PDNSim: An Open-source Static PDN Analysis Tool

Vidya A. Chhabria

PDN analysis is crucial to successful IC design closure, particularly for designs implemented in lower technology nodes that suffer from large wire parasitics and high power densities. As described in Chapter 3, PDN analysis involves IR drop estimation and EM current density analysis. These analyses are computationally expensive and traditionally take several hours to perform using traditional EDA tools.

While ML has found some success in addressing this problem [19, 22, 40, 81], the license-based system from today's commercial EDA tool vendors has created a scalability challenge for the adoption of ML techniques as they require tremendous amounts of ground-truth training data. With each PDN simulation taking several hours in industry-scale designs with large power grids with billions of nodes, creating a training dataset with hundreds of datapoints can quickly run into several days with a single license.

This chapter presents PDNSim [39], an open-source static PDN analyzer, developed as a part of the OpenROAD project which creates a fully autonomous, open-source toolchain for digital layout generation with a focus on the RTL-to-GDSII. Unlike commercial EDA tool counterparts that are closed source, PDNSim is open-source and integrated with the OpenROAD application [125] with a permissive BSD 3-clause license. The open-source nature of PDNSim addresses the scalability challenge for ground-truth training data generation as the permissive BSD-3 license allows running multiple simulations of PDN analysis in parallel. Further, while commercial EDA tools typically serve large design houses in taping out chips, PDNSim serves a different audience [189]. It provides accessible source code for PDN-related algorithmic to EDA researchers as it

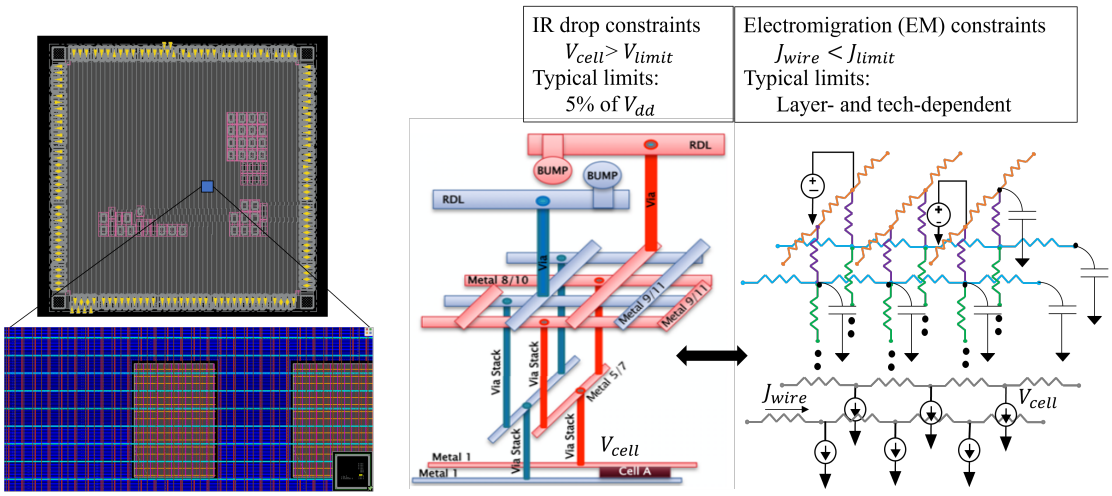

Figure 11.1: PDN analysis: (a) Floorplan of a 14nm RISC-V core; (b) multi-layer power grid that carries voltage from the C4 bumps down to the logic cells; (c) PDN modeled as a RC circuit with voltage sources and current sources.

exposes internal data structures and APIs which are otherwise unavailable in commercial EDA tools. It lowers barriers to entry for non-expert designers with simple push button flows set up as a part of OpenROAD-flow-scripts [125].

PDNSim is an integral part of the power integrity and PDN synthesis tasks of the OpenROAD project and has been used extensively as a part of OpenLane [41] to support the tape-outs done as a part of the Efabless MPW and ChipIgnite [42] programs with the SkyWater 130nm PDK. This chapter describes the software development, capabilities, and usage of PDNSim and also provides several example results across a variety of designs in different technology nodes.

