10 Design Closure

10.1 Introduction 10-1
Evolution of the Design Closure Flow • Introduction of Design Constraints
10.2 Current Practice 10-13
Concept Phase • Logic Design • Floorplanning • Logic Synthesis • Placement • Logic/Placement Refinement • Introduction of Clocks • Postclocking Optimizations • Routing • Postrouting Optimization and Final Signoff
10.3 The Future of Design Closure 10-28
Power-Limited Performance Optimization • Design for Variability
10.4 Conclusion 10-30

Peter J. Osler
IBM Systems and Technology Group
Essex Junction, Vermont

John M. Cohn
IBM Systems and Technology Group
Essex Junction, Vermont

10.1 Introduction

Design closure is the process by which a VLSI design is modified from its initial description to meet a growing list of design constraints and objectives. This chapter describes the common constraints in VLSI design, and how they are enforced through the steps of a design flow.

Every chip starts off as someone's idea of a good thing: "If we can make a part that performs function *X*, we will all be rich!" Once the concept is established, someone from marketing says "In order to make this chip and sell it profitably, it needs to cost $*C* and run at frequency *F*." Someone from manufacturing says "In order to make this chip's targets, it must have a yield of *Y*%." Someone from packaging says "And it has to fit in the *P* package and dissipate no more than *W* watts." Eventually, the team generates an extensive list of all the constraints and objectives that need to be met in order to manufacture a product that can be sold profitably. The management then forms a design team, consisting of chip architects, logic designers, functional verification engineers, physical designers, and timing engineers, and tasks them to create the chip to these specifications. Other chapters in this book have dealt with the details of each specific step in this design process (e.g., static timing analysis, placement, routing, etc.). This chapter looks at the overall *design closure* process, which takes a chip from its initial design state to the final form in which all of its design constraints are met.

We begin the chapter by briefly introducing a reference design flow. We then discuss the nature and evolution of design constraints. This is followed by a high-level overview of the dominant design closure constraints that currently face a VLSI designer. With this background we present a step-by-step walk-through of a typical design flow for *application-specific integrated circuits* (*ASICS*) and discuss the ways in which design constraints and objectives are handled at each stage. We will conclude with some thoughts on future design closure issues.

10.1.1 Evolution of the Design Closure Flow

Designing a chip used to be a much simpler task. In the early days of VLSI, a chip consisted of a few thousand logic circuits which performed a simple function at speeds of a few MHz. Design closure at this point was simple: if all of the necessary circuits and wires "fit," the chip would perform the desired function. Since that time, the problem of design closure has grown orders of magnitude more complex. Modern logic chips can have tens to hundreds of millions of logic elements switching at speeds of several GHz. This improvement has been driven by the Moore's law of scaling of technology which has introduced a whole host of new design considerations. As a result, a modern VLSI designer must simultaneously consider the performance of his/her chip against a list of dozens of design *constraints* and *objectives*[*] including performance, power, signal integrity, reliability, and yield. We will discuss each of these design constraints in more detail in Section 10.1.2. In response to this growing list of constraints, the design closure flow has evolved from a simple linear list of tasks to a very complex, highly iterative flow such as the following simplified *ASICS* design flow:

1. *Concept phase*: The functional objectives and architecture of a chip are developed.
2. *Logic design*: [1] The architecture is implemented in a register transfer level (RTL) language, then simulated to verify that it performs the desired functions.
3. *Floorplanning*: The RTL of the chip is assigned to gross regions of the chip, input/output (I/O) pins are assigned and large objects (arrays, cores, etc.) are placed.
4. *Synthesis*: [2] The RTL is mapped into a gate-level netlist in the target technology of the chip.
5. *Placement*: [3] The gates in the netlist are assigned to nonoverlapping locations on the chip.
6. *Logic/placement refinement*: [4,5] Iterative logical and placement transformations to close performance and power constraints.
7. *Clock insertion*: Balanced buffered clock trees are introduced into the design.
8. *Routing (or wiring)*: [6] The wires that connect the gates in the netlist are added.
9. *Postwiring optimization*: [7–10] Remaining performance, noise, and yield violations are removed; final checking is done.

We will use this reference flow throughout the chapter to illustrate points about design closure.[†] The purpose of the flow is to take a design from concept phase to working chip. The complexity of the flow is a direct result of the addition and evolution of the list of design closure constraints. To understand this evolution it is important to understand the *life cycle* of a design constraint. In general, design constraints influence the design flow via the following five-stage evolution:

1. *Early warnings*: Before chip issues begin occurring, academics and industry visionaries make dire predictions about the future impact of some new technology effect.
2. *Hardware problems*: Sporadic hardware failures start showing up in the field due to the new effect. Postmanufacturing redesign and hardware re-spins are required to get the chip to function.

[*] The distinction between constraints and objectives is straightforward: a constraint is a design target that must be met in order for the design to be considered successful. For example, a chip may be required to run at a specific frequency in order to interface with other components in a system. In contrast, an objective is a design target where more (or less) is better. For example, yield is generally an objective, which is maximized to lower manufacturing cost. For the purposes of this chapter, the distinction between constraints and objectives is not all that important and we will use the words interchangeably.

[†]See Chapter 2, Volume 1 of this handbook for a good overview of the entire design flow.

3. *Trial and error*: Constraints on the effect are formulated and used to drive postdesign checking. Violations of the constraint are fixed manually.
4. *Find and repair*: Large number of violations of the constraint drives the creation of automatic postdesign analysis and repair flows.
5. *Predict and prevent*: Constraint checking moves earlier in the flow using predictive estimations of the effect. These drive optimizations to prevent violations of the constraint.

A good example of this evolution can be found in the coupling noise constraint. In the mid-1990s (180 nm node), industry visionaries were describing the impending dangers of coupling noise long before chips were failing [11]. By the mid-late 1990s, noise problems were cropping up in advanced microprocessor designs. By 2000, automated noise analysis tools were available and were used to guide manual fix-up [12]. The total number of noise problems identified by the analysis tools identified by the flow quickly became too many to correct manually. In response, CAD companies developed the noise avoidance flows that are currently in use in the industry [13].

At any point in time, the constraints in the design flow are at different stages of their life cycle. At the time of this writing, for example, performance optimization is the most mature and is well into the fifth phase with the widespread use of *timing-driven* design flows. Power- and defect-oriented yield optimization is well into the fourth phase; power supply integrity, a type of noise constraint, is in the third phase; circuit-limited yield optimization is in the second phase, etc. A list of the first-phase impending constraint crises can always by found in the *International Technology Roadmap for Semiconductors* (*ITRS*) [14] 15-year-outlook technology roadmaps.

As a constraint matures in the design flow, it tends to work its way from the end of the flow to the beginning. As it does this, it also tends to increase in complexity and in the degree that it contends with other constraints. Constraints tend to move up in the flow due to one of the basic paradoxes of design: *accuracy vs. influence*. Specifically, the earlier in a design flow a constraint is addressed, the more flexibility there is to address the constraint. Ironically, the earlier one is in a design flow, the more difficult it is to predict compliance. For example, an architectural decision to pipeline a logic function can have a far greater impact on total chip performance than any amount of postrouting fix-up. At the same time, accurately predicting the performance impact of such a change before the chip logic is synthesized, let alone placed or routed, is very difficult. This paradox has shaped the evolution of the design closure flow in several ways. First, it requires that the design flow is no longer composed of a linear set of discrete steps. In the early stages of VLSI it was sufficient to break the design into discrete stages, i.e., first do logic synthesis, then do placement, then do routing. As the number and complexity of design closure constraints has increased, the linear design flow has broken down. In the past, if there were too many timing constraint violations left after routing, it was necessary to loop back, modify the tool settings slightly, and reexecute the previous placement steps. If the constraints were still not met, it was necessary to reach further back in the flow and modify the chip logic and repeat the synthesis and placement steps. This type of looping is both time consuming and unable to guarantee convergence i.e., it is possible to loop back in the flow to correct one constraint violation only to find that the correction induced another unrelated violation.

To minimize the requirement for frequent iteration, the design flow has evolved to use the concept of *variable detail accuracy* in which estimates of downstream attributes, e.g., wire length and gate area, are used to drive upstream optimization of timing and power dissipation. As the design evolves, these approximations are refined and used to drive more precise optimizations. In the limit, the lines that separate two sequential steps such as synthesis and placement can be removed to further improve downstream constraint estimates. In today's most advanced design closure flows, the lines between once-discrete steps have been blurred to the point that steps such as synthesis, placement, and routing can be simultaneously co-optimized [4]. The evolution from discrete stand-alone design steps to integrated co-optimizations has had a profound influence on the software architecture of design closure systems. Modern design closure suites are composed of three major components: a central database, a set of optimization engines, i.e., logic optimization, placement, and wiring, and a set of analysis engines, i.e., *static timing analysis*, *power analysis*, *noise analysis*, etc. The central database manages the evolving state of the design. The optimization engines modify the database directly, while the analysis engines track incremental changes in the database and report back on the results. By allowing a certain degree of independence between optimization engines and analysis tools, this *data-driven* architecture greatly simplifies the addition of new design constraints or the refinement of existing ones. More detail on data-driven design closure architectures can be found in the chapter on design flows [15].

The evolution of chip timing constraints provides a good overall illustration of a constraint's movement up the flow. In the first integrated circuits, analyzing performance consisted of summing the number of blocks in each path. After a couple of technology generations, hardware measurements began to show that this simple calculation was becoming less accurate. In response, performance analysis tools were extended to take into consideration that not all gates have the same delay. This worked for a time, until measured hardware performance again began to deviate from prediction. It was clear that gate output loading was beginning to become a factor. The performance analysis flow was modified to include a factor that modified the delay of a gate based on the total input capacitances of the downstream gates it drove. As performance increased further, wire delay started to gain in importance. Wire-length measures were used to calculate total wire capacitive load, which was converted into a delay component. As performance increased further, wire resistance became a factor in all interconnect, so delays were approximated using a simple, single-pole *Elmore delay* model. In the early 1990s, it was observed that the Elmore approximation was a poor predictor of wire delay for multi-sink wires, so more complex *moment matching* [16] methods were introduced. As timing analysis matured further, it became increasingly difficult to correct all of the timing constraint problems found after routing was complete. As the flow matured from *find and repair* to *predict and prevent*, it became increasingly important to accurately predict total wire delay as part of a *timing-driven* design flow. Crude models of wire delay were added to placement, in order to shorten preferentially wires on critical paths [3]. The first placement-based wire delay models used gross estimates of total wire length such as calculating the bounding box of all pins on a wire. As interconnect delay grew in importance, such nonphysical wire-length estimates proved to be poor predictors of actual wire delay. Delay estimates based on Steiner-tree approximation of routed wire topologies were introduced to predict delay better. As interconnect delay increased further, it became necessary to push wire-length calculation even further back into preplacement logic synthesis. Initially, this was done using simple *wireload models*, which assigned average wire length based on estimates of placed block size. Eventually this too proved too inaccurate for high-performance logic. By the late-1990s, *placement-driven synthesis* [4] was introduced to bias logic synthesis based on predictions of critical wire length. As the handling of timing constraints matured, it also began to contend more with other constraints. During the early days of design closure, chip-timing problems could be mitigated by increasing the drive strength of all circuits on a slow path. Doing this, however, increases active power and increases the chance of coupling noise onto adjacent wires. These design trade-offs are discussed in the context of the actual design closure flow in the chapter on design flows [15].

We will now shift our focus to learn more about the specific design constraints that are facing chip designers. Armed with this, we will then begin a detailed walk-through of a typical design closure flow.

10.1.2 Introduction of Design Constraints

Chip designers face an ever-growing list of design constraints, which must be met for the design to be successful. The purpose of the design closure flow is to move the design from concept to completion while eliminating all constraint violations. This section will briefly introduce the current taxonomy of design closure constraints, objectives and considerations, and discuss the impact and future trends for each. For the sake of this discussion we will divide the constraint types into economic, realizability, performance, power, signal integrity, reliability, and yield.

10.1.2.1 Economic Constraints

While not explicitly technical, the economic constraints governing design closure are perhaps the most important constraints of all. Economic constraints pertain to the overall affordability and marketability of a chip. Economic constraints include *design cost*, *time to market*, and *unit cost*.

- *Design-cost constraints* govern the *nonrecurring expense* (*NRE*) associated with completing a design. This includes the cost of the skilled resources needed to run the design flow, the cost of any test hardware and additional design passes which occur due to errors. These costs also include the amortized cost of facilities, computer resources, design software licenses, etc., which the team needs to complete the task. Design cost can be traded off against other design constraints such as performance and power because increased effort and skill generally will yield better optimization. Missing a design cost estimate can be a very serious problem as cost overruns generally come out of projected profit.

