Transistor-Level Layout of Integrated Circuits

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Vorgelegt von

Jan Schneider

aus

Bad Godesberg

Bonn, Mai 2014

Angefertigt mit Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter:	Prof. Dr. Stefan Hougardy
2. Gutachter:	Prof. Dr. Jens Vygen
Tag der Promotion:	11. Juli 2014
Erscheinungsjahr:	2014

Contents

Chapter 1

Introduction

> Integrated circuits will lead to such wonders as home computers—or at least terminals connected to a central computer—automatic controls for automobiles, and personal portable communications equipment. The electronic wristwatch needs only a display to be feasible today.
>
> Gordon E. Moore, April 1965

Integrated circuits, commonly known as "chips", have been invented in the late 1950s. The original prototype with a single transistor on a layer of Germanium, which was developed at Texas Instruments in 1958, is depicted in Figure 1.1. One year later, Jack Kilby submitted the patent application for "unique integrated electronic circuits fabricated from semiconductor material" [Kil64].

Since then, the complexity of computer chips has seen a steady exponential growth. Most famously, this behavior has been predicted in an article by Gordon E. Moore [Moo65], thereby establishing the well-known eponymous law. The soon-to-be co-founder of Intel observed that the number of components per integrated circuit doubled every year since the technology has been invented, and stated that "there is no reason to believe [the rate of increase] will not remain nearly constant for at least 10 years." Although Moore overestimated the magnitude of the increase—the number doubled every 2 years since then—the exponential nature of the curve stayed an immutable fact in the rapidly changing semiconductor industry.

Figure 1.2 demonstrates the verity of Moore's law over the course of nearly 60 years. It combines the data points available to Moore in 1965, including Kilby's original integrated circuit, and a range of major CPUs manufactured by IBM and Intel.

In the meantime, transistors have probably become the human invention that has been manufactured the most. By the year 2015, the number of transistors in the world, according to an estimation by Paul Otellini [Jam11], will reach $1.2 \cdot 10^{21}$. Nowadays, mass-produced chips are collections of billions of interconnected transistors. Examples are IBM's Power7+ server CPU from 2012 with over 2 billion transistors, shown in Figure 1.3a in its actual size, and Nvidia's GK110 graphics processing unit with more than 7 billion transistors, which remains the highest number ever achieved by a single-die microprocessor since its release in 2013.
Even higher numbers are reached by memory modules: As they only require very small transistors and allow for arrangements with much regularity, the first microSD cards with 128 GB memory could be manufactured in 2014, assembling roughly half a trillion transistors in a volume of 15 $mm \times 11$ $mm \times$ 1 mm (cf. Figure 1.3b).

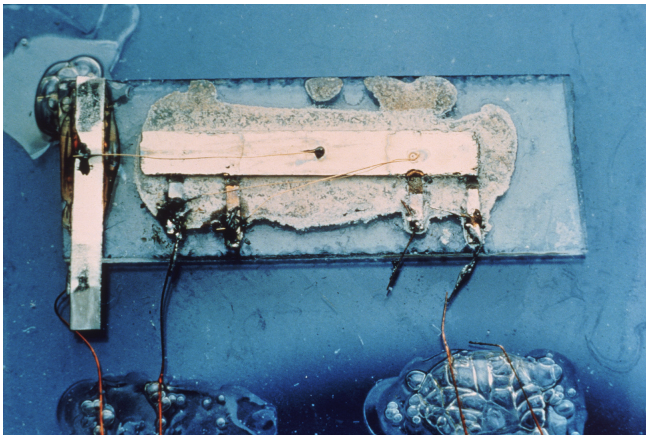


Figure 1.1: Kilby's original prototype of an integrated circuit with one transistor on a scale of 4:1. Courtesy of Texas Instruments.

Automation

Along with the growing complexity of computer chips, their layout has been an increasingly difficult task. While early integrated circuits could essentially be devised manually, later generations required more and more automation to handle the exponential increase of components on each chip. At first, algorithms were applied to improve the arrangement of all transistors on an integrated circuit at once. But soon their number became so large that a hierarchical structure was imposed in order to successfully apply optimization methods. Recent CPUs consist of cores, each of which consist of units, which are collections of macros and cells and, on the bottommost level of hierarchy, the single transistors.

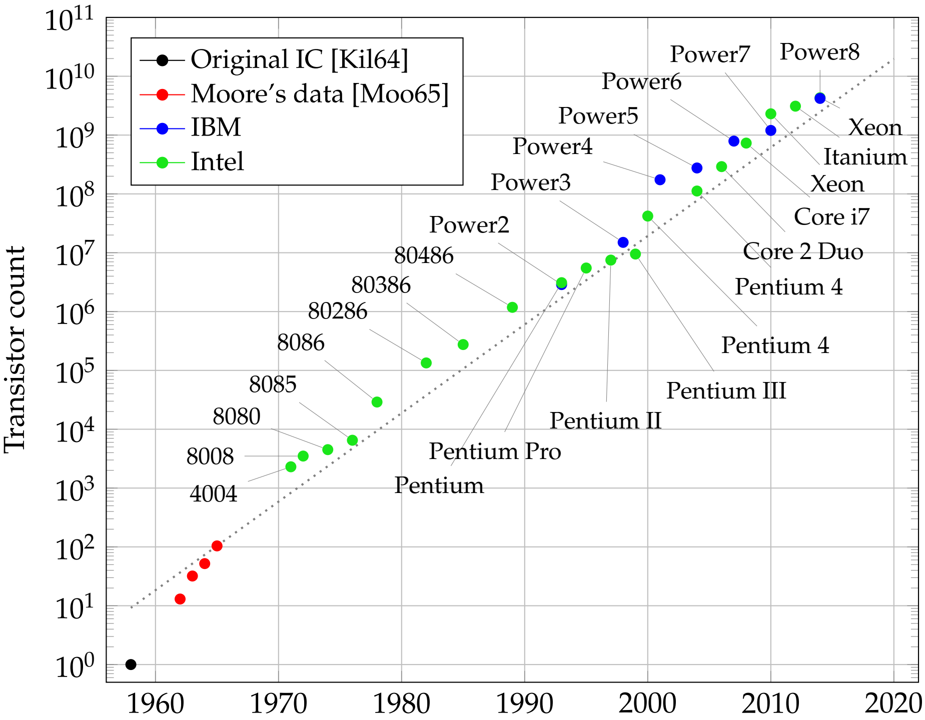
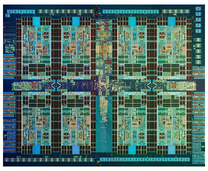


Figure 1.2: Illustration of Moore's law. The dotted line passes through Moore's last data point and doubles every 2 years.



(a) 2.1 billion transistors

(b) 550 billion transistors

Figure 1.3: Modern packaging of transistors shown in their actual sizes: IBM Power7+ CPU and 128 GB microSD card.

In the 1980s, algorithmic approaches to automate the generation of geometric layouts for small functional groups of transistors became a frequent topic in industry as well as academia. The problem was formulated in graph-theoretical terms, algorithms were proposed and improved, and variants were introduced in alignment with the technology's progress.

However, increasingly complicated constraints drove the need to employ a higher level of abstraction. Chip design tools shifted their focus from single transistors to functional groups thereof, modeled merely as rectangular blocks or even larger heterogeneous modules. At the same time, the generation of geometric layouts of transistors and the wires connecting them was either performed by scripts following a small set of simple rules, which is viable for very basic circuits with a low structural complexity, or went back to the hands of human engineers manually drawing shapes into specialized software products. Although new concepts to automate these tasks were still introduced in the literature of the 2000s, corresponding programs usually lacked the ability to exploit the technology's features and could hardly surpass the quality of manually conceived layouts. Hence, the applicability of these tools was restricted to a relatively small set of use cases.

Consequently, most non-trivial transistor-level layouts today are devised by hand, and hundreds of geometric rules have to be obeyed in the process. This work is difficult and time-consuming: For circuits with only a dozen transistors it can take days until a good solution is found. Moreover, depending on the engineer's experience and the time spent on this task, the resulting layouts may vary in their quality.

The importance of good and especially compact solutions is underlined by Table 1.1, which illustrates the cost of a larger chip area for the original Pentium CPU. Just 1% more chip area would have cost Intel over 63 million dollars per year. Had the Pentium processor had a 15% larger die, the additional cost would have amounted to nearly a billion dollars annually. Furthermore, about a third fewer chips would have been produced in the same time.

Table 1.1: Area penalties for the production of the original Pentium CPU, reprinted from [MH96].

	Pentium die	+1% die size	+15% die size
Die size	160.2 mm^2	161.8 mm^2	184.2 mm^2
Die cost	\$84.06	+1.5%	+22.0%
Additional annual cost		+\$ 63 500 000	+\$ 961 000 000
Chips fabricated per week	498 100	−3.1%	−32.2%

Preview

In this dissertation, we present the toolchain BONNCELL and its underlying algorithms. It has been developed in close cooperation with the IBM Corporation and automatically generates the geometry for functional groups of 2 to approximately 50 transistors as visualized in Figure 1.4. Its input consists of a set of transistors, including properties like their sizes and their types, a specification of their connectivity, and parameters to flexibly control the technological framework as well as the algorithms' behavior. Using this data, the tool computes a detailed geometric realization of the circuit as polygonal shapes on 16 layers. To this end, a placement routine configures the transis-

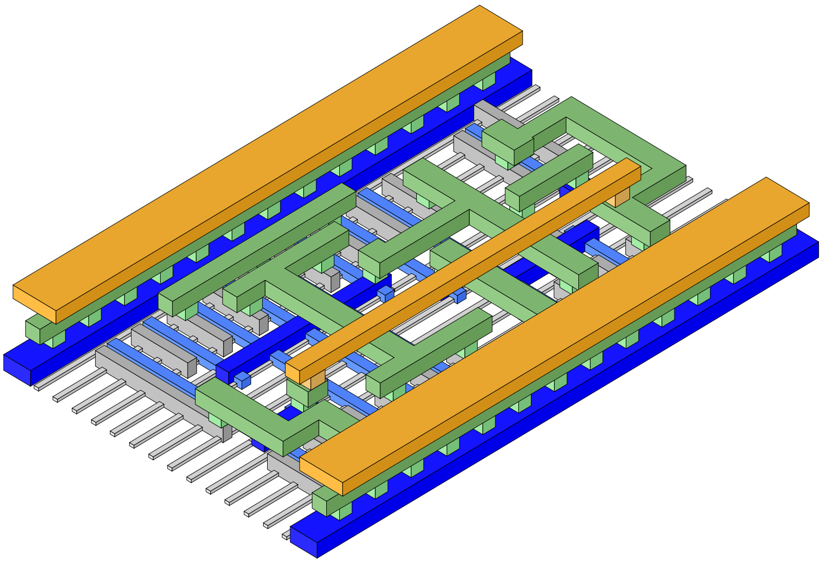


Figure 1.4: Example for a circuit layout generated by BonnCell.

tors and arranges them in the plane, which is the main subject of this thesis. Subsequently, a routing engine determines wires connecting the transistors to ensure the circuit's desired functionality.

BonnCell produces on most real-world instances provably optimal solutions in terms of a multi-criteria target function that primarily favors the most compact realizations possible within the capabilities of the given technology. The purpose of the secondary criteria is to generate routable solutions, i.e. layouts in which the transistors can be connected as needed.

For 90% of the instances in a representative test bed with a large structural variety, fully functional layouts can be generated by BonnCell. Aside from such isolated test instances, our tool is actively used at IBM for the production of new chips. It could, for example, help to finish the initial layout phase of a large memory block in half of the designated timeframe, and it also aided in central decisions during the very early stages of a new technology.

In Chapter 2, we start our discussion by introducing technological concepts, including the basic blueprint of a transistor-level circuit, and a number of notions used to describe their building blocks. Before our own work is presented, a thorough classification of previous work on variants of the transistor-level layout problem is given in Chapter 3, ranging from very early efforts shortly after the invention of integrated circuits to recent publications concerned with modern technologies.

Thereafter, Chapters 4 and 5 contain the description and analysis of our algorithms for the computation of circuit layouts. The former is focused on the arrangement of 1-dimensional rows of transistors, provides formulations in graph-theoretical terms and states implications on the hardness of such problems. The latter is focused on the practical implementation and the combination of the presented algorithms to a flow that handles 2-dimensional layouts rather than single rows of transistors. The chapter also documents many variations of the methods suited for a wide range of use cases.

In Chapter 6, we briefly discuss the routing algorithms of BONNCELL, giving an overview on how interconnections of transistors are realized. The method combines 3 separate techniques, one of which combinatorially seeks a packing of Steiner trees, one of which uses a mixed integer linear program to model technology constraints, and one of which directly operates on the wires' rectilinear shapes to improve the solution quality.

Finally, a detailed analysis of the results is given in Chapter 7. This encompasses a thorough evaluation of the tool's major features based on over 300 real-world test instances, a comparison of BONNCELL output with actual layouts used in the industry, and the documentation of two different yet successful cases in which our toolset has been applied at IBM to great avail. Chapter 8 concludes with some closing remarks and an outlook to future developments.

Acknowledgments

This dissertation would not have been possible without the support of many individuals. First and foremost, I want to express my gratitude to Prof. Dr. Stefan Hougardy for the exceptional supervision of my doctoral studies and the BONNCELL project. I learned a lot from his invaluable feedback and our close cooperation. I also wish to thank Prof. Dr. Bernhard Korte and Prof. Dr. Jens Vygen for providing excellent work conditions at the Research Institute for Discrete Mathematics without which the project's success would not have been possible.

Furthermore, I'd like to emphasize the importance of the extremely good atmosphere in the institute, which is filled with great colleagues and good friends. This is true for the BONNCELL group, especially Dr. Tim Nieberg, Jannik Silvanus, as well as former students Katrin Kölker and Thomas Weyd, all of which participated in the development of an excellent routing engine. But it is equally true for my former team, Dr. Ulrich Brenner and Dr. Markus Struzyna, which was joined by Philipp Ochsendorf, and the rest of the institute, especially Christoph Bartoschek, Dr. Jens Maßberg and Ulrike Schorr.

BONNCELL is also the product of a close and fruitful cooperation with IBM. In particular, I am grateful to Tobias Werner, who initiated, advertized and integrated the tool inside IBM since its early days. I also thank IBM's Karsten Muuss and Raphael Polig, who contributed to the integration of the software's original version.

Last but certainly not least, I wholeheartedly thank my family for the ongoing support during the past 32 years and Maike, who patiently endured my absence during the recent months.

Chapter 2

Preliminaries

> Any sufficiently advanced technology
> is indistinguishable from magic.
>
> Arthur C. Clarke, *Hazards of Prophecy:*
> *The Failure of Imagination*

In the following, we present the basic concepts upon which the subsequent discussion of transistor-level circuit layout is based. Most importantly, we establish terms and concepts connected to field-effect transistors and their operation in integrated circuits. Moreover, a widespread scheme for the geometric layout of functional units on a chip is described that will be followed by the algorithms in BonnCell.

2.1 Graph Theory

The notions related to graph theory and combinatorial optimization used in this thesis are based on the book *Combinatorial Optimization* by Korte and Vygen [KV13]. In this section, we briefly provide some definitions that are essential for the subsequent discussions.

walk

closed walk

The central concepts employed in the following chapters will be walks. Given a graph $G = (V, E)$, a *walk* W is a sequence $(v_0, e_0, v_1, \dots, v_{k-1}, e_{k-1}, v_k)$, where $v_0, \dots, v_k \in V$ and $e_0, \dots, e_{k-1} \in E$, such that $e_i = \{v_i, v_{i+1}\}$ holds for $0 \leq i < k$ and $e_i \neq e_j$ for $i \neq j$. If G is directed, then $e_i = (v_i, v_{i+1})$ is required. We use $V(W)$ and $E(W)$ do denote the vertices and edges covered by W, i.e. $V(W) := \{v_0, \dots, v_k\}$ and $E(W) := \{e_0, \dots, e_{k-1}\}$, and say that W has a *length* of k. A walk is called *closed* if $v_0 = v_k$.

Eulerian walk

(semi-)Eulerian

A *Eulerian walk* in a graph $G = (V, E)$, also called *Eulerian trail* in the literature, is a walk W with $E(W) = E$. The graph is called *semi-Eulerian* if it possesses a Eulerian walk and *Eulerian* if it possesses a closed Eulerian walk.

