Design Flow Quick Start

Contents

1. Introduction

The goal of the DesignFlow Python tool is to provide a replacement for Makefiles, especially the complex system that has been developped for alliance-check-toolkit. It is build upon DoIt (DoIt).

1.1 Task vs. Rules

Both as a tribute to **`Makefile`**, to avoid ambiguties with DoIt and to remember that they are task *generators*, the classes defined to create tasks for the design flow are called `rules`.

1.2 A Warning About Determinism

There is a very important execution difference from a **`Makefile`**. In a **`Makefile`** each rule command is executed in a a separated process, so information is effectively passed through files which are written then read from disk. But in DoIt we are running inside *one* Python process, so while using Coriolis and the Hurricane database, all informations stays *in memory*. Files are driven, but *not re-read* as the database will use the datas already present in memory.

This is not whitout consequences about determism. Let's look at two different scenarii.

1. We run straight from the RTL to the layout, using the rule/task sequence:

   ```
   Yosys => design.blif => blif2vst => design.vst => PnR => design.gds
   ```

   In this case, while `design.vst` is written on disk, the `PnR` stage will not re-read the `vst` file and directly access the data in memory.

2. Run in two separated steps, first we create the `vst` file:

   ```
   Yosys => design.blif => blif2vst => design.vst
   ```

   Then, we perform the `PnR`:

   ```
   design.vst => PnR => design.gds
   ```

   In this case, as the DoIt processess has been restarted between the two tasks, the `PnR` stage *will* read the `vst` file.

The determism in Coriolis is ensured through the unique identifiers of the objects, attributed in creation order. So between thoses two scenarii, the identifiers will change and so the algorithm results. The differences should be minor as the identifiers are used as a *last ditch* test to sort between two objects which cost functions are exactly equal, nevertheless, it will occur.

Note

CORIOLIS is deterministic, meaning that each scenario will always give the same result. The difference is truly *between* scenarii.

2. Using The Design Flow

2.1 Locating the Various Parts

One of the most tricky part of setting up the design flow is to locate where the various components are. The script needs to be able to find:

1. Coriolis, binaries & libraries. This depends widely of your kind of installation and system. The helper script `crlenv.py` supplied both in `alliance-check-toolkit` and Coriolis may help you there. It looks in all the standard locations (that it is aware of) to try to find it.

Note

Usually, ALLIANCE is installed in the same tree as CORIOLIS, so it's setup can be deduced from it.

2. The configurations files for the technology to be used. Here again, the `designflow.technos` module provides you with a set of pre-defined configurations for open sources technologie shipped with Coriolis. For unsupported ones, you may write your own, it should perform the whole initialization of the Coriolis and Hurricane database.
3. Optionnaly the `alliance-check-toolkit`.

2.2 Basic Example of dodo File

This example can be found in `alliance-check-toolkit`, under `benchs/arlet6502/sky130_c4m`. Here

```
from designflow.technos import setupSky130_c4m

setupSky130_c4m( checkToolkit='../../..'
               , pdkMasterTop='../../../pdkmaster/C4M.Sky130' )

DOIT_CONFIG = { 'verbosity' : 2 }

from designflow.pnr       import PnR
from designflow.yosys     import Yosys
from designflow.blif2vst import Blif2Vst
from designflow.alias     import Alias
from designflow.clean     import Clean
PnR.textMode = True

from doDesign import scriptMain

ruleYosys = Yosys   .mkRule( 'yosys', 'Arlet6502.v' )
ruleB2V   = Blif2Vst.mkRule( 'b2v'  , [ 'arlet6502.vst'
                                      , 'Arlet6502.spi' ]
                                    , [ruleYosys]
                                    , flags=0 )
rulePnR   = PnR     .mkRule( 'pnr'  , [ 'arlet6502_cts_r.gds'
                                      , 'arlet6502_cts_r.spi'
                                      , 'arlet6502_cts_r.vst' ]
                                    , [ruleB2V]
                                    , scriptMain )
ruleCgt   = PnR     .mkRule( 'cgt' )
ruleGds   = Alias   .mkRule( 'gds', [rulePnR] )
ruleClean = Clean   .mkRule()
```

You can run it with:

```
ego@home:sky130_c4m> ../../../bin/crlenv.py -- doit list
b2v          Run <blif2vst arlet6502 depends=[Arlet6502.blif]>.
cgt          Run plain CGT (no loaded design)
clean_flow   Clean all generated (targets) files.
gds          Run <Alias "gds" for "pnr">.
pnr          Run <pnr arlet6502_cts_r.gds depends=[arlet6502.vst,Arlet6502.spi]>.
yosys        Run <yosys Arlet6502.v top=Arlet6502 blackboxes=[] flattens=[]>.
ego@home:sky130_c4m> ../../../bin/crlenv.py -- doit pnr
ego@home:sky130_c4m> ../../../bin/crlenv.py -- doit clean_flow
```

Let's have a detailed look on the various parts of the script.

A. **Choosing the technology** Here, we load the predefined configuration for SkyWater 130nm. We also have to give the location of the `alliance-check-toolkit`, it may be relative or absolute.