- *Time-to-market (TTM) constraints* govern the schedule of a design project. This includes the time required for development, manufacturing, and any additional hardware re-spins necessary to yield a functional and manufacturable part in sufficient volumes to meet the customer's requirements. Time-to-market cost can be traded off against other design constraints such as performance and power because, like increased design effort, increased design time generally yields better optimization albeit at the expense of design cost. Missing a time-to-market constraint can imply missing a customer deadline or market window. In competitive market segments, being late to market can mean the difference between huge profits and huge losses.
- *Unit-cost constraints* govern the cost of each manufactured chip. This includes the cost of the chip itself accounting for any yield loss, the package, the cost of the module assembly, and the cost of all testing and reliability screens. Unit cost is a strong function of chip die size, chip yield, and package cost. This can be traded off against design cost and time to market by allowing additional effort to optimize density, power, and yield. Missing a unit-cost constraint can make a chip noncompetitive in the marketplace.

10.1.2.2 Realizability Constraints

The most basic constraint of VLSI design is "does the chip fit and does it work?" These "realizability" constraints pertain to the basic logical correctness of the chip. Realizability constraints include *area constraints*, *routability constraints*, and *logical correctness constraints.*

- *Area constraints* are one of the most basic constraints of any VLSI design, i.e., does the chip "fit" in the desired *die size* and *package*? To fit, the total area required by the sum of the chip's circuitry plus additional area required for routing must be less than the total useable area of the die. The die size and package combination must also support the type and number of I/O pins required by the design. Because silicon area and package complexity are major components of chip cost, minimizing die size and package complexity is a major goal in cost-sensitive designs. The implication of mis-predicting capacity can be great. If the required area is over-estimated, die utilization is low and the chip costs more than it should. If the required area is underestimated, the design must move to a larger die size or more complex package, which implies more cost and additional design time.
- *Routability constraints* ensure that the resulting placement can be completely connected legally by a router. The impact of mis-predicting routability can impact average wire length, which can, in turn, affect timing. If routability is compromised enough, the designers must manually route the overflows that could not be routed automatically. In the worst case, mis-predicting routability can require that the chip be bumped up to a larger die size at great impact to cost and schedule. The challenges of this problem have been growing due to a number of factors such as increasing gate count, increased complexity of metallurgy, including complex via stack rules and multiple wire widths and heights, manufacturability constraints, and increased interrelations between performance and routing. However, the addition of extra routing layers and improvements to routing technology have alleviated this problem to some extent. Routability problems can be mitigated by adding additional area for routing, which contends with chip area constraints.
- *Logical correctness constraints* ensure that the design remains logically correct through all manipulations used by design closure. Modern design closure flows make significant use of local logical transformations such as buffering, inversions, logic cloning, and retiming to meet performance constraints. As these logical transformations have become more complex, there is an increased opportunity for introducing errors. To ensure that logical correctness is maintained, modern flows use equivalence checking to verify that the design function remains unchanged after each design closure step. Obviously, the impact of failing this constraint is that the chip no longer functions as intended.

10.1.2.3 Performance Constraints/Objectives

Once we ensure that the design will fit and is logically correct, the primary function of design closure is to ensure that the chip performance targets are met. Traditionally, chip timing has received the most focus as a design objective. Performance constraints can be divided into *late-* and *early-mode timing constraints.*

- *Late-mode timing* establishes the longest delay, or *critical path*, through all paths which sets the maximum speed a chip can run. Late-mode timing considerations can either be a constraint, i.e., *the chip* must run at least as fast as *x*, or they can be a design objective, i.e., the faster this chip runs the better. Late-mode timing constraints are enforced by comparing late-mode static timing [17] results against desired performance targets and clock cycle time. Late-mode timing constraints violations are removed by decreasing the gate or interconnect delay along critical paths. Gate delays can be reduced by decreasing logic depth, increasing gate drive strength, increasing supply voltage, or substituting gates with *low threshold* (low V_t) logic. Interconnect delay can be reduced by decreasing wiring length, adding buffers, or widening wires. Late-mode timing constraint violations can also be resolved by allowing more time for a path to evaluate through the addition of *useful clock skew.* Meeting the late-mode timing constraint for all paths is becoming increasingly difficult due to the combination of increasing chip complexity and clock speeds, combined with an increasingly pronounced "roll-off" of technology performance scaling. Both device performance and interconnect performance are failing to keep pace with the decade-long Moore's law improvement rate. More worrisome is that, as illustrated in Figure 10.1, interconnect performance is scaling even more slowly than gate performance. In fact, increases in wiring resistance are beginning to cause *reverse scaling* in which the relative interconnect delay actually increases with each new technology node. This implies more design closure effort in additional buffering and multi-cycle pipelining of long signals. Optimizations for late-mode timing constraints on one path may contend with late-mode timing constraints on other paths. For example, increasing the gate size to reduce delay on one critical path may increase gate load on another critical path. Late-mode timing optimizations also contend with power optimization. Most techniques used to optimize late-mode timing, e.g., gate sizing, buffering, and low-threshold logic substitution increase total chip power.
  The impact of missing a late-mode timing constraint is that the chip is slower than required. In most designs, missing the late-mode timing constraints implies increasing the cycle time of a machine. This may imply that the system specification must be renegotiated, or at worst case, redesigned at great cost of money and time. In some rare cases however, the final timing objectives may be negotiable, i.e., one may be able to sell a slower microprocessor for less money.

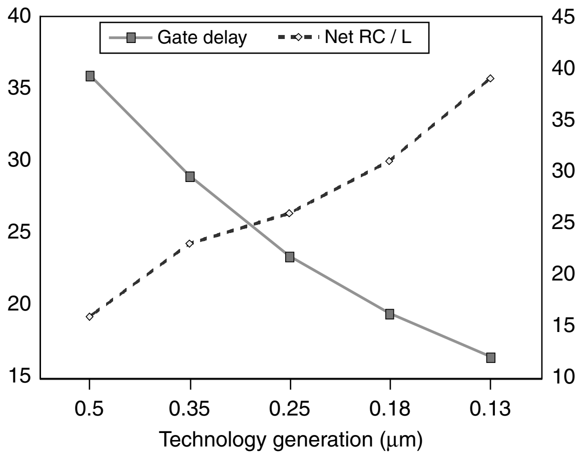


FIGURE 10.1 Interconnect and gate delay scaling trends.

- *Early-mode timing constraints* are designed to prevent a path from being too fast to be correctly captured by the capture clock. Unlike late-mode timing constraints, early-mode timing is always a constraint and never an objective. A single early-mode constraint violation means that the chip will not function at any speed. Early-mode timing issues are difficult to estimate early in the design flow. They generally must be analyzed and corrected after final clock routing is added to the design. At this point, static timing can be used to verify *hold* tests to ensure that the final logic state analyzed is stable before the capture clock fires. This testing must account for possible clock skew induced by manufacturing variations. It is necessary to run static timing over multiple *process corners*, e.g., fast wiring with slow logic, fast logic with slow wiring, to predict these cases correctly. Poor control of clock overlap can also give rise to a large number of early-mode timing violations. Early-mode timing violations are removed by adding delay elements such as buffers to fast paths. By adding circuitry, early-mode constraints contend with capacity, power, and late-mode timing constraints.

10.1.2.4 Power Constraints/Objectives

Power dissipation can either be a constraint, i.e., the chip must consume no more than x Watts, or it can be a design objective, i.e., the less power this chip uses, the longer the battery will last. As chip geometries have scaled, total chip power has become an increasingly important design closure constraint. Power can be divided into two types: *active* power and *static* power.

- *Active power* is dissipated through the charging and discharging of the capacitance of the switching nodes. Active power is proportional to the sum of $\frac{1}{2}FSCV^2$ of all switching signals on a chip, where F is the clock frequency, S the *switching factor*, i.e., the average fraction of clock cycles in which the signal switches, C the total capacitive load presented by logic fanout and interconnect, and V the supply voltage. Active power increases due to higher chip logic densities, increased switching speeds, and a slowdown in voltage scaling due to limits on scaling gate oxide thickness. Active power can be lowered by decreasing gate size, decreasing switched wire load, and decreasing switching frequency or duty cycle via clock gating. Reducing gate size to reduce active power contends directly with performance optimization, which gives rise to the essential power/performance trade-off.
- *Static* or *leakage power* is related to current that leaks through a device channel or gate even when it is turned off. Leakage power is increasing relative to active power due to the use of smaller active FET gate geometries and thinner FET gate. Leakage power is a function of supply voltage, temperature, device threshold voltage and logical state. Figure 10.2 shows the active and leakage power density trends by technology node. Static power can be mitigated by substituting in *high threshold* (*High* V_t) logic, by lowering supply voltage, or by removing power to inactive portions of the chip via *power gating*. Using High V_t logic and lowering supply voltage contends directly with performance optimization.

In high-performance applications such as server microprocessors, power constraints are generally imposed by the amount of heat that a particular chip, system, and package combination can remove before the chip temperature rises too high to allow proper function. In low-performance applications such as consumer products, factors such as battery life, packaging, and cooling expense are the major limits. Missing either an active or static power constraint can necessitate costly redesign of chip, package, or system. In the worst case, it can render a chip unusable in its intended application.

10.1.2.5 Signal Integrity Constraints

Signal Integrity constraints prevent chip function from being disrupted by electrical *noise*. Signal integrity constraints include *power integrity constraints* and *coupling constraints*.

- *Power integrity constraints* are used to ensure that the chip power supply is robust enough to limit unacceptable supply voltage variations. As chip logic elements switch, current is sourced through the chip power routing. The current must either come from off-chip or from the reserve capacity provided by on-chip capacitance provided by the diffusion structures and routing attached to the power bus and any *decoupling capacitors* (*DCaps*), which are structures added to provide small reservoirs of charge to smooth out switching transients. A switching event with net current I, which

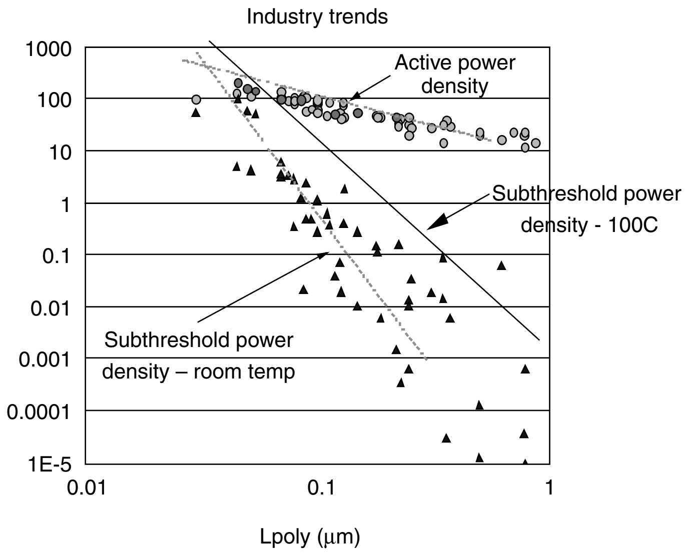


FIGURE 10.2 Active and static power trends.

cannot be supplied by local DCaps, induces a voltage drop $\Delta V = IR + L\, \mathrm{d}I/\mathrm{d}T$, as it flows through the resistance R and the inductance L of the power supply routing. These components are depicted in Figure 10.3. The voltage drop breaks down roughly into three components: *IR drop*, which is affected by power bus resistance and average current flow, *steady-state AC voltage drop*, which is affected primarily by intra-cycle current variation and local decoupling capacitance, and *switching response* which is affected primarily by the switching current variation and package inductance. When added together, the voltage drops induced by each switching event lead to potentially large variations in supply voltage both in space and time. Figure 10.4 shows a map of the spatial distribution of the average or steady-state voltage drop across a single, large ASIC calculated by IBM's ALSIM tool. The resulting voltage fluctuations cause time-varying delays across a chip. If large enough, these delay variations lead to late- or early-mode timing violations. Power voltage transients can also introduce disruptive electrical noise into sensitive analog circuitry. The relative magnitude of power voltage variations has increased as chip total power and operating frequency have increased, and supply voltages have decreased. Power supply integrity can be improved by increasing the wire width, package power/ground pin count, and via count used for power supply routing. Transient power integrity can be further improved by the judicious use of DCaps. These optimizations contend with capacity and routability constraints.

- *Coupling constraints* are used to ensure that inter-signal coupling noise does not disrupt chip timing or logical function. Noise is always a design constraint as even a single violation is sufficient to render a chip inoperable. Coupling noise occurs when a voltage transition on a noisy *aggressor* wire causes current to be injected into an adjacent sensitive victim wire through the mutual capacitance of the two wires. The injected current, ΔI, is proportional to $C\ \mathrm{d}V/\mathrm{d}T$, where C is the mutual capacitance and $\mathrm{d}V/\mathrm{d}T$ the time rate of change of the voltage transition. If the coupled signal exceeds the logic threshold on the victim wire, an incorrect logic value or *glitch* is induced in the victim circuit as shown in Figure 10.5. If the victim wire is transitioning during the coupling event, its delay is affected.

If not modeled correctly, this variation in delay can cause unexpected variations in chip performance. If these delay variations are large enough, they can lead to improper operation. Coupling has increased markedly with increased switching speeds, decreased supply voltage, decreased inter-wire spacing, and increased wire aspect ratios. Coupling can be reduced by segregating noisy and sensitive wiring, increasing inter-wire spacing, and, in extreme cases, by adding shielding wires. All of these optimizations contend with the routability constraint.

