Jens Lienig, Juergen Scheible
Fundamentals of Layout Design for Electronic Circuits
ISBN 978-3-030-39283-3, Springer 2020

5.4 Verification

When physical design is complete, the layout must be fully verified. This verification step validates functional correctness and design manufacturability. Effectively, the main objective of verification is to ensure the correct functionality of the design and to minimize the risk of problems occurring during manufacturing.

Designing an electronic system is both challenging and time-consuming. Issues can be encountered during the process that jeopardize or completely scupper the design. Research and practical experience have shown that if a fault is left undiscovered it becomes far more costly to correct at a later stage. In fact, the cost of correcting a fault increases by an order of magnitude for every layout-processing step in which it remains undiscovered. The immediate goal is therefore to discover faults as early as possible in the flow. This requires multiple verification steps.

The product cannot be tested or validated during the design process as it does not yet exist at this stage. However, during design reliable and informative criteria need to be specified for checks that will occur in subsequent stages. These criteria can be derived from the technological and functional constraints; we describe this operation in Sect. 5.4.1, and then elaborate on the individual verification techniques in subsequent sections.

As visualized in Fig. 5.31, any comprehensive design verification process includes the following checks: *formal verification* (Sect. 5.4.2), *functional verification* (Sect. 5.4.3), *timing verification* (Sect. 5.4.4), and *physical verification*. Physical verification can be further split into *geometric verification* (Sect. 5.4.5) and verification based on *extraction* and *netlist comparison* (LVS, Sect. 5.4.6). As a physical

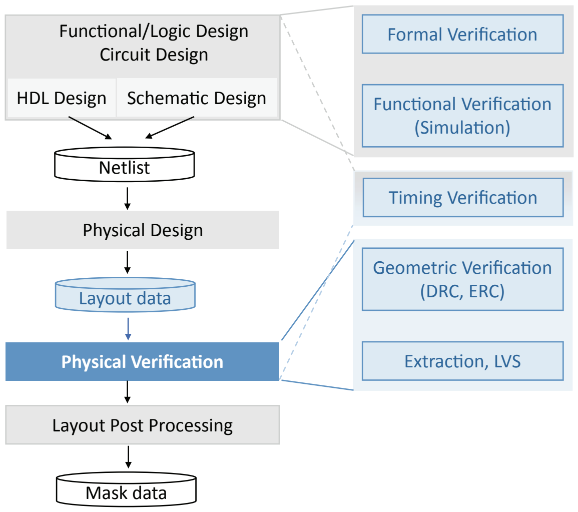


Fig. 5.31 Verification steps (right) that are discussed in Sects. 5.4.2–5.4.6

designer needs to know which verification steps have been applied prior to layout generation, we cover these “early-on” verifications (formal, functional and timing) first, before presenting in detail physical verification procedures (DRC, ERC, Extraction, LVS).

5.4.1 Fundamentals

As we know, electronic systems are designed in many steps due to their high complexity. Each design step produces a new intermediate result that brings us ever closer to the final design. These intermediate results must be checked for violations of the specified technological and functional constraints. The functional constraints are derived from the project, including some that are defined at project kick-off, one example being the signal-to-noise ratio of a signal defined in the specification. Others could be defined during the design, an example here being the maximum permissible IR drop in a line. The technological constraints are specified by the fab (aka the fabricator); they are based on the manufacturing limits of the technology being used.

We noted in Chap. 2 (Sect. 2.3.4) that it is critical that the result of an IC design, i.e., the finished layout, is correct, because if an IC with inherent design flaws is fabricated, serious financial losses would be incurred by the defective chip. Violations are therefore checked for with advanced computer-based tools. To enable this to happen, the relevant constraints must be converted to a format that can be used by these verification tools.

The constraints are converted in a two-stage mapping process. Figure 5.32 illustrates this for all constraints. Constraints based on physics/reality are converted in the first mapping to formal rules or standards, which are formulated in a readable format as text, for example. This formal description of the constraints is a meta format (middle column in Fig. 5.32). These meta descriptions are then converted to the data formats needed for the verification tools (right column in Fig. 5.32).

Following the second mapping, the constraints are available for the verification tools as a *technology file*, for example (right-hand column in Fig. 5.32). A verification tool, such as a DRC tool, can then check the intermediate result of a design step automatically for compliance with preset criteria. If a violation is found during the check, it is labeled as an *error*.

Figure 5.32 shows clearly the significance of the different categories of constraints. (We introduced these categories in Sect. 4.5.2.) The technological and functional constraints are converted into the physical design domain (right column in Fig. 5.32) to enable an automatic check. Compliance with these checks determines whether a chip or a PCB can be manufactured, and whether or not it is fit for purpose. The purpose of the remaining constraint category, the design-methodology constraints, is to enable this mapping and thus automatic processing.