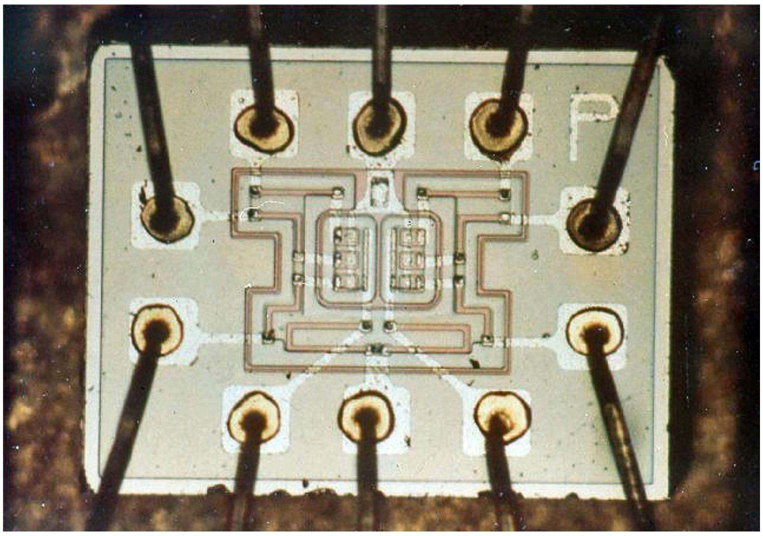


Figure 2.1: Dual `NOR3` integrated circuit from the Apollo Guidance Computer of 1966, one of the first computers using integrated circuits.

2.2 Chips

We now turn towards the technological principles of chips, most notably the geometry and electrical properties of transistors and how appropriately interconnected groups of transistor can be used to realize electrical circuits.

2.2.1 Transistors and Integrated Circuits

transistor

source, drain, gate

Transistors are the basic building blocks of every electronic device. They serve as switches devoid of movable parts and have three external connections called *source*, *drain*, and *gate*. By applying a voltage at the gate one can control if the connection between the two other contacts is conducting, that is a current can flow between source and drain, or insulating.

n-type, p-type

There is a multitude of different technical implementations of a transistor with variations in the used materials, the size, the geometry, electrical properties, and so on. Although these variations are not subject to this work, it is important to note that there are two fundamentally different, complementary types of transistors: *n-type* and *p-type*. An n-type device will transfer electrons from source to drain ("switched on") if a voltage is applied to the gate and it will fail to do so otherwise ("switched off"). A p-type device acts exactly conversely.

FET, n-FET, p-FET

integrated circuit

The transistors subject to discussion in this work are *field-effect transistors* (FETs), and the two types are called *n-FETs* and *p-FETs* accordingly. The name refers to the eponymous physical law that is employed to implement the device's functionality. They constitute the active part of an *integrated circuit* (IC), also called "chip". An IC is a complex electronic circuit that is built from a single piece of semiconductor material. It is doped at appropriate locations to form the switching elements, namely the FETs, and amended by several layers of metal connections that join the transistors, thereby realizing a desired logic function, a memory element, or other features. A very simple example is shown in Figure 2.1.

A property that does not affect the geometry of a FET but is important for the later discussion is the *Vt level*, or Voltage threshold. This property refers to the degree to which the substrate at the base of the transistor is doped. A FET can usually be manufactured with one of several Vt levels: A lower level corresponds a higher power demand and faster operation, and a higher level on the other hand leads to less power demand at the cost of a slower operation.

Vt level

Figure 2.2 depicts several representations of a FET: The isometric illustration in Figure 2.2a outlines its 3-dimensional geometric structure. The blue cuboid in the middle is the gate and the two darker gray cuboids are the source and drain contacts. If the voltage applied to the gate is appropriate, current can flow through the light gray area below the three contacts, otherwise this area functions as an insulator.

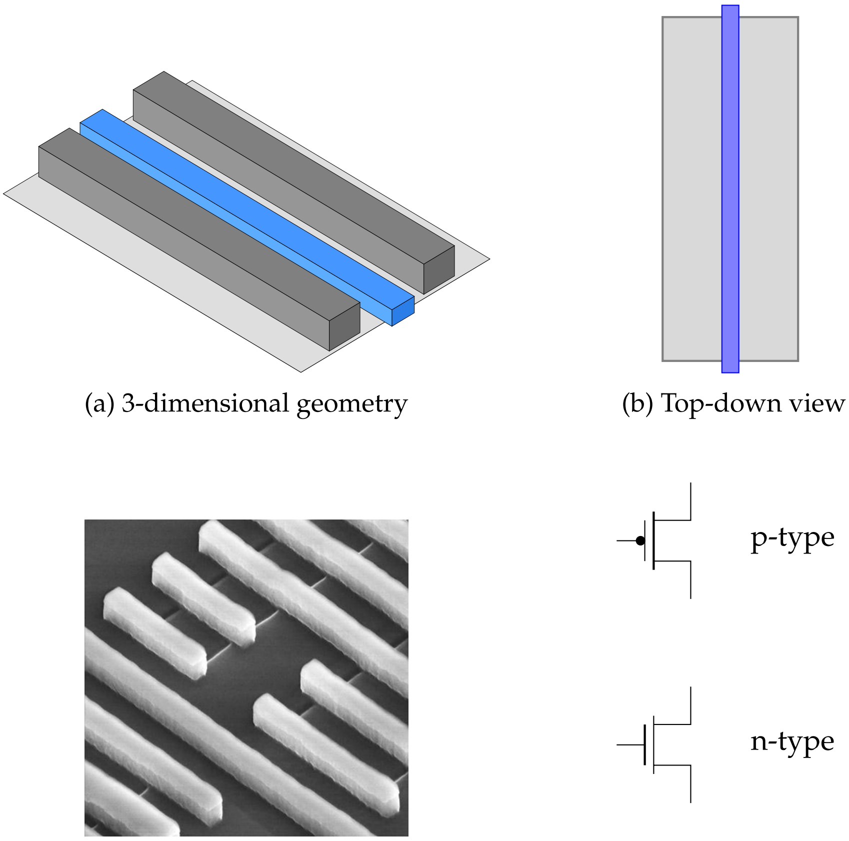
Figure 2.2b depicts the same FET in a simplified 2-dimensional top-down view that will be used to illustrate the algorithms presented in the upcoming chapters. Figure 2.2c depicts an actual electron-microscopic photography of several transistor in Intel's 32 nm technology. The symbols in Figure 2.2d are used in circuit diagrams to indicate n-type and p-type transistors.

2.2.2 FinFETs

In recent years a new type of FET has been developed to drive the ongoing reduction of feature sizes on chips. So-called *FinFETs* work by the same principles but have a different geometry. Instead of a flat diffusion area embedded into the planar substrate below the contacts, the channel between source and drain is formed in erect fins that raise above the substrate. The gate envelops those fins so that the relevant contact area between gate and diffusion material consists of the vertical side portions of the fins and their top rather than just the plane 2-dimensional footprint of the polysilicon cuboid that forms the gate. As a consequence, apparently shorter gates, in the sense of the 2-dimensional footprint, have a larger active area through their use of the vertical dimension. FinFETs are also named 3D or Tri-gate transistors. Figure 2.3 illustrates the 3-dimensional structure of such devices.

FinFET

The aspect of FinFETs that is relevant for the design of algorithms is the discretization of the gate size. While this parameter was previously given as a length, for example in nanometers, now the number of fin/gate intersections is the pertinent value. This value is always a small integer, often smaller than 10, which has consequences for algorithms especially related to the folding technique explained in Section 2.3.1.



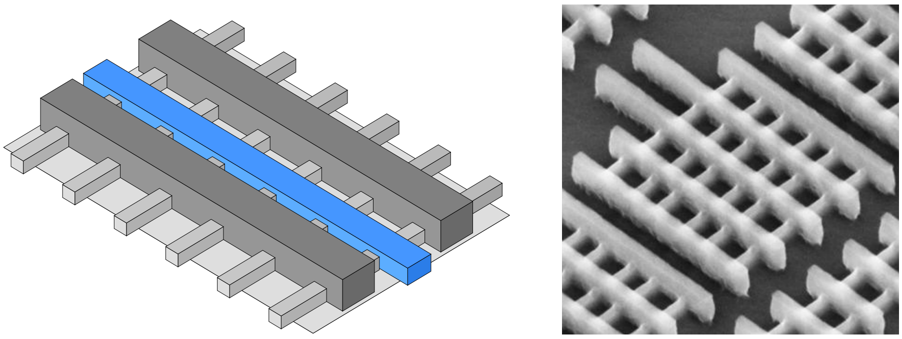
(a) 3-dimensional geometry

(b) Top-down view

(c) Electron-microscopic photography

(d) Symbols

Figure 2.2: Several representations of a classic planar field-effect transistor. The photography actually shows multiple FETs.



(a) 3-dimensional geometry

(b) Electron-microscopic photography

Figure 2.3: Geometry of modern FinFETs.

2.2.3 CMOS

Having established the possibility to implement switches on an integrated circuit, the question arises how to assemble them to realize the desired complex functionality of a chip like the evaluation of a logic function or the storage of a bit. The prevalent principle employed on modern chips to achieve this goal is the *CMOS* technology (complementary metal-oxide-semiconductor) invented by Wanlass [Wan67]. The idea is to use two electric potentials, or power levels, to represent the logic 0 and the logic 1. The output of a circuit is connected to exactly one of those power levels—depending on the power levels of the input signals. This is done through an appropriately connected network of transistors which is typically gated by either the power levels at the inputs or by intermediate results computed within such an elementary circuit. Although several different notations are used depending on context and technology, we denote the two power levels by GND (ground, the logic 0) and V_{DD} (logic 1).

CMOS

Take for example the easiest possible CMOS circuit, the inverter, as depicted in Figure 2.4a. Assume that the input signal x_{in} is connected to the V_{DD} potential. Then in the bottom half of the diagram an n-FET opens a conducting channel, thereby creating a connection between the output signal x_{out} and GND. At the same time, the p-FET in the upper half is switched off such that a short between the two different power supplies is avoided. If on the other hand x_{in} is connected to GND, then the upper FET is conducting and the bottom FET is insulating. To sum up, the setup realizes the logic function $x_{\text{out}} \leftarrow \neg x_{\text{in}}$.

Figure 2.4b shows a more complex example that realizes the logic function $x_{\text{out}} \leftarrow \neg(x_1 \vee (x_2 \wedge x_3) \vee (x_4 \wedge x_5))$. As before, all the transistors in the bottom half of the diagram are n-type devices and all transistors in the top half are p-type. Note that these two networks act "dual" in the sense that one creates a conducting channel if and only if the other does not. This very simple dual structure may be violated in more complicated CMOS circuits, for example if several interconnected CMOS circuits are combined or other related schemes like dynamic CMOS or Domino logic, which are also subsumed under the CMOS paradigm, are used. A comprehensive discussion of the CMOS technology, including these aspects, can be found in [Bak11].

2.3 Transistor-Level Layout

Informally speaking, transistor-level layout is the problem of generating a geometric realization of a given CMOS circuit topology. Such a functional implementation of an electric circuit is also denoted *cell*. Being leaves in the tree that describes the hierarchy on a chip, transistor-level cells are sometimes called *leaf cells*. As their 3-dimensional structure is highly complex, it is neither clear how to arrange transistors and their interconnections within a

(leaf) cell

chip so that "good" layouts are generated nor what metric is used to measure the quality of a layout. Moreover, the physics involved in the manufacturing of integrated circuits imposes a wide variety of restrictions on the feasible layout patterns. For these reasons, a large number of different schemes, and thus algorithms, have been proposed and implemented during the last 40 years. Chapter 3 contains an overview of existing approaches.

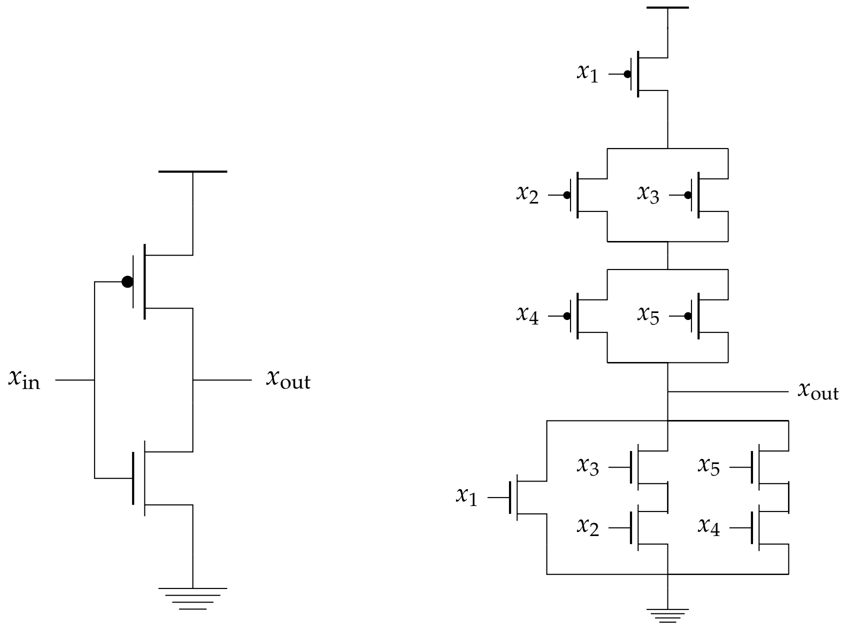
In this section, we give a description of the cell model that serves as a basis for this work, i.e. a blueprint of the CMOS cells generated with our algorithms, including degrees of freedom and restrictions. Formal definitions of subproblems are provided where these problems are discussed in detail. We start with the means by which single transistors can be laid out.

2.3.1 Layout of Transistors

A single transistor, even though it is the atomic unit of an integrated circuit, can be laid out in multiple ways. In this Section, we discuss the relevant aspects of the FETs' geometry.

Swapping

As apparent from the workings of a transistor, the source and drain contacts are interchangeable. It is not important in which direction the electrons move



(a) CMOS inverter (b) Function $\neg(x_1 \vee (x_2 \wedge x_3) \vee (x_4 \wedge x_5))$

Figure 2.4: Examples for CMOS cells.

through the conducting channel. Hence, it is possible to exchange the connections of the source contact and the drain contact without modifying anything else, resulting in an *unswapped* and a *swapped* status for every FET. (un)swapped

Diffusion Sharing

The most important technique to save area is *diffusion sharing*. If two FETs diffusion sharing are supposed to be placed next to each other, and the source/drain contacts facing the neighboring FET must be electrically connected, then the diffusion areas can be merged, forming a single source/drain contact in the middle of the construct with two, possibly different, gates to its sides. Figure 2.5 illustrates the concept.

As a generalization, more complex rules for the minimum distances between FETs might hold, for example the following:

- FETs may overlap (i.e. diffusion sharing is possible) if the opposing contacts must be electrically connected and their gates have the same length.
- FETs may abut as in the left section of Figure 2.5 if the opposing contacts must be electrically connected but their gates do not have the same length (i.e. the right configuration in the figure would be illegal).
- Otherwise a small gap must be introduced between the FETs if their types match and an even larger gap must be used if their types do not match.

Usually the simpler first concept with the only alternatives being "sharing" and "no sharing" has been discussed in the literature. The algorithms presented in this work are able to handle distance functions of arbitrary complexity.

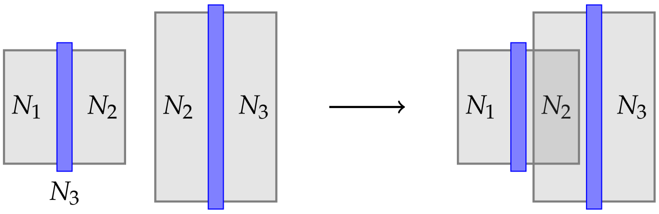


Figure 2.5: Two FETs sharing diffusion area. Net N_2 connects the right contact of the left FET and the left contact of the right FET, so the contact may be shared by both transistors.

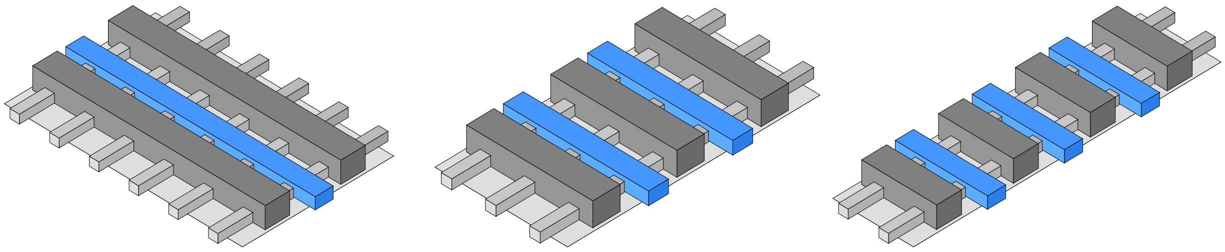
Folding of Single FETs

A major aspect of the layout of single transistors is *folding*: A large FET that folding is supposed to cover many fins, which would normally implicate a geometry with a high aspect ratio, is split into several smaller devices that are connected in parallel. Because those smaller devices all have the same source and drain contacts, they can share their diffusion areas and form a row of overlapping transistors. Figure 2.6 illustrates the geometry of a large FET that is split into two or three smaller FETs. In the process, the long gate is divided into two respective three segments, called *fingers*. The picture shows how the aspect ratio of the FET's footprint changes with the number of fingers.

finger

The other contacts of the folded device alternately belong to the source and the drain contact. All pieces of metal that constitute the drain contact must be connected electrically for the layout to work correctly. The same is true for the parts of the source contact and the fingers, which are all part of the gate. Swapping such a construct means to change the contacts from the pattern source/drain/source/... to the pattern drain/source/drain/...

