The Bedrock of Silicon: A Comprehensive Guide to Libraries and Data Inputs in VLSI Physical Design

The Anatomy of a Digital Design Foundation

Introduction to the Physical Design Flow: From Netlist to GDSII

The creation of a modern integrated circuit (IC) is a monumental feat of engineering, transforming an abstract logical idea into a tangible silicon chip containing billions of transistors. The final stage of this transformation is known as physical design, a process that converts a logical description of a circuit, typically in the form of a gate-level netlist, into a physical layout.[1] This layout, ultimately represented in a format like GDSII, is the manufacturing blueprint that dictates the precise position of every component and the intricate web of metallic routes that interconnect them.[1]

This journey from netlist to GDSII is far more than a simple geometric mapping. It is a complex, multi-objective optimization problem governed by three competing metrics: Performance, Power, and Area (PPA).[2] The goal is to produce the smallest possible chip (minimum Area) that runs at the required speed (maximum Performance) while consuming the least amount of energy (minimum Power). Achieving this balance requires a sophisticated set of Electronic Design Automation (EDA) tools and, critically, a rich and unambiguous set of input data that defines every aspect of the design, from the components used to the rules of the manufacturing process.

The Three Pillars of Input Data: A High-Level Overview

To begin any physical design project, EDA tools require three fundamental categories of information. These pillars form the complete context necessary to translate logic into a physical reality, answering the questions of *what* to build, *how well* it must perform, and *what it is built from*.[3]

The Logical Blueprint: The Gate-Level Netlist (.v,.vhd)

The primary input that defines the circuit's functionality is the gate-level netlist.[3] Generated by a logic synthesis tool from a higher-level Register-Transfer Level (RTL) description (written in Verilog or VHDL), the netlist is the structural blueprint of the design.[4] It contains no physical information; it is a purely logical description of components and their connectivity.

A netlist consists of:

- **Component Instances:** Instantiations of pre-defined building blocks, such as standard cells (e.g., AND2, DFF), larger macros (e.g., SRAMs, PLLs), and I/O pads.[4] Each instance is given a unique name.
- **Logical Connectivity:** A description of the "wires" or "nets" that connect the ports (pins) of these instances, defining the flow of signals through the circuit.[4]

For example, a simple Verilog netlist for a half-adder might look like this [4]:

Verilog

```
module half_adder (C, S, A, B);
 input A, B;
 output C, S;

 AND2 U1 (.Y(C),.A(A),.B(B));
 EXOR U2 (.Y(S),.I1(A),.I2(B));
endmodule
```

In this example, half_adder is the design module. A and B are input ports, while C and S are output ports. U1 and U2 are unique instance names for the standard cells AND2 (an AND gate) and EXOR (an XOR gate), respectively. The connectivity defines that the output C is driven by

the Y pin of instance U1, whose inputs A and B are connected to the primary inputs of the module.

The Performance Contract: Design Constraints

While the netlist describes the circuit's structure, it says nothing about its required performance. This is the role of design constraints, which form a contract that the final physical layout must satisfy.[4] These constraints guide every decision made by the EDA tools during optimization. The industry-standard format for conveying these constraints is the Synopsys Design Constraints (SDC) file.[4]

Constraints can be broadly categorized as:

- **Timing Constraints:** These define the performance targets. They include clock definitions (period, waveform), I/O timing requirements (input and output delays relative to a clock), and timing exceptions like false paths (paths that are logically impossible and should be ignored by the timing analyzer).[4]
- **Optimization Constraints:** These are design rule constraints (DRCs) that must be met for electrical integrity, such as maximum signal transition time (slew), maximum fanout, and maximum capacitance on any given net.[4]
- **Physical Constraints:** These provide initial guidance for the physical layout, such as floorplanning guidelines, I/O pin locations, and power grid specifications.[1]

The Component Catalog: The Role of Libraries

The netlist is merely a collection of names like AND2 and EXOR. For an EDA tool to understand what these components are, how they behave, and what they look like physically, it needs access to a comprehensive set of libraries.[3] These libraries are the component catalog, providing detailed characterization data for every single element instantiated in the netlist.

Crucially, each component requires multiple representations, or "views," tailored for different tasks within the physical design flow.[8] A placement tool needs a physical view to understand a cell's size and pin locations, while a timing analysis tool needs a timing view to calculate signal delays through it. These distinct views are stored in different library files, which form the core subject of this report. Without these libraries, the netlist is an un-interpretable document, and the constraints are an unenforceable contract.

Defining the Physics - Technology and Interconnect Data

Before any cells can be placed or routed, the EDA tool must first understand the fundamental "laws of physics" for the specific semiconductor manufacturing process being used. This information is provided by the foundry (the fabrication plant) in a set of files that define the properties of the silicon wafer, the metal layers, and the rules for their construction. These files form the foundational canvas upon which the entire design will be painted.

The Foundry's Rulebook: The Technology File (.tf,.tech.lef)

The technology file is the master rulebook provided by the foundry that contains all process-specific information required for the physical layout.[1] It defines the manufacturing constraints and the physical and electrical characteristics of all the available layers. Different EDA vendors use slightly different formats, with the most common being the

.tf file for the Synopsys tool ecosystem and the technology LEF (.tech.lef) for the Cadence ecosystem.[4]

Regardless of the format, the contents are conceptually similar and include:

- **Layer Information:** A detailed definition of every layer in the process stack, from the base silicon layers up to the top-most metal layer. For each layer, it specifies its name, number, type (e.g., ROUTING for metal, CUT for a via, MASTERSLICE for a base layer), preferred routing direction (horizontal or vertical), pitch, minimum width, thickness, and electrical properties like sheet resistance (Rsh) and capacitance per unit area (Carea).[4]
- **Design Rules:** A comprehensive set of manufacturing rules that must not be violated. This includes minimum spacing between wires on the same layer, minimum area for any polygon, rules for via construction, and complex rules for metal density and antenna effects, which prevent manufacturing defects.[4]
- **Manufacturing Grid:** A definition of the finest possible grid for the manufacturing process (e.g., MANUFACTURINGGRID 0.005 for a 5nm grid). All geometric shapes in the final layout must have coordinates that snap to this grid.[11]
- **Site Definition:** A crucial definition that specifies the basic building block for standard cell placement. A SITE definition (e.g., SITE CORE_7H) defines the height of the standard cell rows and their vertical placement grid. It also specifies symmetry properties (e.g.,

SYMMETRY Y), which allows cells to be flipped vertically without violating design rules.[11]

The separation of the technology file from the files describing the cells themselves is a critical abstraction. The technology file defines the "canvas and paint" (layers, rules, grid), while the cell libraries (discussed in Section 4) define the "stamps" (cell layouts) that can be placed on that canvas. This modularity allows a foundry to provide a single technology file for a process, which can then be used by multiple different vendors to create compatible cell libraries.