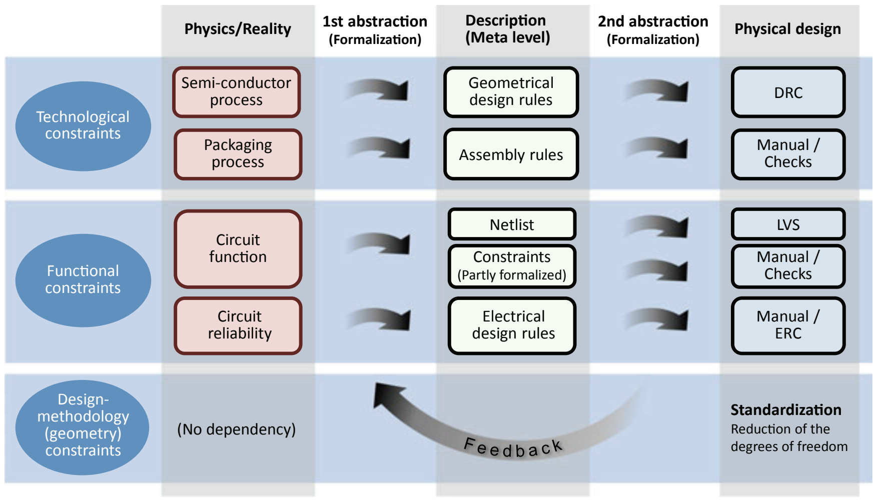


Fig. 5.32 Making technological and functional constraints accessible to verification tools requires two conversions. The first conversion results in formal rules (middle column), which are then converted into the data formats of the respective verification tool (right column). Design-methodology constraints provide a feedback from the second to the first abstraction; they also limit the degrees of freedom within physical design

Let us consider an example: DRC tools need the design rules written in a specific format. Complex design rules can be described in this format by routines comprising numerous commands (examples are given in Chap. 3, Sect. 3.4.3). Although the syntax and semantics of these languages are very powerful, they are still limited. Hence, the rules need to be described in such a manner in the meta level that they enable the second mapping in the physical design without any information loss.

Design-methodology constraints are restrictions that must be adhered to when the design rules of the meta level are drawn up so that these rules can be programmed subsequently. Hence, the design-methodology constraints in Fig. 5.32 represent a feedback from the second mapping to the first. Within physical design (i.e., the result of the second mapping in Fig. 5.32, that is, the column on the right), design-methodology constraints promote standardization by restricting the degrees of freedom.

Design engineers should always remember that both mappings are abstractions, as the real constraints (shown on the left in Fig. 5.32) are formalized by these mappings. This means that a formal constraint, such as a physical design constraint, is not equivalent to the real requirement in the technology or in the physical design solution. We will illustrate this with an example of a technological constraint.

It may not always be possible to exactly model a technological constraint by a design rule because of the complexity of modern technologies. In these cases, a small margin of safety is introduced into the design rule w.r.t. the real requirement. This causes layout results to fail the DRC even though they have not violated the real technological constraint. This type of error is called a *dummy error* or *false error*.

The impact of dummy errors can be waived when interpreting DRC results, and the layout structure is left unchanged. However, the design engineer must be very experienced and understand exactly the underlying technology to properly handle these cases.

Geometric rules within the physical design domain (right column in Fig. 5.32) are typically conservatively formulated to ensure that the real requirements (i.e., the original technological constraints) are met with certainty. Technological constraints therefore tend to be met with a safety factor. In fact, manufacturability is typically one hundred percent assured with a perfect DRC result. Contrast this with the situation for functional constraints, where they can only be modelled in part in modern design environments. As such, they cannot be completely checked.

This is an important observation as it is the reason for further checks, such as simulations, and also one reason why a valid *verification* ("Has the circuit been correctly designed?") can nevertheless result in an invalid *validation* ("Has the correct circuit been designed?") (Chap. 4, Sect. 4.4, cf. Fig. 4.18).

Before we discuss different verification methods in physical design in the following subsections in detail, we first classify them. Table 5.1 presents an overview of

Table 5.1 Different options for verifying an electronic circuit as presented in the following subsections. For the sake of completeness, we also include testing, i.e., to validate a circuit design from a customer perspective

Check	What is checked?	How is it checked?	Method
Model check	Logical characteristic (Assumption is true?)	Mathematical models	Formal verification
Equivalence check	Logical equivalence of two descriptions	Mathematical models	Formal verification
Simulation	Circuit behavior versus specification	Virtual experiment (stimuli and output)	Functional verification
DRC (OPC, RET)	Layout versus technological constraints (manufacturability)	Geometrical design rules	Geometric verification
LVS	Layout versus schematic	Netlist extraction from layout, rule based	Geometric verification
PEX (plus simulation)	Impact of parasitics on circuit behavior	Parameter extraction from layout, rule based; followed by simulation	Geometric and functional verification
ERC	Layout versus electric process boundaries (reliability)	Connectivity extraction from layout, rule based	Geometric verification
Testing	Compliance for practical usage	Real experiment, customer checking	Validation