It is worth to note that the area covered by a FET with a single finger is $2wL$, where w is the distance between two neighboring gates and L the length of the gate, and decreases as the number of fingers increases. For 2 fingers the area is $3w\frac{L}{2}$, for 3 fingers $4w\frac{L}{3}$, and so on. The area converges to wL, which is half of the classic single finger realization.



(a) 1 finger (b) 2 fingers (c) 3 fingers

Figure 2.6: The same FET with 1, 2, and 3 fingers.

Folding of Multiple FETs

The concept of folding can be extended to multiple FETs at once. If, say, one FET F_1 is split into F_1' and F_1'', and another FET F_2 is split into F_2' and F_2'', then it may be possible to lay out an overlapping row of the transistors $F_1'F_2'F_2''F_1''$. In other words, the fragments of the original FETs are placed in an interleaving pattern. The scheme is neither limited to just two large FETs nor to just two fragments per large FET. Even though the connections between the gates become longer compared to the variant $F_1'F_1''F_2''F_2'$, the resulting interleaving placement may have favorable properties in some sense. An example is shown in Figure 2.7.

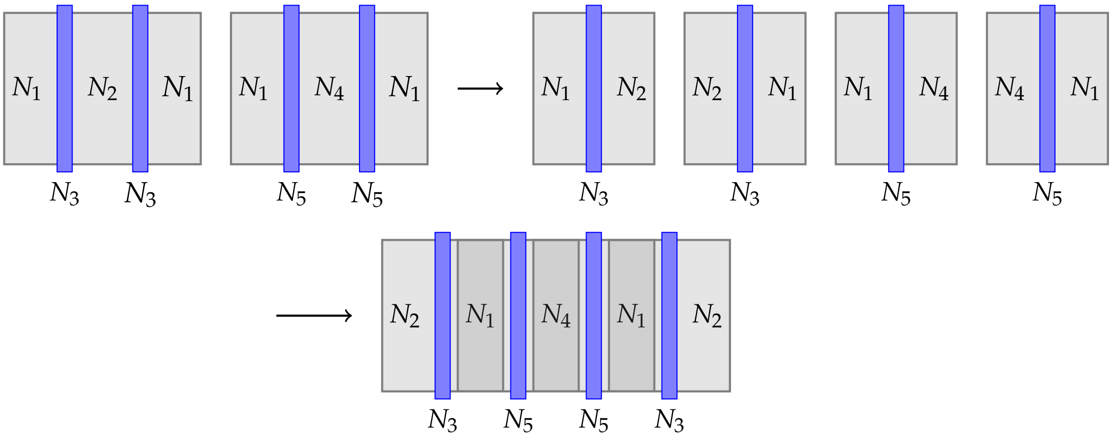


Figure 2.7: Two FETs are folded into an interleaving pattern.

In principle the fragments of a large FET could even be placed completely independent of each other when just viewed as a set of ordinary smaller FETs. However, for electrical reasons this is usually not allowed in real-world technologies and can thus be deactivated.

2.3.2 Layout of Cells

Having finished the discussion of single transistors, we will now describe the basic geometry of cell layouts, i.e. collections of interconnected FETs that implement a more complex functionality.

Layer Stack

The 3-dimensional structure of an integrated circuit is modeled sufficiently well with rectilinear shapes on a certain set of layers. Every component is specified by a 2-dimensional rectilinear polygon and a layer that corresponds to an interval in z direction. Our illustrations usually depict their projection to the x-y-plane and use colors to indicate the layer.

The set of layers together with information regarding their parameters and connectivity is called *layer stack* . Unless otherwise specified, we use the layer stack given in Table 2.1, which is a simplification of the actual layer stack from the examined real-world 14 nm technology.

layer stack

PC, which stands for polysilicon, is the bottommost wiring layer and holds the FET gates. Wiring is only allowed in the vertical direction such that the routing on this layer is essentially limited to prolonging the gates beyond the boundary of the FETs. M0 is the layer which mostly holds the source and drain contacts of the FETs but may also contain interconnections in both directions. M0 physically resides on the same level as PC, which means that in regions where their 2-dimensional projections intersect the two wires are

Table 2.1: Layer stack used in this work.

Name	FET parts	Properties
PC	Gate	Polysilicon, only vertical
M0	Source / Drain	Horizontal and vertical, connects PC
V0		Vias between M0 and M1
M1		Horizontal and vertical
V1		Vias between M1 and M2
M2		Only horizontal

also electrically connected. M1 and M2 are two metal layers that are completely reserved for routing and do not contain any parts of the transistors. Wires can only lie horizontally on M2, whereas both directions are possible on M1. Connections between these layers are called *vias*. They occupy the layers V0 (which links M0 to M1) and V1 (which links M1 to M2) and must have a square footprint.

via

Power Structure

Due to the ubiquity of both power potentials in the CMOS technology it is essential to construct circuits in a way that V_{DD} and GND can be accessed as easily as possible. This important constraint led to the scheme of *circuit rows*: The chip area, or large parts thereof, are divided into horizontal rows of the same height. Between these rows an easily accessible horizontal power rail is added to all metal layers—M0, M1, and M2. The electric potentials of these wires alternate between V_{DD} and GND every second row. The concept is illustrated in Figure 2.8.

circuit row

Most basic CMOS cells fit in exactly one of the circuit rows, so their height is prescribed by the technology and optimizing for their area corresponds to minimizing their width. Wires between neighboring rows are usually routed

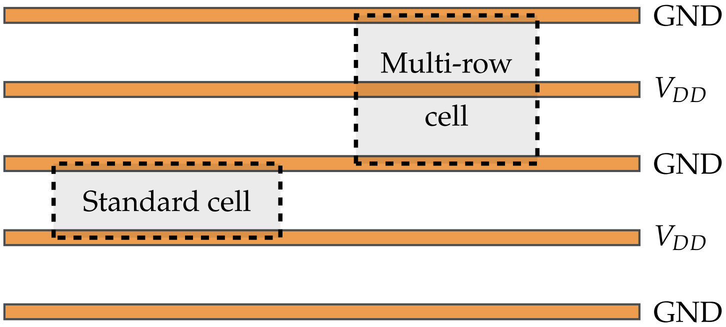


Figure 2.8: Power structure on a chip with two cell outlines.

Table 2.2: Properties of the cell model used in this work.

1. CMOS circuits with arbitrary topology (multiple outputs, non-uniform FET sizes, non-uniform Vt levels, non-dual structure, user-defined constraints). In particular, no natural pairing of n/p-FETs is given.
2. FETs may be reordered, swapped, and folded (single or multiple FETs).
3. Within a circuit row, transistors lie in two rows: n-FETs tend to be in the bottom row, p-FETs in the top row. This is not enforced.
4. All gates lie on a regular track grid.
5. In multi-row cells, the structure of every second row is vertically mirrored.
6. A GND power rail bounds the cell from below, a V_{DD} power rail from above.
7. Intra-cell routing is on PC (only connections between horizontally aligned gates) and three metal layers (cf. Table 2.1).
8. Connections between power and source/drain contacts lie on M0.
9. Routing between different circuit rows is on M1 in multi-row cells.
10. Routing is performed on a 3-dimensional virtual routing grid.

on a higher level of hierarchy in layers above M2 that are not considered in our model of transistor-level CMOS cells. However, we will also discuss multi-row cells that cover more than one circuit row. In this case parts of the power rails on M1 may be omitted to create space for intra-cell routing.

Within a circuit row, n-FETs are usually arranged in a row near the GND power rail, which may lie on the bottom or the top edge of the cell depending on the parity of the row index, and p-FETs are analogously placed near the V_{DD} power rail. The reason for this is that in CMOS cells n-FETs and p-FETs, if connected to the power at all, are mostly connected to the according potentials. This scheme has been employed by many authors during the last decades and is, mostly, followed in our presentation, too.

This finishes the discussion of the basic concepts of transistor-level layout as it appears today. Before theoretical aspects as well as the implementation of the new toolset BonnCell are presented, we revisit the existing literature regarding the topic. Although many of the aforementioned features already emerge in these works, the rapid progress of technology since the invention of integrated circuits and CMOS spawned a large number of different cell models, and motivated a likewise number of approaches to automate the task of generating their geometry.

Table 2.2 summarizes several important properties of our cell model in a semi-standardized format that has been used by several authors in the past, e.g. [MH91, GTH96, GH98, IIA04a]. Figure 2.9 depicts a full cell layout, including the power strips on the boundary, of a transistor-level netlists. The three variants belong to the same layout: The first two are 2-dimensional projections where shapes are colored by metal layer and electrical connectivity, respectively, and the third illustration shows its 3-dimensional structure.

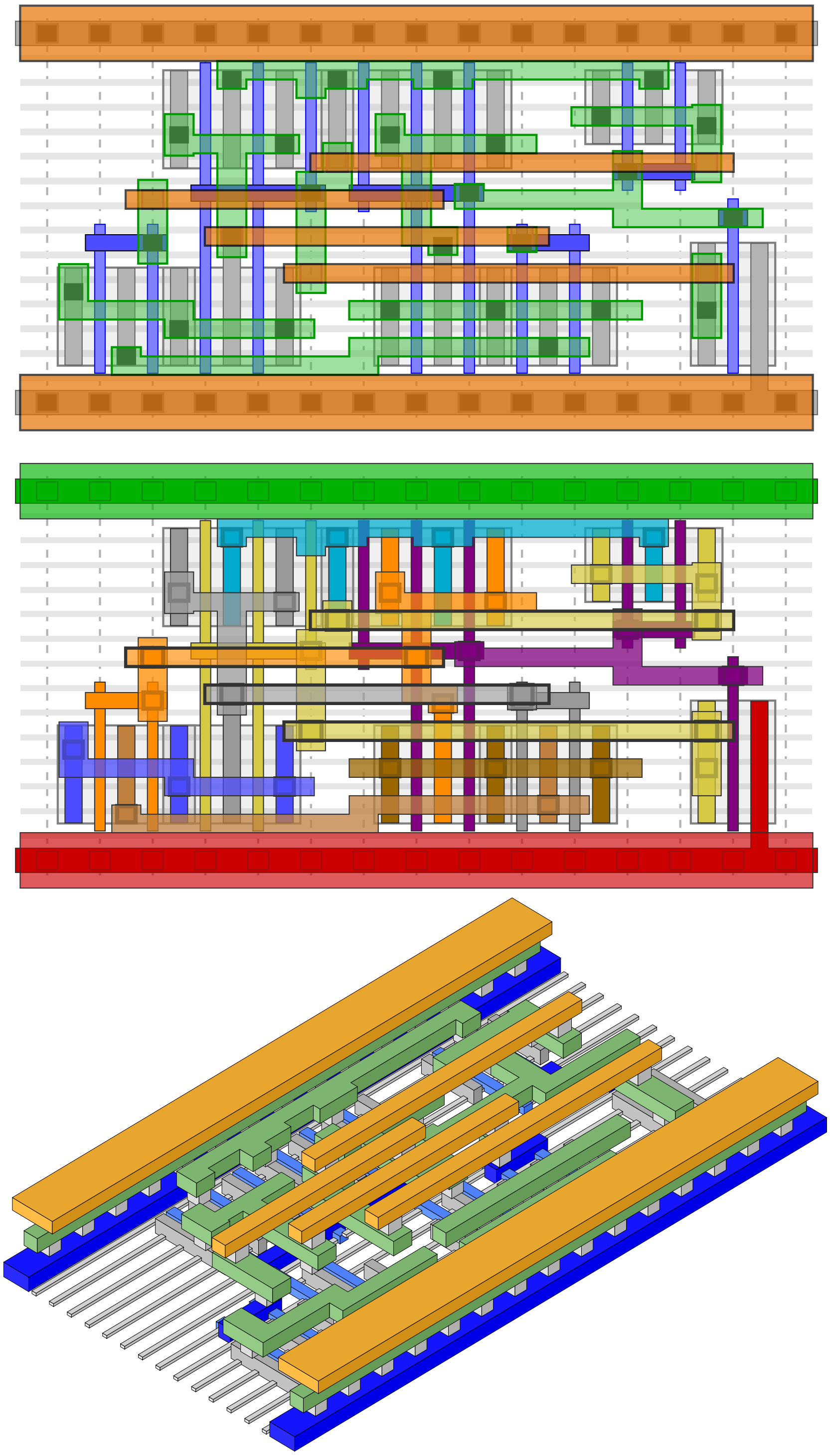


Figure 2.9: Example for a 2-dimensional standard cell layout using two FET rows within a single circuit row.

Chapter 3

Previous Work

The road to wisdom? – Well, it's plain
and simple to express:
    Err
    and err
    and err again
    but less
    and less
    and less.

Piet Hein, 1905–1996

As for many decades every new technology generation has been significantly more complex than the preceding one, the automatic layout of such units has become—and remained thenceforth—the topic of academic research. In this chapter, we follow the development of the literature discussing transistor-level layout, with a focus on static CMOS cells, in a roughly chronological order. We thereby group papers by their predominant features, like the folding of transistors as in Section 3.6, or by their algorithmic approach, e.g. the use of Satisfiability-based methods in Section 3.8.

3.1 Early Days of CMOS

In the years following the invention of the integrated circuit [Kil64], and subsequently the CMOS technology [Wan67], the number of transistors per chip and the importance of area was still small enough that layouts could be devised manually. The need for miniaturization and, consequently, automation did not pick up pace until ICs were used as switching elements in computers several years later.

The early attempts to automate the physical layout of integrated circuits are hard to classify using today's distinction between transistor-level layout and the global placement of precomposed standard circuits. These layers of hierarchy were not yet established and overall layout schemes were almost as numerous as authors publishing articles on their optimization. The only commonality of these works was the need of some layer of abstraction between the data structures handled by the algorithms and the geometric shapes of the final IC layout.

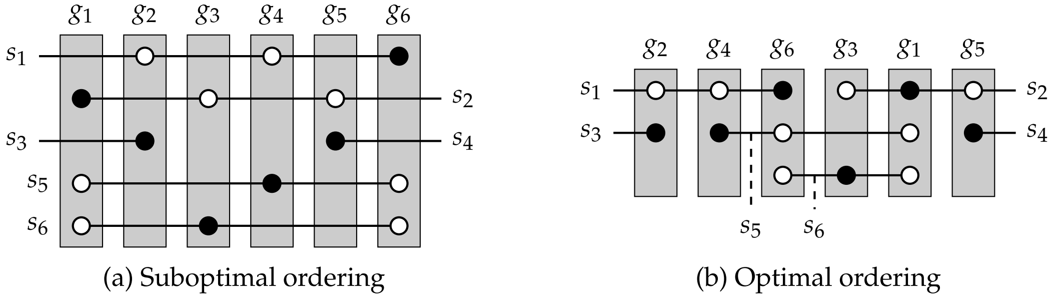
Feller [Fel76] presents a picture of "one of the earliest (1968–1969) LSI chips to be completely laid out using automatic placement and routing programs". The approach relied on a handcrafted standard cell library with circuits ranging from inverters and NORs to flip-flops and multiplexers. A tool called PRF (for "Placement, Routing, Folding") included a heuristic algorithm that arranges all required standard cells in a 1-dimensional row and adds the wires needed to connect the cells next to it. Then the row is wrapped around several times to achieve a roughly quadratic layout, leaving enough space between the segments of the row to hold the wrapped routing. Feller also mentions two subsequent improvements of the algorithms that directly arranged the standard cells in the 2-dimensional plane. However, as custom standard cells are used and important techniques like diffusion sharing are not employed, this approach cannot yet be considered as a transistor-level layout tool.

It was apparent that a layer of abstraction is needed between the geometric shapes of the final IC layout and, depending on the application, the data structures handled by algorithms or the human layouter who needs to conceptualize a cell layout. Gibson and Nance [GN76] suggested a notation that helps to generate good cell designs by hand. Although no automation was involved in this paper, their notation—or variants thereof—was also employed to develop layout software.

The usage of graph-theoretical terminology in the description of a placement and routing algorithm was introduced by Rose and Oldfield [RO71] in 1971. They used a computer to find an (almost) planar embedding of a graph that models a netlist and then to generate a geometric realization of the solution. Although some of the placement objects were transistors, abstracted as axis-parallel rectangles, the article referred to printed circuit boards. The system relied on human interaction to improve the otherwise unusable automatic result. Similar yet more elaborate techniques were applied in [EMP73] to integrated circuits.

