuration file.

contributors:

Listing 2: complete example of `config.yml` entry for `contributors` section

```
contributors:
  contrib_lib_list:
    - common
    - lib_ox
    - lib_stfc
```

contrib_lib_list:

A list of contributors for source-code and development within the project. The values here define the ownership and responsibility of items stored in the repository.

Note: It is expected that modifications, enhancements and bugfixes should be undertaken by the owner of the source-code in question.

---

fpga:

Listing 3: complete example `config.yml` entry for `fpga` using `xilinx` and `altera`

```
fpga:
  supported_vendors:
    - xilinx
    - altera
  xilinx:
    license:
      xilinxd_license_file:
        - 2100@<LICENSE.SERVER1>
    paths:
      root_path: /path_to/xilinx/<TOOL_VERSION>/
      root_name: vivado_rootdir
      additional_paths:
        installdir: Vivado/<TOOL_VERSION>
        xilinx_vivado: Vivado/<TOOL_VERSION>
      add_to_path:
        - Vivado/<TOOL_VERSION>/bin
        - Vivado_HLS/<TOOL_VERSION>/bin
   family:
```

(continues on next page)

(continued from previous page)

```
    - kintex_ultrascale
  devices:
    - xcku040-ffva1156-2-e
    - xcku040-fbva676-1-c
  boards:
    - avnet_ku040
    - itpm_v1_1
  tools:
    - 2016.2
    - 2016.3
    - 2018.1
    - 2018.3
  default_lib:
    xil_defaultlib

altera:
  license:
    lm_license_file:
      - 1800@<LICENSE.SERVER1>
  mode:
    supported_modes:
      - 0
      - 1
    env_name: quartus_64bit
    ld_library_path_0:
    ld_library_path_1:
  paths:
    root_path: /path_to/Altera/prime/<TOOL_VERSION>
    root_name: quartus_installdir
    additional_paths:
      quartus_rootdir: quartus
      qsys_rootdir: qsys/bin
    add_to_path:
      - quartus/bin
      - nios2eds/bin
      - quartus/sopc_builder/bin
      - hld/bin
  family:
    - arria_10
    - stratix_10
  family_mappings:
    arria_10:     'Arria 10'
    stratix_10:   'Stratix 10'
  devices:
    - 10AS066N2F40I2SGES
    - 1SX280LU3F50I3XG
  boards:
    - a10_soc_dev_kit
    - xpressgx_s10fh200g
  tools:
    - pro/17.0
    - pro/17.1
    - pro/18.0
    - pro/18.1
    - std/17.0
    - std/17.1
    - std/18.0
```

(continues on next page)

(continued from previous page)

```
      - std/18.1
```

supported_vendors:

List of supported simulation vendors. Each entry in this list should have a corresponding entry under *supported_vendors:*.

<FPGA VENDOR>:

license:

A dictionary of `key value` pairs defining the license system environment variable name and values to assign when configuring the system environment. Values can be a single value, or a list of values. For `xilinx`:

Listing 4: example `config.yml` entry for `fpga license`

```
xilinx:
   license:
     xilinxd_license_file:
       - 2100@<LICENSE.SERVER1>
       - 2100@<LICENSE.SERVER2>

# or:

xilinx:
   license:
     xilinxd_license_file:
       2100@<LICENSE.SERVER1>
```

paths:

A dictionary of `key value` pairs defining path based system environment variables and values to assign when configuring the system environment. Supported `keys`:

root_path: Path to `<FPGA VENDOR>` root installation path.

> **Note:** *Tag Substitution* works here, where `<TOOL_VERSION>` is defined in the `--settings` YAML file.

root_name: Name of system variable to create with reference to `root_path:`

add_to_path: A list of sub-paths relative to `root_path:` to append to the system `$PATH` variable.

```
xilinx:
  paths:
      root_path: /path_to/xilinx/<TOOL_VERSION>/
      root_name: vivado_rootdir
      additional_paths:
        installdir: Vivado/<TOOL_VERSION>
        xilinx_vivado: Vivado/<TOOL_VERSION>
      add_to_path:
```

(continues on next page)

(continued from previous page)

```
        - Vivado/<TOOL_VERSION>/bin
        - Vivado_HLS/<TOOL_VERSION>/bin
```

Will result in the following system environment variables to modified or created when `tool_version: 2018.3` is set in the *tool_version:* file as an example:

```
setenv VIVADO_ROOTDIR '/path_to/xilinx/2018.3'
setenv INSTALLDIR '/path_to/xilinx/2018.3'
setenv XILINX_VIVADO '/path_to/xilinx/2018.3'
setenv PATH '/path_to/xilinx/2018.3/Vivado/2018.3/bin':${PATH}
setenv PATH '/path_to/xilinx/2018.3/Vivado_HLS/2018.3/bin':${PATH}
```

mode:

A dictionary of `key value` pairs defining execution mode of *FPGA* tool. Support `keys` with example `values`:

supported_modes: A list of supported modes which are accepted by the system environment variable named by *mode:* under `env_name:`

env_name: Name of the system environment variable which sets the tools mode of execution.

ld_library_path_<MODE_IN_SUPPORTED_MODES>: Path to append to `LD_LIBRARY_PATH` which points to location of non-standard libraries required for the correct operation of the simulation tool when running in `<MODE_IN_SUPPORTED_MODES>` bit mode.

Listing 5: example `config.yml` entry for `fpga mode`

```
altera:
  mode:
    supported_modes:
      - 0
      - 1
    env_name: quartus_64bit
    ld_library_path_0:
    ld_library_path_1:
```

Will result in the following system environment variables to be modified or created, using `tool_mode: 1` is set in the *tool_mode:* as an example:

```
setenv QUARTUS_64BIT 1
```

family:

A list of *FPGA* families supported by `FPGAFlow`. The names here should match the corresponding name as it appears in the *FPGA* vendor's project file.

devices:

A list of *FPGA* devices supported by `FPGAFlow`. The names here should match the corresponding part number as it appears in the *FPGA* vendor's project file.

boards:

A list of hardware platforms to which can be targeted by `FPGAFlow`.