Procedurally, the technology file is the very first piece of physical information that must be loaded into the EDA tool. It establishes the foundational context—defining the layers, rules, and sites—that all other physical library files will reference.[4] Attempting to load a cell library before the technology file would be like trying to read a map without the legend; the symbols would be meaningless.

Modeling the Wires: Interconnect Parasitic Data

In early semiconductor technologies, the delay of a signal was dominated by the switching time of the transistors within the logic gates. However, as process nodes have shrunk into the deep sub-micron realm (28nm and below), the physical dimensions of the wires (interconnect) have become so small that their parasitic resistance (R) and capacitance (C) now contribute significantly to the total path delay—often accounting for 50% or more.[13] Consequently, accurately modeling these interconnect parasitics is paramount for achieving timing closure.

The complexity of these parasitic effects has grown with each technology node. Simple, uniform RC-per-unit-length models are no longer sufficient. At advanced nodes, the capacitance of a wire is heavily influenced by its proximity to neighboring wires (crosstalk capacitance), the wires on the layers above and below it, and even the local density of metal fill shapes inserted for manufacturability. This has led to the development of more sophisticated modeling files.

Interconnect Technology Format (ITF)

The Interconnect Technology Format (ITF) is a detailed, technology-dependent ASCII file provided by the foundry.[4] It provides a precise model of the cross-sectional profile of the process, describing the thickness, physical attributes, and dielectric properties of all conductor and dielectric layers.[14] This file serves as the source data for parasitic extraction tools, enabling them to perform accurate analysis for timing, signal integrity, power, and

reliability.[14]

Table Look-Up Plus (TLU+)

While ITF provides the raw data, parsing it and running complex field-solver calculations on-the-fly during place and route would be computationally prohibitive. To address this, the ITF data is pre-processed to generate a more efficient format called TLU+.[4]

A TLU+ file is a binary, highly efficient lookup table that stores pre-computed RC coefficients.[4] Instead of calculating parasitics from first principles, the EDA tool's extraction engine can quickly look up the appropriate RC values from these tables based on the local geometric context of a wire segment, including its width, its spacing to neighbors, the local metal density, and even temperature effects.[4] This approach provides the accuracy of a detailed physical model with the speed required for interactive physical design.

In practice, a P&R tool is typically provided with a set of TLU+ files representing different process corners (e.g., a min set for best-case C and a max set for worst-case C) and a map file. The map file is a simple text file that correlates the layer and via names used in the main technology file (.tf) with the potentially different names used in the ITF and TLU+ files, ensuring consistency across the entire toolchain.[4] If TLU+ files are not available, the tool may fall back to using the ITF directly, though this generally results in longer runtimes and less accurate extraction.[13]

The Building Blocks - A Deep Dive into Standard Cells

With the physical rules of the foundry established, the next step is to understand the components used to build the design. For most digital application-specific integrated circuits (ASICs), these components are standard cells.

From Transistors to Gates: The Standard Cell Concept

A standard cell is a group of transistors and interconnects that implements a basic boolean logic function (e.g., AND, OR, XOR, inverter) or a storage function (e.g., flip-flop, latch).[8] Each

cell is a piece of full-custom, highly optimized intellectual property (IP). The key characteristic of the standard cell methodology is that all cells in a given library share a fixed height but have a variable width depending on their complexity.[8] This uniform height is what enables EDA tools to automatically place them into clean, organized rows, forming the foundation of modern automated digital layout.[8] A standard cell library is a collection of hundreds or even thousands of these pre-designed and pre-characterized cells, providing the fundamental vocabulary for digital design.[8]

The Standard Cell Taxonomy: Optimizing for PPA

A standard cell library is not merely a collection of unique logic gates; it is a meticulously curated "optimization toolkit." To give synthesis and place-and-route tools the flexibility to meet stringent PPA targets, libraries provide multiple versions of the same logic function, each with different characteristics. This variety is the key to automated PPA optimization. If speed were the only goal, all cells would be designed for maximum performance. If area were the only goal, all would be designed for minimum size. The existence of a wide variety of cells proves that physical design is an exercise in managing trade-offs, and the tool's job is to select the optimal cell for each specific instance in the netlist to meet the global design goals.

Classification by Density (Track Height)

One of the primary ways cells are differentiated is by their height, which is measured in "tracks." A track corresponds to the pitch of the second metal layer (M2), representing the number of horizontal routing wires that can pass over the cell within its boundary.[4] The track height directly impacts the trade-off between routability, performance, and area.

Common library variants include [16]:

- **High Performance (HP):** Taller cells (e.g., 12-track) that can accommodate larger transistors. These cells offer the best performance (highest drive strength, lowest delay) but consume more area and power.
- **High Density (HD):** Medium-height cells (e.g., 9- or 10-track) that provide a balance between performance and area. These are often the workhorse libraries for many designs.
- **Ultra-High Density (UHD):** Shorter cells (e.g., 7- or 8-track) with smaller transistors, designed for maximum placement density and low power consumption, at the cost of reduced performance.

Classification by Performance (Threshold Voltage - Vth)

Within a library of a specific track height, cells are further subdivided by their threshold voltage (Vth), which is the gate voltage required to turn a transistor "on." The Vth has a direct and significant impact on both the speed and the leakage power of a transistor.[4] A lower

Vth allows a transistor to switch on faster (improving performance) but also allows more current to "leak" through when it is supposed to be "off" (increasing static power consumption).

Most modern libraries provide three main Vth variants [16]:

- **Low-Vth (LVT):** These cells have the lowest threshold voltage, making them the fastest available. However, they also suffer from the highest sub-threshold leakage current.
- **Standard/Regular-Vth (SVT/RVT):** These cells offer a baseline balance between performance and leakage.
- **High-Vth (HVT):** These cells have the highest threshold voltage. They are slower than SVT and LVT cells but have significantly lower leakage power.

