I/O LIBRARY USAGE APPLICATION-NOTE

Version 1.3
July 2003

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Arcadia, C Level Design, C2HDL, C2V, C2VHDL, Calaveras Algorithm, CoCentric, COSSAP, CSim, DelayMill, Design Compiler, DesignPower, DesignWare, Device Model Builder, Enterprise, EPIC, Formality, HSPICE, Hypermodel, I, InSpecs, in-Sync, LEDA, MAST, Meta, Meta-Software, ModelAccess, ModelExpress, ModelTools, PathBlazer, PathMill, Physical Compiler, PowerArc, PowerMill, PrimeTime, RailMill, Raphael, RapidScript, Saber, SmartLogic, SNUG, SolvNet, Stream Driven Simulator, Superlog, System Compiler, TestBench Manager, Testify, TetraMAX, TimeMill, TMA, VERA, and VeriasHDL are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Apollo II, Apollo-DPII, Apollo-GA, ApolloGAII, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BCView, Behavioral Compiler, BOA, BRT, Cedar, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, Cosmos SE, CosmosLE, Cosmos-Scope, Cyclelink, Davinci, DC Expert, DC Expert Plus, DC Professional, DC Ultra, DC Ultra Plus, Design Advisor, Design Analyzer, DesignerHDL, DesignTime, DFM-Workbench, DFT Compiler SoCBIST, Direct RTL, Direct Silicon Access, DW8051, DWPCI, Dynamic-Macromodeling, Dynamic Model Switcher, ECL Compiler, ECO Compiler, EDAnavigator, Encore, Encore PQ, Evaccess, ExpressModel, Floorplan Manager, Formal Model Checker, FormalVera, FoundryModel, FPGA Compiler II, FPGA Express, Frame Compiler, Frameway, Gatran, HDL Advisor, HDL Compiler, Hercules, Hercules-Explorer, Hercules-II, Hierarchical Optimization Technology, High Performance Option, HotPlace, HSPICE-Link, Integrator, Interactive Waveform Viewer, iQBus, Jupiter, Jupiter-DP, JupiterXT, JupiterXT-ASIC, JVXtreme, Liberty, Libra-Passport, Library Compiler, Libra-Visa, LRC, Mars, Mars-Rail, Mars-Xtalk, Medici, Metacapture, Metacircuit, Metamanager, Metamixsim, Milkyway, ModelSource, Module Compiler, MS-3200, MS-3400, NanoSim, Nova Product Family, Nova-ExploreRTL, Nova-Trans, Nova-VeriLint, Nova-VHDLlint, OpenVera, Optimum Silicon, Orion_ec, Parasitic View, Passport, Planet, Planet-PL, Planet-RTL, Polaris, Polaris-CBS, Polaris-MT, Power Compiler, PowerCODE, PowerGate, ProFPGA, Progen, Prospector, Proteus OPC, Protocol Compiler, PSMGen, Raphael-NES, RoadRunner, RTL Analyzer, Saber Co-Simulation, Saber for IC Design, SaberDesigner, SaberGuide, SaberRT, SaberScope, SaberSketch, Saturn, ScanBand, Schematic Compiler, Scirocco, Scirocco-i, Shadow Debugger, Silicon Blueprint, Silicon Early Access, SinglePass-SoC, Smart Extraction, SmartLicense, SmartModel Library, Softwire, Source-Level Design, Star, Star-DC, Star-Hspice, Star-HspiceLink, Star-MS, Star-MTB, Star-Power, Star-Rail, Star-RC, Star-RCXT, Star-Sim, Star-Sim XT, Star-Time, Star-XP, SWIFT, Taurus, Taurus-Device, Taurus-Layout, Taurus-Lithography, Taurus-OPC, Taurus-Process, Taurus-Topography, Taurus-Visual, Taurus-Workbench, Test Compiler, TestGen, TetraMAX TenX, The Power in Semiconductors, TheHDL, TimeSlice, TimeTracker, Timing Annotator, TopoPlace, TopoRoute, Trace-On-Demand, True-Hspice, TSUPREM-4, TymeWare, VCS, VCS Express, VCSi, Venus, Verification Portal, VFormal, VHDL Compiler, VHDL System Simulator, VirSim, and VMC are trademarks of Synopsys, Inc.

Service Marks (℠)

DesignSphere, SVP Café, and TAP-in are service marks of Synopsys, Inc.

SystemC is a trademark of the Open SystemC Initiative and is used under license.
All other product or company names may be trademarks of their respective owners.

Printed in the U.S.A.
Document Order Number:

Table 1: Revision History

Revision	Date	Who	Contents
1.0	24-Dec-98	Remi Peyrouse	Initial revision
1.1	12-Feb-99	Remi Peyrouse	Added STAR_RC flow usage.
1.2	24-Feb-03	Chuanhai Li	Update power pad usages, Fix grammatical errors.
1.3	24-Jul-03	Jiajun Tang	update Star_RCXT flow

Table of contents

A - How to use this document

The purpose of this application note is to give the user of Synopsys tools the basic material to use the Synopsys library pads.

This document can be read as a flow, or as a list of solution for any issue that you can find. These solutions have been checked in our QA testing flow. This document doesn't intend to reflect all the specific cases related to each design but to provide useful solutions. These solutions might not be optimum for your application.

This document gives also hints for the different steps of pad usage, from the verilog netlist to the DRC and LVS with the aspect of the routed pads.

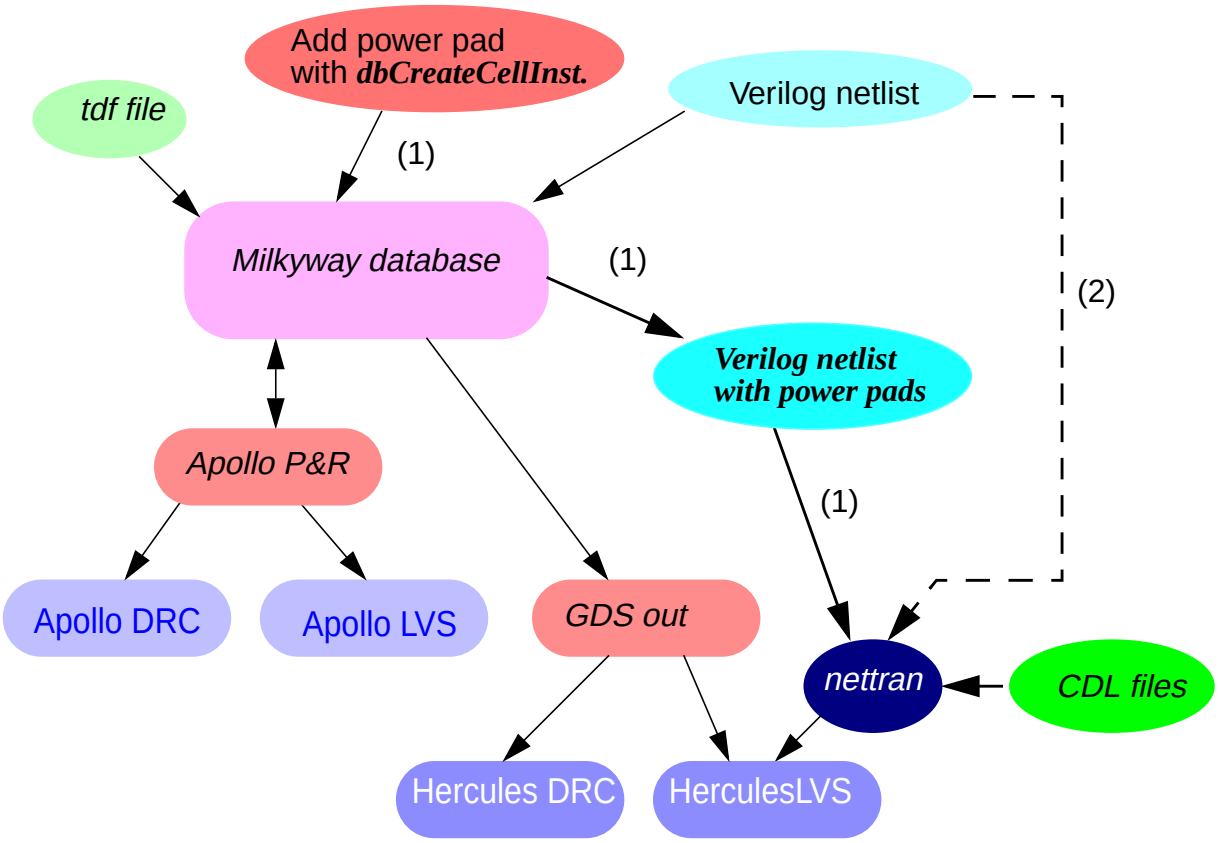
This document is written with the aim to be a generic document independent of the library technology. That is to say this document can be applied to any Synopsys pad library, but does not include specific values for each of them.