Note: The values here will be used by *Tag Substitution* to populate the `<NAME>` and corresponding fields when processing the `settings.yml` YAML file (see *Settings*) and when determining the `boards_layout:` directory structure.

---

tools:

A list of *FPGA* tool versions supported by `FPGAFlow`.

Note: The values here will be used by *Tag Substitution* to populate `<TOOL_VERSION>` when processing the `settings.yml` YAML file (see *Settings*) and when setting up the system environment and paths required to allow `FPGAFlow` to execute the correct versions of *FPGA* vendor's tools. This also determines then''build'' directory structure and which `vendor_ip` to use in the project.

`default_lib:`

The name of the default library used by the *FPGA* vendor.

simulation:

Listing 6: complete example `config.yml` entry for `simulation` using `modelsim`

```
simulation:
  supported_vendors:
    - modelsim
  modelsim:
    license:
      mgls_license_file:
        - 1717@<LICENSE.SERVER1>
        - 1717@<LICENSE.SERVER2>
    paths:
      root_path: /path_to/modelsim/<SIM_VERSION>/modeltech
      root_name: model_tech
      additional_paths:
        modelsim_installdir: bin
      add_to_path:
        - bin
    mode:
      supported_modes:
        - 32
        - 64
      env_name: mti_vco_mode
      ld_library_path_32:
        - /path_to/non-standard/lib32
      ld_library_path_64:
        - /pack_to/non-standard/lib64
  xilinx:
```

(continues on next page)

(continued from previous page)

```
    sim_libs:
      - 'unisims_ver'
      - 'unimacro_ver'
      - 'unifast_ver'
      - 'secureip'
      - 'xpm'
      - 'unisim'
      - 'unimacro'
      - 'unifast'
      - 'simprims_ver'
    verilog_sim_libs: None
    vhdl_sim_libs: None
  altera:
    sim_libs: None
    verilog_sim_libs:
      - 'altera_ver'
      - 'lpm_ver'
      - 'sgate_ver'
      - 'altera_mf_ver'
      - 'altera_lnsim_ver'
      - 'fourteennm_ver'
      - 'fourteennm_ct1_ver'
      - 'cyclone10gx_ver'
      - 'cyclone10gx_hip_ver'
      - 'cyclone10gx_hssi_ver'
      - 'tennm_ct1_ver'
      - 'tennm_ver'
      - 'twentynm_ver'
      - 'twentynm_hip_ver'
      - 'twentynm_cssi_ver'
    vhdl_sim_libs:
      - 'altera'
      - 'lpm'
      - 'sgate'
      - 'altera_mf'
      - 'altera_lnsim'
      - 'fourteennm'
      - 'fourteennm_ct1'
      - 'cyclone10gx'
      - 'cyclone10gx_hip'
      - 'cyclone10gx_hssi'
      - 'tennm_ct1'
      - 'tennm'
      - 'twentynm'
      - 'twentynm_hip'
      - 'twentynm_hssi'
```

supported_vendors:

List of supported simulation vendors. Each entry in this list should have a corresponding entry under *supported_vendors:*.

<SIM TOOL VENDOR>:

supported_tools:

A list of supported versions. The items in this list are used to differentiate between installed versions of the tool on the host system. Therefore it is expected that each version listed is installed on the system using the value in the list to define the version of the tool. This allows multiple tools to be installed for device and legacy support.

license:

A dictionary of `key value` pairs defining the license system environment variable name and values to assign when configuring the system environment. Values can be a single value, or a list of values. For `modelsim`:

Listing 7: example `config.yml` entry for `simulation license`

```
modelsim:
   license:
     mgls_license_file:
       - 1717@<LICENSE.SERVER1>
       - 1717@<LICENSE.SERVER2>

# or:

modelsim:
   license:
     mgls_license_file:
       1717@<LICENSE.SERVER1>
```

paths:

A dictionary of `key value` pairs defining path based system environment variables and values to assign when configuring the system environment. Supported `keys`:

root_path: Path to `<SIM TOOL VENDOR>` root installation path.

> **Note:** *Tag Substitution* works here, where `<SIM_VERSION>` is defined in the `--settings` YAML file.

root_name: Name of system variable to create with reference to `root_path:`

add_to_path: A list of sub-paths relative to `root_path:` to append to the system `$PATH` variable.

Listing 8: example `config.yml` entry for `simulation paths`

```
modelsim:
  paths:
    root_path: /path_to/modelsim/<SIM_VERSION>/modeltech
    root_name: model_tech
    additional_paths:
      modelsim_installdir: bin
    add_to_path:
      - bin
```

Will result in the following system environment variables to modified or created when `sim_version: 10.6e` is set in the *sim_version:* file as an example:

```
setenv MODEL_TECH '/path_to/modelsim/10.6e/modeltech'
setenv MODELSIM_INSTALLDIR '/path_to/modelsim/10.6e/modeltech/bin'
setenv PATH '/path_to/modelsim/10.6e/modeltech/bin':${PATH}
```

mode:

A dictionary of `key value` pairs defining execution mode of simulation tool. Support `keys` with example `values`:

supported_modes: A list of supported modes which are accepted by the system environment variable named by *mode:* under `env_name:`

env_name: Name of the system environment variable which sets the tools mode of execution.

ld_library_path_<MODE_IN_SUPPORTED_MODES>: Path to append to `LD_LIBRARY_PATH` which points to location of non-standard libraries required for the correct operation of the simulation tool when running in `<MODE_IN_SUPPORTED_MODES>` bit mode.