During optimization, EDA tools leverage this variety strategically. On timing-critical paths where every picosecond matters, the tool will use fast LVT or SVT cells to meet setup time requirements. On paths with ample timing slack, the tool can swap in slower but more power-efficient HVT cells to reduce the overall static power consumption of the chip without impacting its performance.[4]

The Extended Family: Specialized Cells

Beyond the logic and sequential cells, a standard cell library must also contain a variety of non-logic, physical-only cells that are essential for building a functional, manufacturable, and robust chip.[16]

- **Filler Cells:** After placement, there will be gaps in the standard cell rows. Filler cells are inserted into these empty spaces to ensure the continuity of the N-well and diffusion layers across the row, which is a requirement for many manufacturing processes. They also help meet local polygon density rules.[16]
- **Tap Cells (Well Taps):** These cells provide a low-resistance connection for the N-well to VDD and the P-substrate to VSS (Ground). Placing tap cells at regular intervals is critical

to prevent a parasitic condition known as "latch-up," where a short circuit can form between the power rails.[16]

- **Decoupling Capacitors (Decaps):** These cells are essentially on-chip capacitors built using transistor gates, placed between VDD and VSS. They act as small, local charge reservoirs distributed across the chip. When a large number of transistors switch simultaneously, they create a sudden demand for current, which can cause the local power supply voltage to droop. Decaps supply this instantaneous current, helping to stabilize the power grid.[16]
- **End Cap Cells:** These are special cells placed at the beginning and end of each standard cell row to properly terminate the well and diffusion layers at the row boundaries.[16]
- **Tie-High/Tie-Low Cells:** These cells provide a robust and electrically clean way to connect the input pins of other cells to a constant logic '1' (VDD) or logic '0' (VSS). Using a dedicated tie cell is more reliable than connecting directly to the main power rails.
- **Clock Cells:** The clock network is the most critical signal network in a digital design. Libraries provide special buffers and inverters that are specifically characterized for use in the clock tree. These cells are designed to have symmetrical, balanced rise and fall times to minimize clock duty cycle distortion.[16]

The Two Faces of a Cell - Physical and Timing Library Views

A fundamental concept in modern physical design is abstraction. To manage the immense complexity of a billion-transistor design, EDA tools do not work with the full, transistor-level layout of every cell. Instead, they use simplified, abstract "views." This principle is most evident in the separation of a cell's physical representation from its timing and power representation. This separation of concerns is a classic engineering trade-off: it allows the geometric engines of the P&R tool (placer, router) to operate on simple, lightweight physical data, while the complex analysis engines (timer, power analyzer) work with detailed performance models. This decoupling is essential for enabling EDA tools to handle massive designs in a reasonable amount of time and with a manageable memory footprint.

The Physical Abstract: Library Exchange Format (LEF)

The Library Exchange Format (LEF) is an open, ASCII-based specification used to describe

the physical characteristics of standard cells, macros, and IP blocks.[17] It provides an

abstract view of the cell, containing only the information needed by a place-and-route tool to perform its tasks correctly. It does not include the internal details of the cell's transistor layout, making the file size small and efficient to parse.[17]

Structure: Technology LEF vs. Cell LEF

As introduced in Section 2, the LEF specification is often split into two files for modularity [19]:

1. **Technology LEF:** This file contains the generic process information, such as layer definitions, routing and via design rules, and placement site definitions. It is conceptually similar to the .tf file and is typically read first.[11]
2. **Cell LEF:** This file contains the abstract physical descriptions for every individual standard cell and macro in the library.[19]

Key Statements Explained

A typical entry for a standard cell in a Cell LEF file contains several key statements that define its physical properties. Consider the following conceptual example for a NAND gate [4]:

```
MACRO NAND_1
  CLASS CORE ;
  FOREIGN NAND_1 0.0 0.0 ;
  ORIGIN 0.0 0.0 ;
  SIZE 4.5 BY 12.0 ;
  SYMMETRY X Y ;
  SITE core ;
  PIN A
    DIRECTION INPUT ;
    PORT
      LAYER m1 ;
        RECT 0.5 2.0 1.0 2.5 ;
```

```
    END
  END A
  PIN B
    ...
  END B
  PIN Y
    DIRECTION OUTPUT ;
    PORT
      LAYER m1 ;
        RECT 4.0 5.0 4.5 5.5 ;
    END
  END Y
  OBS
    LAYER m1 ;
      RECT 0.0 0.0 4.5 12.0 ;
  END OBS
END NAND_1
```

- MACRO <cell_name>: This is the top-level statement that begins the definition of a cell.[4]
- CLASS: Defines the type of cell. CORE is used for standard cells intended for placement in rows, while PAD would be used for I/O cells.[20]
- SIZE: Specifies the width and height of the cell's placement boundary in microns. This is one of the most critical pieces of information for the placement tool.[4]
- SITE: Names the placement site from the Technology LEF that this cell is designed to fit into, ensuring it aligns correctly within a row.[4]
- PIN <pin_name>: Begins the definition for each logical pin of the cell.[4]
    - DIRECTION: Specifies whether the pin is an INPUT, OUTPUT, or INOUT.[4]
    - PORT: This is a crucial subsection that defines the physical manifestation of the pin. It contains one or more geometric shapes (e.g., RECT on a specific LAYER) that represent the physical metal polygons where a router can connect a wire to access this logical pin.[4]
- OBS (Obstruction): Defines "keep-out" regions within the cell's boundary. The OBS statement tells the router where it is *not* allowed to place wires, typically to prevent shorting over the cell's internal routing.[19]

The Performance Abstract: Liberty Timing Format (.lib)

While LEF describes a cell's physical body, the Liberty Timing Format (.lib) file describes its soul: its performance, power consumption, and electrical behavior.[4] Liberty is an

industry-standard, ASCII-based format that contains all the characterization data needed by synthesis, timing analysis, and power analysis tools.[12] This data is not derived from simple formulas; it is the result of running thousands of highly accurate SPICE simulations on the cell's full transistor-level netlist under a wide variety of operating conditions.[12]

Characterization and PVT Corners

The performance of a transistor is highly sensitive to variations in the manufacturing **P**rocess, the operating **V**oltage, and the junction **T**emperature (PVT).[22] A transistor from a "fast" corner of the process will switch more quickly than one from a "slow" corner. Similarly, a higher supply voltage and lower temperature generally lead to faster performance.

To ensure a chip will function correctly across all expected operating conditions, designs are analyzed at multiple PVT corners. A "corner" represents a specific combination of these three variables chosen to model an extreme of behavior.[23] For example:

- **Worst-Case Slow (WCS):** Slow process, low voltage, high temperature. This corner is used for setup timing analysis to ensure the chip can meet its clock frequency target under the slowest possible conditions.
- **Best-Case Fast (BCF):** Fast process, high voltage, low temperature. This corner is used for hold timing analysis to check for race conditions under the fastest possible conditions.