3.2 Linear Gate Arrays

In 1967, Weinberger [Wei67] proposed a layout scheme that was frequently discussed in the literature. It utilizes the functional completeness of the NOR function, i.e. the fact that every logical function can be expressed only by using NOR gates. The scheme arranges a number of NOR gates in a horizontal 1-dimensional array. Each of those takes the form of a vertically elongated *n*-



(a) Suboptimal ordering

(b) Optimal ordering

Figure 3.1: Weinberger-type linear gate array. Vertical rectangles are NOR gates, horizontal lines are nets. The example was adapted from [SOH$^+$81].

channel MOS transistor that is gated by an arbitrary number of input signals. Figure 3.1a shows a schematic representation of such a circuit.

The combinatorial problem that arises is to choose the ordering of the transistors such that the horizontal connections on the metal layer can be performed with as few vertical tracks as possible. The layout in Figure 3.1b for example requires only three instead of five wiring tracks. The resulting logic circuit is bounded by the ground power rail near the bottom border and two other power rails, V_{DD} and V_{GG}, near the top border.

Weinberger gives several examples of how this "basic pattern" can be improved in specific situations. Although he does not propose a general algorithm for the optimization of such layouts, he does note that "the lack of standardization may preclude the use of computer aids" as part of the motivation for his layout scheme.

The first algorithms that automate the layout of Weinberger-type circuits are given in [Lar71]. One program reorders a sequence of custom-defined Boolean formulas and a second program, taking the reordered sequence as input, assigns tracks to variables and terms in order to generate an area-efficient circuit.

A reformulation of the problem in combinatorial terms and a precise definition of an optimization target that ultimately aims at the minimization of chip area can be found in [YKK75]. However, only two heuristic methods based on local improvements are described that do not guarantee to find a global optimum. In [AT78], the result is improved by describing a branch and bound method that finds optimum solutions in finite runtime. While the branching is limited for runtime reasons so that the output may not be optimal in practice, the methods usually find better solutions and require significantly less runtime than the heuristic approaches from [YKK75].

Ohtsuki et al. [OMK$^+$79] related the problem to interval graphs and applied graph-theoretical results to obtain a first hardness result. They represent the netlist by a graph (V, E) and observe that a Weinberger-type layout corresponds to an interval graph $(V, E \cup F)$ such that the number of required placement tracks is equal to this graph's clique number. Because finding a smallest augmentation F that minimizes the clique number is NP-complete, the problem of finding a track-minimal Weinberger layout for which a minimum amount of wiring is required is NP-hard. The authors then present a polynomial-time algorithm that finds at least an inclusion-wise minimal F with the required property.

The approach from [SOH$^+$81] is to generate an initial greedy solution and apply local improvement steps. Although they do not compute optimal solutions, they do include a method that restructures the logic of the netlist in order to reduce the area. Nevertheless, their layouts—portions of the random logic in a calculator—are 10%–40% larger than those laid out manually by skilled engineers.

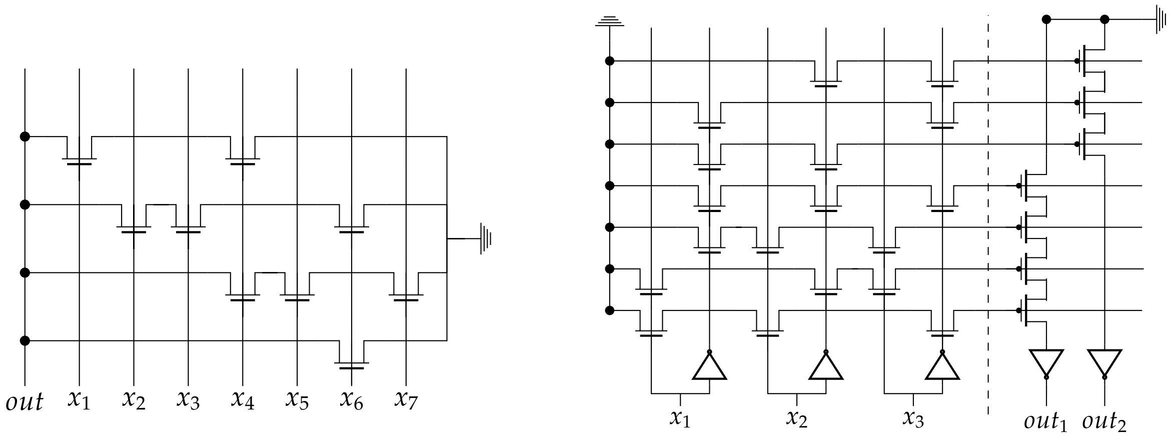
3.3 Two-Dimensional Gate Matrix

In 1980, Lopez and Law [LL80] proposed a new layout style for CMOS circuits that was picked up, modified and optimized by many other authors. Instead of a 1-dimensional array of transistors they use a 2-dimensional structure. Vertical columns correspond to polysilicon material serving as gate contacts and their interconnection at the same time. In other words, devices gated by the same signal are arranged in one column of the matrix that is associated with this particular signal. Transistors assigned to the same row are then connected in series with a horizontal connection on the diffusion or metal layer. The idea is illustrated in Figure 3.2a.

While the authors of this so-called "polysilicon oriented gate matrix" scheme do not provide algorithms optimizing such layouts, they claim that following this pattern greatly eases the design process. At the same time the area requirement of the results supposedly does not exceed the area of layouts created with older, more time-consuming methods. To simplify the handling of gate matrices, a machine-readable symbolic notation system, which is in part based on the system from [GN76], is introduced.

A similar approach was followed by [AMW82] for decoder-like CMOS circuits and extended to other logic functions in [PB83]. This "metal-oriented gate matrix" is based on the intersection of metal tracks with orthogonal diffusion lines. Polysilicon is added in parallel to the metal where transistors need to be inserted.

In [PZSB84], a workflow was presented that serves as a recipe for circuit designers to transform a set of logic functions into a metal oriented gate matrix. The method is based on Karnaugh maps ([Kar53]) and does not yield optimal results in any way. Moreover, the implementation of a CAD program is described that transforms the manually generated symbolic representation into a geometric drawing of a CMOS circuit. In the process, 11 different "mi-



(a) Gate matrix

(b) Programmable logic array

Figure 3.2: 2-dimensional matrix-like layout styles, adapted from [LL80] and [MH92].

crocells", i.e. possible configurations of a single entry of the gate matrix, are utilized. The authors report that using this system the efficiency of a designer increased from 4–6 transistors per day to 19–27 transistors per day.

A formal definition of the Optimum Gate Matrix Layout Problem and a multi-stage algorithm solving it was given in [WHW85]. The first phase of this method employs the equivalence of the 1-dimensional gate array problem to a minimization of the largest clique in an interval graph as presented in [OMK$^+$79]. To address the realizability as a gate matrix, a heuristic method to reorder the columns is used, probably involving the need to increase the distance between the polysilicon lines. Although the results of the algorithm are not optimal and several constraints which are important in practice are not considered, the authors provide the first formal and complete description as well as an implementation of a method that automatically generates gate matrix layouts from a netlist. An improved version of the method was later published in [HW89]. Several practical constraints like power routing, I/O gates, I/O nets and different transistor sizes are incorporated in this enhancement.

A further generalization of the formal description from [WHW85] was given in [WLL88]. The simulated annealing technique, which was introduced in [KGV83], is used to simultaneously improve the required number of tracks and the total wire length. Moreover, this approach allowed logically equivalent changes of the netlist topology. The algorithm provided by [SC90] also allows netlist modifications to achieve more compact gate matrices. A mincut based partitioning heuristic is applied that leads to an algorithm with an overall runtime of $\mathcal{O}(n \log n)$, where n is the number of equations required to describe the logic function of the cell.

Ho and Sastry [HS91a] introduced a variant called “flexible transistor matrix”, which utilizes one additional metal layer. Their algorithm is based on iterative vertical and horizontal min-cut computations and some local improvement steps. In addition, it supports different transistor sizes and is also able to create cells with a prescribed aspect ratio. Mainly due to the second metal layer, layouts are smaller than traditional gate matrices by one third.

Another class of regular 2-dimensional layouts which is frequently discussed in the literature is called programmable logic array (PLA). The typical structure of a PLA consists of an “AND plane” on the left with n-type transistors, forming a conjunction of input variables in each row of the matrix, and an “OR plane” with p-type transistors, connecting arbitrary product terms, i.e. rows, in a disjunction. Every column in the OR plane then corresponds to one output of the PLA. Figure 3.2b shows a schematic example of a PLA. While the structure of a PLA is easy to implement, it only supports sum-of-product-type circuits and does not use the available die area efficiently.

3.4 One-Dimensional Layout Style

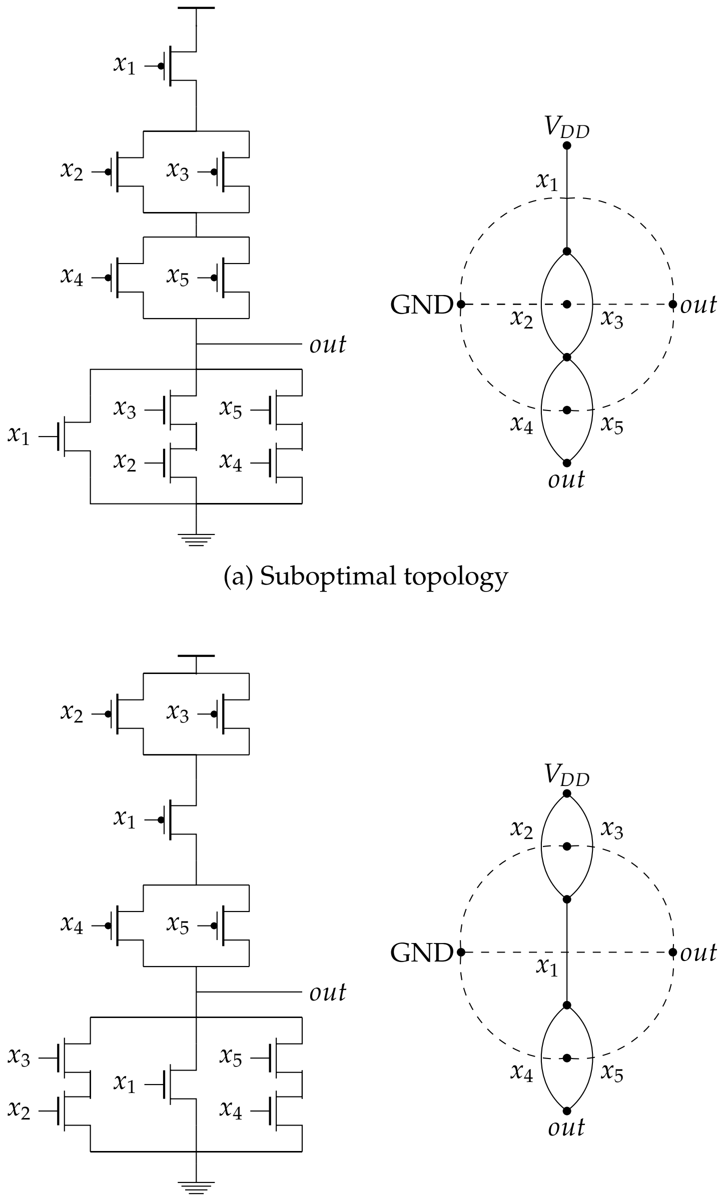
Uehara and vanCleemput [Uv79, Uv81] introduced a new layout scheme, 1-dimensional cells with two transistor rows, and used graph-theoretical notions as an interface between algorithms and the actual metal shapes of a cell’s physical layout. The authors observe that every logic circuit consisting only of AND and OR gates can be modeled by a transistor-level netlist in which the circuit’s output is connected to one potential, the logic 0, through a network of n-FETs and to the other power level, the logic 1, through a network of p-FETs.

3.4.1 Transistor-Level Netlists as Graphs

These two netlists can be represented by a pair of graphs, one for the n-FET network and one for the p-FET network. The vertices of the graphs correspond to the nets while each edge corresponds to a FET and connects the source and drain contacts of that FET. The edges are labeled with the gate signal of the FET, but other than that the nets connecting the gates are not represented in the graph model. Due to the way the logic function is transformed into a netlist, the resulting graphs are series-parallel and are in fact series-parallel duals of each other.

An example is shown in Figure 3.3a: It depicts the logic function $\neg(x_1 \vee (x_2 \wedge x_3) \vee (x_4 \wedge x_5))$ as the circuit diagram of a possible transistor-level netlist and the corresponding pair of series-parallel graphs. Note that the way in which the function is modeled as a netlist is not unique. Figure 3.3b shows another possible realization as well as its graphic representation.

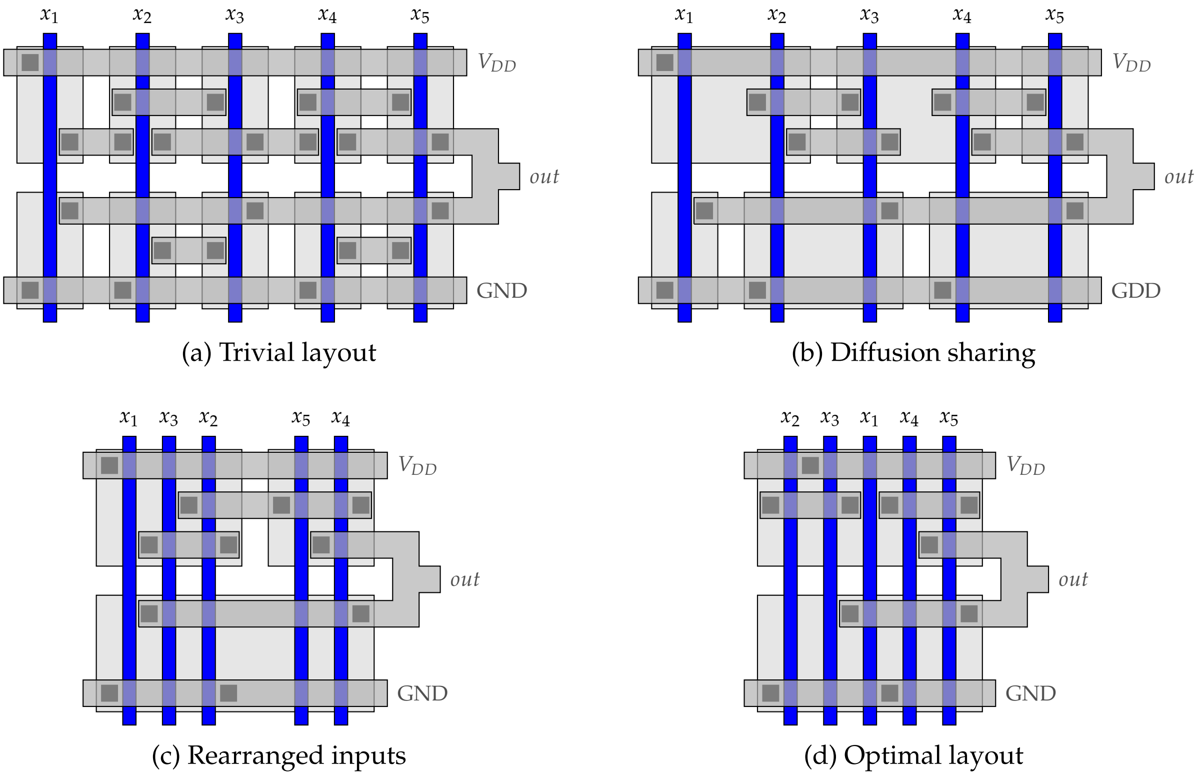
The layout scheme discussed by Uehara and vanCleemput is very simple, yet much literature uses the same model or more general versions thereof.



(a) Suboptimal topology

(b) Optimal topology

Figure 3.3: Two transistor-level netlists, shown as a circuit diagram and using the graph representation, realizing the function $\neg(x_1 \vee (x_2 \wedge x_3) \vee (x_4 \wedge x_5))$. Dotted lines model the n-FET network, the other lines model the p-FET network. The example was adapted from [Uv81].



(a) Trivial layout

(b) Diffusion sharing

(c) Rearranged inputs

(d) Optimal layout

Figure 3.4: Four cell layouts realizing the function from Figure 3.3. The example was adapted from [Uv81].

Even the standard method to implement most CMOS circuits today can be seen as a continuation of this layout style. The cells are laid out with one horizontal row of n-FETs near a GND power strip and another horizontal row of p-FETs adjacent to a V_{DD} power strip. By construction, every input signal is connected to a pair of FETs which are located at the same x-coordinate. In more complex cells it is also possible that input signals are connected to multiple pairs of gates, in which case the gates are connected with metal bridges above the V_{DD} power strip. All FETs in this model have the same size and exactly one finger. A trivial layout of the netlist from Figure 3.3a is shown in Figure 3.4a.