11.1 Problem Statement

The goal of PDN analysis is to estimate the voltage drop between the C4 bumps or power I/O pins and every transistor on the chip. Fig. 11.1(a) shows the top-level layout view of the power grid, which consists of multiple metal layers that connect the C4 bump to the logic gates (Fig. 11.1). The PDN can be modeled as an RC circuit as shown in Fig. 11.1 where the C4 bumps are modeled as voltage sources, logic gates that draw current when they switch as current sources, and PDN wires as parasitics resistances

and capacitances. PDN analysis involves finding the voltage at every node (V_{cell}) on the network and finding the current through every resistor (J_{wire}) in the network.

For static analysis, the capacitances are shorted, and performing PDN analysis amounts to solving the resistor network for voltages at each node. As detailed in chapter 2, algorithmically this involves solving a linear system of equations of the form $GV = J$ where G is the conductance matrix, V is a vector of unknown voltages, and J is a vector of current sources attached to the power grid. In the rest of this chapter, we describe the development of the $GV = J$ system of linear equations in PDNSim and highlight its capabilities as a part of the OpenROAD application.

11.2 PDNSim Software Framework

PDNSim is modularly developed with interfaces to OpenDB (OpenROAD database), and OpenSTA (OpenROAD static timing analysis engine) [190] as shown in Fig. 11.2. OpenDB provides the information on PDN topology including the metal layers in the PDN, the widths of the PDN wires in each layer, locations and dimensions of vias, per-unit resistances, location of instances, etc. and OpenSTA provides the total power per instance including leakage power, internal power, and switching power. PDNSim uses the information from OpenDB and OpenSTA to create the resistor network with voltage sources and current sources as shown in Fig. 11.1(c). It builds the system of equations $GV = J$ and solves it to estimate voltage at each node and current density in each branch of the circuit.
The inputs to PDNSim are the following:

1. A placed design (.def) that has locations of instances and a synthesized PDN
2. The technology library files including (.lef and .lib)
3. Constraints (.sdc) which specify design frequency
4. C4 bump or power pin location file (optional)

PDNSim generates the following outputs:

1. Worstcase and instance-level IR drop report
2. Worstcase and branch-level current density reports

3. PDN connectivity reports which indicate floating PDN stripes and instances and unconnected instances

4. SPICE netlist for the $GV = J$ system of linear equations representing the PDN.

The key routines within PDNSim are highlighted in Fig. 11.2. PDNSim begins by extracting relevant PDN topology information from OpenDB for the net being analyzed. Next, it discretizes the PDN stripes into nodes and creates a resistance network after performing resistance extraction. It then builds the $GV = J$ system of linear equation using modified nodal analysis (MNA), by creating connections between the nodes, inserting the voltage sources (either from the input file or the default bump locations) into the G matrix, and inserting the current sources into the J vector. It then leverages existing sparse matrix libraries to solve the system of equations and obtain the values in V. These values are then processed to obtain the branch current densities. Each of these steps are described below:

Node discretization PDNSim queries OpenDB for all wires of the net being analyzed. It iterates through the wires of the power grid and identifies the different metal layers that are a part of it. Next, for each wire, it creates nodes at all of its via locations. Each via is modeled as two nodes between the two metal layers it connects. In addition to via nodes, PDNSim creates nodes at a finer discretization at the bottom-most metal layer of the power grid (typically the M1/M2 power rails). Fig. 11.3 shows a picture of the OpenROAD GUI which highlighted PDN nodes. The nodes in green, yellow, and maroon (stacked one on top of the other) are the nodes at via locations and the nodes in red are M1 rail nodes.

A finer discretization on the bottom-most layer allows accurate insertion of the current sources based on the location of the instances drawing current. For example,

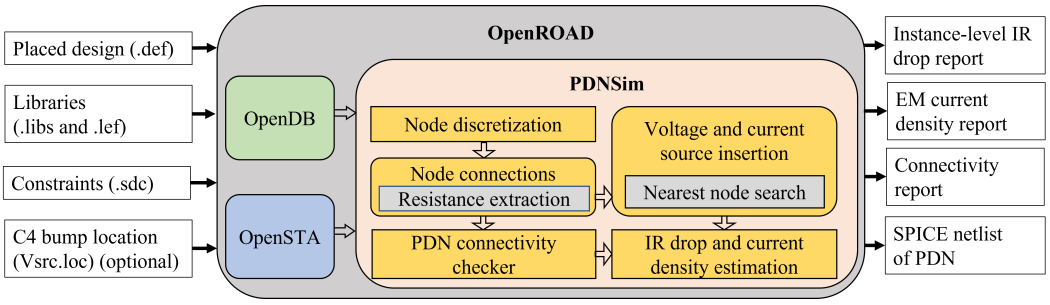

Figure 11.2: PDNSim software framework in the OpenROAD app with interfaces to OpenDB and OpenSTA highlighting the inputs and outputs of PDNSim.