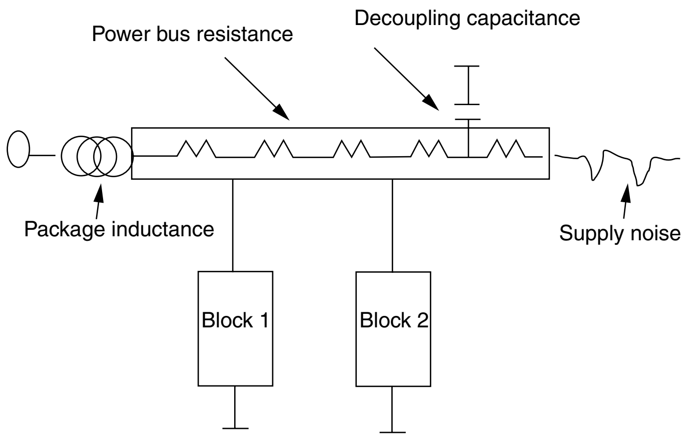


FIGURE 10.3 Power bus voltage drop components.

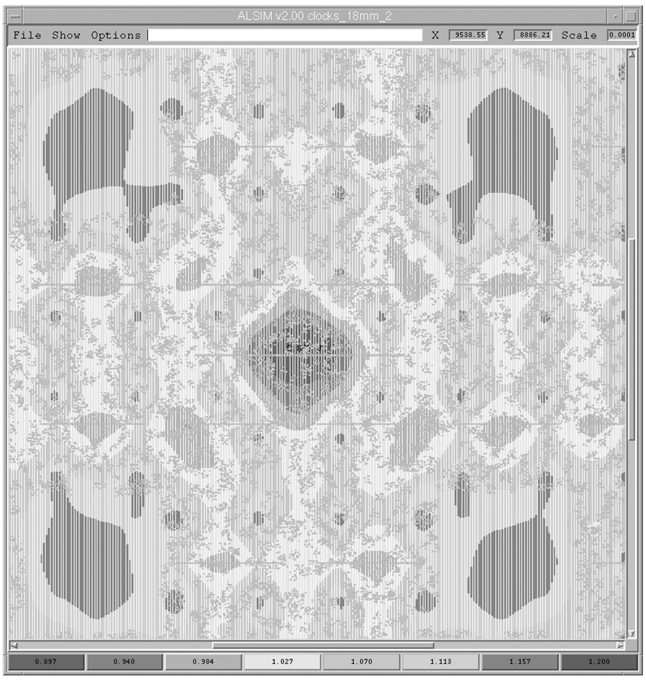


FIGURE 10.4 Power supply voltage drop map.

10.1.2.6 Reliability Constraints

Once the basic performance targets are met, we need to ensure that the chip will function properly through its required working life. Reliability concerns are related to processes that may allow a chip to function correctly immediately after manufacture but may cause the chip to malfunction at some later point in time. In the best case, this type of unexpected chip failure may present a costly inconvenience to the customer. In other, more mission-critical functions such as automotive, avionics, or security, the result of a malfunction can be a risk to life. There are many reliability factors that can affect the long-term operation of a chip. Most fit into one of the three categories: *device wear-out constraints*, *interconnect wear-out constraints*, and *transient disruption constraints.*

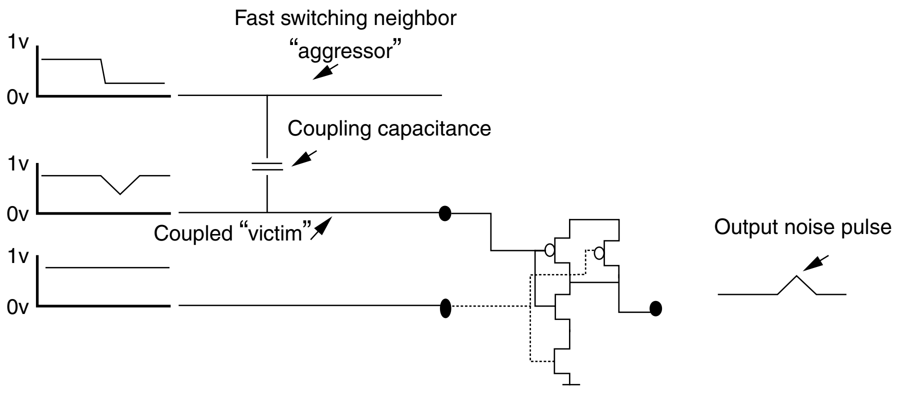


FIGURE 10.5 Coupling-induced logic glitch.

- *Device wear-out constraints* relate to mechanisms that can cause a gradual shift of electrical characteristics of a chip's transistors over time. Two common concerns are hot carrier injection (HCI) and negative bias threshold instability (NBTI). HCI occurs when electrons in the channel of a transistor are accelerated by the high electric field found near the drain of devices, which are on or switching. These highly energetic electrons are injected into the gate oxide where they create electron or hole traps. Over time these traps lead to charge build-up in the gate, which effectively causes a threshold voltage shift for the device. HCI is accelerated for devices, which have high applied gate voltages, high switching activity, and drive large loads. NBTI is similar in effect, but does not require high electric fields. It affects both switching and nonswitching devices. NBTI is accelerated by high operating temperatures and can be induced during burn-in test. The effect of these threshold shifting mechanisms is increasing over time due to the use of thinner gate oxides and lower threshold voltages required by scaling. Shifting the threshold of devices has a direct effect on the delay through a device. Delay can either increase or decrease with time depending on the nature of the injected charge and the type of device. In time, the performance shift may be large enough to cause the chip to malfunction. Device wear-out can be minimized by using the lowest switching voltage necessary and reducing load capacitance on the outputs of high-slew signals, both of which must be traded off against performance.
- *Interconnect wear-out constraints* relate to mechanisms that can cause a gradual shift in the electrical characteristics of chip interconnect with time. The principal mechanism is *electro-migration* (*EM*). EM occurs when ballistic collisions between energetic electrons and the metal atoms in the interconnect cause the interconnect atoms to creep away from their original position. This movement of metal can thin wires to the point that their resistance increases or that they fail completely. Like device wear-out, EM-induced wire wear-out can affect delay to the point that the chip begins to malfunction. In the limit, EM can cause wires to completely open, which clearly changes the function of the chip. Electromigration is accelerated by increased current densities. Additionally, the new low-permittivity dielectric materials introduced to help performance have inferior thermal characteristics. The result is an increase in wire self-heating, which further accelerates EM. EM problems can be mitigated by widening high-current wires and lowering the loads on high duty-cycle signals. These mitigations contend slightly with both routability and performance constraints.
- *Transient disruption constraints* relate to mechanisms which cause a sudden failure of devices or interconnect. The two most common mechanisms are *electro-static discharge* (*ESD*) and *soft error*

upset (*SEU*). ESD occurs when unacceptably high voltage is inadvertently presented to chip structures. This can occur either due to *induced charge build-up* during manufacturing or *a postmanufacture ESD event.* Induced charge build-up can be caused by certain manufacturing processes which involve high electric fields that can induce charge on electrically isolated structures such as transistor gates. If too much charge is induced, the resultant electric field can damage sensitive gate oxides. Induced charge ESD is mitigated by insuring that all gate inputs are tied to at least one diffusion connection. This allows a leakage path to ground, which prevents build-up of dangerously high fields. In some cases, this requires the addition of special *floating gate contacts.* Postmanufacture ESD events occur when high voltages are inadvertently applied to the chip I/O by improper handling, installation, or grounding of equipment and cables. When an ESD event occurs, the gate of any devices connected to the transient high voltage is destroyed due to the high field induced in its gate oxide. In some cases, the wiring that connects the pin to the device may also be destroyed due to the induced transient currents. Postmanufacture ESD events can be minimized by proper chip handling and by the addition of *ESD diodes* on all chip I/Os. These diodes protect chip I/O by shunting high-voltage transients to ground.

Soft error upsets are recoverable events caused by high-energy charged particles which either originate from outer space or from nearby radioactive materials. The carriers induced by the charged particle as it travels through the silicon substrate of the chip can disrupt the logic state of sensitive storage elements. The amount of charge required to upset a logic gate is dependent on its *critical charge* or Q_{crit}. As device structures shrink, the amount of charge required to cause logic upset is decreasing. SEU is currently a concern for memory arrays and dynamic logic, though some projections show that standard logic latch structures might also soon be susceptible to particle-induced soft errors. SEU is best addressed at the architecture level through the addition of logical redundancy or error-correction logic, which is not a design closure step *per se.*

10.1.2.7 Yield Constraints

For all its sophistication, semiconductor manufacturing remains an inexact science. Random defects and parametric variations can be introduced at almost any step of manufacturing which can cause a chip not to function as intended. The more the number of chips affected by manufacturing errors, the more the chips that must be manufactured to guarantee a sufficient number of working chips. Mis-predicting yield can require costly additional manufacturing, expensive delays, and possible product supply problems. In this way, yield might be considered a cost-oriented design constraint. In many cases, though, maximizing yield is considered an economic design objective. The two types of yield constraints that must be considered during design closure are *defect-limited yield* and *circuit-limited yield*:

- *Defect-limited yield constraints* relate to a product that is rendered faulty during manufacturing due to *foreign material defects* or *printability defects.* Foreign material defects result either when small bits of material accidentally fall on the chip surface or the mask reticule and interfere with the proper creation of a chip structure. Printability defects result when local geometry, chip topography, optical interference, or other manufacturing processes prevent correct printing of a desired width or spacing. These defects may take the form of unintended shorts between adjacent structures, unexpected holes in insulating materials such as device gates, or unintended opens in a conductor. Defect-limited yield can be improved by decreasing the amount of *critical area*, i.e., the inter-geometry spacing which is less than or equal to the size of likely defects. Critical area can be minimized by using *relaxed* or *recommended design rules* rather than *minimum design rules* wherever density will allow. This additional spacing contends with capacity and routability constraints.
- *Circuit-limited yield constraints* relate to yield loss due to the effect of manufacturing variations on chip performance. Despite advances in every phase of processing, there remain uncontrollable variations in the properties of the dimensions and materials of the finished product. These small variations cause identically designed devices or wires to have significantly different electrical characteristics. The most-studied variation is *across chip line-width variation* (*ACLV*), which creates

perturbations in the critical gate length of devices. ACLV has many causes including uneven etching due to local shape density variations as well as lithographic distortions caused by optical interactions between shape regions. As interconnect dimensions have shrunk, variations in wire delay are becoming equally significant. Recent data showing interconnect delay on long, unbuffered nets as high as ±60% have been reported [18]. Large amounts of this variation are introduced by both lithographic distortion and issues related to etch rate variations in processing steps such as *chemical mechanical polishing* (*CMP*). Many trends are contributing to the growing concern on parametric yield including increasingly deep *subresolution lithography* and increasingly complex manufacturing processes. In addition, decreased device and interconnect dimensions contribute to the relative impact of variation. For example, decreased device channel dimensions give rise to *micro-implant* dopant variation in which the distribution of a countable number of implanted dopant ions creates small differences in device thresholds. Similarly, gate oxides are approaching dimensions of only ten or so atomic layers. In such small configurations, a change of just one atomic layer can induce a quantized threshold voltage shift of nearly 10% as shown in Figure 10.6.

These parametric variations in turn give rise to statistical variations in design performance characteristics such as delay and leakage power. Figure 10.7 shows a typical manufacturing distribution of a large sample of *performance screen ring oscillator* (*PSRO*) circuits used to characterize the performance of a microprocessor. In the example, the PSRO circuits in the left-most tail of the distribution repre-

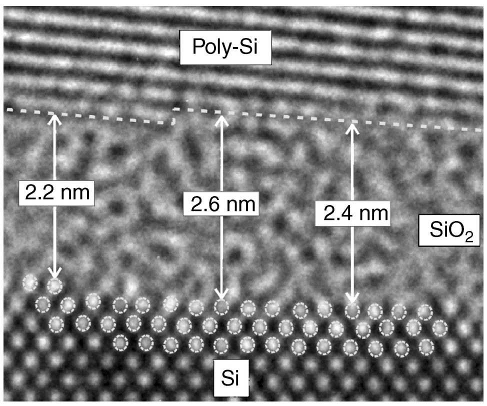


FIGURE 10.6 Gate oxide thickness variation.

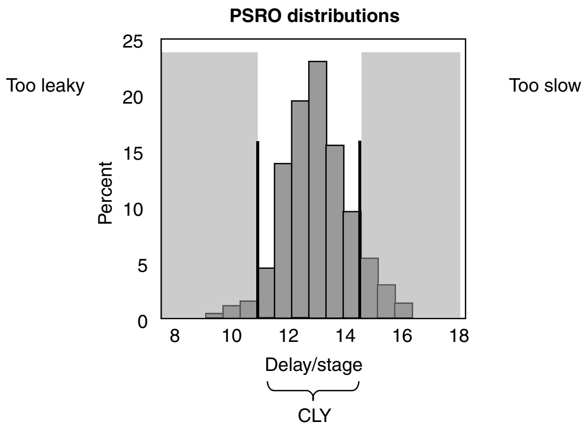


FIGURE 10.7 Ring oscillator performance distributions.