Listing 9: example `config.yml` entry for `simulation mode`

```
modelsim:
  mode:
    supported_modes:
      - 32
      - 64
    env_name: mti_vco_mode
    ld_library_path_32:
      - /path_to/non-standard/lib32
    ld_library_path_64:
      - /pack_to/non-standard/lib64
```

Will result in the following system environment variables to be modified or created, using `sim_mode:` `32` is set in the *sim_mode:* as an example:

```
setenv MTI_VCO_MODE 32
setenv LD_LIBRARY_PATH '/path_to/non-standard/lib32':${LD_LIBRARY_PATH}
```

xilinx:

sim_libs:

A list of standard `verilog` and `VHDL` library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

verilog_sim_libs:

A list of standard `verilog` library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

Note: For `xilinx` this can be unset, i.e. left empty: `verilog_sim_libs:`

---

vhdl_sim_libs:

A list of standard VHDL library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

Note: For xilinx this can be unset, i.e. left empty: vhdl_sim_libs:

---

altera:

sim_libs:

A list of standard verilog and VHDL library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

Note: For altera this *must* be unset, i.e. left empty: sim_libs:

---

verilog_sim_libs:

A list of standard verilog library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

vhdl_sim_libs:

A list of standard VHDL library mappings for adding to simulations. These are precompiled once and should be available in the path defined by *SIM_PRE_COMPILED_LIBS_PATH*

vhdl:

Listing 10: example

```
vhdl:
```

documentation:

Configuration for documentation tools.

Listing 11: complete example of config.yml entry for contributors section

```
documentation:
  required:
    - doxy_version
    - dot_version
    - sphinx_version
  supported_languages:
```

(continues on next page)

(continued from previous page)

```
    - vhdl
    - python
  doxygen:
    supported_tools:
      - 1.8.14
    paths:
      root_path: /path_to/doxygen-<DOXY_VERSION>/
      root_name: doxygen_installdir
      add_to_path:
        - bin
      internal_references:
        doxypypy_script: ../../../scripts/shell/doxypypy/py_filter
        doxy_template: templates/doxygen/<DOXY_VERSION>/templates/doxy_template.
↪Doxyfile
        doxy_extra_style_sheets:
          - templates/doxygen/<DOXY_VERSION>/stylesheets/regtables.css
  dot:
    supported_tools:
      - 2.38.0
      - 2.40.1
    paths:
      root_path: /path_to/graphviz-<DOT_VERSION>/bin
      root_name: dot_path
  sphinx:
    supported_tools:
      - 1.8.2
    paths:
      internal_references:
        sphinx_conf_template: templates/sphinx/<SPHINX_VERSION>/templates/conf.py
        sphinx_make_template: templates/sphinx/<SPHINX_VERSION>/templates/Makefile
```

required:

This provides a list of required field from the *Settings* file required to execute the script correctly. The script expects each of the list items to be populated in the settings YAML file ready for *Tag Substitution*.

supported_languages:

A list of languages supported by the script for the auto-generation of documentation.

doxygen:

Configuration for `doxygen` tool used to auto-document `VHDL` files.

support_tools:

A list of supported versions.

Note: Versions of these tools should be available on the host system and one of these values must be included in the

settings file (see *Settings*).

paths:

Configuration of the paths for the installation directories for doxygen.

root_path: Path to doxygen root installation path.

> **Note:** *Tag Substitution* works here, where <DOXY_VERSION> is defined in the --settings YAML file.

root_name: Name of system variable to create with reference to root_path:

add_to_path: A list of sub-paths relative to root_path: to append to the system $PATH variable.

internal_references: A list of reference names and values for addition specific items to the doxygen configuration. This includes references to additional scripts, template files and a list of css files.

> **Note:** internal_references: are relative to: $REPO_ROOT/tools/fpgaflow/scripts/python/
>
> *Tag Substitution* works here, where <DOXY_VERSION> is defined in the --settings YAML file.

sphinx:

Configuration for sphinx tool used to auto-document python files.

Note: Sphinx and its dependencies is installed in the the python virtual environment and managed by pipenv

support_tools:

A list of supported versions.

Note: Versions of these tools should be available on the host system via pipenv virtual environment and one of these values must be included in the settings file (see *Settings*).

paths:

internal_references: A list of reference names and values for addition specific items to the sphinx configuration. This includes references template files.

> **Note:** internal_references: are relative to: $REPO_ROOT/tools/fpgaflow/scripts/python/
>
> *Tag Substitution* works here, where <DOXY_VERSION> is defined in the --settings YAML file.

dot:

Configuration for `graphviz` tool used by `doxygen` for the generation of diagrams.

support_tools:

A list of supported versions.

Note: Versions of these tools should be available on the host system and one of these values must be included in the settings file (see *Settings*).

---

paths:

Configuration of the paths for the installation directories for `graphviz`.

root_path: Path to `graphviz` root installation path

Note: *Tag Substitution* works here, where `<DOT_VERSION>` is defined in the `--settings` YAML file.

---

root_name: Name of system variable to create with reference to `root_path:`

environment:

Configuration options for the user environment, to prevent user based locations and information being stored in source-code repositories.

Listing 12: complete example of `config.yml` entry for `environment` section

```
environment:
  required:
     - REPO_ROOT
     - PIPENV_SHELL
     - BITBUCKET_USERNAME
     - GIT_USERNAME
     - SVN_USERNAME
     - SIM_PRE_COMPILED_LIBS_PATH
  exclude_keywords:
     - 'build/'
     - 'synosys'
     - 'aldec'
     - 'cadence'
     - '.svn'
     - 'template'
     - '.Xil'
     - '.cache'
     - '.hw'
     - '.ip_user_files'
     - 'scripts'
     - 'ref_design'
```

(continues on next page)

(continued from previous page)

```
    - 'bin'
    - 'docs'
    - 'settings'
    - 'constraints'
    - '.settings'
    - 'modelsim_'
    - 'vunit_output'
```

required:

This provides a list of required system environment variables required to successfully execute the script correctly. The script expects each of the list items to declared prior to the script executing and are validated when the script is executed.