Because the cell's behavior is different at each corner, a separate and complete .lib file is generated for each PVT corner that needs to be analyzed.[9] An EDA tool performing Multi-Mode Multi-Corner (MMMC) analysis will load all of these

.lib files simultaneously to verify the design across the entire operating space.[9]

Key Groups and Syntax Explained

A Liberty file is structured into hierarchical blocks called "groups." A simplified entry for an AND gate might look like this [4]:

Code snippet

```
library(my_lib) {
  time_unit : "1ns";
  voltage_unit : "1V";
  operating_conditions(WCCOM) {
    process : 1.0;
    temperature : 125;
    voltage : 0.9;
  }

  cell(AND2_X1) {
    area: 8.000;
    function: "(A & B)";

    pin(A) {
      direction: input;
      capacitance: 0.005;
    }
    pin(B) {... }

    pin(Y) {
      direction: output;
      max_capacitance: 0.1;

      timing() {
        related_pin: "A";
        cell_rise(delay_template_7x7) {
          index_1("0.01,..., 0.2");  /* input transition */
          index_2("0.005,..., 0.1"); /* output capacitance */
          values(... );
        }
        rise_transition(slew_template_7x7) {... }
      }
      timing() {
        related_pin: "B";
       ...
      }
    }
  }
}
```

- library block: This top-level group defines global attributes that apply to all cells, such as

the time_unit and voltage_unit, and defines the specific operating_conditions (PVT corner) for which this file was characterized.[16]

- cell block: Contains all the information for a single cell. This includes its area (used by synthesis for area optimization) and its Boolean function (used by synthesis and verification tools).[4]
- pin block: Defines the characteristics of each pin, such as its direction and its input capacitance (the load it presents to a driving cell).[4]
- timing block: This is the most critical part of the .lib file. It defines a "timing arc" from a specific input (related_pin) to an output. Inside this block are the multi-dimensional lookup tables (LUTs) that model the cell's behavior.[12]
    - **Delay Tables (cell_rise, cell_fall):** These LUTs provide the propagation delay through the cell. The table is indexed by the two primary factors that affect delay: the transition time (slew) of the input signal and the total capacitive load on the output pin.[4]
    - **Output Slew Tables (rise_transition, fall_transition):** These LUTs predict the transition time of the output signal, again indexed by input slew and output load. The output slew of one cell becomes the input slew for the next cell in the path, allowing for accurate path delay calculation.[4]
    - **Power Tables (rise_power, fall_power):** Similar LUTs are included to model the dynamic power consumed by the cell during a switching event.
    - For sequential cells like flip-flops, the timing block also contains LUTs for timing checks like setup and hold times, which are also dependent on the slew of the clock and data signals.[12]

Delay Models: NLDM vs. CCS

Over time, the models used to represent delay within the Liberty format have evolved to maintain accuracy at advanced nodes.

- **NLDM (Non-Linear Delay Model):** The traditional model, which uses the 2D lookup tables described above. NLDM models the cell's driver as a simple voltage source and assumes the output load is a single, lumped capacitor. This is reasonably accurate for older technologies but breaks down at smaller nodes where the resistive component of the interconnect becomes significant.[25]
- **CCS (Composite Current Source):** A more advanced and accurate model that characterizes the cell's output driver as a time-varying current source rather than a voltage source. CCS is better at modeling the effects of high-resistance interconnects and other complex behaviors seen in deep sub-micron processes. It provides higher accuracy but at the cost of significantly larger .lib files due to the more complex data

being stored.[25]

The following table provides a concise comparison of the LEF and LIB file formats, crystallizing the concept of the dual-abstract model.

Aspect	LEF (.lef)	LIB (.lib)
Primary Purpose	Physical Abstract View	Performance Abstract View
Key Contents	Cell dimensions (size), pin locations and shapes, routing blockages, layer rules, site definitions.[4]	Timing arcs (delay, slew), power consumption, input capacitance, setup/hold constraints, logical function.[4]
Generated From	Abstracted from the cell's full GDSII layout.	Characterized from extensive SPICE simulations of the cell's transistor netlist.[22]
Used By	Placement, Routing, Floorplanning, and Physical Verification tools.[9]	Synthesis, Static Timing Analysis (STA), Power Analysis, and Optimization engines.[9]
Analogy	The cell's **physical footprint** or architectural blueprint.	The cell's **performance datasheet** or electrical specification.

The Evolving Blueprint - Representing the Physical Layout

While LEF files describe the library of available components and LIB files describe their performance, a different format is needed to describe the *actual design* as it is being implemented by the place-and-route tool. This format must capture which library cells are used, where they are placed, and how they are routed.

The Design Snapshot: Design Exchange Format (DEF)

The Design Exchange Format (DEF) is an open, ASCII-based specification that represents the complete physical state of a design at any point during the P&R flow.[27] It serves as a "snapshot" of the layout, effectively combining the logical connectivity from the netlist with the physical data generated by the EDA tools.[7]

The relationship between LEF and DEF can be understood through an analogy: the LEF file is like a library of fonts, defining the physical shape and properties of each available character (cell). The DEF file is the document itself, specifying which character (cell instance) is placed at which coordinate on the page (the chip die). To properly view or edit the document (the DEF file), one must have access to the font library (the LEF file) it references.

DEF files are used ubiquitously in the physical design flow to save the state of the design after each major stage (e.g., design.fp.def after floorplanning, design.place.def after placement, design.cts.def after clock tree synthesis, design.route.def after routing). They are also the primary mechanism for transferring a design between different tools in a flow, for example, from a floorplanning tool to a placement tool.[28]

Key Sections Explained

A DEF file is organized into several distinct sections, each describing a specific aspect of the physical layout. The key sections include:

- DIEAREA: Defines the overall dimensions of the chip, specifying the coordinates of the bounding box for the core and die area.[4]
- ROWS: Defines the horizontal placement rows for the standard cells. Each row definition specifies a name, the SITE it is built from (referencing the site definition in the technology file/LEF), its starting coordinate, and its orientation.[4]
- COMPONENTS: This is the heart of the placement information in a DEF file. This section lists every single standard cell and macro instance from the netlist. For each component, it specifies its unique instance name, the library cell it corresponds to (<modelName>), its placement status (+ PLACED), its precise (x y) coordinate, and its orientation (e.g., N for North, FS for Flipped South).[4]
- PINS: Defines the physical properties of the design's external I/O pins, including their name, location, layer, and geometry.[4]
- SPECIALNETS: This section is used to describe the routing of special nets, most

commonly the power and ground (VDD and VSS) grid. Power grid synthesis is typically performed early in the flow, and this section captures the geometry of the power rings and straps.[4]
- NETS: After the router has run, this section is populated with the detailed geometric description of the signal nets. For each net, it lists the pins it connects and then describes the exact path of the wires and vias across multiple metal layers that form the connection.[4]
- BLOCKAGES: Defines placement or routing "keep-out" zones within the floorplan where the tool is not allowed to place cells or routes, respectively.[4]

As the design progresses through the P&R flow, the DEF file becomes progressively more complete. An initial floorplan DEF might only contain DIEAREA, ROWS, and PINS. After placement, the COMPONENTS section is fully populated. After routing, the NETS section is filled in, resulting in a complete description of the final layout.

