The Architect's Blueprint: A Comprehensive Guide to VLSI Floorplanning from Fundamentals to Future Frontiers

The Foundation of Physical Design: Understanding Floorplanning

In the intricate and demanding world of Very Large-Scale Integration (VLSI), where billions of transistors are orchestrated on a single piece of silicon, the physical design process transforms a logical circuit description into a manufacturable layout. Within this process, floorplanning stands as the most foundational and arguably most critical stage. It is the architectural blueprint of the chip, a phase where high-level decisions have a profound and often irreversible impact on the final product's performance, power consumption, area, and ultimately, its commercial viability.

What is Floorplanning? A Precise Definition

At its core, floorplanning is the strategic process of arranging major functional blocks, Input/Output (I/O) pads, and other key physical structures on the die.[1] It is an early, pivotal step in the physical design flow that establishes the overall topology and structure of the chip's layout, setting the stage for all subsequent, more granular implementation tasks.[3]

A crucial distinction must be made between floorplanning and the subsequent stage of placement. Floorplanning operates at a higher level of abstraction, dealing with large, pre-designed blocks known as macros or Intellectual Property (IP) cores (e.g., memory blocks, processor cores, analog circuits), and partitions of the design. These blocks may have fixed

dimensions (hard macros) or offer some flexibility in their shape (soft or semi-soft macros).[5] This stage is often a semi-automated or even manual process, heavily guided by the design architect's intent and experience. In contrast, placement is a highly automated, algorithm-driven process that takes the floorplan as a fixed canvas and meticulously places millions of individual standard cells (the basic logic gates like NAND, NOR, and flip-flops) into the predefined rows and regions.[5] In essence, floorplanning is analogous to creating a city plan—deciding where to locate the residential districts, industrial zones, and major highways. Placement is the subsequent task of arranging every individual house on its designated street within those districts.

This distinction highlights the dual nature of floorplanning as both a science and an art.[1] The "science" is rooted in combinatorial optimization, where algorithms strive to minimize concrete, measurable metrics such as total chip area and estimated wirelength. The "art" resides in the human-guided navigation of a complex, multi-dimensional problem space with numerous, often conflicting, objectives. Achieving an optimal balance between performance, power, area, and routability requires a deep, intuitive understanding of the design's architecture and data flow—a skill honed through experience.[6]

The Critical Role of a Good Floorplan: Why It's the "Make or Break" Stage

The quality of a chip is fundamentally determined by its floorplan. A well-conceived floorplan can render the complex downstream tasks of placement, Clock Tree Synthesis (CTS), and routing a "cake walk," ensuring a smooth path to design closure.[7] Conversely, a poor floorplan creates a cascade of intractable problems that no amount of optimization in later stages can resolve, dooming the design to failure.[7] The decisions made during floorplanning directly dictate the chip's final Power, Performance, and Area (PPA) metrics, which are the primary measures of its quality and competitiveness.[1] A suboptimal floorplan can lead to an inflated die size, excessive power consumption, and degraded performance, ultimately affecting the Integrated Circuit's (IC) manufacturing cost, operational reliability, and even its functional lifespan.[7]

The paramount importance of floorplanning is mandated by a fundamental physical reality of modern semiconductor technology: the dominance of interconnect delay. As manufacturing processes have advanced into the deep-submicron era (e.g., 7nm, 5nm, and beyond), the performance of transistors has scaled dramatically, leading to a significant reduction in gate delay—the time it takes for a logic gate to perform its function. However, the interconnects—the metal wires that connect these gates—have not scaled at the same rate. The resistance and capacitance of these wires per unit length have increased, making them

relatively slower.[10] Consequently, the time it takes for a signal to travel along a wire now constitutes the majority of the total path delay in a critical path. This physical shift means that the physical distance between communicating blocks, a quantity determined entirely by the floorplan, has become the primary bottleneck for chip performance. Therefore, floorplanning has evolved from a simple block-packing exercise into the principal stage for performance optimization in any modern VLSI design.

The Core Objectives of Floorplanning

Floorplanning is a multi-objective optimization problem, tasked with simultaneously achieving several competing goals. The primary objectives include:

- **Minimize Area:** The most straightforward objective is to arrange all blocks in the most compact way possible. This minimizes the overall die size, which directly reduces the manufacturing cost per chip, as more chips can be fabricated on a single silicon wafer.[1]
- **Minimize Timing / Reduce Wirelength:** To meet performance targets, blocks that have a high degree of connectivity or lie on timing-critical paths must be placed physically close to one another. This shortens the interconnecting wires, reducing their parasitic resistance and capacitance, which in turn minimizes signal propagation delay.[2]
- **Enable Routability / Reduce Congestion:** The floorplan must allocate sufficient space and create clear, unobstructed pathways for the millions of wires that will be added during the routing stage. A poorly planned arrangement can create "congestion hotspots"—areas where the demand for routing tracks exceeds the available supply—making it impossible to connect all the components without creating electrical shorts or other violations.[7]
- **Ensure Power Integrity / Reduce IR Drop:** A robust Power Delivery Network (PDN) must be planned to supply a stable and clean voltage to every component on the chip. This involves creating a grid of power and ground wires that minimizes voltage drop (IR drop), a phenomenon that can slow down transistors and cause timing failures.[7]

Floorplanning in the VLSI Physical Design Flow

The floorplanning stage is situated early in the back-end or physical design phase of the overall VLSI design cycle. A typical flow proceeds as follows:

1. **RTL Design & Verification:** The chip's functionality is described using a Hardware Description Language (HDL) like Verilog or VHDL.

2. **Logic Synthesis:** The HDL code is translated into a gate-level netlist, which is a description of the circuit in terms of standard logic cells and their interconnections.
3. **Partitioning:** For very large designs, the netlist is divided into smaller, hierarchical blocks.
4. **Floorplanning:** The physical locations, shapes, and orientations of these blocks and other major components are determined.
5. **Placement:** Standard cells are placed into the rows defined by the floorplan.
6. **Clock Tree Synthesis (CTS):** A network is built to distribute the clock signal to all sequential elements with minimal timing differences (skew).
7. **Routing:** The physical wire connections between all cell pins are created.
8. **Physical Verification:** The final layout is checked for manufacturability (DRC), correctness (LVS), and other electrical rules before being sent for fabrication (GDSII tape-out).[4]

Floorplanning takes the synthesized gate-level netlist and a set of physical and timing library files as its primary inputs.[8] Its output is a physical layout plan, typically encapsulated in a Design Exchange Format (DEF) file, which serves as the foundational input and set of constraints for the subsequent placement stage.[7]

Preparing the Canvas: Pre-Floorplanning Essentials

Before the creative and analytical work of floorplanning can begin, a rigorous preparatory phase is required to gather and validate all necessary inputs. This phase acts as a critical quality gate, ensuring that the Place and Route (PnR) tool is equipped with accurate and consistent data. Neglecting this step inevitably leads to the "Garbage In, Garbage Out" syndrome, where flawed inputs result in a flawed and often unworkable physical design.