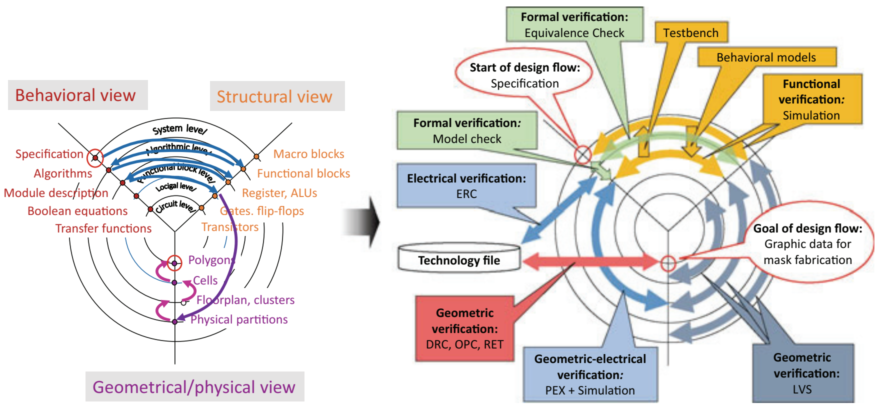

Behavioral view
Structural view
System level
Algorithmic level
Functional block level
Logical level
Circuit level
Specification
Algorithms
Module description
Boolean equations
Transfer functions
Macro blocks
Functional blocks
Register, ALUs
Gates, flip-flops
Transistors
Polygons
Cells
Floorplan, clusters
Physical partitions
Geometrical/physical view
Formal verification: Equivalence Check
Testbench
Behavioral models
Start of design flow: Specification
Functional verification: Simulation
Formal verification: Model check
Electrical verification: ERC
Technology file
Goal of design flow: Graphic data for mask fabrication
Geometric verification: DRC, OPC, RET
Geometric-electrical verification: PEX + Simulation
Geometric verification: LVS

Fig. 5.33 Illustration of the various verification methodologies (cf. Table 5.1) using the Y-chart (right), where the top-down design style is visualized (left)

the various options available for verifying a circuit. Table 5.1 also includes a method that is beyond the scope of this book, testing, which *validates* a circuit design with regard to a customer's requests (see Fig. 4.18 in Chap. 4). Please also note that we omit *assertion-based verification* (ABV) where designers use assertions to capture specific design intent. This verification methodology can be addressed nowadays by formal verification techniques (model checking) as well as traditional simulation strategies.

Figure 5.33 relates the verification methodologies to the Y-chart (Chap. 4, Sect. 4.2.2).

5.4.2 Formal Verification

The goal of *formal verification*, also called *formal functional verification*, is to prove the correctness of a circuit implementation with respect to its specification. More specifically, it shows the correctness of an intended circuit regarding a specific formal specification or property, using formal mathematical methods. The best known formal verification methods are "model checking"—often called "property checking" in commercial tools—and "equivalence checking".

Model checking verifies a certain property of a design or an implementation. It proves (or disproves) that a design under verification, often described in HDL code, satisfies its specifications, i.e., that it behaves as expected in every way (and only as expected). Both the design model under verification and the specification are formulated using precise mathematical language. Essentially, a given structure must satisfy a given logical formula in this check.

Equivalence checking, on the other hand, compares two circuit descriptions. It exhaustively checks that two design representations, such as HDL code and a derived gate level netlist, provide the same functional behavior. There are different approaches to executing this type of proof. For example, both circuit descriptions can be represented by a normalized notation, such as a netlist syntax, to simplify the comparison. Equivalence checking is the primary methodology for synthesis verification.

Formal verification delivers either a successful verification result or demonstrates (i) that the circuit description does not meet a desired property (model checking), or (ii) that two circuit descriptions are not the same (equivalence checking).

Formal verification is part of the early design steps, such as HDL-based netlist generation, that we covered in Sect. 5.1. As the layout designer does not deal directly with this pre-layout verification method, we will not explore it further here; more information on formal verification can be found in the literature: for example, [13] is a well-written and easy-to-grasp introduction to the topic.

5.4.3 Functional Verification: Simulation

A circuit's functional correctness can be verified by simulation. Here, typical input patterns, so-called *stimuli*, are used to check whether the simulated outputs are identical to the intended outputs. Alternatively, the stimuli can be applied to the design behavioral description and to the final gate description. In this case, their responses are compared and evaluated.

Any differences in simulation results can be caused by (i) design errors or (ii) simulation errors or inaccuracies. In both cases, further investigations are required. If, however, the simulation results are identical with the design values, confidence in the correctness of the design is boosted. Unfortunately, simulation can never guarantee that a design is correct in its entirety.

Simulating a circuit's behavior before actually building it can greatly improve design efficiency by flagging design faults early in the flow and providing insight into the circuit's behavior. Almost all IC design relies heavily on simulation. The best-known analog simulators are based on the principle of (or directly derived from) SPICE (Simulation Program with Integrated Circuit Emphasis); digital simulators are often using Verilog or VHDL syntax (Sect. 5.1).