In the electronic version of this document, the cross-references are in ***italic red.*** The words that can be found in the index are in ***italic bold.***

B - Design flow

The ***Figure 2.0.0.1*** shows the proposed flow using the IO library.

Figure 2.0.0.1: Proposed flow usage.



There are two options for the Verilog netlist:

-1- pads added in apollo, and new netlist generated including the power pads.

-2- with pads always

The ***Figure 2.0.0.1*** shows both options.

C - Pads types

C.1 Introduction

The libraries that Synopsys provides allow the user to use different IO types

- ***"I/O pads"***.

-***"Clock pads"***

- ***"Crystal oscillator pads"***

- ***"Power pads usages"***

- ***"Corner pads"***

- ***"Pad filler cells"***

The libraries that Synopsys provides allow the user to use different power ***buses*** for the IO ring.

There are 3 ***power buses***:

- ***vdd***: the standard power bus.

- ***vddq:*** the quiet power bus.

- ***vddo***: the I/O power bus (noisy).

The I/Os of the Synopsys libraries use these buses in a novel way. Furthermore there is a patent pending on the IO design that allows to reduce the noise on the noisy power bus.

Power pads that Synopsys provides in the IO libraries allow the power buses to be used in four different ways: ***"Power pads usages"***, ***"Basic solution"***, ***"Intermediate solution (1)"*** and ***"Intermediate solution (2)"***.

C.2 Power pads usages

C.2.1 Optimal solution

The user can use the following combination of pad : ***pvdi***, ***pv0i***, ***pvda***, ***pv0a***, ***pvdd***, ***pv0d***.

Advantage: This combination uses all of the power buses separately to give the optimal noise reduction.

Cost: The cost of this combination is the number of power pads.

C.2.2 Basic solution

The user can use the following combination of pad : ***pvdi, pvde, pv0f***. This combination shorts all the VSSx power buses. You supply all the VSSx power ring with the same power supply.

Advantage:This combination gives the minimum number of power pads.

Cost: This combination doesn't allow you to separate the noisy IO bus from the core power bus. Noise reduction using the patented design is not available.

C.2.3 Intermediate solution (1)

The user can use the following combination of pad : ***pvdi***, ***pv0i***, ***pvde***, ***pv0e***. This combination will allow you to separate the core and IO power supplies..

Advantage: This combination uses less IO than the optimal solution.

Cost: This solution has to be adopted if the core has to be isolated from the noisy IO power ring. Noise reduction using the patented design is not available, thus the IO logic level can be affected by the switching of the output.

C.2.4 Intermediate solution (2)

The user can use the following combination of pad : ***pvdi***, ***pvdd, pv0b***, ***pvda***, ***pv0a***. This combination will allow you to separate the noisy power supply from the quiet power supply. We assume that the core ground supply net is quiet, because this combination short the core ground supply(VSS) with the quiet IO ground supply(VSSQ).

Advantage: This combination uses less IO than the optimal solution. Noise reduction using the patented design is available (assuming that the core power is not too much noisy).

Cost: The noise reduction depends on the noise introduced by the core on the power nodes. This is not so optimal than the Intermediate solution (1).

D - Verilog netlist

In the Verilog netlist the I/O and power pads have to be considered.

D.1 I/O pads

The I/O pads, input pad, output pad, tristate, bidirectional, have to be instantiated with the following syntax (explicit ***declaration*** of the port):

```
pc3b01 I0(.I(IN),.CIN(OUT),.OEN(OEN),.PAD(PAD_pc3b01));
```

This example connects the pin PAD of the 1X dive bidirectional IO to the node PAD_pc3b01.

There is no issue with the regular IO pads, their usage is same as any other macro block.

D.2 Clock pads

The usage of the clock pad is to buffer an internal core signal, for example, the internal clock. This clock has to be provided to the chip through an other I/O type.

The clock pad has to be instantiated with the following syntax:

```
pc3c01 I0(.CP(OUT),.CCLK(IN));
```

The clock pad in the Passport library does not include bonding (pin PAD). Thus the user does not need to connect pin PAD in the Verilog netlist. ***See "Special IO width" on page 17.***

D.3 Crystal oscillator pads

The oscillator pad has to be instantiated with the following syntax:

```
pc3x12 I0(.Z(OUT),.XOUT(OUT),.XIN(IN),.EN(ENB));
```

The oscillator pad in the Passport library includes two bondings (pin XIN and pin XOUT).

For the usage of this pad, Synopsys advises to use the ***"Optimal solution"*** of the power pads set, the ***jitter*** is then reduced due to the ***noise*** attenuation.

D.4 Power pads

There are two ways to create the power pads: either creates the instance in the Verilog netlist or creates them in the ***Apollo*** tool.

In the advised flow, the designer does not include the power pad in the Verilog netlist. The user must use ***dbCreateCellInst*** to add them in the design. ***See "Cell Creation in Apollo" on page 14.***.

This chapter discusses the case where the Verilog includes the power pad description in the netlist.

The main advantage of describing the power pad in the Verilog netlist is that the power pad will be automatically routed to the specified power ring.

D.4.1 Power pad model implementation

The power pad is actually a direct input pad connected to a ***protection*** device. The structure is shown in the ***Figure 4.4.1.1***. The protection device insures the ESD protection, the 5v tolerancy...etc

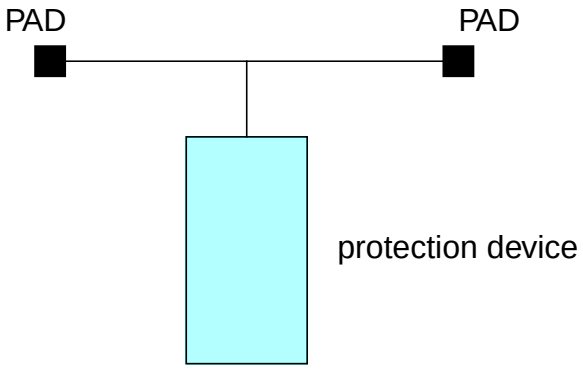


Figure 4.4.1.1

The ***Figure 4.4.1.1*** clearly shows one port in this cell: the node PAD. That is to say the model also will include only one port. Its implementation will have the following aspect (pad pvdi, ***section C.2, "Power pads usages"***).

```
pvdi power0_pvdi(.PAD(VDD));
```

You will notice that you connect VDD to the port PAD. There is no port VDD on the pad pvdi, even if this pad supplies VDD. Our ***convention*** calls the *bonding pad* node: PAD. The port that supplies and where you actually connect your signal is the *bonding pad.*

You have to notice that the node VDD is the global node and can be named as you want:

```
pvdi power0_pvdi(.PAD(yournode));
```

The main purpose of the power pad pin naming (port PAD), is to target multiple power designs. The usage of the name VDD might be confusing. With the Synopsys convention, the usage in the multi-powered chip might be:

```
pvdi power0_pvdi(.PAD(VDD_core));
pvdi power1_pvdi(.PAD(VDD));
pvdi power1_pvdi(.PAD(VDD_clk));
```

Note1: You will notice that in the FRAM view of the library, the pin PAD is PAD_1 or PAD_2. There is no issue on that, the pin name is still PAD, the pin name to be routed in this cell is still PAD.