EDA-Specific Ecosystems - The Synopsys Data Model

While open ASCII formats like LEF, DEF, and LIB are essential for interoperability, they are not optimized for the high-performance demands of modern EDA tools. For speed and memory efficiency, major EDA vendors ingest these standard files and compile them into a proprietary, unified, binary database format. This internal data model becomes the single source of truth that all of the tool's engines—placer, router, timer, extractor—operate on. This section focuses on the evolution and structure of the data model within the Synopsys ecosystem.

From Files to a Database: Milkyway (ICC) and NDM (ICC2)

The shift from ASCII files to an integrated binary database is a direct response to the data management crisis posed by Moore's Law. A file-based system struggles with the interdependencies and sheer scale of designs at 10nm, 7nm, and below. A unified database architecture enables faster data access, a lower memory footprint, and, most importantly, tighter correlation between different analysis engines that all operate on the same consistent data.

- **Milkyway:** For many years, the standard database format for Synopsys tools like Design Compiler (DC) and IC Compiler (ICC) was Milkyway.[4] A Milkyway library is a directory structure containing binary files that store both logical and physical information. It uses different "views" for different purposes; for example, the

CEL view contains the full layout information, while the FRAM view is a lightweight physical abstract (conceptually similar to LEF) used for place and route.[4]

- **NDM (New Data Model):** With the introduction of IC Compiler II (ICC2), Synopsys rolled out a completely redesigned database called the New Data Model (NDM).[4] NDM was architected from the ground up to provide the scalability and capacity needed for today's giga-scale designs, capable of handling over 500 million instances in a single, compact data model.[32] An NDM library is a single, unified database that is created by merging all the necessary source files—the logical libraries (.lib), physical libraries (.lef), and the technology file (.tf)—using a dedicated tool called the ICC2 Library Manager.[31] Once the NDM is created, the original source files are no longer needed for the P&R flow.

Understanding Library Types in the Tool Environment

Once the libraries are loaded into the tool's database, they are assigned different roles that define how they are used during the design process. Understanding these roles is critical for correctly setting up a synthesis or P&R run.

Reference Library vs. Design Library

This distinction separates the immutable, vendor-provided components from the specific design being worked on.

- **Reference Library:** A read-only library containing the pre-designed, pre-characterized components like standard cells, macros, and I/O pads. These are the building blocks provided by a library vendor or foundry.[4] In the Synopsys context, these are the NDM (.ndm) or Milkyway (mw_reference_library) databases containing the cell abstracts.[29]
- **Design Library:** The active, read/write library that contains the specific chip design being implemented. The design library does not contain the cell definitions themselves; instead, it *references* the cells from the reference libraries. It holds the top-level netlist and all the data generated during the P&R process, such as the placement coordinates and routing geometries for every cell instance.[4]

Link Library vs. Target Library (Logical Libraries)

This distinction, applied to the logical libraries (.db format, which is a compiled .lib), is a powerful mechanism for controlling the optimization process.

- **Link Library (link_library):** This is a comprehensive list of all logical libraries that the tool needs to *resolve all cell references* found in the input netlist. The tool searches through the libraries in this list to find a definition for every single instance, whether it's a simple NAND gate, a complex SRAM, or an I/O pad. If a cell instantiated in the netlist cannot be found in any of the link libraries, the tool will issue an "unresolved reference" error.[35]
- **Target Library (target_library):** This is a *subset* of the link library. The target library specifies the set of cells from which the tool is *allowed to choose new cells* during optimization. For example, when the tool needs to insert a buffer to fix a timing violation, it will only choose a buffer from a library listed in target_library. This provides a crucial control mechanism. Typically, the target_library will contain the standard cell libraries, but it will *exclude* macros like memories and PLLs. This tells the tool: "You need to understand what this memory is and connect to it (via the link_library), but you are forbidden from ever modifying, replacing, or creating a new one (because it's not in the target_library)." This is fundamental for protecting IP blocks and managing hierarchical design flows.[35]

The create_lib Command: Assembling the Design Environment in ICC2

In the modern ICC2 flow, the create_lib command is the primary Tcl command used to create a new design library and establish the entire physical design context.[38]

The typical syntax is as follows [39]:

create_lib <design_lib_name> -technology <tech_file.tf> -ref_libs {<list_of_ndm_ref_libs>}
The process this command encapsulates is:

1. **Prerequisite:** A one-time, offline process must be completed using the ICC2 Library Manager to convert the vendor-provided ASCII files (.tf, .lef, .lib) into the proprietary NDM reference library (.ndm) format.[31]
2. **Execution:** The create_lib command is run within the main icc2_shell.
3. It creates a new directory on disk, which will serve as the design library for the chip being implemented.
4. The -technology option associates this new library with the foundational technology file, loading all the layer and design rule information.
5. The -ref_libs option provides a list of all the NDM reference libraries that this design will use. This command links the design library to the cell catalog, making all the standard

cells and macros available for instantiation and analysis.

After create_lib is successfully executed, the EDA tool environment is fully configured. It understands the manufacturing process, knows the complete catalog of available parts, and has a workspace ready. The next step is to load the design-specific netlist and constraints.

Advanced Design Specifications

Beyond the fundamental libraries and timing constraints, modern System-on-Chip (SoC) designs require additional, high-level specifications to define complex power architectures and clocking strategies. These are typically captured in dedicated constraint files.

Defining Power Intent: The Unified Power Format (UPF)

As power consumption has become a first-order design concern, techniques like power gating (shutting off power to idle blocks) and multi-voltage domains have become standard practice. Describing these complex power architectures requires a specialized format. The Unified Power Format (UPF) is the IEEE standard for specifying a design's power intent.[41]

The purpose of UPF is to describe the power architecture in a portable, tool-independent manner. It allows a designer to define [41]:

- **Power Domains:** Regions of the chip that can be independently powered up or down.
- **Supply Nets and Ports:** The definition of all power and ground supplies in the design.
- **Power Switches:** The header or footer cells that act as switches to gate power to a domain.
- **Isolation Strategies:** The insertion of special isolation cells on signals that cross from a power domain that might be turned off to one that is always on. This prevents corrupted signals from propagating out of a powered-down block.
- **Level Shifters:** The insertion of level-shifter cells on signals that cross between domains operating at different voltage levels.
- **State Retention:** The specification of which registers need to retain their state even when their power domain is shut off, requiring the use of special state-retention flip-flops.