Popular simulators frequently include both analog and event-driven digital simulation capabilities, known as mixed-mode simulators. This means that any simulation may contain components that are analog, event-driven (digital or sampled-data), or a combination of both. Mixed-mode simulation is performed on three levels; (i) with primitive digital elements that use timing models and the built-in digital logic simulator, (ii) with subcircuit models that use the actual transistor topology of the IC, and (iii) with in-line Boolean logic expressions. An entire mixed-signal analysis can be driven from one integrated schematic.

Simulation tools usually interface to a schematic editor, a simulation engine, and on-screen waveform display (Fig. 5.34). These tools allow a designer to quickly

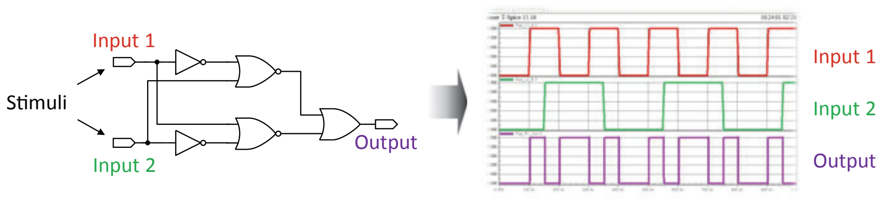


Fig. 5.34 Example of a five-gate circuit with XOR functionality and a (correct) waveform display

modify a simulated circuit and see what effect the changes have on the output. Simulators also typically contain extensive model and device libraries.

When verifying a circuit by simulation, we should always keep in mind that simulation-based verification requires a long execution time, especially for large designs. Worse still, there is usually a lack of a comprehensive set of stimuli to validate the entire design. It is impossible to consider all possible input patterns and circuit states even for fast simulators and small circuits. (For example, the exhaustive simulation of a multiplier for two 32-bit binary numbers would require 2^{64} input patterns, which would take 5,849 years of execution time even with a simulation rate of 100 million multiplications per second [8].) This often forces the designer to rely on some method of random stimuli generation, despite the requirement of full design coverage [5]. Hence, some design errors may remain undetected due to "wrong stimuli".[3]

5.4.4 Timing Verification

The term *timing verification* is used to describe the process of checking whether a digital circuit's timing is still valid after its layout has been produced.

Critical paths in a circuit are calculated during logic synthesis; all paths are checked for worst-case delay times caused by changing signals. The resulting critical path defines the fastest possible clock rate at which the circuit can produce correct output signals. The circuit's layout must meet timing constraints, as well. Essentially, the circuit must pass two timing checks: maximum delay, which is related to setup (long-path) constraints, and minimum delay, which relates to hold (short-path) constraints (Fig. 5.35). Setup checks characterize the performance, whereas a non-passing hold check indicates a faulty circuit.

One approach for timing verification is *dynamic timing analysis*. Here, all wire capacities and resistances are extracted from the layout and the circuit is simulated considering these values. This method is very time consuming as many stimuli must be considered. Restricting dynamic timing analysis to the critical path (defined during

[3]The so-called "Pentium FDIV bug" is an infamous example, where a well-simulated Intel processor returned incorrect binary floating-point results when dividing a number, causing a $475 million loss for Intel [6].

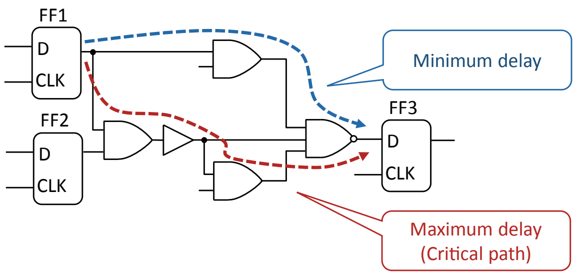


Fig. 5.35 Illustrating minimum and maximum delays resulting from the shortest and longest paths

logic synthesis) does not help because this path is undefined at this later stage—place and route could have easily generated a different critical path than was calculated during logic synthesis.

Static timing analysis (STA) has been developed as a more efficient timing verification method. It is based on the netlist extracted from the layout considering wire capacities and resistances too. The signal delays calculated on all paths are compared to the timing constraints defined by the designer. Specifically, STA propagates actual arrival times (AATs) and required arrival times (RATs) to the pins for every gate or cell. STA quickly locates timing violations, and diagnoses them by tracing out critical paths in the circuit that are responsible for these timing failures [8]. In the past, logic synthesis was then repeated using more restrictive timing constraints on these critical paths; nowadays, STA produces as output an optimized netlist.

Logic gates and wires along with their respective delays are inputs for dynamic and static timing analysis. While gate delays are specified in the timing models in a library, wire delays are calculated using a variety of techniques. Among these techniques, the moment-based technique is widely applied today where impulse responses from the RLC network are analyzed by means of time-frequency transformation methods [5]. Another moment-based interconnect delay calculation uses the first moment of the impulse response; this is known as the Elmore delay model [4].