Note2: The node to be connected with the port PAD is the power node of you design which the user usually connects the pin PAD with ***aprPGConnect.***

D.5 Corner pads

The ***Corner*** pads do not appear in the verilog netlist.

They have to be introduced in the design at the cell instantiation step in ***Apollo***. The Synopsys tool, for instance Apollo, allows to create an instance of a cell that is not in the netlist. ***See "Cell Creation in Apollo" on page 14.***

D.6 Pad filler cells

The filler cells do not appear in the verilog netlist. ***See "Pad filler cells usage" on page 18.***

E - Cell Creation in Apollo

Cells that do not appear in the netlist can be created with the ***Apollo*** command "***dbCreateCellInst"***, for instance the power pads, Corner pads, filler cells.

E.1 Power pads

If we use this method, ***See "Power pads" on page 12.***, to create the ***power*** pad, they will not be routed automatically. We need to indicate the connectivity of the instance that has been created.

You will have to connect the power pad in ***Apollo*** in the netlist before the routing. You will use the command "***aprPGConnect***".

The following example shows how to connect the pad pvdi to the global net VDD (the dialog box is also available in ***Apollo***).

```
aprPGConnect
setFormField "Connect/Disconnect PG" "Net Name" "VDD"
setFormField "Connect/Disconnect PG" "Port Pattern" "PAD.*"
setFormField "Connect/Disconnect PG" "Net Type" "Power"
setToggleField "Connect/Disconnect PG" "Net SubType" "Pad" 0
setToggleField "Connect/Disconnect PG" "Net SubType" "Core" 1
setToggleField "Connect/Disconnect PG" "Cell Types" "Macro" 0
setToggleField "Connect/Disconnect PG" "Cell Types" "Pad" 1
setToggleField "Connect/Disconnect PG" "Cell Types" "Std Cell" 0
setFormField "Connect/Disconnect PG" "Update Tie Up/Down" 1
setFormField "Connect/Disconnect PG" "Cell Master Pattern" "pvdi"
formOK "Connect/Disconnect PG"
formYes "Dialog Box"
```

Note: This method can also be applied to create power and ground connection for ***standards cells***.

E.2 Corner pads

The libraries provide only one corner pad that is used for all 4 Corner. If a library includes 4 Corner, they will have the same design unless specified in the datasheet.

E.2.1 Creation

The syntax to create the ***Corner*** pads is as following:

dbCreateCellInst `(geGetEditCell) "<library full path name>" "<name of the cell>" "<name of instance>" "<rotation>" "NO"  '(coordonnees XY)`

This syntax is also available in the Apollo manual.

The instances usually created are:

```
dbCreateCellInst   (geGetEditCell)   "cb18io220_fr"   "pfrelr.FRAM"
                                     "cornerll" "270" "NO"  '(10 10)
dbCreateCellInst   (geGetEditCell)   "cb18io220_fr"   "pfrelr.FRAM"
                                     "cornerlr" "0" "NO"  '(10 10)
dbCreateCellInst   (geGetEditCell)   "cb18io220_fr"   "pfrelr.FRAM"
                                     "cornerul" "180" "NO"  '(10 10)
dbCreateCellInst   (geGetEditCell)   "cb18io220_fr"   "pfrelr.FRAM"
                                     "cornerur" "90" "NO"  '(10 10)
```

Note: It is possible to create only the upper left corner cell, ***Apollo*** will manage to use this one in the other corner in the correct ***orientation***.

The corner creation operation must be done after the ***binding*** of the Verilog netlist in the Synopsys library (***axgBindNetlist***).

E.2.2 Placement

The placement of the corner pads can be done by hand, or they can be included in the ***tdf*** file common to the ***"Regular IO placement"***.

F - IO placement

F.1 Regular IO placement

The IO ***placement*** is done with the help of a tdf file. The tdf file is loaded after the creation of the instance. ***See "Creation" on page 14.***

The ***tdf*** file includes the datas for the location of the IO and/or their order. This file has to be written by the designer. One helpful solution is to create the tdf file with the following syntax:

```
pad "cornerll" "bottom"
pad "cornerur" "top"
pad "cornerlr" "right"
pad "cornerul" "left"

# Library specific values setup:
define step 86.4       ; iocellwith + spacing
define cornerwidth 391 ; corner cell width
define offset2corner 0 ; first pad abutted to corner

define offset cornerwidth ; corner cell width
pad "I0_pc3b01" "left" 1 (set! offset (+ offset2corner offset))
pad "O1_pc3b01d" "left" 2 (set! offset (+ step offset))
pad "power11_pv0d" "left" 3 (set! offset (+ step offset))
pad "I2_pc3b01u" "left" 4 (set! offset (+ step offset))
pad "O3_pc3b02" "left" 5 (set! offset (+ step offset))
pad "I4_pc3b02d" "left" 6 (set! offset (+ step offset))
pad "O5_pc3b02u" "left" 7 (set! offset (+ step offset))
pad "I6_pc3b03" "left" 8 (set! offset (+ step offset))
pad "O7_pc3b03d" "left" 9 (set! offset (+ step offset))
```

The first line defines the corner pads ***location***.

The define statement allows variables like the ***pitch*** (step), width of corner pad to be defined.

Each pad is then instantiated with the keyword "pad". Each of them is on left, right, top or bottom side. The order is optional, and the last value defines the location. This location is calculated using the offset variable, but can be hard-coded.

F.1.1 Orientation.

In the latest revision of the Synopsys's libraries the ***orientation*** of the pad is predefined.

F.1.2 TDF file (IO location)

The method of defining the IO location is to use a tdf file where the location is explicated relative to the side where it is located.

i.e.:

```
pad "VDD_PAD$8" "right" 0 2738
```

This syntax will place the pad like in the ***Figure 6.1.2.1***.

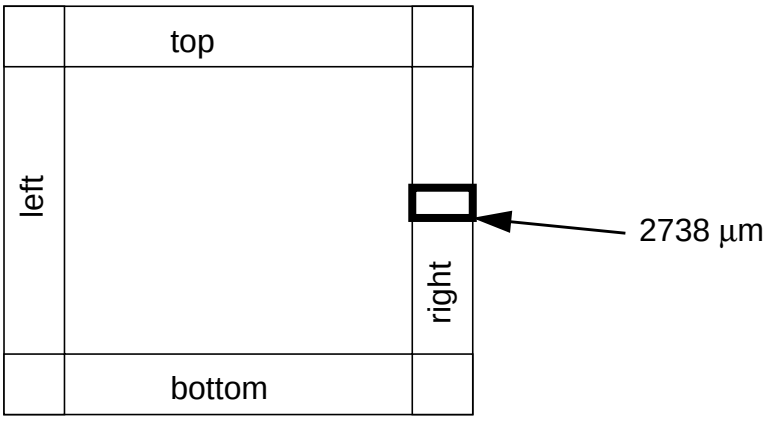


Figure 6.1.2.1: IO location

F.1.3 Pitch

The pitch of the IO is defined in the corresponding library data sheet.

The way to place the IO on a pitch in Apollo is to define a variable offset in the tdf file. ***See "Regular IO placement" on page 16.***

F.1.4 Special IO width

The ***"Clock pads"*** and the ***"Crystal oscillator pads"*** width may be different to that of the other IO of the library.

* The clock pad does not include bonding. Then its width does not follow the ***"Pitch"*** rules. We reduced the width of this pad that might lead the designer to increase the number of IO on the die area.

* The crystal oscillator pad uses two ***sites***. Two bondings occupy these two sites. The ***pitch*** of these two bonding is the minimum pitch allowed by the ***technology***.

F.1.5 Corner pads

See "Placement" on page 15.