Design Import: Gathering the Ingredients

The first step in the physical design flow is to load all the required design files into the PnR tool. The order of loading is significant; typically, physical libraries containing the geometric information of cells are loaded before the timing libraries that characterize their performance.[7] Each file provides a different piece of the puzzle, defining the logical, physical, timing, and manufacturing constraints of the design.

Table 1: Floorplanning Input File Summary

File Type	Extension(s)	Content Description	Role in Floorplanning	Source(s)
Logical Connectivity	.v, .vhd, .edif	Gate-level netlist describing standard cells, macros, and their interconnections.	Defines the complete set of components to be placed and their logical connectivity. It is the basis for initial area estimation and fly-line analysis.	[7]
Physical Libraries	.lef	Library Exchange Format. Contains abstract physical views of all cells (macros and standard cells), including their dimensions, pin locations, and metal blockage layers.	Provides the physical "shape" and interface for every component, dictating how they fit together on the die.	[7, 15]
Timing Libraries	.lib	Liberty Timing Format. Characterizes the timing (e.g.,	Essential for timing-driven floorplanning and subsequent	[7, 15]

		propagation delay, setup/hold times) and power consumption of each cell under various operational conditions (process, voltage, temperature).	optimization stages to ensure the design meets its performance goals.	
Technology Data	.tf, Tech LEF	Technology file. Defines the rules for the specific manufacturing process, such as the number of metal layers, their electrical properties (resistance, capacitance), via rules, and the manufacturing grid.	Sets the fundamental physical rules of the silicon canvas upon which the layout will be drawn.	[7]
Timing Constraints	.sdc	Synopsys Design Constraints. Specifies the design's performance requirements, including clock definitions (frequency,	The primary input that guides the tool's timing-driven optimization efforts. The floorplan must be structured to make these	[7]

		uncertainty), I/O delays, false paths, and multicycle paths.	constraints achievable.	
Physical Constraints	.def, .fp	Design Exchange Format / Floorplan file. An optional input that can provide a pre-existing floorplan, pre-placed cells, or specific pin locations to be used as a starting point or a hard constraint.	Allows for incremental design changes or the enforcement of system-level architectural decisions.	[7]
Power Specifications	.upf, .cpf	Unified Power Format / Common Power Format. Defines the design's power intent, including multiple power domains, voltage levels, isolation rules, and level shifter requirements.	Critical for modern low-power designs, guiding the creation of a complex power grid and the placement of power management cells.	[16]

Pre-Floorplan Sanity Checks: A Critical Quality Gate

Once the design data is loaded, it is imperative to perform a series of sanity checks to verify the quality and consistency of the inputs.[7] This step is not merely a technical debugging exercise; it is a fundamental risk mitigation strategy with significant financial and schedule implications. A modern System-on-Chip (SoC) design cycle can span many months and consume millions of dollars in engineering resources and EDA license costs. A fundamental error in the input netlist, such as a combinational timing loop, or an omission in the constraints file, like an unconstrained critical path, may not be detected during initial logic simulation. If such a "showstopper" bug is only discovered late in the design cycle—for example, after weeks of placement and routing have been completed—it can invalidate a substantial amount of work, forcing a costly and time-consuming restart from the floorplanning stage or even from the RTL design phase. This represents a major schedule slip and budget overrun. Therefore, the pre-floorplan sanity checks serve as the most cost-effective and rapid point at which to identify and rectify these critical issues, acting as a crucial financial control point for the entire project.

The comprehensive checklist for these sanity checks can be categorized as follows:

- **Library Checks:** Verifying the integrity of the physical and timing libraries. This includes checking for missing cell definitions, missing pin information within a cell, or duplicate cell definitions that could confuse the tool.[7]
- **Design Checks:** Analyzing the logical structure of the netlist for common issues that can cause problems in physical implementation. These include inputs with floating (unconnected) pins, nets with multiple drivers (contention), combinational timing loops which can cause oscillations, empty modules that waste area, and assign statements which can create unintentional feedthrough paths.[7]
- **Constraint Checks:** Validating the completeness and correctness of the timing constraints (.sdc file). This involves ensuring that all sequential elements (flip-flops) are driven by a defined clock, that there are no unconstrained paths between registers (which the tool would otherwise ignore during optimization), and that input and output delays are properly specified to model the chip's external environment.[7]

The Art of Partitioning: Divide and Conquer

For the vast and complex SoCs common today, attempting to floorplan and place the entire design as a single, flat entity is computationally infeasible. Partitioning is the essential strategy of "divide and conquer," where the chip is broken down into smaller, more

manageable blocks.[4] This approach is fundamental to managing design complexity, enabling parallel development, and achieving a better overall PPA.

Logical vs. Physical Partitioning

The concept of partitioning exists at two distinct levels of the design process:

- **Logical Partitioning:** This occurs early in the design cycle, during the Register-Transfer Level (RTL) design phase. The system architect divides the design into functional sub-modules based on the chip's architecture, such as a CPU core, a GPU, a memory controller, or a high-speed serial interface. This is a high-level decision that defines the logical hierarchy of the design.[4]
- **Physical Partitioning:** This takes place during the physical design stage. It involves mapping the pre-defined logical partitions to distinct physical regions on the die. This step includes defining the physical boundaries, aspect ratio, and pin locations for each partition, effectively creating a floorplan at the partition level.[18]

Implementation Styles

Based on the partitioning strategy, a design can be implemented using one of two primary styles:

- **Flat:** The entire design is treated as a single partition. This approach is suitable for small to medium-sized designs. Its main advantage is superior area utilization, as no physical space needs to be reserved for routing channels or power structures between partitions.[7]
- **Hierarchical:** The design is implemented as a collection of distinct physical partitions (or blocks). Each block is implemented independently—often by different teams working in parallel—through its own floorplanning, placement, and routing flow. These completed blocks are then integrated at the top level of the chip. This methodology is indispensable for managing the complexity of very large SoCs.[7]

Criteria for Deciding Partitions

The primary goal of partitioning is to create blocks that are functionally self-contained and

have minimal interaction (i.e., a low number of connections) with other blocks.[17] This minimizes routing complexity at the top level. Key criteria for defining partitions include:

- **Design Hierarchy:** Aligning physical partitions with the logical hierarchy defined in the RTL is the most natural approach.[7]
- **Timing Criticality:** Grouping all the logic that forms a timing-critical path into a single partition ensures that these components can be placed physically close together, minimizing interconnect delay.[7]
- **Functionality and Clock Domains:** Grouping logic with related functionality or logic that operates on the same clock domain simplifies the implementation and timing closure process for that partition.[7]

How Partitions are Reconnected

Once the individual partitions are physically implemented, they must be connected at the top level of the chip. This involves two key concepts:

- **Pin Assignment:** The logical connections between partitions, defined in the top-level netlist, must be mapped to physical pin locations on the boundaries of each partition. The process of assigning these pins is a complex optimization problem in itself. A poor pin assignment can lead to significant routing congestion at the top level, as wires may have to cross over each other in convoluted paths to reach their destinations.[19]
- **Feedthroughs:** In some cases, a signal may need to travel from a pin on one side of a partition to a pin on the other side without interacting with any of the logic inside. This requires the creation of a **feedthrough path**, which is essentially a dedicated wire, often with buffers, that cuts across the partition. Feedthroughs must be planned during the partition's floorplan stage, as they consume valuable internal routing resources. A key objective in partitioning is to minimize the number of cross-partition I/Os to reduce the need for complex pin assignments and resource-consuming feedthroughs.[7]

Constructing the Floorplan: A Step-by-Step Guide

With the design data imported and validated, the process of constructing the floorplan begins. This is a methodical procedure where the physical designer, aided by the PnR tool, makes a series of foundational decisions that define the chip's physical architecture. The typical flow involves defining the overall chip dimensions, placing I/O structures, creating the

internal core structure, and establishing the power delivery network.

Defining the Chip Canvas: Die Size, Aspect Ratio, and Core Utilization

The first step is to define the physical boundaries of the design. This involves estimating the total silicon area required and determining its shape and how densely it will be populated.

- **Die Size Estimation:** The tool performs an initial estimation of the required die size based on the sum of the areas of all standard cells from the netlist, all macros from the physical libraries, and any required I/O cells.[1] This provides a starting point for the chip's dimensions.
- **Aspect Ratio (AR):** The aspect ratio is the ratio of the core's height to its width (AR=Height/Width).[7] An aspect ratio of 1.0 results in a perfectly square core, while values greater or less than 1.0 produce a rectangular shape. The choice of AR is a critical decision. It must balance the availability of horizontal and vertical routing resources; for instance, a tall, narrow floorplan will have more horizontal routing tracks but fewer (and longer) vertical ones.[21] The AR may also be constrained by external factors, such as the chip's packaging or its integration into a larger system-level design.[1]
- **Core Utilization:** This metric defines the percentage of the core area that will be occupied by placeable components (standard cells and macros).[7] It is calculated as Utilization=(AreaStdCell+AreaMacro)/AreaCore. A typical target utilization for modern designs is in the range of 70% to 80%.[8] The remaining 20-30% of the core area is not wasted space; it is essential "white space" that is reserved for two critical purposes: allowing the placement of new buffers and inverters that are added during timing and CTS optimization, and providing the necessary tracks for signal routing.

Core utilization serves as the primary control knob for managing routing congestion. Setting a high utilization target (e.g., >85%) is tempting as it reduces the die size and thus the chip's cost. However, it leaves very little room for routing, drastically increasing the risk of congestion hotspots, which can make the design unroutable or prevent it from meeting timing requirements.[1] Conversely, a low utilization (e.g., <65%) provides ample routing space, making design closure easier, but it wastes expensive silicon real estate. Therefore, the selected utilization target represents a strategic trade-off between manufacturing cost and the risk associated with achieving design closure.

The Chip's Interface: I/O and Pad Placement

The Input/Output (I/O) pads and pins are the physical entities that form the electrical interface between the chip and the external world, such as a printed circuit board (PCB).[1]

- **Pad Ring Creation:** The arrangement of these I/O pads around the periphery of the core forms a structure known as the "pad ring".[7] The placement of individual pads is guided by system-level connectivity requirements (e.g., placing the memory interface pads on the side of the chip that will face the DRAM) and by timing constraints.
- **Specialized Pad Cells:** The pad ring is not composed solely of signal pads. It also includes several types of specialized, physical-only cells:
  - **Core Power and Ground Pads:** These connect the on-chip power grid to the external power supply.
  - **I/O Power and Ground Pads:** These provide a separate, often higher-voltage, supply for the I/O drivers.
  - **Corner Pads:** These are placed at the corners of the die to ensure the physical and electrical continuity of the pad ring.[7]
  - **Filler Pads:** These are used to fill any gaps between other pads to maintain the integrity of the ring structure.[7]
- **Core to I/O Spacing:** A specific clearance or channel must be maintained between the boundary of the core (where standard cells are placed) and the inner edge of the I/O pads. This space is crucial for routing the main power and ground rings that supply the core and for routing the signals from the I/O pads to the core logic.[7]

Structuring the Core: Row Configuration and Cell Orientation

The core area, defined by the floorplan, must be structured to accommodate the millions of standard cells that will be placed within it.

- **Standard Cell Rows:** The core is filled with a series of horizontal **Standard Cell Rows**. These rows act as the tracks upon which the standard cells are placed.[7]
- **Standard Cell Sites:** Each row is composed of fundamental units called **Standard Cell Sites** or **Unit Tiles**. A site represents the smallest placeable unit in the design. The height of a site is equal to the height of the standard cells, and its width is typically determined by the width of the smallest cell in the library (often a filler cell). All standard cells must have a width that is an integer multiple of the site width.[7]
- **Cell Orientation:** To maximize area efficiency and facilitate power sharing, standard cell rows are typically arranged with alternating orientations. The most common configuration is the "Butt and flip" method, where adjacent rows are placed directly next to each other (butted), and every other row is flipped vertically (e.g., R0, R1F, R2, R3F...). This allows the VDD power rail of one row and the VSS (ground) rail of the adjacent row to be shared, saving significant area compared to configurations that require a gap between every

row.[7]

Power Planning: Energizing the Chip

Concurrent with the structural definition of the floorplan, the designer must create the Power Delivery Network (PDN). This is a critical sub-task often referred to as power planning, and it is integral to the floorplanning stage, not an afterthought.[1] A robust PDN is essential for delivering a stable voltage supply to every transistor on the chip.

The PDN is typically constructed as a hierarchical grid of metal wires:

- **Power Rings:** One or more wide metal rings are created around the periphery of the core. Separate, dedicated rings are often created around large power-hungry macros.[13]
- **Power Straps:** A mesh of horizontal and vertical metal straps is laid out over the entire core area. These straps are connected to the peripheral rings and serve to distribute power from the edges of the chip into the center.[13]
- **Standard Cell Rails:** Within each standard cell row, thin metal wires, known as rails, run horizontally. These rails connect directly to the VDD and VSS pins of the standard cells and tap into the power from the overlying strap mesh.[13]

Designing the PDN involves addressing two major power integrity challenges:

- **IR Drop:** The metal wires of the PDN have a finite resistance. As current flows through these wires to power the cells, a voltage drop ($V=I \times R$) occurs. This is known as IR drop. If the IR drop is excessive, the voltage supplied to a cell will be lower than intended, which can significantly increase its delay, leading to timing violations, or in severe cases, functional failure.[7] IR drop can be **static**, caused by average leakage currents, or **dynamic**, caused by the large instantaneous current draw when many cells switch simultaneously. Mitigation techniques include using wider and thicker power straps (often on higher, less resistive metal layers), increasing the number of straps to create a denser grid, and strategically placing decoupling capacitors (DCAPs) to act as local charge reservoirs.[13]
- **Electromigration (EM):** This is a long-term reliability issue caused by the physical displacement of metal atoms due to the momentum transfer from a high-density flow of electrons. Over time, this can lead to the formation of voids (open circuits) or hillocks (short circuits) in the power grid, causing the chip to fail in the field.[13] EM is exacerbated by high current density and high temperatures. The mitigation strategies are similar to those for IR drop: using wider wires to reduce current density and avoiding excessively long, unbroken stretches of metal.[13]

The Floorplan Outputs: Hand-off to Placement

The culmination of the floorplanning stage is a comprehensive physical database that is handed off to the placement tool. This database is typically stored in a **Design Exchange Format (DEF)** file and contains the following finalized information [7]:

- The defined core boundary and overall die area.
- The precise locations, layers, and orientations of all placed I/O ports.
- The final, fixed locations and orientations of all hard macros.
- The complete definition of the standard cell rows, including their location and orientation.
- The detailed layout of the power delivery network (rings and straps).
- The locations and types of all placement blockages.

This DEF file provides the complete physical context and set of constraints within which the placement and routing engines will operate.

Mastering the Craft: Strategies for a Perfect Floorplan

Moving beyond the procedural steps of floorplanning, achieving an optimal layout requires a strategic approach that balances competing objectives and anticipates downstream challenges. This section delves into the advanced techniques and decision-making processes that distinguish a merely functional floorplan from a high-performance, robust, and cost-effective one. These strategies involve the artful placement of large macros, the judicious use of placement blockages, and the integration of timing constraints directly into the floorplanning process.

Advanced Macro Placement: The Art of Placing the "Big Rocks"

The placement of macros is the single most impactful decision made during floorplanning. These large blocks act as immovable islands in a sea of standard cells, fundamentally dictating the chip's data flow, defining the shape of the areas available for standard cell placement, and creating the major channels and choke points for routing.[7]

Fly-line Analysis

Before any macros are placed, a crucial initial step is to perform **fly-line analysis**. Fly-lines are straight, virtual lines drawn on the layout canvas that represent the logical connections (nets) between macros and between macros and I/O pins.[7] A dense bundle of fly-lines between two blocks indicates a high degree of connectivity and a strong justification for placing them close together. This visualization provides an immediate, high-level understanding of the chip's communication architecture and guides the initial grouping and placement of macros.

Strategic Guidelines & Trade-offs

Achieving an optimal macro placement involves adhering to a set of proven guidelines while intelligently navigating the necessary trade-offs:

- **Place Macros on the Periphery:** Whenever possible, macros should be placed around the edges of the core area. This strategy is highly beneficial because it creates a large, contiguous, and regularly shaped (ideally rectangular) region in the center for the standard cells.[7] A contiguous standard cell area is far easier for automated placement and CTS tools to handle efficiently, leading to better timing and lower congestion. Placing a large macro in the center of the core acts as a massive obstacle, fragmenting the standard cell area and forcing global signals to take long, convoluted detours around it.
- **Optimize Macro Orientation:** The orientation of a macro should be chosen to minimize the distance between its pins and the logic it connects to. Ideally, pins should face towards the core if they connect primarily to standard cells, or towards other macros if they form a critical macro-to-macro interface.[7] In advanced technology nodes, the orientation of macros may be restricted due to lithographic constraints related to polysilicon directionality, limiting the designer's flexibility.[7]
- **Ensure I/O Proximity:** Macros that communicate heavily with external signals should be placed near their corresponding I/O pads on the chip's periphery. This minimizes the length of long I/O routes, which can be a significant source of delay and signal integrity issues.[24]
- **Provide Adequate Channel Spacing:** A common mistake is to pack macros tightly together to minimize area. This is often counterproductive. Sufficient channels must be left between macros. These channels are not wasted space; they are essential resources for routing signal nets between the macros, for routing the power straps that supply them, and, critically, for placing buffers and inverters during timing optimization and CTS.[7]

Insufficient channel spacing is one of the most common and severe causes of localized routing congestion.[7]

- **Shape the Standard Cell Area:** The arrangement of macros implicitly defines the shape of the remaining area available for standard cells. It is crucial to avoid creating odd or problematic shapes, such as long, narrow channels or enclosed "traps" where standard cells can be placed but are difficult to route to. The goal is to create a homogeneous, contiguous, and ideally square-like area for the standard cell logic, as this provides the most flexibility for the placement and routing tools.[7]

Guiding the Placer: Blockage and Halo Management

To enforce the strategic decisions made during macro placement and to guide the subsequent automated placement of standard cells, designers use **placement blockages**. These are defined regions where the placement tool is forbidden or restricted from placing cells.[7] They are an essential tool for proactively managing congestion and controlling the distribution of logic.

Types of Placement Blockages

- **Hard Blockage:** An area where no cells of any type are allowed to be placed. Hard blockages are used to reserve areas exclusively for routing, to prevent the placer from putting cells in geometrically awkward regions (like the sharp corners formed by two adjacent macros), or to control the generation of power rails.[7]
- **Soft Blockage:** An area where general-purpose standard cells are forbidden, but buffers and inverters are permitted. This is a powerful technique for creating routing channels between blocks where timing can be optimized by buffer insertion without adding to the logical complexity and congestion of the area.[7]
- **Partial Blockage:** A region where the placement tool is allowed to fill only a certain percentage of the available sites with standard cells. This is used to gently reduce the cell density in areas that are anticipated to be congested, thereby alleviating routing pressure without completely blocking off the area.[7]

Halos (Keep-outs)

A **halo**, also known as a keep-out margin, is a special type of blockage that is defined relative to a macro rather than by absolute coordinates on the die. The halo forms a perimeter around the macro and moves with it if the macro is relocated.[7] Halos are typically defined as soft blockages and are extremely useful for ensuring that a clear zone is maintained around a macro's pins for routing access and for the placement of drivers or buffers that connect directly to those pins.

Timing-Driven Floorplanning: Racing Against the Clock

In performance-critical designs, the floorplan cannot be created based on geometric considerations alone. **Timing-driven floorplanning** is an approach that explicitly uses the timing constraints defined in the SDC file to guide the placement of blocks, with the primary objective of ensuring the chip can operate at its target frequency.[2]

Understanding Timing Paths

Static Timing Analysis (STA) is the method used to verify a design's timing performance. STA tools analyze millions of **timing paths** in the design. A valid timing path typically starts at an input port of the chip or the clock pin of a sequential element (like a flip-flop) and ends at an output port or the data input pin of another sequential element.[26] The tool calculates the total delay along each path and compares it to the time allowed by the clock period, reporting the difference as

slack. A negative slack indicates a timing violation that must be fixed.