Any timing-related circuit simulation *after* layout generation requires layout-dependent timing information for the current operating condition to be simulated. The Standard Delay Format (SDF) is used for this timing information. The SDF file contains interconnect delays, gate delays and timing checks that are exported from the physical design tools into an abstracted format. Most important here are the delays associated with the interconnections between devices and ports, i.e., the wire-segment delays as laid out during physical design.

Timing verification also requires checking for resistive and capacitive coupling. For example, crosstalk-induced noise occurs when signals in adjacent wires transition between logic values—and capacitive coupling between these wires causes a charge transfer [5]. This capacitance also has a serious impact on the adjacent wire delays. It is therefore crucial that an accurate timing engine is available to calculate the delay of a coupled system in post-layout timing verification.

Verbal expressions such as "the design has closed timing" are commonly used when the design satisfies all timing constraints. More precisely, the term *timing* *closure* denotes the process of satisfying timing constraints through layout optimizations and netlist modifications [9]. These layout optimizations include timing-driven placement and timing-driven routing. As they are of importance for a layout designer, let us elaborate on these two procedures with additional details.

Timing-driven placement optimizes circuit delay, either to satisfy all timing constraints or to achieve the highest possible clock frequency. It uses the results of STA to identify critical nets and attempts to improve signal propagation delays through these nets. As we introduced in Sect. 5.3.2, timing-driven placement can be categorized as net-based or path-based. There are two types of net-based techniques—(i) delay budgeting assigns upper bounds to the timing or length of individual nets, and (ii) net weighting assigns higher priorities to critical nets during placement [9]. Path-based placement seeks to shorten or speed up entire timing-critical paths rather than individual nets. Although it is more accurate than net-based placement, path-based placement does not scale to large, modern designs because the number of paths in some circuits, such as multipliers, can grow exponentially with the number of gates [9].

After detailed placement, clock network synthesis and post-clock network optimization, the *timing-driven routing* phase aims to correct the remaining timing violations. It seeks to minimize one or both of (i) maximum sink delay, which is the maximum interconnect delay from the source node to any sink in a given net, and (ii) total wirelength, which affects the load-dependent delay of the net's driving gate [9]. Specific methods of timing-driven routing include generating minimum-cost, minimum-radius trees for critical nets, and minimizing the source-to-sink delay of critical sinks [9].

If outstanding timing violations still remain, further optimizations, such as re-buffering, are applied.

5.4.5 Geometric Verification: DRC, ERC

The term *geometric verification* summarizes all checks that are executed at the finished (geometric) layout or during layout design. Most notable here are the design rule check (DRC) and the electrical rule check (ERC).

Every chip manufacturer provides geometrical design rules (Chap. 3, Sect. 3.4) for their technology to the chip-designing organization (cf. Fig. 4.19 in Chap. 4). They are stored in *technology file* which is part of the design suite for a given technology (process design kit, PDK). These rules are a prescription for preparing photomasks that can be applied during IC design and which deliver a manufacturable layout. More precisely, a design rule set specifies certain geometric and connectivity restrictions to ensure sufficient margins to account for variability in the applied semiconductor manufacturing process.

As discussed earlier (Chap. 3, Sect. 3.4.2 and Chap. 4, Sect. 4.5.2), geometrical design rules can be separated into width, spacing, extension, intrusion, and enclosure rules (Fig. 5.36, left; cf. Fig. 3.20 in Chap. 3). Another category of rules that can be

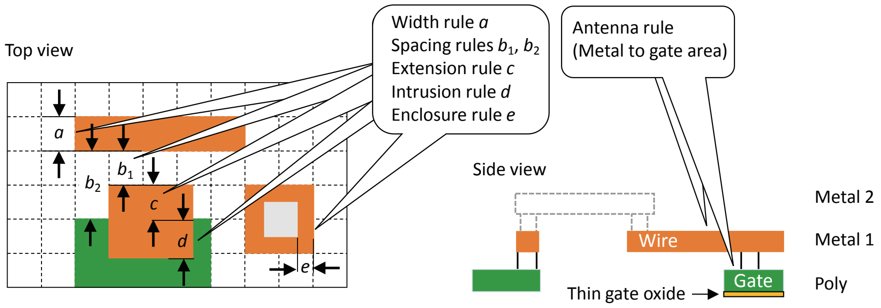


Fig. 5.36 Visualization of the basic DRC checks (width, spacing, extension, intrusion, and enclosure rules, cf. Fig. 3.20 in Chap. 3) and of the antenna rule, which is the allowable ratio of poly or metal area to gate area (right)

checked during geometric verification are antenna rules (Fig. 5.36, right), which we will elaborate on below.

