intel foundry

P1278.3 (Intel 18A) PDK 0.9GA Library Development Kit, Cadence Tools

User Guide

Intel Confidential
Revision 1.0, June 2024
Disclosed pursuant to CNDA

Terms and Conditions

Intel Foundry (“Intel”) provides the information in this document in connection with the evaluation, provision, or sale of Intel services and products.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in service contracts for Intel services or in Intel’s Terms and Conditions of sale or use for Intel products, Intel assumes no liability whatsoever in connection with its provision of this document. Intel disclaims any express or implied warranty, relating to sale and/or use of Intel services and products, including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright, or other intellectual property right.

Except as otherwise explicitly and contractually provided, products manufactured, packaged, or otherwise processed by Intel Foundry are not intended for use in medical, lifesaving, or life sustaining applications.

Intel may make changes to its process and/or product specifications or descriptions at any time, without notice.

The information contained in this document may not apply to the Intel service or product that you are evaluating, including for reasons of future purchase or interoperability, or have purchased or intend to purchase from Intel. Further, multiple versions of this document may exist, including newer versions that may or may not be applicable to the Intel service or product of relevance to you. It is your responsibility to identify the Intel document and document version that will meet your needs. Intel assumes no liability whatsoever in connection with your use of or reliance on information herein arising by mistake, including mistakes in document identification.

You may not rely on the absence or characteristics of any features or instructions marked "reserved," "undefined," “evaluation,” or otherwise identified for inclusion or modification in a future version or release. Intel reserves such markings for potential future definition and is not responsible or liable for conflicts or incompatibilities arising from inappropriate reliance.

Intel services or products may contain modifications, revisions, addendums, change summaries, known issue identifications, defects or errors known as errata that may cause the service or product to deviate from previously published specifications. All such changes or errata are available on request.

Contact your local Intel representative, sales office, or distributor to obtain the latest applicable specifications before placing any order for Intel services or products.

Document Revision History

Revision Number	Date	Comments
1.0	June 2024	Initial version. Supports pdk783_r0.9GA.

Contents

Tables

Figures

1 Introduction

This guide provides step-by-step training labs that demonstrate generation of the various collateral required to develop a standard cell library, which enables digital design implementation in P&R (place and route) tools, such as Cadence Innovus*. The labs use a demonstration library that contains a representative sample of standard cells, created from the libraries as supplied by Intel.

The features and flows described in this document might not apply to all situations. Results and images might vary slightly depending on your Intel PDK, layer stack, and Cadence software versions. However, the overall flow remains the same.

Standard cell libraries are a collection of pre-designed and characterized logic cells that are used in the design of digital ICs (Integrated Circuits). These cells are typically optimized for speed, area, and power consumption, and they are available in various sizes and drive strengths. Standard cell libraries are used to implement the logic of an IC, and they are essential to achieve high-performance and low-power designs.

Table 1: Standard cell libraries

Characteristic	Description
Fixed height	All cells in a standard cell library have the same height or multiples of the base height, which makes it easy to place them in rows. This simplifies the automated layout process and makes it possible to achieve high density designs.
Variable width	The width of each cell can vary depending on the drive strength requirements. This allows for trade-offs between performance and area.
Pre-characterized	Each cell in a standard cell library is pre-characterized for its timing, power consumption, and so on.

Table 2: Component cells of a typical standard cell library

Characteristic	Description
Logic gates	The basic building blocks of digital logic, such as AND, OR, NAND, NOR, and XOR gates.
Flip flops and latches	The memory cells store data and are triggered by a clock signal.
Buffers	Buffers drive high fanout loads or isolate different parts of a circuit.
Clock buffers	Specialized buffers designed to drive clock signals.
Scan cells	Scan cells implement scan chains, which are used for testing and debugging.

Table 3: Benefits of using standard cell libraries

Characteristic	Description
Reduced design time	Standard cell libraries are predesigned and characterized, which can significantly reduce the time it takes to design an IC.
Improved design quality	Standard cell libraries are optimized for speed, area, and power consumption, which can help to improve the quality of an IC.
Reduced design cost	Standard cell libraries are commercially available, which can help reduce the cost of designing an IC.

This document is intended for standard cell library developers and provides instructional content to generate the following collateral from a layout and schematic:

- GDS (Stream file)
- OASIS (OASIS design file)
- CDL (Transistor-level netlist)
- Liberty (Timing model)
- Verilog (Functional model)
- Layout abstract (OpenAccess)
- LEF (Library Exchange Format)
- PGV (Power Grid View)

Some content covered in this document requires expert-level knowledge of the format that is being generated, such as LEF, Liberty, and PGV.

1.1 Terminology

Table 4 lists terms and acronyms used in this document.

Table 4: Terminology

Term/Acronym	Description
AG	Abstract Generator
CAD	Computer Aided Design (tools)
CCS	Concurrent Current Source
CCSN	Composite Current Source Noise
CDL	Circuit Description Language
CC	Custom IC
CIW	Command Interpreter Window
CRF	Custom Reference Flow
DEF	Design Exchange Format
DSPF	Detailed Standard Parasitic Forma
GDS	Graphic Database System
GUI	Graphical User Interface
IC	Integrated Circuit
LDK	Library Development Kit
LEF	Library Exchange Format
LVS	Layout Versus Schematic
MOSA	Mixed-Signal OpenAccess
NLDM	Non-linear Delay Model
NLPM	Normal Liter per Minute
OA	OpenAccess
OCV	On-Chip Variation
P&R	Place and Route
PDK	Process Design Kit
PGV	Power Grid View

Term/Acronym	Description
PVS	Cadence Pegasus Verification System*
SPEF	Standard Parasitic Exchange Format
SPICE	Simulation Program Integrated Circuit Emphasis
STA	Static Path Timing Analysis
VALE	Cadence Virtuoso* Application Library Environment*

1.2 LDK directory structure

Figure 1 shows the LDK (Library Development Kit) directory structure.

Figure 1: LDK directory structure

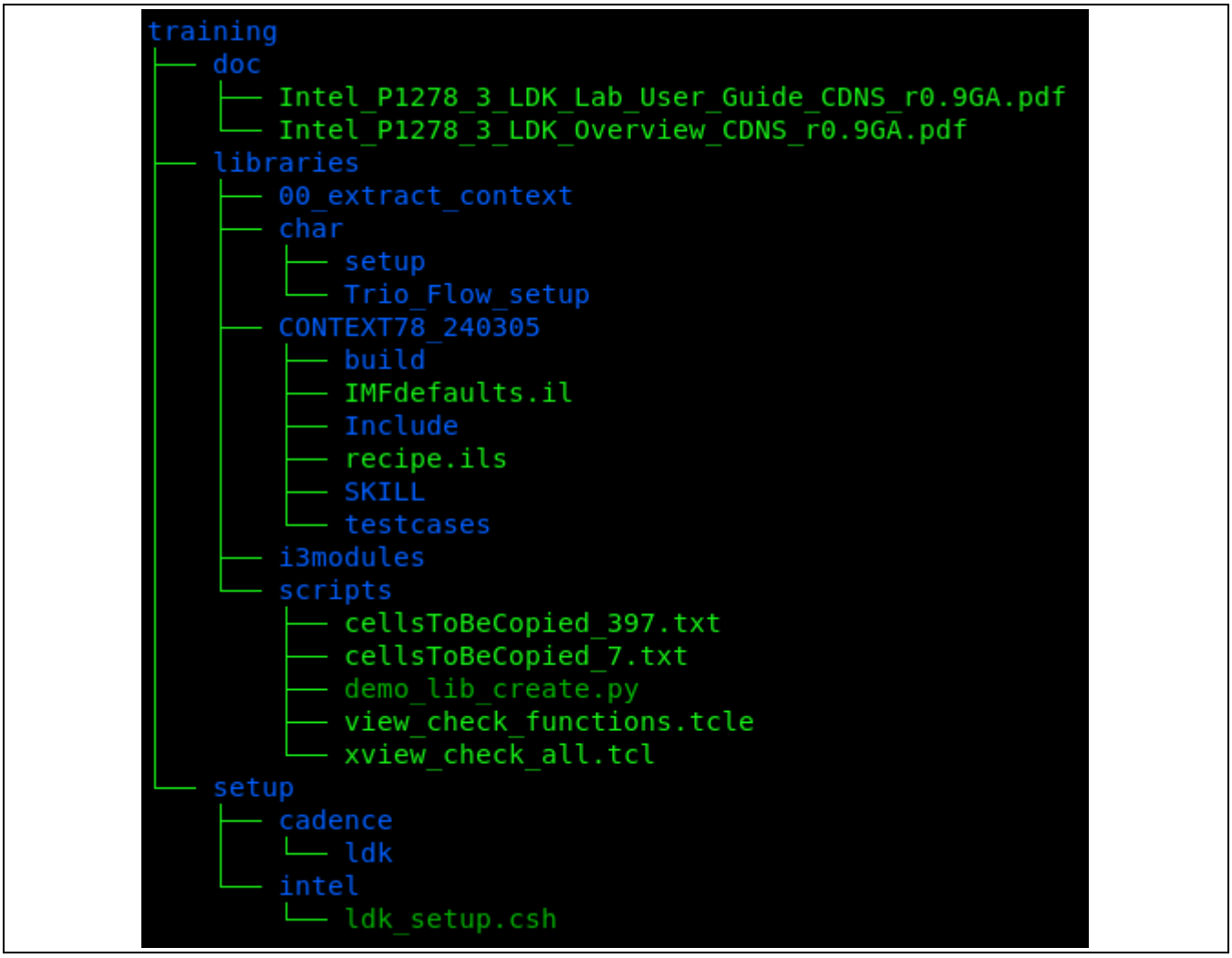


Table 5 provides a description of the directory structure.

Table 5: LDK directory details

Directory/File	Description
P1278.3 (Intel 18A) PDK 0.9GA Process Design Kit User Guide	High-level overview of the P1278.3 (Intel 18A) LDK flow and topics covered in the detailed step-by-step LDK training labs user guide.
P1278.3 (Intel 18A) PDK 0.9GA Library Development Kit User Guide	(This document) User guide with setup information and step-by-step instructions for each LDK training lab.
training/setup/cadence/ldk/cds.lib	File that contains paths to various libraries used in the reference flow.
training/setup/cadence/ldk/.cadence	Directory containing session-specific and tool-specific information stored by Cadence Virtuoso*
training/setup/cadence/ldk/.cdsinit	• File used for tool initialization in Cadence Virtuoso* • Loaded automatically at tool startup • Contains settings for simulation and settings required for the LDK training labs
training/setup/cadence/ldk/files	Directory of additional files used to load specific settings for the LDK training labs.
training/setup/cadence/ldk/sourceme	File that contains SHELL environment variable definitions to set environment variables required for the reference flows. **Note:** Tool paths, such as the Cadence Virtuoso* installation directory, are already configured as part of the user environment.

1.3 Setting up the library development kit

This section covers the tools and kits that you must set up to run the LDK.

1.3.1 Setting up the PDK

Install the following PDK before using the LDK. Refer to Intel *P1278.3 (Intel 18A) PDK 0.9GA Process Design Kit User Guide* for setup information.

PDK version: pdk783_r0.9GA

This LDK training lab uses the opt12 tech option and the following layer stack:

```
m14_2x_1xa_1xb_6ya_2yb_2yc__bm5_1ye_1yf_2ga_mim3x_1gb__bumpp
```

Ensure that the INTEL_PDK and LAYERSTACK shell environment variables are set as outlined in the user guide listed, above.

1.3.2 CAD tools

Table 6 lists the tools and versions used for the LDK training labs. Enable these tools in your environment before running the LDK training labs. Use the version listed, or newer.

Table 6: CAD tools

Vendor	Tool	Version
Cadence	Liberate*	23.12 ISR2, 23.12-s064_1
	MVS*	23.11.001-s034

Vendor	Tool	Version
	Pegasus*	22.24.000-s003
	Quantus*	21.22.017-s209
	Virtuoso*	ICADVM23.1_ISR3
	Voltus*	22.15-e019_1

Note: Most LDK labs use the PDK-aligned tool versions listed above. Changes in tool versions are specified in those labs that require them.

1.3.3 Environment pre-setup for administrator

Perform the following steps to set the environment for the administrator:

1. Extract the LDK kit *tar* file to the desired UNIX directory, and then define the environment variable $INTEL_LDK to point to the installation location for lab users.
2. Make that path read-only for lab users (recommended).
3. Prepare tool environment setup instructions for designers. Refer to the tools specified in CAD tools.
4. Inform users to set the environment variable $INTEL_LDK to point to the original LDK read-only content.
   ```
   setenv INTEL_LDK <LDK installation directory>
   ```
5. Inform users that the LDK training labs require about 10 GB of disk space for each user.
6. If possible, prepare workspace disk area for designers.

1.3.3.1 Creating the demonstration library

Important: The administrator installing the LDK should complete the steps in this section. Creation of the demonstration library is required so that labs can run.

The demonstration library is created using a supplied Python script. The recommended Python version to run the script is v3.7, or higher, for access to the modules used in the script, use *pip install <module_name>*.

The library of standard cells used for the LDK must be created from standard cell libraries provided by Intel. The LDK can be built with a large list of standard cells (397) or a smaller list (7). Table 7 outlines the required libraries to run the demonstration.

Note: It is recommended that you choose the smaller list; all remaining tasks using this collateral proceed much more quickly and are better for learning purposes. If, however, you want to test your compute infrastructure and tool capabilities, use the larger list.

Table 7: Required libraries to have available before running this demonstration

Type	Library	Needed for list
Base	lib783_i0m_180h_50pp_base_hvt	Large
Base	lib783_i0m_180h_50pp_base_lvt	Large
Base	lib783_i0m_180h_50pp_base_svt	Large
Base	lib783_i0m_180h_50pp_base_ulvt	Large, Small
Clk	lib783_i0m_180h_50pp_clk_lvt	Large
Clk	lib783_i0m_180h_50pp_clk_svt	Large
Clk	lib783_i0m_180h_50pp_clk_ulvt	Large

Type	Library	Needed for list
Lvl	lib783_i0m_180h_50pp_lvl_lvt	Large
Lvl	lib783_i0m_180h_50pp_lvl_svt	Large
Lvl	lib783_i0m_180h_50pp_lvl_ulvt	Large
Pwm	lib783_i0m_180h_50pp_pwm_hvt	Large
Pwm	lib783_i0m_180h_50pp_pwm_lvt	Large
Pwm	lib783_i0m_180h_50pp_pwm_svt	Large
Pwm	lib783_i0m_180h_50pp_pwm_ulvt	Large
Seq	lib783_i0m_180h_50pp_seq_hvt	Large
Seq	lib783_i0m_180h_50pp_seq_lvt	Large
Seq	lib783_i0m_180h_50pp_seq_svt	Large
Seq	lib783_i0m_180h_50pp_seq_ulvt	Large, Small
Spcl	lib783_i0m_180h_50pp_spcl_hvt	Large
Spcl	lib783_i0m_180h_50pp_spcl_lvt	Large
Spcl	lib783_i0m_180h_50pp_spcl_svt	Large
Spcl	lib783_i0m_180h_50pp_spcl_ulvt	Large

The script's output is a command file containing the actions required to copy the necessary cells to the $INTEL_LDK/training/libraries/ldk_demo_lib/ directory.

Table 8 describes the base case of running the Python script, from finding all library paths to command file generation:

Table 8: Description of Python script arguments

Arguments	Entry Format
R: root path to standard libraries	Requires full path entry.
L: path of file with library paths	May be either filename or path entry.
M: path of master cell name-library pairs	May be either filename or path entry.
C: file of cells to be copied	One cell name per line.
N: new library path	Requires full path entry.
CF: commands file name	May be either filename or path entry.
V: view(s) to be copied	Must be of the format *layout:schematic:abstract:*, and so on.

1. Change the directory to $INTEL_LDK/training/libraries/scripts
2. Obtain the necessary arguments for each command line entry:
   - The full path to the directory containing the standard cell libraries, referred to as <LDK_STDCELL_LIBRARY_ROOT>.
   - The path to the file listing cells to be copied: **/path/to/CellsToBeCopied.txt**

     Note: This is the only **input** file necessary for the script; choose from CellsToBeCopied_7.txt (small) or CellsToBeCopied_397.txt (large).
3. Run the script to generate the necessary files for command file creation:

```
./demo_lib_create.py [-r  R] [-l  L] [-m  M] [-c  C] [-cf  CF] [-n  N] [-v  V]
```

Filenames may be used as command line entries, as seen in Table 8, for files to be generated in the current working directory; otherwise, the full or relative path is necessary for the script.

- Sample command line entry: (output files are in **green**)

```
./demo_lib_create.py -r <LDK_STDCELL_LIBRARY_ROOT> -l librarySearchPaths.txt -m
mastercellLibPairs.txt -c CellsToBeCopied.txt -cf copyCommands.csh -n
$INTEL_LDK/training/libraries/ldk_demo_lib -v  layout:abstract
```

Note: For all views of a cell to be copied, use *-v all*

4. After executing the command, finding the library paths might take a couple minutes. When finished, the command file to create the new library will appear as specified.
   - Run the command file to create the desired library and cells (Figure 2):

```
source copyCommands.csh
```

Figure 2: Commands executed on the terminal from commandsFile.txt

```
rsync -r --prune-empty-dirs --include="*/" --include="schematic/***" --include="layout/***"
--include="abstract/***" --include="symbol/***" --exclude="*" $INTEL_PDK/library/cellName
$INTEL_LDK/training/libraries/ldk_demo_lib
rsync -r --prune-empty-dirs --include="*/" --include="schematic/***" --include="layout/***"
--include="abstract/***" --include="symbol/***" --exclude="*" $INTEL_PDK/library/cellName
$INTEL_LDK/training/libraries/ldk_demo_lib
rsync -r --prune-empty-dirs --include="*/" --include="schematic/***" --include="layout/***"
--include="abstract/***" --include="symbol/***" --exclude="*" $INTEL_PDK/library/cellName
$INTEL_LDK/training/libraries/ldk_demo_lib
rsync -r --prune-empty-dirs --include="*/" --include="schematic/***" --include="layout/***"
--include="abstract/***" --include="symbol/***" --exclude="*" $INTEL_PDK/library/cellName
$INTEL_LDK/training/libraries/ldk_demo_lib
```

Table 9: Description of additional script purposes

Cases	Arguments	Purpose
Case 1	<input> R: root path to STDCELL libraries <output> L: file with library paths	The script will create a file listing all library paths where standard cells exist.
Case 2	<input> L: library path file from Case 1 <output> M: file containing cell name-library pairs	The script uses the output file from Case 1 to create the master cell-library file.
Case 3	<input> C: file with cell names M: file of cell name-library pairs (Case 2 output) N: path of new library V: view(s) to be copied <output> CF: command file name	The script uses the output file from Case 2 (master cell-library list) and a file containing cells to be copied to generate a command file for creating a new library.

Additionally, the Python script allows you to run separate commands in three separate cases, carrying out the same functionality as the base case run but in three individual steps, as seen in Table 9:

- To run Case 1:

```
./demo_lib_create.py -r <LDK_STDCELL_LIBRARY_ROOT> -l librarySearchPaths.txt
```

- To run Case 2:

```
./demo_lib_create.py -l librarySearchPaths.txt -m mastercellLibPairs.txt
```

- To run Case 3:

```
./demo_lib_create.py -m mastercellLibPairs.txt -c CellsToBeCopied.txt -cf
commandsfile.txt -n $INTEL_LDK/training/libraries/ldk_demo_lib -v  layout:abstract
```

Note: The entry format for the values in each case is as described in Table 8, and as described above for the base case.

For additional help, use: ./demo_lib_create.py [-h]

5. Because the new cells and views are copied into the ldk_demo_lib included in the cds.lib file, start a new session or refresh an existing session of the Cadence Virtuoso* Library Manager to see the new content.

1.3.3.2 Environment variables for LDK setup

Table 10 lists the environment variables required to use the Custom Reference Flow (CRF).

Table 10: Environment variables for LDK setup

Variable name	Path
$INTEL_LDK	<LDK installation directory>

1.3.4 Environment setup for lab user

Follow this procedure from a fresh xterm using tcsh shell before proceeding to labs.

Install the following PDK before using the LDK (this might have been completed by the LDK administrator). Refer to Intel *P1278.3 (Intel 18A) PDK 0.9GA Process Design Kit User Guide* for setup information:

- PDK version: pdk783_r0.9GA

This LDK training lab uses the opt12 tech option and the following layer stack:

```
m14_2x_1xa_1xb_6ya_2yb_2yc__bm5_1ye_1yf_2ga_mim3x_1gb__bumpp
```

1. Ensure that the INTEL_PDK and LAYERSTACK shell environment variables are set as outlined in the user guide for the PDK.
2. Change the directory to a lab workspace with at least 10 GB of available disk space.
   - Your administrator might specify a disk area that can be used.
   - Verify that this directory is empty and has directories for the first use:

```
mkdir "<path to your LDK workspace>"
setenv LDK_WORKAREA "<path to your LDK workspace>"
cd $LDK_WORKAREA/
```

3. Set the environment pointer to the LDK source content as specified by the administrator:

```
setenv INTEL_LDK "<path to tarball/extraction directory>"
```

4. Prepare the work area by sourcing the setup as shown below:
   ```
   source $INTEL_LDK/training/setup/intel/ldk_setup.csh
   ```

This step sets up the CAD tool environment variables and copies the LDK content into your workspace. As a lab user, you can now run the steps of the LDK labs as described in this document.

1.3.5 Reset LDK work area procedure

To return the LDK workspace to its initial state:

1. Close all CAD tools started from the LDK workspace.
2. Close all xterms with LDK environment setup.
3. Remove the files/directory content in the workspace directory.
4. Follow the steps in Environment setup for lab user to reinitialize the lab environment.

1.4 Compute resources

Due to the computationally intensive nature of processing a large number of standard cells, this section provides recommendations for hardware resources to optimize the performance of LDK labs and handle large-scale cell processing tasks.

Section 3, in which the timing (Liberty) models are generated, is the most computationally intensive of all the sections.

The recommended hardware resources for LDK Labs are:

- Small cell list (7 cells)
  - 10 cores, 4G RAM per core
- Large cell list (397 cells)
  - 2000 cores, 4G RAM per core

2 Netlists and Physical Views

This section covers generation of:

- Stream file (GDSII) (Section 2.1)
- OASIS design file (OASIS)
- CDL (Circuit Description Language) (Section 2.3)
- Container cell generation (Section 2.4)
- LVS (Layout Versus Schematic) (Section 2.5)
- Parasitic Extraction and SPEF generation (Section 2.6)

Each of these formats is defined, and a custom utility is demonstrated to generate them for an entire library.

2.1 Stream file (GDSII)

The Stream file format, also referred to as GDS or GDSII, stores mask generation data for designing integrated circuits. It represents a layout of different design layers the way they finally appear on a chip. The Stream format is widely used in the industry for:

- Archiving design data in interchangeable format
- Exchanging intellectual property with other vendors
- Exchanging data between various tools to complete the design cycle
- Transferring data in a compact form between various design groups

To translate a design in the Stream format to/from the OpenAccess database, you can use the XStream translator. XStream consists of two modules, XStream In and XStream Out. XStream In translates designs in the Stream format to the OpenAccess database. XStream Out translates designs from the OpenAccess database to the Stream format.

GDSII is the open-standard Stream file format for transferring or archiving two-dimensional graphical design data. It is a binary, platform-independent format.

A combined GDS contains all standard cells in one file. This file is used during GDS export in Cadence Innovus* to merge in the standard cell layouts.

2.2 OASIS design file (OASIS)

Like a Stream file, the OASIS format defines an encapsulation and interchange format for hierarchical integrated circuit mask layout information. This format also provides specifications to interchange mask data between systems such as EDA software, mask writing tools, and mask inspection repair tools.

The translators consist of two modules, XOasis In and XOasis Out. XOasis In translates designs in the OASIS format to the OpenAccess (OA) database. XOasis Out translates designs from the OpenAccess database to the OASIS format. XOasis translators are based on OASIS SEMI P039-0308 standards.

A combined OASIS file contains all standard cells in one file. This file is used during OASIS export in Cadence Innovus* to merge in the standard cell layouts.

2.3 Circuit Description Language

CDL (Circuit Description Language) is a subset of SPICE (Simulation Program Integrated Circuit Emphasis) language. CDL netlists capture the device-level connectivity and parameters for the standard cells. A combined

CDL netlist contains all standard cells in one file. This file is typically used when running LVS in combination with a Verilog netlist of the design.

2.4 Container cell generation

A container cell is a structured arrangement of shapes on multiple layers that encapsulate a standard cell. These shapes represent the surrounding layer topographies encountered in an actual chip layout, allowing for parasitic extraction that accurately captures the effects of cell placement and routing. LVS and extraction are run on the standard cell placed in a generated container cell.

2.5 LVS

LVS (Layout versus Schematic) is a verification process to ensure that a physical layout matches the circuit schematic (or netlist). LVS is an essential step in the IC design process, as it helps to identify and correct errors that could lead to manufacturing defects or functional failures. In the context of this flow, it is a required step prior to extraction with Cadence Quantus*.

The LVS process involves comparing two netlists: one extracted from the physical layout and one generated from the circuit schematic. The netlists represent the connectivity of the circuit components and the LVS tool checks for any discrepancies between the two netlists. If there are no discrepancies, the layout is said to be LVS-clean.

There are two main steps in the LVS process, as shown in Table 11.

Table 11: LVS process

Step	Description
Extraction	The LVS tool extracts the netlist from the physical layout. This involves identifying the circuit components, such as transistors and resistors, and determining their connections. This is different from parasitic extraction.
Comparison	The LVS tool compares the extracted netlist to the netlist generated from the circuit schematic. If the two netlists match, the layout is said to be LVS-clean.

The LVS results are used during extraction with Cadence Quantus*.

2.6 Parasitic extraction and SPEF generation

Cadence Quantus* creates an extracted view or netlist that includes parasitic and proximity information, which modifies the device model based on the physical implementation. This extracted view is then simulated in the context of the top-level test bench to confirm that performance metrics are met after fabrication. Cadence Quantus* considers the mask shift and line thickness variation information during the extraction. The resulting SPICE netlist, SPEF, or extracted views are then used to determine the impact of these variations on the circuit's electrical performance. An LVS run is required prior to running Cadence Quantus*.

SPEF (Standard Parasitic Exchange Format) is an IEEE standard for representing parasitic resistance, capacitance, and inductance of the wires in a chip in ASCII format. SPEF is used for timing delay calculation and for ensuring the signal integrity of a chip. It is the more popular specification for parasitic exchange between different EDA tools during the various design phases.

An SPEF netlist provides the same information as the DSPF (Detailed Standard Parasitic Format) netlist but in a compressed format that reduces output file size by as much as five times compared to the DSPF output file size. While the DSPF netlist is crafted to be SPICE-equivalent, allowing DSPF netlists to be read by a SPICE simulator, SPEF netlists are incompatible with SPICE.

The extracted netlist used during post-layout verification differs from the schematic netlist in several ways:

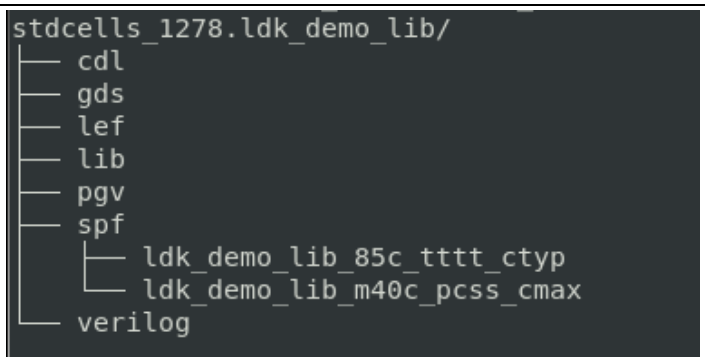
- Device geometries will reflect the actual device geometries found in the layout. For example, the schematic only contains estimates for the source and drain MOS transistor diffusion parameters. In the layout, these will be measured precisely. In addition to source and drain diffusions, the more advanced process may also include WPE (Well Proximity Effect) and LOD (Length of Diffusion) parameters.
- Interconnect parasitics are included for the interconnects. Typically, each net is represented by an RC (Resistor and Capacitor) network. It is also possible to have interconnect inductance effects (L) and mutual coupling (K). However, adding both Ls and Ks to the interconnect model significantly increases the complexity of the netlist and, therefore, the simulation time. These types of parasitics are only extracted for selected critical nets. In general, capacitances should be extracted for all signal nets. In addition, resistances should be extracted for any power nets (to catch IR drops) and any low-impedance and noise-sensitive signal nets.

2.7 User Actions

The generation of the Library Level <Library>.cdl, <Library>.gds and Cell Level <Cell>.spf for each PVT corner is accomplished through a custom utility developed for this LDK.

The delivery directory used in this LDK module uses the structure shown in Figure 3:

Figure 3: Delivery directory structure



The autoLib utility presented here generates output data to a working directory as chosen by the user and is then transferred, as needed, to that delivery directory, above.

There are four main steps:

1. Getting Started (see Section 2.7.1)
2. Auto Lib Generation form (see Section 2.7.2)
3. Mode GDS/CDL generation (see Section 2.7.2.1)
4. Mode SPEF generation (see Section 2.7.2.2)

2.7.1 Getting started

1. Change the directory to the LDK working area ./ldk_r0.9_cdns/training/setup/cadence/ldk, and then source the sourceme file if this is the first time launching the Cadence* tools from the shell you are working in.

   Prior to invoking Cadence Virtuoso*, ensure the Cadence Virtuoso* Application Library Environment (VALE) Framework is available according to the local VALE setup installation and modified accordingly in the <ldk>.cdsinit file (Figure 4).