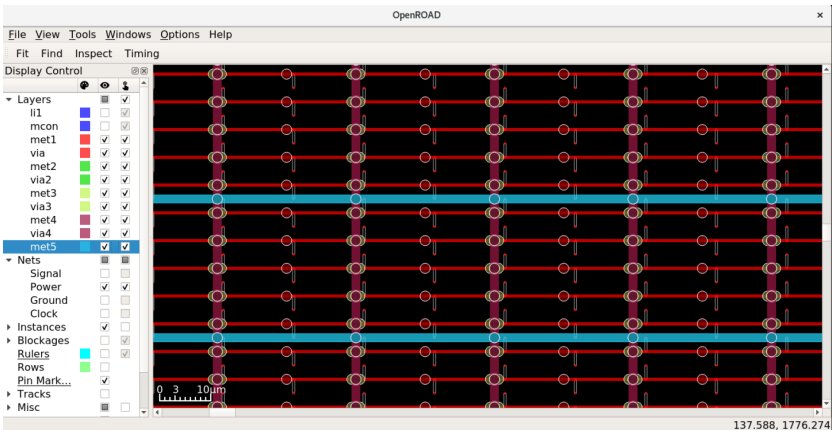

Figure 11.3: PDN nodes visualized in OpenROAD GUI in debug mode highlighting nodes at via locations in the maroon and green stacks and the nodes on the M1 rail at a higher density in red.

Fig. 11.4 shows a region of chameleon design implemented in SkyWater 130nm technology. The picture of the power maps and IR drop maps at two different M1 PDN node pitches. The figures in the top row show the power and IR drop map for a node pitch of 27μm and the bottom row show the power and IR drop map for a node pitch of 3μm. The total number of nodes in the power grid determines the size of the G matrix which in turn determines runtimes and accuracy of the tool, where the larger discretizations provide faster but less accurate solutions as shown in the figure where the IR drop map is obtained at a much higher resolution in the bottom row when compared to the top row (white boxes). The high-resolution IR drop map allows for accurate instance-based IR drop annotation.

Node connections and resistance extraction Once the nodes have been created, PDNSim builds connections between them which involves creating an adjacency matrix-like representation of the power grid model where the nodes of the circuit are a graph and the edges are resistances. As the connections are being made, PDNSim populates the G matrix with the conductance values between the nodes by performing a simple resistance extraction based on the distance between the nodes, metal layer, width of the power wire, and the obtained per-unit and via resistance from OpenDB. The extracted resistance value is then filled into the appropriate indices of the G as per MNA.

Voltage and current source insertion The C4 bumps are modeled as voltage sources as shown in Fig. 11.1(c). These VDD and VSS voltage sources are inserted at C4 bump