DRC software uses the aforementioned technology file, sometimes called a *DRC deck*, during the verification process; the layout data is usually provided in the GDSII/OASIS standard format. DRC has evolved from simple measurements and Boolean checks to much more sophisticated rules that modify existing features, insert new features, and check the entire design for process limitations such as layer density. Modern design rule checkers perform complete verification checks on geometrical design rules (Chap. 3, Sect. 3.4). The DRC tool either flags any violations directly in the layout (Fig. 5.37) or it produces a report of design rule violations.

In some special design cases, the designer may not choose to correct any DRC violations. Here, carefully "stretching" or waiving certain design rules is a tactic

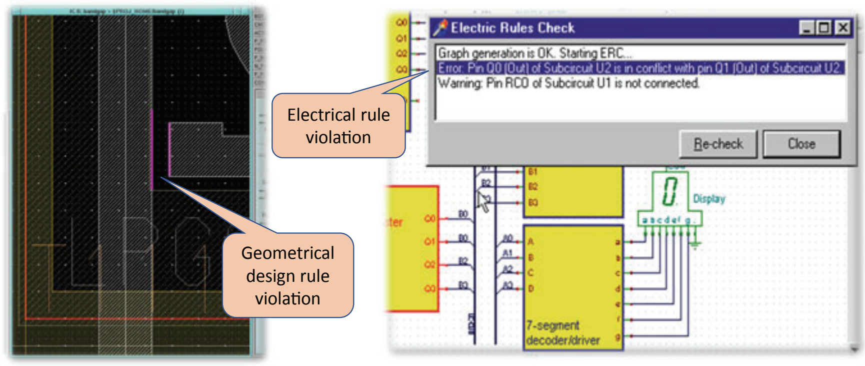

Graph generation is OK. Starting ERC...
Error: Pin Q0 (Out) of Subcircuit U2 is in conflict with pin Q1 (Out) of Subcircuit U2.
Warning: Pin RCO of Subcircuit U1 is not connected.

Fig. 5.37 The design rule check (DRC) verifies that geometrical design rules are met, exposing a minimum distance violation (left). In contrast, the electrical rule check (ERC, right) checks for inconsistencies in the electrical network that can be determined from the geometry and connectivity in the circuit schematic or layout. Simply speaking, DRC does syntax analysis on the layout and ERC performs syntax analysis on the network

used to increase performance and component density at the expense of yield. Obviously, the more conservative the design rules are, the more likely the design will be manufactured correctly; however, performance and other objectives could suffer.

As already mentioned, *antenna rules* can be included in the DRC. A so-called *antenna* is an interconnect, i.e., a conductor such as polysilicon or metal, that is only partially complete during the manufacturing process. During this time, as the layers above are not processed yet, this interconnect is temporarily not electrically connected to silicon or grounded during the wafer processing steps (see Fig. 5.36, right). Charge can accumulate on these (temporary dead-end) connections during the manufacturing process to the point at which leakage currents are generated and permanent physical damage can be caused to thin transistor gate oxide that lead to immediate or delayed failure. This destructive phenomenon is known as the *antenna effect*. Fabs normally supply antenna rules that are often expressed as an allowable ratio of polysilicon and metal area to gate area. There is one such ratio for each interconnect layer, which is then verified during the DRC. Sometimes a specific ratio of the polysilicon and metal shapes' circumference to the gate area is additionally required because the charges are preferably collected at the antenna edges.

As design for manufacturability (DfM) has gained importance, DRC tools increasingly include checks for manufacturability which go beyond the basic geometrical design rules. This encompasses the Boolean operations and sizing functions that we covered in Chap. 3; relations between different layers can also be included in an automatic verification. Again, these rules are directly provided by the IC manufacturer.

Finally, we must point out that DRC can be extremely runtime intensive as the checks usually run on each sub-section of the circuit to minimize the number of errors that are detected at the top level. Modern designs can have DRC runtimes of up to a week. Most design companies require DRC to run in less than a day in order to achieve reasonable cycle times since the DRC will likely be executed several times prior to design completion.

So far we have hopefully conveyed to the reader that the DRC ensures that the circuit will be *manufactured* correctly. It should also be clear that correct *functionality* cannot be checked this way, this is left to the simulators and verifiers that manipulate circuit behavior and that we covered earlier in Sects. 5.4.2–5.4.4.

Now let us investigate the "middle ground" between simple layout checking and complex behavioral analysis, which is the domain of *electrical rule checkers* (*ERC*). Electrical (design) rules make a circuit more robust, for example, by guarding it against damage from electro-static discharge; they also improve its reliability by reducing aging due to electrical overstress. These rules are highly dependent on (i) the applied semiconductor technology, (ii) the circuit type, and (iii) the circuit's future use as a component in a large system environment. In addition, electrical rules are often complemented by design-house specific rules and experience-based rules.

Hence, electrical rule checking is a methodology used to validate the robustness and reliability of a design both at the schematic and layout levels against various "electrical design rules". It verifies the correctness of power and ground connections and checks for floating nets or pins and open and short circuits. For example, by propagating the power, ground, input, and clock signals through the circuit's schematic and/or layout, it is possible to check for incorrect output drives, inconsistencies in signal specifications, unconnected circuit elements, and much more. The results are either visualized inside the schematic/layout editor or presented in a table (see Fig. 5.37, right).

