Standardized design environment and methodologies enable simultaneous implementation of 28nm designs with a single flow

Cyrille Thomas

Cyrille.Thomas@bull.net

Bull SAS

Paris, France

www.bull.com

ABSTRACT

Bull SAS is a leading developer of high performance computing designs in Europe. Using a configurable, pre-validated standardized flow and integrated GUIs in Synopsys' Lynx Design System, we were able to quickly deploy a new 28-nm RTL-to-GDSII flow. This environment allows our geographically distributed engineers to simultaneously work on multiple projects targeting different technologies. In addition, the new intuitive visualization capabilities in Lynx provide us with dynamic access to project metrics and trends such as execution profile reports, licences usage, metrics vs metrics, enabling us to make the data-driven decisions on design constraint changes and designer re-allocation to help meet schedule. In this paper we will talk about few of the features in Lynx such as job distribution optimization, 3rd party tool adoption and benefits such as improved ramp-up that helped us achieve our goals on a recent tapeout.

Table of Contents

Table of Figures

Introduction

Context

The growing number of projects, the shortening development time, the new physical issues introduced in 28nm-and-below nodes, the cost of masks, respins but also licenses, the number of geographic locations, the need for increased productivity, the disappearance of teams dedicated to CAD, all these challenges involve a robust and standardized physical design flow, easy to maintain and in which it is easy to track the specifics of a project or even a block. It is also vital to be able to quickly retrieve relevant information about status or trend over time for a given chip. We explain in this paper which methodologies we chose to meet these needs.

Our starting point is the workflow proposed by Synopsys (Lynx Design System). We illustrate how we changed the reference scripts to add an additional degree of flexibility, how we proceeded to add our specificities while allowing Lynx flow updates as easily as possible. We also discuss how it is possible to use Lynx Design System as a decision support through the use of a large number of metrics stored natively in a PostgresSQL database. We present some details about the use of revision control and job distribution tools or the integration of third party tools. We finish by listing the improvements that we can still make on this flow used by Bull.

The target audience would be engineers responsible for methodology and physical implementation flow and managers looking for efficient way to track status and trends in a multi-projects, multi-sites context as well as fast path to move to advanced nodes with tight schedule and resources.

Our company

Bull is an Information Technology company, dedicated to helping Corporations and Public Sector organizations optimize the architecture, operations and the financial return of their Information Systems and mission-critical related businesses.

Bull focuses on open and secure systems, and as such is the only European-based company offering expertise in all the key elements of the IT value chain.

As a computer maker, Bull has built one of the most powerful supercomputers in Europe and have already delivered 3 petaflop systems in the world.

1. Chip Design Challenges at Bull

The Servers department in the Innovative Products division includes 70+ ASIC engineers covering logical design, physical design as well as verification.

Most of our ASICs are High Performance Computing chips, aka HPC chips, using advanced nodes (28-nm for the current ones). Frequencies are in the 800MHz-1GHz range. Some of the chips can contain more than 200MB of SRAM or can instantiate Serdes running up to 25Gb/s. Die sizes can be as large as 20mm X 30mm and power consumption can reach 160W.

1.1. The Old way of doing things

Some years ago, the number of engineers in logical design, verification and physical design teams was three times lesser than they are now. We were working on a single project at a time, in a single location, in France.

1.1.1. Reference flow

The Physical design flow (mainly Place & Route, chip finishing, Static Timing Analysis and physical verification) was based on Synopsys Reference Methodology scripts, with some modifications taking into account our experience, handling bugs with corresponding workarounds but also adding tips and tricks in order to improve QoR. All the scripts were located in a "flowref" directory with some "pnr", "sta", "drc" or "lvs" subdirectories.

When some scripts were updated by the engineer responsible for the flow, an email would be sent to the "blocks" and "top-level" physical designers in order to warn them about this new situation and it was their responsibility to update their files. Of course, their scripts were quite different from the "flowref"ones due to the specificities of each block (number of clocks, rectilinear L-shaped blocks needs to use some additional commands, etc.) In these conditions, it was not easy to take into account an update of the reference flow. In the same way, the definition of the IP to use for a given block was also "local" (defined in the scripts related to the block). New memories delivery (impacting all the blocks) required manual update to files for each block. The number of blocks and number of physical designers made it quite manageable but it was risky and hard to ensure that everyone took into account these important updates.

1.1.2. Metrics

Metrics like WNS (Worst Negative Slack), TNS (Total Negative Slacks) or NVP (Number of Violating Paths) used for blocks status and management reporting were painful to collect at that time. This information could be found in the qor_snapshot report for a given block but we still needed to put all those data together in a spreadsheet. It was time-consuming for everyone to prepare this data on a regular basis.

1.2. New challenges

In the last few years, the number of projects in parallel, number of designers but also number of site locations for verification and physical design teams have been multiplied by a factor of 3. We now have :

- 3 concurrent projects,
- 20 front-end engineers, 20 verification engineers and 20 physical designers
- A growing physical design team, spread across multiple sites: Paris, Nice, Grenoble in France, and UK in Europe
- A mix of different nodes: primarily 40-nm and 28-nm
- Much more aggressive schedules
- Large amounts of data to monitor
- Lack of schedule predictability

In these conditions, the previous approach was not acceptable anymore. How could we ensure that everyone is using the latest flow updates, and good versions for tools and IP? We needed a global approach with an easy way to track flow deviations from blocks/designers and to obtain metrics.

The Lynx Design System seemed to be a good solution to address these challenges.

2. Lynx Design System At a Glance

The Lynx Design System is a chip design system environment. It consists of 4 major components.

2.1. Major Componants

2.1.1. Production Flow

The production flow in Lynx is a complete RTL-to-GDSII flow supporting multiple design styles such as hierarchical designs, Unified Power Format designs (hierarchical and non-hierarchical) and Multiple-Instantiated Modules (MIM). The flow consists of ready-to-use TCL scripts which show the recommended methodology for a specific product and release. The scripts are updated with every update of the underlying Galaxy tools so that it always takes advantage of new features in the tools.

This flow enables tuning and customization in multiple ways. It has the capability to source custom scripts before (pre-script) and after (post-script) the main script of a given task (which script contains a main command such as *place_opt* for standard cells placement task, *clock_opt* for clock tree synthesis task, etc). By adding your commands to these pre-script and post-script sections, you can customize the flow without impacting the reference scripts. An example would be to run additional rounds of *psynopt* (which performs incremental placement optimization on a design) after main *place_opt* by creating a script fragment with *psynopt* calls as a post-script for the *icc_place_opt* task.

Each script also has high-level parameters/variables called TEVs (Task Environment Variable) that can control or configure the script's operation. An example of a TEV would be a variable that contains the flags to pass to *place_opt* command. Instead of changing the reference script when you want to change the *place_opt* options, you just pass the right value via TEV(place_opt_options) by setting it to "*-congestion –optimize_dft*" for instance.

We can also plug in third-party tools into the flow quite easily (more details on how we integrated third-party tools into the flow is described in section 0). One of the advantages of the way the flow has been designed is that it can support multiple technologies while still being a single uniform flow. This is made possible by compartmentalizing the technology-specific settings and scripts as technology plugins that are loaded or used by the Lynx production flow.

2.1.2. Technology Plugins

In Lynx, technology plugins contains technology-specific scripts. Technology-specific scripts are scripts that contain configuration settings or run commands that are specific to a given technology. In this way, the production flow can support a given technology just by loading the appropriate technology plugin. One example is a script to insert TCD cells. A 65nm design would not have TCD cells but a 40nm design must. Even within 40nm designs, the name and insertion configuration may be different across foundries. By making the script to insert TCD cells a technology-specific script, the production flow can insert TCDs just by running this script. The flow does not need to be modified for every technology. Another example would be the list of DBs, Milkyways and other library files required by the flow. This list is stored in a techonology-

specific file called *common.tcl*. When the flow runs, it loads the libraries that *common.tcl* says it should load.

As part of the technology plugins, there is also Library Quality Assurance plugins to validate library files for issues that may cause design problems such as pin mismatches, cell mismatches and other issues.

2.1.3. Runtime Manager

The Runtime Manager is a GUI (and shell) for editing and viewing flows graphically, creating and running blocks, tuning block parameters, viewing reports and logs, interactive debug and pretty much do everything you need to get a design from RTL to tapeout. You can run this via GUI or shell mode. It needs two main folders to work correctly: "*scripts_global*" containing all the flow scripts and "*blocks*" containing all the data (Milkyway, reports but also block-specific scripts). This structure is created when installing Lynx environment, with some templates and blocks to be used a starting point. We will discuss the way to handle these two major directories later in the document.