F.1.6 Pad filler cells usage

The filler cells are instantiated in the ***Apollo*** tool (see PostPlace menu).

The filler cells must be specified in Apollo in a list from largest to smaller.

The Synopsys IO library uses IO filler to fill the gap that may exist between two IOs. This solution is adopted by Synopsys to extend the well and the diffusion between the IOs. The IO ring must not be connected using the Apollo route IO ring command.

The filler cells use a numbered naming convention. This number is relative to the width size of the filler. The size and number of filler cells depend on the library technology file.

F.2 Staggered IO placement

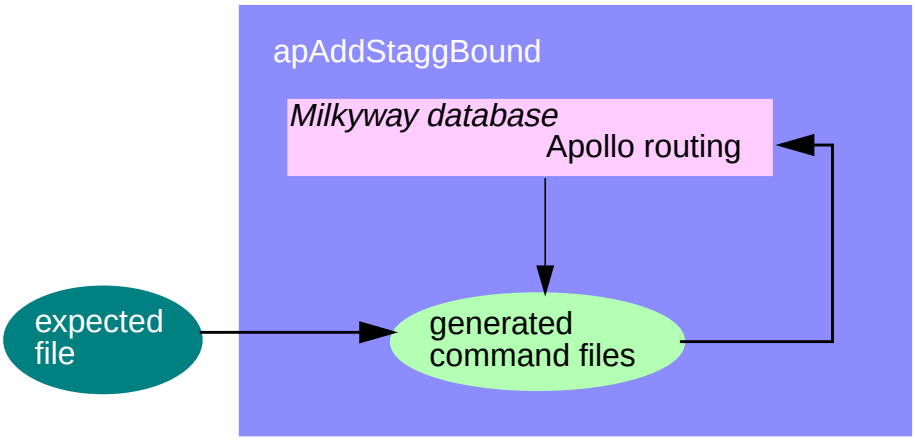


Figure 6.2.0.1apAddStaggBound flow

F.2.1 Introduction

The staggered IO library includes common IO cells. Each cell has a size correspondent to the maximum sizing with ***external*** bonding pad. But the cell does not include the geometries of the bonding.

The implementation of the bondings (internal and external) over the common IO cells is done with two cells. These two cells are provided in the same staggered IO library.

F.2.2 Placement of common IO cells

The placement of the IO cells is actually the same as the regular IO library cells. The flow differs by an extra operation that consists of adding the ***bonding*** over the placed cells.

That is to say the design with staggered IO, is just as easy to use as the design with the regular IO.

The figure ***Figure 6.2.2.1*** shows a routing view in Apollo of the staggered IO , cb18sio220, before the adding of bonding pads.

You can see the blockage of the cells is missing at the left and bottom side of the figure because the bonding has not been added.

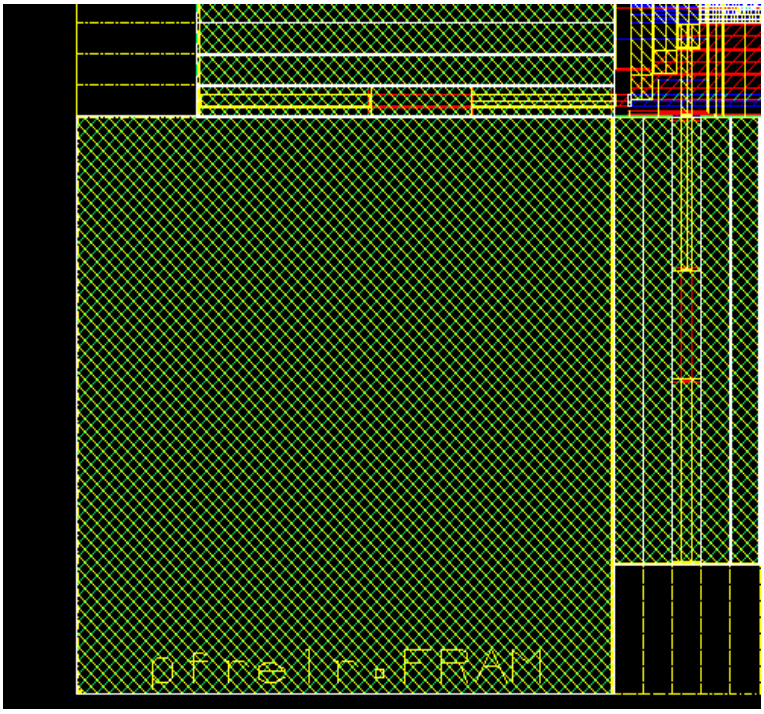


Figure 6.2.2.1Staggered IO routing before bonding placement.

F.2.3 Placement of the bonding

After the placement of the IO as per regular IO, you have to add the internal and external bonding pad.

Synopsys provides a script to achieve this operation. The script that adds staggered bonding pads in Apollo is called ***apAddStagBond***. A ***"TDF file (IO location)"*** has to be loaded previously in the Apollo library prior to use this script.

The purpose of this script is to get the placement and the order of the IO, to place over them the correct bonding pad respectively ***inside*** and ***outside***. The ***clock*** pad and the ***corner*** must be excluded from this operation. To achieve that these instances have to be declared. Meanwhile the ***pci*** pad and the crystal ***oscillator*** pad which use two sites on the ring have to be treated in order to add two consecutive bonding pads on. They have also to be declared.

To achieve the declaration of clock pads and crystal oscillator pads, the designer must complete a file in which the instance names of these special pads are described.

i.e:

```
@EXCLUDED "cornerll" "cornerur" "cornerlr" "cornerul" “pc3c01”
@XALPAD “pc3x11_01” pc3x11_02” “pp5b01”
```

This file (***exp_pad.txt***) is created if the script is run without the option “-exp” that is mandatory.

The usage of the script is:

```
Usage: apAddStagBond -lib <libname> -cell <cellname> -exp <extra-
file>
```

The ***libname*** is the Apollo library design name (your working library).

The ***cellname*** is the cell where bonding will be added.

The ***extrafile*** is the file including the special cells (see above).

If you run the script without option you will see:

```
apAddStagBond > Add bond pad on staggered IO in a library Apollo
design.

            The IO which bond pad will be added have to be defined
in a tdf file that you load in Apollo.
              apAddStagBond generates automatically a template for
"extrafile" if not specified.
           This file should include the excluded pad instance name
that no bonding will added (at least Corner), and the oscillator
instance name for witch twobondings will be added.

               WARNING: no -exp <file> specified, crated template
exp_pad.txt

Usage: apAddStagBond -lib <libname> -cell <cellname> -exp <extra-
file>
```

The file `exp_pad.txt has been created and includes:`

```
@EXCLUDED "cornerll" "cornerur" "cornerlr" "cornerul"
@XALPAD
```

The figure ***Figure 6.2.3.1*** shows the design after bonding placement. The blockage can be seen on all the IO areas. The alternative disposition of the bonding is not visible at this step.

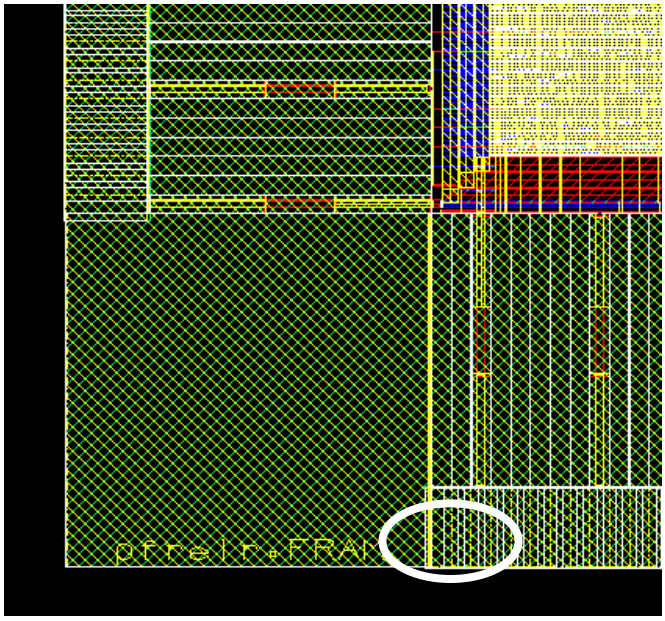


Figure 6.2.3.1Staggered IO routing after bonding placement.