Electrical rules are often specified as topological structures rather than single device/pin checks. Geometrical rules from the layout are also associated with these topologies to ensure proper design function, performance, and yield. Some rules, such as voltage-dependent metal spacing rules, combine both geometrical and electrical checks.

5.4.6 Extraction and LVS

The *layout versus schematic tool*, often abbreviated as the *LVS check*, compares the original netlist, that was used to generate the layout, with a netlist that has been extracted from the layout produced. This proves finally that the generated layout corresponds exactly with the original netlist. More precisely, the LVS check ensures that circuit design and layout design match by checking for (i) the electrical connectivity between device instances, (ii) the correct device instances in the netlist and the layout, and (iii) function-critical device instance parameters. This tool and the DRC are the most important verification tools in any IC design flow.

In order to compare both netlists, the LVS tool must first *extract* a netlist from the layout data. This is performed in an extraction step. It requires a technology-dependent extraction file containing three definitions:

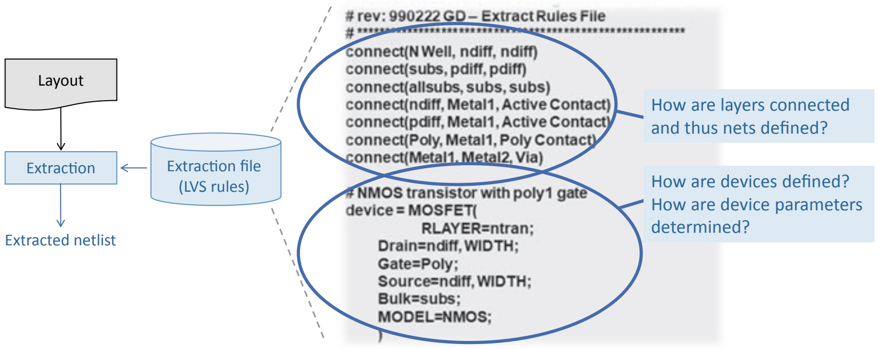
- How are the layers connected, i.e., what forms a *net*?
- What combination of polygons and layers form a *device*?
- Which device polygon properties determine the electrical *parameters*?

Figure 5.38 visualizes the contents of such an extraction file. The contents of a netlist can only be derived from a given layout with these three pieces of information (layer connections, devices, device parameters), as the layout, after all, consists only of polygons.[4]

The extraction algorithm is able to generate a netlist from the graphics data of the layout based on this description. The procedure is as follows:

(1) Defining the basic devices
  (a) Determine all geometrical structures that represent the basic devices.
  (b) Separate the basic devices from the other layout structures.

[4]It is important to note why we do not take any other layout information into account, such as library information. This would, of course, greatly simplify the task and speed up netlist recognition. However, any error in the library would then be considered as well. The final netlist check would then check identical netlists as both lists would be affected by the same library-based error(s). This would render the LVS useless.


Layout
Extraction
Extraction file (LVS rules)
Extracted netlist
How are layers connected and thus nets defined?
How are devices defined? How are device parameters determined?

```
# rev: 990222 GD – Extract Rules File
# ****************************************************
connect(N Well, ndiff, ndiff)
connect(subs, pdiff, pdiff)
connect(allsubs, subs, subs)
connect(ndiff, Metal1, Active Contact)
connect(pdiff, Metal1, Active Contact)
connect(Poly, Metal1, Poly Contact)
connect(Metal1, Metal2, Via)

# NMOS transistor with poly1 gate
device = MOSFET(
            RLAYER=ntran;
      Drain=ndiff, WIDTH;
      Gate=Poly;
      Source=ndiff, WIDTH;
      Bulk=subs;
      MODEL=NMOS;
      )
```

Fig. 5.38 An extraction file is needed to extract the netlist from the layout polygons as this can only be achieved by knowing which polygon configuration forms a via or a device

(2) Determine electrical nodes
Determine all geometrical structures that form electrically linked units. This is an intra-mask operation.

(3) Generating the netlist
   (a) Determine the nodes to which geometrical structures adjacent to basic devices belong.
   (b) Assign the connection types (e.g., gate and source in the case of transistors).