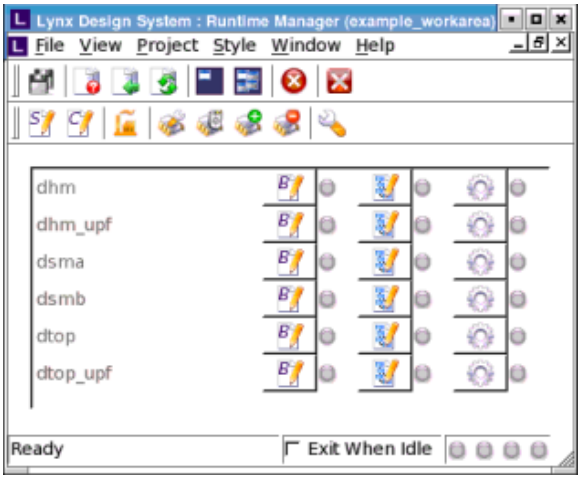


Figure 1: RunTime Manager main window

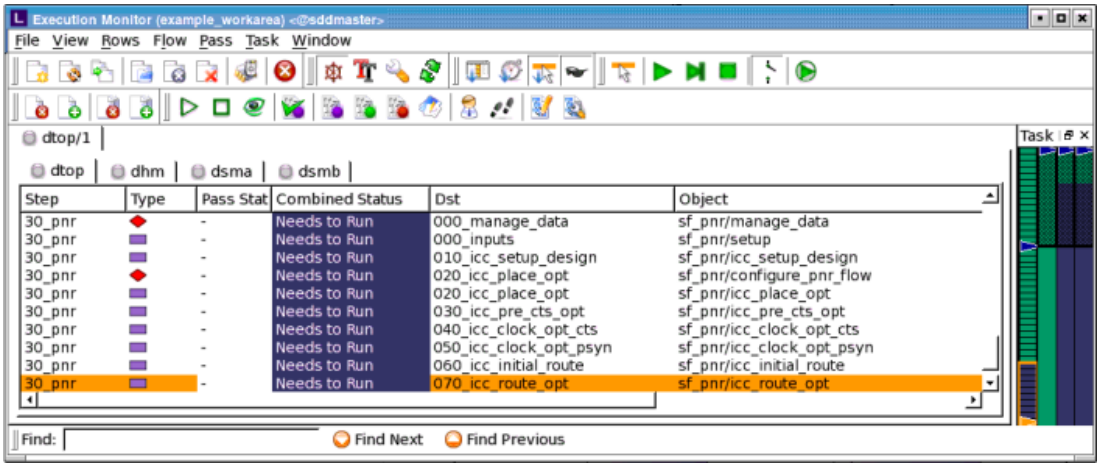


Figure 2: Execution Monitor

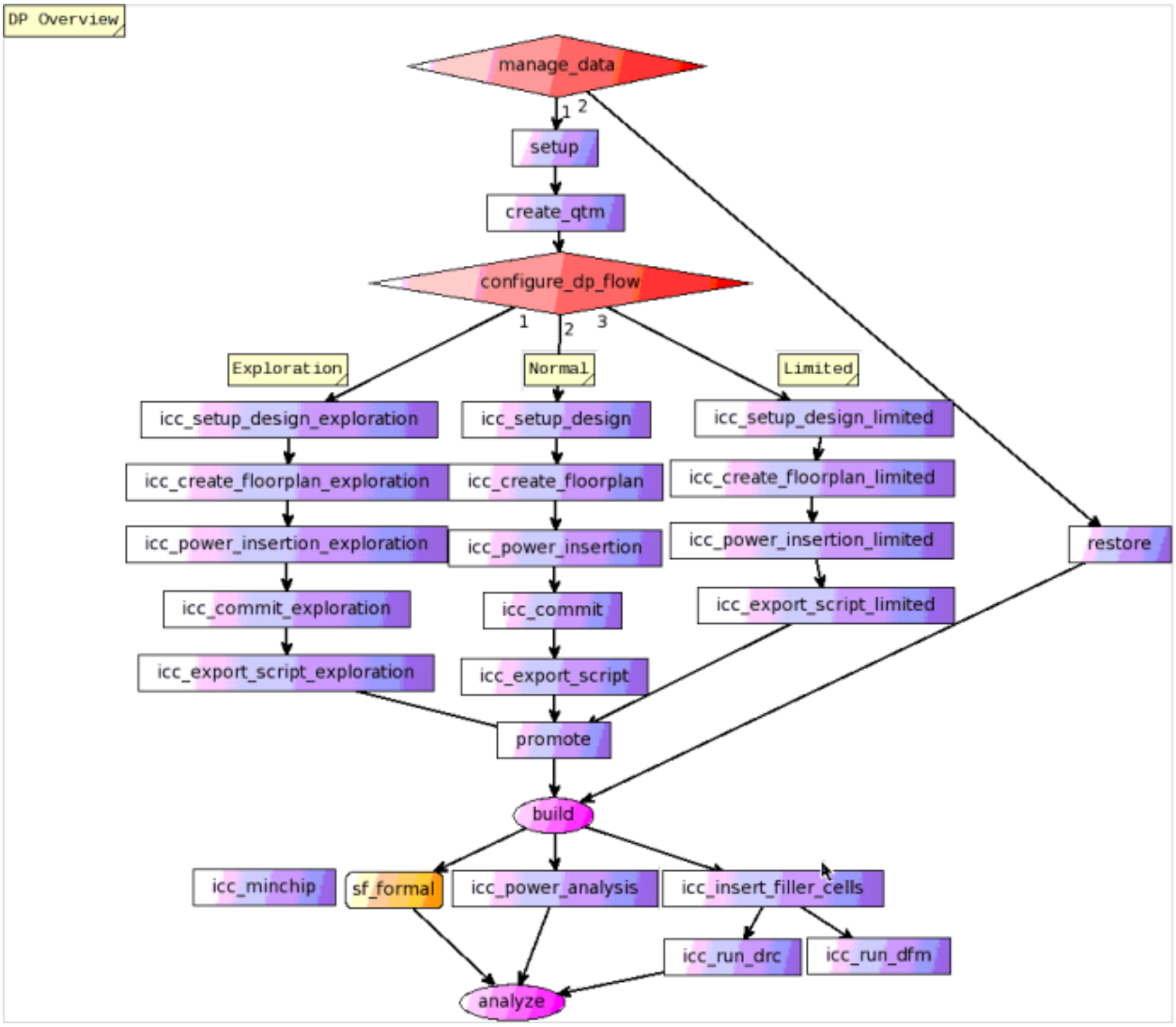


Figure 3: Design Planning sub-flow example

Figure 3: shows a design planning sub-flow that can be viewed or modified graphically through Runtime Manager. Each blue box is an ICC task (which will create floorplan, implement power mesh, etc).

This subflow contains three branches. The first one, which is the exploration branch, wil be used at the beginning of a project, allowing quick loops for floorplan convergence, by skipping parts like tap cells insertion. The second branch, which is the normal one, will use the data from exploration run (macro placement, blockage, etc.) and will execute all required parts for a real design (tap cells and spare cells insertion, etc.). The last limited one will be used when a very small RTL change happens but there is no change at floorplan level.

2.1.4. Management Cockpit

The Management Cockpit is the GUI that is used to create and view up-to-date graphical reports about the design status and trends. When a flow runs in Lynx, important metrics such as those for WNS, runtime and leakage are transmitted to a central database (that can be shared across multiple projects). You can then use the Management cockpit to create reports based on these metrics. There are various types of reports available such as License usage, Dashboard, Flow Summary that are used to report on different aspects of your project.

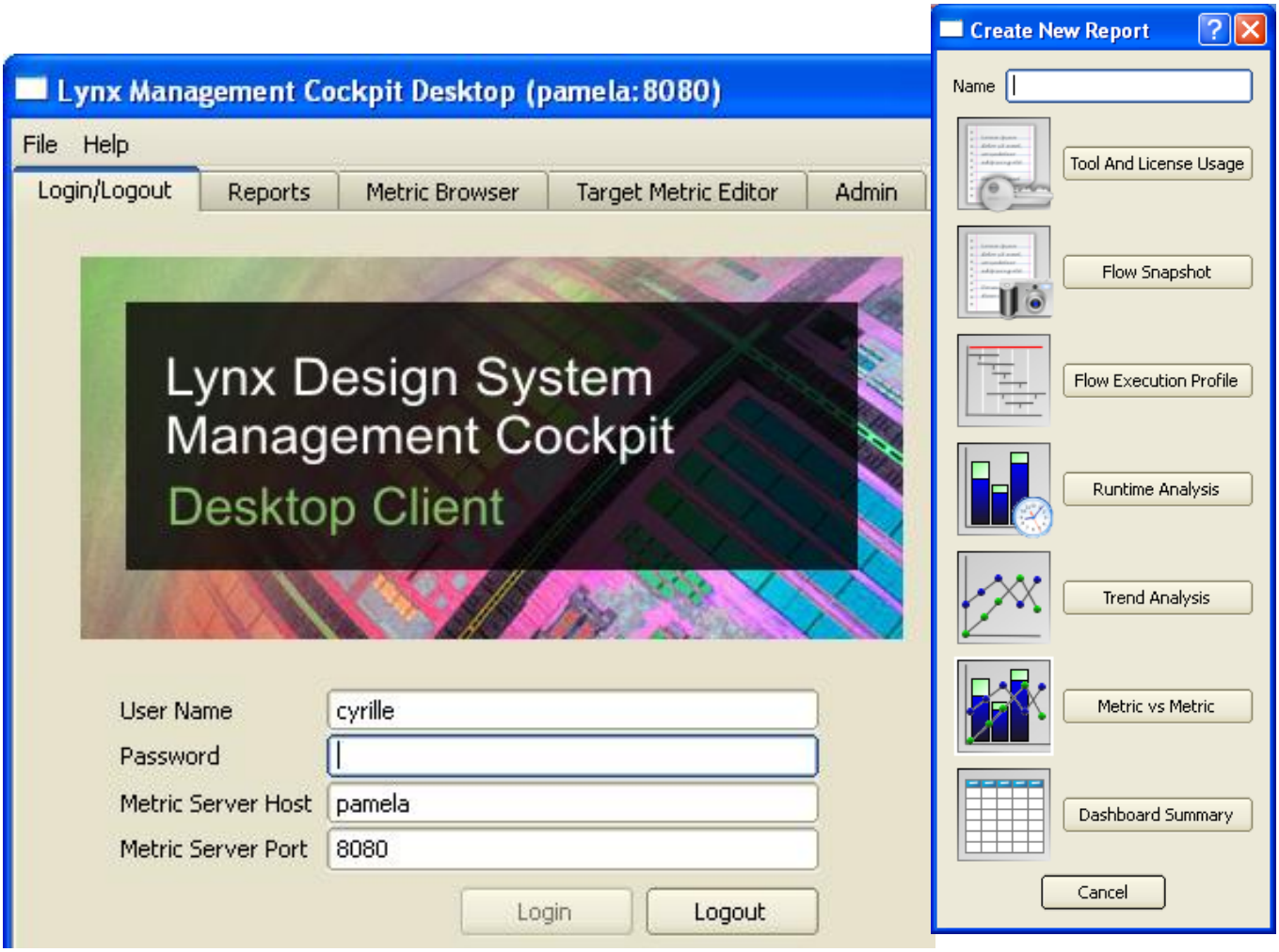


Figure 4: Management Cockpit and reports

Let us take a dashboard report as an example, it is very easy to first filter the metrics for a given project (i.e., for a given chip), then for only some blocks of this project and, then, for a given period. Each row of the table could define a block and each column the Worst Negative Slack (WNS) for a given task. The first column would give us the WNS for *place_opt* task, the second one would give the WNS for *clock_opt_cts*, etc. As described before, the metrics are read from a PostgreSQL database.

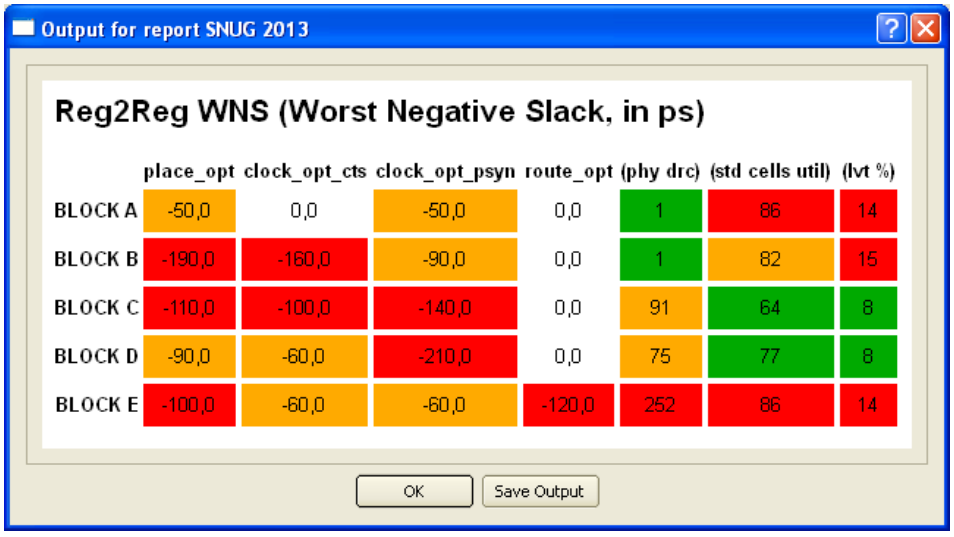


Figure 5: Dashboard report example

A trend analysis report could give the Worst Negative Slack of the main clock for a given block, and a given task, during the last 6 months.