The key idea to optimize the area occupied by such layouts is to employ diffusion sharing. If two adjacent contacts of neighboring FETs belong to the same net, the connection can be made by connecting the diffusion areas of both FETs, eliminating the need to use the metal layer (Figure 3.4b). If this removes the diffusion gaps on both FET rows for a given x-coordinate, the cell size can be decreased by reducing the distance between the neighboring gates. To increase the number of tracks that can be saved, it is allowed to reorder the gates arbitrarily (Figure 3.4c). Furthermore, it is allowed to horizontally swap the FET. In the figure, this happens, for example, with the FETs gated by x_3. The topology of the transistor-level netlist also matters: As shown in Figure 3.4d, the realization in Figure 3.3b can be laid out without any gaps in the diffusion—this is not possible using the other netlist topology.

Uehara and vanCleemput observe that a sequence of transistors that can be placed without diffusion gaps corresponds to a path in the netlist's graphic representation. They conclude: if there is a sequence of the input signals such that the corresponding edge progression forms a Eulerian walk in both graphs, then this gate order leads to an optimal layout, i.e. a layout without any gaps in the diffusion area. The sequence $(x_2, x_3, x_1, x_4, x_5)$ has this property in the graphs from Figure 3.3b, and no such sequence exists for the graphs from Figure 3.3a. In general, the cell size can be minimized for a fixed netlist topology by finding a minimum-size set of sequences of input signals such that every input signal is contained in exactly one sequence and the edge sets corresponding to these sequences are walks in both graphs. Such a solution would be $\{(x_1, x_3, x_2), (x_4, x_5)\}$ for the example in Figure 3.3a.

3.4.2 Algorithmic Approaches

Based on these observations, a polynomial-time heuristic algorithm is proposed that does not guarantee to find optimal solutions. Based on the fact that solutions without gaps can easily be found if the netlist is generated only by AND and OR gates with an odd number of inputs, the approach first inserts pseudo-inputs to gates with an even number of inputs, then solves the problem optimally, and finishes by introducing diffusion gaps when removing the previously inserted pseudo-inputs.

Nair et al. [NBR85] describe an algorithm that returns a provably optimum solution taken over all possible netlist topologies. If the optimum solution happens to be devoid of diffusion gaps, the algorithm even runs in linear time. In other cases, no polynomial runtime can be guaranteed. The algorithm of Maziasz and Hayes [MH87] finds a solution with the minimum number of gaps in linear runtime, but it does assume a fixed topology for the netlist. A parallel version of the algorithm that runs on a linear number of processors and finishes in logarithmic runtime was presented in [HS91b]. Finally, Nyland and Reif [NR96] found an algorithm that solves the original problem, i.e. minimizing the number of gaps over all placements and all logically equivalent netlist modifications, in linear runtime.

A natural extension is to examine a larger class of graphs. While simple CMOS cells are indeed representable by a pair of dual series-parallel graphs, more complex cells do not employ this specific netlist structure. For this purpose *two-terminal graphs* (TTGs) have been studied in the context of transistor-level cell layout. A two-terminal graph $G = (V, E)$ is a planar graph with a fixed embedding and vertices $s, t \in V$ on the outer face such that $G + \{s, t\}$

is 2-vertex-connected. The problem formulated by Uehara and vanCleemput can also be posed for such graphs.

In this case, a proof is given in [UTK88] that it is NP-hard to find a minimum number of edge sequences that correspond to walks in both the given TTG and its dual. This is true even for a fixed netlist topology. However, if one only asks for a single edge sequence that corresponds to a Eulerian walk in both graphs, the answer can be found in linear time [CCM95]. The result was later extended [CHH99] by showing how to decide in linear time if a logically equivalent netlist exists for which such a dual Eulerian walk exists.

For a fixed topology, the problem of finding a minimum number of transistor chains is solved by the exponential-time algorithm given in [HHLH89]. However, the formation of p/n-transistor pairs is done heuristically, so no optimal solution is found in general.

Müller and Lengauer [ML86, LM88] considered a similar problem for dynamic CMOS gates. In this case only a single row of FETs must be ordered, reducing the problem to single series-parallel graphs. They consider two approaches to the problem: The first is again the minimization of diffusion gaps, i.e. finding a topology with a series-parallel graph representation that can be covered with a minimum number of walks. The other approach is based on the idea that one can replace a transistor by two identical parallel transistors, both gated by the same input signal. In the graph representation, this corresponds to the duplication of an edge. Hence, one can also ask for the netlist topology that minimizes the number of edge duplications required to make the graph Eulerian. The authors attribute these formulations to an unpublished work of R.H.J.M. Otten related to the Yorktown Silicon Compiler [BBC$^+$85], a chip design toolset developed by IBM during the 1980s.

Müller and Lengauer devise linear-time algorithms for both problems, i.e. minimizing the number of paths required to cover the graph as well as minimizing the number of input duplications. Furthermore, they provide an example for a class of series-parallel graphs that can be covered with $\mathcal{O}(1)$ paths but require $\Omega(n)$ edge duplications to make them Eulerian, where n is the number of FETs. As the number of duplications is always an upper bound for the number of paths, they suggest that minimizing the number of diffusion gaps is the most useful approach to optimize the size of CMOS cells.

McMullen and Otten [MO88] found a linear-time algorithm for the case of a single row of transistors with a series-parallel structure even when logically equivalent reorderings of the netlist are allowed.

3.4.3 Computational Complexity

While minimizing the number of required gaps was shown to be easy in many cases, it is in addition desirable to find an optimal solution that minimizes the number of required routing tracks among all optimal solutions.

This number is usually defined in the sense of a Weinberger-type transistor chain (Figure 3.1). For this problem, Chakravarty et al. [CHR91] proved NP-hardness even in the restricted case of a single transistor row with an underlying series-parallel network structure for which no topology modifications are allowed. In order to obtain this result, they reformulate the task as the search for a Eulerian walk in a series-parallel graph that minimizes the clique number of an associated "track graph" and reduce an NP-hard restriction of the bin packing problem to it.

3.5 Extensions of the One-Dimensional Layout Style

The cell model proposed by Uehara and vanCleemput could easily be modeled in graph-theoretical terms, yet it was very restricted in respect to the variety of the cell layouts that could be represented. As the technology continued to develop, many authors extended the model to adapt to new possibilities as well as requirements arising in the manufacturing of CMOS cells.

The software TOPOLOGIZER presented in [KW85] automatically generates 1-dimensional cell layouts with two transistor rows in which n-FETs and p-FETs can be placed jointly in both rows. The placement phase starts with a random placement and iteratively applies exchange operations that improve the number of neighboring contacts that connect the same net. Here, "neighboring" does not only mean adjacent transistors on the same row but also contacts on the same track but different rows. TOPOLOGIZER also provides a routing module. Similar to the placement, it starts with a very simple routing before iteratively applying rules that locally improve the layout.

The toolset from Wimer et al. [WPF87] was designed to automatically generate two-row cell layouts without requiring the netlist structure to be representable by a pair of dual series-parallel graphs. In this work, FETs can be paired if they share a gate, source or drain contact. The method first determines transistor pairs, finds chains of those pairs, selects a minimum—and, for a given target function, best—set of chains to realize the cell, and then enumerates the $2^n n!$ possibilities to place the chains (where n is the number of chains). If n is too large, a random sample of placements is evaluated. The algorithm requires an exponential runtime but allows for a flexible multi-criteria target function rather than just optimizing cell area. Wimer et al. also describe a heuristic channel routing approach to generate wires.

In [SSK$^+$88] a target function is introduced that primarily optimizes the cell area, but as secondary criteria considers routability modeled by gate alignment and total net length. The algorithm aims to identify known functional groups in a cell, like an `AND` construct, and replaces those by predefined optimal arrangements. The authors also describe a channel routing algorithm to compute the wiring.

Several graph-based techniques like computing Hamiltonian paths, bipartite matchings, and placements with optimal quadratic wire length are used by Chen and Chow [CC89] with the goal to compute transistor placements with a small area and good routability at the same time. No assumptions regarding the structure of the netlist are made and FETs of varying sizes are supported. The algorithm does not find optimal solutions, albeit having exponential runtime, but is still able to generate a cell layout with 192 transistors.

Madsen's method [Mad89] includes the possibility of prescribed locations for external connections, giving the user more control over the cell layout based on other nearby cell. It supports variable and fixed netlist topologies, but Madsen does neither prove optimality in any sense nor does he show a polynomial runtime.

Handling routability considerations in a provably optimal fashion was first done by Bar-Yehuda et al. [BYFPW89]. They define a target function with 3 criteria, namely cell area, gate alignment, and total wire length, and describe a branch and bound algorithm that finds a lexicographically best placement for which a simple track-based routing exists. The tool is able to handle arbitrary netlist structures.

GENAC [OLL89] also follows a branch and bound approach to find, after an initial rule-based pairing of n/p-FETs, an area-optimal linear ordering of the transistor pairs, while at the same time looking for an optimum solution with low routing density. In addition, the authors mention that the transistor folding does not happen in a trivial greedy fashion, but as to minimize a cost function that, although heuristic, is meant to improve the resulting cell area.

Another graph-theoretic approach was employed by GRAPES [HF87, HF92]: After an initial min-cut based clustering phase that splits large instances into modules of around 100 transistors, the input ordering is determined by encoding the circuit diagram as PQ trees. Because PQ trees represent sets of permutations, the authors propose a method to merge two such trees, each corresponding to one of the two transistor rows.

The goal of PARROT PUNCH [LSR06] is to arrange a large number of transistors in 1-dimensional circuit rows such that the resulting layout has favorable timing properties. The tool takes only a specification of the cell's logic functionality rather than a full netlist as input. Starting from this specification, it generates the netlist such that the estimated placement size is minimum, reducing the number of required transistors compared to standard cell schemes in the process. Timing analysis is performed after the layout generation and used to resize individual transistors on the critical path through a cell. This saves area compared to the usual approach which applies timing optimization on a gate-level netlist.

From the perspective of modern manufacturability issues, 1-dimensional cell layout is performed in [WLC^{+}13]. Much like in older works the authors strive to maximize diffusion sharing while achieving a small number of required routing tracks, but in addition solutions are preferred which have a

small number of "stage-like line-end gaps". These unwanted structures, gaps in the metal strips on neighboring tracks with a small but positive offset, proved to be problematic in the manufacturing of the chip.
For the placement algorithm, the authors show that sets of 3 transistors that can be placed without a gap can be found by applying Ullmann's subgraph isomorphism algorithm [Ull76] to a certain graph that models the netlist and one of six "connection subgraphs" of constant size. To find all placements with a minimum number of diffusion gaps the algorithm enumerates sets of such elementary chains that may overlap. Then all possible placements of minimum size are evaluated in terms of routing density and expected stage-like line-end gaps and the best one is returned. In the detailed layout generation after the placement, a maximum independent set is computed to optimize the number of tracks required on the second metal layer. Finally, a branch and bound method finds a routing on this layer that is optimal in terms of the proposed measure for the manufacturability.

3.6 Stacked Devices

Hill [Hil85] presented the first system, called SC2, that was able to perform FET folding (cf. Section 2.3.1), also called *stacking* in the academic literature, and thereby introduced one of the most important features of today's CMOS layout. SC2 also splits large cells into smaller sub-instances, subsequently processing each of those independently. The placement algorithm first couples n-FET/p-FET pairs with the same gate net, then rearranges these pairs using a Kernighan-Lin heuristic, and finally flips single transistors in an exponential yet practically feasible branch and bound method such that the number of diffusion gaps is as small as possible. The folding of transistors is done by simply splitting large FETs into smaller ones and connecting them in parallel. The author states that a practical upper limit for the number of FETs is 200–400.
In the development of POLLUX [MD88] not only cell area was targeted, but the routability was considered as well. The resulting algorithm finds input orderings with a small (yet not necessarily optimal) number of diffusion gaps. Solutions are preferred if a routing exists that does not require more than a given number of horizontal tracks. The author's approach also supports transistors of non-uniform size, including a greedy a priori folding of transistors which are too large to fit in a row. Moreover, additional layout styles using two metal layers and power strips near the cell's center are proposed.
After folding has been incorporated in very simple ways, if at all, into automatic layout systems during the 1980s, Gatti et al. [GML89] defined what they called "full stacked layout", thereby initiating the systematic research on transistors with flexible aspect ratios. The authors propose to exploit the

fact that very long transistors, whose geometric form traditionally needed a large aspect ratio, can also be realized using more than one gate contact by connecting the alternating source and drain contacts with comb-like wiring. With this technique, the geometry of the FETs can be closer to a square, a property which usually has a good impact on placement tools. In [GH98] four variants concerning the possible interdependence of folding and placement algorithms were formulated:

- *Static placement, static folding:* For given orderings of the FET rows, i.e. a placement, and given finger numbers for each FET, find orientations for the FETs such that the number of diffusion gaps, i.e. the cell area, is minimized.
- *Static placement, dynamic folding:* For a given placement, find finger numbers and orientations for the FETs such that the cell area is minimized.
- *Dynamic placement, static folding:* For given finger numbers, find a placement, including orientations, such that the cell area is minimized.
- *Dynamic placement, dynamic folding:* Among all possible folding configurations and placements, including orientations, find the solution that minimizes the cell area.

A notable example for a system with static placement and dynamic folding is LiB [HHLH91]. The layout generation tool roughly follows the classical two-row style, but reserves non-rectangular areas between the rows for the routing. After forming initial FET clusters to reduce the instance sizes to a manageable magnitude, an optimal transistor chain is formed for each cluster. The placement of the chains is done using a Kernighan-Lin type heuristic to reduce the routing density as much as possible. After a placement has been decided upon, a folding module chooses finger numbers and FET orientations.

An exact algorithm for the problem of static placement and dynamic folding is presented by Her and Wong [HW93]. Among all minimum-width placements the algorithm returns the folded placement which minimizes the cell height, defined as the maximum of the sum of the n-FET's height and the p-FET's height on a given track. The runtime of the dynamic programming approach is polynomial in the number of required placement tracks.

An extension of the technique in [WPF87] is presented by Malavasi and Pandini [MP95]. It reduces symmetries and combines the algorithm with a branch and bound approach, yielding an algorithm that does not only arrange transistors within the cell but also determines favorable aspect ratios, i.e. folding styles, for every transistor. The algorithm provably optimizes a complicated target function that models cell area as well as favorable electrical properties.

XPRESS [GTH96] supports transistor folding in a framework that primarily tries to minimize the cell width but also targets the total wire length as a secondary optimization goal. Although the transistor chain generation is exhaustive in nature and thus finds optimal solutions among all possible transistor foldings, the heuristic parts of the method like the pairing of n- and p-FETs, which is performed before the placement algorithm starts, prevents it from finding globally optimal solutions. XPRESS supports arbitrary netlist structures but does not modify the given topology.

Instead of just the area, the algorithm from [BR96] minimizes a target function that favors diffusion sharing but also incorporates performance considerations in the form of criticality weights for parts of the cell. In fact, area minimization is the special case for uniform criticality weights. The method also addresses specific symmetry constraints appearing in analog circuits. It finds in linear runtime a global minimum of the target function for a given folding of the transistors by covering a modified circuit graph with a minimum number of walks.

CELLERITY [GMD$^+$97] supports transistor folding in an exhaustive way: For every possibility to fold a subset of the transistors, the whole design flow is executed and the layout with the fewest number of tracks is returned. Instead of just allowing the classical 1-dimensional cells with two rows, a three-row image with n-FETs in the center and p-FETs near the top and bottom cell border as well as a four-row image with up to two vertically neighboring n-FETs near the bottom and two p-FETs in the top half are supported. The software is able to place external ports within the cell's boundary and includes a maze router that is not limited to a single routing layer.

An algorithm that finds in time $\mathcal{O}(n^2 \log n)$, where n is the number of transistors, a folding configuration with a provably optimal cell area is presented in [KK97]. In this work, the width of the 1-dimensional cell is only influenced by the number of fingers in the two FET rows and the height is only influenced by the highest n-FET and the highest p-FET. Diffusion sharing is not considered in this model.

Assuming a dual netlist structure in which n-FETs and p-FETs always come in pairs, [Ber01] describes a dynamic programming approach that optimally solves the problem of dynamic placement and dynamic folding as an extension of the branch and bound method introduced in [BYFPW89]. The primary optimization target is the cell width and the secondary target models the desire to use for each FET, if possible without increasing the total width, as few fingers as possible.

Theoretical results concerning the computational complexity of several variants of the problem to find minimum-width transistor chains have been found by Weyd [Wey11]. He augmented the linear-time solvable problem as considered in [ML86] with several aspects employed in transistor-level layout since then, including folding (with prescribed or variable finger numbers) and more complex rules regarding the distance between neighboring FETs. We revisit and extend these results in Section 4.2.