Figure 4: CDS_MVS_IMF_CM Shell Variable

```
; To enable VALE: (define shell env variable CDS_MVS_IMF_CM to the VALE installation path)

setShellEnvVar("CDS_MVS_IMF_CM" "<pathToVALE>/CONTEXT78_231205")
```

2. Launch Cadence Virtuoso*:

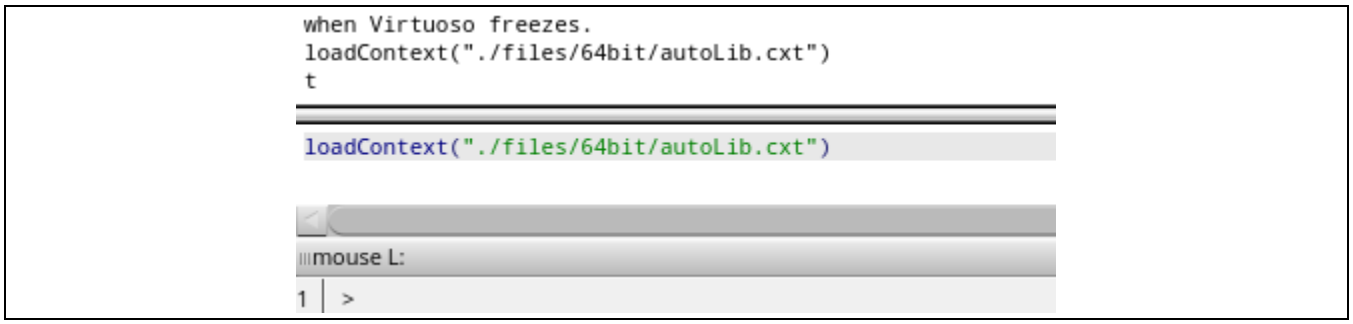
```
virtuoso &
```

   The Command Interpreter Window (CIW) and Library Manager display.

3. Load the autoLib utility in the CIW entry field using command shown in Figure 5 (you can copy/paste the following string):

```
loadContext("./files/64bit/autoLib.cxt")
```

Figure 5: CIW:loadContext command



4. While in the CIW, enter **Bindkey** <F10> to display the Auto Lib View Generation (autoLib) utility (Section 2.7.2).

2.7.2 Auto Lib View Generation form

Figure 6 is the default view for the Auto Lib View Generation form. For this LDK module, there are several defaults to assist in the autoLib generation in the P1278 Technology already present in the form.

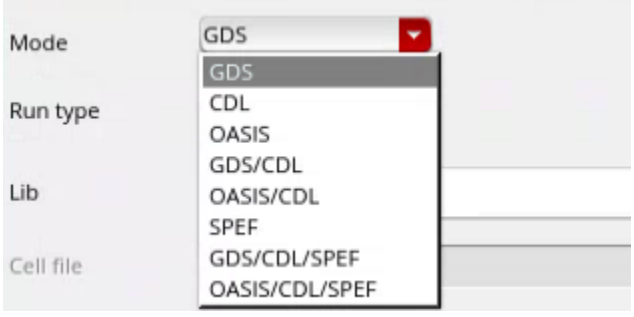
Figure 6: Default view for the Auto Lib View Generation form

Mode:

There are four Modes for output generation (Figure 7):

- GDS for the entire library
- OASIS for the entire library
- CDL for the entire library
- SPEF or Cell.spf for each cell in the library

Figure 7: Mode options for output generation



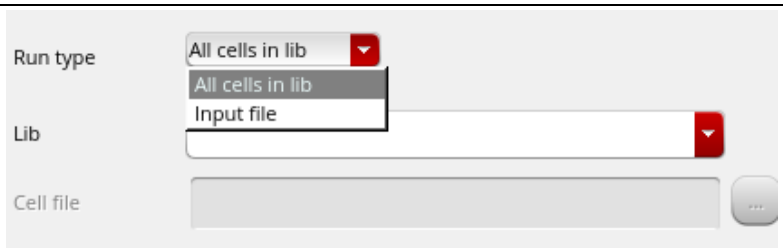
Each output mode can be run independently or there are two options for combined modes. The SPEF mode generates a container cell from library layout and calls in layout_extract. Cadence Pegasus* LVS is run on this container cell to generate needed information for Cadence Quantus* extraction and .spf generation.

Run Type:

There are two options for Run Type to specify a cell list for the autoLib to work with (Figure 8).

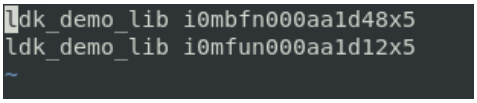
- **All cells in lib:** The entire set of cells from the library specified in the *Lib* field are run for the specified modes.
- **Input file:** Selected operations are only performed on the lib/cell pairs listed in the file.

Figure 8: Run Type options



The Input file should contain only two columns of lib and cell names. Figure 9 shows the libCellFile provided in the current working directory:

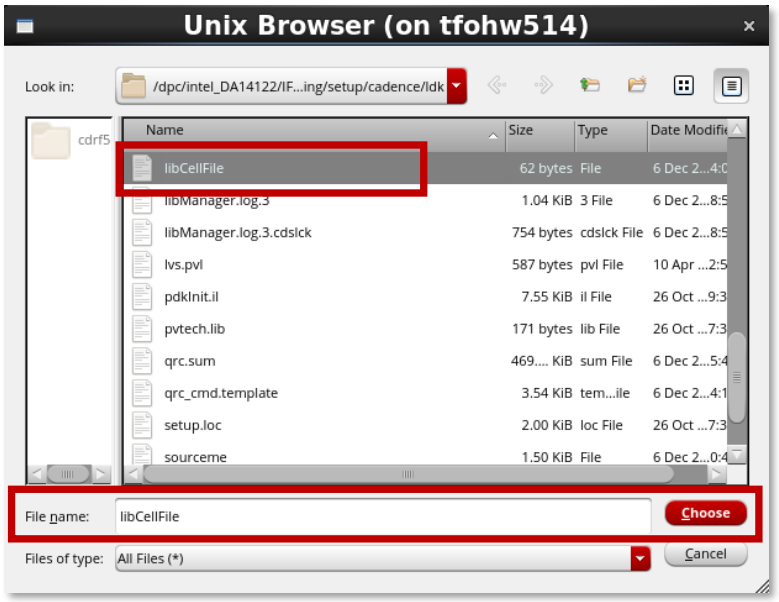
Figure 9: Input file example file structure



2.7.2.1 Mode GDS/CDL generation

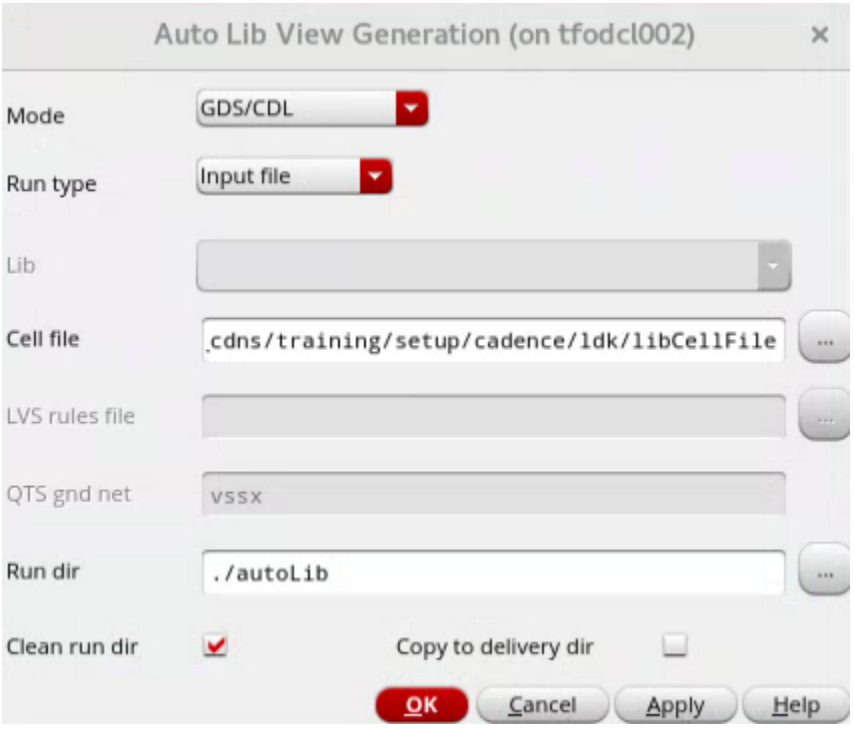
1. In the Auto Lib View Generation form, select **Mode – GDS/CDL**.
2. Select **Run type – Input file.** The *Cell file* field is now editable, while the *Lib* field is not.
3. Select the file chooser on the right of the *Cell file* field to open a UNIX Browser (Figure 10).
4. Select **libCellFile**.
5. Click **Choose**.

Figure 10: UNIX browser for Cell file field



The Auto Lib View Generation form should look like Figure 11:

Figure 11: Auto Lib View Generation form in GDS/CDL mode

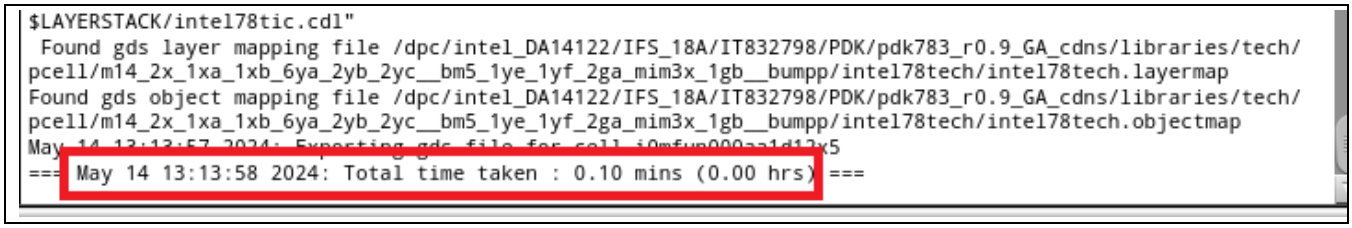


6. Click **Apply**.

   The default Run directory name is ./autoLib; you can change the name, as desired.

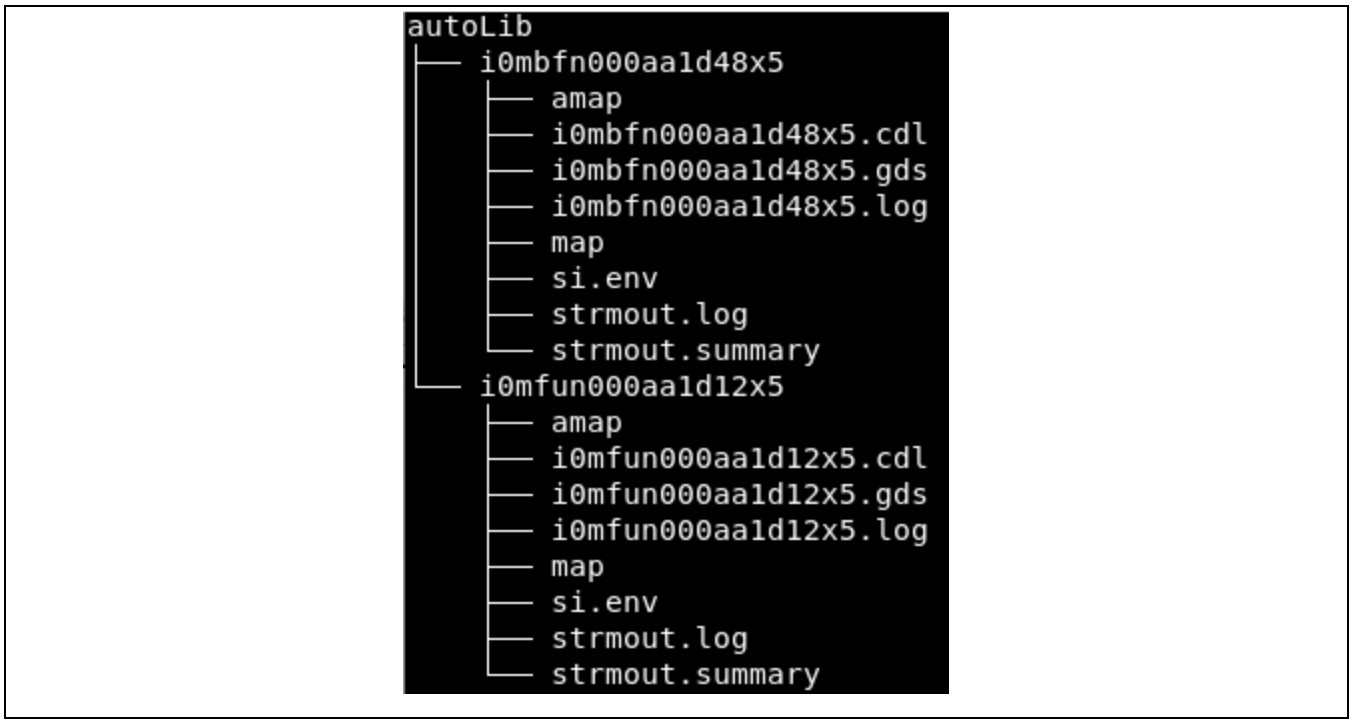
   Several messages are displayed in the CIW output pane. The entire operation is complete when the final message "Total time taken" displays (Figure 12).

Figure 12: CIW output messages



The structure of Run directory autoLib is shown in Figure 13. The individual directories that are also cell names contain the cell-level gds and cdl files. These cell level files are combined for the library level in the ldk_demo_lib.gds and ldk_demo_lib.cdl file when the option *Copy to delivery dir* is enabled.

Figure 13: Run dir autoLib directory structure

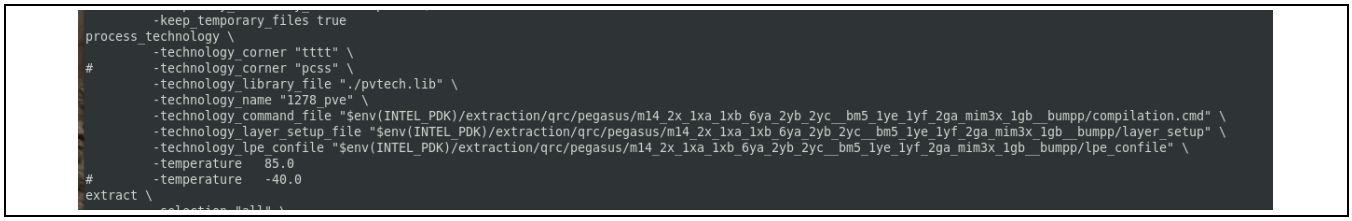


2.7.2.2 Mode SPEF generation

SPEF (.spf) file generation is only run on one set of PVT corners at a time. The file is of dspf format but referred to as SPEF in this LDK. To enable automation for each cell in a library or Input file, a qrc_cmd.template file is used by the autoLib utility to customize the command file for each cell when running Cadence Quantus*.

1. Using your preferred editor, open the ./qrc_cmd.template file. This file requires very little editing and only when switching between the two covered PVT corners in the delivery directory. View the area for process_technology (Figure 14). Close the file when done viewing.

Figure 14: Technology section of qrc_cmd.template file



Note: You can use other corners in the SPEF generation operation, however, the other corners are not available in the delivery directory. Ensure the *Copy to delivery dir* option is deselected if corners other than those listed in the delivery directory are used. Other corners are written in the specified Run dir (Figure 15).

Figure 15: Delivery directory structure

2. SPEF generation requires Cadence Pegasus* LVS to be run before Cadence Quantus* extraction. The autoLib utility needs the full path to the lvs.pvl.

   In the UNIX window enter the following:

   ```
   echo $INTEL_PDK/runsets/pegasus/pvl/lvs.pvl
   ```

   The output in the UNIX terminal is the <full_path_to_lvs.pvl> for the loaded PDK.

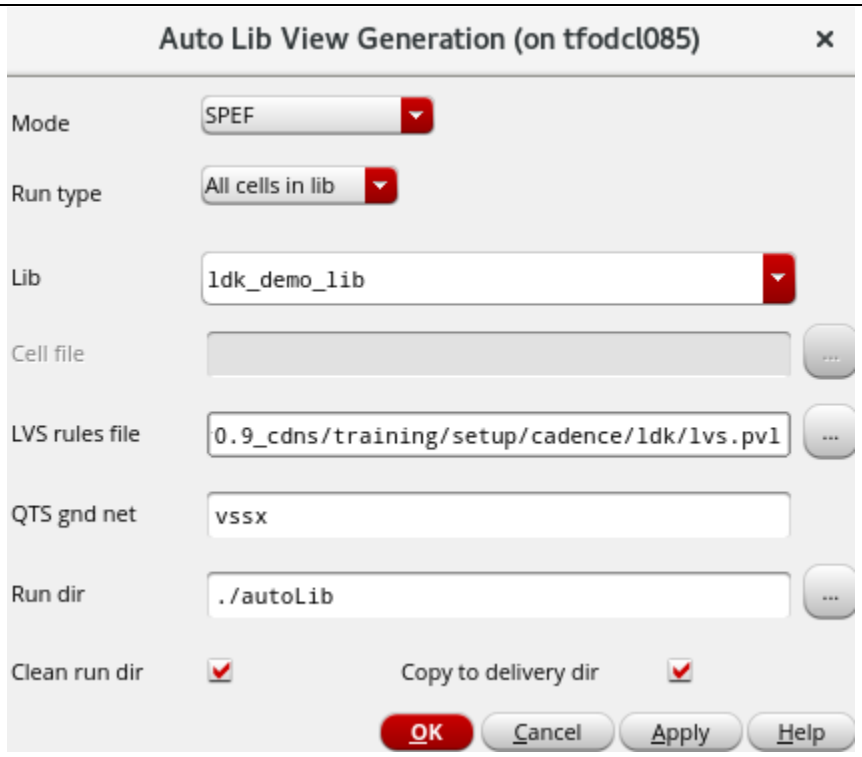
   Copy/paste the full path into the *LVS rules file* field of the *Auto Lib View Generation* UI.

3. If the Auto Lib View Generation form was closed, select **Bindkey** <F10>.

   If running subsequent operations, it is recommended to use **Apply** instead of **OK** on the form to retain entry field values.

4. See Figure 16 for the values for the Auto Lib View Generation form. The *Run type* is now **All cells in lib**. Choose **ldk_demo_lib** from the *Lib* menu. Select **SPEF** from the *Mode* menu and ensure **Copy to delivery dir** is selected. Set *QTS gnd net* to **vssx**.

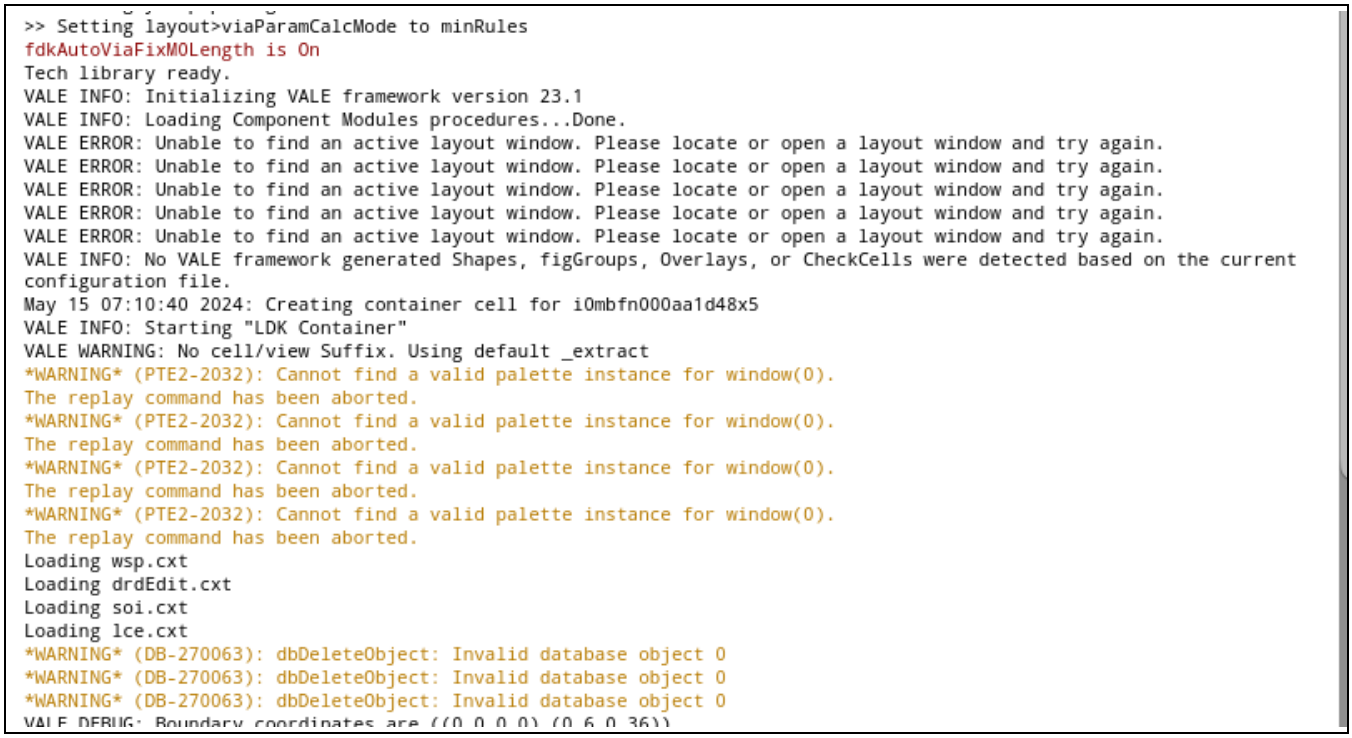
Figure 16: Auto Lib View Generation form in SPEF mode



5. C lick **Apply**. Several messages display in the CIW. The first operation is to create a container cell. A GDS is output for <cell>/layout_extract. A CDL file is written for the <cell>. LVS file and is then run. This can take several minutes for each cell (Figure 17).

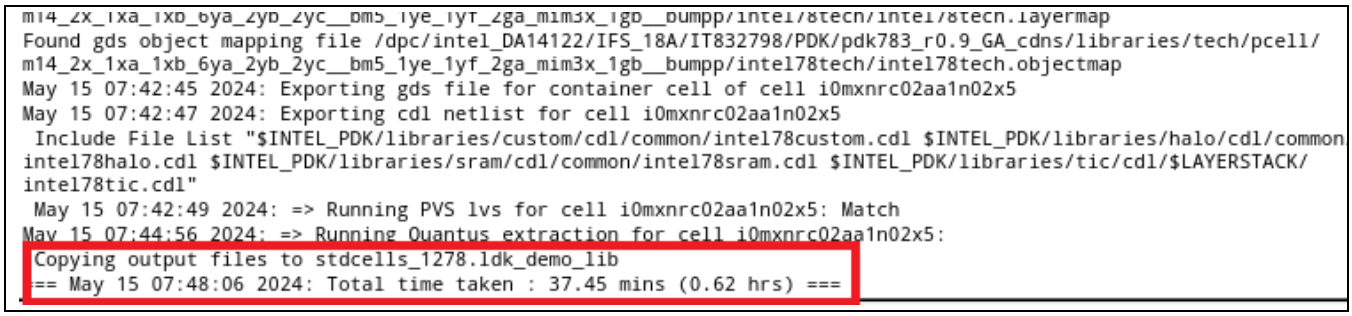
   Note: The VALE messages can be ignored and will be removed in a future release.

Figure 17: CIW Output messages during SPEF generation



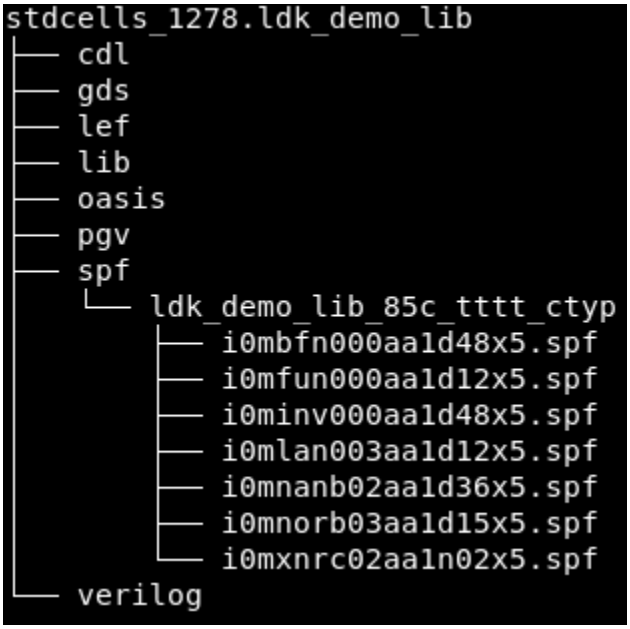
The Keyword messages are displayed while the qrc_cmd.template file is customized for the cell under operation. When the operation is complete after the last cell, a message is displayed to indicate copying of output files to the delivery directory when the **Copy to delivery dir** option is set. When the final message "Total time taken" displays, the entire operation is complete (Figure 18).

Figure 18: CIW Output message after SPEF generation



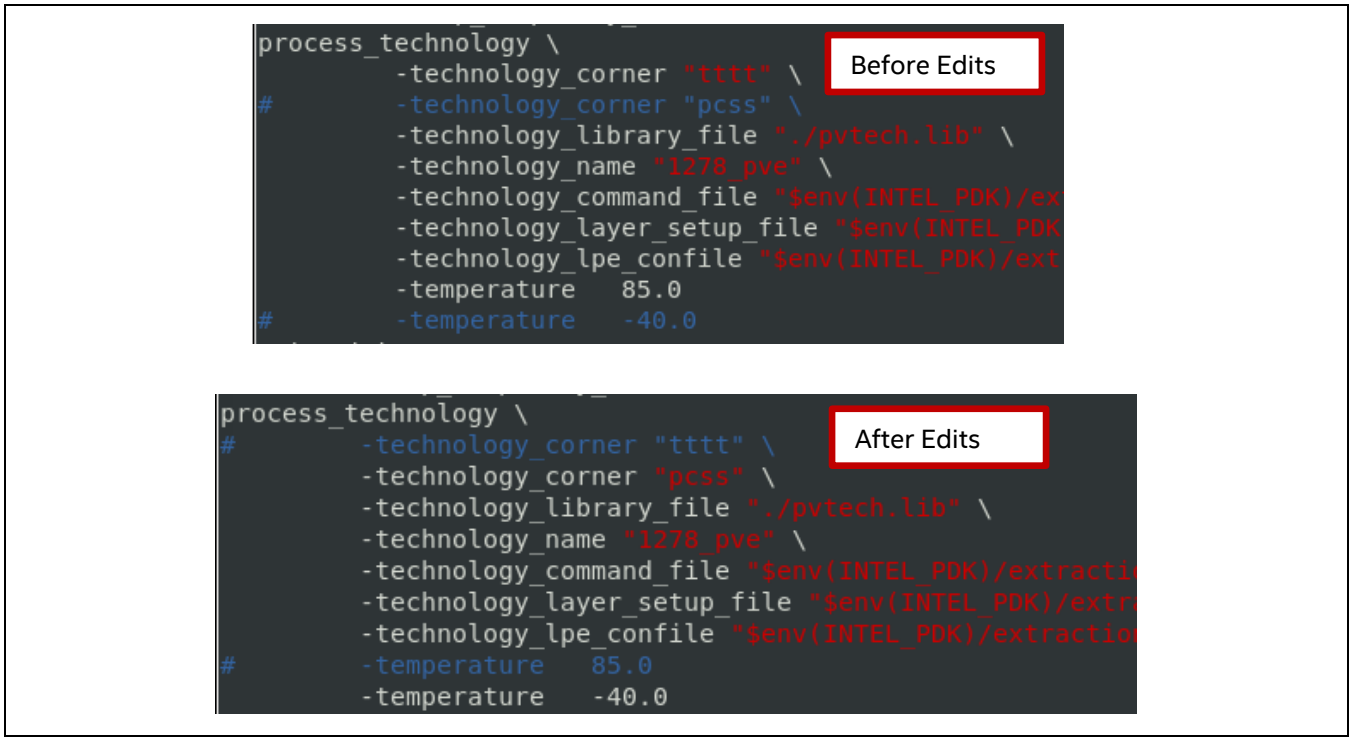
The delivery directory now contains the <cell>.spf files for the specific corner (Figure 19).