REPO_ROOT

Sets the top-level root path where all projects/libraries will be checkout out to and processed from.

PIPENV_SHELL

Path to the executable for the shell `pipenv` should run.

BITBUCKET_USERNAME

The username used to access user controlled private `bitbucket` *cloud* hosted repositories.

GIT_USERNAME

The username used to access user controlled private `git` *server* hosted repositories (which may be `bitbucket`).

SVN_USERNAME

The username used to access user controlled private `subversion` repositories.

SIM_PRE_COMPILED_LIBS_PATH

The root path to precompiled Modelsim libraries.

exclude_keywords:

A list of exclude keywords to use throughout the flow. Files or directories containing any term in the list will automatically be excluded from the project build.

Electronic System Design Group

1.1.1.6 EXCLUDE Files

In addition to *exclude_keywords:* defined in *Configuration* provision to add exclusions on a path basis is included via the use of `EXCLUDE` files. If an `EXCLUDE` file is placed in a directory it will be parsed, line-by-line, for terms to exclude.

> **Warning:** The exclusion is very aggressive, a line containing only `v` would exclude all `.vhd`, `.v` and any file or path containing the character `v` from the search. Therefore it is recommended that file names are included on a line-by-line basis, including a `/` prefix to ensure no other files with similar names are excluded.

Note:

- Wildcard(s) `*`, `%` are **not** supported.
- Search terms are case sensitive.

Electronic System Design Group

1.1.1.7 Installation

`FPGAFlow` has been developed for use in a Linux based environment.

Prerequisites

Required Software

The following software is required:

–	Version
python	>= 2.7.14
pipenv	>= 2018.05.18
svn	>= 1.9.0
git	>= 1.8.0

Python Modules

Python Modules are handled by `pipenv` and stored in: `./scripts/python/fpgaflow/Pipfile`

Module	Version
breathe	4.11.0
colorlog	3.1.4
doxypypy	0.8.8.6
gitpython	2.1.11
logging	0.4.9.6
lxml	4.3.0
numpy	1.16.0
pygments	2.2.0
pyyaml	3.13
releases	1.6.1
sphinx-rtd-theme	0.4.2
sphinx	1.8.2
svn	0.3.46
vunit_hdl	4.0.8

Additional Modules

The modules below can be installed from `$REPO_ROOT/tools/<MODULE>/scripts/python/<MODULE>-<ORG>/dist/` using the `*.tar.gz` sdist source file.

pipenv_installer.py is provided in `$REPO_ROOT/tools/<MODULE>/scripts/python/utils` to aid the installation of additional modules from `$REPO_ROOT/tools`. See *Utils*.

Module	Version
xml2vhdl-ox	0.1.16

Installed Tools

`FPGAFlow` assumes at least one version of either of the following tools are installed on the system where `FPGAFlow` is executed from:

- Xilinx Vivado
- Intel PSG Quartus Prime

`FPGAFlow` assumes at least one version of each of the following tools are installed on the system where `FPGAFlow` is executed:

- Modelsim
- doxygen
- graphviz

Paths and licensing for these tools are configured in the `config.yaml` file passed to `FPGAFlow` using the `--config` argument (see *Configuration* for details).

Required Environment Variables

The following Environment Variables are required by `FPGAFlow`. These ensure that no personal locations are stored in version management systems (see: *required:*).

REPO_ROOT Sets the top-level root path where all projects/libraries will be checkout out to and processed from.

BITBUCKET_USERNAME The username used to access user controlled private `git` *cloud* hosted repositories.

GIT_USERNAME The username used to access user controlled private `git` *server* hosted repositories.

SVN_USERNAME The username used to access user controlled private `subversion` repositories.

PIPENV_SHELL Path to the executable for the shell `pipenv` should run.

SIM_PRE_COMPILED_LIBS_PATH The root path to precompiled Modelsim libraries.

Installation Instructions

1. Checkout/Clone `FPGAFlow` and `xml2vhdl` from their repositories:

```
git clone https://$BITBUCKET_USERNAME@bitbucket.org/mjroberts/fpgaflow.git $REPO_ROOT/
↪tools/fpgaflow
git clone https://$BITBUCKET_USERNAME@bitbucket.org/ricch/xml2vhdl.git $REPO_ROOT/
↪tools/xml2vhdl
```

2. Create or Modify Configuration YAML file to add tool paths, versions and license paths to: `$REPO_ROOT/tools/fpgaflow/scripts/python/fpgaflow/config/<YOUR_CONFIG>.yml` (see *Configuration* for details).
3. Create or Modify Settings YAML file to define project: `$REPO_ROOT/tools/fpgaflow/settings/<YOUR_SETTINGS>.yml` (see *Settings* for details).
4. Navigate to path where `FPGAFlow` will be executed.
5. Install the python virtual environment using `pipenv_installer.py` (see *pipenv_installer.py*) ensuring the version of python meets the *Prerequisites*.

```
pipenv_installer.py xml2vhdl-ox fpgaflow-stfc
```

6. Run `FPGAFLow` using `pipenv` in `shell` or `run`:

```
pipenv shell
python ./projectflow.py --config ./config/<YOUR_CONFIG>.yml --settings ../../../
↪settings/<YOUR_SETTINGS>.yml
## Or:
pipenv run python ./projectflow.py --config ./config/<YOUR_CONFIG>.yml --settings ../.
↪./../settings/<YOUR_SETTINGS>.yml
```

Links

- https://docs.pipenv.org
- https://vunit.github.io/documentation.html

Electronic System Design Group

1.1.1.8 Known Issues

1. Script fails to determine top-level dependency if the top-level is in the `boards` category.
2. Vendor IP Common Wrapper files are not parsed for `vendor_ip` without a `<IPSUBCAT>`.