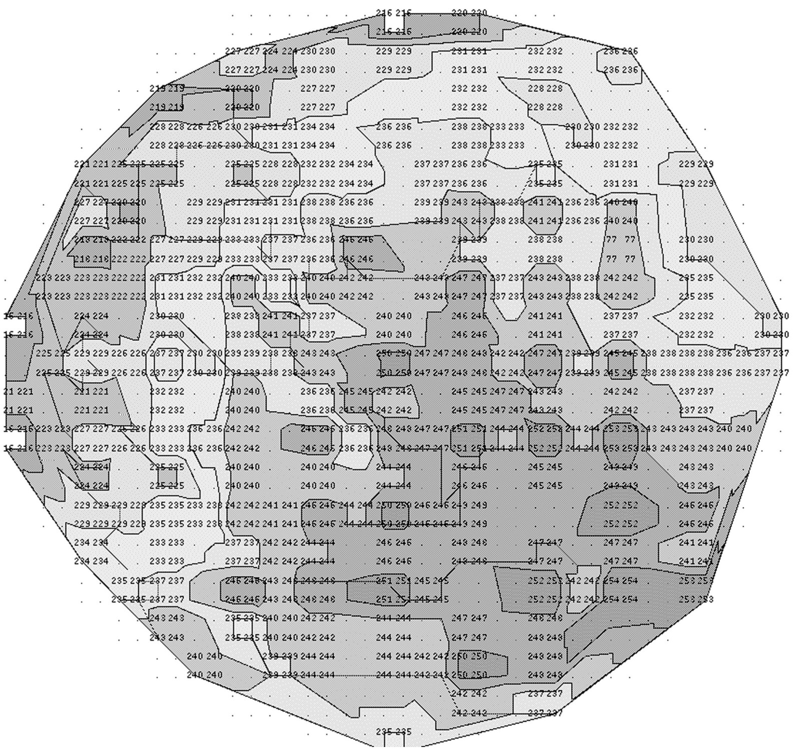


FIGURE 10.8 Wafer map of inter-die parametric variations.

sent the fastest circuits, but these same circuits violate the leakage power constraint and must be discarded. Similarly, the PSRO circuits on the right tail have low leakage, but they fail to meet the minimum late-mode timing constraint and must also be scrapped. The portion of the distribution between these two bounds is the circuit-limited yield.

These variations may be observable both when measuring the same device on different chips (*inter-die*) or between identical devices on the same chip (*intra-die*). Both inter-die and intra-die variations cause the actual performance of a given chip to deviate from its intended value. For example, Figure 10.8 illustrates inter-die variations as measured using identical ring oscillators placed on each die on a 200-mm silicon wafer. Areas of identical shading have identical frequency measurements. The total range of variation is 30%.

Parametric yield is emerging as a design closure constraint for structured logic. We will discuss how this will affect the design closure flow in our discussion of the future of design closure in Section 10.3.

10.2 Current Practice

In this section, we will examine the design closure implications of each phase of the ASIC design flow (Figure 10.9). At each phase we will examine the design constraints that are addressed, estimations and approximations that are made, and the trade-offs that can be made between constraints. We will also explore the interactions — both forward and backward — between the phases.

10.2.1 Concept Phase

The concept phase concerns itself with setting the overall scope of a chip project. During the concept phase, many aspects of the design are estimated: area, operating frequencies, voltages, power dissipation, I/O count, wiring uplift to area, yield percentages, and requirements for special processing (such as

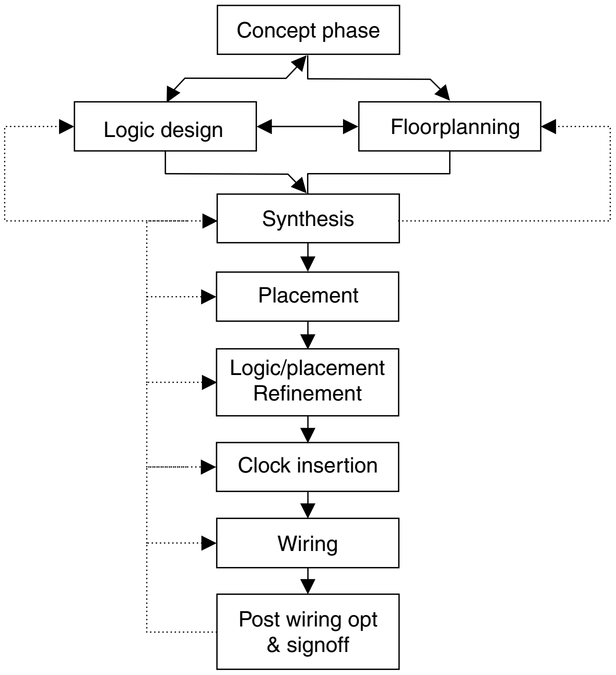


FIGURE 10.9 The design closure flow.

Embedded Dynamic RAM, analog circuits, or multiple threshold voltage [V_t] logic). All of these factors are estimated and combined into a business case: the chip cost to manufacture, its yield, its design time, its design cost, and its market price. Mis-estimating any of these factors can significantly impact the business case — sometimes to the point of jeopardizing the entire project.

Many trade-offs are made during this phase. The design is still at an abstract state and as a result changing significant aspects is easy to do: changing die sizes, making the power design more robust, adding or deleting RAMs or functional units to trade off power for speed, speed for area, and area for design time. The challenge is to estimate characteristics of the design as accurately as possible with only an inexact notion of what the design will eventually look like. The aspects of the design that need to be estimated are:

- *Amount of logic*: This is estimated based on a number of factors, including the size requirements of large reused components, e.g., memory, embedded processors, I/O circuitry, arithmetic functions, and other data-path functions. The amount of small gate-level *glue* or *dust logic* is estimated by comparing with past designs, experience, and technology insights. From the logic count we can accurately calculate required active logic area.
- *Die size*: This is derived from active logic by adding extra space for routing. First, a target placement density is chosen. Then an uplift factor based on empirical rules of achievable *wireability* is applied to the chosen placement density. A typical wireability curve is shown in Figure 10.10. This resulting value may be adjusted up based on the type of design, for example, a densely interconnected structure such as a cross-bar switch will require additional routing area. Finally, the density can be adjusted up or down by trading off design time. Higher densities can be achieved with extra effort in manual placement. Finally the limiting factor in die size may be the number of I/Os, for example, the die size of an extremely high pin-count design with simple logic functionality will be defined by the I/O count.
- *Defect-limited yield*: This factor is calculated from logic count and die size. Yield prediction is based on empirical tables as in Figure 10.11, which take into consideration logic density, die size, and technology information.

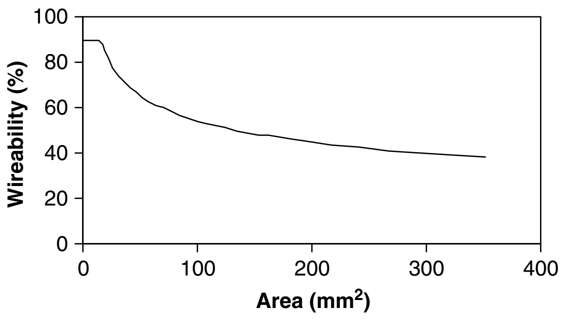


FIGURE 10.10 Chip wireability curve.

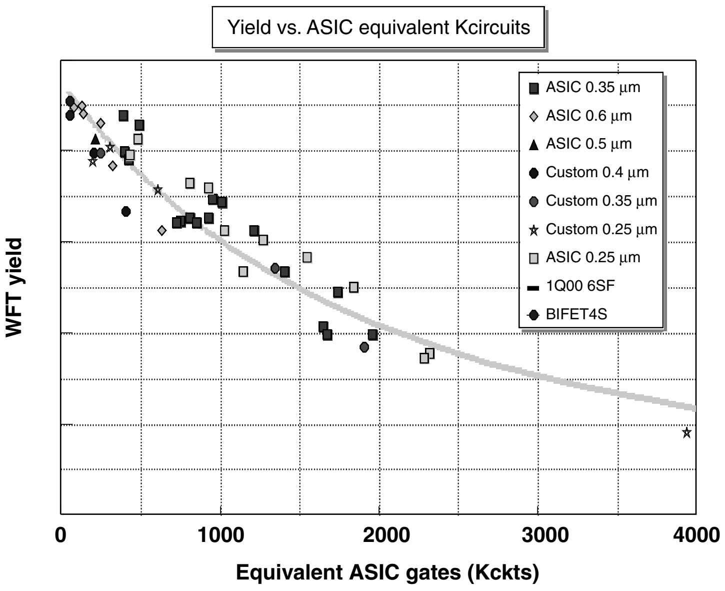


FIGURE 10.11 Yield vs. gate count plot.

- *Performance*: This is estimated by examining factors such as maximum logical path length, which is in itself derived in the absence of an actual design by experience with similar designs, information about technology parameters, *a priori* knowledge of the performance of embedded IPs, required I/O rates, estimated interconnect delays based on projected die size, voltage, and power limitations. Performance is a strong function of voltage, and as such the voltage is set to achieve performance targets defined by system or marketing constraints.
- *Power*: Active power is estimated based on performance, voltage, technology parameters, and empirical information about switching factors. Leakage power is estimated based on logic count, technology parameters, and voltage. There are significant architectural levers that can affect the power, for example, voltage or frequency scaling can be specified and significant sub-systems can be identified as candidates for clock gating.
- *Package*: Once the power is known, the package can be determined — generally the cheapest package that supports the frequency of operation, power dissipation, I/O requirements, and mechanical constraints.
- *Unit cost*: This is derived from yield, die size, and package calculations. Projected volumes from marketing are also a factor in unit-cost calculations.

In reality, the calculations are more complicated than this linear list implies. There are many trade-offs that are made, for example, architectural accommodations that favor performance at the expense of power and die size, such as adding additional arithmetic units to improve throughput. There are trade-offs balancing areas for yield by adding redundancy, such as extra word lines on array structures or area for reliability trade-offs by adding error-correction logic. Design effort is balanced against unit cost, or time to market, or performance, or power. The calculations outlined above are made and revisited as different trade-off decisions are made.

Platform-based design, where a design is made up of one or more reused large components including processors, integer, and floating-point arithmetic units, digital signal processors, memories, etc., allows for a much higher degree of accuracy of these early estimates.

After all the trade-offs have been made, the product of the concept phase is an architectural specification of the chip and a set of design constraints. These are fed forward to the floorplan and logic design phases.

10.2.2 Logic Design

The logic design phase involves implementing the register transfer logic (RTL) description of the chip based on the concept phase architectural spec. First, the specification is mapped into a hierarchical RTL structure in as clear and concise a fashion as possible and the details are filled in. Then the RTL is simulated against a set of test cases to ensure compliance with the specs. Because logic design and floorplanning are so intimately linked, the two steps generally proceed in parallel with each other.

There are three main design closure aspects that are dealt with during the logic design phase — performance, power, and routability. Because the design is so easy to change at this phase, mitigations of problems in these areas are easy to implement. However, it is difficult to measure any of these parameters directly from the RTL, which has not yet been mapped to gates, and is not yet placed or routed. There are some virtual prototyping tools that can provide quick but low-accuracy measurements based on either table-based analysis of the RTL or a quick-and-dirty encapsulated pass through synthesis, placement, and routing. In general, insight into a design's performance, power consumption, and routability is fed back to this phase from subsequent phases such as logic synthesis and placement.

The basic performance-improving RTL change involves identifying a set of critical paths, and modifying the RTL in some way to reduce logic depth. Reducing the amount of logic in a path is done by analyzing the logical content of the path and identifying portions that are not required. Owing to the hierarchical nature of design, there are often logical redundancies that can be removed by restructuring the RTL hierarchy. It is also possible to improve performance via *path balancing* or *retiming*; moving logic from one side of a long path's source or capture *latches* or *flip-flops* to the other.

There is also significant leverage for mitigation of power issues at the RTL . Chip logic can be modified to use *clock gating* which saves active power by shutting off clock switching to idle portions of logic or *power gating,* which saves both active and static power by switching off power to unused portions of the design. Both clock and power gating require careful attention to ensure that gating signals are calculated correctly and arrive in time to allow logic to stabilize as it comes out of its idle state.

In addition to these logical transformations, power/performance trade-offs can be made by using *frequency scaling* or *voltage scaling*. In frequency scaling, portions of the design that can run more slowly are segregated into more power-efficient, lower-frequency clock domains while more performance-critical logic is assigned to higher-frequency, and therefore higher power, domains. Voltage islands allow a similar power/performance trade-off by assigning less performance-critical logic to a lower voltage "island." Using a lower supply voltage saves both active and static power at the cost of additional delay. Voltage islands also require the addition of *level shifting logic* which must be added to allow logic level translation between circuits running at different voltages. The granularity of voltage islands need to be chosen carefully to ensure that the benefits of their implementation outweigh their performance, area, and power overhead.