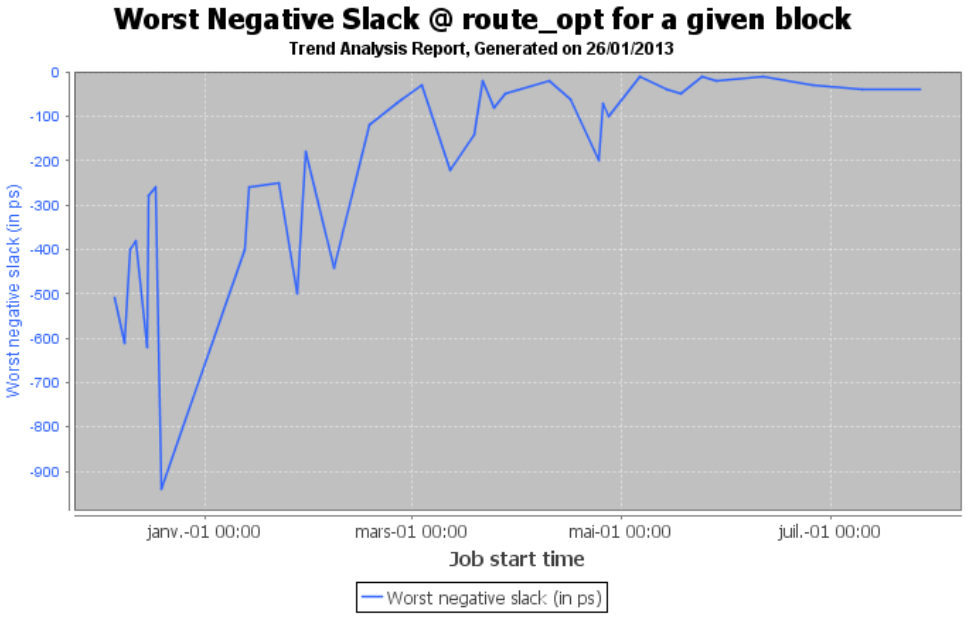


Figure 6: Trend analysis report example

2.1.5. Parallel jobs with Early Completion

RunTime Manager allows to start a new task as soon as the implementation part of the previous task (in green) has ended and a design database written to disk, without waiting for the last section of the script, which is the report generation part (in blue), to complete. This overlapping of tasks contributes to reduce turnaround time.

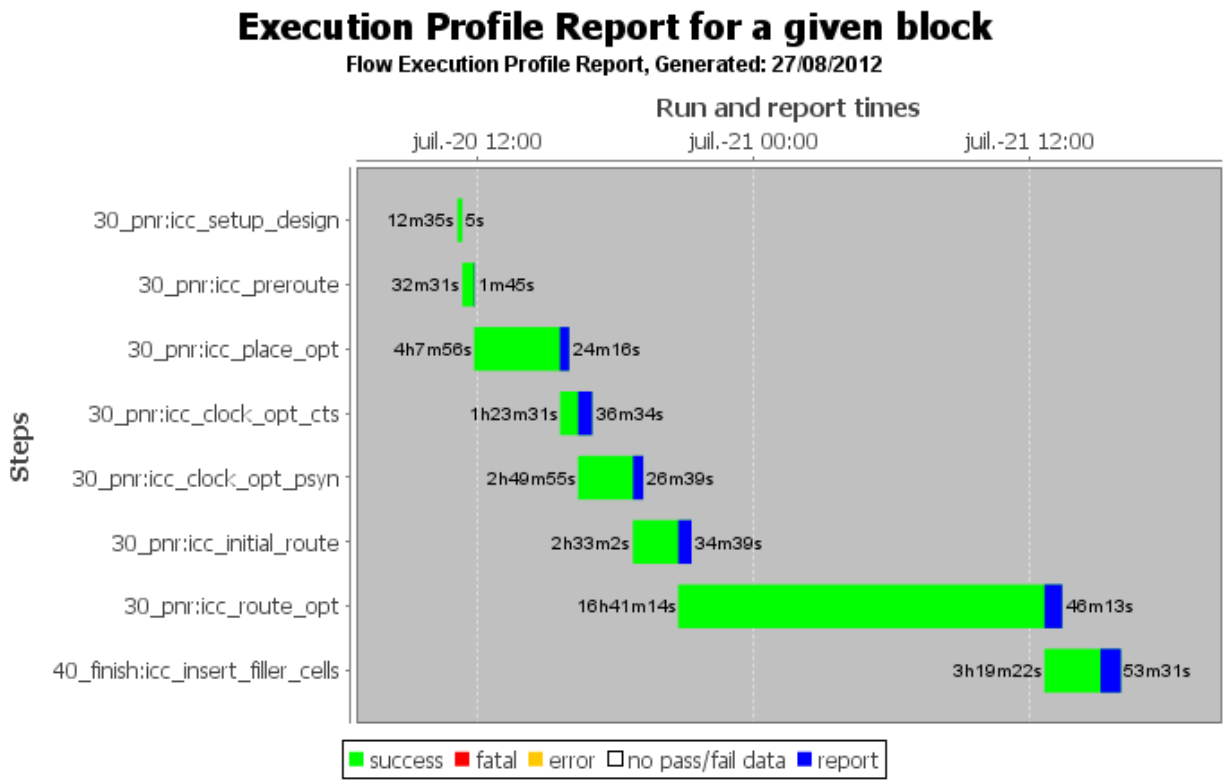


Figure 7: Execution profile report example

Note that the report above is also coming from Management Cockpit and can be created in a few seconds.

2.2. Metrics and records in SQL database

RunTime manager will automatically generate a set of reports at the end of each task: timing reports for setup and holds, summary for physical informations (block size, number of cells and corresponding cell area, etc).

Here an example of the reports for the *place_opt* task:

icc.check_legality
icc.check_physical_design
icc.check_physical_design.1
icc.check_scenarios
icc.list_mw_cels.final
icc.list_mw_cels.initial
icc.list_mw_cels.intermediate
icc.mcmm.report_ilm
icc.mcmm.report_scenario_options
icc.mcmm.report_scenarios
icc.place.check_scan_chain
icc.place.report_bounds
icc.place.report_buffer_tree_qor
icc.place.report_clock_gating
icc.place.report_congestion
icc.place.report_ideal_network
icc.place.report_isolate_ports
icc.place.report_net_fanout
icc.place.report_placement_utilization
icc.place.report_scan_chain
icc.report_constraint
icc.report_design_physical
icc.report_net_fanout
icc.report_power
icc.report_qor
icc.report_qor_snapshot
icc.report_qor_snapshot.old
icc.report_routing_rules
icc.report_threshold_voltage_group
icc.report_timing.max
icc.report_timing.min
icc.report_timing.zic
icc.report_units
icc.setting.report_antenna_rules
icc.setting.report_app_var
icc.setting.report_delay_estimation_options
icc.setting.report_extraction_options
icc.setting.report_ignored_layers
icc.setting.report_optimize_dft_options
icc.setting.report_pnet_options
icc.setting.report_preferred_routing_direction
icc.setting.report_route_opt_strategy
icc.setting.report_route_opt_zrt_crosstalk_options
icc.setting.report_route_zrt_common_options
icc.setting.report_route_zrt_detail_options
icc.setting.report_route_zrt_global_options
icc.setting.report_route_zrt_track_options
icc.setting.report_si_options
icc.setting.report_timing_derate

From these reports, the Lynx Design System generates metrics for each task executed by Runtime Manager. These metrics are written into individual task log files as messages. The format of a metrics message in a log file is:

SNPS_INFO : METRIC | <metric-type> <metric-name> | <metric-value>

where, <metric-name> is the name of the metric, <metric-value> is the individual value of the metric and <metric-type> is the data type of the value.

The most important metrics for each run (Worst Negative Slack, Total Negative Slacks, Number of Violating Paths for each clock group, etc) are printed out in the log file. These metrics can be found at the end of the main log file:

```
SNPS_INFO   : ICC Report Generation Ending    : Wed Dec 12 16:42:24 2012
## ------------------------------------------------------------------------------
## End Of File
## ------------------------------------------------------------------------------
SNPS_INFO   : SCRIPT_STOP  : /physical_design/nderouich/bxi/rtl2gds/scripts_global/pnr/icc_reports.tcl
SNPS_INFO   : PROC_START : sproc_metric_time_elapsed
SNPS_INFO   : PROC_STOP  : sproc_metric_time_elapsed
SNPS_INFO   : METRIC | TIME INFO.ELAPSED_TIME.REPORT | 461
SNPS_INFO   : INFO.ELAPSED_TIME.REPORT | 00:00:07:41
## metric generation
sproc_metric_design
```

```
SNPS_INFO   : PROC_START : sproc_metric_design
SNPS_INFO   : Missing metric: wire_length
SNPS_INFO   : METRIC | DOUBLE  PHYSICAL.AREA | 2609366.760000
SNPS_INFO   : METRIC | DOUBLE  PHYSICAL.WIDTH | 2534.350000
SNPS_INFO   : METRIC | DOUBLE  PHYSICAL.HEIGHT | 1029.600000
SNPS_INFO   : METRIC | PERCENT PHYSICAL.UTIL | 55.9657
SNPS_INFO   : METRIC | DOUBLE  PHYSICAL.WLENGTH | NaM
SNPS_INFO   : METRIC | PERCENT PHYSICAL.CONGESTION | NaM
SNPS_INFO   : METRIC | INTEGER VERIFY.DRC.NUM_ERRORS | 0
SNPS_INFO   : METRIC | INTEGER VERIFY.DRC.NUM_TYPES | 0
SNPS_INFO   : METRIC | STRING  VERIFY.DRC.TOOL | ICC
SNPS_INFO   : METRIC | DOUBLE  LOGICAL.CELL_AREA | 1460349.072935
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_MACROS | 32
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_MODULES | 159151
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_INSTS | 159183
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_FLIPFLOPS | 19345
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_PORTS | 807
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_NETS | 116928
SNPS_INFO   : METRIC | INTEGER LOGICAL.NUM_PINS | 694120
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_CELLS.LVT | 4.99
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_CELLS.MEM | 0.03
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_CELLS.SVT | 94.98
SNPS_INFO   : METRIC | INTEGER PWR.VTH_NUM_INSTS.LVT | 5382
SNPS_INFO   : METRIC | INTEGER PWR.VTH_NUM_INSTS.MEM | 32
SNPS_INFO   : METRIC | INTEGER PWR.VTH_NUM_INSTS.SVT | 102407
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_AREA.LVT | 0.52
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_AREA.MEM | 90.49
SNPS_INFO   : METRIC | PERCENT PWR.VTH_PERCENT_AREA.SVT | 8.99
SNPS_INFO   : METRIC | INTEGER STA.LOGICAL_DRC.TOTAL | 61
SNPS_INFO   : METRIC | INTEGER STA.LOGICAL_DRC.TRANS | 0
SNPS_INFO   : METRIC | INTEGER STA.LOGICAL_DRC.CAP | 61
SNPS_INFO   : METRICS sproc_metric_design took 0 seconds
SNPS_INFO   : PROC_STOP  : sproc_metric_design
sproc_metric_power
SNPS_INFO   : PROC_START : sproc_metric_power
SNPS_INFO   : PROC_START : sproc_clean_string
```

These metrics are pushed to Management Cockpit Database Server, which collects, processes, and stores the metrics, making them available for viewing, analyzing, and generating reports using the clients. Servers consolidate metrics from multiple Lynx Design System projects. The metrics captured are user-definable and user-extensible. Individual metrics are stored in records. Each row (record) corresponds to an executed task in a Lynx Design System design flow. The Server has two databases, primary and secondary. The primary database stores metrics of live projects. The secondary database is as a backup, which can be updated periodically. The database Server also stores reports, making them available remotely over network. When a report is run, current metric records are pulled directly from the Database Server.