Electronic System Design Group

1.1.1.9 Directory Layouts

Root Directory Processing

`FPGAFlow` operates on directory structures as defined in the *Configuration* YAML file. See *<CATEGORY>_layout:*.

The root directory of each layout is automatically determined by the script, if modifying the directory layouts the following rules **must** be applied to the layouts:

- The identifier to determine the root directory of `libraries` is the `<NAME>` tag.
- The identifier to determine the root directory of `boards` is the `<TOOL_VERSION>` tag.

Note: `vendor_ip` directory layout processing is handled separately.

`FPGAFlow` will parse the root directory for each enabled location and automatically add the first level of directories below the root directory. These are used to determine the locations of design files required to run designs through `FPGAFlow`.

First Level Directories

The following mappings are hard-coded in the script. See *projectmanager.ProjectDependency.top_dir_lookup*.

`constraints`

Directories matching any of following names are deemed to be the location of `constraints`:

- `constraints`
- `constr`
- `constrs`

`docs`

Directories matching any of following names are deemed to be the location of `docs`:

- `docs`
- `doc`
- `documents`

- documentation

netlists

Directories matching any of following names are deemed to be the location of `netlists`:

- netlists
- net
- nets

scripts

Directories matching any of following names are deemed to be the location of `scripts`:

- scripts
- script

settings

Directories matching any of following names are deemed to be the location of `settings`:

- settings
- setting

simulation

Directories matching any of following names are deemed to be the location of `simulation`:

- simulation
- sim
- sims

src

Directories matching any of following names are deemed to be the location of `src`:

- src
- srcs
- sources

Electronic System Design Group

1.1.1.10 Settings

The script uses settings using a YAML file passed using the `--settings` argument (see *Arguments*).

The description of the YAML file is described here.

```
python ./projectflow.py --settings <SETTINGS.YML>
```

Sections

<CATEGORY>:

The category of the project from list of *categories:* in the configuration file (see *Configuration*). Ths defines the repository layout of the project (see *<CATEGORY>_layout:*).

name:

Name of the project.

Note: The value here is used by *Tag Substitution* replacing the `<NAME>` tags in the *Configuration* file.

---

contrib_lib:

Name of the contributor's library. This defines the owner of the project and is used in the path to the location/structure of the project as it is stored in the externally hosted repository.

Note: *contributors:* defines a list of all of the contributors. The value here is used by *Tag Substitution* replacing `<CONTRIB_LIB>` tags in the *Configuration* file.

---

top_level:

Name of the top-level entity `HDL` component. This can be a `str` or `list` and the entity name should appear in the `HDL` filename. If no top-levels are supplied the script will search for files in library (determined from *name:*) for filenames which contain *name:* and one term from: `['top_', '_top']`. If none are found the script will exit.

Used by `projectflow` in conjunction with *board_id:* to generate the FPGA Vendor project name, if no *board_id:* is supplied:

```
<TOP_LEVEL>
```

where `<TOP_LEVEL>` is the first item in *top_level:* list (or the str of *top_level:* is supplied as a single entry).

The project name can be overwritten by *project_name_override:*

top_level_tb_entity:

Name of the top-level test-bench used when executing `VUnit` simulations.

vendor:

Name of the *FPGA* vendor the project is targeting.

Note: The value here is used by *Tag Substitution* replacing `<VENDOR>` tags in the *Configuration* file.

family:

Name of the *FPGA* family the project is targeting.

Note: The value here is used by *Tag Substitution* replacing `<FAMILY>` tags in the *Configuration* file.

tool_version:

Version of the *FPGA* vendor tool to use for the generation of the project.

Note: The value here is used by *Tag Substitution* replacing `<TOOL_VERSION>` tags in the *Configuration* file.

tool_mode:

Value to set the corresponding system environment variable to define the *FPGA* vendor tool operating mode. Sets either 32 or 64 bit operation.

device:

FPGA vendor defined device identifier to use for the generation of the project.

board_id:

A hex represented 32 bit number to insert into the *FPGA* project via a top-level generic named: `g_board_id`. This will be converted into a integer.

Listing 13: Generated Quartus settings file entry for board_id

```
set_parameter -name g_board_id 0
```

Listing 14: board_id generic

```
generic(
    g_board_id        : integer := 0
);
```

This can then be used in user logic using:

Listing 15: board_id user logic constant

```
constant c_board_id         : std_logic_vector(31 downto 0) := std_logic_vector(to_
↪unsigned(g_board_id, 32));
```

Note: This is a **board** only option.

board_timestamp_enabled:

The script will generate a Unix 32 bit timestamp for when the corresponding *FPGA* project is generated by the script. If this option is enabled using `True` the timestamp will be inserted via a top-level generic named: `g_board_timestamp`.

Listing 16: Generated Quartus settings file entry for board_timestamp

```
set_parameter -name g_board_timestamp 0
```

Listing 17: board_timestamp generic

```
generic(
    g_board_timestamp         : integer := 0
);
```

This can then be used in user logic using:

Listing 18: board_timestamp user logic constant

```
constant c_board_timestamp     : std_logic_vector(31 downto 0) := std_logic_vector(to_
↪unsigned(g_board_timestamp, 32));
```

Note: This is a **board** only option.

board:

Predefined hardware to target when generating the project. If supplied `projectflow` will use this value to generate the *FPGA* Vendor project name in the form:

```
<BOARD>_<TOP_LEVEL>
```

where `<BOARD>` is the *board_id:* name and `<TOP_LEVEL>` is the first item in *top_level:* list (or the str of *top_level:* is supplied as a single entry).

The project name can be overwritten by *project_name_override:*

board_settings:

Name of the settings file to parse when associating `board` with design.

Warning: Options in this file will overwrite options in the calling settings file.

project_name_override:

Option to override the automatically generated project name with a custom name.

sim_vendor:

Simulation vendor to use when running simulations.

sim_version:

Version of the simulation tool to use when running simulations.