Routability can also be optimized during the logic design phase by identifying congested regions and restructuring the RTL hierarchy such that either the congested regions are all in the same hierarchically designed unit allowing for better logical optimization and placement, or in extreme cases the RTL can be hand-instantiated and hand-placed. This is particularly useful in regular dataflow or "bit-stacked" logic.

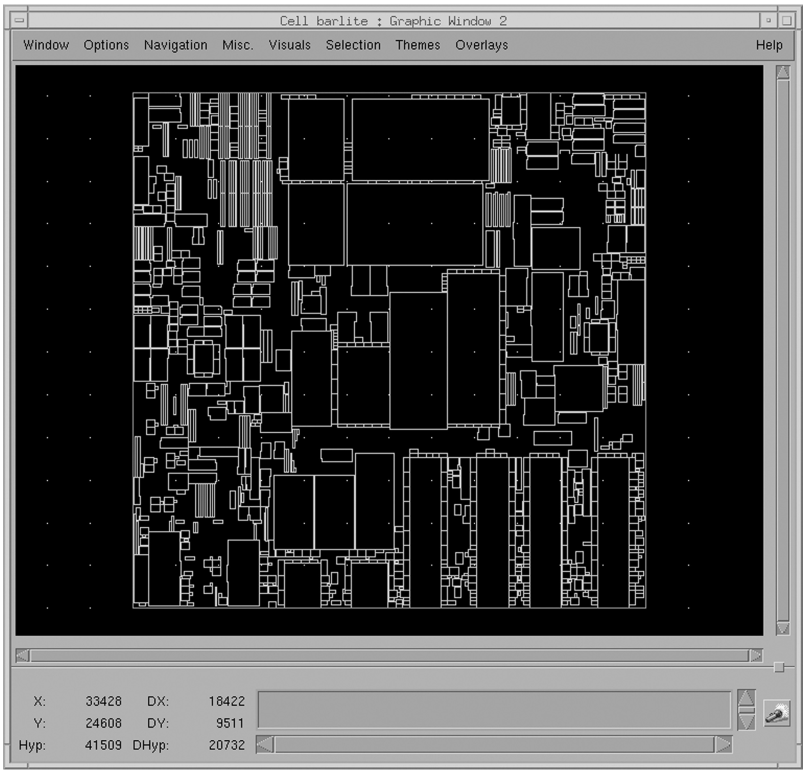
Owing to the enormous amount of simulation time required to ensure logical correctness, logic design is the most time-consuming phase of the design closure flow, and it happens in parallel with the rest of the flow. At regular intervals, the RTL is brought into a consistent state and the rest of the design closure flow is launched. The purpose of these trial runs is to give engineers responsible for subsequent steps opportunities to tune their recipes, and provide feedback to the logic designers and floorplanners about factors such as late paths and poor structure.

The product of this phase is a hierarchical RTL design and amended constraints. These are fed forward to the final stages of floorplanning and the logic synthesis phase.

10.2.3 Floorplanning

The floorplanning phase prepares a design for the placement of the standard cell logic. Owing to their tight linkage, this phase generally proceeds in parallel with the logic design phase. Design work at this phase includes placing large objects, creating power grids and some portions of the clock distribution logic, placing I/O cells and pads, and wiring the cells to the pads, and creating circuit rows in the remaining areas for the placement of the "dust logic." Large objects consist of I/O cells, Random Access Memory (RAMs), Content Addressable Memory (CAMs), register arrays, large clock buffers, DCaps, and analog circuits such as phase lock loops. In a hierarchical design, subcell pin locations are assigned as part of the floorplan, as are restricted placement areas. This step also calculates rough load values for global nets, which can be used to guide logic synthesis. Figure 10.12 shows the floorplan of a large ASIC design.

By establishing large object and I/O placement, floorplanning has a large impact on interconnect delay on critical paths. As interconnect scaling continues to worsen, the importance of floorplanning is increasing. The major design closure aspects that are treated in this phase are wireability, performance, power-supply integrity, and power. The initial large-object placement is guided by insights about their interconnectivity and expected participation in critical paths. The floorplan is refined based on feedback from subsequent steps, for



X: 33428 DX: 18422
Y: 24608 DY: 9511
Hyp: 41509 DHyp: 20732

FIGURE 10.12 ASIC floorplan showing large objects.

example, insight into critical performance and congestion issues garnered from postplacement-phase global wiring and timing runs.

Wireability is the primary design closure consideration dealt with during the floorplanning phase. The placement of the large objects significantly affects the eventual congestion of the design. Sets of large objects with a high degree of interconnection are placed close together, with central areas of reduced placement density defined to accommodate the high wiring load — the classic example of this is a cross-bar switch. Initially, the large objects are placed based on *a priori* knowledge of design interconnectivity, or from insights about connectivity garnered from floorplanning tools. There are a number of such tools that can provide assistance, ranging from the simplest that give an abstract view of the large objects, the dust logic, and their interconnectivity as in Figure 10.13, to virtual prototyping tools that do quick low-accuracy synthesis, placement, and routing and give almost real-time feedback. Later in the design closure flow, feedback from the actual placement and routing steps is used to adjust the location of the large objects to reduce congestion.

Performance problems are also addressed during the floorplanning phase. The key action is to place large objects that have timing-critical connections close together. Initially this is done based on *a priori* knowledge of the timing paths in the design. The floorplan is adjusted as either low-accuracy timing feedback from virtual prototyping tools, or higher accuracy feedback from timing runs performed after placement and global wiring become available. Floorplanning insights from these sources are used to drive restructuring of the RTL back into the logic design phase to keep clock domains and critical logic closer together.

Power supply integrity can also be addressed during the floorplanning phase. DCaps are in general placed to provide power supply isolation or specifically around particularly noisy elements, such as CAMs and large clock drivers. Early power supply design is based on factors such as the location of chip power pins, power requirements of large fixed objects, expected dust-logic densities, and locations and sizes of voltage islands. Postplacement feedback is used to augment the power-supply design if necessary. If power density is estimated to be too high in certain areas, the placement density in these areas may be reduced.

Power is addressed indirectly during floorplanning. The most important power optimization at this stage is the planning of the voltage islands introduced in the logic design section. Each voltage island requires the design and analysis of its own separate power supply and power pin routing.

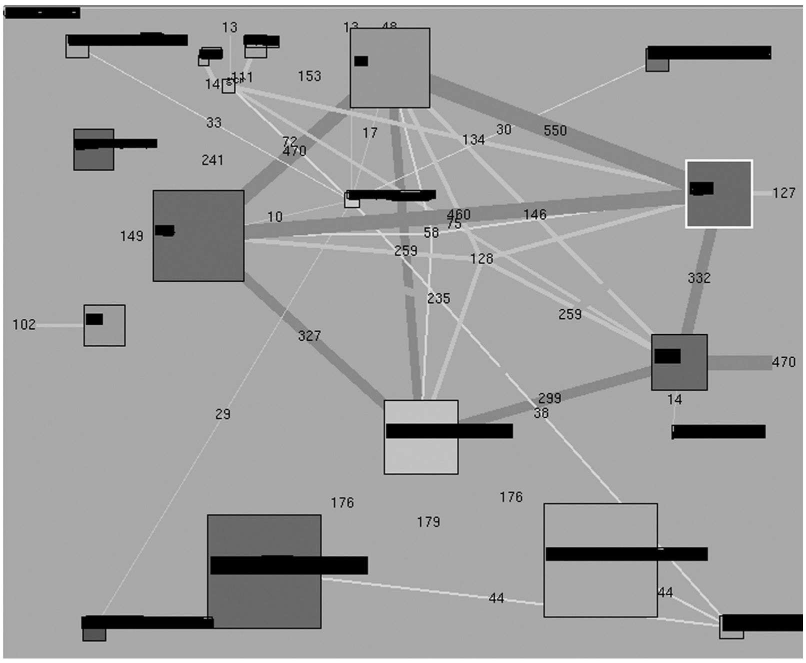


FIGURE 10.13 Interconnectivity and logic size visualization tool.

During the floorplanning phase, there are a number of trade-offs that are made. Addition of DCaps or lowered placement density regions can cause extra wire length, which impacts performance. Overdesign in the power distribution can significantly affect routability, which again can impact performance. Finding the right design points among all these factors can often take a number of iterations between floorplanning, logic design, and the subsequent design closure flow steps.

Handoff from the floorplanning phase is a detailed floorplan, including large object placements, power grids, move bounds, pin assignments, circuit rows, I/O wiring, and amended constraints. Handoff from here goes to placement.

10.2.4 Logic Synthesis

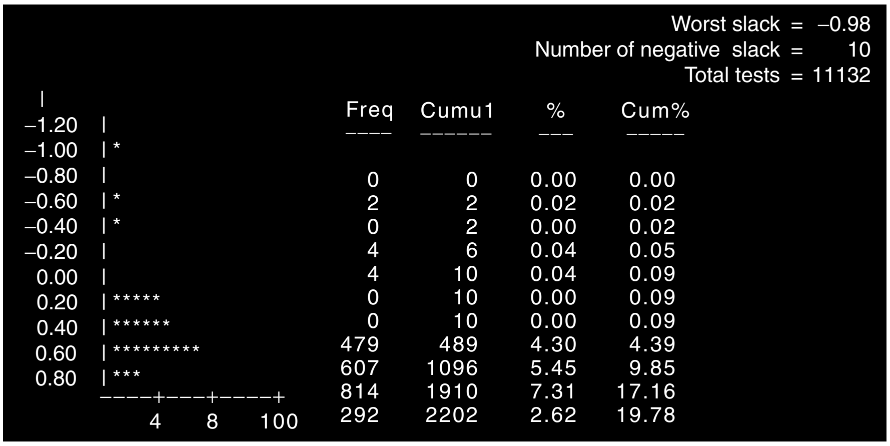
The purpose of the logic synthesis step is to map the RTL description of the design to a netlist rendered in library elements of the chosen technology meeting the performance targets with the fewest number of gates.[‡] The main design closure issues addressed in the phase are performance, power, and area. First, the RTL is compiled into a technology independent netlist format. Then, this netlist is subjected to logical analysis to identify and remove redundancies and balance cones of logic. Next, the optimized technology independent netlist is mapped into technology-dependent gates, and finally, the technology dependent netlist is timed and critical timing paths are corrected.

Because of the relative ease of running logic synthesis with different optimization targets, designers use it to explore the design space and make the best performance, power, and area trade-offs for their design. In order to achieve the area, power, and performance goals, logic synthesis applies a number of techniques to the technology-mapped netlist. In order to do this, these factors need to be measured: logic area is measured by adding up the sizes of the various gates; power is assumed to be a function of gate size — reducing gate sizes reduces power. Measuring performance is more complicated, and the logic synthesis is the first phase to rely heavily on static timing analysis. Since the placement of the dust logic has yet to be defined, *wire-load models*, usually a function of fanout, chip size, and technology, are used to estimate parasitic effects. There are subtle interactions between the wire-load-based parasitic estimation of the logic synthesis phase and subsequent placement and logic/placement refinement phases. If the wire-load models overestimate loading, then the power levels in the gates in the resulting design passed to placement will be excessively large, with no real way to recover the over-design. However, if the parasitic estimation is optimistic, then optimization in this phase will not focus on the correct problems. In general, erring on the side of optimism produces better results, especially with the advent of truly effective timing driven placement flows, to the point that many chips are now synthesized with zero-wire-load models for local signals, and load estimates for global signals derived from the floorplanning step. Since the clock distribution circuitry has yet to be added, idealized clock arrival times are applied to launch and capture clocks at latches. This step must also anticipate the impact of buffer insertion that will be performed during and after placement to prevent interconnect delay from being over estimated.

There are a number of environmental factors that need to be set in synthesis to guide static timing, such as voltage and temperature, based on information from the concept phase, guard banded for manufacturing variation and reliability factors such as HCI and NBTI. Performance is measured using incremental static timing analysis, and transforms that trade-off area, performance, and power applied. The basic approach to easing power and area problems is reducing gate size via a global repowering step, where all the gate sizes in the design are determined simultaneously. Then, critical paths are individually timing-corrected using a slack-take-down approach — an ordered list of the critical paths is created, the top path has one or more timing optimization transforms applied to it which moves the path "toward the good" in the critical path list, and the process loops back to creating a new ordered list of bad paths, etc. (see Figure 10.14).