With a zoom on the cell's bottom, it is possible to see the name of the bonding cell overlaps the IO cell's one , see ***Figure 6.2.3.2***.

Figure 6.2.3.2Bonding overlaps an IO..

G - Connect the power pads

G.1 Core power supply

The core can be supplied by either ***pvdi***/***pv0i*** or ***pv0f***, ***pv0b***. Only these power pads can be connected to the power core. Only these pads' ***FRAM*** include a pin accessible from the core to allow routing with ***Apollo***.

G.2 Padring power supply

The pads that supply the padring are either ***pvda***/***pv0a*** or ***pvdd***/***pv0d*** or ***pvde***/***pv0e***. These pads do not include an accessible port and pin to allow routing to the core with ***Apollo***.

H - Aspect of the routing

H.1 IO routing

The figure ***Figure 8.1.0.1*** shows the routing access to the IO in metal2 layer and metal3 layer. On the left hand side top corner, you can see pad filler cells between the corner and the IO.

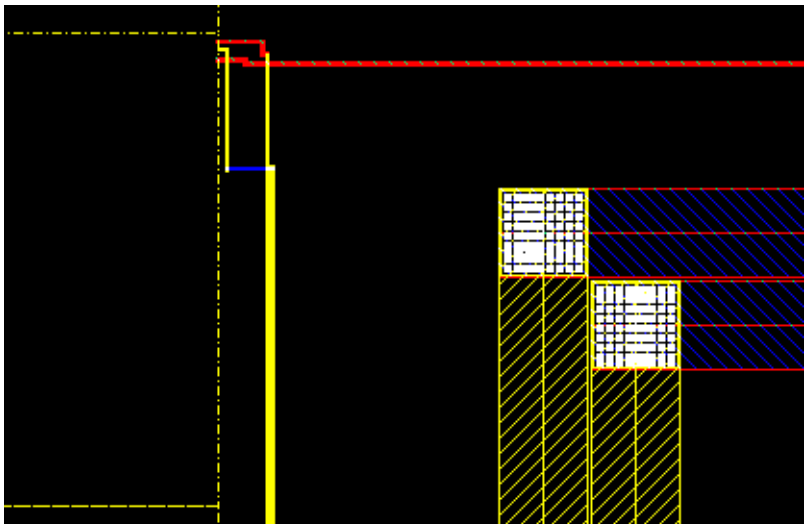


Figure 8.1.0.1: IO routing.

H.2 Clock pad routing

The figure ***Figure 8.2.0.1*** shows the clock pad routed. The clock pad of the cb18io220 has a width smaller than the IO pad. There is no pin PAD on the clock pad because this pad doesn't include bonding.

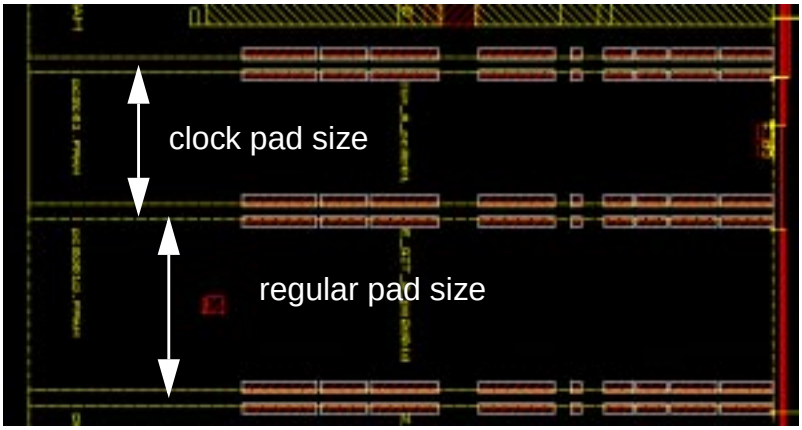


Figure 8.2.0.1: clock pad routing

H.3 Power pad routing

H.3.1 Core power pads

The ***Figure 8.3.1.1*** shows the global aspect of the power pads routed .

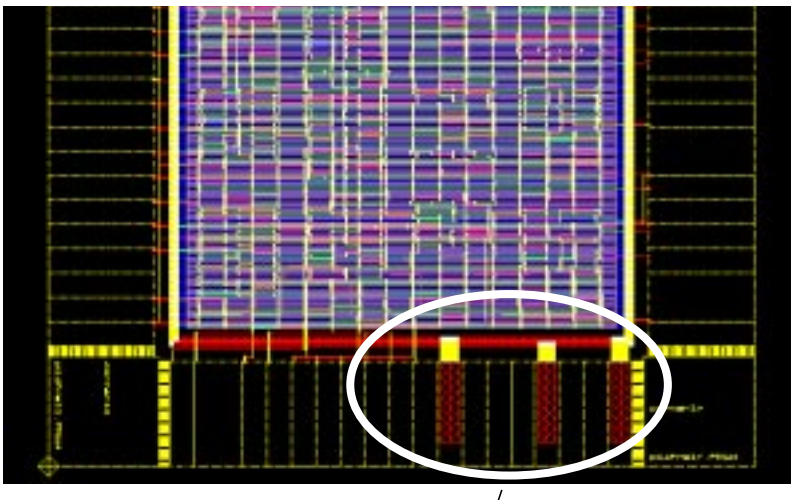


Figure 8.3.1.1Core power pads

The ***Figure 8.3.1.2*** shows the detail of the routing of the power pads.

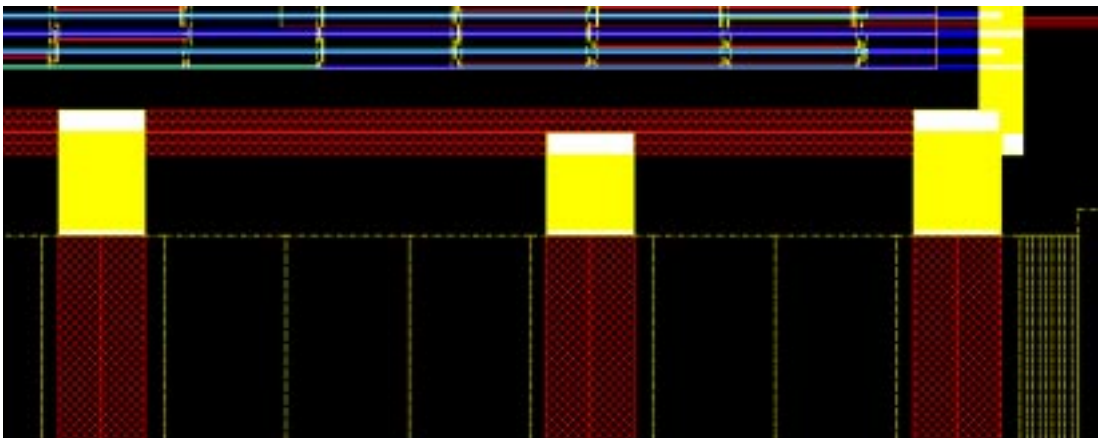


Figure 8.3.1.2Core pad power: detail

H.3.2 Padring power pads

The figure ***Figure 8.3.2.1*** shows the pads pvdi and the pad pvde. The pin PAD of pvde is not routed to the core. Meanwhile the pin PAD of pvdi is routed to the core ring.