Note: The value here is used by *Tag Substitution* replacing `<SIM_VERSION>` in the *Configuration* file.

sim_mode:

Value to set the corresponding system environment variable to define the simulation tool operating mode. Sets either 32 or 64 bit operation.

sim_init_script:

Name of the simulation initialisation script to usw when running simulations in GUI mode.

Note: Use: `$env(REPO_ROOT)` to avoid including user specific absolute paths in this file.

use_glbl:

If `True` adds glbl dependencies to `xilinx` simulations.

Note: This is a `xilinx` only option and is ignored for `altera` projects.

vunit_args:

Allows options from `settings.yml` YAML file to be passed to `VUnit` as if they are command-line arguments. For detailed information see: VUnit Documentation.

Listing 19: example `settings.yml` entry for `vunit_args`

```
vunit_args:
  version:                False
  list:                   False
  files:                  False
  log_level:              warning
  output_path:            simulation/vunit_output
  test_patterns:
    - None
  keep_compiling:         True
  num_threads:            2
  elaborate:              False
  verbose:                False
  xunit_xml:              xunit_report.xml
  coverage:
  unique_sim:             False
  exit_0:                 False
  no_color:               False
  quiet:                  True
  xunit_xml_format:       'bamboo'
  dont_catch_exceptions:  False
  export_json:            ''
  with_attributes:
  without_attributes:
  compile:                False
  fail_fast:              False
```

`version:`

Reports the version being used. See: vunit_cli

> **Warning:** If `True` This will report the version number and exit, not running `VUnit`. The script will raise a warning log if this option is set.

`list:`

Lists the test-cases being used *without* executing the simulation. See: vunit_cli

> **Warning:** If `True` This will list the test-cases being used by simulation and exit, not running `VUnit`. The script will raise a warning log if this option is set.

`files:`

Lists the HDL files being used *without* executing the simulation. See: vunit_cli

> **Warning:** If `True` This will list the files being used by simulation and exit, not running `VUnit`. The script will raise a warning log if this option is set.

log_level:

Sets the log-level for simulation. See: vunit_cli

output_path:

Sets the output_path for generated simulation files. See: vunit_cli

test_patterns:

List of test-cases or test-case patterns to simulate. – `None` will **not** run any test-cases.

keep_compiling:

If `True` keep compiling HDL if errors are found. Otherwise will exit on first error. See: vunit_cli

num_threads:

Number of threads to use running simulations. See: vunit_cli

elaborate:

Use elaborate. See: vunit_cli See: vunit_cli

verbose:

Run in verbose mode. See: vunit_cli

xunit_xml:

Name of the XML Report file. See: vunit_cli

coverage:

Use full coverage. See: vunit_cli

unique_sim:

Use unique simulations. See: vunit_cli

exit_0:

Exit with code `0` on errors. See: vunit_cli

no_color:

Disables colour logging when `True`. See: vunit_cli

xunit_xml_format:

Sets the format for the generated `xunit_xml` report file. Set to `'bamboo'`. See: vunit_cli

dont_catch_exceptions:

From VUnit Documentation: "Let exceptions bubble up all the way. Useful when running with "python -m pdb".". See: vunit_cli

export_json:

From VUnit Documentation: "Export project information to a JSON file."

`json` file will be generated in the `output_path:`. Leave blank to skip generation. See: vunit_cli

with_attributes:

List of test-cases or test-case to simulate which **do** have these attributes. Leave blank or have a single item of `''` to ignore.

From VUnit Documentation: "Only select tests with these attributes set"

without_attributes:

List of test-cases or test-case to simulate which **do not** have these attributes. Leave blank or have a single item of `''` to ignore.

From VUnit Documentation: "Only select tests without these attributes set" See: vunit_cli See: vunit_cli

compile:

When `` `True `` compiles `HDL` and exits. See: vunit_cli

fail_fast:

When `True` stop on first failed test. See: vunit_cli

repository_config:

Defines location in externally hosted source-code repository for the top-level project and each dependency required to execute *`projectmanager`*

git:

Listing 20: example `settings.yml` entry for `respository_config`

```
repository_config:
  git:
    <NAME>:
      enabled:             True
      vunit_default:       False
      url:                 <https://<GIT_BITBUCKET_URL/>
      path:                tools/xml2vhdl
      branch:              master
      tag:
      required_vhdl_libs: []
```

Warning: *Tag Substitution* has not been tested here. <> Should be replaced by actual values.

<NAME>:

The name of the project or project dependency.

enabled: Location is processed by the script if set to: `True`, otherwise skipped if set to: `False`.

vunit_default: Location is processed by *Vunit'* if set to: `True`, otherwise skipped by `Vunit` if set to: `False`.

url: Top-Level URL of externally hosted `git` repository hosted on `bitbucket`.

> **Note:** This is the `clone` URL from `bitbucket` with the `username@` portion removed. `username@` will be generated by *`projectmanager`* from `$BITBUCKET_USERNAME` system environment variable.

path: Path of location on the local file-system. Should match project path based on *categories:* and *<CATEGORY>_layout:* for the project or project dependency being defined.

branch: Specific `git branch` to use.

tag: Specific `git tag` to use.

> **Note:** `tag:` is **not** used in the context of using `FPGAFlow` to clone git repositories. The user is expected to manage `tags` and `branches` for development purposes after the repository has been cloned.

required_vhdl_libs: List of required VHDL libraries, names matching corresponding `<NAME>` entry in *git:* or *subversion:*.