This slack-take-down is repeated until all paths meet the performance constraint or there are no more optimizations that can be applied to the top critical path. The slack-take-down optimization scenario is

[‡] For a full description of logic synthesis techniques see Chapter 2, Volume 2 of this handbook.



```
                                                           Worst slack =  –0.98
                                                 Number of negative  slack =     10
                                                           Total tests = 11132
    |
 –1.20  |                  Freq    Cumu1      %       Cum%
 –1.00  | *                ----    ------    ---      -----
 –0.80  |
 –0.60  | *                  0         0    0.00      0.00
 –0.40  | *                  2         2    0.02      0.02
 –0.20  |                    0         2    0.00      0.02
  0.00  |                    4         6    0.04      0.05
  0.20  | *****              4        10    0.04      0.09
  0.40  | ******             0        10    0.00      0.09
  0.60  | *********          0        10    0.00      0.09
  0.80  | ***              479       489    4.30      4.39
        ----+---+-----+    607      1096    5.45      9.85
                           814      1910    7.31     17.16
            4   8   100    292      2202    2.62     19.78
```

FIGURE 10.14 Timing-slack histogram.

used extensively in the subsequent design closure phases. A short list of some of these optimizations used to improve timing on the critical path includes:

- *retiming* where logic is moved across latch boundaries to balance the amount of logic between latches;
- *rewiring* where timing critical signals are moved "later," i.e., further toward the sinks in a cone of logic to reduce the overall path delay;
- *refactoring* where a group of logic is mapped back into technology independent form and then resynthesized to preferentially shorten the logic length of a critical path;
- *cloning* where a section of logic is replicated to allow logical fan-out to be divided over more drivers; and
- *repowering* where a logic function is replaced by a similar function with higher drive strength (repowering used the other way, to reduce gate sizes, is the work-horse transform for both area reduction and power mitigation in the logic synthesis phase).

The first time that accurate gate count and timing are available in the logic synthesis phase is after technology mapping. Previous phases have all relied on gate-count and timing estimates. Surprises encountered when these numbers become available cause looping back to the floorplanning phase to reallocate space due to excess logic, or to reposition floorplanned elements for performance reasons, or go back to the logic design phase for various area and performance mitigations available in that phase.

The output of the logic synthesis phase is an area, performance, and power optimized technology-mapped netlist. This is combined with the floorplan and the most recent updated list of constraints and passed on to the placement phase.

10.2.5 Placement

The purpose of the placement phase is to assign locations to the logic that shortens timing-critical wires and minimizes wiring congestion.[§] Figure 10.15 shows the placement of a small portion of a larger ASIC design.

Until recently, placers solved congestion and performance problems by creating a minimum-wire-length placement — both min-cut or quadrisection placement techniques provide good results. As interconnect delay has become more dominant, it has become necessary to make the placement flow more timing driven. An effective timing-driven placement flow involves two placement passes: *global placement* and *detail*

[§]For a full description of placement techniques see [3].

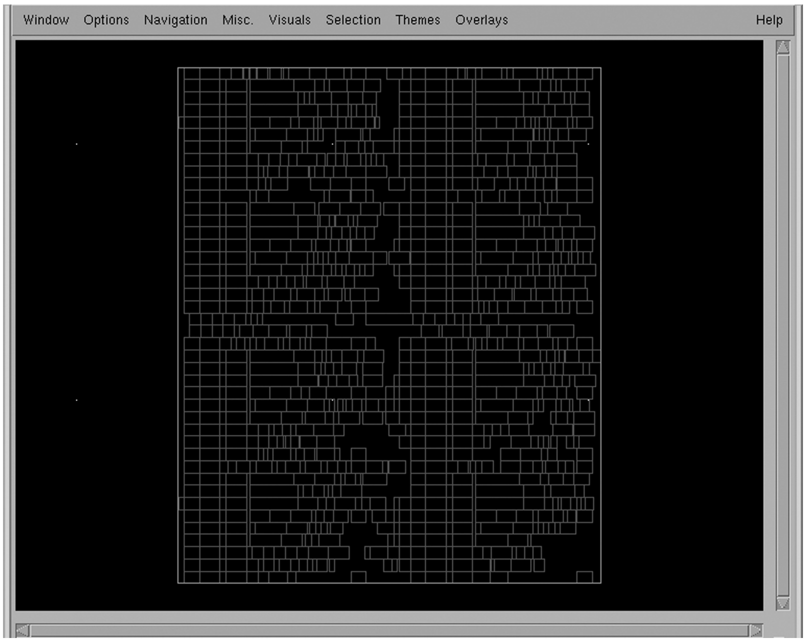


FIGURE 10.15 Logic placement.

placement. Before the global placement, the design is preprocessed to remove artifacts such as buffers on large-fanout nets, repeaters on long nets, scan and clock connections, which may bias the placement improperly. Then, a timing-independent congestion-mitigation placement is run. Next, a series of timing optimizations including gate sizing, buffer tree insertion and long wire buffering is performed. These optimizations require only analysis of slew and capacitance violations. Resized gates and new buffers and repeaters are added to the design without regard to legal placement constraints. The design is then timed using ideal clocks and interconnect delay calculated from Steiner estimates of wire topology. The timing problems that are identified are the "hard" problems that need to be fixed by the timing-driven detail placement. A set of attraction factors or *net weights* are then calculated, that bias the placement engine to move timing-critical objects closer together. These net weights are used to guide a second placement step on the optimized design. Finally, scans are reconnected and *N*-well contacts are added.

A key point in this timing-driven flow is the stability of the placement algorithm — a small change in the input to the placement engine, such as adding attractions between a small percentage of the objects being placed, must generate a small change in the result. If this is not the case, although the timing-driven placement will have pulled the timing-critical objects from the first placement closer together, a completely new set of timing problems will manifest. This same stability can help limit the disruption caused by late *engineering changes* (*EC*) to chip logic.

Another key consideration in placement is design *hierarchy.* All of the previous design steps, i.e., concept, logic design, floorplanning, and logic synthesis, rely on hierarchy to limit problem complexity. At the placement stage it is possible to either keep the design hierarchy set at logic design, or *flatten* it. If the hierarchy is kept, each hierarchical unit is placed separately, and then the units are combined to form the chip. In flat design the borders between some or all logical hierarchies are dissolved and the flattened logic is placed together. Generally, flattening a design allows significantly better optimization of performance and power during placement and subsequent design steps and requires less manual effort. On the other hand, retaining all or some of the hierarchy allows better parallelization of design effort, easier reuse of design, and faster incorporation of design changes. In addition, hierarchy is a natural way to keep the highest frequency portions of a design physically compact, which can help reduce clock skew. It should be noted that some flat placement flows provide some form of *move bounds* mechanism to manage the proximity of the most performance-critical circuits. The debate on advantages and disadvantages of flat vs. hierarchy continues in the industry and is worthy of its own chapter.

The output from the placement phase is a placed, sized netlist including buffers, repeaters, and N-well contacts which is passed to the logic/placement refinement phase.

10.2.6 Logic/Placement Refinement

The purpose of the logic/placement refinement phase is to apply placement, timing and congestion aware transformations to the logic to correct performance and power problems left over from the placement phase. The timing-driven flow outlined in the placement phase is excellent at localizing large numbers of gates to solve general performance and routability problems. However, upon the exit from that phase, there are performance-critical paths that need to be fixed individually. In addition, local power and routability issues are considered while making these optimizations. This phase has two steps. First, electrical problems such as slew limit exceptions at signal sink pins and capacitance limit exceptions at output pins are corrected. These problems are fixed by gate sizing, buffering of large fan-out nets, and repeater insertion. Second, the design is timed and optimized using the slack-take-down approach.

The timing environment for the second step includes using ideal clocks, worst-case timing rules, and parasitics extracted from Steiner wires. Also, this is the first place in the design closure flow where useful information about spatially dependent power-supply *IR* drop is available — this information is applied to the timing model where it is used to adjust individual gate delays. Feedback from this analysis can be used to guide redesign of the power supply routing.

The logic optimizations used in the second step are similar to the timing correction transforms mentioned in the logic synthesis phase; here they have been augmented to also consider assignment of locations to changed gates, impact to placement density, and wiring congestion. When logic is modified at this step, its placement must be *legalized* to a legitimate nonoverlapping placement site. This generally involves moving nearby logic as well, which can induce new timing or congestion constraint violations. These new violations are queued to be optimized. The step is complete when no more resolvable violations exist. An important part of this phase is the tools infrastructure that allows for the incremental analysis of timing and congestion caused by simultaneous changes to the logical netlist and the placement [15]. As the placement changes, previously calculated interconnects delays are invalidated, and when new timing values are requested, these values are recalculated using new Steiner estimates of routing topology. To measure congestion, an incrementally maintained probabilistic congestion map is used.

Routability can be affected in a number of different ways in this phase. Congestion information is used to assign lower placement densities in overly congested regions. As transforms are applied and logic moves, it avoids these low-density regions, mitigating congestion. This can impact timing in that gates that cannot be placed in their optimal locations due to placement density constraints, are placed further away. Another method of congestion mitigation is via congestion avoidance buffer placement, guiding the routes that long nets take by placing the repeaters along those nets in less-congested regions. Figure 10.16 shows a postplacement chip routability map.

Power and area can be recovered at this stage with the application of gate resizing. Despite the fact that the bulk placement is already done, reducing the gate sizes wherever possible provides for additional space for subsequent changes and additions, and reduced power. Also, at this point there is enough accuracy in the timing to do *mixed threshold logic* optimization. Mixed threshold logic gates have the same logical function and footprint of their standard threshold counterparts. They differ in their use of high or low threshold transistors. High threshold logic can be used to reduce static power at the cost of increased delay — off-critical-path standard-threshold-voltage gates can be replaced with their high-threshold-voltage equivalents. These substitutions are done in a "least impact to timing" fashion, where the set of possible swaps is generated, and the swap that degrades the overall timing of the chip the least is chosen. Then the set of possible swaps is regenerated based on the new design and the function is repeated until there are no more swaps that meet the criticality-plus-guardband specification. In contrast, low threshold logic can be used to decrease path delay on critical paths at the cost of increased power. Figure 10.17 shows a delay

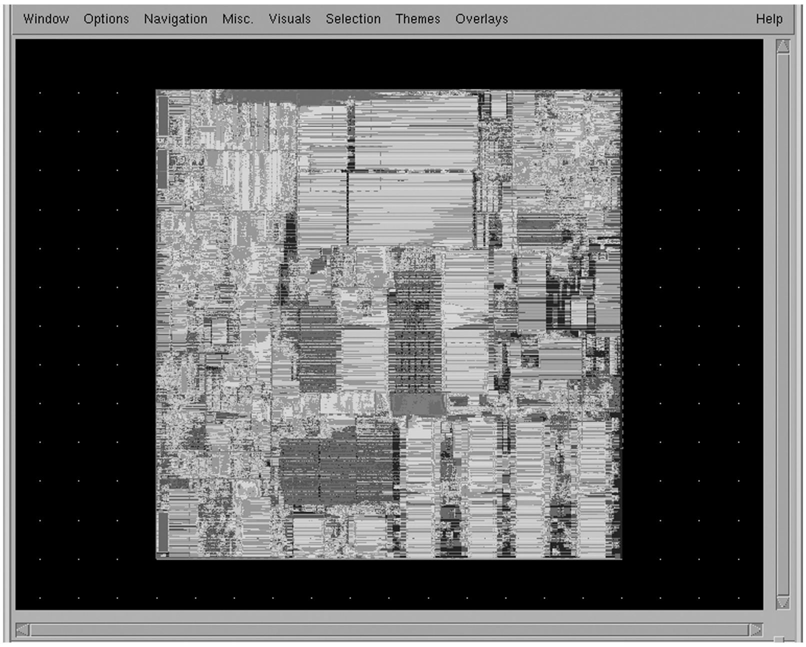


FIGURE 10.16 Chip routability map.

comparison between low and standard threshold logic. These substitutions are done generally by sequentially substituting logic on the most critical paths until all timing violations are resolved or until the amount of low threshold logic exceeds some static power-limited threshold. Low-V_t substitutions can be done at this phase or after clocking when more timing accuracy is available. Note that the introduction of multiple threshold voltages has two main costs: the first is an additional mask per threshold voltage, and the second is timing complexity associated with threshold-voltage mis-track.

The product of this phase is a timing-closed legally placed layout, which is passed to the *introduction-of-clocks* phase.

10.2.7 Introduction of Clocks

This phase inserts clock buffers and wires to implement the clock distribution logic. Figure 10.18 shows the clock distribution logic of a large ASIC design. By this phase, the amount of design optimization that can be accomplished is growing more limited. The main design closure issues dealt with in this phase are performance, signal integrity, manufacturability, and power supply integrity. The first step in this phase is the clustering of latches and the placement of the first-stage clock buffers to drive those clusters. Then, the first-stage buffers are clustered and the process repeats recursively. The clock buffers are placed with priority in the dust-logic regions, which causes dust logic to be moved out from under the buffers. After the clock buffers have been placed, the clock wires are inserted into the design.