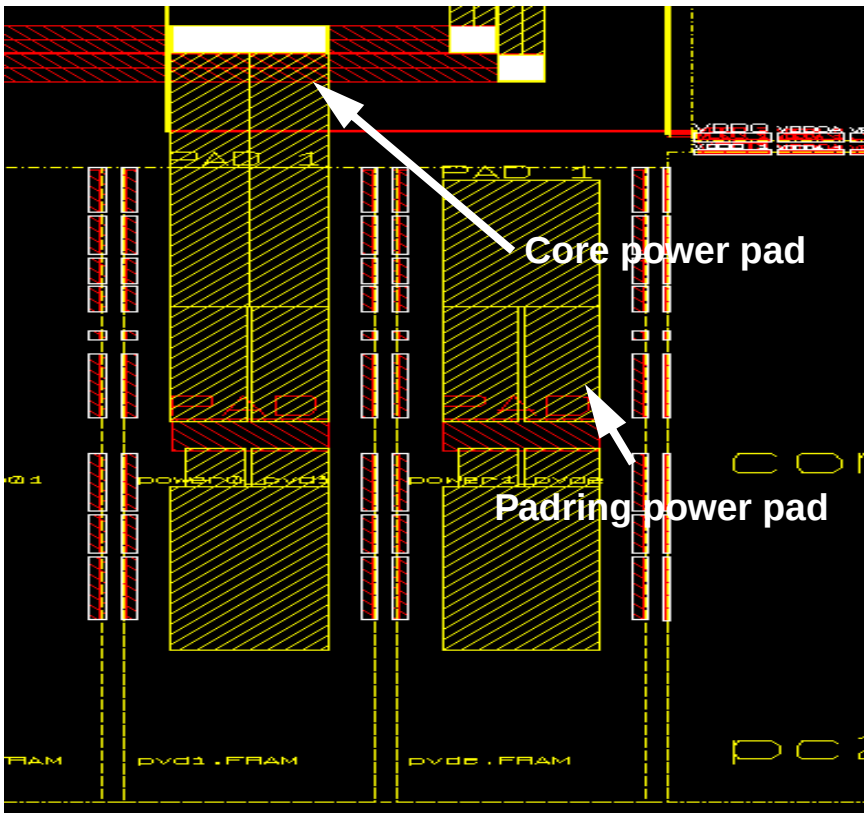


Figure 8.3.2.1: padring power pads

H.4 Staggered IO placement

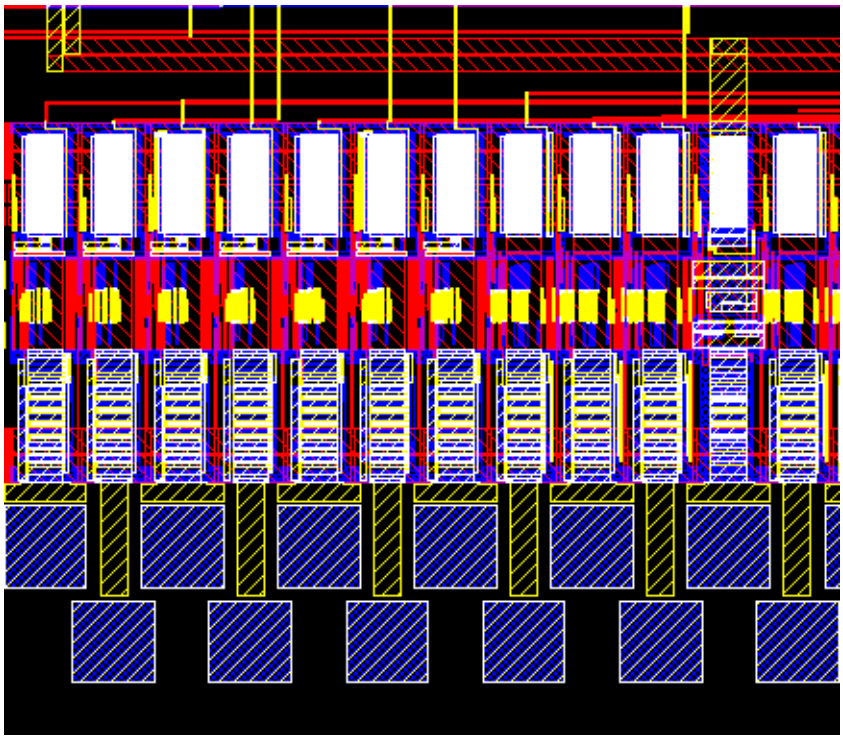


Figure 8.4.0.1: Staggered IO placement

The figure ***Figure 8.4.0.1*** shows the result of the usage of the provided script described in the section ***"Placement of the bonding"***. The inside and outside bondings have been added to the IO. The clock, corner and filler pads are skipped. Two bondings have been added to the oscillator pads respectively, one inside and one outside.

I - LVS

I.1 LVS in Apollo

The following issue will appear if the power pads are instantiated in the Verilog netlist that is bound in the ***Apollo*** library. ***See "Power pads" on page 12.***. Due to the ***"Padring power supply"*** section, the pins PAD will not be routed to any node. The ports PAD are floating

Then you can find this kind of error in the Apollo LVS:

```
ERROR : Floating port PAD, connected to net VDDO, at cell power1_pvda
ERROR : Floating port PAD, connected to net VDDQ, at cell power8_pvdd
ERROR : Floating port PAD, connected to net VSSO, at cell power10_pv0a
ERROR : Floating port PAD, connected to net VSSQ, at cell power11_pv0d
```

This issue **does not appear** if the power pads are not in the Verilog netlist bound in Apollo, and if they are created with the command ***dbCreateCellInst*** .

In any of both cases the LVS with ***Hercules*** gives successful result.

I.2 LVS with HERCULES

The LVS with Hercules can be done by comparing the ***gdsii*** format of the routing with the spice netlist of the design.

The gdsii format is streamed out from the ***Milkyway*** data base.

The spice netlist is generated with the "***nettran***" utility. The netlist Verilog is merged with the cdl of each cells (IO, standard cells, compiler). At this step the Verilog netlist must include the power pads.

I.2.1 General issue in LVS

The general issue in the LVS is to match each net and device of the netlist with each net and device of the extracted layout.

The Hercules LVS deck provided by Synopsys unit includes the option "***pushdown pin***". This option allows the B cell net "net a" and "net b" to be connected by the cell A through the hierarchy. See ***Figure 9.2.1.1***

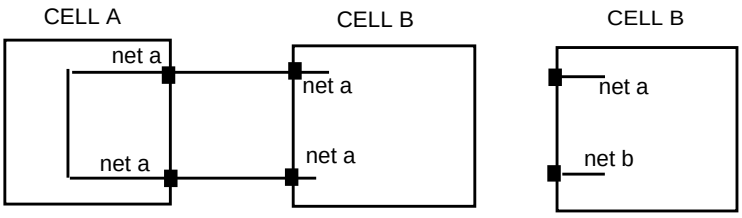


Figure 9.2.1.1

I.2.2 Specific LVS issue in Synopsys IO libraries

The general issue exposed in ***"General issue in DRC"*** appears in the IO LVS with ***Hercules***. The issue is due to the 4 different ***"Power pads usages"***. The power pads used (cell A) short several nets of other IO (cell B). It means the IOs might include different number of nets following the type of power pads used for the power supply.

Synopsys only provides one ***cdl*** library. In this library the netlist of IO includes the 3 power nodes. Each of them is independent (to be used with the ***"Optimal solution"***).

To use the netlist with the other ***"Power pads usages"***, the cdl option ***.EQUIV*** has to be set in the cdl netlist by the user in the following way:

"Basic solution": the **pv**x**f** shorts all power, then all power rings of IO will be shorted. There exists only one power net in the IO, then you must add to the cdl netlist the following:

```
*.EQUIV VDD=VDDQ VDD=VDDO VSS=VSSQ VSS=VSSO
```

"Intermediate solution (1)":the **pv**x**e** shorts quiet and noisy power supply. There exist only two power nets in the IO, then you must add to the cdl netlist the following:

```
*.EQUIV VDDO=VDDQ VSSO=VSSQ
```

"Intermediate solution (2)":the **pv**x**b** shorts quiet and core power supply. There exist only two power nets in the IO, then you must add to the cdl netlist the following:

```
*.EQUIV VDD=VDDQ VSS=VSSQ
```

The user of the LVS must specify the ***topmetal-layer***, this value is either 3,4 or 5. As the Passport IO libraries are designed to be used with several number of metal layers, the drc uses this option to check the configuration of metal number the user choose.