Figure 19: Delivery directory - spf first corner



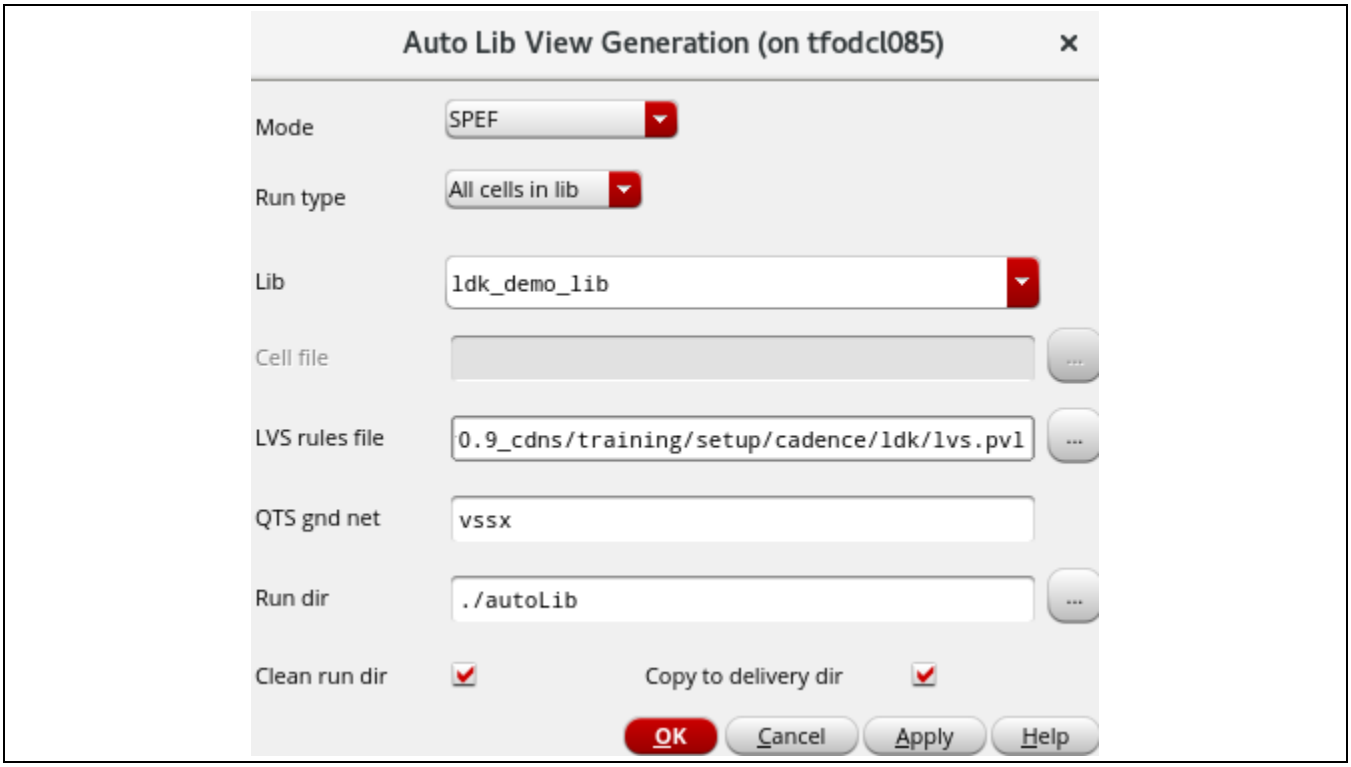
6. Using your preferred editor, open the ./qrc_cmd.template file for editing (Figure 20).
   a. Comment out the original technology_corner “tttt” and uncomment the technology_corner “pcss”.
   b. Comment out the original temperature “85.0”.
   c. Uncomment the temperature line for “-40.0”.
   d. Save and close the file.

Figure 20: qrc_cmd.template edits for spf alternate corner



7. Complete the Auto Lib View Generation form as shown in Figure 21, and then click **Apply**.
   All three modes (GDS/CDL/SPEF) will be run.

Figure 21: Auto Lib View Generation form GDS/CDL/SPEF mode



When the operation is done, <library>.gds and <library>.cdl are also created; all outputs are copied to the delivery directory (Figure 22 and Figure 23).

Figure 22: CIW Output messages - All files moved to delivery directory

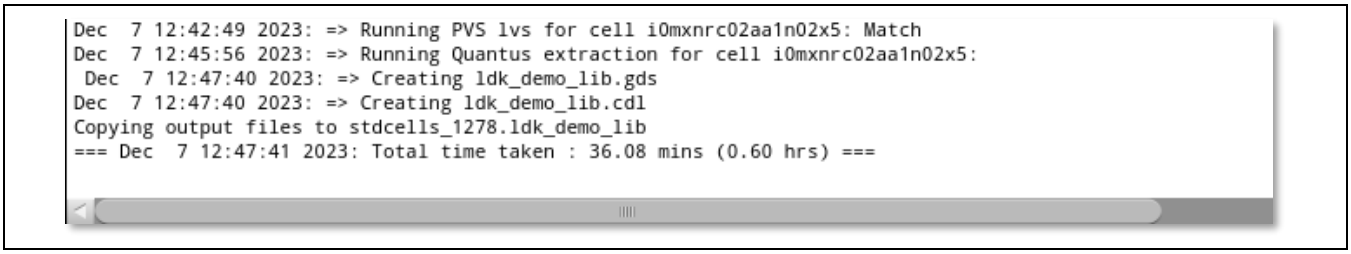
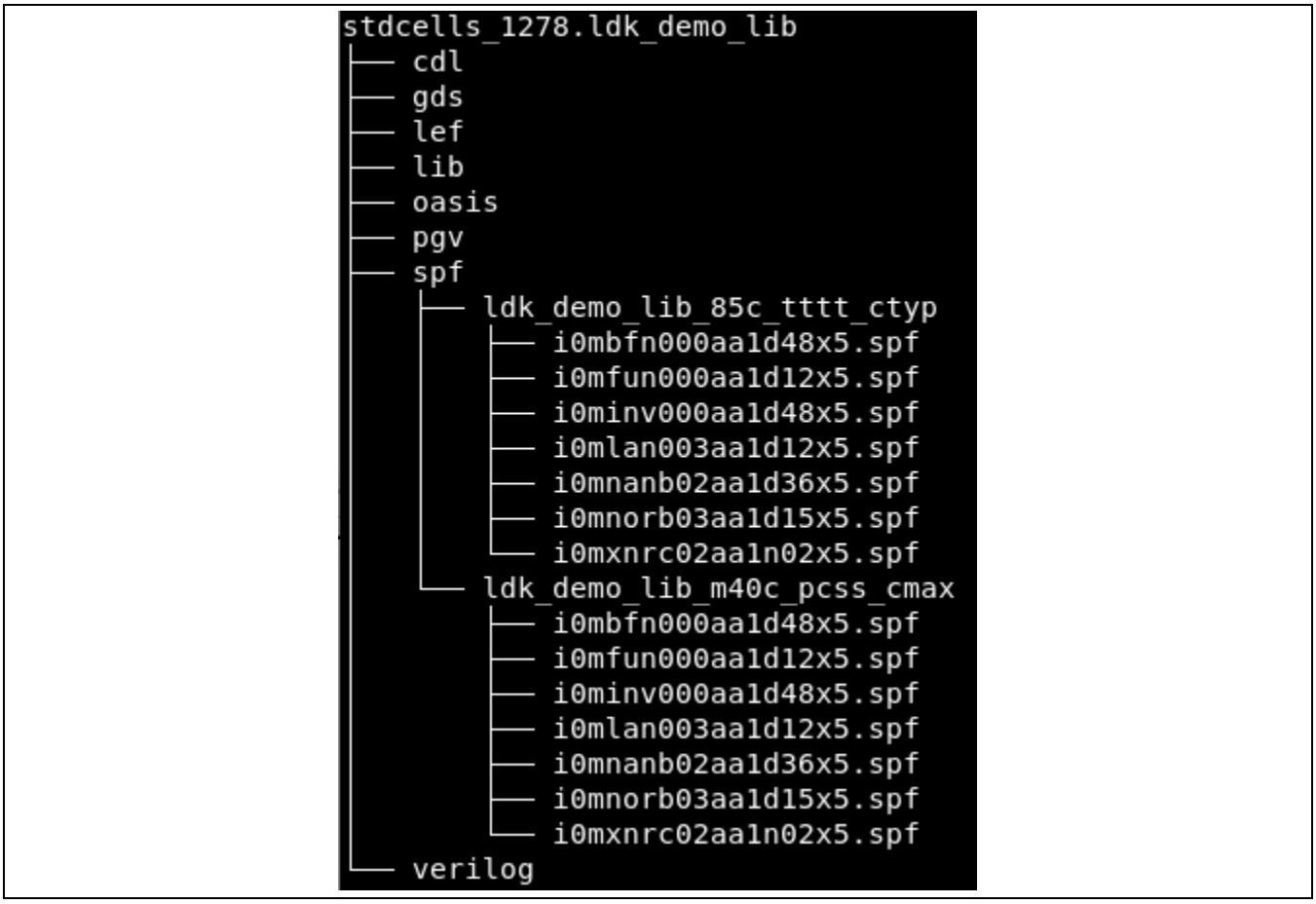


Figure 23: Final step delivery directory output files



3 Timing (Liberty) and Verilog

3.1 Timing (Liberty)

The Liberty timing view is a high-level representation of the timing of the standard cell. It also contains a model of the cell's power consumption. The Liberty model is used by STA (Static Path Timing Analysis) tools, such as Cadence Innovus* and Cadence Tempus* for chip-level timing analysis. It is also used by Power Analysis tools such as Cadence Voltus* for chip-level power analysis.

Cadence Liberate* is the Cadence* library characterization tool that creates the Liberty timing view (.lib file) for standard cells. In this demonstration, Cadence Liberate* is used to characterize several cells from an existing Intel i0m standard cell library.

The standard cell characterization is a two-step process:

1. Generate template, user data, and other characterization setup files.
   - The template file defines the information needed for the cell's characterization.
     - Basic cell information, such as pin name, pin direction, pin type (if it is a clock pin, asynchronous reset or set), supply pins, and so on
     - Arc's table size and index values
     - Optionally, you can manually specify the cell arcs (delay, constraint, power, and so on). For a typical standard cell, Cadence Liberate* automatically determines the cell's function and necessary characterization arcs. Manual arc definition is only done for special cells that Cadence Liberate* does not recognize.
   - The user data file contains information in .lib file that cannot be determined through characterization data, such as cell or pin attributes. One example would be cell area. The user data file is used when Cadence Liberate* generates the final .lib files.
2. Run characterization. In the small cell list, 5 of the 7 cells are in library base_ulvt. The other two cells (a latch and a flip-flop) are in library seq_ulvt. Thus, the demonstration directs you to perform two characterization runs (one for each library).

3.2 Setup environment and run directory

Important: Before proceeding with this section, you must perform the actions in Section 1.3.4, Environment setup for lab user.

1. Change the directory to the LDK working area ./ldk_r0.9_cdns/training/setup/cadence/ldk.
2. Source the sourceme file if this is the first time launching Cadence tools from the shell you are working in.

   To set up the job distribution command, edit the Cadence Liberate* Cluster section in the file $LDK_WORKAREA/training/libraries/char/Trio_Flow_setup/Version1p10/char_tcls/char.tcl (Figure 24).

Figure 24: File char.tcl, Cadence Liberate* Cluster section

```
#################Liberate Cluster##################
## nbjob
# set_var rsh_cmd "nbjob run --target $env(NB_FEEDER) --log-file '/dev/null' --qslot $env(NBQSLOT) --queue $env(NBPOOL)"

## LSF (4G per cpu)
set_var rsh_cmd "/grid/sfi/farm/bin/bsub -q fast-bat -n $threads -J TRIO:CLIENT:MNPVT:${tag} -P MPVT:17.1.4:AE:Eval -W \"200
set_var rsh_kill_cmd "/grid/sfi/farm/bin/bkill " ; # bkill command in your env which allows you to kill jobs
set_var rsh_status_cmd "/grid/sfi/farm/bin/bjobs " ; # bkill command in your env which allows you to check status of jobs

## local machine
# set_var rsh_cmd local
```

Notes:

- The first section is an example of Intel's `nbjobs` command.

  The second section is an example of `lsf` commands used by Cadence*.

  The third section is an example of running on a local machine with multiple CPU cores.

- The back slash character is needed to escape double quote and square bracket used in `rsh_cmd`.

3. Intel standard cell characterization uses driver cell to drive input of standard cells (instead of using an ideal voltage driver).
   a. Update the file $LDK_WORKAREA/training/libraries/char/setup/char_tcls/init.tcl.
   b. Change set DRIVER_CELL_netlist_dir <DRIVER_CELL_dir> to the driver cell netlist directory.
4. Cadence Liberate* characterization requires the use of the Cadence Liberate* Bolt Server for job control and distribution.
   - If the Cadence Liberate* Bolt server is already set up, set the shell environment variable to point to the Bolt server:

     ```
     setenv CDS_BOLT_SERVER <server>
     ```

   - If the Cadence Liberate* Bolt server is not already setup, then start the server on your local machine:

     ```
     $ALTOSHOME/bin/start_bolt
     setenv CDS_BOLT_SERVER `hostname`
     ```

5. Change to the characterization working directory:

   ```
   cd $LDK_WORKAREA/training/setup/cadence/ldk/char
   ```

3.3 Generate template

1. Run the setup script to create the template generation run directory, and then cd to that directory:

```
$INTEL_LDK/training/libraries/char/setup/CreateSetup.csh genTemp
cd genTemp
```

2. To generate the characterization setup files, run the script `run.csh`:

```
./run.csh
```

The run reads all Liberty files in the file list and generates the characterization setup files (Table 12).

Table 12: Characterization setup files

File	Purpose
templates_fi	Template for each cell.
user_data	User data file for each cell.
cells.list_*	Cell list for each library.
tsl_node_*	Custom Intel cell attributes.

The file list (copy of the file genTep_list_libs_ldk_startup) contains only four library files. They cover the seven cells and two PVTs for the small cell list. The full LDK library list is specified in the file genTemp_list_libs_ldk.

The run time for the four libs is about 40 minutes.

3. (Optional) To generate characterization setup files for the large cell list which contains 154 files:

```
cp genTemp_list_libs_ldk list
./run.csh
```

3.4 Run characterization – library base_ulvt

1. Return to your characterization work directory:

```
cd ..
```

2. Run the setup script to create the library characterization run directory, then cd to this directory:

```
$INTEL_LDK/training/libraries/char/setup/CreateSetup.csh i0m base ulvt
cd i0m_base_ulvt
```

3. If desired, edit init.tcl, and then:
   a. Change number of the client jobs:

```
set client <#jobs>
```

   The run time for 20 jobs is about 12 minutes.

   b. To use the extracted netlists generated in Section 2 for characterization, uncomment the following lines:

```
set
netlist_dir  $env(LDK_WORKAREA)/training/setup/cadence/ldk/stdcells_1278.ldk_demo_
lib/spf/

set netlist_area ldk_demo_lib_${spf_temp}c_${stat_skew}_c${default_grp_modeling}
```

4. Execute the script run.csh to run characterization:

```
./run.csh
```

The run characterizes cells specified in the file cell_list and PVT specified in the file active_pvt.pvts in a single run. Two .lib files (per PVT) are generated (Table 13).

Table 13: Lib files generated

.lib File	Models
*_nldm.lib	Basic NLDM / NLPM table model.
*_ccslnt.lib	NLDM / NLPM, CCS, CCSN, Variation.

5. Review the resulting .lib files in the directory ./out_dir_probe_fix_index/PV/allcells/ and then copy the .lib files to the golden database area (Table 13).

```
mkdir -p
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/base_ulvt

cp out_dir_probe_fix_index/PV/allcells/*.lib
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/base_ulvt/
```

Table 14 lists some characterization files to note:

Table 14: Characterization files

Characterization file	Description
cell_list	List of cells to be characterized.
cell_list.*_ldk	Reference cell list of LDK cells.
cell_list.*_ldk_startup	Reference cell list of LDK startup cells.
pvt.tcl	Defines all PVTs (large list with seven PVTs).
active_pvts.list	List of PVT to characterize (subset of PVT defined in file pvt.tcl).
char.tcl	Characterization run script.
init.tcl	Sourced by char.tcl; used for initial user variable setting.

3.5 Generate Verilog model – library base_ulvt

1. Use Cadence Liberate* to create the Verilog model. The tool uses the .lib file as the source information for the cells:

```
liberate --trio $INTEL_LCK/common_tcls/gen_verilog.tcl
out_dir_probe_fix_index/PV/allcells/lib783_i0m_180h_50pp_base_ulvt_tttt_0p700v_85c_tttt
_ctyp_nldm.lib lib783_i0m_180h_50pp_base_ulvt |& tee logs/gen_verilog.log
```

Two Verilog model files are created: cell module and UDP (User-Defined Primitives).

2. Review Verilog model files in the directory ./verilog, and then copy them to the golden database area:

```
cp verilog/*.v
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/verilog/
```

3.6 Run characterization – library seq_ulvt

1. Return to your characterization work directory:

```
cd ..
```

2. Run the setup script to create the library characterization run directory, and then cd to the directory:

```
$INTEL_LDK/training/libraries/char/setup/CreateSetup.csh i0m seq ulvt
cd i0m_seq_ulvt
```

3. Execute the script run.csh to run characterization:

```
./run.csh
```

The run time for 20 jobs is about 25 minutes.

4. Review the resulting .lib files in the directory ./out_dir_probe_fix_index/PV/allcells/ and then copy them to the golden DB area:

```
mkdir -p
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/seq_ulvt

cp out_dir_probe_fix_index/PV/allcells/*.lib
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/seq_ulvt/
```

3.7 Generate Verilog model – library seq_ulvt

1. Use Cadence Liberate* to create the Verilog model. The tool uses the .lib file as the source information for the cells:

```
liberate --trio $INTEL_LCK/common_tcls/gen_verilog.tcl
out_dir_probe_fix_index/PV/allcells/lib783_i0m_180h_50pp_seq_ulvt_tttt_0p700v_85c_tttt_
ctyp_nldm.lib lib783_i0m_180h_50pp_seq_ulvt |& tee logs/gen_verilog.log
```

2. Review the Verilog model files in the directory ./verilog, and then copy to the golden DB area:

```
cp verilog/*.v
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/verilog/
```

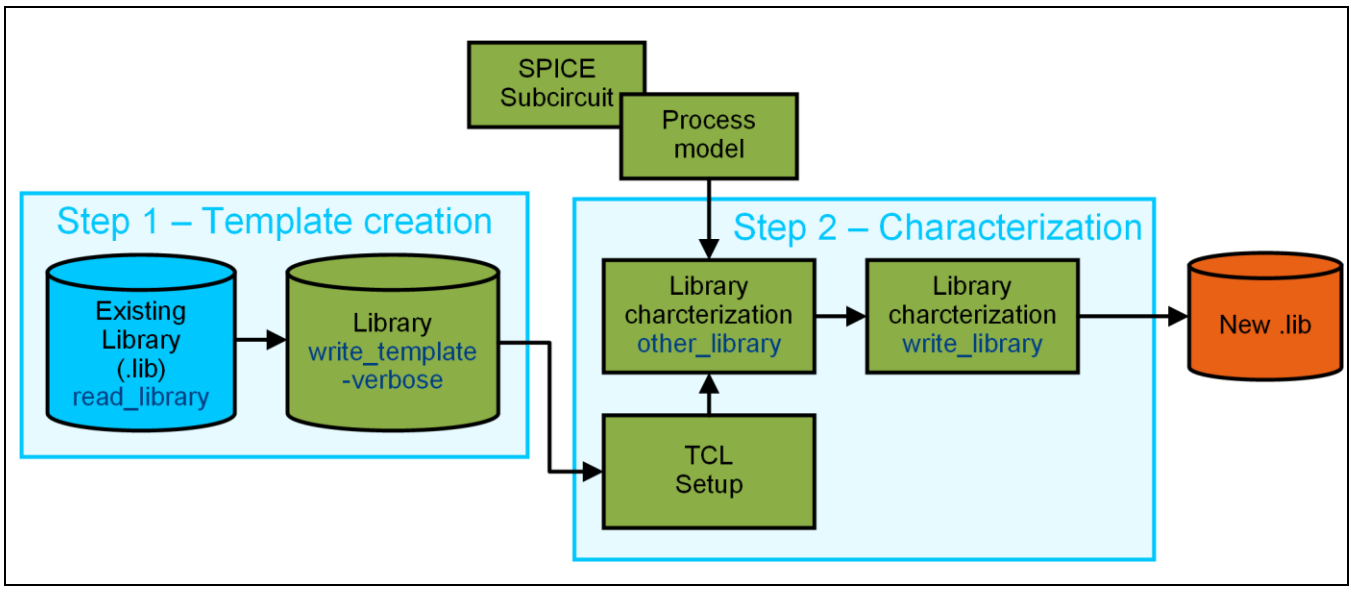
3.8 Library Characterization Kit (LCK)

3.8.1 Overview

At a high level, Liberate characterization is separated into two major steps:

- Template creation
- Characterization

Figure 25: Characterization flow



Note that the LCK (Library Characterization Kit) released is independent of the LDK. For convenience, a version of LCK is included with the LDK release, but might not be the latest released version. You can set the shell environment variable INTEL_LCK to point to the latest version.

3.8.2 Liberate shell environment variable setup

Use the following variables Liberate shell environment variables to set up the Liberate environment:

ALTOSHOME: Liberate install path:

setenv ALTOSHOME <Liberate install path>

PATH: Add Liberate executable path to shell search path:

setenv PATH $ALTOSHOME.bin:$PATH

CDS_AUTO_64BIT: Use 64bit executable for both Liberate and Spectre:

setenv CDS_AUTO_64BIT ALL

CDS_LIC_FILE: Specify license server; syntax = port@hostname:

setenv CDS_LIC_FILE 5280@linws21

ALTOS_QUEUE: Enable license queuing:

setenv ALTOS_QUEUE 1

3.8.3 Template creation (GenTemp)

The template file specifies basic library parameters and cell-specific details. Library parameters such as delay and slew measurement thresholds, min_transition and max_transition, and cell specific details such as pin name, input slew, and output load. For this PDK, the template file information is extracted from an existing Liberty file in the template creation step.

To generate template files:

- **Read Liberty file for cell list and tsl attributes.** Other required attributes are added here.
- **Write_template**
- **Update template**: Any custom update to the template is performed here.
  - Added loop for define_leakage/define to source on client only [Runtime advantage]
  - Updated "special cell" template based on methodology

Figure 26: Section of file genTemp/run.csh

```
#Template generation
if ($flowFlag == genTemp || $flowFlag == genTemp_char) then
   if ($tempFlag == fix_index ) then
      foreach libs (`cat  $lib_list `)
         set lib=$libs
         set libname=`echo "$libs" |awk -F "/" '{print$NF}'`
         set VT_kit=`echo $libname |cut -d _ -f5,6`
         if ($libname =~ *lvl*) then
           set pvt=`echo $libname | cut -d _ -f7,8,10` ; #No issue with <V1>_<V1> PVTs, Need to cross check for <V1>_<V2> PVTs
         else
           set pvt=`echo $libname | cut -d _ -f7,8,9`
         endif
         echo "The $libname is for VT_kit : $VT_kit with $pvt generating template and user data"
         mkdir -p templates_fi/$VT_kit/$pvt
         # generate cell map file.. make sure to check sample map files in flow provided and you may need to make some changes in genCellMap.csh based on source lib file
         liberate --trio ${LTRIO_SCRIPTS_DIR}/genCellMap_fi.tcl  $lib | & tee logs/log.tmpl_map_fi_$tempFlag
         # generate cell level template
         liberate --trio ${LTRIO_SCRIPTS_DIR}/write_template.tcl cells.list_unique $lib $tempFlag | & tee logs/log.wt_$tempFlag
         liberate --trio ${LTRIO_SCRIPTS_DIR}/genfixIndexTmpl.tcl cells.map_unique control_fix_index templates_fi/$VT_kit/$pvt/ | & tee logs/log.tmpl_fmly_fi_${VT_kit}_${pvt}
         mkdir -p user_data
         liberate --trio ${LTRIO_SCRIPTS_DIR}/genUserData.tcl $lib user_data/$VT_kit/$pvt/ | & tee logs/log.usrdata_${tempFlag}_${VT_kit}_${pvt}
         #mkdir Gentemp_celllist
         cp -rpf cells.list_all cells.list_all_${VT_kit}_${pvt}
         cp -rpf tsl_node_all tsl_node_all_${VT_kit}_${pvt}
         end
      endif
   endif
endif
```

Scripts used in the Template Creation (located in the directory common_tcls):

- genCellMap_fi.tcl
- write_template.tcl
- genfixIndexTmpl.tcl
  - Add packet_slave
  - set_pin_VDD/GND
  - Modified "special cell" template
- genUserData.tcl

3.8.4 Characterization – Unified, MPVT

Figure 27: Liberate architecture overview

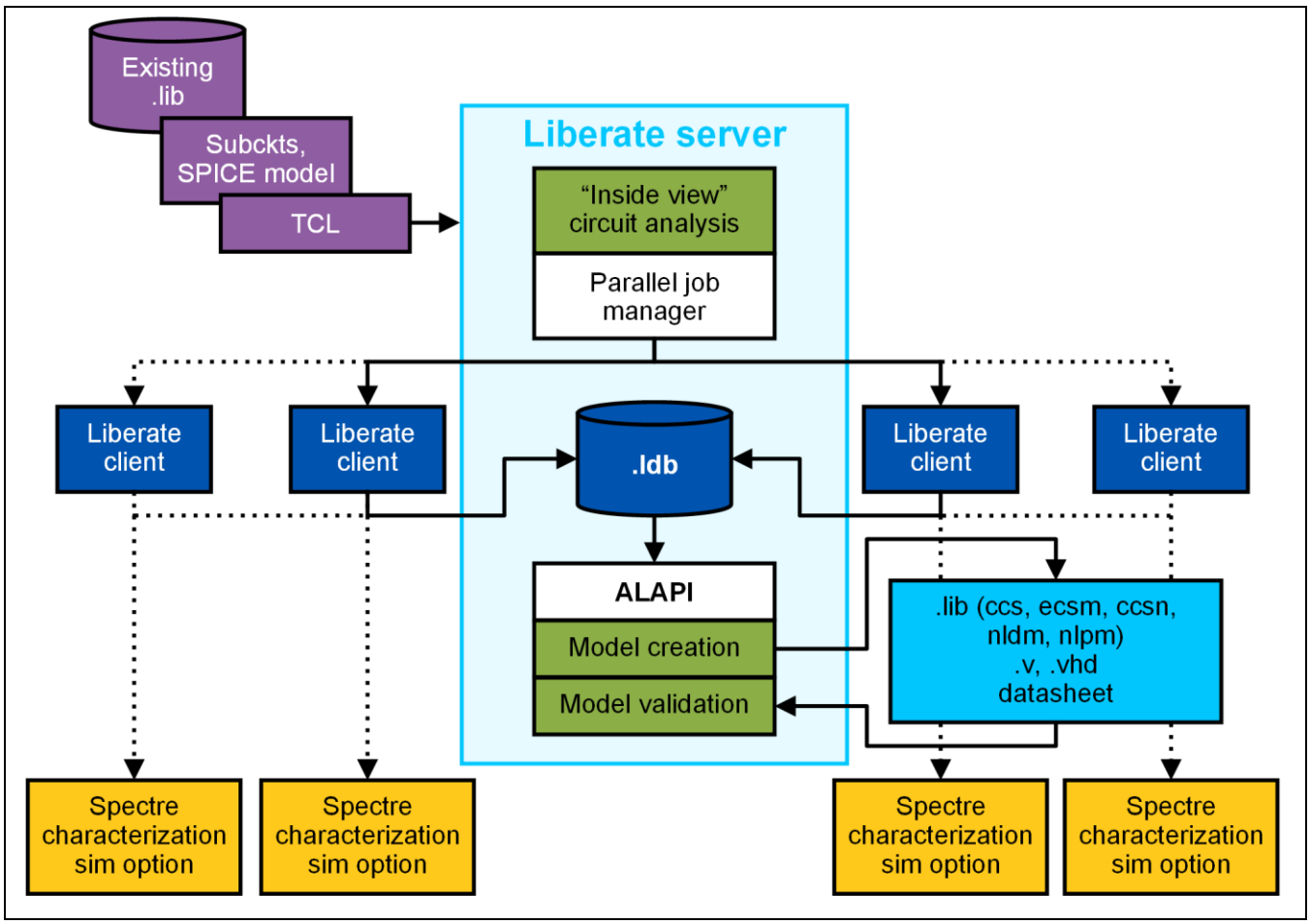
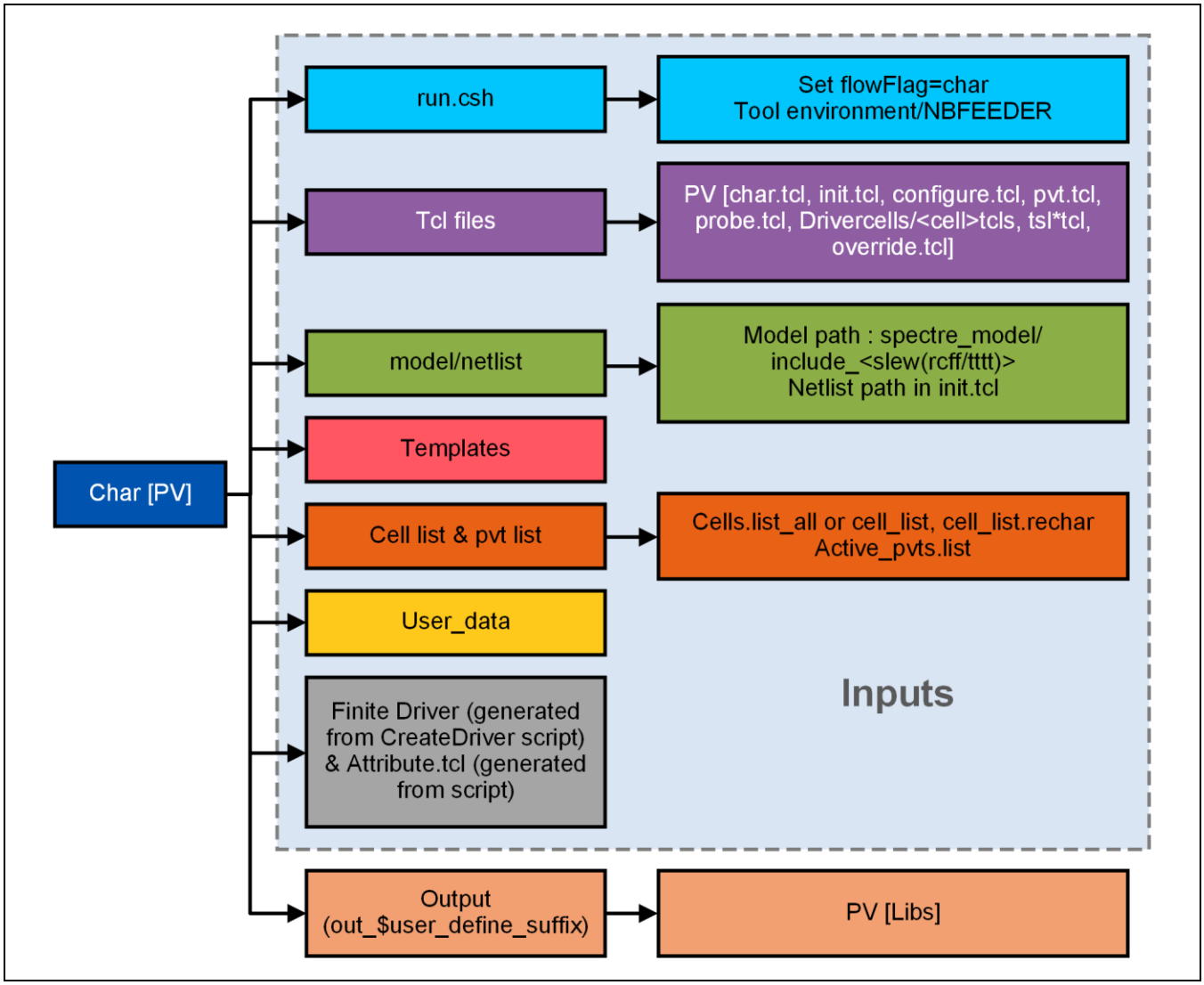