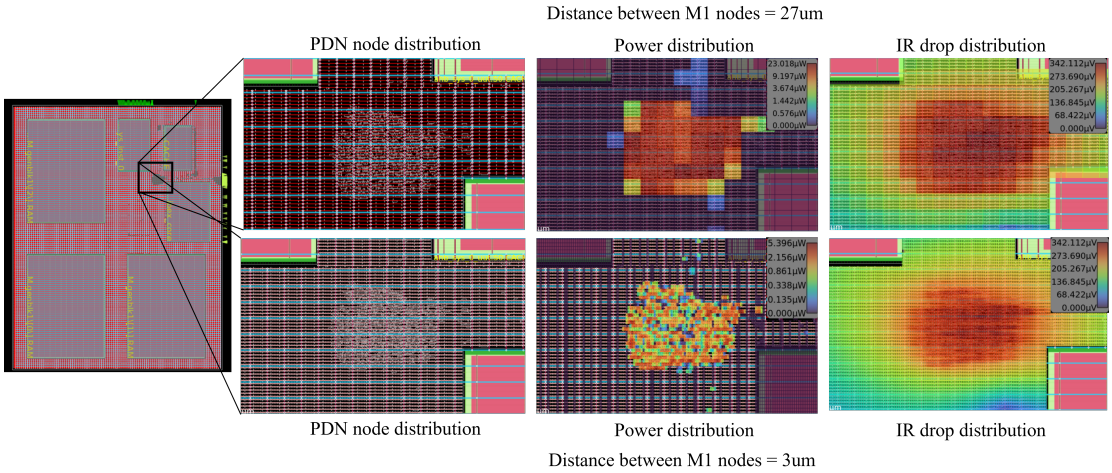

Figure 11.4: Impact of different PDN node discretizations on current source (power map) distributions and IR drop distributions. The top row shows the distributions of current and IR drop for a sparse PDN node pitch (27μm) and the bottom row is for a dense PDN node pitch (3μm) in the bottommost metal layer M1.

locations that are either user-defined in a vsrc file or they default to a checkerboard pattern as highlighted in Fig. 5.10. In either case, the C4 bump is modeled with a default RDL layer that connects the bump to the topmost metal layer of the power grid as shown in the figure. The voltage sources are attached to the nodes on the topmost metal layer of the PDN that is closest to the C4 bump locations. Based on MNA, PDNSim adds an additional row and column in the G matrix for each voltage source and an additional element to the J vector with the value of the voltage source.

Instances are modeled as current sources as shown in Fig. 11.1(c). PDNSim runs OpenSTA [190] under the hood to extract the power per instance in the design. It uses the power values as static current estimates for each instance and adds a current source of the same value to the PDN node that is closest to it. One or more instances may share the same node in which case the currents are accumulated. Therefore, each current source represents the sum of power values of several instances in the vicinity of the node to which it is connected. As previously highlighted in Fig 11.4 the power map resolution depends on the node pitch in M1. Further, large pitches cause several instances to share the PDN node and inaccurately share the same annotated IR drop.

Nearest node search Both the voltage source and current source insertions require finding the nearest PDN node based on the specified C4 bump locations and instance

location respectively. To find the nearest nodes PDNSim uses a runtime-efficient nearest node routine that leverages a near constant-time bounding box-based node query on a hash map-based data structure that stores node pointers. Therefore, the nearest node routine uses the location of the C4 bump or the instance and creates a bounding box of fixed size around it. Based on the bounding box, the query to the hash map data structure returns all nodes within it very rapidly. For example, Fig. 11.5 highlights the nearest node search for the instance current source insertion on the left and the C4 bump voltage source insertion on the right. For the former, the size bounding box is determined by the node pitch in the standard cell rails and for the latter, the bounding box is determined by the PDN stripe pitch in the topmost metal layer. The bounding box query returns the set of nodes highlighted in the white and red bounding boxes respectively. The routine iterates through these nodes, to return the closest node to either the C4 bump or the instance to attach the voltage source or current source. The chosen node is highlighted by the arrows in the figure.

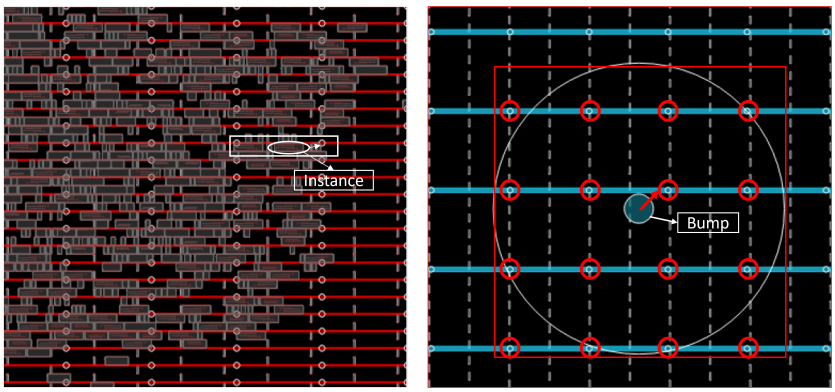

Figure 11.5: Bounding box-based query to find the nearest node to instances (left) and C4 bumps (right).

IR drop and current density estimation Once the $GV = J$ sparse system of linear equations is created using MNA, PDNSim leverages the Eigen library [191] to solve it. The vector V contains the simulated voltages at all the nodes which are annotated back into the node class. Using the location of each node and the annotated voltage drop, PDNSim creates layer-wise IR drop heatmaps. Further, it also outputs instance-level IR drop where all instances connected to the same node share identical IR drop values. Therefore, its important to use finer node discretization for higher instance-level IR drop accuracy (See Fig. 11.4). PDNSim reports the worstcase IR drop and its location

which is useful for correct PDN design.