3.7 Mixed Integer Programming

Integer Programming as a technique to optimize the layout of CMOS cells was introduced by Gupta and Hayes [GH96]. The authors study a 2-dimensional cell model which encompasses a given number of vertically adjacent 1-dimensional rows of n-FET/p-FET pairs. The total cell width is given by the longest of these rows, each of which is influenced by the number of transistor pairs, the number of required diffusion gaps, and the number of wires that need to be routed across multiple rows. To minimize the total cell width, an integer linear program (ILP) is proposed that computes a covering of the circuit graphs by dual walks as well as an assignment of the walks to the circuit rows. Linear constraints are presented that yield an optimal solution in terms of total cell width for a given pairing of n- and p-FETs. The number of variables depends exponentially on the number of FETs because there is a set of variables for every possible dual walk in the circuit graphs.

The authors report that layouts with up to 24 transistors could be handled in practical runtimes of a few minutes. Moreover, the flexibility of the method was limited to single-finger transistors. For these reasons, Gupta and Hayes described several extensions of their method. In [GH97a] the ILP-based algorithm was integrated in a hierarchical flow with the goal to handle larger cells, even if optimality was lost. Using XPRESS [GTH96] as a 1-dimensional placement tool, all dual walk covers of the circuit graphs are determined that lead to minimum-size layouts. The resulting set of dual walks is then used as input for the previously described ILP.

The tool CLIP [GH97b] uses this ILP formulation to determine the minimum width of a 2-dimensional cell layout and then solves a second ILP to find, among all 2-dimensional cell layouts of that width, one that minimizes the cell height by minimizing the required horizontal routing tracks. In addition to the formulation of the ILP that minimizes the channel routing density, the authors reformulate the width minimization ILP to only include a polynomial number of variables and constraints. In [GH98], support for transistor folding is included in the ILP formulation. However, the folding must be determined in advance because the integer program takes a given folding as input.

Cortadella [Cor13] discusses folding from another point of view: In the given technology it is not allowed to share diffusion area between neighboring FETs if the FETs do not have the same height. The folding is constrained in that an interval of possible gate lengths is given for each FET. If for example a FET must have a total gate length in $[9, 10]$, then it can be realized with 1 finger and a height of 9 units, with 2 fingers and a height of 5 units, and so on. This is also the first paper that does not require the single fingers of a folded transistor to be placed directly next to each other and which allows the size of a diffusion gap to have a parameterized size.

The algorithm takes an arbitrary transistor-level netlist and computes a folded netlist in which all transistors have only one finger for which an area-optimal placement is guaranteed to exist. To do so, a mixed integer linear program (MIP) is devised with a polynomial size that assigns finger numbers to all FETs such that the minimum number of diffusion breaks is as small as possible. The formulation is based on a suitable graph model and Eulerian paths. Due to a weakness in the model the MIP solver has to be embedded into a flow in which, in principle, exponentially many MIPs must be computed. The method can be extended to multi-row cells, single FETs that vertically span a whole row, and wire length estimation.

3.8 Satisfiability

In the 2000s, several aspects of the transistor-level layout problem have been formulated in terms of the satisfiability (SAT) problem. The task of SAT is to decide for a given logic formula if values can be assigned to the variables such that the formula evaluates to true. Although it is well-known to be NP-complete [Coo71], software solving SAT instances started to be powerful enough to solve formulations of the addressed layout problems in practically viable runtime.

Iizuka et al. [IIA04b] were the first to transform the transistor-level placement problem to SAT. Their formulation uses variables $x_{t,i,b}$, where $t \in \{N, P\}$ represents the type, $i \in \{1, \ldots, n\}$ is the index of the transistor pair, and $x_{t,i,k-1} x_{t,i,k-2} \ldots x_{t,i,0}$ is the binary representation of the index of the track at which the i-th transistor of type t is placed. Note that if W is the width of the cell, then $k := \lceil \log_2 W \rceil$. Additional variables $y_{t,i}$ encode if the i-th transistor of type t is flipped or not. With these definitions, the following SAT instance has exactly the legal 1-dimensional dual placements as feasible solutions:

$$\bigvee_{0 \le b < k} x_{t,i,b} \oplus x_{t,i',b} = 1 \quad \text{for } 1 \le i < i' \le n \text{ and } t \in \{N, P\}$$

$$\bigvee_{0 \le b < k} x_{t,i,b} \oplus u_{j,b} = 1 \quad \text{for } W \le j < 2^k, 1 \le i \le n \text{ and } t \in \{N, P\}$$

$$\bigvee_{i' \in G_i} \overline{\bigvee_{0 \le b < k} x_{n,i,b} \oplus x_{p,i',b}} = 1 \quad \text{for } 1 \le i \le n$$

$$GAP(t, i, i') \wedge \left(\bigvee_{0 \le j < W-1} C(t, i, j) \wedge C(t, i', j+1) \right) = 0$$
$$\text{for } 1 \le i < i' \le n \text{ and } t \in \{N, P\}$$

Here $u_{j,k-1} u_{j,k-2} \ldots u_{j,0}$ is a binary representation of the integer j and G_i is the set of all p-FETs whose gates are connected to the same net as the gate of the i-th n-FET. The variable $GAP(t, i, i')$ is supposed to equal 0 if and only if the i-th and i'-th FET of type t can overlap if the first is placed directly to the left of the second. The variable $C(t,i,j)$ is 1 if and only if the i-th FET of type t is placed on track j. These conditions can be enforced by several additional clauses that were omitted in the presentation above. To find a minimum-width layout, W is set to n and then increased until a SAT solver finds a feasible solution.

With a similar technique, the routing problem is formulated as a SAT instance. The nets are split into groups with specific properties and it is assumed that all nets from one of the groups are routed in a uniform way that can essentially be parameterized by a single number, namely the index of a routing track. These numbers are represented by bit vectors and a polynomial number of clauses are formulated that are satisfiable if and only if such a constrained routing exists. To find a routable placement, solutions from the placement phase are checked for routability and excluded by a new clause if they turn out to be unroutable.

In [IIA04a] the same authors also look at the layout generation problem from the perspective of yield. The target function is the minimization of the fault probability which correlates to the total critical area, i.e. the area in which a defect must occur in order to produce a broken cell. The critical area is minimized by enumerating all minimum-width layouts using the SAT-based technique and evaluating the fault probability for each of them. The largest cell that can be handled within an hour has 12 single-finger FETs.

The placement algorithm in [TP07] is an extension of [HHLH89]. The branch and bound method finds a placement that provably minimizes the cell width, channel density and estimated wire length in a lexicographic fashion. However, the routing module also facilitates a transformation to SAT. Unlike [IIA04b], this formulation associates variables with edges in a grid graph rather than tracks. Net indices are encoded by splitting a binary representation of the numbers into individual bits. In order to minimize the wire length, the authors provide a set of clauses that are only satisfiable if at most M grid edges are used by wires, subsequently performing a binary search to find the best possible value for M. The flexibility to encode forbidden patterns in the SAT instance is employed to support a limited set of geometric ground rules.

3.9 Free-Form Two-Dimensional Layouts

Although the 1-dimensional style proposed by Uehara and vanCleemput [Uv81] and variations thereof has dominated theoretic advances as well as practical implementation of transistor-level layout tools, some authors have considered layout styles that extend to the 2-dimensional plane without being restricted to prescribed rows and without being constrained to a highly regular structure as employed by gate matrices or PLAs.

An early example describing a method for the automatic layout of NMOS cells is due to Luhukay and Kubitz [LK82]. The placement strategy comprises a set of heuristics that arrange standard cells in a "better than random" way and applying local optimization steps. However, the geometry generation phase does not just insert handcrafted layouts from the standard cell library, but uses algorithmically defined "subcells" to generate the shapes based on design rules and device sizes. It is also worth mentioning, because uncommon at the time, that one of the possible layout schemes uses two instead of just one metal layer for the wiring.

The toolset presented in [WNMD83] has several characteristics of more modern VLSI tools aimed at a higher hierarchy level. It consists of a force-directed placement routine working directly on transistors, a torque-directed orientation routine rotating the placement objects, a wiring phase that connects nets with minimum spanning trees or a track-based method, an assignment of 2-point connections to the available layers and a routing step that heuristically realizes 2-point connections with L shapes such that the number of shorts is as small as possible.

In [NJ85] the netlist is transformed to several auxiliary graphs to which several algorithms are applied in order to obtain a realizable embedding of the netlist into the three available layers, namely diffusion, polysilicon and metal. The techniques include a partitioning into maximal 2-vertex-connected components, a subroutine of the planarity testing algorithm by Hopcroft and Tarjan [HT74] that generates a hierarchy of paths covering a graph, and a 2-coloring satisfying certain properties.

A 2-dimensional layout scheme which is based on the formation of n-FET/p-FET pairs is proposed in [MAU86]. The placement algorithm first computes a heuristic 1-dimensional array of the transistor pairs and then folds this row in a zig-zag fashion. Later, local improvement steps are applied. The routing module consists of a modified maze router [Lee61] that works on the single metal layer which is not used by the transistors themselves.

Poirier [Poi89] describes a system that arranges arbitrary CMOS netlists between two horizontal power strips. However, the distance of the power strips is not fixed and the tool is able to place FETs in more than two rows. The heuristic method groups transistors to clusters and enumerates possible placements of those clusters, including possibilities to fold single FETs.

In [FSA95] transistors are grouped by simulated annealing into chains which can be placed without diffusion gaps. The chains are abstracted as rectangles and placed with a floorplanning technique that involves recursively slicing the cell area with vertical and horizontal cuts. The transistor chains can be rotated by multiples of 90 degrees such that layouts are generated that have gate contacts with non-uniform direction. After the routing step, which is based on a depth first search on a regular grid graph, a final compaction method transforms the symbolic layout into geometric shapes.

Rekhi et al. [RTL95] describe what they call 1-$\frac{1}{2}$-dimensional cells. Here a single row of transistors is, as usual, arranged near each of the two horizontal power rails bordering the cell outline at the top and the bottom side, but instead of forcing the n-FETs near the GND potential and the p-FETs near the V_{DD} potential, n-FETs and p-FETs are allowed to mix in both of the transistor rows. The algorithm uses a heuristic to pair transistors that may be of the same type, then computes groups of transistor pairs that can be placed without diffusion gap, and finally places the groups in a branch and bound method minimizing the number of connections which have to cross a gap.

AKORD, a software by Serdar and Sechen [SS99], combines the free-form 2-dimensional layout style involving vertical and horizontal gates with transistor folding. In addition, "black boxes" are supported—already laid out circuits that can be inserted into a layout and form a new level of hierarchy. The placement method is based on simulated annealing and primarily targets the total wire length. In this approach, transistor grouping and folding can be handled within the placement phase and must not be fixed in advance.

Riepe and Sakallah [RS99, RS03] describe a full 2-dimensional cell layout tool, TEMPO, that rotates transistors by multiples of 90 degrees. Similar to previous 2-dimensional approaches, the placement step is based on simulated annealing. However, it allows more degrees of freedom as it does not rely on statically formed chains of transistors but reconfigures chains and reassigns transistors within the placement phase. Other differences to earlier systems are that TEMPO uses the sequence pair representation of rectangle packings ([MFNK95]) to explore the search space and that it allows for more complex cases of geometry sharing between three or more transistors.

3.10 High Regularity

In recent years, the continuing miniaturization of geometric structures combined with the non-decreasing wavelength of the lasers that are used to manufacture chips led to an explosion of the number and complexity of shape-based design rules. For example, the polysilicon strips forming the gate have a width of approximately 10% of the wavelength used to produce them. In order to attenuate the burden imposed by this problem, both for tools and engineers, much of the literature discussing transistor-level layout has returned to highly regular structures like 2-dimensional gate arrays and similar concepts. The ideal geometry of the metal layers, rectangles of the same size that are arranged in a regular 2-dimensional grid, have been entitled lithographer's dream patterns [MLMS07]. Several approaches were proposed to generate cell layouts that are as close to those patterns as possible.

One such approach is called *via-programmable*: Fixed blocks of transistors and wiring are used to fill the chip area, or a part of it. The varying functionality of the logic blocks is then added by inserting vias at the appropriate positions. The task is then to create such predefined blocks which lead to a minimum number of unused transistors.

A new type of gate array is introduced by Lin et al. [LMSM10] who use so-called VeSFETs (Vertical Slit FETs). This type of transistor has a square footprint on which two diagonally opposing corners form the gate contact and the other two corners are the source respective drain contacts. Every metal layer allows only wires in one direction, but for all multiples of 45 degrees a layer exists that is reserved for wires in this direction. To compensate the VeSFETs' disadvantage that no geometry can be shared between two transistors, the square-shaped transistors of uniform size are packed without any gaps so that 100% of the chip area is occupied by active diffusion areas.

The placement is performed by simulated annealing in two phases. In the first phase the aim is to minimize the number $C_{resource}$ that estimates the routing difficulty after a set of direct connections have been routed without detours, and in the second phase the ratio $\frac{C_{cut}}{C_{resource}}$ is maximized, where C_{cut} is defined as to model the amount of difficult global connections. The routing method also runs in two phases: In the first one a greedy routing iteratively applies a path search to compute the unexpectedly unproblematic connections and in the second phase a SAT formulation is used to find routings for the remaining nets.

Ryzhenko and Burns [RB11] present an intermediate approach. They modify placements of custom cells to match a highly regular pattern on all layers. Only a few routing candidates are considered for every net, each of which also satisfies certain regularity properties. In a branch and bound algorithm the wire candidates are enumerated until a combination of candidates has been found that is void of conflicts. The authors later provided a SAT formulation [RB12], embedded in a multi-stage routing flow, to improve their approach.

3.11 Summary

Until today a wide range of different methods have been employed to automate the layout of CMOS cells, including graph-theoretical methods, branch and bound, (mixed) integer programming, satisfiability models, and so on. Moreover, many different layout schemes have been proposed that roughly fall into the categories 1-dimensional (row-based), 2-dimensional, and structured (e.g. gate matrices)—even though the interpretation of these terms differs and no distinct boundaries between them were established. The dominating design strategy for contemporary integrated circuits primarily resembles the 1-dimensional style whose investigation originates in Uehara's and vanCleemput's seminal paper from 1981.

The algorithms presented in the following chapters can be classified somewhere in between 1-dimensional and 2-dimensional approaches. While our cell model, as explained in Section 2.3, also employs a circuit row structure with intermediate power rails, the placement within the rows has several crucial properties that exceed the traditional structure of 1-dimensional cells. Most notably the transistor rows in a single circuit row cannot be regarded as two independently arrangeable objects, but must be laid out together as they compete for a common area that can be covered by at most one of the rows. Moreover, depending on the parameters, n-FETs and p-FETs may mix in a single row of transistors.

Chapter 4

Transistor Row Placement

> Row, row, row your boat,
> Gently down the stream.
> Merrily, merrily, merrily, merrily,
> Life is but a dream.
>
> Traditional nursery rhyme

This section begins with several definitions to provide a basis for the formal discussion of the 1-dimensional layout problem: the arrangement of transistors in a single horizontal row. Afterwards, we provide a graph-theoretical framework, which is related to covering graphs with walks, that eases the analysis of such layouts. We then establish theoretical properties of the layout problem, including the optimization of layout size and quality (in terms of netlength), and finally devise a number of different placement algorithms that will subsequently be used as tools for the construction of transistor-level layouts with more than one transistor row.

4.1 Definitions

A *net* N represents an electrical potential in a cell, i.e. a region that must be electrically connected by some sort of wiring. Occasionally a *weight* $w(N) \in \mathbb{N}$ is associated with every net.

net

net weight $w(N)$

A *FET* or *transistor* is given as a tuple $F = (N_g, N_s, N_d, L, t, Vt)$, where

FET

- N_g, N_s, and N_d are the nets connected to gate, source, and drain,
- $L \in \mathbb{N}$ is the total length of the gate (associated with the FET's "area"), given as the number of required fin/gate intersections,
- $t \in \{n, p\}$ is the FET's type, and
- $Vt \in \mathbb{N}$ is its Vt level.

$N_g(F)$, $N_s(F)$, $N_d(F)$

$L(F)$

$t(F)$

$Vt(F)$

For a given FET F we denote these parameters by $N_g(F)$, $N_s(F)$, $N_d(F)$, $L(F)$, $t(F)$, and $Vt(F)$. Additionally, a function $L^F_f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$ is associated

$L^F_f(k)$

with every transistor F. Assuming that F is realized with k fingers, the set $L_f^F(k)$ contains the possible finger lengths. In the following we always require $|L_f^F(k)| \leq 1$ for all $k \in \mathbb{N}$, so either a given number of fingers cannot be realized or the finger length is uniquely determined by the number of fingers.

netlist

A *transistor-level netlist*, or simply *netlist*, is a pair $(\mathcal{F}, \mathcal{N})$ of a set of FETs and a set of nets such that $\mathcal{N} = \{N_*(F) \mid F \in \mathcal{F}, * \in \{g, s, d\}\}$. Because the nets can be inferred from the FETs, a netlist is usually identified with the set of FETs.