Managing skew in the clock trees is critically important. Any unplanned skew is deducted directly from the cycle time. The buffers and wires in the clock distribution logic are carefully designed to provide as little unplanned skew as possible. To provide low skew, a number of different clock topologies can be used, including spines, H-trees, and grids. For many years *zero skew* was the optimization goal of clock design. In a zero-skew clock, the clock signal arrives at every latch in a clock domain at precisely the same moment. Recently, flows have begun using *intentional* or *useful skew* to further improve performance. By advancing or delaying clocks to latches on the critical path where there is a significant difference between the slack of the data-input and data-output signal, the cycle time of the design can be improved. Flow-based algorithms are used to recalculate the useful skew targets for all the critical latches in the design. Implementation of a

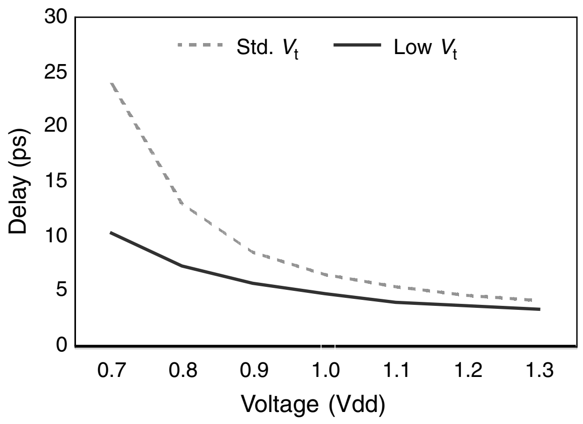
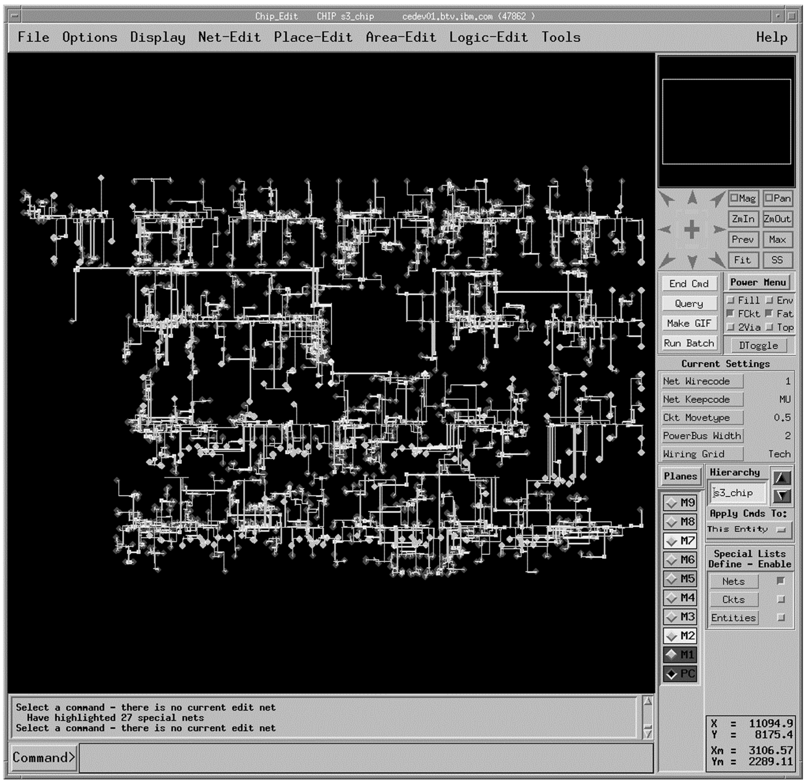


FIGURE 10.17 Effect of low threshold logic on delay.



Current Settings

```
Net Wirecode        1
Net Keepcode       MU
Ckt Movetype      0.5
PowerBus Width      2
Wiring Grid      Tech
```

```
Hierarchy   s3_chip
Apply Cmds To:   This Entity
```

```
Select a command - there is no current edit net
  Have highlighted 27 special nets
Select a command - there is no current edit net
```

```
X  =  11094.9
Y  =   8175.4
Xm =  3106.57
Ym =  2289.11
```

FIGURE 10.18 ASIC clock plan.

skew schedule is done by clustering latches with similar skew targets, and delaying clock arrival times to them by the addition of delay gates or extra wires. Calculating the skew schedule and then implementing it in this phase can yield a significant cycle-time improvement. However, if this skew-schedule is fed back to the logic-synthesis/placement/refinement phases, further gains in both performance and area can be realized.

Optimizing the clock for power is an important task, because a significant portion of a chip's active power is dissipated in the clock logic and routing. Clock buffers are tuned to ensure that the minimum gate size is used to provide for clock slew and latency requirements. Since most clock power is dissipated in the last stage or *leaves* of the clock tree, particular care is applied to minimize leaf level clock wire length. There are also interesting power trade-offs to be made around the way clock gating is handled in a clock tree. Moving clock-gates as close to the root of the clock trees as possible allows for the greatest amount of the clock tree to be turned off. However, there are generally both gated and nongated versions of the same clock required. The farther up the clock tree the gates are moved, the greater the amount of the design that must be spanned by both trees, which increases the total capacitance of the clock tree. At some point, the added power savings of moving the clock further toward the root is offset by the extra power burned in the additional capacitance of the second clock tree.

Transitions on clock wires have crisp edge rates, and as a result can cause coupling noise to adjacent wires. In addition, sequential elements can be susceptible to noise on their clock inputs. As a result, clock wires are sometimes shielded with parallel wires or given nonminimum spacing from adjacent wires to protect against coupling noise and to reduce signal dependent *clock jitter*. The switching of high-power clock buffers can also introduce significant power supply noise. To combat this, it is a good practice to surround main clock buffers with a large number of DCaps.

As manufacturing variation has become worse, it has become increasingly important to do *variation aware clocking*. Variation aware clocking techniques include the use of matched clock buffers, the use of wider metal lines, and preferential use of thicker wiring layers. In sensitive cases, it is necessary to perfectly match the order of the layer and via usage of all paths to skew-sensitive latches. It is also helpful to group latches which share critical timing paths on to the same branches of a clock tree. Timing can then use *common path pessimism removal* (*CPPR*) to account for the improved skew tolerance derived from the shared portion of the clock tree.

The output of this phase is a fully placed, presumably routable layout with fully instantiated and routed clock trees. This is passed to the postclocking optimizations phase.

10.2.8 Postclocking Optimizations

The purpose of this phase is to clean up the performance problems caused by the introduction of the clocks. There are two factors which drive the performance to degrade. First, the clock distribution logic can now be timed fully using 2.5D or 3D extractions of the real clock wires. Up until this point, all timing has been done with idealized clocks. If the idealized clocks were assigned properly with guardbands for skew, the introduction of the real clocks is fairly painless — only a few problems surface. The second factor that drives the performance to change is the replacement of the logic under the clock buffers — when the clock buffers are placed in the design, other logic is moved out from underneath them. The performance problems introduced by these two factors are fixed by applying the same timing and placement aware transforms that are applied in the logic/placement refinement phase.

In addition to fixing the performance problems caused by the introduction of the clocks, the added accuracy of timing the real clock distribution logic in this phase allows for the correction of hold-time problems. The workhorse hold-time fix is the introduction of delay gates between the launch and capture latches. Sometimes the launch and capture clocks can be de-overlapped to eliminate hold-time violations. Because this involves reworking the clock distribution logic, this is usually the option of choice only when a large number of hold-time violations can be eliminated with a single change.

To analyze properly timing problems at this stage, and to ensure that any fixes inserted do not break other timing paths, timing must be run on multiple timing "corners" simultaneously. The four standard corners are: slow-process/worst-case-voltage-and-temperature, slow-process/best-case, fast-process/worst-case, and fast-process/best-case. This requirement for multiple simultaneous timing runs further complicates the infrastructure requirements of the design-closure tool flow. As variation effects become more pronounced, this set of corners must be extended to include the possibility of *process mis-tracking* between device types and interconnect layer characteristics. Recent work in *variation-aware* and *statistical static timing analysis* provides much of the benefit of exhaustive corner analysis at much lower cost in tool run time [18].

Finally, in preparation for inevitable engineering changes, all empty spaces in the dust-logic regions of the chip are filled with gate array-style "filler cells." These cells provide unused transistors, which can be configured into logic gates by customizing one or more wiring layers. This technique can be used to implement emergency fixes for design problems found late in the flow or after hardware has been built. It is often possible to provide a patch to a design by rebuilding only one or two mask levels.

The output of the postclocking optimizations phase is a timing-closed clock instantiated presumably routable netlist. This is passed to the routing phase.

10.2.9 Routing

The purpose of the routing phase is to add wires to the design.[¶] By this phase, all of the main timing objectives of the design should be met. It is very difficult for timing problems to be fixed in routing; rather it is routing's job to deliver on the "promises" made by the wire length and congestions estimates used in synthesis, placement, and so on. There are three major design closure issues dealt with in this phase: performance, signal integrity, and manufacturability. Routing is generally broken up into two steps: *global* routing and *detail* routing. The goal of global routing is to localize all wires on the chip in such a way that all the routing capacity and demands of the chip are roughly balanced. To do this, the chip is partitioned into horizontal and vertical wiring tracks, where each track has some fixed capacity. Steiner wires are generated for all the nets, and these are laid into the tracks. Routability is optimized during global routing by permuting the track assignments to flow excess capacity out of highly utilized tracks. At this step, the wiring *porosity* of fixed objects is also analyzed to ensure correct calculation of routing capacity. As interconnect scaling worsens, it is becoming increasingly important to assign layer usage as well as track assignments during global routing. Overestimating track and layer capacity can have large impacts on the actual routed net length. The important performance-related design closure mitigation available during global routing is the preferential routing of timing-critical nets, where preidentified timing-critical nets are routed first, and to the extent possible will follow a direct path with the minimum number of jogs between source and sinks. Coupling noise can also be addressed at this phase by forcing the global router to segregate noisy and sensitive nets into different global routing tracks [19].

After the global routing step comes detail routing. In this step, the global routes are mapped into the real wire resources on the chip. In this phase, the real wire topology is determined and all wire widths, layer, via, and contact choices are made. As wires are added, wiring congestion increases and average wire length goes up. To ensure that timing constraints are met, timing-critical signals are wired first. This gives them access to the shortest wire length and preferred wiring layers. In addition, certain signals may be assigned to be wired on specific layers or specific widths for electrical reasons. For example, long timing-critical wires may be designed as wide wires on *thick* upper wiring layers to minimize resistance. Particularly skew-sensitive situations such as busses or differential signals may require *balanced routing* in which the lengths, topology, and layer usage of two or more skew-sensitive wires are matched. Wide busses are often prewired to ensure that they have balanced delay.

In addition to managing performance, wiring must optimize for coupling noise. The detail router can be guided to segregate noisy and sensitive nets [19], and to insert additional empty space between them to further reduce coupling if needed. In some cases it may be necessary to route adjacent grounded shielding nets to protect particularly sensitive signals, or to prevent interference by a particularly noisy signal. Coupling issues on busses can be improved by using *random Z-shaped routing* to prevent overly long parallel wires. Inductance effects in large busses can be mitigated somewhat by the addition of interspaced power or ground wires which serve as low-resistance current return paths.

Manufacturability can also be optimized during the routing phase. Routing can add redundant vias and contacts to improve both reliability and yield. As interconnect variability becomes more important, routing can also add wide wires and matched vias and layers to manage back-end-of-the-line variability on sensitive wires. Overall manufacturability can be further improved by adding extra *fill shapes* during routing to ensure more uniform shapes density, which improves dimensional control during lithography and CMP.

[¶]For a full description of routing techniques see [6].

If congestion estimates were inaccurate or other constraints such as coupling decreased routability, routing may not be able to embed all wires. The remaining wiring *overflows* must be manually embedded. Figure 10.19 shows a section of a fully routed ASIC.

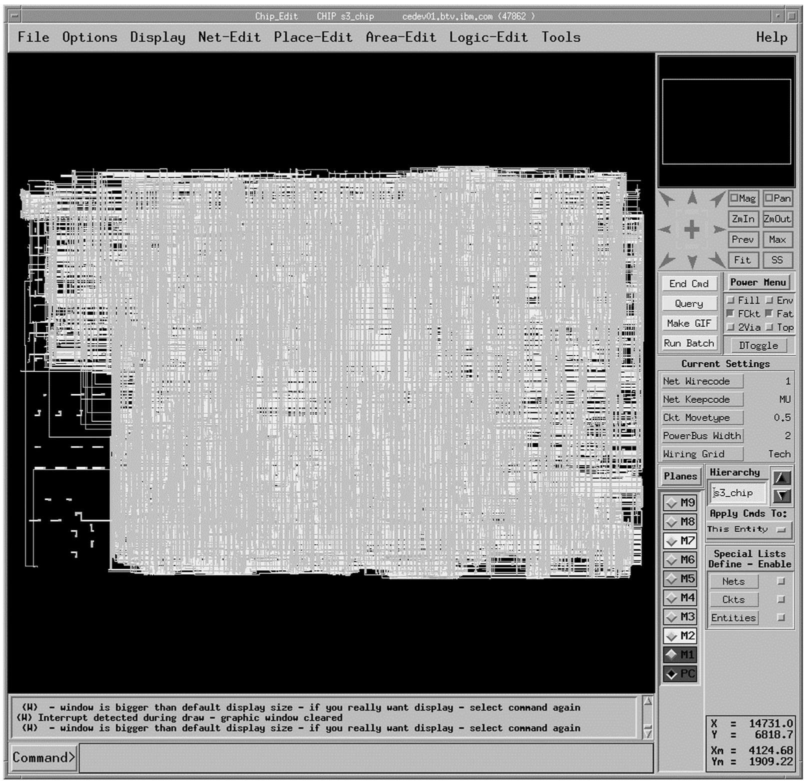
The output of the wiring phase is a wired legally placed netlist. This is passed to the postrouting optimization phase.