Figure 28: Characterization flow directory and file structure



To create Char setup:

Source CreateSetup.csh <Ref Area> <kits to run>

Example:

```
source CreateSetupcsh $INTEL_LCK kits
```

Update file CreateSetup.csh

- set vt=ulvt → set Vt type
- set tech=i0m → set technology

Critical input files:

- **run.csh:** Shell run script
- **pvt.tcl, active_pvts.list:** Define PVTs and the PVTs to characterize

- **init.tcl:** Common or major characterization settings
- **probe.tcl:** Define constraint probes
- **driver.tcl:** Define finite drivers
- **char.tcl:** Liberate characterization run script

run.csh

Figure 29: Section of run.csh file related to characterization

```
#Char
if ($flowFlag == char || $flowFlag == genTemp_char  ) then
    if ( -f cells.list_all || -f cell_list)   then
        ln -s cells.list_all cell_list
    else
       echo "Error: Create cells.list_all or cell_list to run char"
       exit 0
    endif
    touch cell_list.rechar
    liberate --trio  char.tcl ${runDir}_${tempFlag}  |& tee logs/log.${runDir}_${tempFlag}

    #To run compare library; Ref lib path need to be updated
    foreach libs (`ls out_${runDir}_${tempFlag}/PV/allcells/*ccslnt.lib`)
        set targetlib=`ls $libs`
        set libname=`echo "$targetlib" | awk -F "/" '{print$NF}'`
        set Reflib=`ls /dpc/intel_DA14122/IFS_18A/IT832798/STDCELLS/stdcells_1278.lib783_i0s_160h_50pp_pdk050_r3v0p0_ev2_ph2_CDNS_EDA_1_newlibs/*/lib/${libname}.gz`
        if ( -f $targetlib && -f $Reflib) then
            echo "Found Target lib: $targetlib \n Ref lib: $Reflib \n starting compare library"
            mkdir -p out_${runDir}_${tempFlag}/PV/allcells/QA
            liberate --trio --trio compare_para.tcl $Reflib $targetlib Ref_vs_Trio_$libname cell_list | & tee logs/cmp_Ref_vs_Trio_$libname.log
            mv cmp_Ref_vs_Trio_$libname.txt out_${runDir}_${tempFlag}/PV/allcells/QA/.
        endif
    end

    #QA flow
    foreach libs (`ls out_${runDir}_${tempFlag}/PV/allcells/*ccslnt.lib`)
        set targetlib=`ls $libs`
        set libname=`echo "$targetlib" | awk -F "/" '{print$NF}'`
        if ( -f $targetlib ) then
            ##Custom check setup+hold
            liberate --trio QA_setup_plus_hold_rec_plus_rem_check_and_update.tcl $targetlib 2e-12 200e-12 | & tee out_${runDir}_${tempFlag}/PV/allcells/QA/QA_setup_hold_check_$libname.log
            #Cusotm check log parsing
            liberate --python generate_run_summary.py -ldb_dir  out_${runDir}_${tempFlag}/PV/allcells/ldb.ldb.gz -report_dir out_${runDir}_${tempFlag}/PV/allcells/QA/QAlogs

            #Liberate_LV checks, currently under progress
            ./run_QA.csh $targetlib out_${runDir}_${tempFlag}/PV/allcells/QA
        endif
    end
endif
```

PVT-related files:

- **pvt.tcl:**
  - All PVTs are defined in this file
  - Vector/number of arcs of the default PVT are used across all PVTs
- **active_pvts.list:**
  - Specify PVTs (defined in file pvt.tcl) to be characterized

Liberate PVT-related commands:

- **define_pvt:** Define the PVTs
- **get_pvts:** Get a list of defined PVTs
- **set_pvt:** Set the active PVT

init.tcl: This file provides a central location for all the common characterization input settings that you might need to update, such as:

- The number of clients to use for distributed processing
- The specific methodology to enable/disable control
- Input paths

Figure 30: Example of file init.tcl

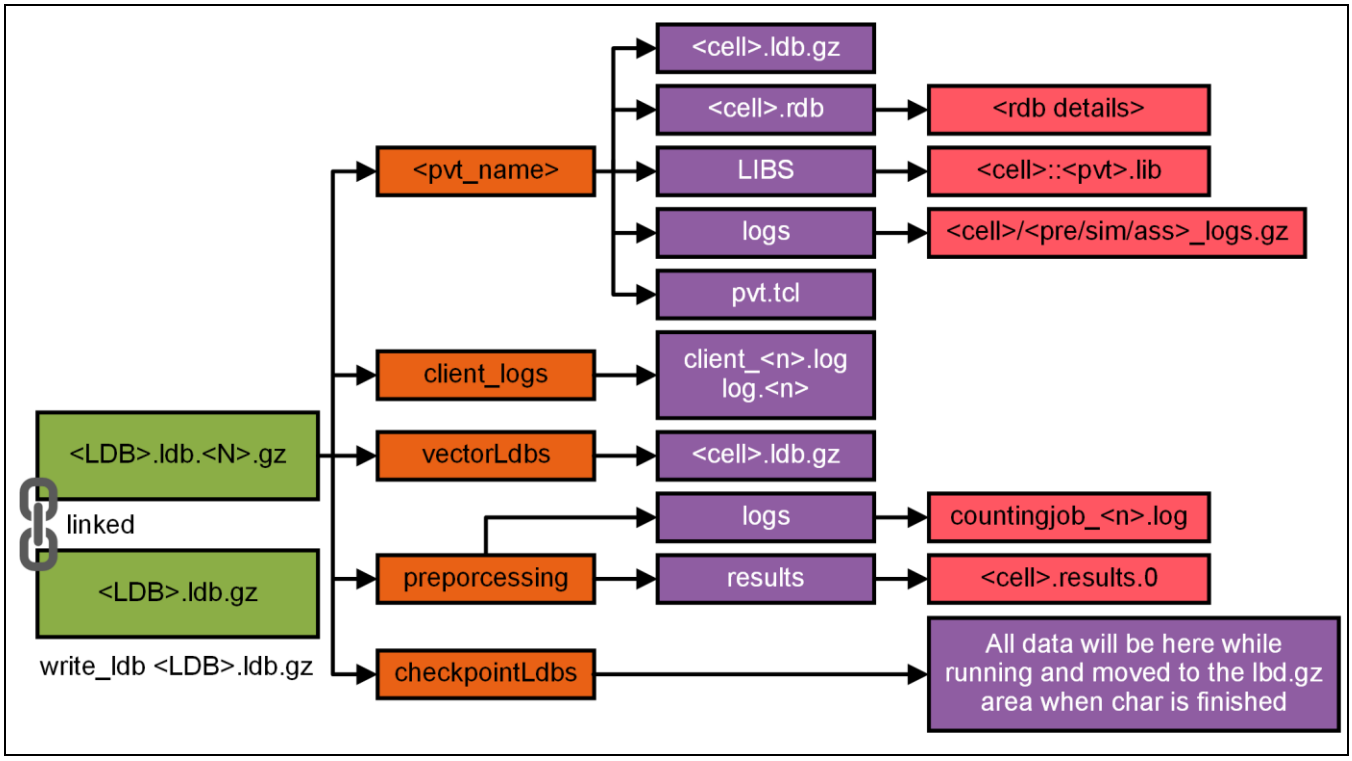
```
set client 200
set custom_probe 1 ; # if set to 1 pass probe.tcl path in probe_list variable.
set finite_driver 1

# Input files for PV/RV flow
set library  lib783_i0s_160h_50pp_${kit}_${VT}
set technology        lib783_i0s_160h_50pp_${kit}_${VT}
set netlist_area      ${technology}_${temp}c_${stat_skew}_c${default_grp_modeling}
set netlist_dir /dpc/intel_DA14122/IFS_18A/IT832798/STDCELLS/stdcells_1278.lib783_i0s_160h_50pp_pdk050_r3v0p0_ev2_ph2_CDNS_EDA_1_newlibs/${kit}_${VT}/spf
set model_file [pwd]/spectre_model/include_${skew}
set DRIVER_CELL_netlist_dir /dpc/intel_DA14122/IFS_18A/IT832798/rohitsh/from_Intel/i0s_finite_Driver/
set config_file [pwd]/configure.tcl
set commontcls $::env(LTRIO_SCRIPTS_DIR)
set probe_list [pwd]/probe.tcl
set tslnode tsl_node_all_${kit}_${VT}_${pvt_name}
set libname  ${library}_${op_cond}_${stat_skew}_c${default_grp_modeling}
set override $rootdir/override.tcl
set glbclk 0
set enable_3D_constraint 1
##
# Restrict  all specified messages define with message ID, kindly  review message thoroughly before adding to the list
set setMsgID {LIB-507 LIB-186 LIB-10032 LIB-511 LIB-907 LIB-214 LIB-6507}
set autoSPC ""

if {[regexp (DEBUG) $tag]} {
set DEBUG 1
} else {
set DEBUG 0
}
set recovery_flow 1
## BMC settings
if {[regexp (BMC) $tag]} {
set BMC_RUN 1
set BMCarcs $rootdir/BMC_arcs.tcl
set trials 30000
set monte_thread 10
} else {
set BMC_RUN 0
}
```

Constraint probes: probe.tcl. Define custom constrain measurement type and probe nodes.

Figure 31: Example of file probe.tcl

```
if {[regexp (fuz043m) $pktCell]} {
####Start Constraint settings for cell fuz043m
set_constraint_criteria -cells *fuz043m* -probe {o so nk5 nk6 nk4 nk3}
set_constraint_criteria -cells *fuz043m* -type {setup} -probe {nk5 nk6 nk4 nk3 o -metric "constraint_slew_degrade" -slew_degrade 0.5 so -metric "constraint_slew_degrade" -slew_degrade 0.5 }
set_constraint_criteria -cells *fuz043m* -type {recovery} -probe {nk5 nk6 nk4 nk3 o -metric "constraint_slew_degrade" -slew_degrade 0.5 so -metric "constraint_slew_degrade" -slew_degrade 0.5 }
set_constraint_criteria -cells *fuz043m* -type {hold} -probe {nk3 nk5 nk6 o so}

set_var -cells *fuz043m* -type {setup} constraint_signal_restore_check 1
set_var -cells *fuz043m* -type {setup} constraint_signal_restore_ref_nodes {nc2 nc1}
set_var -cells *fuz043m* -type {setup} constraint_signal_restore_probe_nodes {nk3 nk4 }
set_var -cells *fuz043m* -type {setup} constraint_signal_restore_ref_threshold 0.8
set_var -cells *fuz043m* -type {setup} constraint_signal_restore_probe_threshold 0.65
####End Constraint settings for cell fuz043m
}
```

Figure 32: Characterization output directory structure



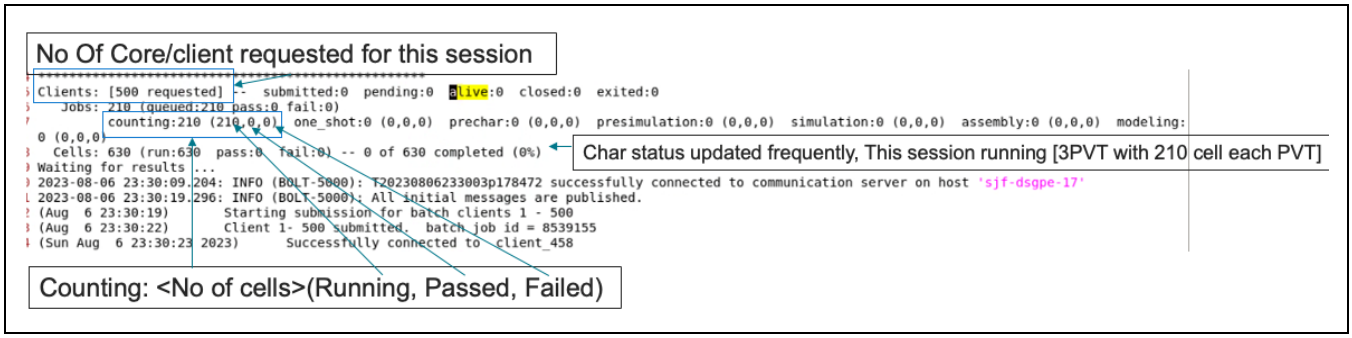
Characterization output directory structure:

- **LDB file:** Per cell, written under each PVT directory
- **Client log file:** Contains information about different counting, simulation and assembly jobs, command, log file, and so on
- LDB directory created will be <LDB>.ldb.<N>.gz
- Latest directory will always be linked to <LDB>.ldb.gz
  - This linking ensures that read_ldb <LDB>.ldb.gz always points to the latest LDB directory (no need to specify the value of <N>)
- In fresh char flow, N =1, for example:
  - LDB.ldb.1.gz
  - ln -s LDB.ldb.1.gz LDB.ldb.gz
  - Example: out_dir_probe_fix_index/PV/allcells/ldb.ldb.1.gz/
    - checkpointLdbs
    - client_logs
    - preprocessing
    - rcss_0p495v_125c
    - tttt_0p550v_85c
    - tttt_0p700v_85c
    - vectorLdbs

Characterization Master Log: logs/log.<rundir>_fix_index

Figure 33 shows a summary reported in the log file upon completion of all jobs.

Figure 33: Log file, summary



- **One shot (R, P, F):** Status of cell characterized on single client; disabled by default
- **Pre char (R, P, F):** Status of prechar step (pre-driver, max load generation)
- **Simulation (R, P, F):** Status of simulation packets
- **Assembly (R, P, F):** Status of cell-level modelling; collects simulation data and write cell level LDB and LIB files
- **Modeling (R, P, F):** Status of Liberty-level LIB files

3.8.5 Monte Carlo Flow

Use the Monte Carlo Flow (BMC) is to verify LVF data generated by Liberate. You can run Monte Carlo simulation for a specified cell/arc/PVT, and then compare with the Liberty file generated from characterization.

In the file run.csh, change flowFlag to char_BMC.

Create BMC_arc.tcl. This file is also generated by compare_library.

Update the file run.csh:

```
set flowFlag=char_BMC
```

Figure 34: Example of run.csh file

```
#Char Montecarlo
set flowFlag=char_BMC
if ($flowFlag == char_BMC  ) then
   if ( -f cells.list_all || -f cell_list)   then
      ln -s cells.list_all cell_list
   else
      echo "Error: Create cells.list_all or cell_list to run char"
      exit 0
   endif
   touch cell_list.rechar
   variety --trio  char.tcl ${runDir}_BMC_${tempFlag}  |& tee logs/log.${runDir}_BMC_${tempFlag}
   #To run compare library; Ref lib path need to be updated
   foreach libs (`ls out_${runDir}_${tempFlag}/PV/allcells/*ccslnt.lib`)
       set targetlib=`ls $libs`
       set libname=`echo "$targetlib" | awk -F "/" '{print$NF}' | sed 's/ccslnt/BMC/g'`
       set Reflib=`ls out_${runDir}_BMC_${tempFlag}/PV/allcells/$libname`
       if ( -f $targetlib && -f $Reflib) then
           echo "Found Target lib: $targetlib \n Ref lib: $Reflib \n starting compare library"
           mkdir -p out_${runDir}_${tempFlag}/PV/allcells/QA
           liberate --trio --trio_compare_para.tcl $Reflib $targetlib BMC_vs_Trio_$libname cell_list | & tee logs/cmp_BMC_vs_Trio_$libname.log
          # mv cmp_BMC_vs_Trio_$libname.txt out_${runDir}_${tempFlag}/PV/allcells/QA/.
       endif
   end
endif
```

Update the file init.tcl:

```
## BMC settings
if {[regexp (BMC) $tag]} {
set BMC_RUN 1 -> Enable BMC flow
set BMCarcs $rootdir/BMC_arcs.tcl -> Arcs to run BMC set trials 30000 -> Number of
samples
set monte thread 10 -> Number of core to run samples } else {
set BMC_RUN 0 -> disable BMC flow
}
```

- Here, the number of core will be: Number of client * monte_thread. Example: 200core * 10 = 2000 cores
- Monte_thread setting enabled Spectre job distribution and run samples in parallel

Example: 10 threads will distribute 30k samples, that is, 300 samples/thread.

Update char.tcl.

Figure 35: Example of char.tcl file

```
if { [regexp "fix_index" $tag] } {
    if {$BMC_RUN == 1} {
       ## Variety based BMC Characterization
       puts "Info (trio-flow): sourcing selected arc/slews in file $BMCarcs"
       source $BMCarcs
       char_variation -monte  -monte_trials $trials -user_arcs_only  -skip_variation {mpw} -skip {cin leakage power mpw
 }  -cells  $cells  -thread  $threads -extsim spectre
    } else {
       ## Trio based nominal Characterization
       char_library -lvf -skip_variation {mpw} -ccs  -ccsn  -cells  $cells  -thread  $threads -extsim spectre
    }
}
```

```
set pvts [get_pvts]
if { $BMC_RUN == 1} {
  foreach pvt $pvts {
       set_pvt $pvt
       puts "Info (trio-flow): Modeling BMC libs"
       write_variation -conf_interval -format sensitivity_plus_nom -overwrite  -filename $rundir/${libname}_BMC.lib ${l
ibname}
  }
}
```

3.8.6 Distributed Processing – setup Netbatch distribution

Update run.csh:

```
setenv DRM 1
setenv NBJOB 1
```

Figure 36: Example of run.csh file

```
if ($NBJOB == 1) then
    setenv CDS_BOLT_SERVER scc236001.zsc9.intel.com,scc236002.zsc9.intel.com,scc236003.zsc9.intel.com:scc236004.zsc9.intel.com,scc238001.zsc9.intel.com,scc238002.zsc9.intel.com:scc238
003.zsc9.intel.com,scc238004.zsc9.intel.com,scc240001.zsc9.intel.com:scc240002.zsc9.intel.com,scc240003.zsc9.intel.com,scc240004.zsc9.intel.com
    setenv CDS_LIC_FILE 5280@cadence55p.elic.intel.com
    setenv LM_LICENSE_FILE 5280@cadence55p.elic.intel.com
endif

 ##Modify variables below to setup SGE/LSF/NC. char.tcl also needs to be updated
 if ($DRM == 1) then
     setenv NBQSLOT /cds/lib/leaf/stdpri
     setenv NBPOOL  zsc9_normal
     setenv NBCLASS "SLES12&&8G"
     setenv LM_PROJECT AD-CORPLIB
 endif
```

Update char.tcl:

Figure 37: Example of char.tcl file

```
41 # LSF distribution settings
42 set_var packet_clients $client
43 set_var packet_arcs_per_thread 1
44 set     threads 1
45 puts "Info : (Trio) no of CPU given by user $client"
46 ##################Liberate Cluster#########################################################################################################################
47 if {$::env(DRM) == 1} {
48    set_var rsh_cmd "nbjob run --target $env(NB_FEEDER) --log-file '/dev/null' --qslot $env(NBQSLOT) --queue $env(NBPOOL)"
49 } else {
50
51     set_var rsh_cmd local
52     puts "Info : (Trio) Overriding no of CPU given by user $client to 5 "
53     set_var packet_clients 5
54
55 }
```

4 Layout Abstracts and LEF Files

4.1 Layout abstracts

The library modeling tool AG (Abstract Generator) creates layout abstracts for standard cells, macro blocks, and IOs from detailed layout information in OA, GDSII, or OASIS formats. It is included with the Cadence Virtuoso* Studio installation.

An abstract is a high-level representation of a layout view. An abstract contains information about the type and size of cells, the position of pins or terminals, and the overall size of blockages. The abstracts generated are based on the physical layout and logical data, process technology information, and specific cell-modeling requirements. After P&R (Place and Route) is complete, the abstracts are replaced with layouts with all the layers required for manufacturing.

Abstracts are used in place of full layouts by digital P&R tools such as Cadence Innovus* to improve performance.

Layout views contain the full detailed layout of a cell, however, abstracts only contain information about:

- Type and size of a cell
- Location of pins
- Obstructions
- Antenna data for signal pins (if available)

The functionality for abstract generation is available through two interfaces, which access the same executable. The standalone AG interface can generate abstracts for a single library at a time. The standalone AG interface is used in this demonstration. It can be run in either GUI or non-GUI mode. This interface provides advanced options and is designed for experienced users and library developers.

The integrated AG interface launched from the Cadence Virtuoso* design environment can generate abstracts for individual or multiple library cells. This interface is simple and is designed for novice users who are unfamiliar with the standalone application or who find the advanced options of the standalone application unnecessary. AG retrieves the technology information from the technology library to which the design library is attached. In this demonstration, the technology library is intel78tech.

Features of AG include the ability to:

- Create pins from shapes under the label text. Text and pin shapes can also be at levels of hierarchy lower than the current level of the cell.
- Check the consistency of cell name, pin name, and pin direction between LEF and Verilog or LIB files.
- Control extraction by layer, hierarchy, and distance. Gate and diffusion areas can also be extracted at the same time.
- Create Detailed, Cover, or Shrink-wrapped blockages for each layer.
- Cut out pin access from Cover blockages.
- Calculate process antenna effects, including partial area metal for boundary pins.
- Automate the flow by recording commands from a graphical AG session to a replay file which can be rerun later either graphically or non-graphically.

Abstract generation using AG is a three-step process consisting of:

- **Pins** step:
  - AG maps text labels to terminal names and creates physical pin shapes corresponding to the geometry overlapping the origin of the text labels.
  - Identifies pin geometries for connectivity extraction.
  - Creates the P&R boundary.
  - Matches the pins created with the logical view present and appends the appropriate pin direction.
  - Specifies hierarchy depths for text label and metal geometry search for text-to-pin mapping and pin creation.
  - Has options to remove unnecessary text from pin labels or to replace text in a pin label.
  - Has an option to preserve layout labels in the final Abstract view.
- **Extract** step:
  - In the Extract step, AG traces the connectivity hierarchically through the physical shapes in the layout to modify pin shapes, starting from the pins generated during the Pins step.
  - Constructs the correct database model for strong, weak, and “must connect” pins.
  - AG can also use the antenna options to create library process antenna information.
- **Abstract** step:
  - In the Abstract step, AG adjusts the pin shapes created during the Extract step to create the final shapes required by P&R tools.
  - Then, AG applies a layer blockage model that you select to create the final blockage geometry in the abstract.

4.2 Library Exchange Format

LEF (Library Exchange Format) defines the elements of IC process technology and the associated library of cell models. DEF (Design Exchange Format) defines the elements of an IC design relevant to the physical layout, including the netlist and design constraints. The LEF and DEF files allow the exchange of placement and routing information with various databases using the LEF/DEF translators. For more information on LEF and DEF, see the LEF/DEF Language Reference.

The placement and routing tools on the OpenAccess database read and write LEF and DEF files using the core OpenAccess LEF/DEF translator. The CIC customized OpenAccess LEF/DEF translator is written as a Cadence Virtuoso* Studio design environment-specific application for these translators.

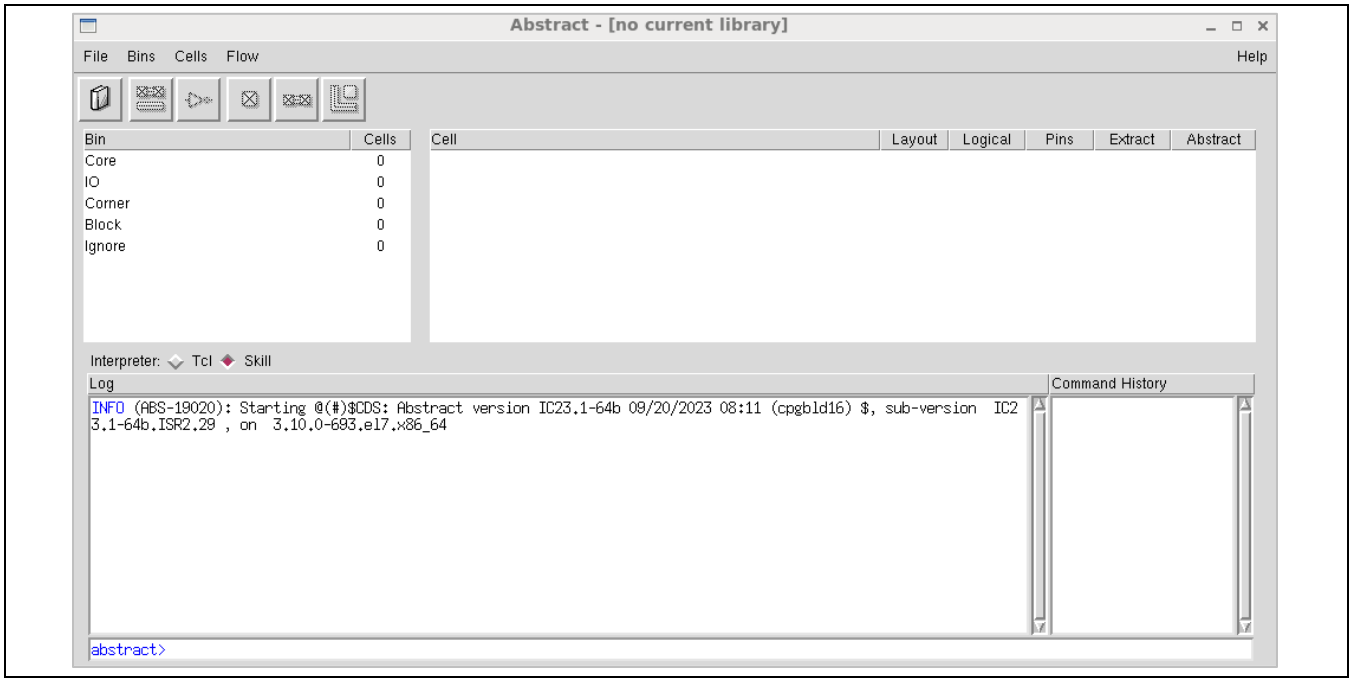
Once the layout abstracts have been created with AG, the LEF translator creates the standard cell LEF files.

4.3 User actions

Change the directory to the LDK work area ./ldk_r0.9_cdns/training/setup/cadence/ldk, and then source the sourceme file, if this is the first time launching Cadence tools, from the shell you are working in.

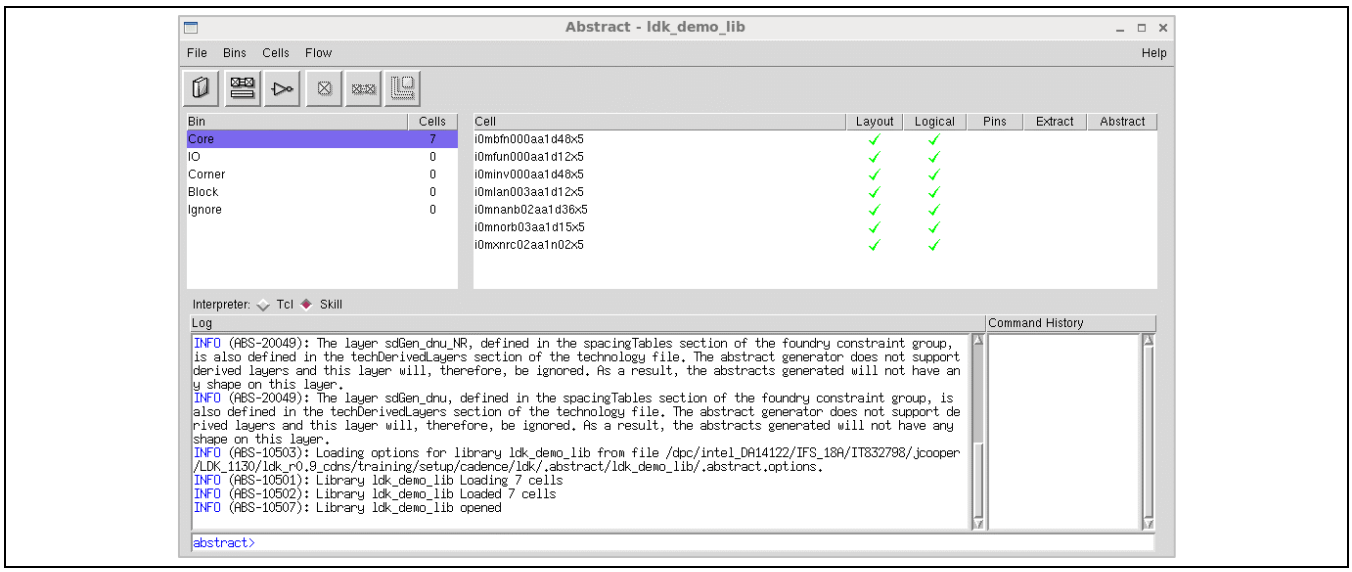
1. Type the command `abstract &` to launch the AG (Figure 38).

Figure 38: AG User Interface (UI)



2. From the AG menus, select **File > Library > Open.** Then select the library **ldk_demo_lib**. Click **OK**. (Figure 39 shows that library as having been selected.)