   If you want to use another one, it up to you to configure Coriolis at this point by any means you see fit.

```
from designflow.technos import setupSky130_c4m

setupSky130_c4m( checkToolkit='../../..'
               , pdkMasterTop='../../../pdkmaster/C4M.Sky130' )
```

B. **Loading the various task/rule generators that we will use**, from the `designflow` namespace. The rules are named from the tool they encapsulate.

```
from designflow.pnr      import PnR
from designflow.yosys    import Yosys
from designflow.blif2vst import Blif2Vst
from designflow.alias    import Alias
from designflow.clean    import Clean
PnR.textMode = True
```

C. **Creating the rule set.** Each rule generator as a static method `mkRule()` to create a new task. The three first parameters are always:

   1. The name of the task (the `basename` for DoIt).
   2. A target or list of targets, must be files or `pathlib.Path` objects.
   3. A dependency or list of dependencies, they can be files, `pathlib.Path` objects, or other tasks. We can see that the `Blif2Vst` rule uses directly the `Yosys` one (the input file will be the *first* target of the `Yosys` rule).
   4. Any extra parameters. A set of flag for `Blif2Vst`. The `PnR` rule takes an optional callable argument, *any* callable. In this case we import the `scriptMain()` function from `doDesign()`.

   There are two more special rules:

   - `Alias`, to rename a rule. It this case `gds` is defined as an alias to `PnR` (because it generate the gds file).
   - `Clean` to create a rule that will remove all the generated targets.

Note

The clean rule is named `clean_flow` because DoIt already have a `clean` arguments which would shadow it.

```
PnR.textMode = True

from doDesign import scriptMain

ruleYosys = Yosys    .mkRule( 'yosys', 'Arlet6502.v' )
ruleB2V   = Blif2Vst.mkRule( 'b2v'   , [ 'arlet6502.vst'
                                       , 'Arlet6502.spi' ]
                                     , [ruleYosys]
                                     , flags=0 )
rulePnR   = PnR      .mkRule( 'pnr'   , [ 'arlet6502_cts_r.gds'
```

```
                                                , 'arlet6502_cts_r.spi'
                                                , 'arlet6502_cts_r.vst' ]
                                              , [ruleB2V]
                                              , scriptMain )
ruleCgt   = PnR     .mkRule( 'cgt' )
ruleGds   = Alias   .mkRule( 'gds', [rulePnR] )
ruleClean = Clean   .mkRule()
```

3. Rules's Catalog

3.1 Alliance Legacy Tools

Support for the Alliance legacy tools. They are run through sub-processes. For more detailed documentation about those tools, refer to their **man** pages.

1. `Asimut`, vhdl simulator.
2. `Boog`, logical synthesys. Map a vhdl behavioral description to a standard cell library (works with `boom` & `loon`).
3. `Boom`, behavioral description optimizer (works with `boog` & `loon`).
4. `Cougar`, symbolic layout extractor.
5. `Dreal`, real layout (gds, cif) editor.
6. `Druc`, symbolic layout drc.
7. `Flatph`, flatten a layout, fully or in part.
8. `Genpat`, pattern generator (for use with `Asimut`).
9. `Graal`, symbolic layout editor.
10. `Loon`, netlist optimizer for surface and/or delay (works with `boom` & `boog`).
11. `Lvx`, netlist comparator (*Layout Versus Extracted*).
12. `S2R`, symbolic to real translator (to gds or cif).
13. `Vasy`, Alliance vhdl subset translator towards standard vhdl or Verilog.

3.2 Current Tools

1. `Blif2Vst`, translate a **blif** netlist (Yosys output) into the Alliance netlist format **vst**. This is a Python script calling Coriolis directly integrated inside the task.
2. `PnR`, maybe a bit of a misnomer. This is a caller to function that the user have to write to perform the P&R as he sees fit for it's particular design.
3. `Yosys`, call the Yosys logical synthesyser. Provide an off the shelf subset of functionalities to perform classic use cases.

3.3 Utility Rules

1. `Alias`, create a name alias for a rule.
2. `Clean`, remove all the generated targets of all the rules. The name of the rule is `clean_flow` `to not interfer with the |DoIt| clean arguments. Files not part of any rules targets can be added to be removeds. Then, to actually remove them, add the ``--extras` flag to the command line.

   ```
   ego@home:sky130_c4m> ../../../bin/crlenv.py -- doit clean_flow --extras
   ```

3. `Copy`, copy a file into the current directory.

3.4 Rule Sets

For commonly used sequences of rules, some predefined sets are defined.

1. `alliancesynth`, to apply the logical Alliance logical synthesis set of tools. From vhdl to optimized **`vst`**. The set is as follow:

   ```
   x.vbe => boom => x_boom.vbe => boog => x_boog.vst => loon => x.vst
   ```

   An additional rule using `vasy` is triggered if the input format is standard vhdl.
2. `pnrcheck`, complete flow from Verilog to symbolic layout, with drc and lvx checks. Uses Yosys for synthesis.
3. `routecheck`, perform the routing, the drc and lvx check on an already placed design. Use symbolic layout.