By capturing this intent in a UPF file, the P&R tool can automatically implement the entire low-power infrastructure, inserting the correct power switches, isolation cells, level shifters,

and retention cells in the appropriate locations.[42]

Guiding the Clock: Clock Tree Synthesis (CTS) Constraints

The clock network is arguably the most critical network in a synchronous digital design. Its proper implementation is vital for the chip to function at its target frequency. While the SDC file defines the *what* of the clock (e.g., its period), a Clock Tree Synthesis (CTS) specification file defines the *how*—the detailed strategy for building the clock tree.[4]

The goal of CTS is to build a buffer/inverter tree that distributes the clock signal from its source (e.g., a PLL) to every sequential element (the "sinks") in the design while meeting several key objectives [44]:

- **Minimize Skew:** Ensure the clock signal arrives at all sinks at as close to the same time as possible.
- **Minimize Insertion Delay (Latency):** Reduce the total time it takes for the clock to travel from its source to the sinks.
- **Maintain Signal Integrity:** Ensure the clock signal has sharp, clean edges (low transition time) everywhere in the network.

A CTS specification file provides the tool with detailed instructions and constraints to achieve these goals, including [4]:

- **Targets:** Explicit targets for maximum skew and maximum insertion delay.
- **DRC Constraints:** Stricter-than-normal electrical constraints for the clock nets, such as max_transition and max_capacitance, to ensure high-quality clock signals.
- **Cell Lists:** A specific list of buffers and inverters that the CTS tool is allowed to use. Often, only specially designed "clock cells" with balanced rise/fall times are permitted.[16]
- **Topology Rules:** Definition of the clock tree roots, identification of "don't touch" sub-trees, and specification of "exclude pins" that should not be considered part of the clock tree.
- **Non-Default Rules (NDRs):** Instructions to the router to use special rules for the clock nets, such as using double-wide wires or enforcing extra spacing from other signals to shield the clock from noise.

These three constraint files—SDC, UPF, and CTS specs—form a hierarchy of design intent. The SDC sets the overall performance contract. The UPF overlays the power architecture, which can introduce new timing complexities. The CTS constraints then provide a detailed implementation strategy for the clock network within the broader context defined by the SDC and UPF.

Putting It All Together - A Practical Guide to Starting a Design

Having explored the individual files and libraries, this final section synthesizes these concepts into a practical, step-by-step guide for setting up a physical design project. It addresses the critical procedural dependencies and illustrates how all the disparate pieces of data are linked together in a cohesive flow.

The Critical First Step: Why Libraries Must Be Loaded Before the Design

A common point of confusion for beginners is the strict procedural order required when loading data into an EDA tool. The libraries must *always* be loaded before the design netlist. The reason for this lies in a fundamental dependency chain.

The design netlist is, at its core, a list of cell instances and their connections. A line like AND2X1 U1 (.A(n1),.B(n2),.Y(n3)); is meaningless to the tool on its own. It raises several questions: What is an AND2X1? How big is it? Where are its pins A, B, and Y? How fast is it? How much power does it consume?

The libraries are the source of all these answers. Loading the technology file (.tf) is akin to defining the coordinate system and the laws of physics for the chip. Loading the reference libraries (.ndm or .lef and .lib) is like loading the dictionary of all available components, providing the physical, timing, and functional definition for every cell like AND2X1.[10]

Only after this complete context has been established can the tool read and understand the design netlist. When the tool parses the netlist, it can now link each instance to its corresponding definition in the loaded libraries. If the design were loaded first, the tool would encounter thousands of "unresolved reference" errors because it would be reading instance names for which it has no definition.[36]

A Step-by-Step Walk-through of the Initial Setup (ICC2 Example)

The following steps outline the typical sequence for initiating a new physical design project in the Synopsys IC Compiler II environment.

1. **Library Preparation (Offline):** This is a one-time setup step performed by a library or CAD team. The ASCII library files from the foundry or IP vendor (.tf, .lef, .lib) are compiled into the tool's native, high-performance NDM reference library format (.ndm) using the ICC2 Library Manager tool (icc2_lm_shell).[31]
2. **Start the Tool:** Launch the main P&R environment: icc2_shell.
3. **Create the Design Library:** The first command in any design script is to create the workspace for the chip. This is done using the create_lib command, which points to the foundational technology file and links all the necessary NDM reference libraries prepared in the previous step.[40]
   Tcl
   ```
   # Define paths to reference libraries
   set ndm_libs [glob /path/to/ndm/*.ndm]

   # Create the design library
   create_lib my_design.ndm \
     -technology /path/to/tech/foundry.tf \
     -ref_libs $ndm_libs
   ```
4. **Load the Design Netlist:** With the library context established, the gate-level Verilog netlist is read into the tool using the read_verilog command. The tool will now parse the netlist and link every cell instance to its definition in the reference libraries.
   Tcl
   ```
   read_verilog my_design.v
   ```
5. **Apply Constraints:** The design intent is loaded by reading the SDC and UPF files. The tool now understands the performance and power goals it must achieve during optimization.
   Tcl
   ```
   read_sdc my_design.sdc
   read_upf my_design.upf
   ```
6. **Sanity Checks:** Before proceeding to the time-consuming stages of physical design, it is crucial to run a series of checks to ensure all inputs are loaded correctly and are consistent with one another. Commands like check_library verify consistency between the logical and physical libraries, and check_timing ensures the SDC constraints are reasonable and properly applied.[35]

At this point, the design is fully loaded, the context is established, and the goals are defined. The project is now ready to move into the first stage of physical implementation:

floorplanning.

How Everything is Linked: A Holistic View of the Data Flow

The entire VLSI design and implementation process can be viewed as a continuous data flow, where information from different sources is progressively combined and enriched to create the final physical layout.