Using the voltage at every node, PDNSim also performs current density estimation where it uses the resistance value of the branch and the annotated voltage values at the two nodes it connects to find the current through the branch. It reports worstcase current and average current across all wires in the power grid for EM checks. These current estimates can be compared with EM limits to check the reliability of the PDN.

Power grid connectivity checker In addition to estimating the voltage at each node and the current density in each branch of the PDN, PDNSim also checks the connectivity of the power grid. The connectivity checker is based on a simple graph-like traversal of the adjacency matrix starting from one of the PDN nodes that is connected to a voltage source. The traversal checks if all other nodes are reachable through a recursive search. At the end of the traversal, the presence of unreachable nodes implies that there are floating power grid wires or even instances that are unconnected to the power grid. PDNSim flags and reports the locations of floating power wires and instance names that are not connected to the power grid. The connectivity checker allows for catching power supply-related LVS errors early in the physical design cycle.

The connectivity checker is vital to correct PDN synthesis as it flags several instances of missing power stripes within macro channels, missing connections between the macro power grid and the standard cell power grid, and missing connections between the power grid and the VDD and VSS I/O pins. The PDNSim connectivity checker has been more useful in designs that do not have uniform power grids, such as those with macros. Macros typically come with their own power grids that have been independently designed and therefore they act as blockages to standard cell power grid wires in lower metal layers. However, if the macros are on the same power domain as the rest of the standard cells there must be connections between these two independently designed grids through vias/pins in higher metal layers.

Due to blockages and independently designed power grids there may be several instances where regions that lie between blockages (macro channels) fail to have a power stripe for the standard cells that are placed between macros or there may be several instances where the power grid stripes of the macro in the upper metal layers fail to connect to the power grid of the standard cell due to differences in pitches (misalignment). Fig. 11.6 shows an instance of this issues in chameleon design in SkyWater 130nm technology where the power grid checker correctly identified instances (macros) that receive no voltage and locations where the macro power grid is not connected to the standard cell power grid in the upper metal layer.

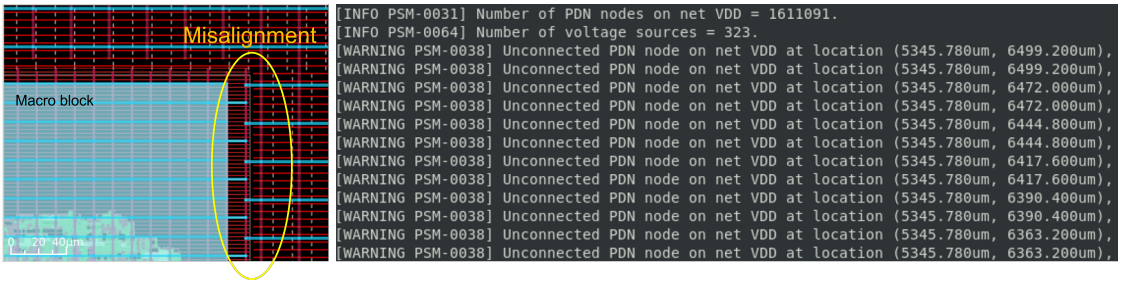

Figure 11.6: An output of PDNSim, flagging misalignment between the higher metal layers of the macro power grid and standard cell power grid. Misalignment on all four sides of the macro results in the macro receiving no voltage.

PDN SPICE netlist generation As a part of PDNSim, we have also developed a SPICE netlist writer. The SPICE netlist represents the $GV = J$ sparse system of linear equations and can directly be simulated using SPICE. This netlist is useful to check the correlation of the Eigen sparse matrix solver within PDNSim with a ground-truth SPICE solver. Further, the netlist is also useful for debugging purposes as it includes the different node names their locations, the resistance values between them, and the current and voltage sources connected to them.