Figure 39: AG UI after library open

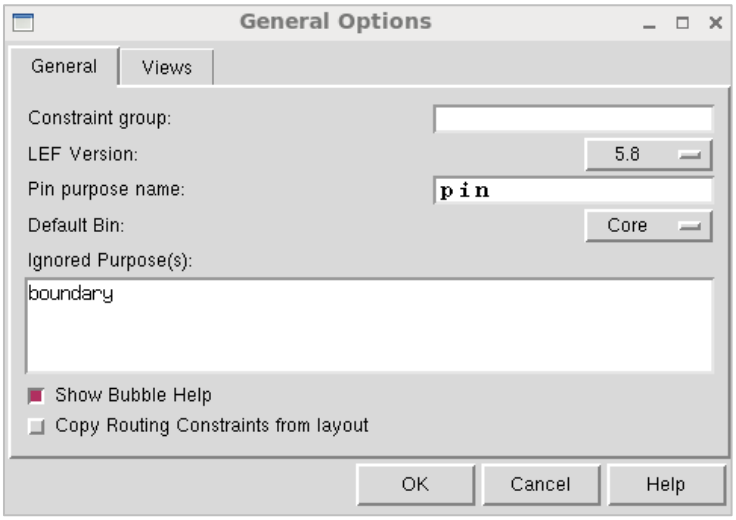


All the cells should be in the *Core* bin. If not, select the cells in the other bin(s), and then use **Cells > Move...** to move them into the *Core* bin.

Bins are like buckets for placing cells. Each cell in a library is always contained in one bin. Each bin is associated with specific options, which allows the cells in a library to be distributed into mutually exclusive sets for processing. For example, AG uses the option settings for a bin to determine how to treat the cells in the bin. There are five predefined bins, one for each of the five main types of cells: *Core*, *IO*, *Corner*, *Block* and *Ignore* (to store any cells you do not want to process at a particular time). Cells placed in the *Ignore* bin are excluded from the set of cells to be processed but not deleted.

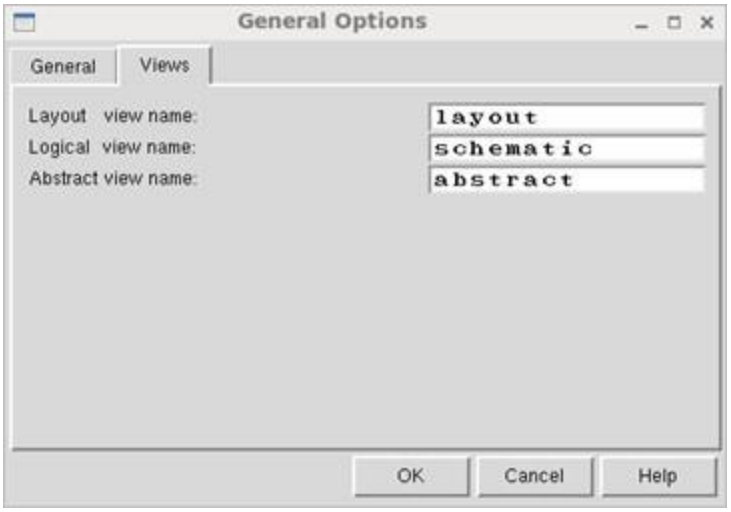
3. From the AG menus, select **File > General Options...**. On the *General* tab, *Pin purpose name* should be pre-set to **pin**. If not, update it (Figure 40).

Figure 40: General Options, General tab



4. On the *Views* tab, *Logical view name* should be pre-set to **schematic**. If not, update it, and then click **OK** (Figure 41).

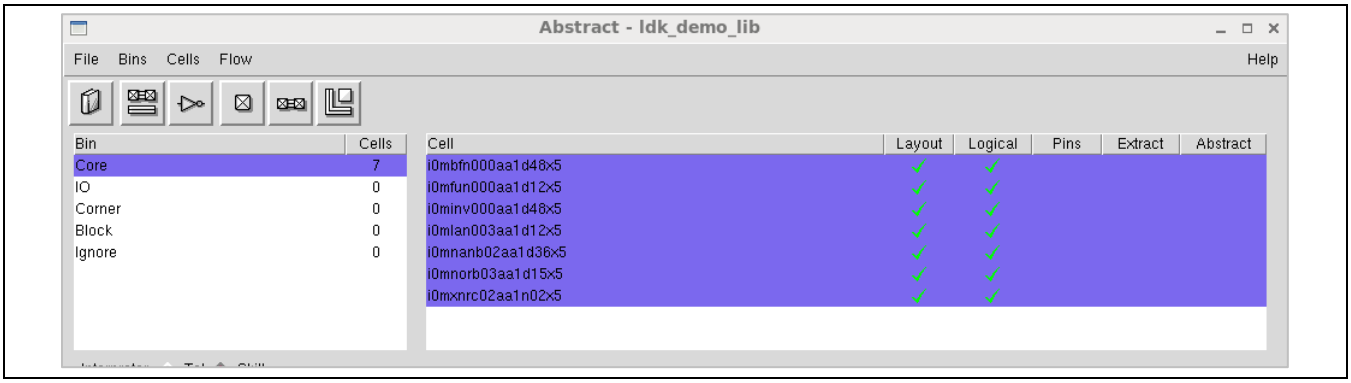
Figure 41: General Options, Views tab



5. Select the *Core* bin, highlighting it. From the AG menus, select **Cells > Select All** (Figure 42)**.**

   All cell libraries are processed in the next steps.

Figure 42: AG UI, Core bin



4.4 Options File

AG starts with a set of default options for each bin. As options are changed, they are saved in the options file .abstract/<library name>/.abstract.options. When AG is opened, and a library is loaded, the options file is read, restoring the options from the previous run. An options file is provided in the LDK with the recommended settings for this technology. The file is portable, and you can copy the entire .abstract directory into your working area (from which Cadence Virtuoso* is launched) before launching AG when creating your own standard cells.

4.5 Pins Step

4.5.1 Overview

The Pins step describes the first step in the abstract generation process. Usually, no pins are explicitly defined in the layout view, so they must be generated before you can continue the abstract generation process. AG derives pins from the text labels in the layout view and places the locations at the text origins.

- If there is a Logical view present, AG only creates pins that match logical view terminals.
- AG copies the placement status of the pins in the layout view to the abstract view. The supported placement states are locked, placed, fixed, none, and unplaced.
- AG automatically detects shapes placed in WSP (Width Spacing Pattern) regions and ignores these shapes while creating pins under labels and PR boundaries.

AG provides several options that support a variety of layout and text mapping conventions. To check options for mapping and controlling the generation of pins, see Specifying Pin Mappings for Abstract Generation in the *Cadence Virtuoso* AG User Guide*.

AG performs the following functions during the Pins step:

- Maps text labels to terminal names and creates terminals
- Creates physical pin shapes corresponding to the geometry overlapping the text labels
- Creates the place-and-route boundary

Use the *Running step Pins* form to view and modify the bin options relevant to the Pins step.

An overview of the Pins steps is as follows:

- Select one or more cells in the main window, and then click the **Pins** button. Alternatively, select the `Flow > Pins…` command.
- This opens the *Running step Pins* form. The form contains four tabs: *Map*, *Text*, *Boundary*, and *Blocks*. You can use the radio buttons on the left to navigate to the tab pages for the different steps and bins you are using in the form.
- When satisfied with the options settings, click **Run** to run the step.

4.5.2 Demonstration

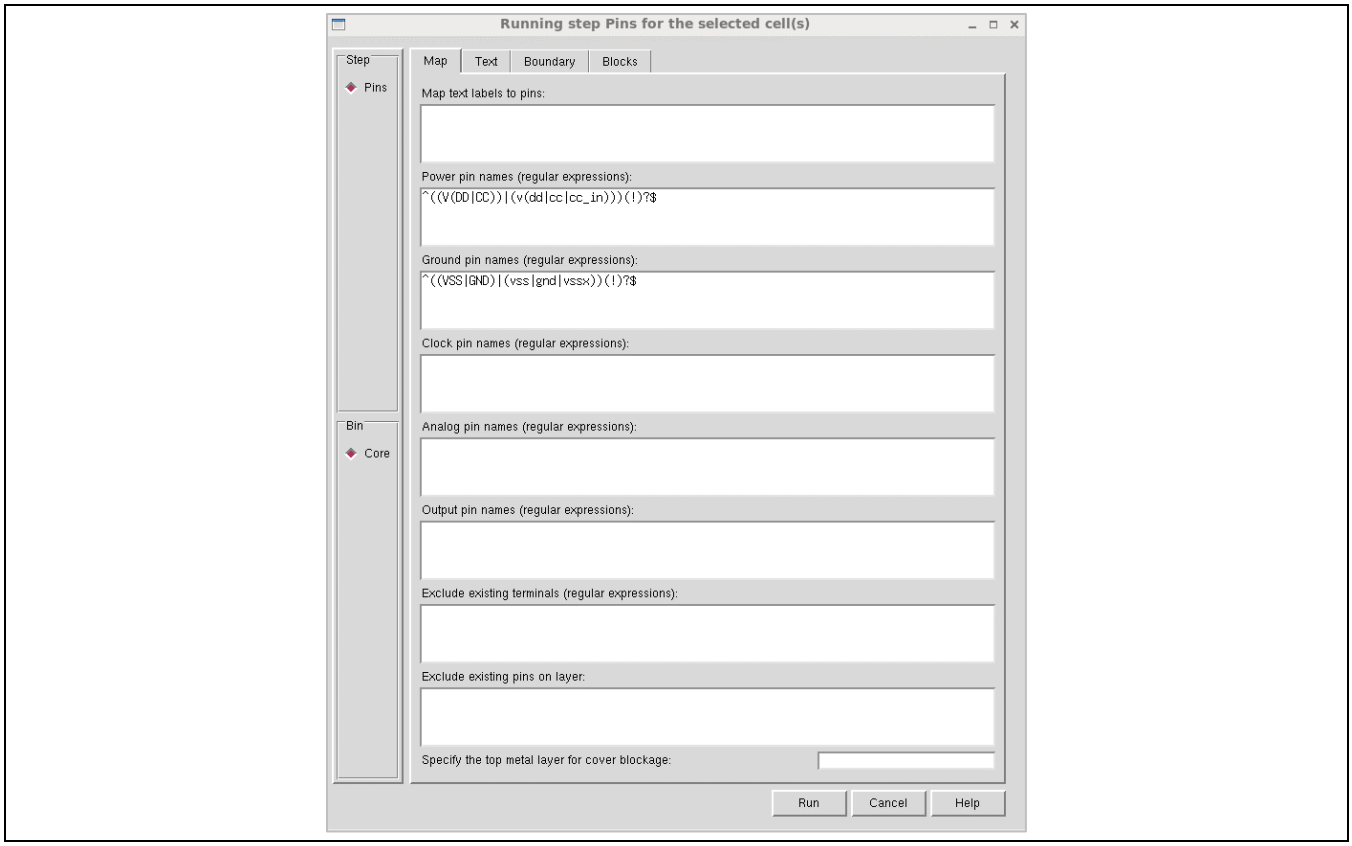
1. From the AG menus, select **Flow > Pins...** Review the settings on the *Map* tab (Figure 43).

   A regular expression must be entered that identifies the Power and Ground pin names. Clock, Analog, and Output pin names are optional; the Liberty model defines these three pin names and those definitions take precedence over the Abstract/LEF.

   The modification of the Power/Ground pin names is as follows:

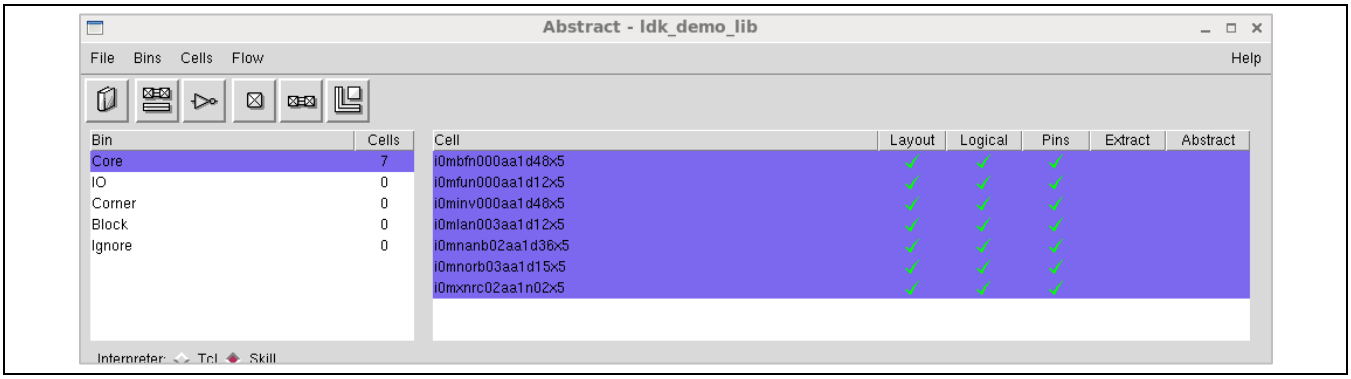
   - Power pin names: ^((V(DD|CC))|(v(dd|cc|cc_in)))(!)?$
   - Ground pin names: ^((VSS|GND)|(vss|vssx|gnd))(!)?$

Figure 43: Running step Pins, Map tab



2. Review the settings on the other tabs: *Text*, *Boundary*, and *Blocks*. The following options are important to consider:
   - On the *Boundary* tab, *Create boundary* is set to **off**. This is the recommended setting for standard cells since they should contain a boundary (prBoundary object) that defines the height and width, instead of relying on the layer shapes to define it.
   - On the *Blocks* tab, *Create power pins from routing* should be disabled since it is not applicable to standard cells.
3. Click **Run** on the *Running step Pins* form (Figure 44).

Figure 44: AG UI, results from Pins step



A yellow exclamation point (!) could appear instead of a green checkmark to indicate warnings were issued for the cell. It is recommended that the warnings be reviewed.

For each standard cell in the library, a view named *abstract.pin* is created.

4.6 Extract Step

4.6.1 Overview

The Extract step is the second step in the abstract generation process, and traces the connectivity between shapes and terminals, starting at the pin purpose shapes created in the Pins step. The shapes created during extraction are created on the purpose net at the top level of the extract view.

AG performs the following functions during the Extract step:

- Extracts each terminal net, one shape at a time.
- Constructs the correct database model for strong, weak, and must-join pins.
- Creates library process antenna information for custom blocks and standard cells.

To run the Extract step:

1. Select one or more cells in the main window, and then click the **Extract** button. Alternatively, select **Flow > Extract**. The `Flow > Extract…` command runs as far as the Extract step for the selected cells.
2. When you select the command, the *Running Form* is displayed. Use this form to view and modify the bin options relevant to the Extract step and any prior steps that must be run.
3. When satisfied with the options settings , click **Run** to run the step.

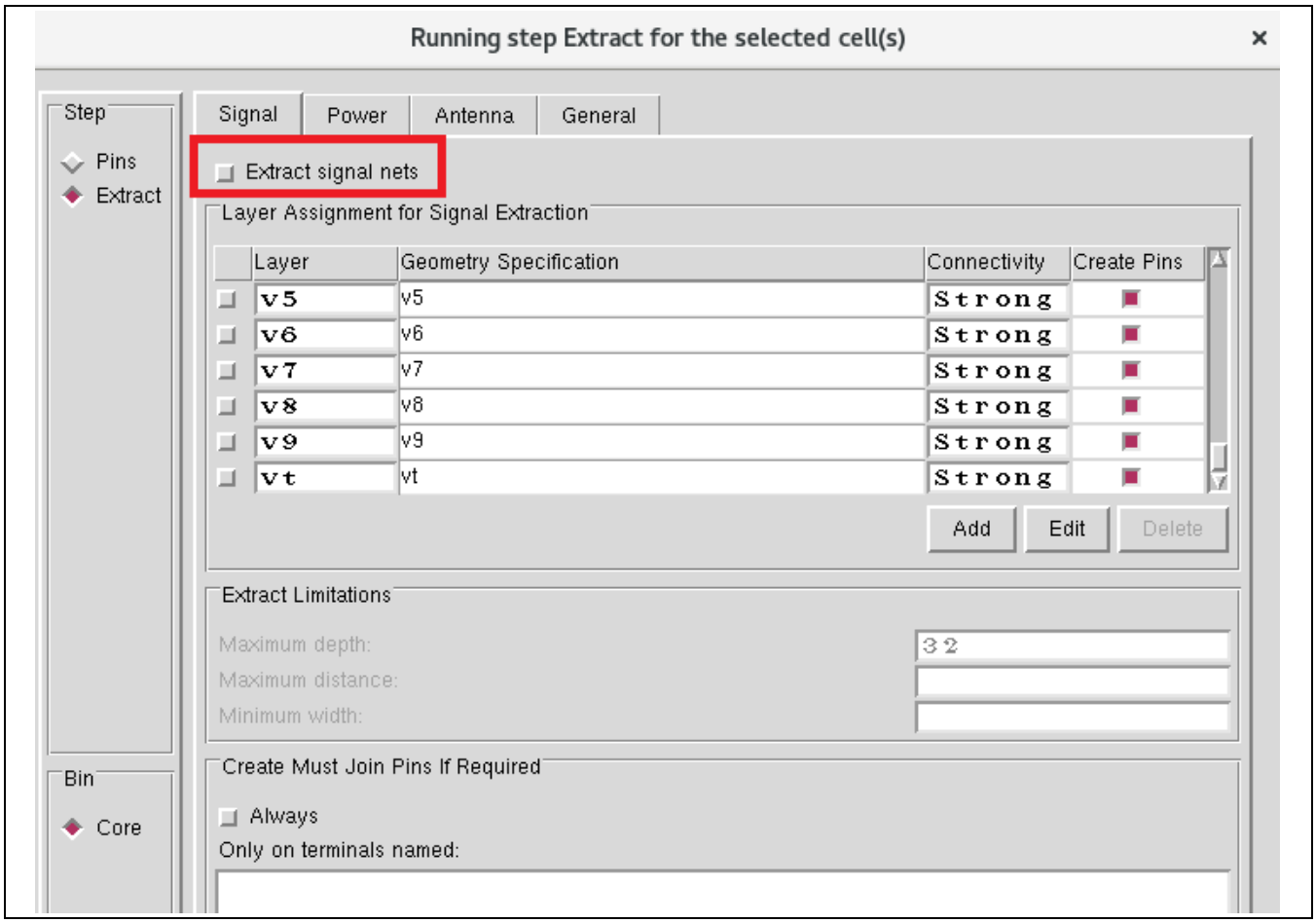
4.6.2 Demonstration

1. From the AG menus, select **Flow > Extract...**
2. On the *Signal* tab, note that *Extract signal nets* is deselected (Figure 45).

   When the *Extract signal nets* option is selected, full extraction is performed; the geometry found during the extraction is turned into pin geometry.

   To use the pins created in the layout and duplicate them in the Abstract view, deselect *Extract signal nets* so that AG does not extract them.

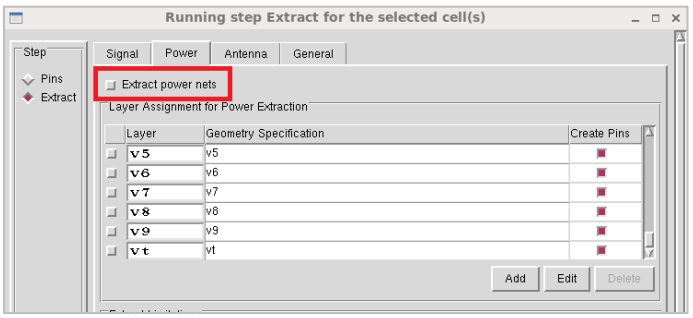
Figure 45: Running step Extract, Signal tab, Extract signal nets option



3. On the *Power* tab, *Extract power nets* is deselected (Figure 46).

   For the power and ground rails for standard cells, the recommended approach is to create pins in the layout as duplicated in the Abstract view, instead of having AG extract them.

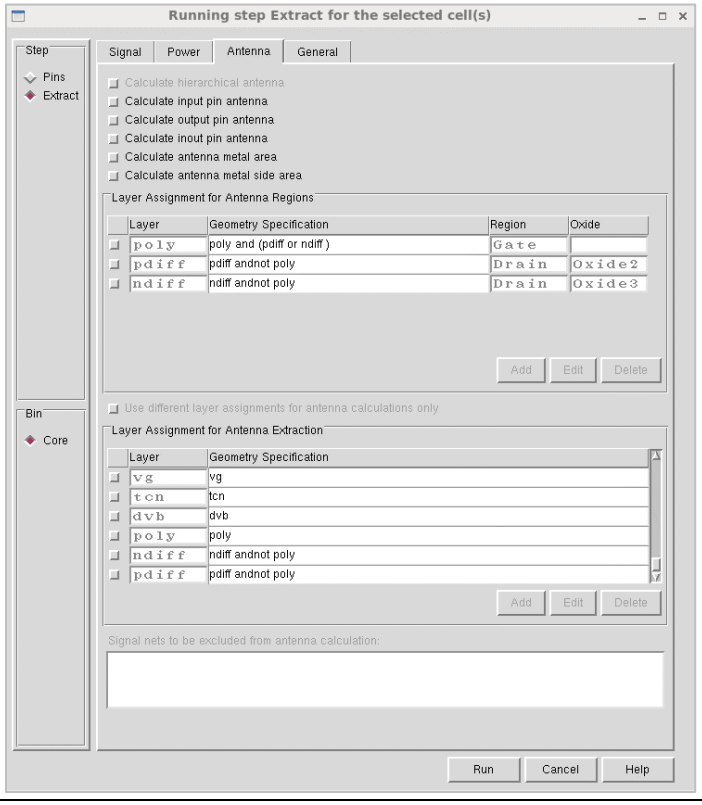
Figure 46: Running step Extract, Power tab, Extract power nets option



4. On the *Antenna* tab, no options are enabled; this capability is under development for this technology (Figure 47).

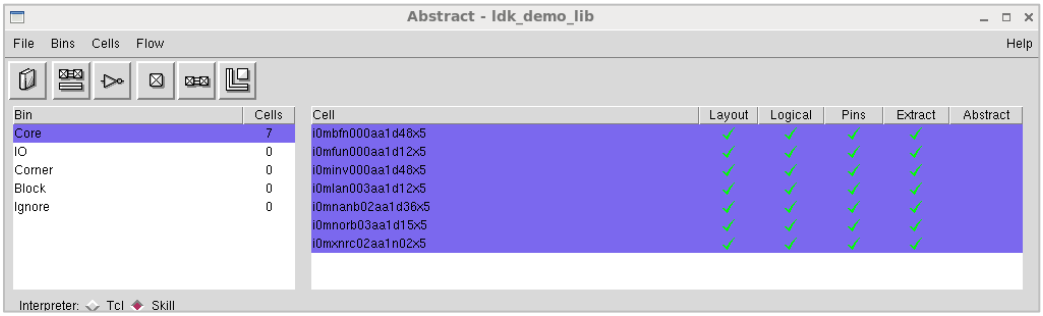
   "Antenna effect" is a common name for charge accumulation in isolated nodes of an integrated circuit during processing. It may damage transistors and degrade performance if discharged through the gate oxide. Adding antenna information into the layout abstract and LEF enables the P&R tools to calculate and mitigate the effect by jumping metal layers or adding an antenna diode.

Figure 47: Running step Extract, Antenna tab



5. Click **Run** on the *Running step Extract* form (Figure 48).

Figure 48: AG UI, results from Extract step



For each standard cell in the library, a view named *abstract.ext* is created.

4.7 Abstract Step

4.7.1 Overview

The Abstract step describes the third stage of the abstract generation process. AG adjusts pin shapes, performs blockage modeling, creates sites, and calculates overlap layers.

To run the Abstract step:

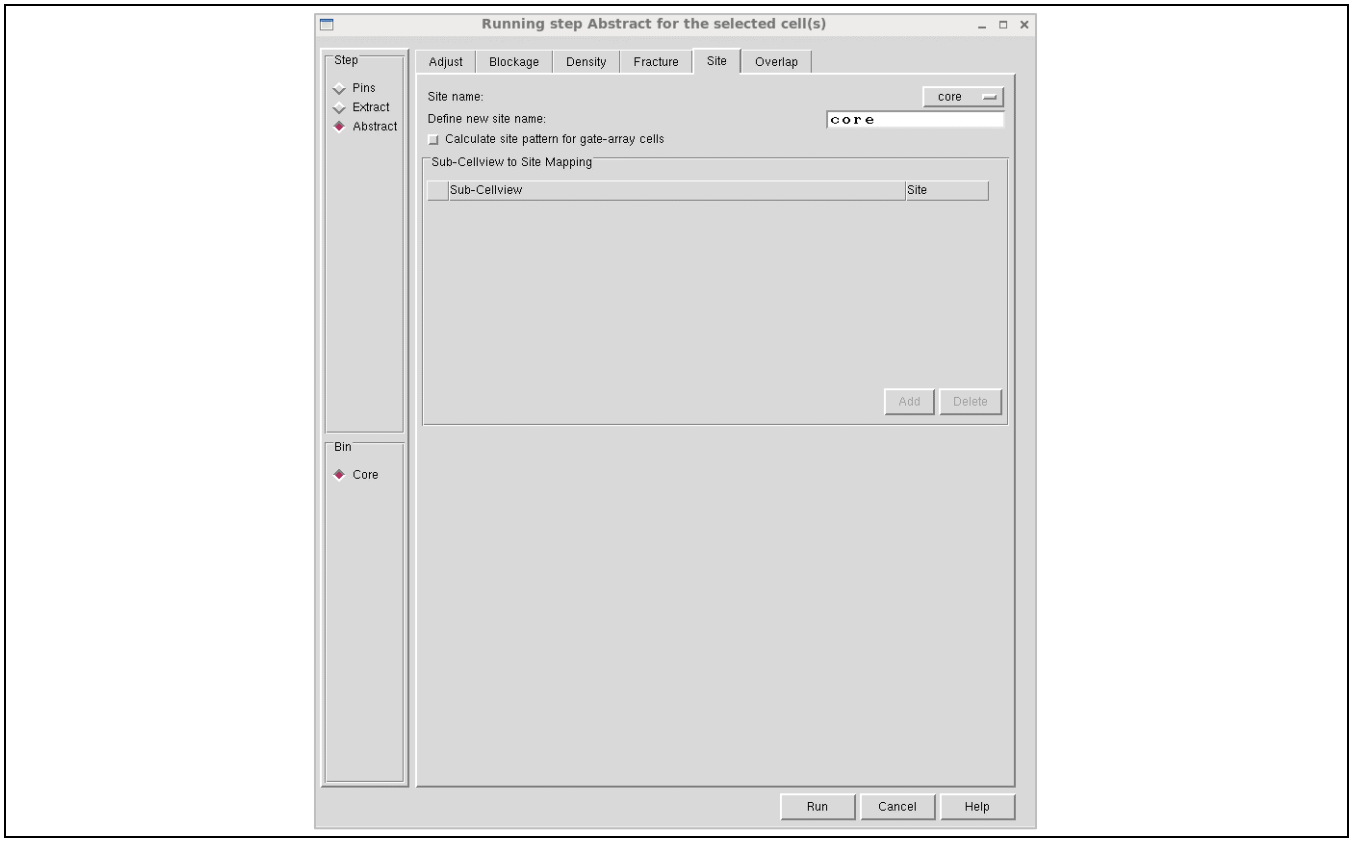
- Select one or more cells in the main window, and then click **Abstract**. Alternatively, select **Flow > Abstract...**.
  - This opens the *Running step Abstract* form.
  - There are six tabs on the form: *Adjust*, *Blockage*, *Density*, *Fracture*, *Site*, and *Overlap*.
- Select the options on this form, and then click **Run**.

4.7.2 Demonstration

1. From the AG menus, select **Flow > Abstract...**
2. On the *Site* tab, *Define new site name* is pre-set to **core** (Figure 49). This corresponds to the site name used in the PDK for single height cells.

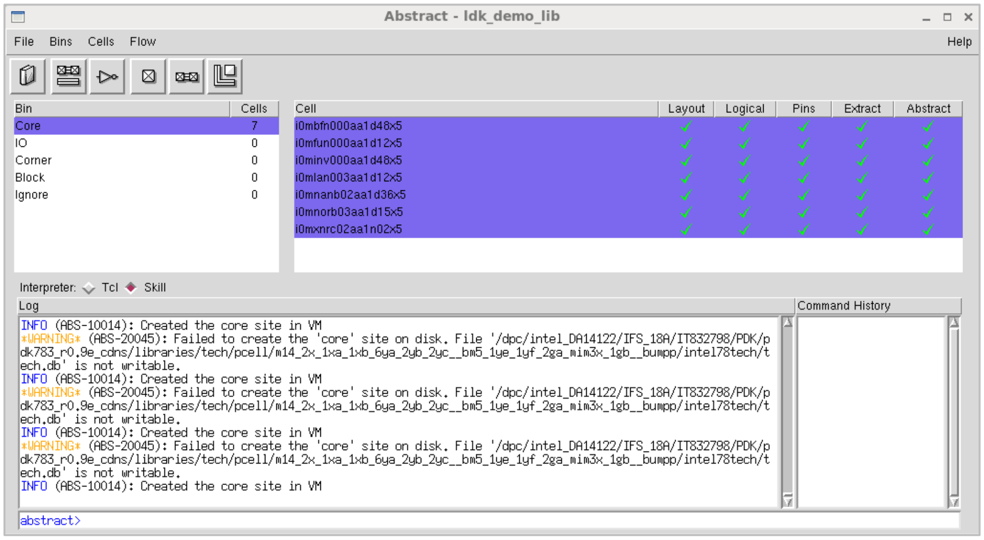
   Different site names for different cells cannot be accomplished in a single step in AG or through a post hook. Site specification in AG for cells is based on the bin type assigned to the cells. Libraries with single height (site: core) and double height (site: core2h) cells must be processed separately as groups. Select all the single height cells and set the site name to *core*, and then generate the abstracts. Do the same for the double height cells.