Configurations, Placements, Layouts

configuration

A *configuration* of a FET F is a tuple $(n_f(F), l_f(F), c_l(F))$ that represents the physical realization of an abstract transistor. It consists of

$n_f(F)$

- $n_f(F) \in \mathbb{N}$, the number of fingers (associated with the transistor's "width"),

$l_f(F)$

- $l_f(F) \in L_f^F(n_f(F))$, the length of a finger (the transistor's "height"), given as the number of fins it intersects, and

$c_l(F)$

- $c_l(F) \in \{N_s(F), N_d(F)\}$, specifying the net connected to the leftmost source/drain contact (equivalent to the transistor's swap status).

Because $|L_f^F(k)| \leq 1$ is generally assumed, we henceforth omit $l_f(F)$ from the configuration: If a FET can be configured with k fingers, then $l_f(F)$ is uniquely determined by this number. Ideally, $L(F) = n_f(F) \cdot l_f(F)$ should hold, but this is not enforced. A configuration of a set $\mathcal{F}$ of transistors is an assignment of configurations to every $F \in \mathcal{F}$. A configuration of a netlist $(\mathcal{F}, \mathcal{N})$ is a configuration of $\mathcal{F}$. If a configuration has already been fixed, a FET or a set of FETs is called *configured*.

$c_r(F)$

For a given configuration of a FET F we also define $c_r(F)$ as the net connected to the rightmost contact, that is

$$c_r(F) := \begin{cases} c_l(F) & \text{if } n_f(F) \text{ is even} \\ N_s(F) & \text{if } n_f(F) \text{ is odd and } c_l(F) = N_d(F) \\ N_d(F) & \text{if } n_f(F) \text{ is odd and } c_l(F) = N_s(F). \end{cases}$$

$n_f^{\min}(F), n_f^{\max}(F)$

Now let $n_f^{\min}(F) := \min\{k \mid L_f^F(k) \neq \varnothing\}$ be the minimum and $n_f^{\max}(F) := \max\{k \mid L_f^F(k) \neq \varnothing\}$ the maximum number of fingers a transistor can be configured with. Figure 4.1 illustrates the definitions.

placement

A *1-dimensional placement* of a FET F, or just *placement* if the type is apparent from the context, is a number $x(F) \in \mathbb{N}$. It represents the x-coordinate of the leftmost gate, given as a multiple of the technology-defined track width. A *1-dimensional layout* of a FET, or just *layout*, is a pair $(x(F), C(F))$ of a placement and a configuration of F. By analogy with the definition of a configuration we extend those definitions to sets of transistors and netlists.

layout

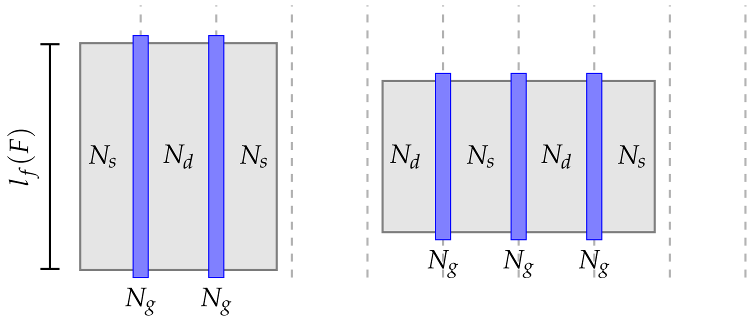


Figure 4.1: The same transistor F with configuration $(2, 3, N_s)$ placed at $x(F) = 0$ and with configuration $(3, 2, N_d)$ placed at $x(F) = 4$. In both cases $c_r(F) = N_s$ holds.

FET Distances, Legality

Not all possible layouts correspond to physical layouts that would function, let alone be manufacturable. Transistors may not overlap arbitrarily and must fulfill specific constraints. To handle such constraints, a general *distance function d* is employed. Such a function takes two configured FETs and returns the minimum distance between the two FETs if the first one is placed to the left of the second one. In particular, $d(F', C', F'', C'') = 0$ means that if F' with configuration C' is placed to the left of F'', configured according to C'', then both FETs may overlap by 1 track, i.e. diffusion sharing is possible. If the configuration is unambiguous in the context, we only write $d(F', F'')$ instead of $d(F', C', F'', C'')$. For $d(F', F'') = 1$ the two FETs' diffusion areas may abut but not overlap, i.e. 1 track between the FETs cannot be used for one of the FETs' fingers, and for $d(F', F'') = 2$ a free track between the FETs' diffusion areas must be introduced, meaning that 2 possible finger locations between the FETs cannot be used. Figure 4.2 illustrates these cases.

distance

While this type of distance function is very general, more specific examples are discussed in the upcoming sections that model actual constraints from real-life technology. In particular, we say a distance function is *simple* if its value only depends on the differences between the FET heights, the FETs' Vt levels, and their types. In other words, the three properties $|l_f(F_l) - l_f(F_r)| = |l_f(F'_l) - l_f(F'_r)|$, $(Vt(F_l), Vt(F_r)) = (Vt(F'_l), Vt(F'_r))$, and $(t(F_l), t(F_r)) = (t(F'_l), t(F'_r))$ must imply $d(F_l, F_r) = d(F'_l, F'_r)$.

simple distance

A distance function provides a notion of legality. Given a layout (x, C) of a netlist $(\mathcal{F}, \mathcal{N})$ and a distance function d, we call the layout *legal* if

legal layout

$$x(F') \geq x(F) + n_f(F) + d(F, F')$$

holds for each two distinct FETs $F, F' \in \mathcal{F}$ with $x(F) \leq x(F')$.

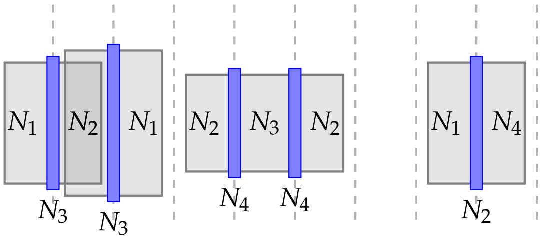


Figure 4.2: Example layout Λ with $W(\Lambda) = 7$ where the FETs are placed at their minimum distances if $d(F_1, F_2) = 0$, $d(F_2, F_3) = 1$, and $d(F_3, F_4) = 2$ (if the FETs are numbered from left to right).

Layout Width, Netlength

An important parameter of a layout is its size. In the 1-dimensional context it is solely determined by the required number of horizontal tracks, so we define for a layout $\Lambda = (x, C)$ of a FET set $\mathcal{F}$ the *layout width*

layout width $W(\Lambda)$

$$W(\Lambda) := \max_{F \in \mathcal{F}} \{x(F) + n_f(F)\}.$$

The optimal width of a netlist is given by

$$W_{\text{opt}}(\mathcal{F}) := \min\{W(\Lambda) \mid \Lambda \text{ is a legal layout of } \mathcal{F}\}.$$

$T_{sd}(F,N)$

To estimate the amount of wiring required to electrically connect the nets in a layout, two different types of netlengths are introduced. They are defined using the sets $T_{sd}(F, N)$ that contain all half-tracks, i.e. integer multiples of half the technology-defined track width, on which F must be connected by net N, precisely

$$T_{sd}(F,N) := \begin{cases} \{2x(F) + 2k \mid 0 \le k \le n_f(F),\ k \text{ even}\} & \\ \qquad\qquad \text{if } N \in \{N_s(F), N_d(F)\} \text{ and } c_l(F) = N \\ \{2x(F) + 2k \mid 0 \le k \le n_f(F),\ k \text{ odd}\} & \\ \qquad\qquad \text{if } N \in \{N_s(F), N_d(F)\} \text{ and } c_l(F) \neq N \\ \varnothing \quad \text{otherwise.} \end{cases}$$

$T_{sd}(N)$ $T_g(N)$

Then $T_{sd}(N) := \bigcup_{F \in \mathcal{F}} T_{sd}(F, N)$ is the set of all tracks on which N is connected to a source or drain contact. The set of all half-tracks on which N is connected to a gate is given by

$$T_g(N) := \{2k + 1 \mid x(F) \le k < x(F) + n_f(F) \\ \text{for some } F \in \mathcal{F} \text{ with } N_g(F) = N\}.$$

$T(N)$ gate / total netlength

We set $T(N) := T_g(N) \cup T_{sd}(N)$. Then the *gate netlength* $NL_g(\Lambda, N)$ and the *total netlength* $NL(\Lambda, N)$ of a net are defined by

$$NL_g(\lambda, N) := \max T_g(N) - \min T_g(N) \quad \text{and}$$
$$NL(\lambda, N) := w(N) \cdot (\max T(N) - \min T(N))\,.$$

If no weights are given in the context, we assume $w(N) = 1$ for every net N. Note that the gate netlength is not affected by net weights. Finally, we define the netlengths of the layout as $NL_g(\Lambda) := \sum_{N \in \mathcal{N}} NL_g(\Lambda, N)$ and $NL(\Lambda) := \sum_{N \in \mathcal{N}} NL(\Lambda, N)$. An example is given in Table 4.1.

$NL_g(\Lambda), NL(\Lambda)$

Placement Problem, Configuration Graphs

The central optimization problem associated with 1-dimensional transistor placements is the minimization of the layout size.

TRANSISTOR ROW PLACEMENT PROBLEM

Instance: A transistor-level netlist $(\mathcal{F}, \mathcal{N})$.

Task: Find a legal layout Λ of $(\mathcal{F}, \mathcal{N})$ that minimizes $W(\Lambda)$.

In order to examine theoretical properties and algorithms related to this problem, it is helpful to model a netlist as a graph. Given a netlist $(\mathcal{F}, \mathcal{N})$ together with a configuration C, we define the *configuration graph* as the graph $G^C_{\mathcal{F}} := (V^C_{\mathcal{F}}, E^C_{\mathcal{F}})$, where

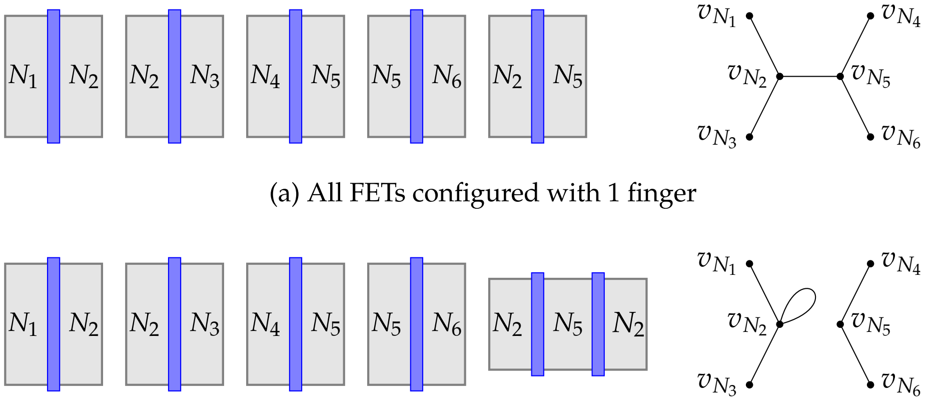
configuration graph

$$V^C_{\mathcal{F}} := \{v_N \mid N \in \mathcal{N}\} \quad \text{and}$$
$$E^C_{\mathcal{F}} := \{\{v_{c_l(F)}, v_{c_r(F)}\} \mid F \in \mathcal{F}\}.$$

The vertices model the nets and the edges model the transistors by linking the net connected to the leftmost contact to the net connected to the rightmost contact. If a FET has an odd number of fingers, the according edge connects source with drain, otherwise the edge is a loop adjacent to either the source or the drain vertex. Two examples are given in Figure 4.3. The *directed configuration graph* is the same graph with the only difference that the edges are directed from the leftmost contact to the rightmost contact.

Table 4.1: Netlengths for the layout in Figure 4.2.

Net	T_{sd}	T_g	NL	NL_g
N_1	$\{0,4,14\}$	$\emptyset$	14	0
N_2	$\{2,6,10\}$	$\{15\}$	13	0
N_3	$\{8\}$	$\{1,3\}$	7	2
N_4	$\{16\}$	$\{7,9\}$	7	2
Λ	–	–	41	4



(a) All FETs configured with 1 finger

(b) One of the FETs configured with 2 fingers

Figure 4.3: Two possible layouts of 5 FETs and the corresponding configuration graphs. The covering number $\gamma(G)$ decreases from 3 to 2, the total number of fingers increases by 1.

4.2 Optimization of the Layout Size

$d^{0/2}(F, F')$

For the theoretical exploration of the TRANSISTOR ROW PLACEMENT PROBLEM we restrict to one of the most simple distance functions that models diffusion sharing and already captures many properties from real-world technologies. The function always permits diffusion sharing if the shared contact is connected by the same net. Otherwise, a gap of 1 track between the diffusion areas is required. Formally, this distance function is given by

$$d^{0/2}(F, F') := \begin{cases} 0 & \text{if } c_r(F) = c_l(F') \\ 2 & \text{otherwise.} \end{cases}$$

Walk Covers

walk cover

covering number $\gamma(G)$

For a graph $G = (V, E)$, a *walk cover* $\mathcal{W}$ is a set of walks in G such that every $e \in E$ is contained in exactly one walk in $\mathcal{W}$. The notion is extended to directed graphs with directed walks. The *covering number* $\gamma(G)$ of a graph G is defined as the minimum cardinality of a walk cover of G. It can be determined easily in linear runtime.

Lemma 1. *Let G be an undirected graph and $G_1, \ldots, G_k$ the connected components of G that have at least one edge. If n^i_{odd} denotes the number of vertices in G_i that have an odd degree, then*

$$\gamma(G) = \sum_{i=1}^{k} \max\{1, \frac{n^i_{odd}}{2}\}.$$

Proof. This is a simple consequence from the fact that a connected graph possesses a Eulerian cycle if and only if all vertices have even degree [HW73]: Let G be connected and n_{odd} the number of odd-degree vertices in G. Then one can add $\frac{n_{\text{odd}}}{2}$ edges to obtain a Eulerian graph G' with a Eulerian cycle C. When the added edges are removed, C is split into $\frac{n_{\text{odd}}}{2}$ walks. If G is Eulerian and has at least one edge, then one walk is required to cover G even though $n_{\text{odd}} = 0$, and hence $\gamma(G) \leq \max\{1, \frac{n^i_{\text{odd}}}{2}\}$. For the other direction, observe that at least one walk must start or end in every odd-degree vertex. The lemma is obtained by applying this argument to every connected component that is not an isolated vertex. □

Lemma 2. *Let G be a directed graph and $G_1, \ldots, G_k$ the subgraphs associated with the connected components of the underlying undirected graph that have at least one edge. If we denote $\Delta_i := \sum_{v \in V(G_i)} \max\{0, \delta^+(v) - \delta^-(v)\}$, then*

$$\gamma(G) = \sum_{i=1}^{k} \max\{1, \Delta_i\}.$$

Proof. The lemma is deduced in analogy to Lemma 1, using the fact that a directed graph G whose underlying undirected graph is connected has a Eulerian cycle if and only if $\delta^+(v) = \delta^-(v)$ holds for all $v \in V(G)$. □

Walk covers were first applied to transistor-level layout problems by Uehara and vanCleemput [Uv79]. The following lemma, which is an extension of their observations, connects the covering number to optimal FET layouts.

Lemma 3. *For a set $\mathcal{F}$ of FETs let*

$$W^{0/2}_{opt}(\mathcal{F}) := \min\{W(\Lambda) \mid \Lambda \textit{ is a legal layout of } \mathcal{F} \textit{ with respect to } d^{0/2}\}$$

denote the width of an optimal layout. Then

$$W^{0/2}_{opt}(\mathcal{F}) = \min\{2\gamma(G^C_{\mathcal{F}}) - 2 + \sum_{F \in \mathcal{F}} n_f(F) \mid C = (n_f, l_f, c_l) \textit{ is a configuration of } \mathcal{F}\}.$$

Proof. The edge sequence of a walk in a configuration graph corresponds to a sequence of FETs that can be laid out in a consecutive row such that all adjacent FETs share their diffusion areas. On the other hand, two edges that do not form a walk of length 2 in $G^C_{\mathcal{F}}$ cannot share their diffusion areas by the definition of $d^{0/2}$. Consequently, the minimum number of gaps in the diffusion area equals $\gamma(G) - 1$, and each of the gaps must span at least 2 tracks. Since no two fingers can share the same track, at least $\sum_{F \in \mathcal{F}} n_f(F)$ tracks are required to hold the fingers of a given configuration. Summing up gap size and finger numbers yields the value given in the lemma. □

Figure 4.4 illustrates the correspondence between a walk cover and a layout.