1. **The Sources:** The process begins with two primary sources of information. The **Foundry** provides the process-defining data: the technology file (.tf), interconnect models (.itf, which are compiled into TLU+), and design rules. The **Library Vendor** (which may be the foundry or a third party) designs standard cells that comply with the foundry's rules. They characterize these cells to create the timing/power libraries (.lib) and abstract their physical layouts to create the physical libraries (.lef).
2. **The Design:** The **Design Team** writes RTL code (Verilog/VHDL) describing the chip's functionality. They also write the SDC and UPF files that specify the design's performance and power goals.
3. **Synthesis:** A **Synthesis Tool** takes the RTL, the SDC, and the logical libraries (.lib) as input. It translates the RTL into a gate-level **netlist** (.v), which is an interconnection of cells chosen from the .lib files to meet the timing constraints defined in the SDC.
4. **Physical Design Setup:** The **Physical Design Engineer** gathers all the necessary inputs: the netlist from synthesis, the SDC and UPF from the design team, and all the library files (.tf, .lef, .lib, TLU+) from the foundry and library vendor.
5. **Database Creation:** The **EDA Tool (e.g., ICC2)** ingests all these ASCII files. The Library Manager first compiles the technology and cell libraries into an efficient binary **NDM** format. The main tool then uses create_lib to establish a design library that links to these NDM reference libraries.
6. **Progressive Refinement:** The tool then proceeds through the physical design stages:
    - During **Placement**, it uses the physical information from the NDM (originally from LEF) to legally place each cell instance from the netlist. This process is guided by the timing information (from LIB) and interconnect estimates (from TLU+) to place critical cells closer together. The state of the design is saved as a **DEF** file.
    - During **Clock Tree Synthesis** and **Routing**, the tool adds buffers and wires to the design. The DEF file is continuously updated to reflect these changes.
7. **Signoff and Manufacturing:** Once the layout is complete and passes all verification checks (timing, physical, electrical), the final database is streamed out as a **GDSII** file. This file is sent to the foundry to be manufactured.

The entire flow is a process of progressively adding physical reality to an initially abstract logical design. Each file format serves as a standardized container for a specific type of

information required at a particular stage of this complex and fascinating journey from idea to silicon.

The following table serves as a quick-reference guide, summarizing the key input files and their roles in the VLSI physical design flow.

File Extension	Full Name	Primary Purpose	Key Consumer/Stage
.v /.vhd	Verilog / VHDL Netlist	Describes the logical connectivity of gates and macros.	Input to the entire Physical Design flow.
.sdc	Synopsys Design Constraints	Defines timing constraints (clocks, I/O delays) and optimization goals.	Synthesis, All P&R Stages (Placement, CTS, Routing).
.tf /.tech.lef	Technology File	Defines foundry process rules, layers, and physical characteristics.	Library preparation, entire P&R flow. Must be loaded first.
.itf	Interconnect Technology Format	Detailed ASCII model of interconnect cross-sections for parasitic extraction.	Source for TLU+ generation.
.tluplus	Table Look-Up Plus	Binary lookup tables for fast and accurate interconnect RC extraction.	All P&R stages for timing and SI analysis.
.lef	Library Exchange Format	Physical abstract of standard cells/macros (size, pin locations,	Library preparation, Placement, Routing.

		blockages).	
.lib /.db	Liberty Timing / Database	Timing, power, and functional models for cells. Characterized for PVT corners.	Synthesis, Static Timing Analysis, Power Analysis, Optimization.
.def	Design Exchange Format	A snapshot of the physical layout (component placement, routing geometry).	Output of P&R stages (Placement, CTS, Route).
.upf	Unified Power Format	Defines the design's low-power intent (power domains, switches, etc.).	All P&R stages for low-power design implementation.
.ndm	New Data Model	Synopsys ICC2 proprietary binary database merging.tf,.lef, and.lib.	The core database for the entire ICC2 P&R flow.

Conclusion

The intricate process of VLSI physical design is fundamentally enabled by a well-defined and comprehensive set of input data. This report has detailed the critical libraries, file formats, and data models that form the bedrock of modern chip implementation. The journey from a logical netlist to a manufacturable layout is not a single step but a constrained translation, guided by a hierarchy of information.

At the base lies the foundry's technology data (.tf, TLU+), which dictates the immutable physical rules and electrical properties of the manufacturing process. Built upon this foundation are the standard cell libraries, which serve as a rich optimization toolkit. The

critical abstraction of separating a cell's physical view (.lef) from its performance view (.lib) is what enables EDA tools to efficiently manage the immense complexity of giga-scale designs. This separation of concerns allows geometric engines to focus on layout while analysis engines perform detailed timing and power calculations.

The design itself is captured in evolving formats, with the DEF file acting as a progressive snapshot of the physical layout. Modern EDA tools further optimize this ecosystem by compiling these standard formats into high-performance, proprietary databases like Synopsys's NDM, a necessary evolution to handle the data capacity challenges of advanced process nodes. Finally, higher-level design intent for power (.upf) and clocking (CTS specs) is layered on top of the fundamental timing constraints (.sdc), allowing for the automated implementation of sophisticated, multi-objective design goals.

Understanding this ecosystem—the purpose of each file, the data it contains, and its dependencies on others—is not merely an academic exercise. It is the essential, practical knowledge required for any engineer to successfully navigate the complex, challenging, and ultimately rewarding process of transforming a digital design into a physical reality.

Works cited

1. Inputs of physical design | ODP - SlideShare, accessed September 11, 2025, https://www.slideshare.net/slideshow/inputs-of-physical-design/79931343
2. Standard-Cell Libraries 101: What They Are & How They Shape Your VLSI Design, accessed September 11, 2025, https://vlsifacts.com/standard%E2%80%91cell-libraries-101-what-they-are-how-they-shape-your-vlsi-design/
3. Physical design (electronics) - Wikipedia, accessed September 11, 2025, https://en.wikipedia.org/wiki/Physical_design_(electronics)
4. pd_inputs_html.pdf
5. VLSI Design Flow - GeeksforGeeks, accessed September 11, 2025, https://www.geeksforgeeks.org/electronics-engineering/vlsi-design-flow/
6. Data inputs for PD tools - This is "MY" Blog - WordPress.com, accessed September 11, 2025, https://reghuraj312.wordpress.com/2010/09/13/data-inputs-for-pd-tools/
7. What Is Design Exchange Format | PDF - Scribd, accessed September 11, 2025, https://www.scribd.com/document/398938925/What-is-Design-Exchange-Format
8. Standard cell - Wikipedia, accessed September 11, 2025, https://en.wikipedia.org/wiki/Standard_cell
9. asic - Please explain tech.lef , tech.lib - Electrical Engineering Stack ..., accessed September 11, 2025, https://electronics.stackexchange.com/questions/77068/please-explain-tech-lef-tech-lib
10. TECHNOLOGY FILE - VLSI DESIGN, accessed September 11, 2025, https://vlsiwikipedia.blogspot.com/p/technology-file.html