Figure 49: Running step Abstract, Site tab



3. The default settings can be used for all the other options; click **Run**. Results are generated (Figure 50).

Figure 50: AG UI, results from Abstract step



The warnings indicate that the site "core" is not declared in the PDK technology file and is created temporarily; you can ignore the warnings.

4. Exit AG.

4.8 Customization of the Abstract Generation Flow

Tasks that you otherwise cannot perform directly by using the available AG options can be done using an advanced methodology based on the concept of calling custom code—also called a hook—during the abstract generation flow. This section briefly describes the predefined hooks and how to customize them.

AG provides predefined hooks, which are automatically called before and after running a flow step (Pin, Extract, or Abstract) (Table 15). There are therefore, two predefined hooks for every flow step.

Table 15: Predefined hooks in AG

Pre-defined hooks	When called or run
PinsPreHook	Called prior to running the Pins step.
PinsPostHook	Called after running the Pins step.
ExtractPreHook	Called prior to running the Extract step.
ExtractPostHook	Called after running the Extract step.
AbstractPreHook	Called prior to running the Abstract step.
AbstractPostHook:	Called after running the Abstract step.
ExitHook	Run even if AbstractPostHook is not specified, when the tool is closed.

AbstractPostHook is used in this demonstration and accomplishes three things:

- Removes any unnecessary cell view properties from the abstract
- Adds a property so that the FIXEDMASK attribute is added in the LEF
- Converts via blockage to a pin for the via directly under a signal pin

1. On the command line, type:

```
more .abstractrc
```

2. Optionally, view the file ./.abstract/AbstractPostHook.il.

The abstract view is stored in the OpenAccess database and is accessible through the Cadence Virtuoso* Library Manager. The abstract views are exported to LEF in Section 4.9.

An optional clean-up step is to delete the intermediate views *abstract.pin* and *abstract.ext* from the Library Manager: select **Edit > Delete By View**.

4.9 Exporting LEF

4.9.1 Overview

(LEF) Library Exchange Format defines the elements of IC process technology and the associated library of cell models. Once the layout abstracts have been created with AG, the LEF translator creates the standard cell LEF files (macro LEF).

If the digital P&R tool, Cadence Innovus*, uses the Mixed-Signal OpenAccess (MOSA) mode, the technology information is read from the PDK technology library, and abstract views are used for the standard cells.

If the Cadence Innovus* uses the LEF/DEF mode, the technology information is read from the technology LEF and the macro LEF is used for the standard cells.

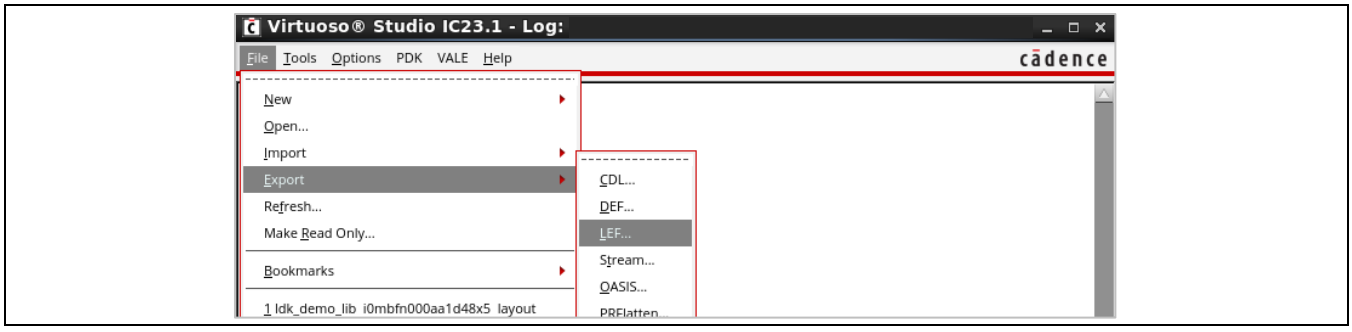
4.9.2 Demonstration

1. Start Cadence Virtuoso*:

```
virtuoso &
```

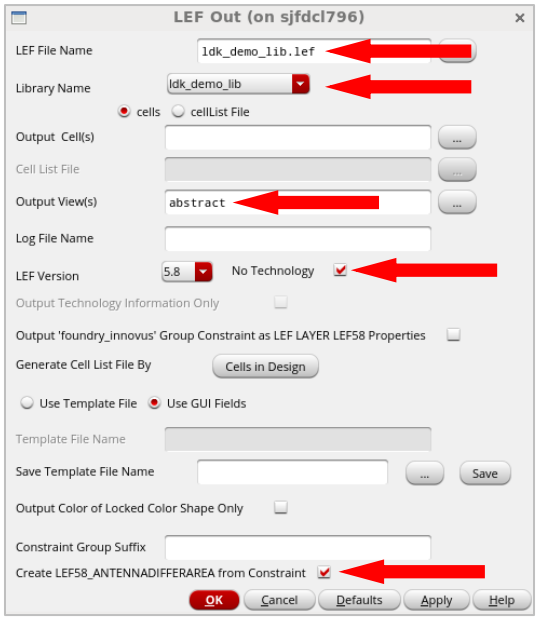
2. From the CIW, select **File > Export > LEF...** (Figure 51).

Figure 51: CIW menu to export LEF



3. Complete the *LEF Out* form as shown in Figure 52, and then click **OK**. Click **OK** again for the pop-up that follows; the warnings can be ignored.

Figure 52: LEF out form



This creates the LEF file ldk_demo_lib.lef.

4. Open the file ldk_demo_lib.lef in a text editor to review its contents.
5. Move the newly created LEF file into the library collateral collection area. The following sections reference this area:

```
mv ldk_demo_lib.lef  ./stdcells_1278.ldk_demo_lib/lef
```

5 Power Grid Library

Cadence Voltus* IC Power Integrity Solution requires power grid libraries for dynamic power calculation and static and dynamic IR drop analysis. The Power Grid libraries are generated for standard cells, memories, IOs, and custom macros in the design.

For standard cells, the IR/EM view is identical to the Early view due to the simplicity of the standard cells. This section covers topics relevant to standard cell libraries only.

Library Generation takes LEF, GDSII, or a combination of LEF and GDSII as input, and creates a binary power grid view of the cell contents within the cell library database. Power grid views contain the physical and electrical representation of all the cells and blocks in the design. They contain the necessary information required for power and rail analysis:

- For rail analysis, it contains RC parasitics, tap currents, and geometries
- For power analysis, it contains netlist and output capacitance loading

The inputs to Power Grid Library Generation are:

- Technology LEF
- Cell LEF with power pins
- Cell GDS (for macro blocks)
- Extraction technology file for Cadence Quantus* (qrcTechFile)
- Layer map file for GDS to qrcTechFile technology file
- Layer map file for LEF to qrcTechFile technology file
- SPICE device models (Spectre or HSPICE)
- Subcircuit SPICE-format netlist, preferably with device X,Y coordinates

This is required so that tap current is snapped to the nearest device and gives the most accurate load for the device. Without xy coordinates, it results in an even distribution of the tap current across the entire device.

Library Generation outputs the following files:

- A cell library database that contains the following:
  - LEF technology information, which includes the following:
    - Layer information for each layer: its name, relationship to other layers, routing width, and direction
    - Via information for each via: its name, the layers that it connects, and a set of geometries accessible as polygons
  - Cell information, which includes the following:
    - Geometric views of the cell
    - Port information
    - Power grid views
    - Process technology data
  - A bounding box for each cell
- Text report and summary files having detailed information about the power grid library

5.1 Types of power grid views

Different PGVs (Power Grid Views) are used during rail analysis based on the accuracy mode selected. The PGV types correspond to the accuracy mode of rail analysis. The following are the power grid view types:

- **Early** view: Contains the current distribution factor and the SPICE-simulated decoupling capacitance information at the power ports of the cell. It does not include the parasitic associated with the interconnect, which prevents IR drop within the block or cell from being analyzed. Early power grid views are the default PGV views for the rail analysis XD accuracy mode, which is recommended for the early design phase.
- **IR** view: Performs reduction on the extracted power grid and merges small value current taps together without significant impact on accuracy. Current designs in many cases contain some embedded memories for boosting performance. The bit cells of the memory contribute to the creation of many current taps; however, they do not contribute significantly to the IR, but affect the tool performance. This IR view suppresses those current taps that have a very small current value, thus improving both memory usage and tool performance. The IR view is the default for rail analysis HD mode.
- **EM** view: Contains the extracted power grid from the GDSII data of a cell. It creates current taps on the bottom conducting layer or at the locations of the via layers above it. GDSII and XY SPICE netlist are required to generate accurate current distribution and decoupling capacitance, but you can manually force current values and capacitance to be assigned to these taps. You can also set up a trigger file to manually define the dynamic current behavior or run SPICE simulators to obtain the current waveform.

5.2 Required collateral

Table 16 lists the files required to generate PGV views:

Table 16: Required collateral

File	How provided
PDK Technology LEF file	Provided by the foundry.
Spice Model files	Provided by the foundry.
Cadence Quantus* Technology file	Provided by the foundry.
Cadence Quantus* Layermap file	Provided by the foundry.
STD Cell Transistor Level xDSPF files	Must be generated beforehand (See Section 2, Netlists and Physical Views)
STD Cell LEF file	Must be generated beforehand (See Section 4, Layout Abstracts and LEF Files)

Along with this, additional information about power and ground pins, as well as a cells list file containing the list of cells to process is also needed.

Note: The powergate cells require special handling for PGV generation and are outside the scope of this demonstration. For help with powergate cells, contact the Cadence support team.

5.3 User actions

Change the directory to the LDK working area ./ldk_r0.9_cdns/training/setup/cadence/ldk, and then source the sourceme file if this is the first time launching Cadence tools from the shell you are working in.

1. Change to the pgv directory:

```
cd ./pgv
```

Set the paths to the required SPEF files in the spef_file_list file. They should all point to each of the seven SPEF files generated earlier in Section 2, Netlists and Physical Views, delimited by new lines (Figure 53).

Figure 53: Paths to the SPEF files in spef_file_list

```
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mbfn000aa1d48x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mfun000aa1d12x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0minv000aa1d48x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mlan003aa1d12x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mnanb02aa1d36x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mnorb03aa1d15x5.spf
../stdcells_1278.ldk_demo_lib/spf/ldk_demo_lib_85c_tttt_ctyp/i0mxnrc02aa1n02x5.spf
```

The path might vary if the you change the SPEF generation output path in Section 2, Netlists and Physical Views. Replace the path with the appropriate path.

2. Open the run.tcl file, and then review the contents (Figure 54).

Figure 54: Contents of the run.tcl file

```
eval_legacy {setBetaFeature vtsEnable_GAA_BackSideMetal 1}
set pdk_release_path $env(INTEL_PDK)
set metal_stack $env(LAYERSTACK)
set lib_type i0m_180h_50pp_tp1

set lef_list [list \
$pdk_release_path/apr/cadence/$metal_stack/${lib_type}/p1278.lef \
../stdcells_1278.ldk_demo_lib/lef/ldk_demo_lib.lef \
]
read_physical -lef $lef_list


set_advanced_pg_library_mode -lef_pin_short true -use_embedded_spectre false -tap_node_distance 2 -decap_frequency 1e8 -default_frequency 2e


set_pg_library_mode  \
               -cell_type stdcells  \
               -spice_netlist_file  spef_file_list \
               -spice_models ${pdk_release_path}/models/core/spectre/${metal_stack}/p1278_3.scs \
               -spice_corners tttt \
               -extraction_tech_file ${pdk_release_path}/extraction/qrc/techfiles/${metal_stack}/tttt/qrcTechFile \
               -power_pins {gtdout 0.8 vcc 0.8} \
               -ground_pins {vssx} \
               -current_distribution propagation \
               -temperature 100 \
               -cells_file ./cells.list \
               -design_qrc_layer_map  ${pdk_release_path}/extraction/qrc/asic/${metal_stack}/asic.qrc.map

write_pg_library -out_dir ../stdcells_1278.ldk_demo_lib/pgv

## validation:
set rules {
#########Rules######
CINT_MIN 1.00e-16
CINT_MAX 1.00e-15
}
echo $rules > validate_rule.txt

check_pg_library \
               -list { ../stdcells_1278.ldk_demo_lib/pgv/stdcells.cl } \
               -check_parameters \
               -lef_consistency \
               -rule_file validate_rule.txt \
               -output Validate_PGV \
               -summary

exit
```

As seen in Figure 54, the inputs are given to the `set_pg_library_mode` command, after which we write the power grid library into a directory, and then validate the run results.

3. Type the following command in the terminal:

```
voltus -stylus -no_gui
```

Note: Steps 5-17 are performed in the run.tcl file. You may type `source run.tcl` in the command line to do the next few steps.

4. Type the following code into the terminal, and then press **Enter**:

```
eval_legacy {setBetaFeature vtsEnable_GAA_BackSideMetal 1}
set pdk_release_path $env(INTEL_PDK)
set metal_stack $env(LAYERSTACK)
set lib_type i0m_180h_50pp_tp1
set lef_list [list \
$pdk_release_path/apr/cadence/$metal_stack/${lib_type}/p1278.lef \
../stdcells_1278.ldk_demo_lib/lef/ldk_demo_lib.lef \
]
```

These commands set the required variables that are used in the next steps.

5. Type the following command into the terminal, and then press **Enter**:

```
read_physical -lef $lef_list
```

This reads the LEFs that are needed to generate the PGV flow. The LEF list contains the technology LEF file, as well as the LEF file of the library that requires PGV views.

6. Type the following command into the terminal, and then press **Enter**:

```
set_advanced_pg_library_mode -lef_pin_short true -use_embedded_spectre false -
tap_node_distance 2 -decap_frequency 1e8 -default_frequency 2e8
```

The `set_advanced_pg_library_mode` command is used to specify any advanced power grid generation features that might be needed (Table 17).

Table 17: set_advanced_pg_library_mode options

Option	Usage
lef_pin_short true	Shorts the unconnected LEF pins and ports to establish electrical connectivity.
use_embedded_spectre false	Disables the use of the embedded Cadence Spectre* in Cadence Voltus* and instead uses whichever Spectre is defined in the environment.
tap_node_distance 2	Specifies at which interval a tap point must be placed on the ports; a tap distance of 2 microns is used in this demonstration.
decap_frequency	Specifies the decoupling capacitance frequency; set to 100 MHz for this demonstration.
default_frequency	Specifies the default frequency of nets; set to 200 MHz for this demonstration.

7. Type the following command into the terminal, and then press **Enter**:

```
set_pg_library_mode \
               -cell_type stdcells \
       -spice_netlist_file  spef_file_list \
               -spice_models
${pdk_release_path}/models/core/spectre/${metal_stack}/p1278_3.scs \
               -spice_corners tttt \
               -extraction_tech_file
${pdk_release_path}/extraction/qrc/techfiles/${metal_stack}/tttt/qrcTechFile \
               -power_pins {gtdout 0.8 vcc 0.8} \
               -ground_pins {vssx} \
               -current_distribution propagation \
               -temperature 100 \
-cells_file ./cells.list \
       -design_qrc_layer_map
${pdk_release_path}/extraction/qrc/asic/${metal_stack}/asic.qrc.map
```

Use `set_pg_library_mode` to specify the options being used when generating the PGV views. Table 18 details the options used in this case.

Table 18: set_pg_library_mode options

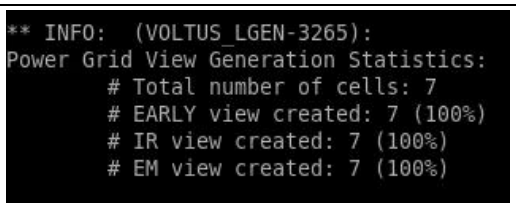
Options	Usage
cell_type	Specifies the type of cells in the library that PGV views are being generated for. This demonstration uses standard cells.
spice_netlist_file	Points to the path of the netlist files being used for PGV generation. These must be transistor-level netlist files.
spice_models	Points to the SPICE model that should be used for simulation.
spice_corners	Specifies which corner to use for simulation. In this demonstration, we use the tttt corner. If needed, you can change the simulation corner by changing this option.
extraction_tech_file	Points to the RC techfile to be used for extraction. We are matching the simulation corner (tttt) for the extraction techfile. The simulation corners and the extraction techfile corners must match.
power_pins	Specifies which pins in the design are power pins and their voltages.
ground_pins	Specifies which pins in the design are ground pins.
current_distribution	Specifies the method of current distribution in the cell: propagation, dynamic_simulation, or specified in a current_region file. This demonstration uses the propagation method.
temperature	Specifies the temperature used for thermal-aware EM/IR analysis.
cells_file	Specifies the file which contains the list of cells for which to generate PGV views.
design_qrc_layer_map	Points to the layer map file mapping the LEF layers to Cadence Quantus* layers.

8. Type the following command in the terminal, and then press **Enter**:

```
write_pg_library -out_dir ../stdcells_1278.ldk_demo_lib/pgv
```

This outputs the PGV views to the specified location. This step should start the actual PGV view generation. Figure 55: shows that the views were successfully generated. Any warnings that might occur during generation are explained in Section 5.4, Warning Explanations.

Figure 55: View generation complete

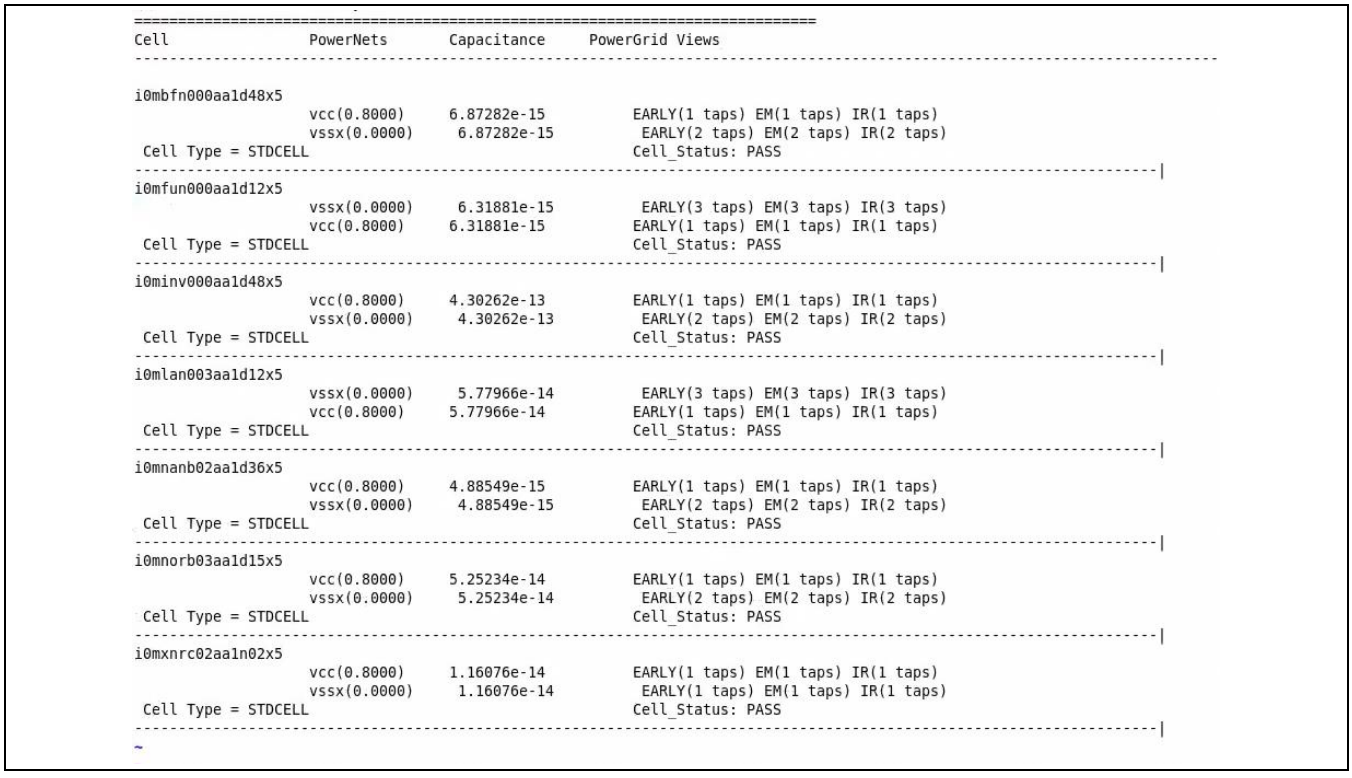


9. Next, the report, summary files, and the actual PGV views are generated in the same specified directory. You can now review those files to see if there are any discrepancies.
10. Open the following file in a text editor:

```
../stdcells_1278.ldk_demo_lib/pgv/stdcells.report
```

Figure 56 shows that the intrinsic capacitances for each cell are within range and do not vary significantly between vcc (power) and vssx (ground).

Figure 56: Content of the stdcells.report file



11. Close the report file, and then open the summary file located at ../stdcells_1278.ldk_demo_lib/pgv/stdcells.summary. This is a much more detailed report on each cell. Because the summary file is much longer, only one cell is shown below for reference (Figure 57).

Figure 57: Contents of the stdcells.summary file

```
Cell: i0mbfn000aa1d48x5   Type: STDCELL         Power Grid Views:  Early  4/4  EM  4/4  IR  -/-   Dynamic Tap Current views:  -/-   Power Gate Views:  -/-
Date Created: 2023-Dec-06 05:39:48 (2023-Dec-06 13:39:48 GMT)
Last Modified: 2023-Dec-06 05:40:34 (2023-Dec-06 13:40:34 GMT)
Bounding Box: (0, 0) - (6000, 3600)
Real Bounding Box: (-850, -400) - (6850, 4000)
Origin: (0, 0)
Foreign: (0, 0)
Total power-net capacitance : 6.87282e-15
Total ground-net capacitance : 6.87282e-15
Total power-net internal-load : 0
Total ground-net internal-load : 0
Total power-net current : 0
Total ground-net current : 0
Temperature: 100
Technology: p1278_3opt12x0p9_tttt
md5sum of techfile: 35caeb2f298946b070f04835f1bd6c55
Cell has FIXEDMASK
Layers containing obstructions: viat, m0, via0, m1
Number of Pins: 4
Tap Node Distance : 2um

  Pin Name        Net Name        Direction       Type            Library Layers
    a             a               INPUT           SIGNAL          via0, m1
    o             o               OUTPUT          SIGNAL          via0, m1
    vcc           vcc             INOUT           POWER           bm0
    vssx          vssx            INOUT           GROUND          bm0

PowerGrid Views

  Net: a              Supply Type: SIGNAL          Supply Voltage: 0.0000
                       Number of partitions Promoted / Internal: 3 / 3
    View                Resistors       Taps / Total Current (Total Worstcase Current )    Library Layers
      EARLY             0               1 / - (-)                                          via0, m1
      EM                0               1 / - (-)                                          via0, m1
      IR                0               1 / - (-)                                          via0, m1

  Net: o              Supply Type: SIGNAL          Supply Voltage: 0.0000
                       Number of partitions Promoted / Internal: 12 / 12
    View                Resistors       Taps / Total Current (Total Worstcase Current )    Library Layers
      EARLY             0               4 / - (-)                                          via0, m1
      EM                3               4 / - (-)                                          m1
      IR                3               4 / - (-)                                          m1

  Net: vcc            Supply Type: POWER           Supply Voltage: 0.8000
                       Number of partitions Promoted / Internal: 1 / 1
    View                Resistors       Taps / Total Current (Total Worstcase Current )    Library Layers
      EARLY             0               1 / - (-)                                          bm0
      EM                0               1 / - (-)                                          bm0
```

12. Designers can also define their own rules and constraints for passing and failing conditions of intrinsic capacitance. You can also validate their generated PGV views by using the `check_pg_library` command, as shown in steps 14-16.
13. Type the following command in the terminal, and then press **Enter**:

```
set rules {
#########Rules######
CINT_MIN 1.00e-16
CINT_MAX 1.00e-15
}
echo $rules > validate_rule.txt
```

This creates a validate_rule.txt file which contains the rules described above. CINT_MIN defines the minimum capacitance value and CINT_MAX defines the maximum. You can choose which max and min capacitance values are suitable for your design.

14. Type the following command in the terminal, and then press **Enter** (see Table 19):

```
check_pg_library \
                -list { ../stdcells_1278.ldk_demo_lib/pgv/stdcells.cl } \
                -check_parameters \
                -lef_consistency \
                -rule_file validate_rule.txt \
                -output Validate_PGV \
                -summary
```

Table 19: check_pg_library options

Option	Usage
list	Specifies the list of standard cell PGV views to validate.
check_parameters	Checks a list of electrical parameters, such as CINT_MIN and CINT_MAX. The list of parameters checked is available in the man pages and also in the product manual.
lef_consistency	Checks whether the LEF and PGV files are consistent.
rule_file	Specifies the rules file, which is used for validation.
output	Specifies where the output of the validation is created.
summary	Used when a more detailed report is needed.

Figure 58 shows what the terminal displays after that command runs:

Figure 58: Result of the check_pg_library command

```
Summary:
PGV and LEF are Consistent
Check file: CheckLefToPgvConsistency.report for detailed Report
```

15. Verify the contents of the Validate_PGV directory; it contains a summary file with detailed information on the Power Grid views.
16. Type the following command in the terminal:

```
exit
```

5.4 Warning explanations

Table 20: Warning explanations

Warning	Description
`**WARN: (IMPLF-200/201)`	Related to the antenna rules. This is a known issue and will be addressed in a future release.
`**WARN: (VOLTUS_LGEN-3087)`	Can be ignored because we use the following option: `-design_qrc_layer_map ${pdk_release_path}/extraction/qrc/asic/${metal_stack}/asic.qrc.map` This covers all the routing layers in the tech node. Other items not covered are ignored as they not extracted for ASIC flow anyway.

Warning	Description
`** WARN: (VOLTUS_LSIM-5004)`	X2.mM22:b bulk node is not connected to a voltage source. These warnings can happen due to the following scenarios: • A bulk node is connected to a pin that is not present at the hierarchy level that we are using • A bulk node is actually floating

To verify if these warnings are legitimate, view the SPEF file manually and check whether the bulk is floating. In this case, they are not floating and these warnings can be ignored. Usually, if a cell is LVS-clean, these warnings are ignorable. There are cases where they are floating and LVS cannot catch it, so it is recommended to always check.

6 Quality Assurance (QA)

6.1 Timing (Liberty) QA

Cadence Liberty LV* has many Liberty checking features. The following checks are performed in this LDK:

- **Data Range:** Check that data is within the specified range using the `validate_data_range` command. You can specify thresholds for min and max range in the file. The tool also checks for two or more consecutive zeros in the data table.
- **Monotonicity:** Check data is monotonically increasing (when input slew or output load is increasing) using the `validate_monotonicity` command in Liberate LV. Data types checked include:
  - cell_rise
  - cell_fall
  - rise_transition
  - fall_transition
  - mpw
- **CCS versus NLDM:** Compare the CCS data to the NLDM data in a single library and report any difference that exceeds the defined tolerance. Tolerance threshold is set to 1 ps, 1%.
- **CCSN:** Data consistency check using the `check_ccsn` command. It checks for CCB attributes and overall CCSN data and structure.
- **LVF:** LVF data consistency checks using the `check_lvf_data` command in Cadence Liberate LV*. This checks for:
  - Out-of-bound values
  - OCV out-of-range values
  - Consistency between LVF moments and OCV early/late sigma values
  - Early / late sigma ratio
  - OCV sigma by nom ratio
  - Delay / trans sigma ratio
- **Validate LVF:** Check using the `validate_lvf_data` command in Cadence Liberate LV*. This checks for:
  - Missing OCV arc
  - NLDM table and OCV table having different size
  - Any zero value for OCV

Detail information of these checks can be found in the Cadence Liberate LV* manual.

6.1.1 Setup Environment and run directory

1. Change the directory to the LDK work area ./ldk_r0.9_cdns/training/setup/cadence/ldk.
2. If this is the first time launching Cadence tools from the shell you are working in, then source the sourceme file.
3. Change to the characterization working directory:

```
cd $LDK_WORKAREA/training/setup/cadence/ldk/char
```

6.1.2 Pre-requisite

Existence of the following .lib files is required in the directory $LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/base_ulvt.

- lib783_i0m_180h_50pp_base_ulvt_tttt_0p700v_85c_tttt_ctyp_ccslnt.lib
- lib783_i0m_180h_50pp_base_ulvt_tttt_0p700v_85c_tttt_ctyp_nldm.lib

The files were generated in Section 3.8.

6.1.3 QA – Liberty

1. Run the setup script to create the QA run directory, and then cd to the directory:

```
$INTEL_LDK/training/libraries/char/setup/CreateSetup.csh qa
cd qa
```

2. Run QA on ccslnt lib:

```
./run_QA.csh
$LDK_WORKAREA/training/setup/cadence/ldk/stdcells_1278.ldk_demo_lib/lib/base_
ulvt/lib783_i0m_180h_50pp_base_ulvt_tttt_0p700v_85c_tttt_ctyp_ccslnt.lib
```

   Run time is less than five minutes.

   The log files are written to the directory ./logs/.

   The output reports are written to the directory ./reports/<lib_name>/. For example: ./reports/lib783_i0m_180h_50pp_base_ulvt_tttt_0p700v_85c_tttt_ctyp_ccslnt/

   If the specified lib file is an NLDM lib file, only Data Range and Monotonicity checks are run. The other checks apply to data such as CCS, CCSN, and LVF that does not exist in the NLDM lib file.

6.2 Power Grid views QA

The PGV view generation section (Section 5) has its own QA section which is covered by the check_pg_library option (steps 15 and 16). These steps cover the basic QA checks, such as parameter checks and LEF consistency checks. These are detailed in Section 5; there are no lab steps in this section.