See the release note for any other library LVS specific issue.

J - DRC

J.1 DRC in Apollo

The DRC in Apollo is done on the ***FRAM*** view and does not check the detailed view.

Note:

The DRC has to be done, with the option “Treat fat ***blockages*** as thin wire” if the routing has been done with same option (see Preroute instance form -> DRC). The Passport IO allows this option because the layout has been designed in this purpose.

J.2 DRC with HERCULES

The drc in ***Hercules*** can be run on a ***gdsii*** file format streamed out from the ***Milkyway*** data base.

J.2.1 General issue in DRC

There is no general issue with the DRC in the Passport Libraries.

J.2.2 Specific DRC issue in Synopsys IO libraries

The user of the DRC must specify the ***topmetal-layer***, this value is either 3,4 or 5. As the Passport IO libraries are designed to be used with any of the metal3, metal4 and metal5 as upper metal layer, the drc uses this option to check the right rules of the configuration.

See the release note for any other library DRC specific issue.

K - SDF back annotation

K.1 Introduction

The figure ***Figure 11.1.0.1*** shows the SDF flow generation following the routing in the Synopsys tools ***Figure 2.0.0.1***.

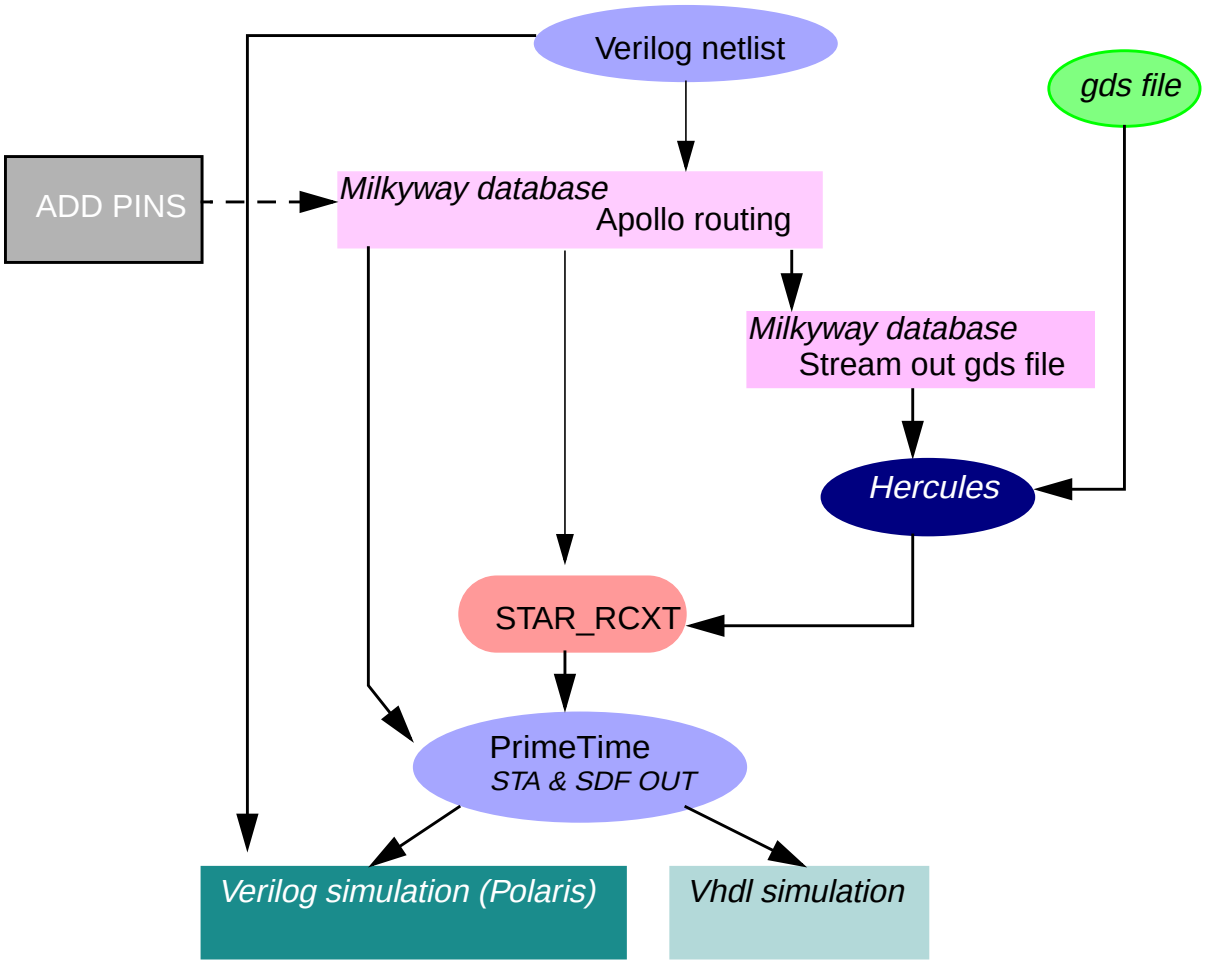


Figure 11.1.0.1: SDF back annotation flow.

In the SDF back annotation flow, star_RCXT can work on your design data which in Milkyway format directly, or another data format generated by Hercules. The result of Star_RCXT can be directly accepted by PrimeTime for STA & SDF out for simulation.

Furthermore due to the huge amount of IO filler cells it is advised to ***explode*** these cells in the database generated by ***Hercules.*** The provided file (.tech) includes this option.

K.2 SDF file generation.

The ***sdf*** file can be generated in three different ways:

"Star_RCXT"

"Apollo SDF output"

"Synopsys output"

The three ways are supported by the IO libraries and are described in the following sub-chapters.

K.2.1 Star_RCXT

The script provided in the .../star_rcxt/.../bin ***runstarxt18_cell***.pl allows to generate easily a ***SPEF*** file from a ***gds*** file. This gds file can be streamed out from ***Milkyway*** database.

then **SPEF** file can be transformed into **SDF** format by using Synopsys Tool **PrimeTime**.

This is more accurate method.

K.2.2 Apollo SDF output

Apollo allows you to generate a SDF file. The interconnection delay is coming from Apollo tech file, and is not extracted by ***Star_RCXT***.

K.2.3 Synopsys output

Synopsys allows you to generate a SDF file. The ***interconnection*** delay is provided from the ***estimated*** wire load. It is not extracted by ***Star_RCXT***, neither correspond to the routing wire length from Apollo.

K.3 PIO file.

The timing analysis using the IO cells need to take in account the load of the PAD port of the IO. This load usually depends on the ***package***. The load information is introduced at different steps depending on the flow.

K.3.1 Star_RCXT

These loads are declared in a file named the Primary IO file (PIO). This file is used by ***star_rxct*** as an option (-pio). The ***IOPATH*** timing in the sdf file takes in account the load specified by the ***pio*** file.

The file might also specified in the ***.tech*** file used by ***star_rcxt*** with the option: ***PRIMARY_IO_FILE***.

K.3.2 Apollo SDF output.

The SDF output from Apollo can take in account the loads on the IO PAD ports if these datas have been previously loaded into the data base. A tdf or a clf file has to be loaded including the following commands:

* tdf file including ***"tdfSetLoadCap"*** and "***tdfSetSrcCapacitanceList***"

* clf file including "***dbSetCellPortTypes***" and "***definePortCapacitance***"

K.4 IOPATH issue for IO cells.

This issue is only relevant with ***"Star_RCXT"*** flow.