11. Technology LEF - VLSI Pro, accessed September 11, 2025, https://vlsi.pro/technology-lef/
12. Liberty Timing File (LIB), accessed September 11, 2025, https://www.csee.umbc.edu/courses/graduate/CMPE641/Fall08/cpatel2/slides/lect05_LIB.pdf
13. TLU+ File - SoC Physical Design, accessed September 11, 2025, https://physicaldesign-asic.blogspot.com/2020/07/tlu-file.html
14. Interconnect Technology Format (ITF) - SoC Physical Design, accessed September 11, 2025, https://physicaldesign-asic.blogspot.com/2020/07/interconnect-technology-format-itf.html
15. bharath19-gs/synopsys_ICC2flow_130nm: This repo is for synopsys icc2 flow for skywater 130nm PDK - GitHub, accessed September 11, 2025, https://github.com/bharath19-gs/synopsys_ICC2flow_130nm
16. Digital VLSI Design Lecture 4: Standard Cell Libraries, accessed September 11, 2025, https://www.eng.biu.ac.il/temanad/files/2017/02/Lecture-4-Standard-Cell-Libraries.pdf
17. Library Exchange Format - Wikipedia, accessed September 11, 2025, https://en.wikipedia.org/wiki/Library_Exchange_Format
18. What is LEF? Competitors, Complementary Techs & Usage | Sumble, accessed September 11, 2025, https://sumble.com/tech/lef
19. LEF/DEF 5.8 Language Reference -- 1, accessed September 11, 2025, http://coriolis.lip6.fr/doc/lefdef/lefdefref/LEFSyntax.html
20. Input files for Physical Design -> Lef and tf file | PDF | Desktop Publishing - SlideShare, accessed September 11, 2025, https://www.slideshare.net/slideshow/input-files-for-physical-design-lef-and-tf-file/271495289
21. liberty - maaldaar, accessed September 11, 2025, http://www.maaldaar.com/index.php/vlsi-digital-standards/liberty
22. What is Library Characterization? – How it Works & Techniques | Synopsys, accessed September 11, 2025, https://www.synopsys.com/glossary/what-is-library-characterization.html
23. PVT Corners in VLSI: Navigating Process, Voltage, and Temperature Variations - ChipEdge, accessed September 11, 2025, https://chipedge.com/resources/what-are-pvt-corners-in-vlsi/
24. Foundation of PVT Variation, you always wanted to know - Paripath, accessed September 11, 2025, https://www.paripath.com/blog/variation-blog/foundation-of-pvt-variation-you-always-wanted-to-know
25. Liberty File - VLSI Master, accessed September 11, 2025, https://vlsimaster.com/liberty-file/
26. INSTITUTO DE COMPUTAÇÃO - IC-Unicamp, accessed September 11, 2025, https://www.ic.unicamp.br/~reltech/2015/15-01.pdf
27. en.wikipedia.org, accessed September 11, 2025,

https://en.wikipedia.org/wiki/Design_Exchange_Format#:~:text=Design%20Exchange%20Format%20(DEF)%20is,the%20netlist%20and%20circuit%20layout.
28. LEF/DEF 5.8 Language Reference -- 4, accessed September 11, 2025, http://coriolis.lip6.fr/doc/lefdef/lefdefref/DEFSyntax.html
29. What is Milkyway Library and FRAM view? - Learn VLSI, accessed September 11, 2025, https://learnvlsi.com/pd/what-is-milkyway-library-and-fram-view/261/
30. How different is it to use Synopsys ICC2 compared to ICC1? : r/ECE - Reddit, accessed September 11, 2025, https://www.reddit.com/r/ECE/comments/xp6s9j/how_different_is_it_to_use_synopsys_icc2_compared/
31. What is the NDM in VLSI physical design? - VLSI Beginners - Quora, accessed September 11, 2025, https://vlsibeginners.quora.com/What-is-the-NDM-in-VLSI-physical-design
32. IC Compiler II: Industry Leading Place and Route System - Synopsys, accessed September 11, 2025, https://www.synopsys.com/content/dam/synopsys/implementation&signoff/datasheets/ic-compiler-ii-ds.pdf
33. IC Compiler II: Place & Route Solution - Synopsys, accessed September 11, 2025, https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html
34. [synopsys library manager] Bounding Boxes of lib cells. : r/chipdesign - Reddit, accessed September 11, 2025, https://www.reddit.com/r/chipdesign/comments/mz07j7/synopsys_library_manager_bounding_boxes_of_lib/
35. Basics of IC Compiler - VLSI Physical Design, accessed September 11, 2025, http://www.vlsijunction.com/2015/08/ic-compiler-user-guide.html
36. Expert Advise : Difference between target & link library - Digital Design, accessed September 11, 2025, https://rtldigitaldesign.blogspot.com/2015/08/difference-between-target-link-library.html
37. symbol library, link library, etc. : r/FPGA - Reddit, accessed September 11, 2025, https://www.reddit.com/r/FPGA/comments/14mu2me/symbol_library_link_library_etc/
38. Synopsys library_compiler create_physical_lib command does not ..., accessed September 11, 2025, https://www.reddit.com/r/ECE/comments/k590jw/synopsys_library_compiler_create_physical_lib/
39. FC & Icc2 CMNDS | PDF - Scribd, accessed September 11, 2025, https://www.scribd.com/document/807734895/FC-ICC2-CMNDS
40. IC Compiler™ II Design Planning User Guide | PDF | License | Free Software - Scribd, accessed September 11, 2025, https://www.scribd.com/document/629489857/IC-Compiler-II-Design-Planning-User-Guide
41. Unified Power Format - Wikipedia, accessed September 11, 2025, https://en.wikipedia.org/wiki/Unified_Power_Format

42. Unified Power Format (.upf) in VLSI Physical Design | iVLSI ..., accessed September 11, 2025, https://ivlsi.com/unified-power-format-upf-in-vlsi-physical-design/
43. IEEE1801 Unified Power Format, accessed September 11, 2025, https://www.p1801.org/
44. Clock Tree Synthesis | SoC Labs, accessed September 11, 2025, https://soclabs.org/design-flow/clock-tree-synthesis
45. Clock Tree Synthesis | PDF | Electronic Engineering | Computing - Scribd, accessed September 11, 2025, https://www.scribd.com/document/594485090/8-Clock-Tree-Synthesis
46. Synthesis-aware clock analysis and constraints generation - EE Times, accessed September 11, 2025, https://www.eetimes.com/synthesis-aware-clock-analysis-and-constraints-generation/