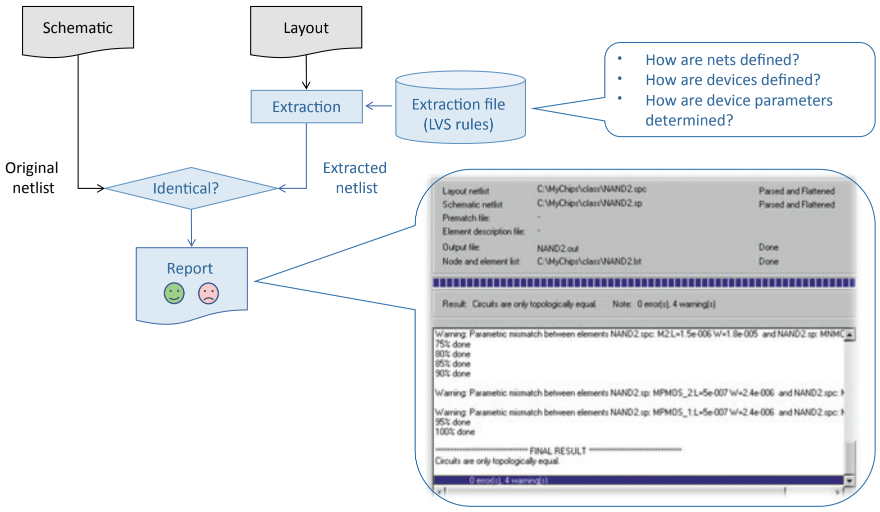
The contents of this netlist are then compared with a netlist derived from the circuit schematic. The entire LVS procedure is depicted in Fig. 5.39.

The LVS compares the output data (layout) with the input data (circuit schematic) w.r.t. the following three circuit-diagram attributes:

- Nets: Are all electrical connections in the circuit schematic—and only these connections—in the layout as well?
- Type of devices: Are all devices from the circuit schematic—and only these devices—present in the layout?
- Parameters of devices: Do all devices in the layout have the electrical parameters specified in the schematic?

The result of the LVS is a report file (see Fig. 5.39) that contains the number and types of devices as well as nodes in the original netlist (from the schematic) and the netlist that was extracted from the layout. This file also lists all non-matching components in both netlists. It is up to the designer to investigate these issues further, as these comparison errors or warnings can be serious faults or simply unrecognizable features flagged by the extraction tool.

One of the major issues with LVS verification is the repeated iterations of design checking required to find and remove these non-matching components between both netlists [5]. As this can be very time consuming, hierarchical verification features (rather than a flat comparison) should be used. Here, memory blocks and other



```
Layout netlist          C:\MyChips\class\NAND2.spc          Parsed and Flattened
Schematic netlist       C:\MyChips\class\NAND2.sp           Parsed and Flattened
Prematch file:          -
Element description file: -
Output file:            NAND2.out                           Done
Node and element list:  C:\MyChips\class\NAND2.lst          Done

Result:  Circuits are only topologically equal.     Note:  0 error(s), 4 warning(s)

Warning: Parametric mismatch between elements NAND2.spc: M2:L=1.5e-006 W=1.8e-005  and NAND2.sp: MNMC
75% done
80% done
85% done
90% done

Warning: Parametric mismatch between elements NAND2.sp: MPMOS_2:L=5e-007 W=2.4e-006  and NAND2.spc: M

Warning: Parametric mismatch between elements NAND2.sp: MPMOS_1:L=5e-007 W=2.4e-006  and NAND2.spc: M
95% done
100% done

****************************** FINAL RESULT ******************************
Circuits are only topologically equal.

0 error(s), 4 warning(s)
```

Fig. 5.39 The LVS methodology is based on a netlist extraction from the layout. This netlist is compared with the original netlist that was used to generate the layout

intellectual-property (IP) elements are compared in a hierarchical manner, while other design elements, such as analog blocks and macro cells, maintain a flat representation [5]. Consequently, verification (debugging) time can be drastically reduced.

So far we have seen how the extraction tool generates a netlist from a layout. Extraction tools also feature *parasitic extraction* (*PEX*). Here, the parasitic effects in the interconnects are calculated. The parasitics in question are: (i) parasitic capacitances, (ii) parasitic resistances, and (iii) parasitic inductances.[5]

Parasitic extraction is required in order to create a more accurate analog circuit model. Based on device models and PEX results, detailed simulations can emulate actual digital and analog circuit responses. Another factor in the rise of interest in parasitics is the importance of wiring capacity in advanced technology nodes: interconnect resistances and capacitances started making a significant impact on circuit performance below the 0.5-μm technology node. Interconnect parasitics cause signal delays, signal noise, and IR drops—all important issues affecting circuit timing and performance especially of analog circuits. In summary, timing analysis, power analysis, circuit simulation, and signal integrity analysis rely on parasitic extraction.

Parasitic extraction methodologies can be broadly divided into (i) field solvers, which provide physically accurate solutions, and (ii) approximate solutions with pattern matching techniques. Since field solvers can only be applied to small problem instances, pattern matching techniques are the only feasible approach to extract parasitics for complete modern IC designs.

[5]Additional parasitic coupling effects are caused by the chip substrate, which is common to all devices. However, these effects are not considered in all simulation tools.

The extraction tool can also be used for antenna checks (Sect. 5.4.5). Here, the gate area and the area of the conductor(s) are extracted, and their ratio is calculated and compared with a reference value.

Finally, the extraction tool is also required for specific ERC functions (Sect. 5.4.5). An example is pin-to-pin checks within the ERC where a specific resistance value should not be exceeded in order to meet ESD requirements.