10.2.10 Postrouting Optimization and Final Signoff

This phase deals with optimizing out the last problems using the most accurate analyses. The main design closure issues dealt with in this phase are performance, signal integrity, yield, and reliability. At this point, since the design is nearly finished, any changes that are made must be very local in nature.

Timing is now performed using fully extracted wires with real clocks. Any timing problems that occur at this point are fixed with gate sizing, buffering, automatic or manual rerouting, and wire-widening.

Now that real wires are available, mutual capacitances can be extracted to drive noise analysis. Noise analysis uses a simple model of coupling combined with analysis of possible logic switching windows derived from timing to determine if adjacent wire switching will create timing violations or logic glitches. If errors are found they may be fixed by reducing common run length, spreading wires, adding shields, adding buffers, resizing gates to modify slew rates, and rerouting to segregate wires. Yield issues may also be analyzed and corrected in a similar manner; wire-limited yield can be addressed at this stage by increasing wire width, increasing inter-wire spacing or decreasing common run length between wires. Yield can also be optimized via wire spreading (see Figure 10.20), though this processing can complicate optical proximity processing and can interfere with wire uniformity.


Chip_Edit CHIP s3_chip cedev01.btv.ibm.com (47862)
File Options Display Net-Edit Place-Edit Area-Edit Logic-Edit Tools Help
Mag Pan ZmIn ZmOut Prev Max Fit SS
End Cmd Query Make GIF Run Batch
Power Menu Fill Env FCkt Fat 2Via Top DToggle
Planes M9 M8 M7 M6 M5 M4 M3 M2 M1 PC
Hierarchy
Special Lists Define - Enable Nets Ckts Entities

```
Current Settings
Net Wirecode      1
Net Keepcode      MU
Ckt Movetype      0.5
PowerBus Width    2
Wiring Grid       Tech

s3_chip
Apply Cmds To: This Entity

X  =  14731.0
Y  =   6818.7
Xm =  4124.68
Ym =  1909.22
```

```
 (W)  - window is bigger than default display size - if you really want display - select command again
(W) Interrupt detected during draw - graphic window cleared
 (W)  - window is bigger than default display size - if you really want display - select command again
Command>
```

FIGURE 10.19 ASIC chip routing.

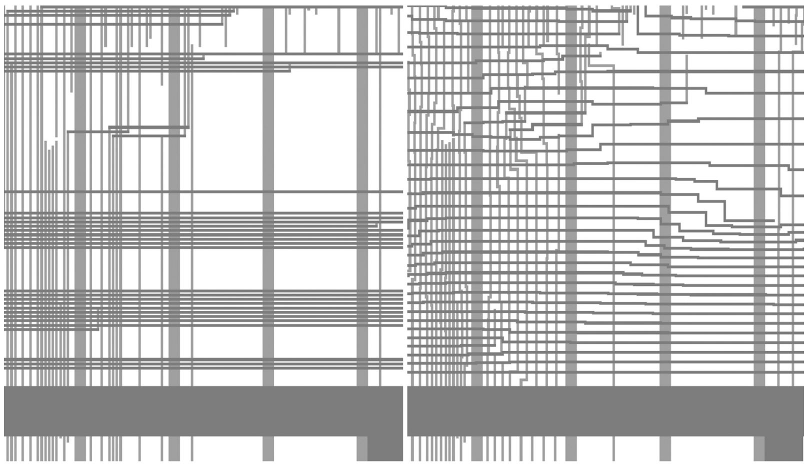


FIGURE 10.20 Before and after wire spreading.

Any timing problems, coupling or yield constraint violations found at this point can be fixed manually or by automatic routing-based optimization, which uses the same logic, placement, and timing optimization framework described in logic/placement refinement. In such an automated flow, wires with constraint violations are deleted and queued for rerouting. Final closure is performed using transformations that adjust gate sizes, do minimal logic modification, and minimally adjust placement density on the offending wire's logical and physical neighbors.

In addition to these final optimizations, there are a number of final checks and optimizations that are run on the design. One of these is ESD *checking and optimization*. The first of these is the *floating gate detection* and the addition of any necessary floating *gate contacts* to ensure that all gates in the design electrically connect to at least one diffusion. The second is *wiring antenna* detection and correction, which remove any unterminated wiring segments that might be left in the design by previous editing. Antennas can also cause ESD problems. Reliability checks are performed to ensure that no wire violates its electromigration limit and any wire that does is modified via wire-widening and load adjustment. Finally, design rule checking and logical-to-physical verification are performed.

Final timing sign-off is performed using *variation aware timing* by checking the design against an exhaustive set of process corners. This exhaustive checking tests the design for robustness to possible mistracking between layers, device strengths, multiple supply voltages, etc. This guarantees conservatively that the design will be manufacturable over the entire process window.

The product of this phase is a fully placed, routed, and manufacturable design, ready to be sent to the mask house.[**]

10.3 The Future of Design Closure

The next big challenges in design closure will be dealing with the increased importance of *power-limited performance optimization* and the increased need to do *design for variability*.

10.3.1 Power-Limited Performance Optimization

Current design closure flows treat performance as the primary optimization objective, with constraints such as power and area handled as secondary concerns. As both active and static power continue to increase, the design flow will have to be modified to treat the power/performance trade-off as the primary

[**] It's time to go home and have a beer!

optimization objective. Because power is best addressed early in the flow, the most leverage will come from advances in the concept and logic design phases. The most important design closure advances in this area will need to be the creation of power-oriented design exploration and optimization flows. To enable these flows, the industry will need to develop more accurate early power prediction techniques.

In the logic synthesis, placement, routing, and refinement stages, the flow will have to be modified to manage the power impact of every optimization rather than sequentially optimizing for performance, and then assessing the power impact. The flow will also have to be modified for more aggressive power management design techniques including wider use of dynamic voltage scaling, and wider use of power gating. Design closure flows will also need to be extended to handle emerging circuit-oriented power management technique. These techniques are likely to include the use of new circuit families and dynamic *body bias*, a technique that lowers static power by dynamically adjusting the gate to body bias.

10.3.2 Design for Variability

In Section 10.1.2.7, we mentioned the trend of increasing parametric variability for both devices and wires as chip geometry continues to shrink. As this has occurred, the relative amount of performance guardbanding has had to increase. This forced conservatism effectively reduces the amount of performance gain extractable from new technology nodes. As we move into 65-nm design and below, we are now entering a phase in design closure where parametric variability is becoming a first-order optimization objective. Design closure for variability can be addressed in two distinct ways. The first is *design for manufacturability* (*DFM*); the second is by doing *statistically driven optimization.*

Design-for-manufacturability concerns have been slowly working their way into the design closure flow. DFM optimizations involve constructively modifying the design to minimize sources of variability in order to make the design more robust to manufacture variations. There are many constructive DFM techniques, which have been or are currently being automated as part of the design closure flow. These include:

Matching devices. Identical circuits tend to track each other better than dissimilar circuits in the face of manufacturing variation. Design closure flows can exploit this by using identical buffering in skew-sensitive circuits such as clocks.

Variation aware routing. Automatic routers are being modified to create more variation-tolerant routing. Optimizations include: the use of wide wires on variation-sensitive routing, the use of geometrically balanced (i.e., matching topology, layer, and via usage) routing for critically matched signals, and the design of maximum common subtrees to reduce process-induced delay variation in skew-sensitive routes such as clocks.

Geometric regularity. Designs with a high degree of device and wiring regularity tend to have lower levels of dimensional variation. This is because both optical lithography and etch processes such as *CMP* respond better with uniform shapes densities. Regularity can be imposed at many levels of the design. At the cell level, regularity can be achieved by placing all critical geometry such as transistor polysilicon gates on a uniform grid. At a placement level, the flow can ensure that all gates see essentially uniform density on critical polysilicon and contact layers. At a routing level, the flow can enforce a fixed routing grid and uniform routing density through careful route planning and the selective addition of routing fill. New regular design styles such as *structured ASIC* [20–22] are being introduced to reduce sensitivity to manufacturing variation.

Adaptive circuit techniques. Chips will include greater number of *adaptive variability management* circuits. These circuits will either use automatic feedback or digital controls to "dial out" variability. For example, adaptive control could be used to cancel a manufacturing-induced offset in a differential receiver, or could be used to adjust out delay variation in a critical signals timing.

In addition to these constructive DFM techniques, a new paradigm for managing variability based on the availability of new *statistical static timing* (*SST*) [18] is emerging. Previous static timing tools characterized gate and wire delays based on specific parametric assumptions: i.e., *best*, *worst*, *nominal*; statistical timing tools, in contrast, characterize delay using statistical delay distributions derived from measured hardware. Rather than calculating path delay as the sum of all the worst-case (or best-case) delays, statistical timing calculates a probability distribution for delays. This distribution indicates the likelihood of a given path having a specific delay. By calculating this measure for all paths, one can determine the percentage of manufactured chips that will run at a given speed. Calculating delay in this way avoids the compounding conservatism of using the worst-case delay of all elements for the circuit. Combining statistical timing with an automatic design closure flow will allow designers to make yield vs. performance trade-offs. By optimizing a design to an acceptable circuit-limited yield value rather than an exhaustive set of worst/best-case parameters, a designer can greatly reduce the performance lost to over-design. Though these techniques are still at the early stages of deployment, we are certain that DFM and statistical design will be the dominant themes in design closure for the next several years.

10.4 Conclusion

We have defined the design closure problem, and the current design closure constraints and how they have evolved. We then explored how these constraints are addressed throughout the design flow. New constraints will continue to evolve and the closure problem will continue to grow more complex and more interesting.

Acknowledgments

The authors would like to acknowledge Dr. Leon Stok, Dr. Ruchir Puri, Dr. Juergen Koehl, and David Hathaway for their helpful input and feedback on this chapter.

References

[1] Design and Verification Languages, chap. 14, this handbook, Vol. 1.
[2] Logic Synthesis, chap. 2, this handbook, vol.2.
[3] Digital Layout Placement, chap. 5, this handbook.
[4] Trevillyan, L., Kung, D., Puri, R., Reddy, L., and Kazda, M., An integrated design environment for technology closure of deep-submicron IC designs, *IEEE Design Test*, 21, 14–22, 2004.
[5] Shenoy, N., Iyer, M., Damiano, R., Harer, K., Ma, H.-K., and Thilking, P., A Robust Solution to the Timing Convergence Problem in High-Performance Design, *International Conference on Computer, Design*, San Jose, CA, May, 1999, pp. 250–254, this handbook, vol.2.
[6] Routing, chap. 8, this handbook, Vol. 2.
[7] Design and Analysis of Power Supply Networks, chap. 20, this handbook, Vol. 2.
[8] Noise Considerations in Digital IC's, chap. 21, this handbook.
[9] Design for Manufacturability in Nanometer Era, chap. 19, this handbook.
[10] Design Rule Checking, chap. 17, this handbook, Vol. 2.
[11] Shepard, K., and Narayanan, V., Noise in deep submicron digital design, *ICCAD 1996*, pp. 524–531.
[12] Shepard, K., Narayanan, V., and Rose R., Harmony: static noise analysis of deep submicron digital integrated circuits, *IEEE TCAD*, pp. 1132–1150, August 1999.
[13] Shepard, K., and Narayanan, V., Conquering noise in deep-submicron digital ICs, *IEEE Des. Test Comput.*, 15, 51–62, 1998.
[14] *International Technology Roadmap for Semiconductors, 2004 Update for Design.* http://www.itrs.net/Common/2004Update/2004_01_Design.pdf
[15] Design Flows, chap. 1, this handbook, Vol. 2.
[16] Pillage L.T., and R.A. Rohrer, Asymptotic waveform evaluation, *IEEE Trans. Comput. Aided Des.* (1991 IEEE Best Paper Award), 352–366, 1990.
[17] Static Timing Analysis, chap. 6, this handbook, Vol. 2.
[18] Visweswariah C., Death, Taxes and Failing Chips, *IEEE/ACM Design Automation Conference*, Anaheim, CA, June 2003, pp. 343–347.

[19] Stohr, T., Alt, H., Hetzel, A., and Koehl, J., Analysis, Reduction and Avoidance of Cross Talk on VLSI Chips, *International Symposium on Physical Design*, Monterey, CA, 1998 (ISPD98).
[20] AMIs XpressArray structured ASIC: http://www.amis.com/asics/structured_asics/XPressArray.html
[21] eASIC http://www.easic.com/
[22] Pileggi, L., Schmit, H., Strojwas, A.J., Gopalakrishnan, P., Kheterpal, V., Koorapaty, A., Patel, C., Rovner, V., and Tong, K.Y., Exploring Regular Fabrics to Optimize the Performance-Cost Trade-off, *Proceedings of IEEE Design Automation Conference*, Anaheim, CA, June 2003.
[23] Power Analysis and Optimization from circuit to Register Levels, chap. 3, this hand book.