> **Note:** `required_vhdl_libs:` is currently only used to ensure the VHDL libraries used by `XML2VHDL` script are included and enabled in `projectflow`.

subversion:

Listing 21: example `settings.yml` entry for `respository_config`

```
repository_config:
  subversion:
    <NAME>:
      enabled:            True
      vunit_default:      True
      url:                <https://<SVN_REPO_ROOT_URL/>
      path:               <CATEGORY>/<CONTRIB_LIB>/<NAME>
      subpath:            trunk
      rev:                HEAD
      required_vhdl_libs: []
```

Warning: *Tag Substitution* has not been tested here. <> Should be replaced by actual values.

<NAME>:

The name of the project or project dependency.

enabled: Location is processed by the script if set to: `True`, otherwise skipped if set to: `False`.

vunit_default: Location is processed by *Vunit'* if set to: `True`, otherwise skipped by `Vunit` if set to: `False`.

url: Top-Level URL of externally hosted `SVN` repository.

path: Path of location in repository. Should match project path based on *categories:* and *<CATEGORY>_layout:* for the project or project dependency being defined.

subpath: `'trunk'`, `'branches/<BRANCH_NAME>'` or `'tag/<TAG_NAME>'` for the project or project dependency being defined.

rev: Specific `SVN` revision or `HEAD`.

required_vhdl_libs: List of required VHDL libraries, names matching corresponding `<NAME>` entry in *git:* or *subversion:*.

documentation:

Listing 22: example `settings.yml` entry for `documentation`

```
documentation:
  project_title:          'Project Documentation Title'
  project_version:        '2019.1'
  project_author:         'Electronic System Design Group, Technology'
  project_org:            'Science and Technology Facilities Council'
  project_release:
  doxy_version:           1.8.14
  dot_version:            2.40.1
  sphinx_version:         1.8.2
  image_path:             common/images
  logo_path:              common/logo
  logo_name:              logo.png
  source_path:            common/source
```

(continues on next page)

(continued from previous page)

```
  language:                vhdl
  restricted_src:          False
  doc_clean:               True
  release_uri:             'https://bitbucket.org/mjroberts/fpgaflow/branch/release'
  issue_uri:               False
  releases_document_name: False
  doxy_mainpage:
    name:                  'README.md'
    autogen_pagelinks:     True
    autostruct:
      - 'readme_header.md'
      - 'readme_description.md'
```

project_title:

The Name of the Current Project Documentation.

project_version:

The Version Number of the Project Documentation in the form:

```
YYYY.NN
```

project_author:

The Project Documentation Author.

project_org:

The Project Documentation Organisation.

project_release:

The Project Documentation Release Tag.

doxy_version:

The Version of `doxygen` to use to build `VHDL` documentation.

Note: This version should be available on the host system and this value must be included in the *documentation: doxygen: support_tools:* within the configuration file (see *Configuration*).

---

dot_version:

The Version of `graphviz` used by `doxygen` when building documentation.

Note: This version should be available on the host system and this value must be included in the *documentation: dot: support_tools:* within the configuration file (see *Configuration*).

---

sphinx_version:

The Version of `sphinx` used when building documentation.

Note: This version should be available on the host system, either globally installed or included in the python virtual environment (see *Python Modules*) and this value must be included in the *documentation: sphinx: support_tools:* within the configuration file (see *Configuration*).

---

image_path:

The path of the image directory to include in the generated documentation. This path is relative to: `./docs`

logo_path:

The path of the logo directory to include in the generated documentation. This path is relative to: `./docs`

logo_name:

The name of the logo file, which must exist in the *logo_path:* to use in the generated documentation.

source_path:

The path additional source directory to include in the generated documentation. This path is relative to: `./docs`

language:

The main language used in the project.

python Documentation will be generated using `sphinx` only.

vhdl Documentation will be generated using `doxygen` from `VHDL` source code. This will then be included and referenced by `sphinx` which provides the top-level documentation for the project.

Note: This language must be included in the *documentation: supported_languages:* within the configuration file (see *Configuration*).

---

restricted_src:

When this option is set `True` `doxygen` is prevented from including source code in the generated documentation. Valid values are: `True` or `False`.

Note: `restricted_src:` defaults to `True` if the option is not included in the settings file.

---

doc_clean:

When this option is set `True` the documentation build paths are deleted prior to building the project documentation. Valid values are: `True` or `False`.

release_uri:

Using releases to handle changelogs. See releases_usage. This is currently configured for `bitbucket` so `/%s` is appended to the end of the supplied URL automatically.

Note: `release_uri:` is optional, if not used it should be set to `False` in the settings file.

---

issue_uri:

Using releases to handle changelogs. See releases_usage.

Note: `issue_uri:` is optional, if not used it should be set to `False` in the settings file.

---

releases_document_name:

Using releases to handle changelogs. See releases_usage. If not defined here it will default to `changelog.rst`.

Note: `releases_document_name:` is optional, if not used it should be set to `False` in the settings file.

---

doxy_mainpage:

These options define the `doxygen` mainpage used to reference any pages generated by `doxygen` from source code, which is then referenced by the top-level documentation generated by `sphinx`.

name: Name of the mainpage to generate.

autostruct: A list of files, which should be placed in the projects *docs/doxygen/source* directory, which should be added to the mainpage used by `doxygen`. The order of the list defines the order the contents of these files will by added to the mainpage.

autogen_pagelinks: Configures if the documentation generated by doxygen should be parsed to (re)generate links to pages to include on the mainpage. Valid values are: True or False.

This will automatically generate a file named: <lowercase_name>_links.md where <lowercase_name> matches the name of the mainpage to generate defined by name:.

Note: When generating documentation doxygen uses either the @page uniquename or \page uniquename tags to define the name of the generated page. By using the convention of suffixing page to the uniquename so that it becomes: uniquenamepage it is possible for the docflow.py script to automatically add these pages as a list of links on mainpage generated by doxygen.

This option will cause doxygen to generate documentation twice, once to generate the pages, the second time to reference the pages on it's mainpage.

Electronic System Design Group

1.1.1.11 Utils

The following utilities are provided with FPGAFlow to aid the management of FPGAFlow on the users system.