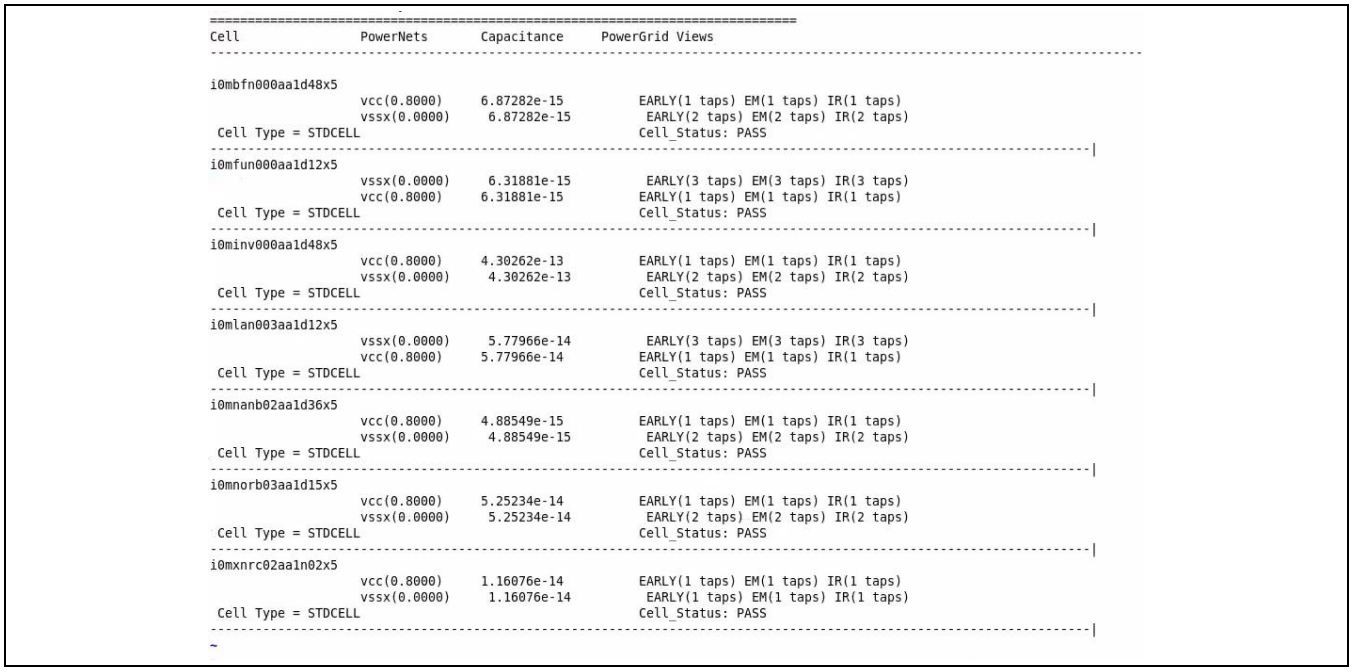
```
check_pg_library
                -list { ../stdcells_1278.ldk_demo_lib/pgv/stdcells.cl } \
                -check_parameters \
                -lef_consistency \
                -rule_file validate_rule.txt \
                -output Validate_PGV \
                -summary
```

Figure 59: LEF Consistency Check results

```
Summary:
PGV and LEF are Consistent
Check file: CheckLefToPgvConsistency.report for detailed Report
```

The output of that step can be found at $LDK_WORKAREA/pgv/Validate_PGV.

Figure 60: Summary file contents



```
================================================================================
Cell                PowerNets       Capacitance     PowerGrid Views
--------------------------------------------------------------------------------------------------------------------------------

i0mbfn000aa1d48x5
                    vcc(0.8000)     6.87282e-15          EARLY(1 taps) EM(1 taps) IR(1 taps)
                    vssx(0.0000)     6.87282e-15          EARLY(2 taps) EM(2 taps) IR(2 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0mfun000aa1d12x5
                    vssx(0.0000)     6.31881e-15          EARLY(3 taps) EM(3 taps) IR(3 taps)
                    vcc(0.8000)     6.31881e-15          EARLY(1 taps) EM(1 taps) IR(1 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0minv000aa1d48x5
                    vcc(0.8000)     4.30262e-13          EARLY(1 taps) EM(1 taps) IR(1 taps)
                    vssx(0.0000)     4.30262e-13          EARLY(2 taps) EM(2 taps) IR(2 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0mlan003aa1d12x5
                    vssx(0.0000)     5.77966e-14          EARLY(3 taps) EM(3 taps) IR(3 taps)
                    vcc(0.8000)     5.77966e-14          EARLY(1 taps) EM(1 taps) IR(1 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0mnanb02aa1d36x5
                    vcc(0.8000)     4.88549e-15          EARLY(1 taps) EM(1 taps) IR(1 taps)
                    vssx(0.0000)     4.88549e-15          EARLY(2 taps) EM(2 taps) IR(2 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0mnorb03aa1d15x5
                    vcc(0.8000)     5.25234e-14          EARLY(1 taps) EM(1 taps) IR(1 taps)
                    vssx(0.0000)     5.25234e-14          EARLY(2 taps) EM(2 taps) IR(2 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
i0mxnrc02aa1n02x5
                    vcc(0.8000)     1.16076e-14          EARLY(1 taps) EM(1 taps) IR(1 taps)
                    vssx(0.0000)     1.16076e-14          EARLY(1 taps) EM(1 taps) IR(1 taps)
 Cell Type = STDCELL                                     Cell_Status: PASS
--------------------------------------------------------------------------------------------------------------------------------|
~
```

This directory has a summary file which can be reviewed to check for data discrepancies, such as large capacitances across rails, and so on.

6.3 Layout Versus Schematic: GDS versus CDL

Use the LVS process to verify that the physical layout implementation connectivity matches the Schematic (or Netlist). The flow steps to run LVS are explained in Section 2.5. There are no lab steps in this section. When running the SPEF generation, it runs the LVS, and then generates the LVS run directory at ./autoLib/<cell_name>/SPEF/ location.

To verify the LVS results, view the lvs.report.cls file from that location, above.

Figure 61: LVS Results file

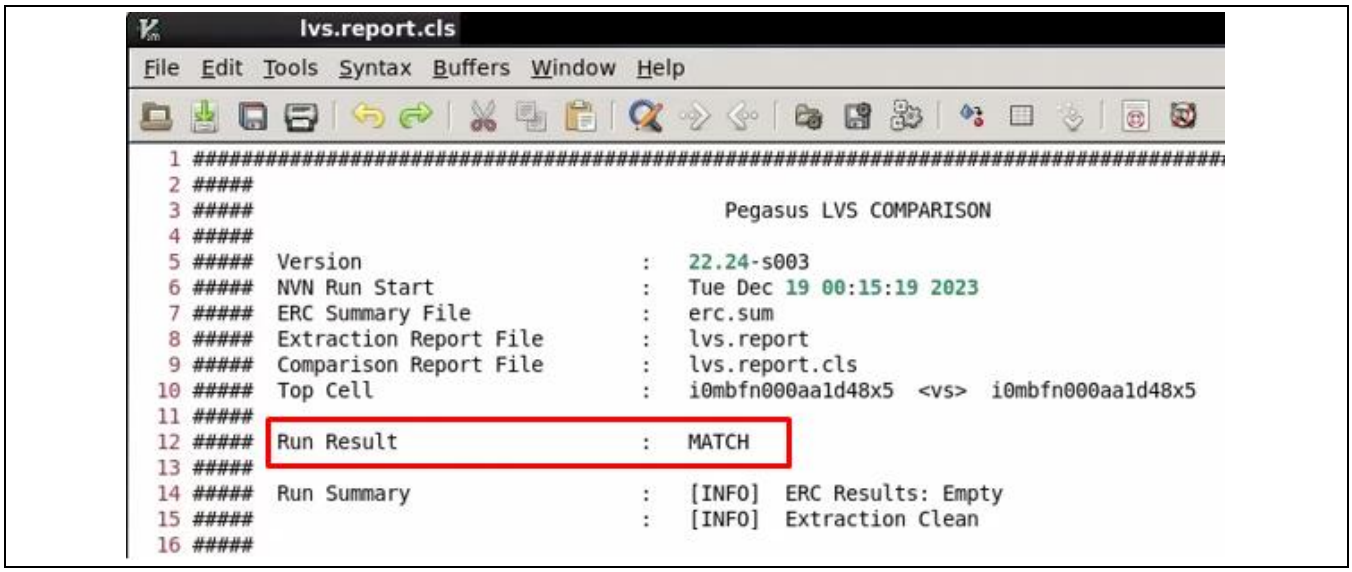


Figure 61 highlights the Run Result representing the LVS run as a match.

6.4 Cadence Pegasus FastXOR*

Use Cadence Pegasus FastXOR* to compare two layout views/GDS files.

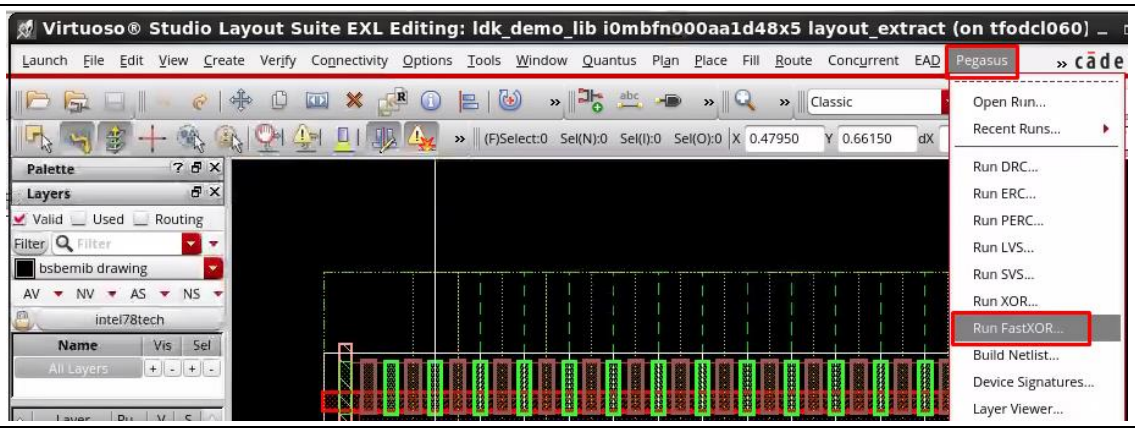
This section compares the GDS file with the layout_extract view of a cell.

1. Open the following cell/view in read mode:

```
Lib/Cell/View:  ldk_demo_lib /i0mbfn000aa1d48x5 /layout_extract
```

2. From the *Layout* menu, select **Pegasus > Run FastXOR...**

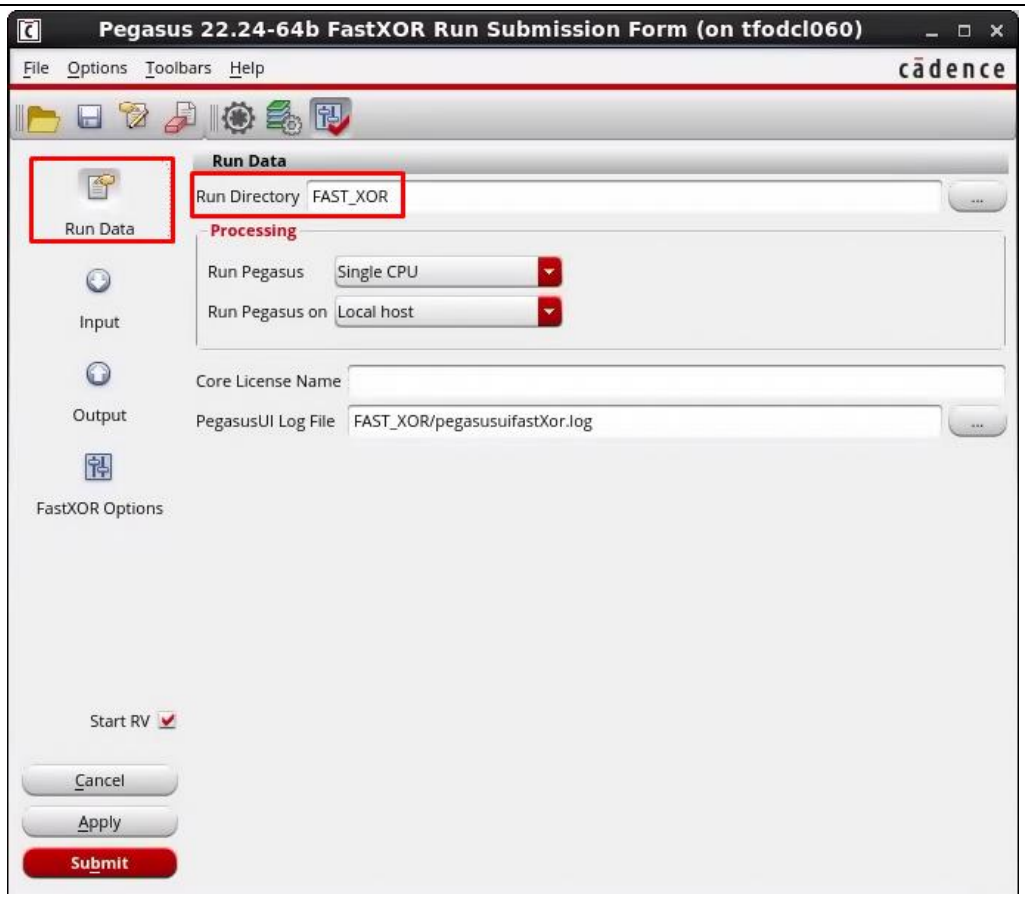
Figure 62: Opening Cadence Pegasus FastXOR*



3. The Cadence Pegasus FastXOR* Run Submission form displays (Figure 63).

   In the *Run Data* section of the form, in the *Run Directory* field, specify **FAST_XOR**.

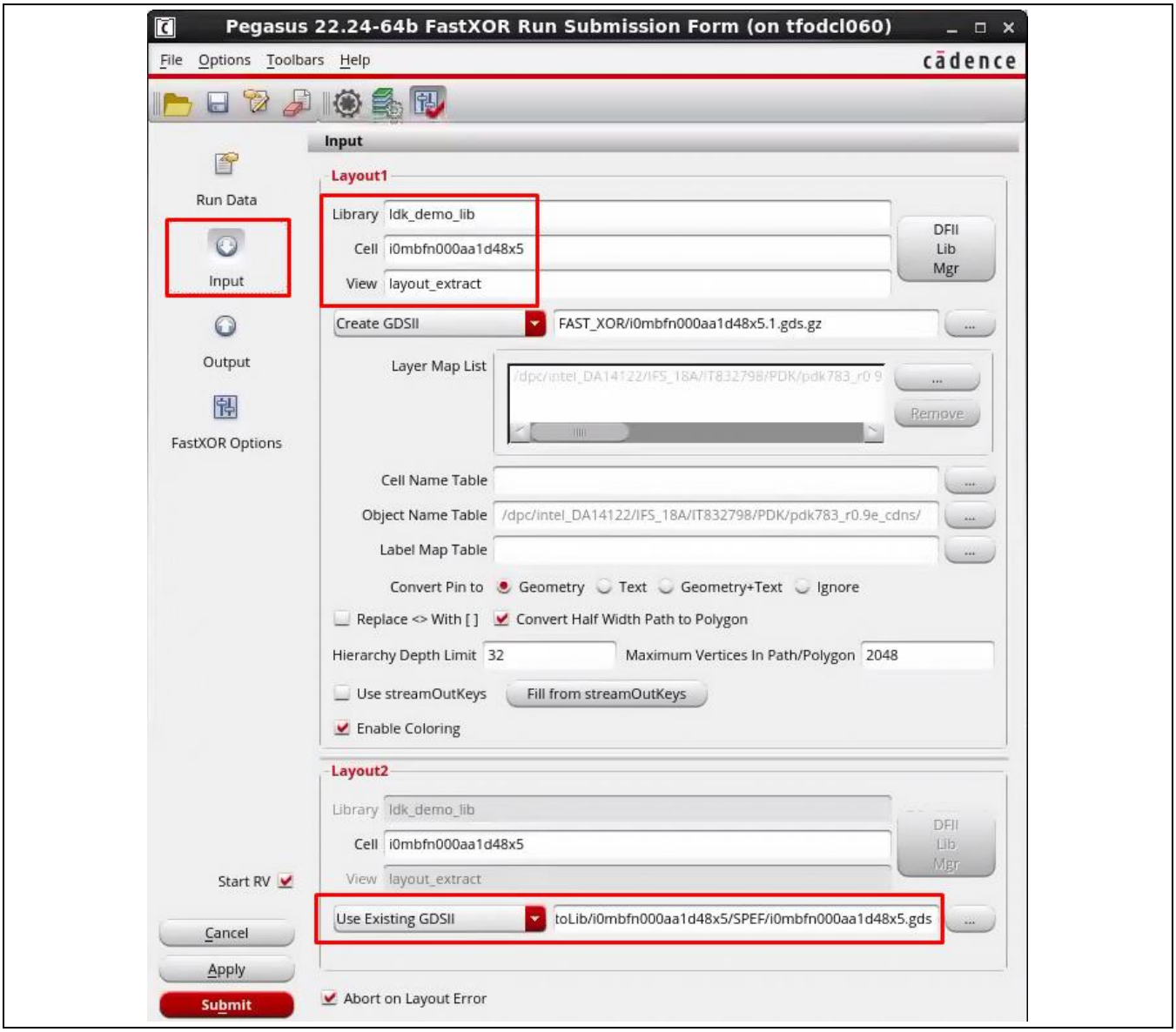
Figure 63: Pegasus FastXOR* form - Run Data section



4. In the *Input* section, in the *Layout1*region, ensure the correct lib/cell/view is selected as highlighted in Figure 64 (this auto-populates because we launched Cadence Pegasus FastXOR* from the same Layout view).
5. In the *Layout2* region, select **Use Existing GDSII**, and then click on the three dots to select the GDSII file.
6. The FastXOR form displays; select the input GDS file: ./autoLib/i0mbfn000aa1d48x5/SPEF/i0mbfn000aa1d48x5.gds"

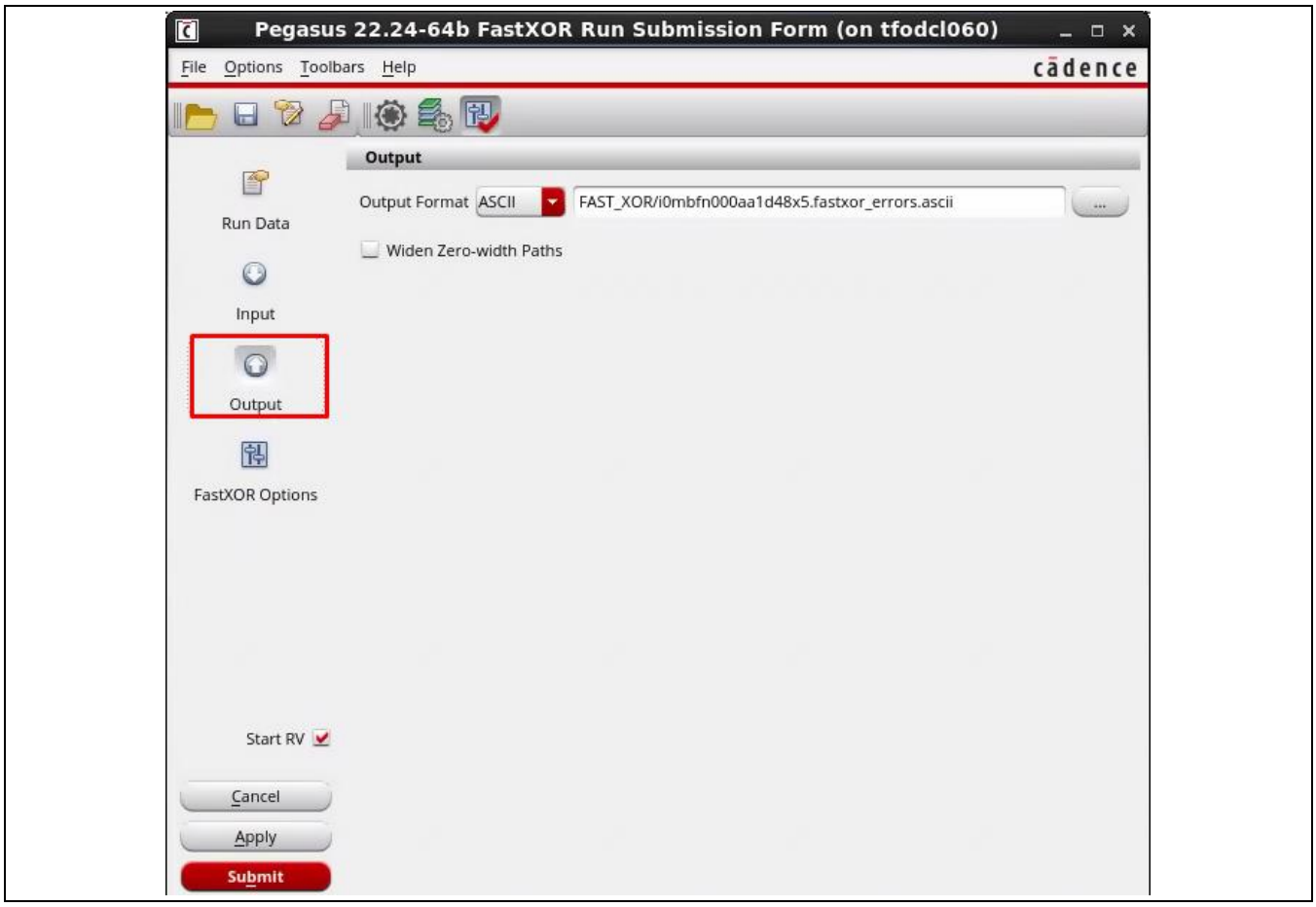
   After the setup, as mentioned above, the FastXOR form should look like Figure 64.

Figure 64: Pegasus FastXOR* form - Input section



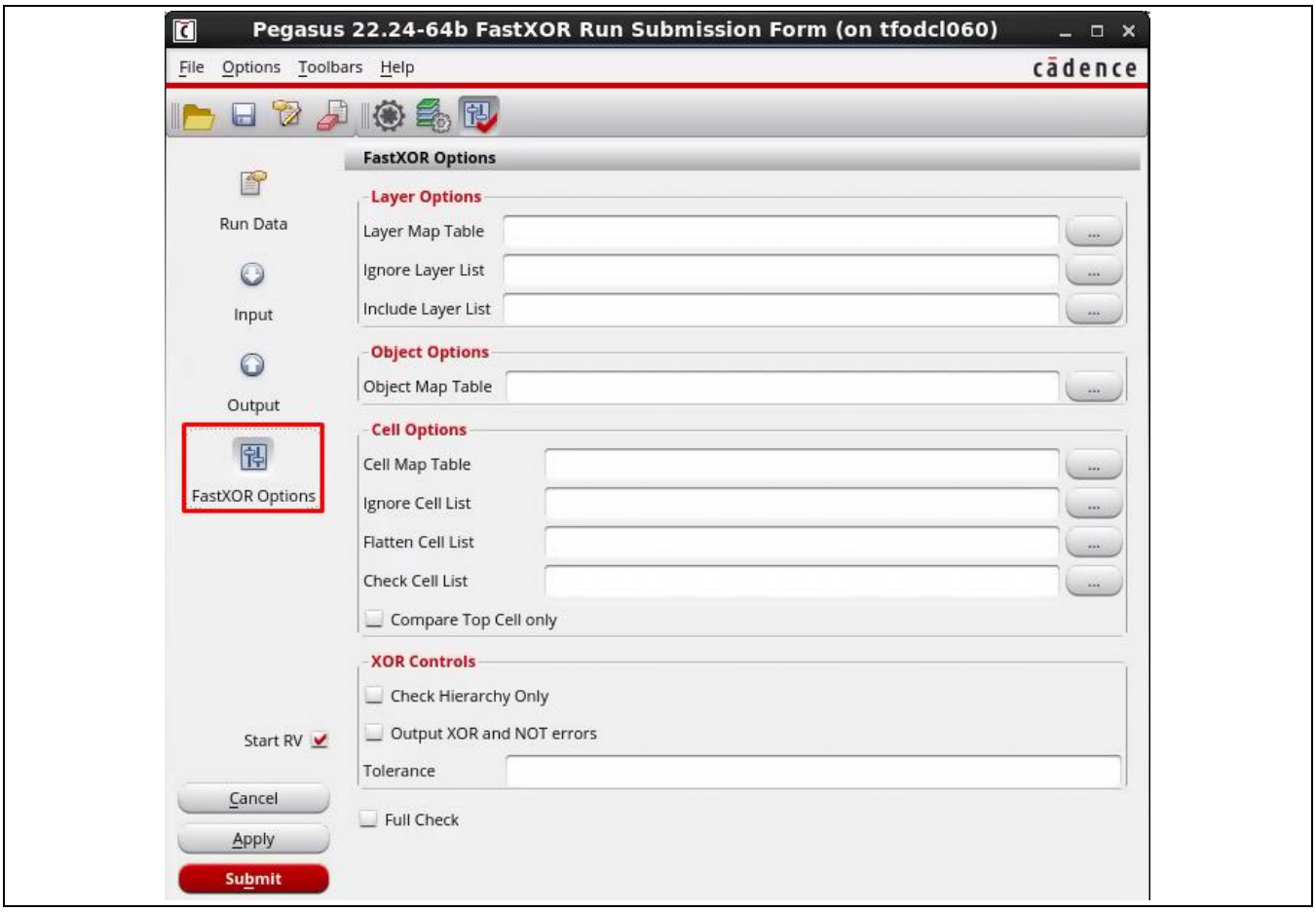
7. Keep the default settings in the *Output* section of the form:

Figure 65: Pegasus FastXOR* form - Output section



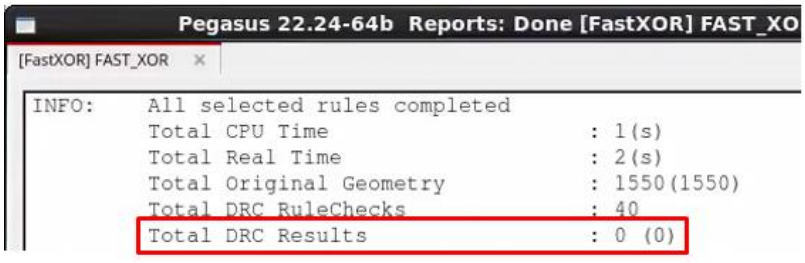
8. In the *FastXOR Options* section, ensure nothing is enabled, and that no other options are specified.

Figure 66: Cadence Pegasus FastXOR* - FastXOR Options section



9. Click **Submit** to run FastXOR. This opens a Cadence Pegasus* Report window. When the run completes, review the comparison (Figure 67). The Total DRC Results is 0, which means the comparison passed the check.

Figure 67: FastXOR report

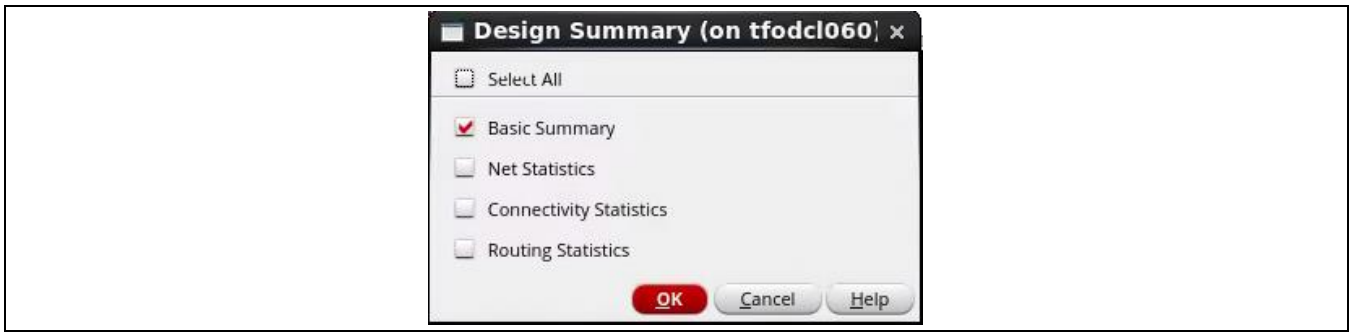


6.5 Layout Summary Report comparison (optional)

In this section, the Layout Summary Report is generated with the layers statistics to compare the layouts.