To give an idea, our current primary database contains 43 000 records. Each of these runs can stand for more than 100 metrics (Worst Negative Slack for each group, etc…). The size of the PostgreSQL database is 2.2GB. We regularly back-up it using transfers in a secondary database through Management Cockpit.

2.3. Grid integration

Lynx has a native support for LSF and SGE (Sun Grid Engine) job queueing tools. It also provides an optional but very useful feature called "Adaptive Resource Optimization" (ARO).

ARO is a compute farm optimization feature in Synopsys' Lynx Design System. The ARO algorithm monitors usage patterns of a job and computes an accurate value to be used for resource reservation of future submissions of the same job. Underestimating memory requirements for submitted jobs often result in 'thrashing' where the system constantly writes to disk to support the job. Conversely, over-estimating memory requirements results in under-utilization of CPU resources. Some cores of a multi-core system may not be utilized due to lack of reservable memory. By dynamically adjusting the required memory and queue, ARO eliminates system thrashing and improves TAT. When the same job is submitted, ARO will dynamically modify the resource reservation provided by the user before it is submitted to the compute farm. ARO can also be configured to dynamically submit jobs to specific queues based on the computed memory and/or runtime values. In doing so, ARO can substantially reduce job pending time (i.e., the time a job sits in queue waiting for a resource) thereby improving Turn-Around Time (TAT). ARO also improves compute farm utilization by ensuring that jobs are assigned to the best queue/machine combination based on past behavior.

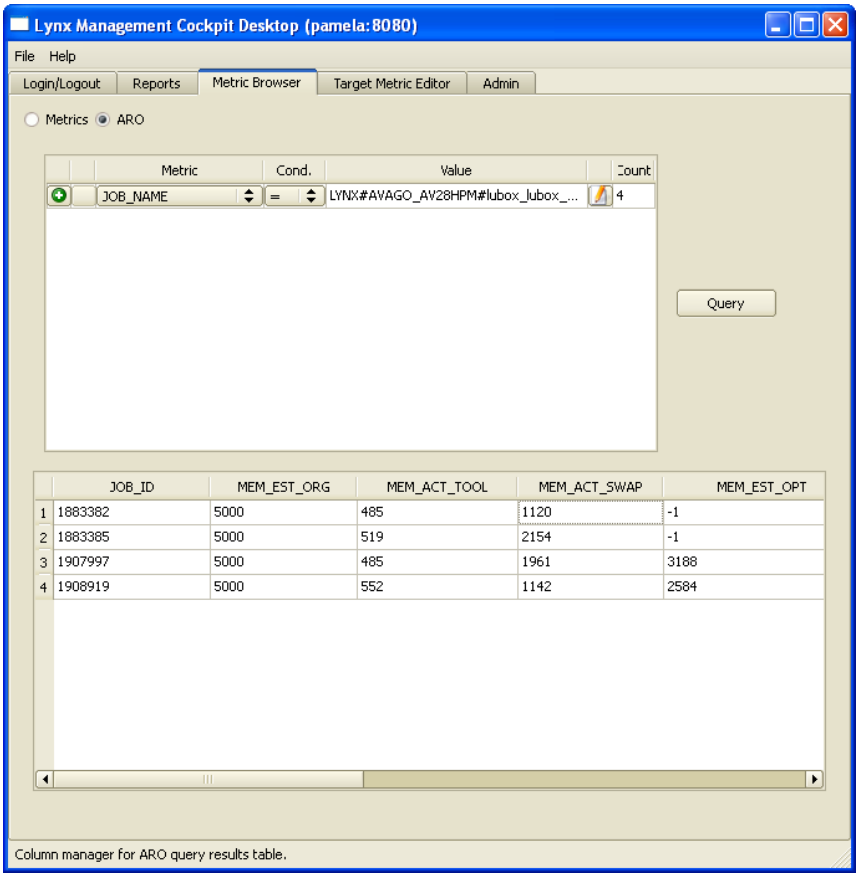


Figure 8: Adaptive Resource Optimization within Management Cockpit

In the Management Cockpit screenshot above, we can see that we ran a same job four consecutive times. The first column of the table indicates the job ID, the second column the amount of memory requested by the user, the third one the amount of memory used by the tool, the fourth column the actual amount of memory used in Linux environment. Memory used by the tool and by Linux should be quite the same but it is not the case. Fortunately ARO allows to choose which memory value to use for optimization. The last column indicates the amount of memory really requested by RunTime Manager. Note that "-1" in this column indicates that RTM used the value requested by the user. In our example, we configured ARO in order to start memory optimization on a given task when two runs have already been completed with success. Then, for the first two runs, RTM used 5000MB, which is the value requested by the user. For the 3rd run, ARO started memory optimization and RTM requested 3188MB. For the 4th run, ARO was able to refine again, by requesting 2584MB only, which is half of the default value, and this amount of memory is not so far away from the highest value, needed during run #2.

This approach applied on dozens of parallel jobs can save huge amount of memory, allowing more jobs to be launched.

2.4. 3rd party tool integration

The ability to add 3rd party tools within Lynx easily was a requirement for us when we decided to use it. A few months after the installation of the Synopsys workflow, we added some Apache DA Redhawk licences for IR drop analysis, in addition to Synopsys PrimeRail. It took less than two days to add a subflow running Apache Redhawk within Lynx for this task. Rather than just invoking Apache, the Redhawk subflow also integrated and used StarRC-XT and PrimeTime in addition to Redhawk itself, making for a complete subflow that can run on Milkyway input

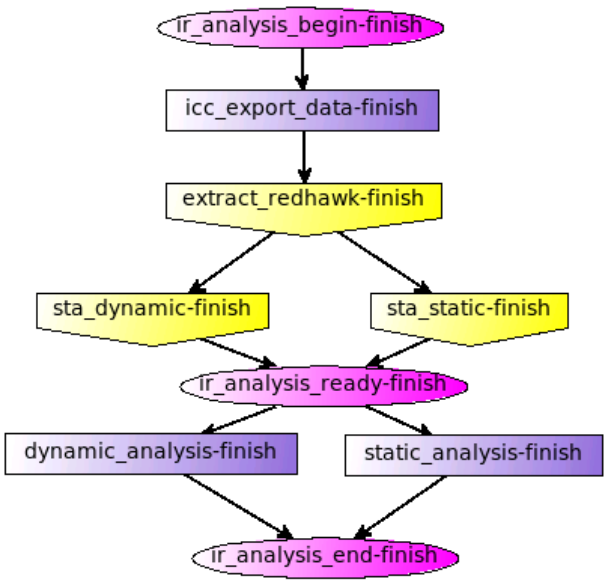


Figure 9: Subflow added for a 3rd party tool

2.5. Modules approach

A clean “module” approach in Linux environment allow to handling easily tools version within Lynx.

We followed recommendation from Reference [1].

```
---------------- /clionfs/da_solaris2/utilities/Modules/modulefiles/synopsys_modules
synopsys_activetcl/8.4.13(default)         synopsys_lynx/2010.12-SP1
synopsys_activetcl/8.5.12                  synopsys_lynx/2010.12-SP3
synopsys_dwip/G-2012.06                    synopsys_lynx/2010.12-SP5
synopsys_dwip/G-2012.06-SP1-1              synopsys_lynx/2011.09-SP1
synopsys_fm/G-2012.06-SP2(default)         synopsys_lynx/2012.06-SP2
synopsys_hercules/B-2008.09-SP3-1          synopsys_mw/D-2010.03-SP5-1
synopsys_hercules/B-2008.09-SP3-2(default) synopsys_mw/E-2010.12
synopsys_hercules/B-2008.09-SP5            synopsys_mw/E-2010.12-SP3
synopsys_hspice/C-2009.09                  synopsys_mw/E-2010.12-SP5
synopsys_hspice/E-2010.12(default)         synopsys_mw/F-2011.09-SP2
synopsys_icc/A-2007.12-SP5-5               synopsys_mw/F-2011.09-SP5-1
synopsys_icc/E-2010.12-SP5-2               synopsys_ns/E-2010.12
synopsys_icc/E-2010.12-SP5-3               synopsys_primerail/E-2010.12(default)
synopsys_icc/F-2011.09-SP1                 synopsys_pts/F-2011.06-SP2
synopsys_icc/F-2011.09-SP2-1               synopsys_pts/F-2011.12-SP2
synopsys_icc/F-2011.09-SP4                 synopsys_pts/F-2011.12-SP3
synopsys_icc/F-2011.09-SP5                 synopsys_pts/G-2012.06-SP3(default)
synopsys_icc/F-2011.09-SP5-1               synopsys_starrc/D-2010.06-SP3-2
synopsys_icc/G-2012.06-SP3(default)        synopsys_starrc/E-2011.06-SP3-1(default)
synopsys_icvalidator/E-2010.12(default)    synopsys_syn/E-2010.12-SP5
synopsys_icvalidator/G-2012.06-SP1-1       synopsys_syn/F-2011-09
synopsys_icwbev_plus/D-2010.06-4(default)  synopsys_syn/F-2011-09-SP2
synopsys_installer/2.0.0                   synopsys_syn/F-2011-09-SP5
synopsys_installer/2.7.0(default)          synopsys_syn/F-2011.09-SP5-1
synopsys_java/1.6.0_23(default)            synopsys_syn/G-2012.06
synopsys_lynx/2010.12                      synopsys_syn/G-2012.06-SP3(default)
```

Then, we just need to define some variables in a Lynx dedicated file (*sytem.tcl*) to choose which version of the tools to use.

For instance, in order to use a particular version for IC Compiler:

```
set SEV(ver_icc) "synopsys_icc/G-2012.06-SP2"
```

Of course, it is also possible to use the default Linux environment when calling the tools by setting to 0 a Lynx SEV (Synopsys Environment Variable), in the same *system.tcl* file.

```
set SEV(ver_enable) "0"
```

3. Simultaneous Design Execution at 28nm using Lynx

The objective is to provide one global flow which can be used for several projects in parallel. Of course, each project has its own specifications (frequency is a good example) and the challenge is to offer some flexibility which can be tracked and supported easily.

3.1. Reference flow

As explained in chapter 2.1.3, RunTime Manager needs a particular structure of folders and files to work fine. "*scripts_global*" directory contain all the reference scripts, grouped by steps in some folders (synthesis, design planning, place and route, etc). All these directories, subdirectories and files are revision-controlled, contained in a *rtl2gds* directory which stands for our main flow.