Net-Based vs. Path-Based Approaches

There are two main strategies for incorporating timing into the floorplanning process:

- **Net-Based:** This is a simpler, more heuristic approach. The designer or a preliminary STA run identifies nets that are part of known critical paths. These nets are then assigned a higher weight or a stricter length constraint. The floorplanning tool's optimization engine

then prioritizes minimizing the length of these high-weight nets, with the assumption that shortening them will help fix the critical path.[10]

- **Path-Based:** This is a more sophisticated and accurate approach. The floorplanner is tightly integrated with a lightweight STA engine. During optimization, the tool analyzes the delay of entire timing paths, not just individual nets. It identifies the path with the Worst Negative Slack (WNS) and makes floorplanning decisions—such as moving two macros closer together—that are specifically aimed at reducing the delay of that most critical path.[27] This allows for a more direct and effective optimization of the chip's overall performance.

The outcome of a timing-driven floorplanning process is a layout where the placement of critical blocks is dictated by performance requirements. For example, two macros that are part of a critical reg2reg path will be forced to be placed physically adjacent, even if this creates a less-than-ideal geometric arrangement, because meeting the timing constraint is the overriding priority.[26]

Clarifying Terminology: Design Planning vs. Floorplanning

In the lexicon of the semiconductor industry and EDA tool vendors like Synopsys and Cadence, the term **"Design Planning"** is often used to describe a broader, more holistic set of early-stage physical design activities that encompasses traditional floorplanning.[30]

Design Planning can be thought of as the complete architectural phase of physical design. It includes not only the placement of macros and I/Os (i.e., floorplanning) but also high-level design partitioning, detailed power network planning, and initial exploration of global routing feasibility.[31] It represents a system-level approach to defining the chip's physical hierarchy and infrastructure before committing to the detailed, computationally intensive stages of placement and routing. In this context, floorplanning is the core activity and primary component within the broader discipline of design planning.[8]

Verification, Consequences, and Iteration

Once an initial floorplan is created, it must be rigorously verified before the design can proceed to the next stage. This qualification step is essential because the consequences of a flawed floorplan are severe, often creating a domino effect of problems that ripple through the entire physical design flow. Understanding these consequences and the high cost of

iterating back to the floorplan stage underscores the immense pressure to achieve a high-quality result from the outset.

Floorplan Qualification: The Final Checklist

Before handing off the design to the placement tool, the floorplan must be subjected to a series of qualification checks to ensure its legality, quality, and feasibility.[7] This checklist serves as a final sign-off for the architectural phase of the layout.

- **Geometric and Legal Checks:**
  - **No Overlaps:** There must be no geometric overlaps between any macros or other fixed blocks.
  - **Grid Alignment:** All macros and I/O ports must be placed on the legal manufacturing grid defined in the technology file.
  - **Notch Avoidance:** The arrangement of macros should not create concave or "notched" regions in the standard cell area. If unavoidable, these areas must be properly managed with placement blockages to prevent congestion.
- **Connectivity and Power Checks:**
  - **I/O Port Integrity:** No I/O ports should be electrically shorted together by the pad ring structure.
  - **Power/Ground (PG) Connections:** The PG connections for all macros and pre-placed cells must be correctly hooked up to the power grid.
- **Layout Quality and Feasibility Checks:**
  - **Blockage Cleanup:** All temporary or unnecessary placement and routing blockages used during the exploratory phase should be removed.
  - **Preliminary Analysis:** An initial, quick analysis of timing and routing congestion should be run. While full timing closure and routing are not expected, this check should indicate that there is a feasible path forward and that no catastrophic issues have been created.

The Domino Effect: Consequences of a Bad Floorplan

A suboptimal floorplan does not create a single, isolated problem. Instead, it triggers a cascade of interconnected issues that compound and magnify at each subsequent stage of the physical design flow, often culminating in a design that is impossible to close. This magnification of initial errors is a critical concept to grasp.

Consider the consequences of a seemingly simple floorplanning mistake: placing two highly connected macros far apart from each other, creating a long critical path.[7]

1. **The Initial Error (Floorplanning):** The macros are placed on opposite sides of the chip.
2. **Placement Stage Consequence:** The placement tool, driven by the timing constraints, attempts to fix this long path. Its only recourse is to insert a chain of buffers along the path and to place the standard cell logic associated with this path in a direct line between the two macros. This leads to an extremely high concentration of cells in this narrow channel, creating a region of very high **cell density**.[9]
3. **CTS Stage Consequence:** The flip-flops associated with this critical path are now physically spread out. To deliver the clock signal to them with low skew, the CTS tool must build a large and complex clock tree with many levels of buffers. This results in high **clock latency** and **skew**, consuming excessive power and further degrading the timing on other paths that are now affected by the unbalanced clock.[32]
4. **Routing Stage Consequence:** The combination of the high cell density from placement and the massive number of nets trying to traverse the channel between the macros creates a severe **routing congestion hotspot**.[9] There are physically not enough metal tracks available to accommodate all the required connections. The router may fail completely, or it may create enormous detours for the wires, which dramatically increases their length and parasitic effects, causing the timing violations to become even worse.
5. **The Final Result:** The design is now a catastrophic failure across multiple domains. The initial timing problem is worse than before, the design is unroutable due to congestion, and the power consumption has ballooned due to the oversized clock tree and excessive buffering. The "small" floorplanning error has been magnified at each step into a complete, multi-domain design failure.

This example illustrates the specific downstream impacts of a bad floorplan:

- **Routing Congestion:** Caused directly by insufficient channel spacing between macros, high pin density at macro edges, or excessively high core utilization.[7]
- **Clock Tree Synthesis (CTS) Issues:** A fragmented standard cell area leads to scattered flip-flop placement, which in turn leads to large, power-hungry clock trees with high latency and skew.[32]
- **Timing Violations:** Long interconnect paths created by poor macro placement are often impossible to fix with downstream optimizations like buffer insertion, leading to persistent, unfixable timing violations.[7]
- **Power Integrity (IR Drop):** An inefficient power grid or poor placement of high-power macros will result in unacceptable voltage drops, jeopardizing the chip's functionality and performance.[7]

Flexibility and Iteration: Can We Go Back?

The VLSI design process is, by nature, iterative.[5] It is common to perform minor loops within a stage (e.g., re-running placement optimization). However, if a fundamental, "showstopper" issue is discovered during placement or routing—such as massive, unresolvable congestion or timing violations that are far from the target—the only viable solution is often to perform a major iteration by returning to the floorplanning stage.[35]

- **Pros of Iterating Back to Floorplan:**
  - **Corrects Fundamental Flaws:** It is the only way to fix architectural layout problems that are the root cause of downstream failures. No amount of placement or routing effort can fix a fundamentally broken floorplan.
  - **Enables Design Closure:** For a design that is otherwise stuck, a floorplan iteration can be the necessary step to unlock a path to successful closure.
- **Cons of Iterating Back to Floorplan:**
  - **Massive Schedule Impact:** This is the most significant drawback. Returning to the floorplan stage means discarding all the computational work performed during placement, CTS, and routing. For a large SoC, these stages can take days or even weeks of runtime on a server farm. A single major floorplan iteration can easily delay a project by several weeks or even months.[30]
  - **High Cost:** The cost of an iteration is multi-faceted. It includes the direct cost of wasted engineering hours and expensive EDA tool license time, as well as the significant opportunity cost associated with a delayed product launch in a competitive market.