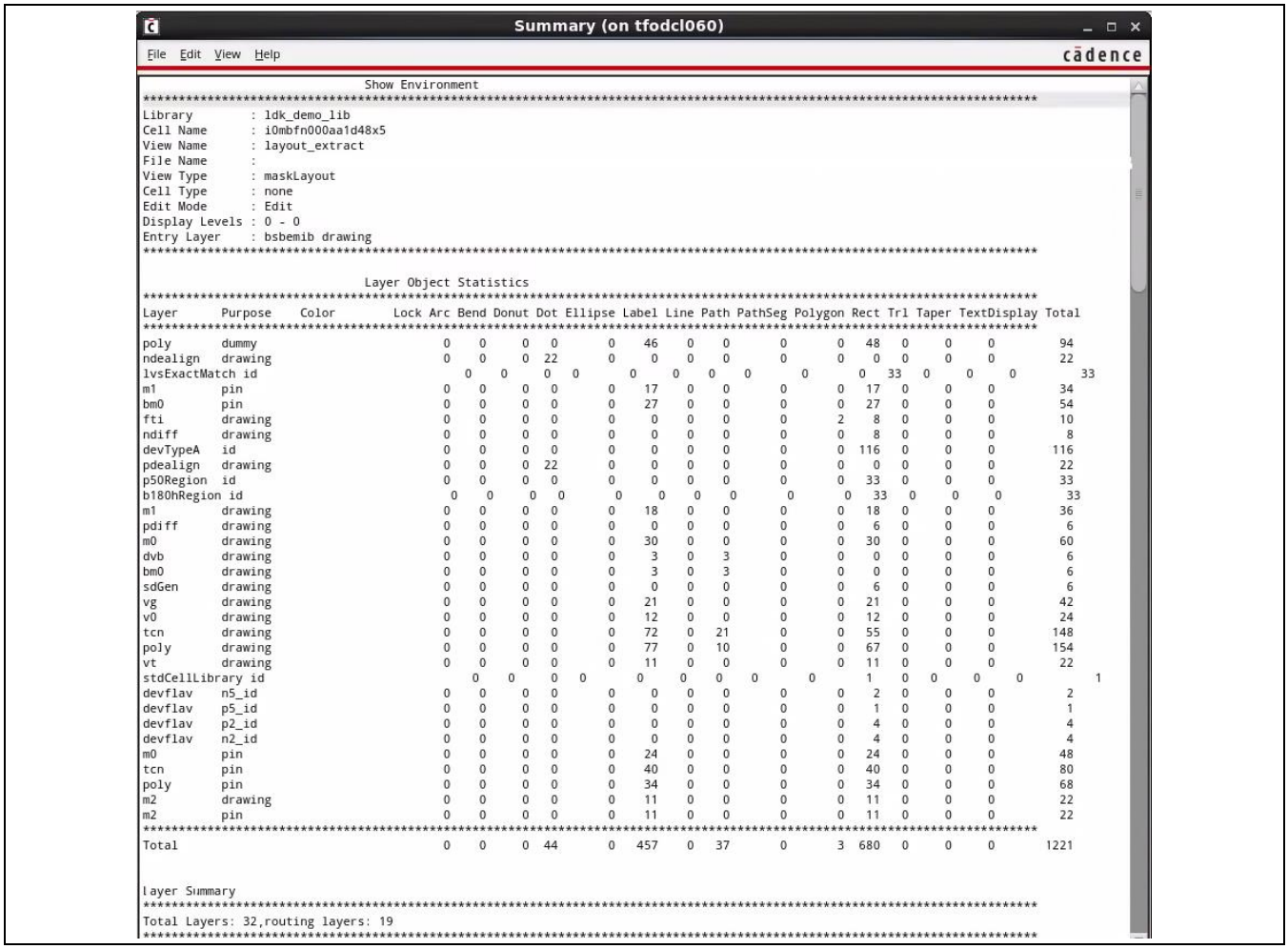
1. From the *Layout* menu, select **File > Summary**.
2. The Design Summary window displays; select only the **Basic Summary** option, and then click **OK**.

Figure 68: Design Summary window



The Summary window for the design displays and shows the Layout Objects Statistics (Figure 69), which shows the total shapes present in the layout.

Figure 69: Design Summary



Summary (on tfodcl060)

File Edit View Help

cadence

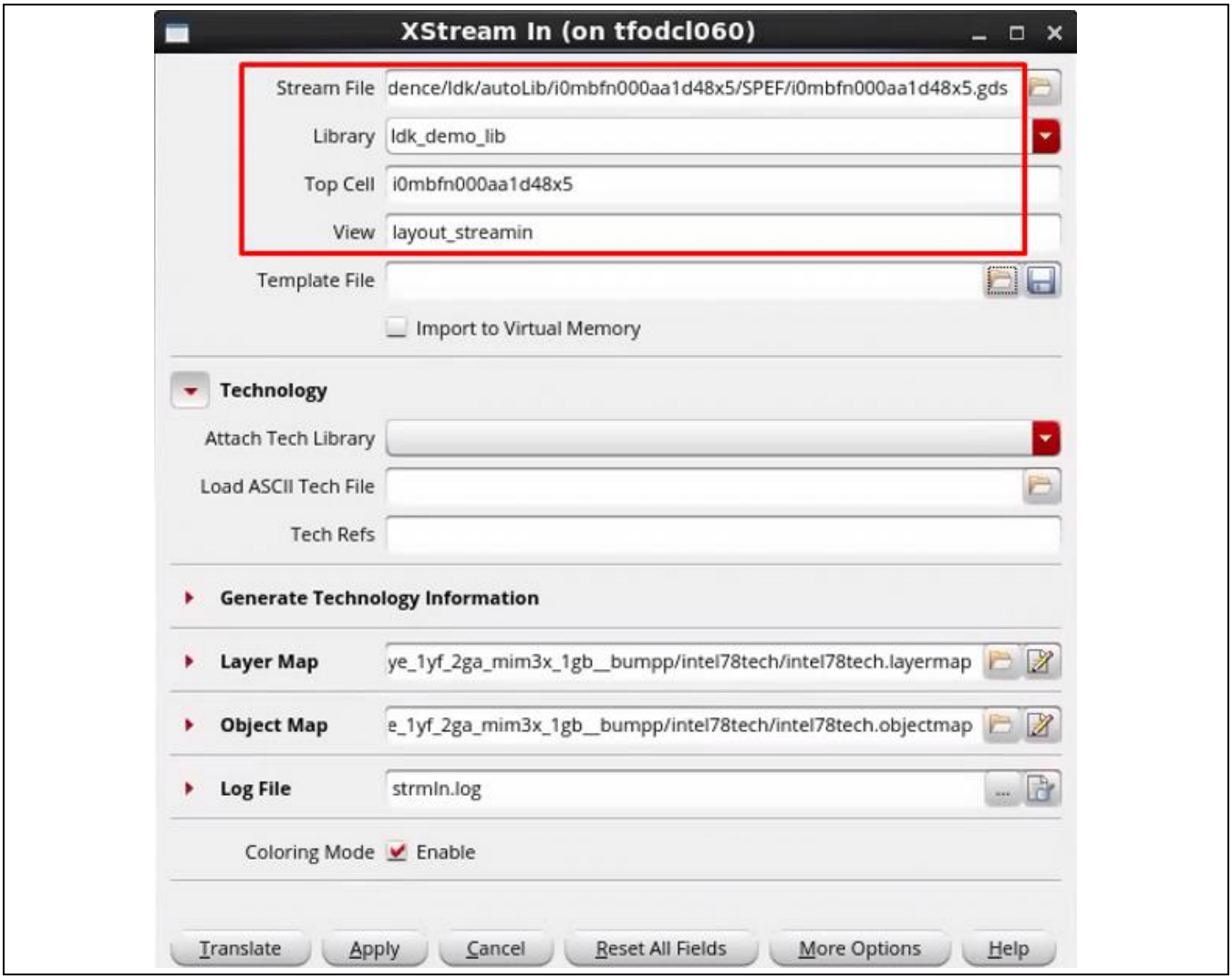
```
                              Show Environment
*****************************************************************************************************************************************
Library        : ldk_demo_lib
Cell Name      : i0mbfn000aa1d48x5
View Name      : layout_extract
File Name      :
View Type      : maskLayout
Cell Type      : none
Edit Mode      : Edit
Display Levels : 0 - 0
Entry Layer    : bsbemib drawing
*****************************************************************************************************************************************

                                 Layer Object Statistics
*****************************************************************************************************************************************
Layer      Purpose    Color        Lock Arc Bend Donut Dot Ellipse Label Line Path PathSeg Polygon Rect Trl Taper TextDisplay Total
*****************************************************************************************************************************************
poly       dummy                     0   0    0     0   0       0    46    0    0       0       0   48   0     0           0    94
ndealign   drawing                   0   0    0    22   0       0     0    0    0       0       0    0   0     0           0    22
lvsExactMatch id                       0   0    0    0   0       0     0    0    0       0       0    0  33     0     0     0          33
m1         pin                       0   0    0    0   0       0    17    0    0       0       0   17   0     0           0    34
bm0        pin                       0   0    0    0   0       0    27    0    0       0       0   27   0     0           0    54
fti        drawing                   0   0    0    0   0       0     0    0    0       0       2    8   0     0           0    10
ndiff      drawing                   0   0    0    0   0       0     0    0    0       0       0    8   0     0           0     8
devTypeA   id                        0   0    0    0   0       0     0    0    0       0       0  116   0     0           0   116
pdealign   drawing                   0   0    0   22   0       0     0    0    0       0       0    0   0     0           0    22
p5ORegion  id                        0   0    0    0   0       0     0    0    0       0       0   33   0     0           0    33
b180hRegion id                        0   0    0    0   0       0     0    0    0       0       0   33   0     0     0          33
m1         drawing                   0   0    0    0   0       0    18    0    0       0       0   18   0     0           0    36
pdiff      drawing                   0   0    0    0   0       0     0    0    0       0       0    6   0     0           0     6
m0         drawing                   0   0    0    0   0       0    30    0    0       0       0   30   0     0           0    60
dvb        drawing                   0   0    0    0   0       0     3    0    3       0       0    0   0     0           0     6
bm0        drawing                   0   0    0    0   0       0     3    0    3       0       0    0   0     0           0     6
sdGen      drawing                   0   0    0    0   0       0     0    0    0       0       0    6   0     0           0     6
vg         drawing                   0   0    0    0   0       0    21    0    0       0       0   21   0     0           0    42
v0         drawing                   0   0    0    0   0       0    12    0    0       0       0   12   0     0           0    24
tcn        drawing                   0   0    0    0   0       0    72    0   21       0       0   55   0     0           0   148
poly       drawing                   0   0    0    0   0       0    77    0   10       0       0   67   0     0           0   154
vt         drawing                   0   0    0    0   0       0    11    0    0       0       0   11   0     0           0    22
stdCellLibrary id                        0   0    0   0   0       0     0    0    0       0       0    1   0     0     0     0           1
devflav    n5_id                     0   0    0    0   0       0     0    0    0       0       0    2   0     0           0     2
devflav    p5_id                     0   0    0    0   0       0     0    0    0       0       0    1   0     0           0     1
devflav    p2_id                     0   0    0    0   0       0     0    0    0       0       0    4   0     0           0     4
devflav    n2_id                     0   0    0    0   0       0     0    0    0       0       0    4   0     0           0     4
m0         pin                       0   0    0    0   0       0    24    0    0       0       0   24   0     0           0    48
tcn        pin                       0   0    0    0   0       0    40    0    0       0       0   40   0     0           0    80
poly       pin                       0   0    0    0   0       0    34    0    0       0       0   34   0     0           0    68
m2         drawing                   0   0    0    0   0       0    11    0    0       0       0   11   0     0           0    22
m2         pin                       0   0    0    0   0       0    11    0    0       0       0   11   0     0           0    22
*****************************************************************************************************************************************
Total                                0   0    0   44   0       0   457    0   37       0       3  680   0     0     0        1221


Layer Summary
*****************************************************************************************************************************************
Total Layers: 32,routing layers: 19
*****************************************************************************************************************************************
```

3. To save the Summary file, select **File > Save As**, and then give the name **layout_extract.summary**.

Next, you stream a GDS file into a Layout view and compare the layout with the previously available Summary report.

1. From the Cadence Virtuoso* CIW, select **File > Import > Stream...** The *XStream In* form displays (Figure 70).
2. Select the Stream file: ./autoLib/<cell_name>/SPEF/<cell_name>.gds
3. Specify names in the *Library*, *Top Cell*, and *View* regions for the GDS file stream-in.

Figure 70: XStream In form



4. Click **Translate**.
5. When the stream-in completes, the following message displays (Figure 71):

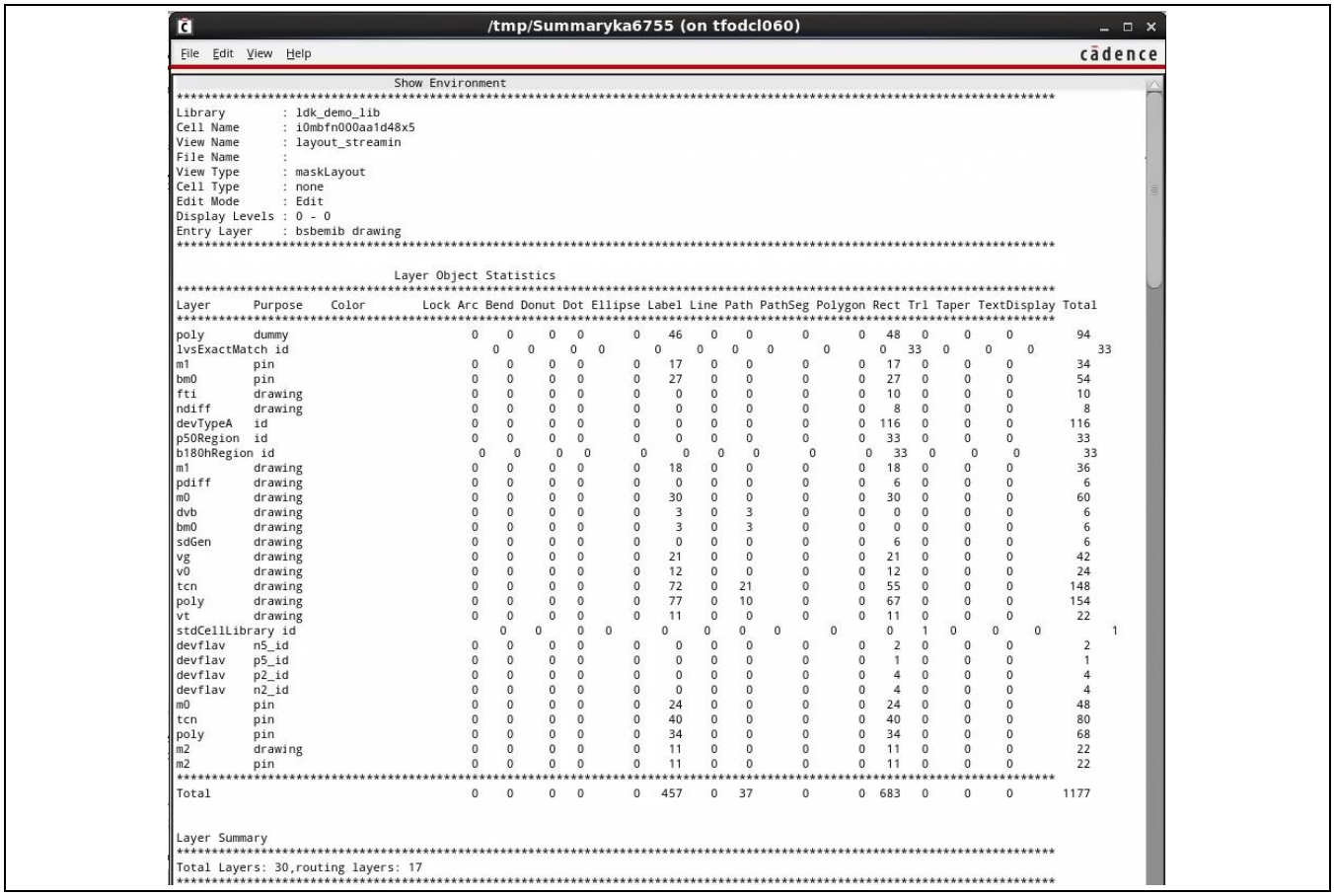
Figure 71: Stream in translation complete notification

6. Click **No**, and then click **Cancel** in the XStream In form.

In the Library Manager, verify the newly generated layout view is available.

1. Open the Lib/Cell/View: ldk_demo_lib / i0mbfn000aa1d48x5 / layout_streamin
2. From the *Layout* menu, select **File > Summary**.
3. In the Design Summary window, select only **Basic Summary**. The Summary window displays (Figure 72).

Figure 72: Summary window



4. Click **File > Save As** to save the Summary file. Name the file: **layout_streamin.summary**.

   In the layout_streamin summary report, 1177 shapes are present; in the layout_extract summary report, 1221 shares are present.

   In the layout_extract, ndealign (22 dot shapes) and pdealign (22 dot shapes) are present only for id purposes. These are ignored in the layermap file and have (0 0) data type value for GDS.

6.6 Schematic versus CDL

Schematic versus CDL is run using the Cadence Pegasus SVS* utility to ensure the exactness of the generated CDL file. This section provides instructions to run schematic versus CDL using Cadence Pegasus* on one cell in the ldk_demo_lib library.

6.6.1 Setup environment and run directory

1. Change the directory to the LDK work area ./ldk_r0.9_cdns/training/setup/cadence/ldk.
2. If this is the first time launching Cadence* tools, source the sourceme file from the UNIX terminal.

6.6.2 User actions

1. Open the .simrc file using the text editor, and then comment out the first two lines using the ";" character:

```
;cdlNetlistType='fnl
;auCdlFnlRetainPathInInstAndNets = t
```

2. Launch Cadence Virtuoso* using the following command:

```
virtuoso &
```

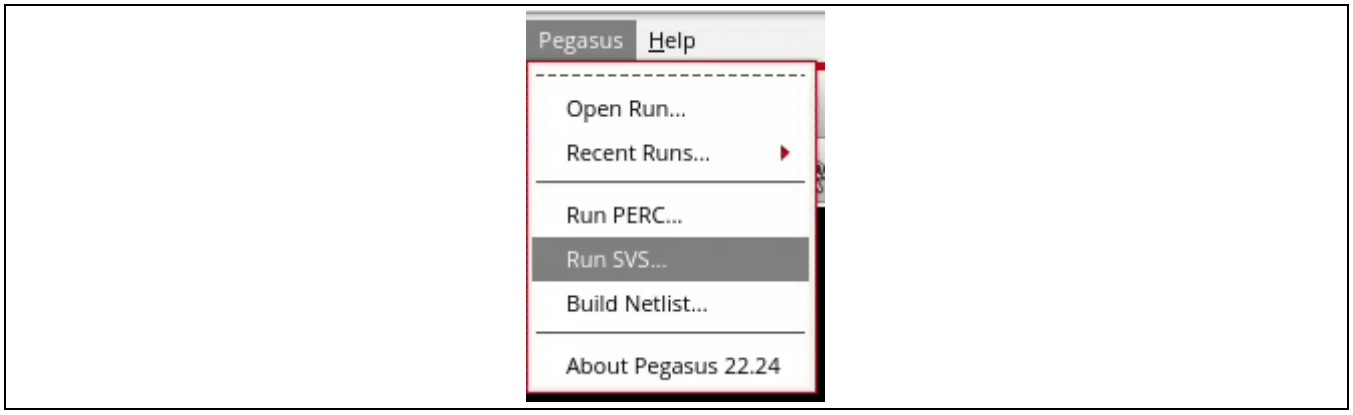
   The Cadence Virtuoso* session should not be launched before changing the .simrc file so that the updates done to the .simrc file are in effect for the current session.

3. In the Library Manager, open the following view for reading:

```
Library: ldk_demo_lib
Cell: i0mbfn000aa1d48x5
View: schematic
```

4. In the Schematic window, select **Pegasus > Run SVS...** (Figure 73).

Figure 73: Run Pegasus SVS



5. In the Cadence Pegasus SVS* window, click **Run Data**, and then type **./SVS** in the *Run Directory* field (Figure 74).

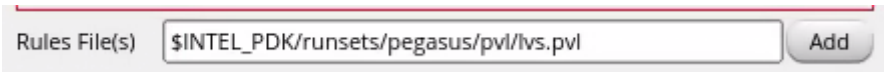
Figure 74: Cadence Pegasus SVS* - Run Data section

6. In the *Rules* tab, in the text box next to *Rules File(s)*, type:

```
$INTEL_PDK/runsets/pegasus/pvl/lvs.pvl
```

7. Click **Add**.

Figure 75: Adding LVS Rules file

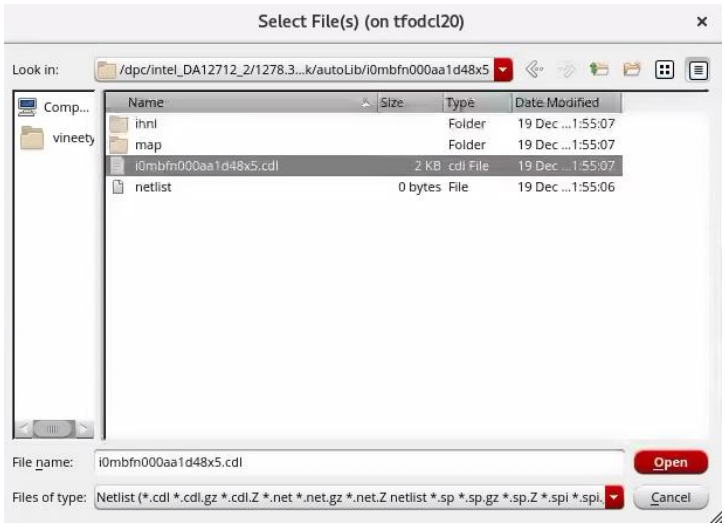


8. Select the *Input* tab; in the *Schematic Side* section, change the radio button to **Netlist**, and then click **Add**.
9. Navigate to ./autoLib/i0mbfn000aa1d48x5, and then click **i0mbfn000aa1d48x5.cdl**.

   Note: You might have to navigate to the correct directory if you used anything other than the default autoLib directory when generating CDL files in Section 2.

10. Click **Open**.

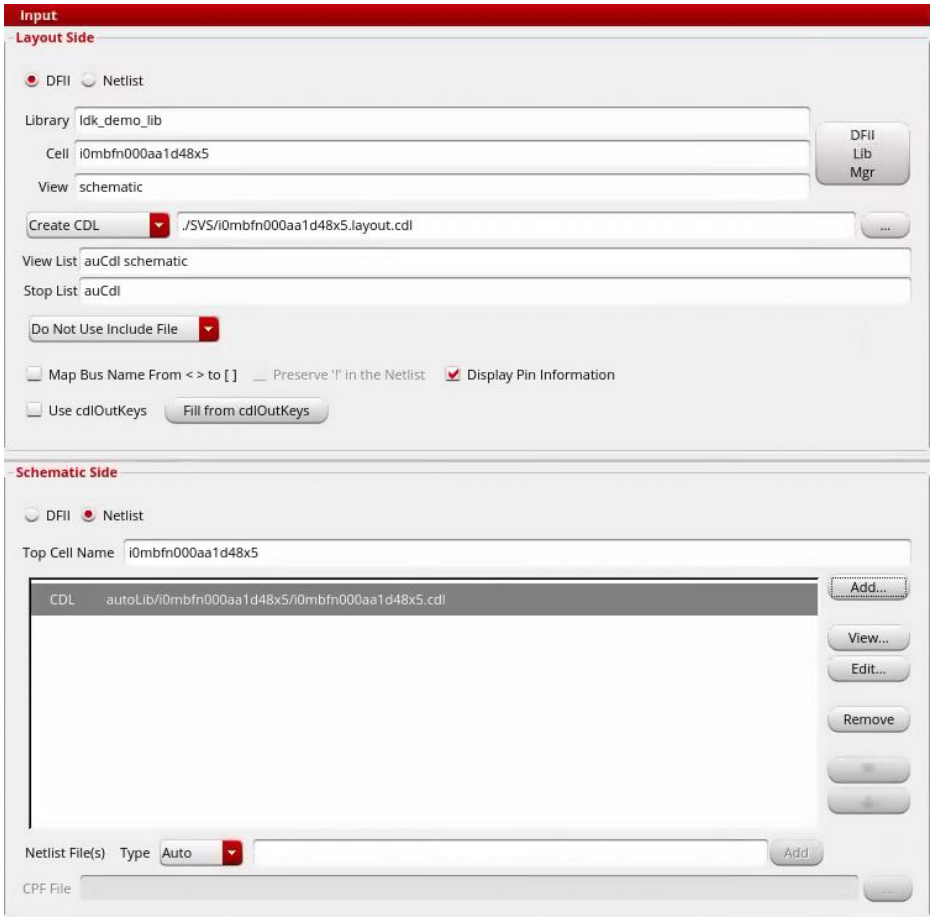
Figure 76: Selecting CDL Netlist



Notice that the schematic side has the cdl file that was generated in Section 2 using the *Auto Lib View Generation* utility, and the layout side has the DFII schematic view. Cadence Pegasus* SVS compares these two.

Even though the view is called *Layout Side*, it is the view that is netlisted by Cadence Pegasus*. Because the view that is being specified is a schematic view, the schematic view is used to create a CDL file for the comparison.

Figure 77: Cadence Pegasus SVS* - Input section



11. Click **Submit**. Cadence Pegasus SVS* starts the comparison, and then returns a report showing that the schematic and CDL netlist match (Figure 78).

Figure 78: SVS Match dialog box

6.7 Flow Testing Standard Cells

To ensure proper functioning of newly developed standard cells, it is recommended to take them through the digital design and implementation flows. This involves synthesizing a gate-level netlist that incorporates the new cells, implementing the design using digital P&R tools, and performing sign-off validation, including both functional and physical validation. The primary purpose of this testing is consistency and naming checks between file formats (LEF, LIB, PGV, CDL, and so on), and documentation of any INFO or WARNING statements which can waived.

It is important to note that the specific flows used in this process are best left to expert users familiar with the foundry process and EDA tools, and is beyond the scope of this document.

6.8 Cross View Check

Use the Cross View Check to compare cells, pins, cell attributes, and pin attributes between different views. Currently supported views are Liberty, LEF, and SPICE-type views (cdl, dspf, spectre, spice).

When performing the comparison between different views, keep in mind that views contain different types of information. For example, both Liberty and LEF views contain area information, so when comparing Liberty and LEF, include cell attribute area in the comparison.

The brute force way is to compare all possible view types against each other. This results in (N-1)! combinations and unnecessary work. The suggested methodology is to use Liberty view as the reference and compare all other view types against it. Table 21 describes each view type comparison, and the type of data that is compared.

Table 21: View type comparison

Reference View	Compare View	Compare Data Types	Cell Attribute	Pin Attribute
Liberty	LEF	Cell, Pin	area	direction
Liberty	CDL	Cell, Pin		direction
Liberty	SPF	Cell, Pin		

This feature is implemented based on the Liberate API and must be run using Liberate.

The source code is located here: $INTEL_LDK/training/libraries/scripts/view_check_functions.tcle

An example run script is provided in the LDK: $INTEL_LDK/training/libraries/scripts/xview_check_all.tcl

Copy xview_check_all.tcl to your local working directory, and then update the view file paths in the script.

To run the cross view check:

```
liberate xview_check_all.tcl
```

Review generated reports: ./xview_check_*.rpt

6.8.1 View Check Command Reference

view_check_read_lib [options]

Read Liberty files and generate view check DB.

Options:

-files <list of input files>

Examples:

set db_lib [view_check_read_lib -files lib/test.lib]

_____

view_check_read_lef [options]

Read LEF files and generate view check DB.

Options:

-files <list of input files>

Examples:

set db_lef [view_check_read_lef -files lef/test.lef]

_____

view_check_read_spice [options]

Read SPICE-type files and generate view check DB.

Options:

-type <type of SPICE>

Supported types: [spice | spectre | dspf | cdl]

-power_pins <list of power pins>

-ground_pins <list of ground pins>

-dir <directory containing input files>

-file_ext <file extension of input files in directory specified by -dir>

-files <list of input files>

Examples:

set db_spf [view_check_read_spice -power_pins vcc -ground_pins vssx -type dspf -dir spf -file_ext spf]

_____

xview_check [options] <db_ref> <db_cmp>

Cross view comparison: comparison between 2 views.

The output report is organized into 4 different sections:

1. Compare Cell: a table listing cells from reference and compare database
2. Compare Cell Attribute: a table listing cells and the cell's attributes
3. Compare Pin: a table listing cells and the cell's pins
4. Compare Pin Attribute: one table for each cell, a table containing all pins and their attributes and values

Output report legend:

< - mismatch
? - missing

Options:

-compare_types <list of data types to compare>

Supported data types: * cell pin cell_attribute pin_attribute

-cells <list of cells>

List of cells to include (or exclude if -exclude_cells). Supports wildcard.

-exclude_cells

-pins

List of pins to include (or exclude if -exclude_pins). Supports wildcard.

-exclude_pin

-cell_attrs <list of cell attributes>

List of cell attributes to include (or exclude if -exclude_cell_attrs). Supports wildcard.

-exclude_cell_attrs

-pin_attrs <list of cell attributes>

List of pin attributes to include (or exclude if -exclude_pin_attrs). Supports wildcard.

-exclude_pin_attrs

-abstol <absolute tolerance>

Absolute tolerance applied to numeric cell / pin attribute comparison; for example, area. Default = 0.001.

-reltol <relative tolerance>

Relative tolerance applied to numeric cell / pin attribute comparison; for example area. Default = 0.01 (1%).

-report <output report file>

Support stdout.

-verbose

Include matched data in report.

Examples:

Compare Liberty vs. LEF: include cell attribute area and pin attribute direction in comparison.

```
set db_lib [view_check_read_lib -files lib/test.lib]
set db_lef [view_check_read_lef -files lef/test.lef]
xview_check -cell_attrs {area} -pin_attrs {direction} -report lib2lef.rpt $db_lib $db_lef
```

Compare Liberty vs. DSPF

```
set db_lib [view_check_read_lib -files lib/test.lib]
set db_dspf [view_check_read_spice -type dspf -power_pins vcc -ground_pins vssx -dir spf -file_ext spf]
xview_check -compare_types {cell pin} -report lib2dspf.rpt $db_lib $db_dspf
```

Compare Liberty vs. CDL

```
set db_lib [view_check_read_lib -files lib/test.lib]
set db_cdl [view_check_read_spice -type cdl -power_pins vcc -ground_pins vssx -files cdl/test.cdl]
xview_check -compare_types {cell pin pin_attribute} -pin_attrs {direction} -report lib2cdl.rpt $db_lib $db_cdl
```

Appendix A. Document References

Table 22 lists other documents referenced in this document. Contact your Intel Representative to obtain these documents.

Table 22: References

Document title	Type
P1278.3 (Intel 18A) PDK 0.9GA Process Design Kit User Guide	Kit
P1278.3 (Intel 18A) PDK 0.9GA Library Development Kit Overview for Cadence Tools, Release 0.9GA_c	