By adding the following location to the user's path these can be accessible from anywhere on the system:

Listing 23: example entry in ~/.cshrc

```
setenv REPO_UTILS $REPO_ROOT/tools/fpgaflow/scripts/python/utils
set path = ($REPO_UTILS $path)
```

pipenv_installer.py

pipenv_installer.py provides a basic method to install the required *Python Modules* as well as *Additional Modules* from $REPO_ROOT/tools without placing the addition tools in the resulting Pipfile.

Note: pipenv_installer.py will backup existing Pipfile and Pipfile.lock files.

Creating Virtual Environment

By running this command, pipenv_installer.py will remove any existing virtual environment and create a new fresh environment containing **only** the *Python Modules*. This will be represented by the resulting Pipfile.

Listing 24: pipenv_installer.py usage

```
# Installation:
pipenv_installer.py
```

Creating Virtual Environment with Additional Modules

By running this command, `pipenv_installer.py` will remove any existing virtual environment and create a new fresh environment containing the *Python Modules*. **and** the *Additional Modules*. The resulting `Pipfile` will only represent the *Python Modules* with **no** reference to the *Additional Modules*.

The most recent version of the additional module will be installed.

Listing 25: `pipenv_installer.py` additional module usage

```
# Installation with Additional Modules:
pipenv_installer.py [<MODNAME>-<ORG> [<MODNAME>-<ORG>] ..]
```

Note: Additional Modules **must** be in the form>: `<MODNAME>-<ORG>`.

Updating Additional Modules

By running this command, `pipenv_installer.py` will uninstall the specified *Additional Modules* from the virtual environment and install the most recent version of the additional module.

The resulting `Pipfile` will only represent the *Python Modules* with **no** reference to the *Additional Modules*.

Listing 26: `pipenv_installer.py` additional module update usage

```
# Updating Additional Modules:
pipenv_installer.py update [<MODNAME>-<ORG> [<MODNAME>-<ORG>] ..]
```

Note:

- Additional Modules **must** be in the form>: `<MODNAME>-<ORG>`.
- Additional Modules can also be installed to an existing virtual environment using ths method.

Electronic System Design Group

1.1.1.12 Vendor IP

Description

`vendor_ip` provides a mechanism to manage *FPGA* specific IP components which are generated via tools provided as a part of the vendor design suite. The `vendor_ip` can then be included in project via the corresponding *Settings* file.

Configuration File

`vendor_ip_layout:`

Listing 27: example of `config.yml` entry for `vendor_ip` layout section

```
vendor_ip_layout:
  - vendor_ip:
    - <VENDOR>:
      - <TOOL_VERSION>:
        - common_wrapper_files:
          - <FAMILY>:
            - <IPCAT>:
              - <IPSUBCAT>:
                - <NAME>:
                  - constraints:
                  - docs:
                  - src:
                    - vhdl:
                    - vhdl_packages:
                    - xml:
        - <DEVICE>:
          - <IPCAT>:
            - <IPSUBCAT>:
              - <NAME>:
                - constraints:
                - docs:
                - src:
                    - ip:
                    - vhdl:
                    - vhdl_packages:
                    - xml:
```

<IPCAT>:

interfaces

Interfaces to externally connected devices.

debug

Components used for debug tasks.

clock_management

PLLs and DCMs.

memory

FIFOs.

<IPSUBCAT>:

Subcategories to logically group `<IPCAT>`.

interfaces

- `phy`
- `jesd204b`
- `external_memory`

<NAME>:

IP name.

Settings File

Listing 28: example of `settings.yml` entry for `ip` section

```
ip:
  <IPCAT>:
    <IPSUBCAT>:
      <IP_NAME>:
        enabled:                                    <True|False>
        library:                                    <lib_name>
        wrapper_file:                               <wrapper_name>_wrapper
  <IPCAT>:
    <IPSUBCAT>:
      enabled:                                      <True|False>
      library:                                      <lib_name>
      wrapper_file:                                 <wrapper_name>_wrapper
```

Common Wrapper Files

Common wrapper files allow wrapper files to be used across multiple devices within the same *FPGA* family. The `./src/vhdl` directories in both `<DEVICE>` and `common_wrapper_files` locations are parsed for a corresponding `.vhd` source file matching the name defined by `wrapper_file:`. If the named file exists in *both* locations the one located in the `<DEVICE>` path will take precedence over the one located in the `common_wrapper_files` path.

The script will generate the search paths for wrapper files automatically, using both the `<IP_NAME>` and `<wrapper_name>` portion of the `wrapper_file:` name as the value for `<NAME>` in the constructed `vendor_ip_layout` structure.

Electronic System Design Group

1.1.1.13 VUnit Integration

`FPGAFlow` uses `VUnit` to handle `HDL` libraries and dependencies for FPGA Vendor project generation and the simulation of `HDL` Modules. Configuration of `VUnit` is handled by the *vunit_args:* section of the *Settings* file.

Standard Libraries

Standard libraries are located in a precompiled area, so that they only require compiling once per version of Modelsim and FPGA tool version.

Currently the script compiles standard libraries for all families and libraries. However for each library to be added to the generated `VUnit` object it needs to be included in the corresponding list of simulation libraries under *xilinx:* or *altera:* in the *Configuration* file.

Standard libraries are generated in:

```
$SIM_PRE_COMPILED_LIBS_PATH/<VENDOR>/<TOOL_VERSION>/<SIM_VENDOR>_<SIM_VERSION>/
↪standard
```

Note: The values here are used by *Tag Substitution* replacing the `<UPPERCASE>` tags with corresponding values from the *Settings* file.

Project IP Libraries

Project specific IP libraries are generated in:

1.1.2 Indices and Tables:

- genindex
- modindex
- search

Python Module Index

Index

G

I

L

M

O

P

R

S

T

V

W