In a design including IO cells, some ports are never routed: they are the ports PAD of the IO. These ports are connected to the package. They are not loaded by any wire, their load can be specified in the ***"PIO file."***

If a SDF file is generated from an Apollo routed design, the SDF will not include the timing delay for the IO.

To achieve a SDF file generation including right IO cells' timing information, the designer needs to add a pin above the port PAD of each IO. These pins will be on the same layer as the IO port (usually metal3). The ***Figure 11.4.0.2*** shows a detail of a routed design with the added pins.

No pin will be added on the direct input pad. That is to say no pin will be added on the pc*d00 that is an analog input pad, also the power pads (pv*), the corner pads and the IO filler. One pin will be added on each pad of the crystal oscillator IO.

A script is provided that allows to add these pins on the pads of your library. The extra file includes the pad's instances with no pins added (the filler are automatically excluded). A ***"TDF file (IO location)"*** has to be loaded previously in the Apollo library prior to use this script.

This script is apAddPinBond.pl, its usage is as following:

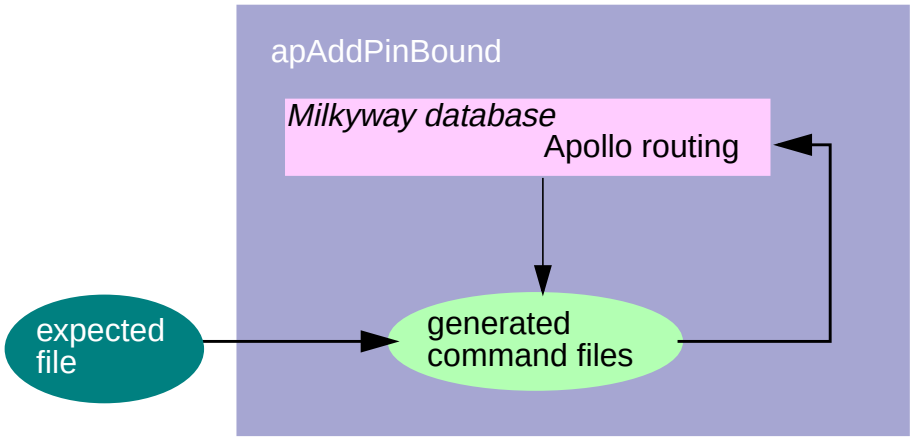


Figure 11.4.0.1apAddPinBound flow

```
apAddPinBond pl > Add bond pad on staggered IO in a library Apollo design.

   The IO which pin pad will be added have to be defined previously in a
tdf file that you load in Apollo.

   apAddPinBond pl generates automatically a template for "extrafile" if
this file has not been specified with the -exp option.
   This file should include the excluded pad INSTANCES name that NO pin
will be added, and the oscillator INSTANCES names for witch TWO pins will
be added.

    Usage:  -lib <libname> -cell <cellname> -exp <extrafile>

    -lib: your working library name.
    -cell: the top cell name witch the pin will be added on.
    -exp: the expected file including the cells which no will not be added
on. The advised instances to declare in this files are those from : Corner
pads,power pads and direct input pads.
```

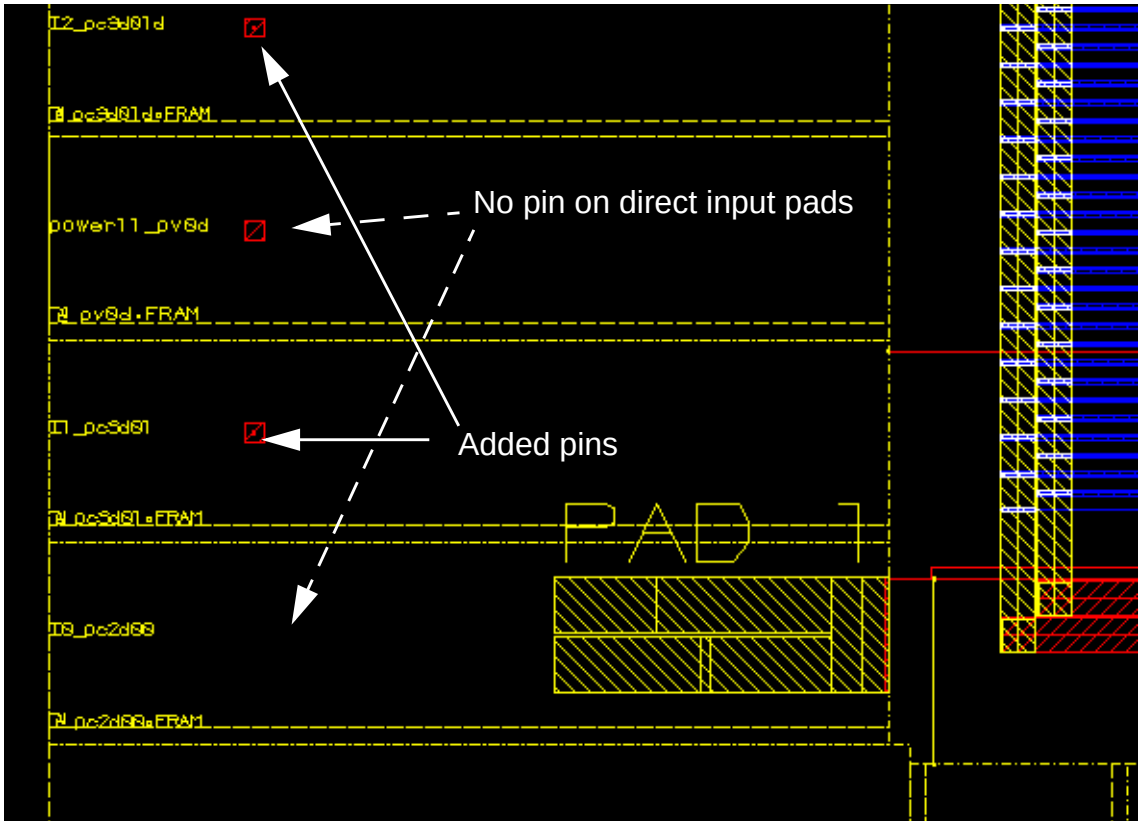


Figure 11.4.0.2Pin placement examples.

K.5 SDF file using Staggered IO.

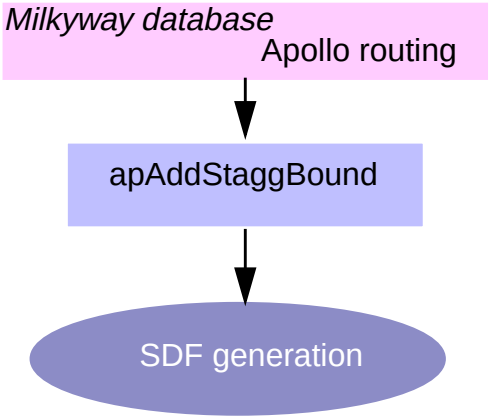


Figure 11.5.0.1

The staggered IO flow usage is described in the ***section F.2, "Staggered IO placement"***.

The bonding pads can be added without issue on the SDF file generation. The

bonding pads generate INTERCONNECT data (see ***Figure 11.5.0.2***) and no IOPATH.

The Verilog netlist doesn't need to be updated to be back annotated.

```
(CELL
  (CELLTYPE "flow_m3_l1_top")
  (INSTANCE )
  (DELAY
   (ABSOLUTE
    (INTERCONNECT I8_pc3t03u/PAD padpin_right_21 (0:0:0) (0:6.96e-
10:2.185e-09))
       (INTERCONNECT  I8_pc3t03u/PAD  bonding_right_8/PAD  (0:0:0)
(0:6.96e-10:2.185e-09))
      (INTERCONNECT  padpin_right_21  bonding_right_8/PAD  (4.688e-
10:4.688e-10:4.688e-10) (4.688e-10:4.688e-10:4.688e-10)))))
```

Figure 11.5.0.2: SDF file sample with staggered bonding.

L - INDEX

X
XALPAD 20