The prohibitively high cost and schedule impact of major iterations are the primary drivers behind the intense industry focus on "first-time right" physical design methodologies. The goal is to invest heavily in the initial planning, analysis, and verification at the floorplan stage to minimize the probability of requiring a costly and disruptive loop back from later stages.

Deep Dive: Algorithms and Future Frontiers in Floorplanning

At its heart, floorplanning is a problem of combinatorial optimization, seeking the best arrangement of objects within a constrained space according to a set of objectives. This section delves into the computational engines that power modern floorplanning tools, exploring the classical algorithms, the data structures used to represent a floorplan, and the transformative impact of emerging technologies like machine learning, thermal-aware design,

and three-dimensional integrated circuits (3D-ICs).

The Algorithmic Engine: How Floorplanners "Think"

The task of finding the optimal placement for a set of blocks is computationally classified as an NP-hard problem. This means that for any non-trivial number of blocks, it is computationally infeasible to explore every possible arrangement to find the absolute best one. Consequently, floorplanning tools rely on sophisticated heuristic and metaheuristic algorithms that are designed to find very good, near-optimal solutions in a reasonable amount of time.[36]

Classical Algorithms

- **Partitioning-Based (Min-Cut):** This is a top-down, "divide and conquer" approach. The algorithm starts with the entire chip and recursively partitions it into two smaller regions (a "cut"), while simultaneously partitioning the netlist of components. The goal at each step is to minimize the number of nets that are cut by the partition boundary. This process continues until each partition contains a single block or a small group of cells. This method is often used for an initial rough placement.[2]
- **Simulated Annealing (SA):** This is one of the most successful and widely used metaheuristic algorithms for floorplanning. Inspired by the physical process of annealing in metallurgy, where a material is heated and then slowly cooled to reach a low-energy, stable state, SA is a probabilistic optimization technique. The process works as follows:
  1. Start with an initial, often random, floorplan configuration.
  2. Define a "cost function" that numerically evaluates the quality of a floorplan (e.g., a weighted sum of area, wirelength, and overlap).
  3. Iteratively make a small, random change to the current floorplan (a "move," such as swapping two blocks or rotating one).
  4. If the move improves the cost (a "downhill" move), it is always accepted.
  5. If the move worsens the cost (an "uphill" move), it is accepted with a certain probability. This probability is high at the beginning (high "temperature") and gradually decreases as the algorithm progresses ("cools").
     This ability to occasionally accept bad moves allows the algorithm to "climb out" of local minima in the solution space and explore a wider range of configurations to find a better, more globally optimal solution.36

Floorplan Representations (Data Structures)

The effectiveness of any floorplanning algorithm is deeply tied to the underlying data structure used to represent the geometric relationships between blocks. The choice of representation defines the algorithm's search space and the efficiency of its operations.[37]

- **Slicing vs. Non-Slicing Structures:** A key distinction is between slicing and non-slicing floorplans. A **slicing floorplan** is one that can be obtained by recursively cutting a rectangle into two smaller pieces, either with a horizontal or a vertical cut. While simple to represent and manipulate, slicing structures cannot represent all possible topological arrangements. A **non-slicing floorplan** is a more general structure that may contain arrangements (like a pinwheel of five blocks) that cannot be created by simple slicing. Non-slicing representations allow for denser packing and are essential for modern, area-constrained designs.[36]
- **Key Representations:**
  - **Polish Expression:** A classic representation for slicing floorplans. It is a string of operands (the modules) and operators (+ for a horizontal cut, * for a vertical cut) derived from a post-order traversal of a binary tree that represents the slicing structure. Its simplicity and efficiency are its main advantages.[37]
  - **Sequence Pair (SP):** A powerful and elegant representation for general non-slicing floorplans. An SP consists of two permutations of the n module names. The relative placement of any two blocks (e.g., 'A is to the left of B' or 'A is above B') is uniquely determined by their relative ordering within these two permutations. This allows the encoding of any possible floorplan topology.[37]
  - **B*-Tree:** An ordered binary tree representation for non-slicing, "compacted" floorplans (where all blocks are pushed to the bottom and left). The B*-Tree is known for its efficient operations and a more constrained (and thus smaller) solution space compared to the Sequence Pair, making it a very popular choice for modern, high-performance floorplanners.[37]

Table 2: Comparison of Floorplan Representation Models

Representation	Type	Key Idea	Complexity/Search Space	Pros	Cons	Source(s)
Slicing Tree /	Slicing	A binary tree	O(n!22n−	Simple, fast	Cannot represent	[40, 41]

Polish Expression		where leaves are modules and internal nodes are H/V cuts.	2/n1.5)	evaluation.	all possible floorplans (limited solution space).	
Sequence Pair (SP)	Non-Slicing	Two permutations of module names define relative block positions.	O((n!)2)	Represents any non-slicing floorplan.	Larger search space, more complex to pack.	[37, 40]
B*-Tree	Non-Slicing	An ordered binary tree representing a compacted floorplan.	O(n!22n−2/n1.5)	Smaller search space than SP, efficient operations, represents non-slicing.	More complex to implement than slicing trees.	[37]
TCG (Transitive Closure Graph)	Non-Slicing	Uses two directed acyclic graphs (H & V) to represent constraints.	O((n!)2)	Geometrically intuitive, supports incremental updates.	Can be complex to maintain graph properties.	[37]

The AI Revolution: Machine Learning in Floorplanning

As design complexity continues to explode, even the most sophisticated classical algorithms struggle to navigate the vast, multi-objective search space efficiently. This has opened the door for a paradigm shift: the application of **Machine Learning (ML)**, and particularly **Reinforcement Learning (RL)**, to physical design automation.[30]

In the RL framework, floorplanning is modeled as a sequential decision-making game. An AI "agent" learns to place macros one by one onto the chip canvas. After placing a block, the agent receives a numerical "reward" based on an estimate of the resulting quality (e.g., a score derived from predicted wirelength and congestion). Through a process of trial and error over millions of training "games," the agent learns a "policy"—a strategy for making placements that maximizes its cumulative reward. In effect, the agent learns the complex, multi-faceted "art" of floorplanning from experience, without being explicitly programmed with human-derived heuristics.[44]

- **Key Benefits:**
  - **Superior Quality of Results (QoR):** Pioneering work by Google demonstrated that an RL-based placer could generate floorplans for high-performance chips that were superior to those produced by human experts in terms of PPA metrics.[44]
  - **Dramatically Reduced Turnaround Time:** The floorplanning process, which can take human design teams weeks or months of manual iteration, can be accomplished by a trained ML model in a matter of hours. This represents a revolutionary acceleration of the chip design cycle.[30]
- **Current Challenges:** The field is still nascent, and significant research challenges remain. These include scaling the techniques to handle designs with millions of placeable objects, the immense computational cost and data required for training effective models, and ensuring that a policy learned on one set of designs can generalize well to new, unseen chip architectures.[43]

Emerging Challenges and Future Research Frontiers

The relentless pace of semiconductor innovation continues to introduce new challenges that require fundamental advancements in floorplanning technology.