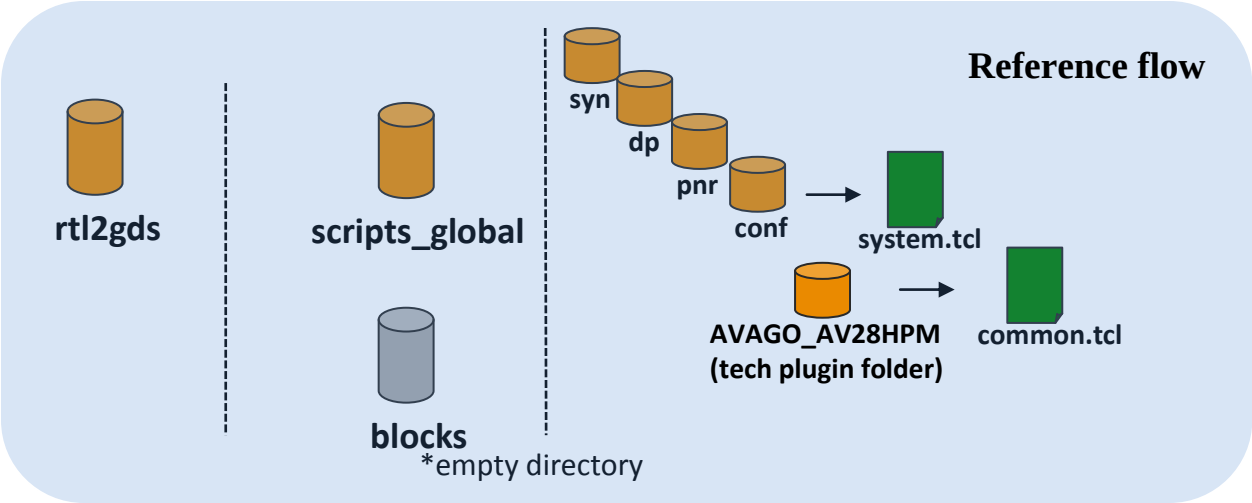


Figure 10: Reference flow

But "scripts_global" directory also contains some configuration files:

- *system.tcl* which defines the project name (used for PostgreSQL records), tools version and job queues to use, etc. This file is in a *conf* sub-directory.
- *common.tcl* which lists all the IP which can be used and defines the corresponding information (linux paths to the physical views, to the timing description files, etc). This file also sets all the default values for Task Environment Variables introduced in section 2.1.1. This *common.tcl* is in a tech plugin folder (*AVAGO_AV28HPM* in our exemple).

As these files define project-dependent or technology-dependent information, they should be different for each project. With the revision control tool, we could use a particular version of these files for a given project (using a branch approach for instance, one branch for each project). But a drawback of this method is that all the projects (and the corresponding blocks) would populate the *blocks* directory which contains all the data (Milkyway, reports but also block-specific scripts) for all the blocks defined through RunTime Manager.

This situation is not acceptable for several reasons:

- Having all the blocks of all the projects in a single *blocks* directory can lead to hundreds of blocks listed when opening the RunTime Manager instead of only having the blocks of a specific project.
- Some projects can be confidential and then imply to have dedicated flow for them.

In these conditions, we decided to use the *rtl2gds* flow previously described as a reference flow only, with no block implemented in this area. As a consequence, the *blocks* directory remains empty.

3.2. Project-specific flow

Once a reference flow is created, each project uses it as a baseline. A mechanism must exist to offer some flexibility in order to allow specificities for a particular chip.

Imagine two chips CHIP_A and CHIP_B, represented by two folders Project_A and Project_B in the revision control tool. Each of these two directories will contain RTL, PSL properties, constraints and some other items for blocks of the corresponding chip. In addition to these files, a *rtl2gds* subdirectory will be used as Lynx root directory for the given project. Then, RunTime Manager will be launched in *Project_A/rtl2gds* or in *Project_B/rtl2gds*.

Now we can discuss how to populate these two *rtl2gds* subdirectories.

Regarding *scripts_global* (containing reference scripts), it will be a symbolic to the reference flow. It means that a physical designer for a given project won't be able to modify the main scripts. It is a way to be sure that everybody is using the same flow.

As described before, we still have some project-specific files (*common.tcl*, *system.tcl*, etc) which have to be handled in *Project_A/rtl2gds* or in *Project_B/rtl2gds*. They will be "real" files (and revision controlled) instead of being symbolic links to reference flow.

For instance:

- *Project_A/rtl2gds/system.tcl* will define the project name as Project_A. For all the runs, this important information will be stored in the PostgreSQL database and then will be used as a filter when extracting metrics within Management cockpit.
- *Project_A/rtl2gds/common.tcl* will list only the IP used by CHIP_A (which can be a subset or a different set of IP used by CHIP_B).
- And all other scripts which have to be specific to a project or a node

We can estimate that only a dozen of scripts will be specific to a project. All the other files (around 300) will remain symbolic links to the reference flow.

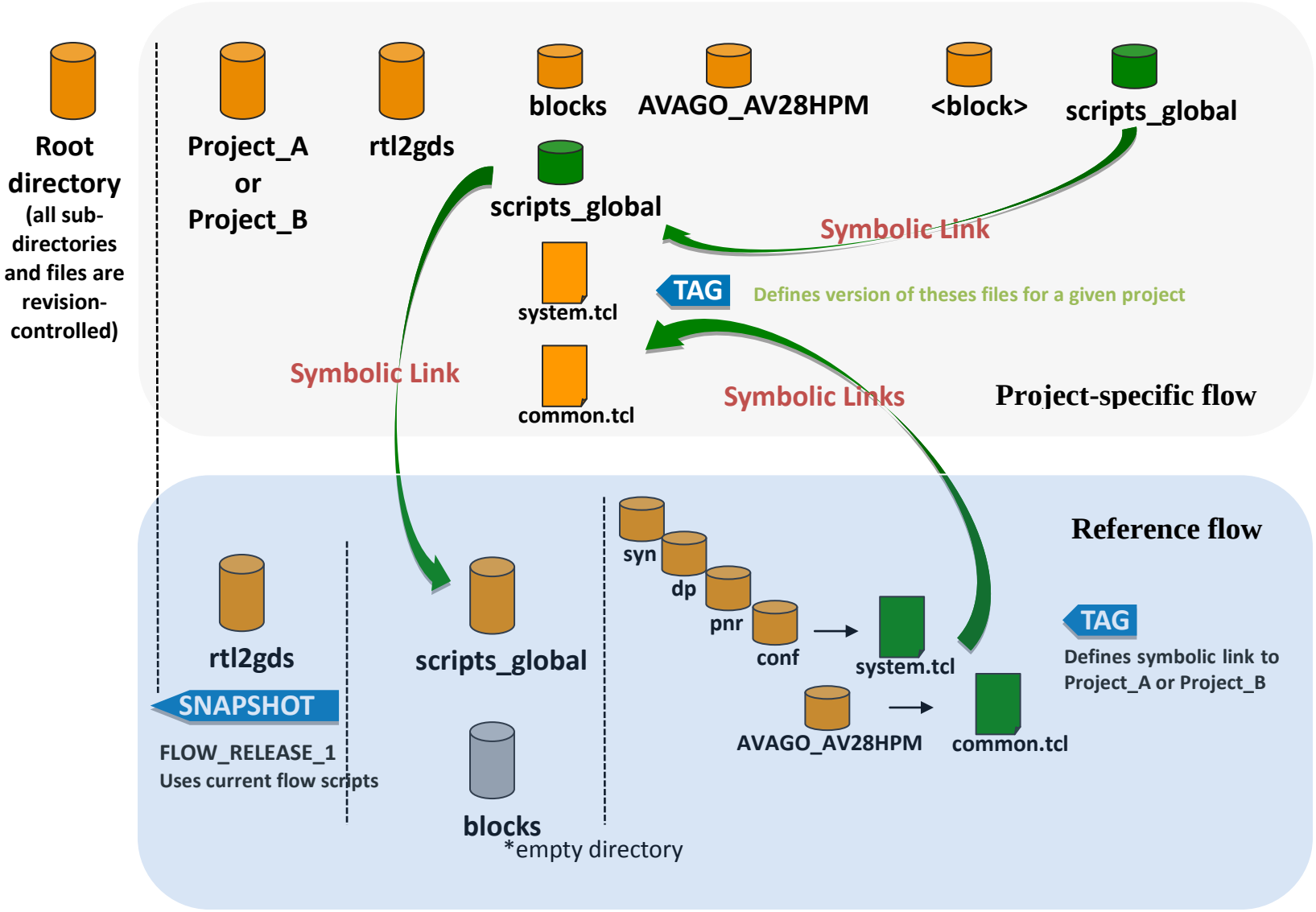


Figure 11: Reference flow and project-specific flow

Figure 11: describes the way to handle project-specific files, through the revision-control tool.

Take *system.tcl* as an example. To make this file a project-specific file for Project_B, we need to:

- Check-out *system.tcl* in the reference flow (the file is in *rtl2gds/scripts_global/conf*)
- Create a new *system.tcl* file in Project_B (for instance in *Project_B/rtl2gds*). An already existing *system.tcl* file (coming from the Lynx production flow or from another Project_A) can be used a baseline
- Modify *system.tcl* file in the reference flow to become a symbolic link to this new Project_B file
- Check-in the new version of the file in the reference flow. In fact, each revision of *system.tcl* in the reference flow is a symbolic to a project-specific *system.tcl*. In other words, the previous version of this file in the reference flow is certainly a symbolic link to *Project_A/rtl2gds/system.tcl*.
- Add a tag (symbolic name) on this new revision of the file in the reference flow
- Optionaly create a snapshot of the reference flow. A snapshot consists of applying a same tag on a collection of files and directories with the additional restriction that these tags cannot be moved. Usually, a snapshot is created by the persons responsible for the reference flow when they reach a significant milestone.

Then, a user of Project_B will have to:

- Choose the correct snapshot and tag(s) for the *rtl2gds* reference flow, in order to:
    - Use the reference flow in a particular state or milestone (*FLOW_RELEASE_1* for instance)
    - Use the correct *system.tcl* (which should be a link to *Project_B/rtl2gds/system.tcl*)
- Choose a correct tag (or snapshot) for the project-specific flow in order to:
    - Use the correct system.tcl within Project_B area. We can imagine that this file has successive revision (due to tool version update, etc)
    - Use all the other correct project-specific files

To summarize, if for some reason, a file which was a symbolic link to the reference flow for CHIP_B needs to become project-specific, we just need to "break" the link, modify the script and then check-in the new version of the file in Project_B environment. Of course, if the corresponding file in the main flow is updated later on, designers of CHIP_B will have to compare their local version with the reference one in order to decide if they need to take into account the modification on their side.

The "blocks" directory will be project-specific for the reason that it will contain the data related to the blocks of a particular chip. As mentioned previously, the *block*s directory in the reference flow is empty (or could contain some blocks used as test cases but not those related to a real project).