11.3 PDNSim Results

PDNSim [39] is developed in C++17 and has a dependency on the Eigen library [191] for solving $GV = J$ sparse system of linear equations. In this section, we present the PDN analysis results from PDNSim across several designs in an open-source SkyWater 130nm [126] technology and a commercial FinFET 12nm technology node. Fig. 11.7 shows sample reports for the four different outputs of PDNSim, i.e., an IR drop report, a maximum current report, PDN connectivity report, and a snippet of a SPICE netlist. The reports state the worstcase and average IR drop and EM-related current, and throws a warning if the power grid is unconnected with a set of locations for the unconnected PDN nodes. The generated SPICE netlist encodes the node location within the node names, the corresponding resistance values, and associated current and voltage sources as shown in the figure. The commands to generate these reports using the TCL enablement of OpenROAD is presented in Appendix A.

We also compare PDN analysis results from PDNSim against "golden" commercial tool analysis in terms of the number of nodes, runtimes, IR drop distributions, and

Table 11.1: PDNSim (PSM) results compared with golden commercial tool results.

Tech	Design	#Nodes		Total run times (s)		Worstcase IR drop (mV)		Worstcase current (A)	
		PSM	Comm. Tool	PSM	Comm. Tool	PSM	Comm. Tool	PSM	Comm. Tool
GF12	gcd	11,768	7,679	3	7	2.36	1.96	3.31E-4	1.76E-4
	aes	982,086	346,492	121	50	104.12	90.38	1.06E-2	0.73E-2
	jpeg	420,401	214,499	88	39	70.34	74.70	2.09E-2	0.98E-2
	coyote	5,716,123	2,241,225	1148	774	0.98	0.66	3.71E-4	1.68E-4
	swerv_wrapper	2,818,708	1,067,614	903	305	0.56	0.37	2.51E-4	1.13E-4
	bp_single	15,309,805	9,958,335	14,651	8,006	8.67	6.14	1.87E-2	1.04E-2
Sky130HD	gcd	7,601	27,179	3	17	0.89	0.43	1.47E-5	1.05E-5
	aes	115,700	532,994	37	62	1.75	1.07	6.19E-5	2.70E-5
	jpeg	944,237	3,379,623	129	788	0.74	0.49	2.18E-5	1.21E-5
	ibex	485,271	1,714,021	74	333	0.47	0.33	1.40E-5	9.70E-6
	chameleon	289,801	527,693	63	69	0.34	0.26	2.99E-4	1.76E-4
	riscv32i	11,274	31,438	2	19	1.94	1.47	2.65E-3	1.68E-3

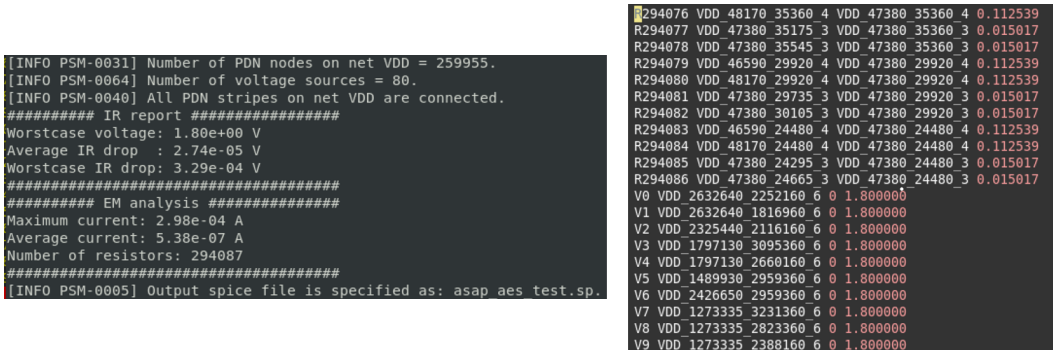

Figure 11.7: Sample IR drop, maximum current reports, and HSPICE netlist snippet.

current density distributions. Table 11.1 shows the results of PDNSim on several designs both with and without macros from OpenROAD-flow-scripts. The table lists the worstcase IR drop, worstcase current for EM checks, and compares the results with results from a commercial tool. It can be seen that PDNSim can perform static PDN analysis on large designs (14M nodes) in times that are comparable to commercial tool runtime. Fig. 11.8 compares the IR drop heatmaps in OpenROAD GUI against IR drop heatmaps from the commercial tools for the jpeg design in the commercial 14nm FinFET technology and open-source SkyWater 130nm. The PDNSim-generated IR drop heatmaps show high fidelity. The differences between commercial tools and PDNSim-reported voltage drops and currents can be attributed to simplified resistance extraction models within PDNSim and differences in OpenSTA-reported power numbers.