Thermal-Aware Floorplanning

With the ever-increasing power density of modern chips, managing on-chip temperature has become a first-order design constraint. Localized "hotspots" can severely degrade performance and compromise the long-term reliability of the chip. **Thermal-aware floorplanning** addresses this by integrating a thermal model directly into the optimization loop. The algorithm's cost function is augmented to include a penalty for high peak temperatures. This drives the floorplanner to find solutions that improve heat dissipation, for example, by placing high-power-density blocks away from each other or adjacent to low-power blocks that can act as heat sinks.[46]

Floorplanning for 3D-ICs

One of the most exciting frontiers in semiconductor technology is the move from planar 2D chips to **3D-ICs**, where multiple silicon dies are stacked vertically and connected using high-density interconnects. This approach offers transformative benefits in terms of integration density, performance, and power efficiency, but it introduces a new dimension of complexity to floorplanning.[48]

The 3D floorplanning problem requires the algorithm to decide not only the (x, y) coordinates for each block but also its z-coordinate—that is, on which die layer it should be placed. Furthermore, it must manage the placement and routing of the vertical interconnects, such as **Through-Silicon Vias (TSVs)**. These TSVs are not free; they consume silicon area, introduce their own signal delay, and can act as conduits for heat, creating complex thermal challenges.[50] The co-optimization of block placement and TSV placement to balance wirelength, delay, power, and thermal constraints is a critical and highly active area of academic and industrial research.[50]

Table 3: 2D vs. 3D Floorplanning Challenges

Feature	2D Floorplanning	3D-IC Floorplanning	Source(s)
Placement Dimensions	2D (x, y) coordinates for blocks.	3D (x, y, z) coordinates; includes die/layer	[48]

		assignment.	
Primary Interconnect	Horizontal/Vertical wires on metal layers.	Wires + Vertical Interconnects (TSVs, MIVs, Hybrid Bonds).	[50]
Optimization Objectives	Area, Wirelength, Timing, Power.	Area, Wirelength, Timing, Power + **TSV Count, TSV-induced Delay, and Thermal Profile**.	[49, 50]
Key Physical Challenge	Routing Congestion.	Routing Congestion + **Thermal Hotspots** due to stacked heat sources and poor vertical heat dissipation.	[31, 51]
EDA Tooling	Mature, well-established algorithms and flows.	Emerging tools and algorithms; requires co-design and analysis of electrical, thermal, and mechanical stress.	[51]

Conclusions

Floorplanning is far more than a simple geometric packing problem; it is the architectural soul of a VLSI chip. As the cornerstone of the physical design flow, the decisions made during this stage propagate and amplify, ultimately determining the success or failure of the entire project. A deep understanding of its principles—from the foundational objectives of PPA to the strategic nuances of macro placement and power planning—is indispensable for any

physical design engineer.

The consequences of a suboptimal floorplan are severe and multifaceted, leading to a cascade of intractable issues in timing, congestion, and power that can derail a project. The high cost of iterating back from later stages places an immense premium on getting the floorplan right the first time through meticulous preparation, analysis, and verification.

Looking forward, the field is evolving at a rapid pace. The computational complexity of modern designs is pushing classical algorithms to their limits, paving the way for the transformative potential of machine learning to automate and optimize this intricate art. Simultaneously, the advent of new technological paradigms like 3D-ICs is introducing fundamentally new challenges, demanding novel algorithms that can navigate a multi-physics optimization space encompassing electrical, thermal, and mechanical domains. The continued innovation in floorplanning algorithms and methodologies will be a key enabler for the next generation of complex, high-performance integrated systems.

Works cited

1. Floorplanning in Physical Design. In the intricate world of VLSI... | by VLSIPD | Medium, accessed September 13, 2025, https://medium.com/@vlsipd4/floorplanning-in-physical-design-052e4fe2bb66
2. Floorplanning in VLSI Design: A Comprehensive Guide - Medium, accessed September 13, 2025, https://medium.com/@ifrit057/floorplanning-in-vlsi-design-a-comprehensive-guide-85a12c097ea9
3. Floorplan (microelectronics) - Wikipedia, accessed September 13, 2025, https://en.wikipedia.org/wiki/Floorplan_(microelectronics)
4. Physical design (electronics) - Wikipedia, accessed September 13, 2025, https://en.wikipedia.org/wiki/Physical_design_(electronics)
5. physical design - Floorplanning vs Placement in VLSI - Electrical ..., accessed September 13, 2025, https://electronics.stackexchange.com/questions/347898/floorplanning-vs-placement-in-vlsi
6. How do you decide best Floorplan in Physical Design?, accessed September 13, 2025, http://www.vlsijunction.com/2023/08/how-do-you-decide-best-floorplan-in.html
7. floorplan_html.pdf
8. What is are floor plans at VLSI? - Quora, accessed September 13, 2025, https://www.quora.com/What-is-are-floor-plans-at-VLSI
9. Congestion in VLSI Physical Design Flow - ChipEdge, accessed September 13, 2025, https://chipedge.com/resources/congestion-in-vlsi-physical-design-flow/
10. Timing-driven floorplanning algorithm forBuilding Block Layout ABSTRACT 1. INTRODUCTION - SPIE Digital Library, accessed September 13, 2025, https://ebooks.spiedigitallibrary.org/proceedings/Download?urlId=10.1117%2F12.235562&downloadType=proceedings%20article&isResultClick=True