3.3. Block Script Flexibility

As explained in chapter 2.1.1, the original Lynx production flow provides pre-scripts and post-scripts mechanism allowing additional commands before or after main commands (additional *psyn_opt* commands in a post-script after *place_opt* command in the main script for instance). But sometimes, we need to replace the main commands themselves for a given block. For instance, we could want to replace a place_opt command by a "*create_fp_placement + place_opt –skip_initial_placement*". Lynx provides a way to source a "block" script instead of "global" script and we could have a dedicated *icc_place_opt.tcl* script for this particular block. But each time a new version of the global script is released, the block designer will have to compare it with his own version.

Given these requirements and conditions, we were looking for ways to add flexibility for the redefinition of the main commands. We decided to split the original Lynx scripts into several sections, redefine them as procedures that can be overriden. We mandated that all the overrides be in a single file for easier tracking.

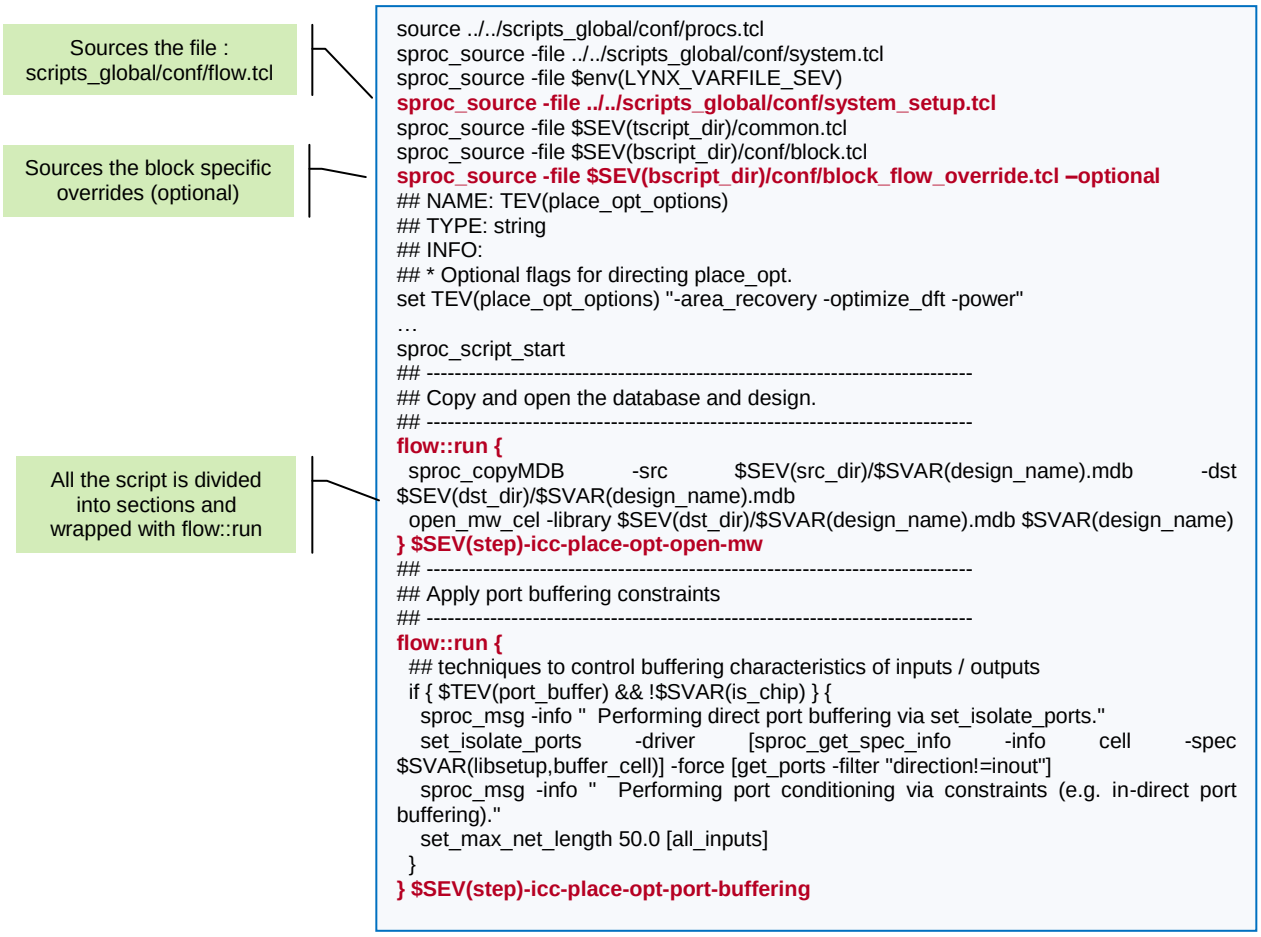


Figure 12:Global script example (icc_place_opt.tcl script in pnr flow)

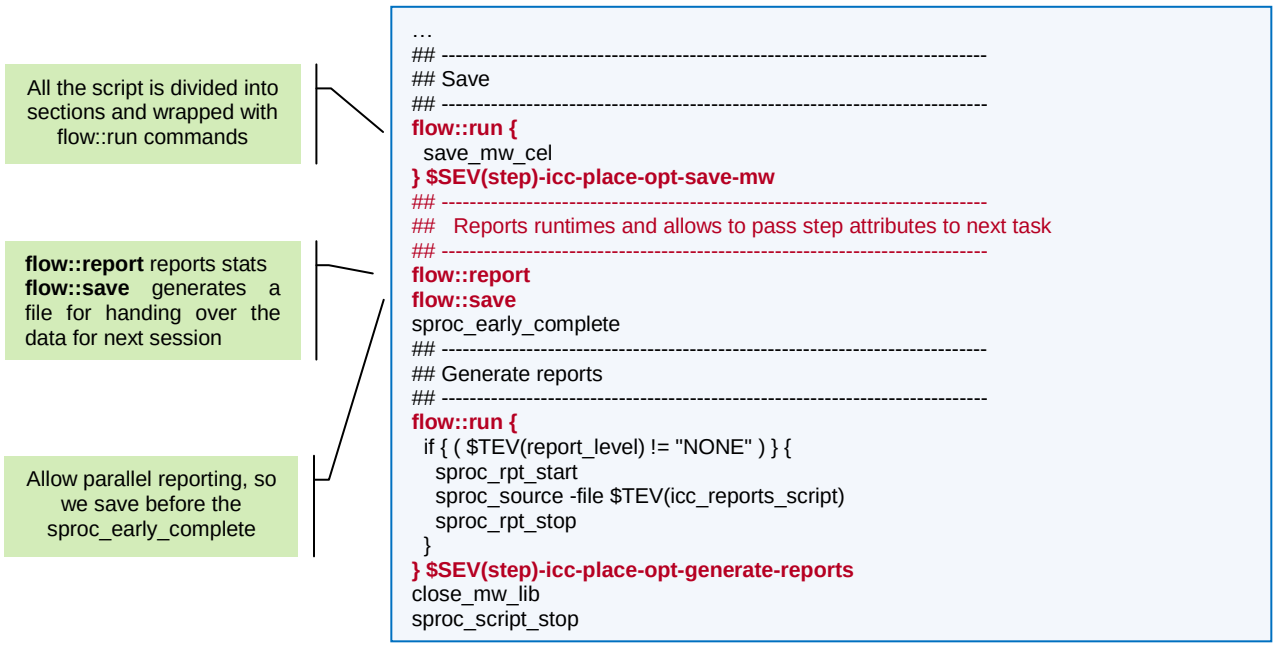


Figure 13: Global script example (icc_place_opt.tcl script in pnr flow)

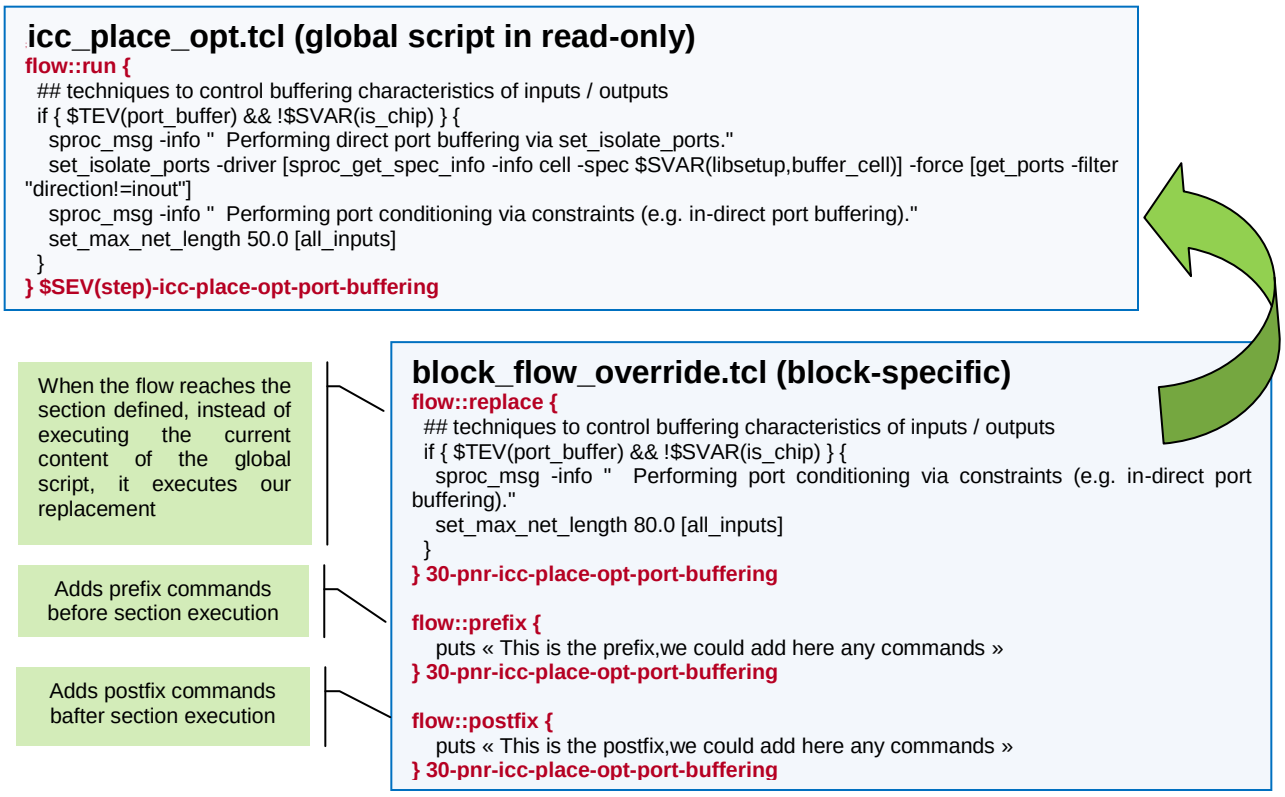


Figure 14: Override example

As explained in chapter 3.1, global scripts are part of a reference flow and can not be modified by users as they could impact all the blocks/top-level work. Additional block-level override capability has been added through several functions defined in *scripts_global/conf/flow.tcl*:

- **flow::run { <*script section*> } step_name**

  Allows the override of the script section

- **flow::replace { <*replacement section*> } step_name**

  Allows the replacement of the content of the step_name by block-specific commands

- **flow::prefix { <*prefix section*> } step_name**

  Allows adding block-specific commands before the default "script section"

- **flow::postfix { <*postfix section*> } step_name**

  Allows adding block-specific commands after the default "script section"

- **flow::report <*pattern*>**

  Allows reporting the runtimes, memory usage… of each step run matching the pattern (by default it reports all steps)

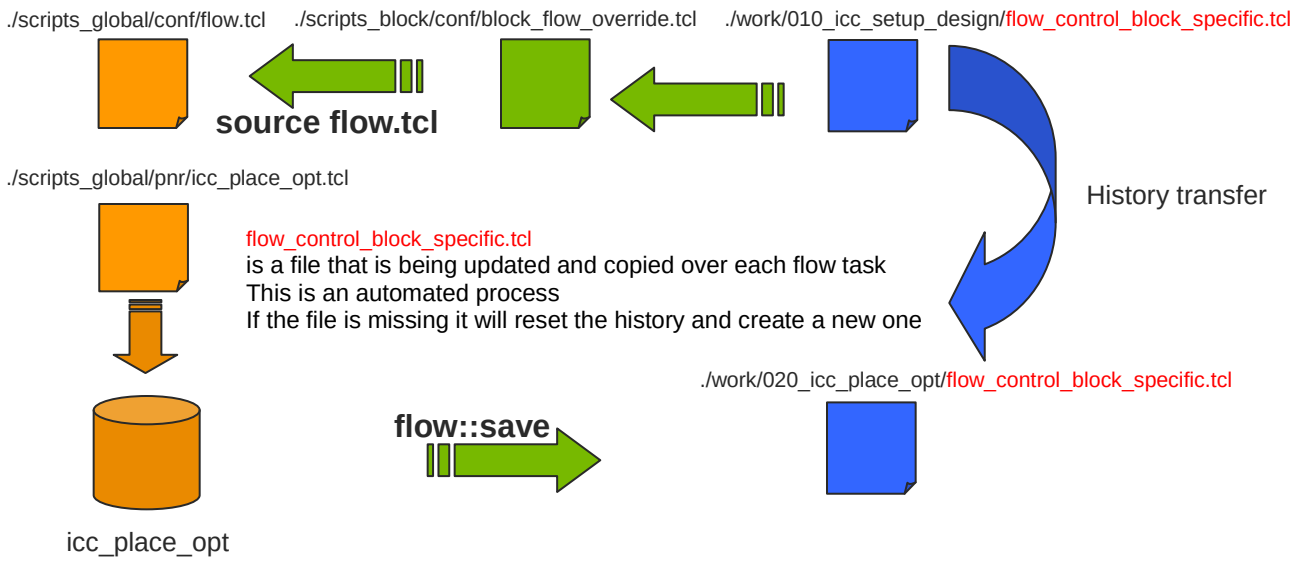
- **flow::report_step <*step name*>**

  Reports which commands have been executed for the step (including any postfix/prefix/replacement if any)

- **flow::save**

  Saves a file with all the current steps run so that it can be handover to the next session

Find below additional information about this flow::save function which is a mechanism for handing over the steps history across sessions.



3.4. Flow update process

As previously explained, we use the original Lynx structure (files and directories) as a starting point and the Synopsys golden methodology scripts as a baseline. But after customization and split process of the scripts into overridable sections, a major part of the *scripts_global* files are modified.

Thus, when Synopsys deliver a new release of Lynx (which consists in new reference scripts for instance), we compare these scripts with previous Synopsys delivery and then decide to take them or not. If yes, we just need to add our customization and sections again in the new files.

For a revision control tool perspective:

- If the file was not there in the previous release, we use the Synopsys original file to create a revision "1" in the reference flow. Then a revision "2" is created by adding sections split and customization, if needed.
- If the file was already there before, and if we decide to use the new version from Synopsys, a new revision is created using the new Synopsys script inluding sections split and customization, if needed
- The revision-controlled file is removed from the database if the script does not exist in the new Synopsys release

If the new release consists in a Management Cockpit update or the introduction of a new feature like Adaptive Resource Optimization, the update process is easier because it does not really impact reference scripts and most of the "system" files in our reference flow are Synopsys ones.

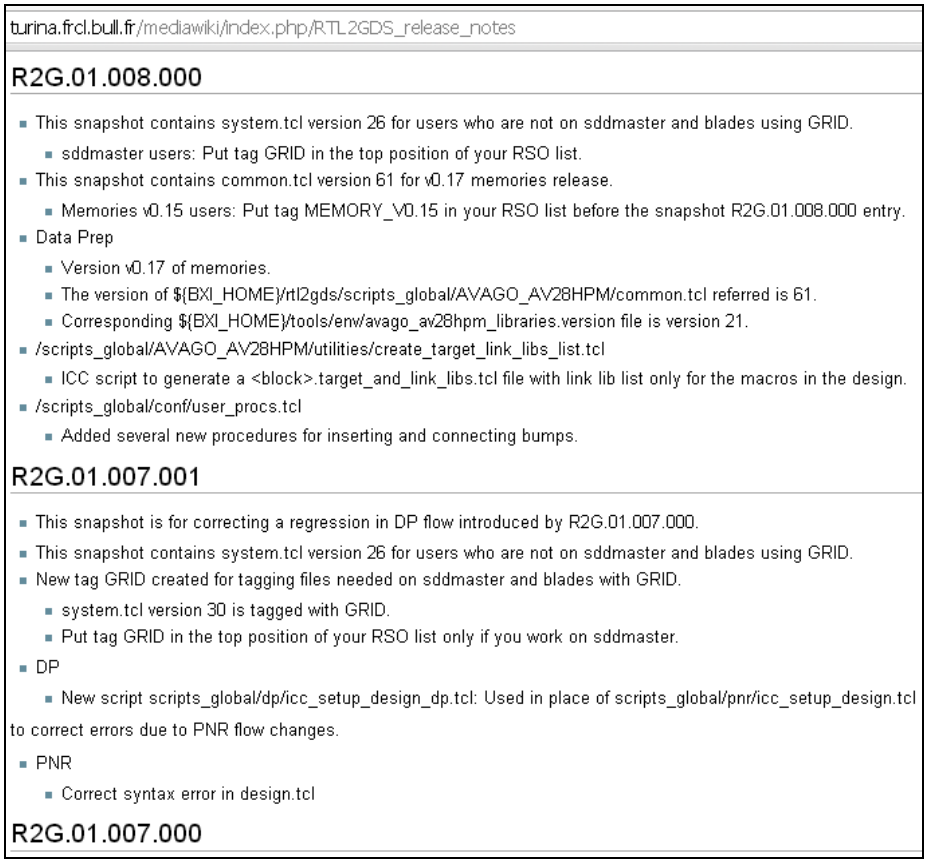
Note that Synopsys also provides a script that can compare 3 Lynx script versions (original, customized and new release) and show the changes required.

3.5. Tags and snapshots

When a script in the reference flow needs to be changed, the engineer responsible for the flow checks out the file, modifies it and then checks-in it. If this update is a major one, it is documented in a "flow" wiki page. Then, a tag is put on the file. This tag means that the new version will have to be taken into account for the next snapshot (which is a capture of all the files sharing the same tag at a given time).

A snapshot is created when:

- A very important bug has been found and the current snapshot is not usable,
- Quite a lot of scripts have been updated, for minor changes, but it is a good time to have a new reference
- Some new features have been added and tested on some blocks


turina.frcl.bull.fr/mediawiki/index.php/RTL2GDS_release_notes

R2G.01.008.000

- This snapshot contains system.tcl version 26 for users who are not on sddmaster and blades using GRID.
  - sddmaster users: Put tag GRID in the top position of your RSO list.
- This snapshot contains common.tcl version 61 for v0.17 memories release.
  - Memories v0.15 users: Put tag MEMORY_V0.15 in your RSO list before the snapshot R2G.01.008.000 entry.
- Data Prep
  - Version v0.17 of memories.
  - The version of ${BXI_HOME}/rtl2gds/scripts_global/AVAGO_AV28HPM/common.tcl referred is 61.
  - Corresponding ${BXI_HOME}/tools/env/avago_av28hpm_libraries.version file is version 21.
- /scripts_global/AVAGO_AV28HPM/utilities/create_target_link_libs_list.tcl
  - ICC script to generate a <block>.target_and_link_libs.tcl file with link lib list only for the macros in the design.
- /scripts_global/conf/user_procs.tcl
  - Added several new procedures for inserting and connecting bumps.

R2G.01.007.001

- This snapshot is for correcting a regression in DP flow introduced by R2G.01.007.000.
- This snapshot contains system.tcl version 26 for users who are not on sddmaster and blades using GRID.
- New tag GRID created for tagging files needed on sddmaster and blades with GRID.
  - system.tcl version 30 is tagged with GRID.
  - Put tag GRID in the top position of your RSO list only if you work on sddmaster.
- DP
  - New script scripts_global/dp/icc_setup_design_dp.tcl: Used in place of scripts_global/pnr/icc_setup_design.tcl to correct errors due to PNR flow changes.
- PNR
  - Correct syntax error in design.tcl

R2G.01.007.000

Figure 15: Snapshots Wiki page

When a snapshot is created, an email is sent to all Lynx users: logical design teams for front-end part (synthesis and DFT) and physical design teams. All information regarding this new snapshot can be found on a Wiki page but it is also added in the email.

The recipients will have to update their “Revision Search Order” within the revision control tool in order to use the files defining this snapshot.

If for some reason, a physical designer wants or needs to experiment a new feature, not already included in a snapshot, he can have access to a “USER_TAG” mechanism. A specific tag can be applied by the engineer responsible for the flow on some files. These files will be accessable by the ones who add the tag in their revision search order, like in Figure 16:.

