Introduction

Stylus is a Cadence tool designed to help designer define a proper CAD tool flow from behavioral to GDSII

The **flowtool** is the core function for

- Automatically generating a set of scripts, with PLACE_HOLDERS, for designers to configure with design and foundry references
- Executing the tool flow, with many knobs for controlling which elements are run, and for inspecting and debugging the sequence of Cadence tools that are run

Cadence tools include:

- Genus: Behavioral synthesis tool
- Innovus: Place&Route tool
- Tempus: Static timing analysis tool
- Voltus: Full-chip electromigration, IR drop and power analysis tool
- Quantus: Parasitic extraction tool
- Conformal: Formal verification via equivalence checking tool

FlowTool Definitions

Flows are defined with

- **Flow steps** (**flow_step**): Associate a set of (tcl) commands to a label

  Flow steps are created with ***create_flow_step***

- **Flow objects**: Flow objects specify a set of actions for a CAD tool to execute

  Typically, one of the Cadence tools are referred with '-tool genus'

  Non-Cadence tools are allowed using the '-tool_options' switch

- **Flow scheduling**: Adds additonal flow actions to the current flow

  The keyword ***schedule_flow*** is used inside a *create_flow_step* definition

Stylus supports generating a generic flow environment with *write_flow_template*

write_flow_template -list gives a list of templates that can be generated for configuring new flows for a design and/or foundry

The command ***write_flow_template*** MUST be run within a Cadence UI

It generates YAML files, e.g., *flow.yaml* and TCL files, e.g., *design_config.tcl*

FlowTool Components

You MUST run the command within Cadence *genus*

```
@genus:root: 1> write_flow_template -tools "genus" -describe stylus
Features for flow templates matching 'stylus'

Template stylus
---------------
Provider    : tool
Max version : 1
Description : Standard flow for block implementation defined using YAML and TCL
This is the default template
+----------------+-------------------------------------------------------------------------+---------+-------+
| Feature        | Description                                                             | Default | Valid |
+----------------+-------------------------------------------------------------------------+---------+-------+
--- the following features are mutually exclusive (dft_style group)
| dft_compressor | Add flow support for scan chains with compression insertion             |         | 0 1 {} |
| dft_simple     | Add flow support for scan chain insertion                               |         | 0 1 {} |
---
| dynamic_view   | single dynamic analysis_view to activate                                |         |        |
| ff_setup       | Enable reading design config from legacy FF flow setup.tcl file         |         | 0 1 {} |
| flow_express   | Enable express synthesis and implementation flow                        |         | 0 1 {} |
| hold_views     | list of hold analysis_views to activate                                 |         |        |
| leakage_view   | single leakage analysis_view to activate                                |         |        |
| report_clp     | Add CLP dofile generation and checks to the flow                        |         | 0 1 {} |
--- the following features are mutually exclusive (report_style group)
| report_defer   | Defer report generation                                                 |         | 0 1 {} |
| report_inline  | Run report generation as part of parent flow versus schedule_flow       |         | 0 1 {} |
| report_none    | Disable report generation                                               |         | 0 1 {} |
---
| report_lec     | Add LEC dofile generation and checks to the flow                        |         | 0 1 {} |
| setup_views    | list of setup analysis_views to activate                                |         |        |
--- the following features are mutually exclusive (synth_style group)
| synth_hybrid   | Physically aware synthesis flow with logical final optimization         |         | 0 1 {} |
| synth_ispatial | Physically aware synthesis flow with ispatial final optimization        |         | 0 1 {} |
| synth_physical | Full physically aware synthesis flow                                    |         | 0 1 {} |
| synth_spatial  | Physically aware synthesis flow with spatial final optimization         |         | 0 1 {} |
---
| use_common_db  | Enable using common DB format for synth and implementation flows        |         | 0 1 {} |
+----------------+-------------------------------------------------------------------------+---------+-------+
```

YAML

YAML is a standard string processing language for describing data serialization, i.e., a series of data processing tasks, using high-level, user-friendly constructs

It describes the flow process in an **outline** style format

- Comments are indicated with '#'
- Key-value pairs are created using ':'
- Collections of actions are created using '-'

Note that YAML uses spaces to create dependencies among statements, like python

Stylus creates flow control extensions in YAML using:

- CMD_<NAME>: Automatically encapsulates content in a flow_step with NAME
  CMD_write_def: write_def out.def
- FILE_<NAME>: Automatically encapsultes file contents in a flow_step via NAME
  FILE_floorplan: floorplan.tcl
- SCHEDULE: Method to schedule new flows in YAML, similar to schedule_flow
  SCHEDULE: -flow report_prects

YAML

Portion of *flow.yaml* in scripts directory generated by *write_flow_template*

```
flows:

#-------------------------------------------------------------------------------
# synthesis
#-------------------------------------------------------------------------------
  synthesis:
    args: -tool genus -owner cadence -skip_metric -tool_options -disable_user_startup
    features:
    steps:
      - syn_generic:
          args: -owner cadence
          features:
          steps:
            - block_start:
            - init_elaborate:
            - init_design:
                args: -owner cadence
                features:
                steps:
                  - read_mmmc:
                  - read_physical:
                  - read_hdl:
                  - read_power_intent:
                  - run_init_design:
                  - read_def:
                      enabled: "synth_spatial || synth_ispatial || synth_physical || synth_hybrid"
            - init_genus:
            - set_dont_use:
            - init_dft:
            - commit_power_intent:
            - create_cost_group:
            - run_syn_generic:
            - block_finish:
            - SCHEDULE:
                args: -flow report_synth -include_in_metrics
                enabled: "!report_none && !report_inline && !report_defer"
      - syn_map:
          ...
```

YAML

Note that **syn_generic** and **syn_map** are *genus* commands

The items '-' are collections of actions to execute for these *genus* commands

The scripts file, ***setup.yaml***, ***flow_config.tcl*** and ***design_config.tcl*** contain most of the **PLACEHOLDER** keywords that need to be filled in for a new design/foundry

In ***setup.yaml***, the following are needed:

- **library_sets**: List of *xxx.lib* files
- **opconds**: Process, temperature, voltage files
- **timing_conditions**: *library_set* objects to associate to a *timing_condition*
- **rc_corners**: *qrc_tech* files for RC corners
- **delay_corners**: *early* and *late* RC corner and *timing_condition* names
- **constraint_modes**: Constraint files to associate with a *constraint_mode*
- **analysis_views**: name of *contraint_mode* and *delay_corner* + others for view
- **lef_files/oa_ref_libs/oa_search_libs**: Files for *read_physical* command
- **init_power_nets/init_ground_nets** + others: Net names to assign
- **design_name/design_process_node/design_flow_effort**, etc: Design attributes
- **add_fillers_cells/add_tieoffs_cells**, etc: Names of std cells

FlowTool Commands

Validate the *setup.yaml* by running flowtool, but only for *init_design*

flowtool -predict summary -flow init_design

flowtool takes many arguments:

- **flowtool -run_tag synthesis_only -to syn_opt.block_finish**
- **flowtool -from prects.block_start -to prects.block_finish -predict verbose**

From within the Cadence UIs, you can run flows:

- **run_flow -step init_design**

To run the entire flow, just use:

- **flowtool**

Viewing the results:

reports directory: Run your browser and point it to the *qor.html*

reports directory: For *genus*, text reports in *syn_generic*, *syn_map* and *syn_opt*

json: Use ***write_metric -format json -file run1.json***

logs directory: Use *vim* to view, look for 'errors' and 'warnings'