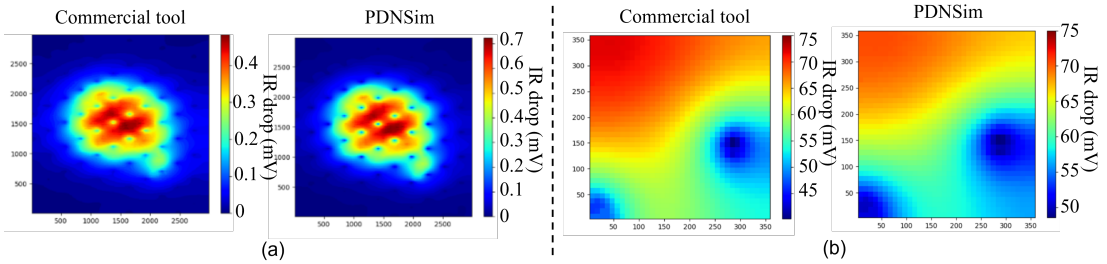

Figure 11.8: Comparison between commercial tool performance and PDNSim on jpeg implemented in: (a) SkyWater 130nm PDK and (b) commercial FinFET 14nm PDK.

More results from PDNSim on different open-source technologies can be found with OpenROAD's nightly regression public gallery [192]. The primary users of PDNSim include the users of OpenLane [41] and OpenROAD-flow-scripts [125] as PDNSim is within the OpenROAD application that these repositories create wrappers around. The users span academia, industry, and government institutions. OpenROAD has had over 14000 clones, over 180 tapeouts in SkyWater 130nm PDK and a mixed signal SOC tapeout in a commercial 12nm FinFET technology with 500K instances and 53 macros.

Appendix A

PDNSim TCL-based Usage

In this appendix, we describe the various TCL commands and arguments needed to generate the different outputs of PDNSim in the OpenROAD application. PDNSim can be downloaded and installed the same way as OpenROAD [125]. It uses a TCL interface to interact with the user.

Listing A.1: PDNSim TCL commands in the OpenROAD application.

```
1 set_pdnsim_net_voltage -net <net_name> -voltage <voltage_value>
2 check_power_grid -net <net_name>
3 analyze_power_grid -net <net_name>
                     [-vsrc <voltage_source_location_file>]
                     [-outfile <filename>]
                     [-enable_em]
                     [-em_outfile <filename>]
                     [-dx]
                     [-dy]
                     [-em_outfile <filename>]
                     [-node_density <node_pitch>]
                     [-node_density_factor <factor>]
4 write_pg_spice  -net <net_name> `
                  -outfile <netlist.sp>
                  [-vsrc <voltage_source_location_file>]
```

OpenROAD has four TCL commands related to PDNSim that are presented in listing A.1. The first TCL command listed on line 1 sets the supply on the power or

ground net specified. It takes two arguments a net name and a voltage value. If this command is not invoked, PDNSim uses a default supply voltage from the liberty value that is available in OpenDB. The second command on line 2 performs the connectivity check on the net specified as an argument. The third command performs power grid analysis and has several arguments as detailed below:

- `net:` (mandatory) is the name of the net to analyze, power or ground net name.
- `vsrc:` (optional) file to set the location of the power C4 bumps/IO pins.
- `dx,dy`: (optional) these arguments set the bump pitch to decide the voltage source location in the absence of a vsrc file. Default bump pitch of 140um used in absence of these arguments and vsrc.
- `enable_em:` (optional) is the flag to report current per power grid segment. outfile: (optional) filename specified per-instance voltage written into file.
- `em_outfile:` (optional) filename to write out the per segment current values into a file, can be specified only if `enable_em` is flag exists.
- `node_density:` (optional) This value can be specified by the user in um to determine the node density on the std. cell rails. Cannot be used together with `node_density_factor`.
- `node_density_factor:` (optional) Integer value factor which is multiplied by standard cell height to determine the node density on the std. cell rails. Cannot be used together with `node_density`. Default value is 5.

The fourth command on line 4 of Listing A.1 creates a SPICE netlist representation of the power grid ($GV = J$ system of equations). The arguments include the net name whose netlist must be extracted, the name of the output netlist file (`outfile`), and the optional `vsrc` file with the locations of the C4 bumps or I/O power pins. The description of all these commands can also be found in PDNSim readme as a part of the OpenROAD GitHub repository [39].