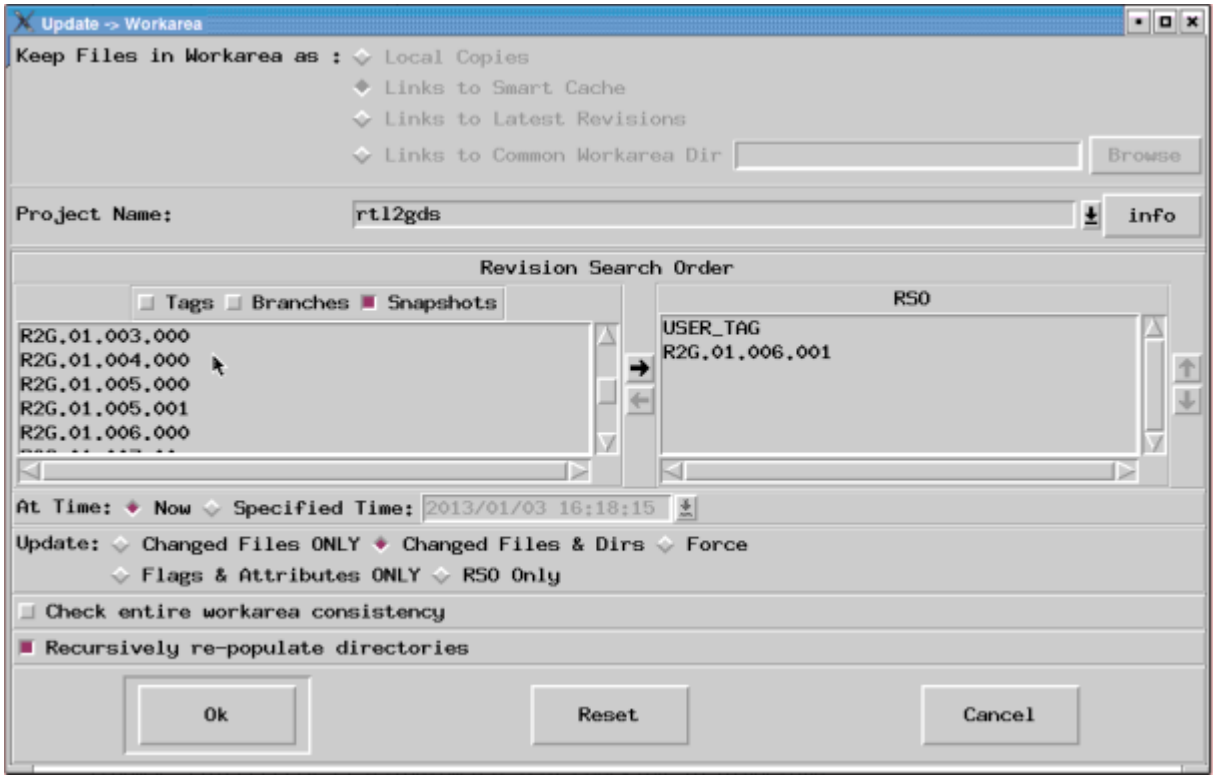


Figure 16: Revision Search Order within the revision control tool

4. Management Cockpit for Decision-making

Management Cockpit has already been introduced in chapter 2.1.4. We discuss below how it can be used for decision making.

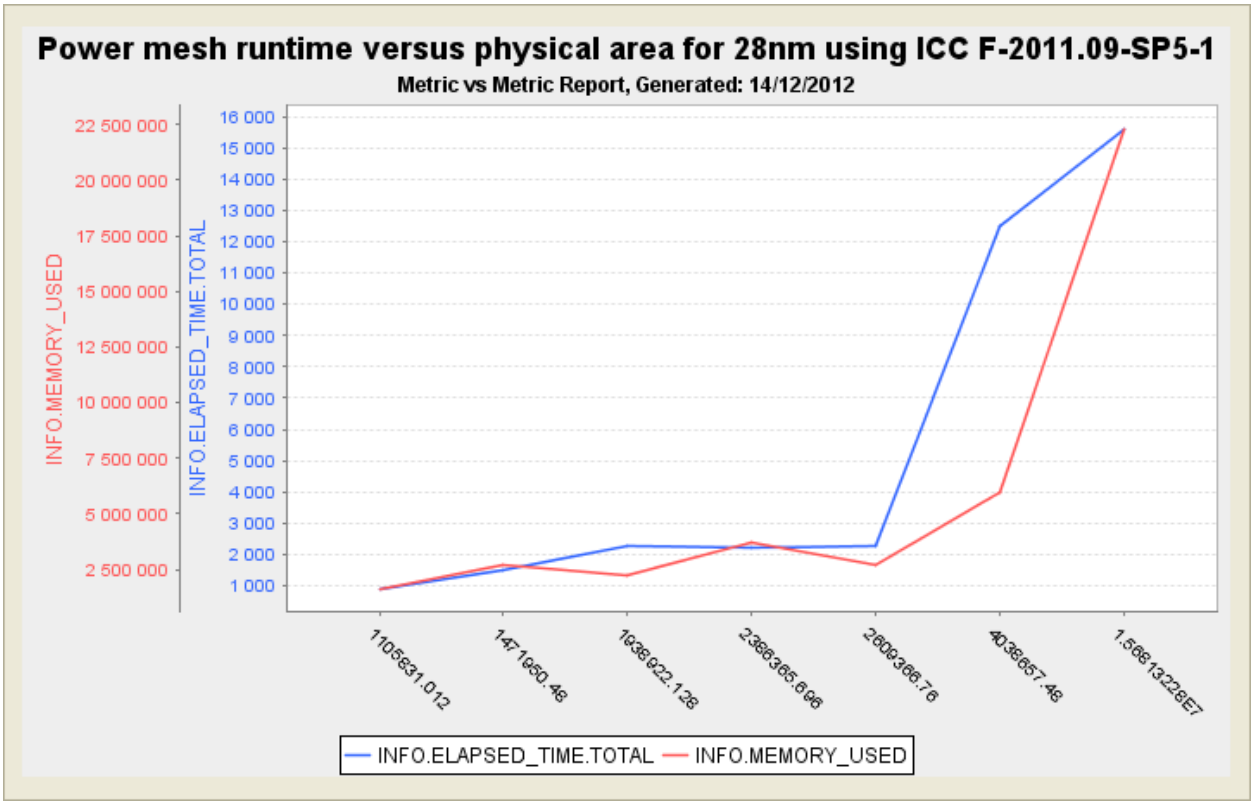


Figure 17: Power mesh Runtime versus Physical Area for 28nm

For our 28nm projects, we use a very conservative and structured power mesh. The corresponding number of vias is so huge that it can have a big impact on runtime and memory usage during the preroute task. And it became a criteria for the maximum physical size of our blocks.

We were able to use the Management Cockpit to decide the partitioning of our design. In Figure 17:, we can see that for blocks from 1mm^2 to 2.6mm^2, the runtime (less than half an hour) but also the memory usage (less than 5GB) are under control. Unfortunately, memory footprint and execution time become unacceptable for blocks with bigger area. For a 15 mm^2 block, we reach 22GB and more than 4 hours. For bigger block size, run time and memory usage become prohibitive.

This kind of reports allowed us to decide the limit for physical dimensions of our blocks, due to a conservative power mesh.

5. Future Work

We introduced Lynx into our workflow quite smoothly. We started with using the Place & Route, chip finishing, STA and physical verification flows on a 40nm project. Then, we extended Lynx usage to synthesis, DFT and Design Planning for the following projects.

Next steps will be to add our IP data prep flow within Lynx, and to have some data and integration checks within this Synopsys workflow.

5.1. Data preparation

Our IP provider exchange data using LEF and GDS2. On our side, we have to create FRAM and CEL views in order to use these cells within ICC tools. We currently run the Milkyway scripts outside of Lynx and the plan is to add them in a dedicated flow that can be run in Lynx

5.2. Data checks

We use several dozens of IP and we regularly encounter consistency issues. With preliminary deliveries, for instance, we had ports direction mismatches between logical/physical views and compiled version for memories (inputs becoming outputs and vice-versa). We also had compilation failures due to missing submodules for some MBIST IP. We could have discovered all these issues before delivering IP to logical/verification team if we had used Lynx Data Checks integrated flow.

We generate our own CEL views using the GDS2 coming from the IP provider and we generate the corresponding FRAM using the LEF. Consistency checks included in Lynx data checks could also allow us to verify that FRAM and CEL are consistent with each other and to be more confident when delivering data to physical designers.

5.3. Integration checks

Integration checks flow within Lynx allows library data test in a real design environment, instantiating all the cells in a block, running synthesis, design planning, place and route, chip finishing and then all physical verifications (DRC, LVS) . Checks like these can detect routeability issues earlier in the project, allow fine-tuning for tech file, implementation tools (playing with variables) and verification tools (setting the physical verification rundecks).

6. Areas for Improvement

Lynx update is the main area for improvement.

As explained in chapter 3.4, "*scripts_global*" files need particular attention when merging customized scripts with new Synopsys version.

But updates can also involve Linux manipulations with root permissions and IT expertise. For ARO feature added in 2011.09-SP2 release (see section 2.3 for more information about this feature), we needed to update PostgresSQL database with new tables, to install newer version of some binaries (mcgetter), to make SGE available on the metrics server, etc.

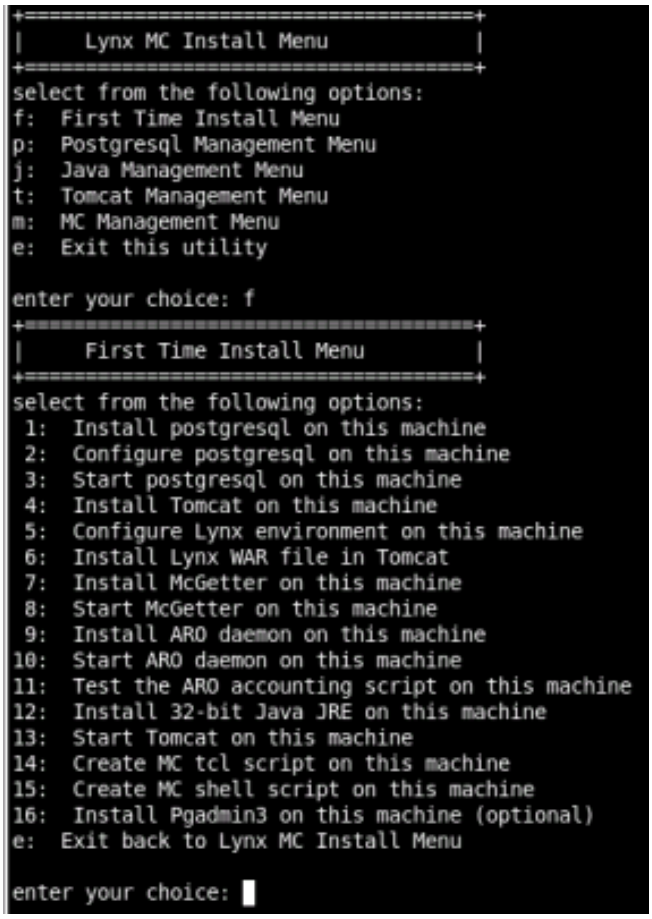


Figure 18: Lynx Management Cockpit Install Menu

Huge improvements have been made with latest Lynx releases, but it still needs some manual operations. Most of the time, a short Webex session with Lynx support team was enough to fix bigger issues.

7. Conclusions

We have explained in this paper how Lynx Design System can allow physical implementation of multiple projects in parallel using a single reference flow while maintaining a high degree of flexibility for the specificities of each project.

We also highlighted the ability of the Management Cockpit to quickly extract relevant metrics for the status or trends of a project.

Even if some areas of improvement exist for this Synopsys workflow, our short-term objective is to use remaining Lynx Design System features (data and integration checks).

8. Acknowledgements

I would thank Sergi Redorta (consultant for Bull) who contributed to the improvement of our physical design flow and methodologies, Ludovic Pinon (Application Consultant at Synopsys) and Aditya Ramachandran (Lynx CAE at Synopsys) for their support on a daily-basis.

9. References

[1] Drag your design environment kicking and screaming into the '90s with Modules! – SNUG Boston 2001