11. A Study of Floorplanning Challenges and Analysis of macro placement approaches in Physical Aware Synthesis, accessed September 13, 2025, https://gvpress.com/journals/IJHIT/vol9_no1/24.pdf
12. logicmadness.com, accessed September 13, 2025, https://logicmadness.com/floorplanning-in-vlsi/#:~:text=Prevents%20Congestion,bottlenecks%20during%20the%20routing%20phase.
13. Powerplanning - VLSI Begin..., accessed September 13, 2025, http://vlsibegin.blogspot.com/p/powerplanning.html
14. Steps in VLSI Physical Design Flow - ChipEdge, accessed September 13, 2025, https://chipedge.com/resources/steps-in-vlsi-physical-design-flow/
15. Physical Design Flow I : NetlistIn & Floorplanning - VLSI Pro, accessed September 13, 2025, https://vlsi.pro/physical-design-flow-i-netlistin-floorplanning/
16. Guidelines and recommendations for macro placement - Home - Ahmed Abdelazeem, accessed September 13, 2025, https://abdelazeem201.github.io/posts/2021/04/blog-post-3/
17. System partitioning in VLSI and its considerations | PPTX | Computing - SlideShare, accessed September 13, 2025, https://www.slideshare.net/slideshow/system-partitioning-and-its-considerations/12268220
18. Guide to SoC Partitioning and Execution Success | by Anand Mirji | Medium, accessed September 13, 2025, https://medium.com/@anan.mirji/guide-to-soc-partitioning-and-execution-success-e127b49d8f5d
19. Floorplanning with pin assignment - Computer-Aided Design, 1990 ..., accessed September 13, 2025, https://mpedram.com/Papers/FP-pinAssign.pdf
20. (PDF) Floorplanning with pin assignment - ResearchGate, accessed September 13, 2025, https://www.researchgate.net/publication/3504433_Floorplanning_with_pin_assignment
21. The Ultimate Guide for Optimal SoC Floorplan - AnySilicon, accessed September 13, 2025, https://anysilicon.com/soc-floorplan/
22. Floorplanning - VLSI - Physical Design, accessed September 13, 2025, https://physicaldesignconcepts.blogspot.com/2019/01/floorplanning.html
23. POWER PLANNING - VLSI- Physical Design For Freshers, accessed September 13, 2025, https://www.physicaldesign4u.com/2020/01/power-planning.html
24. Macro Placement in Physical design | by VLSIPD - Medium, accessed September 13, 2025, https://medium.com/@vlsipd4/7-macro-placement-mistakes-that-kill-your-vlsi-floorplan-and-how-to-avoid-them-216ef8a217f8
25. Macro Placement Guidelines in the Floorplan - YouTube, accessed September 13, 2025, https://www.youtube.com/watch?v=WLX7MZWKpP8
26. Optimizing Floorplan for STA and Timing improvement in VLSI ..., accessed September 13, 2025, https://www.design-reuse.com/article/61229-optimizing-floorplan-for-sta-and-timing-improvement-in-vlsi-design-flow/

27. A floorplanner driven by structural and timing constraints, accessed September 13, 2025, https://www.cs.ou.edu/~thulasi/Misc/floorplanner.pdf
28. Chapter 21: Timing-Driven Placement - GlobalSpec, accessed September 13, 2025, https://www.globalspec.com/reference/63459/203279/chapter-21-timing-driven-placement
29. (Project3) Timing-Driven Floorplanning (Project3) Timing-Driven ..., accessed September 13, 2025, https://course.ece.cmu.edu/~ee760/760docs/760f99proj3v1.PDF
30. How Chip Floorplan Design Automation Accelerates Chip Design ..., accessed September 13, 2025, https://www.synopsys.com/blogs/chip-design/chip-floorplan-design-automation.html
31. Floor-Planning Evolves Into The Chiplet Era - Semiconductor Engineering, accessed September 13, 2025, https://semiengineering.com/floor-planning-evolves-into-the-chiplet-era/
32. Ultimate Guide: Clock Tree Synthesis - AnySilicon, accessed September 13, 2025, https://anysilicon.com/clock-tree-synthesis/
33. A doubt related to CTS in Physical Design. What to do if clock latency is more than the required value ? How to reduce clock latency in CTS stage. : r/chipdesign - Reddit, accessed September 13, 2025, https://www.reddit.com/r/chipdesign/comments/1j7582f/a_doubt_related_to_cts_in_physical_design_what_to/
34. An Efficient Timing and Clock Tree Aware Placement Flow with Multibit Flip-Flops for Power Reduction - IIITD Repository, accessed September 13, 2025, https://repository.iiitd.edu.in/xmlui/bitstream/handle/123456789/412/MT14081_JAISMINE.pdf?sequence=1&isAllowed=y
35. Place and Route: Ultimate Guide - AnySilicon, accessed September 13, 2025, https://anysilicon.com/place-and-route-ultimate-guide/
36. DIFFERENT METHODOLOGIES TO SOLVE VLSI FLOORPLANNING PROBLEM - YMER, accessed September 13, 2025, https://ymerdigital.com/uploads/YMER210862.pdf
37. (PDF) A Review on VLSI Floorplanning Optimization - ResearchGate, accessed September 13, 2025, https://www.researchgate.net/publication/328333442_A_Review_on_VLSI_Floorplanning_Optimization
38. VLSI Cell Placement Techniques. PLACEMENT BY PARTITIONING | by Yash Nikhare, accessed September 13, 2025, https://medium.com/vlsi-cell-placement-techniques/vlsi-cell-placement-techniques-8d216b50ac22
39. Floorplanning in VLSI Physical Design - YouTube, accessed September 13, 2025, https://www.youtube.com/watch?v=kZrLXwqPMis
40. Floorplanning, accessed September 13, 2025, http://www.cs.ucf.edu/~dmarino/ucf/cop3503/lectures/Ewetz-4-12-2018.ppt
41. Floorplanning, accessed September 13, 2025,

https://www.eecs.northwestern.edu/~haizhou/357/lec3.pdf
42. Practical Slicing and Non-slicing Block-Packing without Simulated Annealing - Electrical Engineering and Computer Science, accessed September 13, 2025, https://web.eecs.umich.edu/~imarkov/pubs/conf/glsvlsi04-blobb.pdf
43. Machine Learning in VLSI Design: A Comprehensive Review - ResearchGate, accessed September 13, 2025, https://www.researchgate.net/publication/383014580_Machine_Learning_in_VLSI_Design_A_Comprehensive_Review
44. implementation of machine learning in the field of vlsi ... - JETIR.org, accessed September 13, 2025, https://www.jetir.org/papers/JETIR2302311.pdf
45. Machine Learning Applications in Physical Design: Recent Results and Directions - UCSD VLSI CAD Laboratory, accessed September 13, 2025, https://vlsicad.ucsd.edu/Publications/Conferences/356/c356.pdf
46. (PDF) Simulated Annealing Based Temperature Aware Floorplanning, accessed September 13, 2025, https://www.researchgate.net/publication/220091672_Simulated_Annealing_Based_Temperature_Aware_Floorplanning
47. arXiv:2303.03779v1 [cs.AR] 7 Mar 2023, accessed September 13, 2025, https://arxiv.org/pdf/2303.03779
48. Toward Advancing 3D-ICs Physical Design: Challenges and ..., accessed September 13, 2025, https://ieda.oscc.cc/res/papers/25-ASPDAC_3D-PD.pdf
49. Multi-Objective Optimization in 3D Floorplanning - MDPI, accessed September 13, 2025, https://www.mdpi.com/2079-9292/13/9/1696
50. (PDF) TSV- and delay-aware 3D-IC floorplanning - ResearchGate, accessed September 13, 2025, https://www.researchgate.net/publication/299126509_TSV-_and_delay-aware_3D-IC_floorplanning