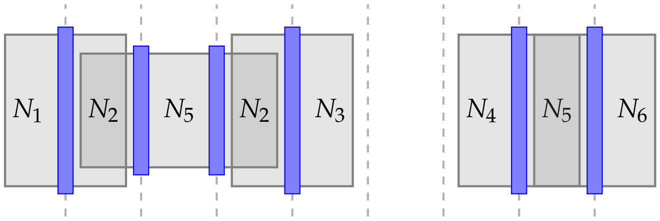


Figure 4.4: Placement with width $W^{0/2}_{\text{opt}}(\mathcal{F})$ that corresponds to the walk cover $\{(v_{N_1}, v_{N_2}, v_{N_2}, v_{N_3}), (v_{N_4}, v_{N_5}, v_{N_6})\}$ in Figure 4.3b.

Hardness of the Placement Problem

Several variants of the TRANSISTOR ROW PLACEMENT PROBLEM can therefore be solved in linear time, namely:

- If the finger numbers n_f are fixed.
- If a placement must be computed in which all $n_f(F)$ are required to be odd.
- If c_l is fixed (i.e. swapping is not allowed) and all $n_f(F)$ are required to be even. In this case directed configuration graphs are used to model the inability to swap FETs.

It is easy to check that in these cases all possible configuration graphs $G^{\mathcal{C}}_{\mathcal{F}}$ are identical, so $W^{0/2}_{\text{opt}}(\mathcal{F})$ can be determined by an application of Lemma 3 to a single configuration graph and choosing the values of n_f as small as possible. An extensive discussion of these results can be found in [Wey11].

If swapping is allowed and all FETs are required to have an even number of fingers, then the minimization of the placement width is structurally equivalent to the VERTEX COVER problem: The values of c_l must be chosen such that the fewest possible number of vertices in the configuration graph has a positive degree, i.e. the number of connected components with at least one edge is minimized. Hence this variant of the placement problem is NP-hard. A more detailed explanation is provided by Weyd [Wey11].

The complexity of the general version of the problem remains an open question: When the finger numbers are neither prescribed nor required to have the same parity. In this case FETs may be increased in size (from 1 finger to 2 fingers) in order to reduce the number of required diffusion gaps, each of which saves 2 tracks.

4.3 Optimization of the Netlength

Alongside the optimization of the width of a layout, its routability is also an important property. A rough yet common measure for this abstract quality is netlength: If less wiring is needed to connect the contacts in a layout, it seems likely that the routing problem becomes easier. This is not true in general, but a close enough approximation to use in contexts where the exact routability, i.e. the existence of a feasible routing, is too hard to evaluate.

TRANSISTOR ROW NETLENGTH MINIMIZATION PROBLEM

Instance: A transistor-level netlist $(\mathcal{F}, \mathcal{N})$.

Task: Find a legal layout Λ of $(\mathcal{F}, \mathcal{N})$ with $W(\Lambda) = W_{\text{opt}}(\mathcal{F})$ such that $NL_g(\Lambda)$ is minimized.

Again we consider the simple distance function $d^{0/2}$ as defined in Section 4.2. Weyd [Wey11] provides proofs that the problem is NP-complete if all FETs are required to have an odd number of fingers and, on the other hand, if all FETs are required to have an even number of fingers. In the following, we present a simplified version of these proofs which does not pose restrictions on the transistor configurations.

Theorem 1. *The* TRANSISTOR ROW NETLENGTH MINIMIZATION PROBLEM *is NP-hard.*

Proof. We show that the associated decision problem, the question if there exists a legal minimum-width layout Λ with $NL_g(\Lambda) \leq k$, is NP-complete. The membership in NP is obvious.
To show NP-completeness, we describe a polynomial transformation from the LINEAR ARRANGEMENT problem. For this let $G = (V, E)$ with $V = \{1, \dots, n\}$ and $E = \{e_1, \dots, e_m\}$ be an instance of this problem. To prove the theorem it suffices to construct a transistor-level netlist $\mathcal{F}_G$ such that there exists a permutation $\pi : V \to V$ with $\sum_{\{v,w\} \in E} |\pi(v) - \pi(w)| \leq k$ if and only if there exists a minimum-width layout Λ of $\mathcal{F}_G$ with $NL_g(\Lambda) \leq m^2 + (2m + 2)k$. We proceed with the definition of such an instance.
The nets in $\mathcal{F}_G$ are grouped into three sets $\mathcal{N}_V := \{N_v^i \mid v \in V, i \in \{1, \dots, 2m + 1\}\}$, $\mathcal{N}_E := \{N_e \mid e \in E\}$, and $\mathcal{N}_\varnothing$, where the first two sets are associated with vertices and edges of G, respectively. All nets in $\mathcal{N}_\varnothing$ are only connected to a single gate contact and thus do neither contribute to the netlength nor influence diffusion sharing options.
For every $v \in V$ we construct $2m$ FETs $\mathcal{F}_v := \{F_v^1, \dots, F_v^{2m}\}$ with the following contacts:

$$
\begin{aligned}
N_s(F_v^i) &= N_v^i && \text{for } 1 \leq i \leq 2m \\
N_d(F_v^i) &= N_v^{i+1} && \text{for } 1 \leq i \leq 2m \\
N_g(F_v^i) = N_g(F_v^{2m+1-i}) &= N_{e_i} && \text{if } v \in e_i \\
N_g(F_v^i),\ N_g(F_v^{2m+1-i}) &\in \mathcal{N}_\varnothing && \text{if } v \notin e_i
\end{aligned}
$$

An example is depicted in Figure 4.5. First note that F_v^i and F_w^j have no source or drain contact in common if $v \neq w$, so the configuration graph $G_{\mathcal{F}_G}^C$ has at least n connected components with edges for every possible configuration C. This implies that $\gamma(G^{C}_{\mathcal{F}_G}) \geq n$ holds for every configuration C. For a configuration with $n_f \equiv 1$ we have $\gamma(G^{C}_{\mathcal{F}_G}) = n$, because then $G^{C}_{\mathcal{F}_G}$ is the disjoint union of n paths.

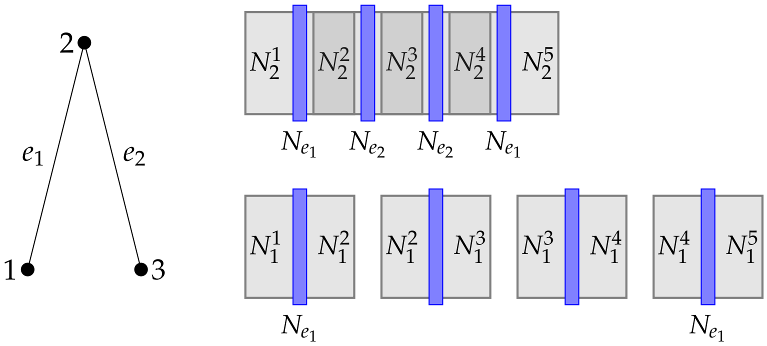


Figure 4.5: Example for the construction of Theorem 1: A graph G and the FET sets $\mathcal{F}_1$ (bottom row) and $\mathcal{F}_2$ (top row, sharing diffusion).

We can conclude by Lemma 3 that every width-minimal configuration must satisfy $n_f \equiv 1$, as otherwise $\sum_{F\in\mathcal{F}_G} n_f(F)$, and consequently $W^{0/2}_{\text{opt}}(\mathcal{F}_G)$, is strictly larger. So from now on we may only consider configurations in which every FET has a single finger.

A group $\mathcal{F}_v$ can only be placed in the order $F_v^1, \ldots, F_v^{2m}$ or exactly vice versa in width-optimal layouts, as every pair of neighboring FETs must overlap. However, flipping such a group horizontally does not change the gate netlength, because the gates in a group are connected symmetrically around the center of the group. Thus we may assume w.l.o.g. that the instance is placed with $n(F) = 1$ and $c_l(F) = N_s(F)$ for all $F \in \mathcal{F}_G$. We call this unique configuration C^*.

It follows that for every permutation $\pi : V \to V$ there exists exactly one placement x_π such that

- $\Lambda_\pi := (x_\pi, C^*)$ is a legal layout
- that is width-optimal, i.e. $W(\Lambda_\pi) = W^{0/2}_{\text{opt}}(\mathcal{F}_G)$,
- and for which the FET groups are ordered from left to right according to π (i.e., $\pi(v) < \pi(w) \to x_\pi(F_v^i) < x_\pi(F_w^j)$ for all $1 \leq i, j \leq 2m$).

For an edge $e_i = \{v, w\} \in E$ the length of N_{e_i}, as illustrated by Figure 4.6, is comprised of three components, namely

- the part of the net between $\mathcal{F}_v$ and $\mathcal{F}_w$, which is $3 + (|\pi(v) - \pi(w)| - 1)(2m + 2)$,
- the part of the net inside $\mathcal{F}_v$, which amounts to $2m - i$, and
- the part of the net inside $\mathcal{F}_w$, which is also $2m - i$.

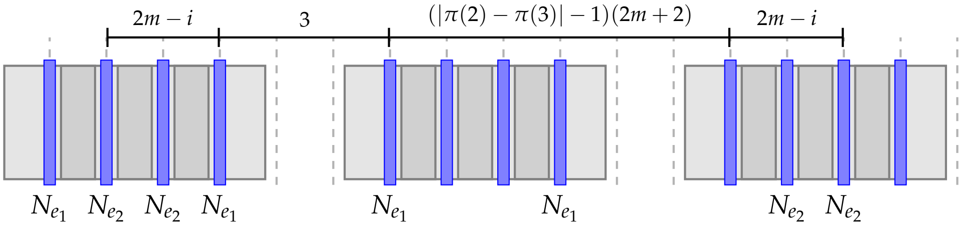


Figure 4.6: Layout $\Lambda_{(2,1,3)}$ and the components of $NL_g(N_{e_2})$ for the graph in Figure 4.5.

The gate netlength of such a layout thus amounts to

$$\begin{aligned}
NL_g(\Lambda_\pi) &= \sum_{e_i=\{v,w\}\in E} 3 + (|\pi(v)-\pi(w)|-1)(2m+2) + 2(2m-i) \\
&= 3m + 4m^2 + \sum_{e_i=\{v,w\}\in E} (|\pi(v)-\pi(w)|-1)(2m+2) - 2i \\
&= m + 2m^2 - \left(2\sum_{i=1}^{m} i\right) + \sum_{\{v,w\}\in E} (|\pi(v)-\pi(w)|)(2m+2) \\
&= m + 2m^2 - m(m+1) + (2m+2)\cdot \sum_{\{v,w\}\in E} |\pi(v)-\pi(w)| \\
&= m^2 + (2m+2)\cdot \sum_{\{v,w\}\in E} |\pi(v)-\pi(w)|.
\end{aligned}$$

We can conclude that there exists a layout Λ of $\mathcal{F}_G$ with $NL_g(\Lambda) \le m^2 + (2m+2)k$ if and only if there exists a permutation $\pi : V \to V$ such that $\sum_{\{v,w\}\in E} |\pi(v)-\pi(w)| \le k$, which is the required property of the transformation. The polynomiality of the transformation follows directly from its construction. □

Note that the same hardness result is obtained when total netlength rather than gate netlength is considered: The latter is the special case for which all source and drain contacts are connected to the same net.

4.4 Algorithms

In practice, several degrees of effort can be made to search for good placements, ranging from fast heuristics to the enumeration of all optimal solutions. Because several variants from this range can be applied in the layout of full cells with two transistor rows, as described in Chapter 5, we provide not just one but several algorithms that generate 1-dimensional placements. In the following, a *FET class* $\mathcal{C}$ denotes a set of FETs such that all $F \in \mathcal{C}$ have the same Vt level and the same type. The distinction into classes has no theoretical significance, but in practice FETs from different classes tend to

FET class

have the strictest requirements on their distance, so from an implementation point of view it is advantageous to make that difference.

The algorithms provided in this section are implemented as part of the BONNCELL toolchain, and therefore we do not make any assumptions on the distance function d. However, we do introduce names for some properties of the function:

$d^{+}_{\text{min}}, d^{\text{cl}}_{\text{min}}, d_{\text{max}}$

d^{+}_{min} : minimum value of d that is larger than 0

$d^{\text{cl}}_{\text{min}}$: minimum value of d for FETs in different classes

d_{max} : maximum value of d taken over all FETs and all configurations

These values can be precomputed in every call of the algorithms. For example, if an instance is processed in which all FETs have the same type, which is a common setup, then d_{max} can be set to the maximum value of d taken over all FET pairs that occur in the instance, even though FETs of different types would have to be spaced even further apart.

$W^{n_f}_{\text{opt}}(\mathcal{F})$

For fixed finger numbers $n_f : \mathcal{F} \to \mathbb{N}$ we further define

$$W^{n_f}_{\text{opt}}(\mathcal{F}) := \min\{W(\Lambda) \mid \Lambda \text{ is a legal layout of } \mathcal{F} \text{ with the given } n_f\}$$

as the minimal size of a legal layout with the given finger numbers.

The following algorithms employ a dynamic programming scheme that recursively constructs a row by extending a placed subset of the transistors—initially the empty set—to the right. In every step, an unplaced FET is chosen and attached to the right of the already placed partial row, preferably but not necessarily at an x-coordinate that is as small as possible.

4.4.1 Lower Bound

For this general scheme to run efficiently it is important to compute lower bounds for the width of the finished placement, given that a subset of the FETs is placed in a specific way. Algorithm 1 provides a general framework to compute such a lower bound.

Lemma 4. *If Algorithm 1 returns W_{LB}, then $W_{LB} \leq W(\Lambda)$ for every legal layout Λ of $\mathcal{F}_l \cup \mathcal{F}_u$ that is configured with the prescribed finger numbers n_f and that extends the layout of $\mathcal{F}_l$. It can be implemented with runtime $\mathcal{O}(|\mathcal{F}_l \cup \mathcal{F}_u|)$.*

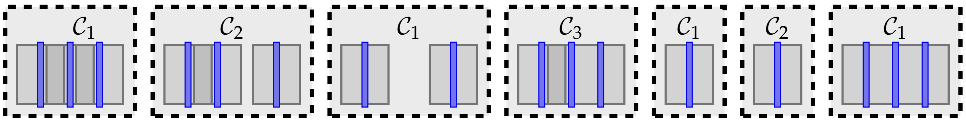


Figure 4.7: Illustration of some definitions from the proof: The layout splits into 3 FET classes $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. Two classes are placed in more than one consecutive interval, so $l_1 = 3$, $l_2 = 2$, and $l_1 = 1$.

Algorithm 1: `LowerBound`

```
  Data: FETs $\mathcal{F}_l$ with a legal layout $\Lambda_l$, unplaced FETs $\mathcal{F}_u$, and $n_f(F)$ for
        all $F \in \mathcal{F}_u$
  Output: An integer $W_{\mathsf{LB}}$
1 $F_{\mathsf{prev}} \leftarrow$ Rightmost FET in $\mathcal{F}_l$
2 num_classes $\leftarrow$ Different FET classes in $\mathcal{F}_u \cup \{F_{\mathsf{prev}}\}$
3 num_class_gaps $\leftarrow 0$
4 forall the FET class $\mathcal{C}$ in $\mathcal{F}_u \cup \{F_{\mathsf{prev}}\}$ do
5     $\mathcal{C}_{\mathrm{odd}} \leftarrow$ FETs in $\mathcal{C}$ with an odd number of fingers
6     $G \leftarrow$ configuration graph of $(\mathcal{F}_u \cup \{F_{\mathsf{prev}}\}) \cap \mathcal{C}_{\mathrm{odd}}$
      num_class_gaps $\leftarrow$ num_class_gaps $+ \max\{0, \gamma(G) - 1\}$
7 return $W(\Lambda_l) + \sum_{F \in \mathcal{F}_u} n_f(F)$
8          $+($num_classes $- 1) \cdot d^{cl}_{min} +$ num_class_gaps $\cdot\, d^{+}_{min}$
```

Proof. Let Λ be a legal layout of $\mathcal{F}_l \cup \mathcal{F}_u$ that extends the given layout of $\mathcal{F}_l$. Then $W(\Lambda)$ is comprised of the number of tracks occupied by the given layout of $\mathcal{F}_l$, which is $W(\Lambda_l)$, the fingers used by FETs in $\mathcal{F}_u$, which can be computed from the input as $\sum_{F \in \mathcal{F}_u} n_f(F)$, and possible gaps between the FETs that have yet to be placed.

Now assume that $\mathcal{F}_u \cup \{F_{\text{prev}}\}$ partitions into the FET classes $\mathcal{C}_1, \dots, \mathcal{C}_k$, and that $\mathcal{C}_i$ is split into l_i groups of FETs, each of which is placed consecutively (c.f. Figure 4.7). Then in total $(\sum_{i=1}^{k} l_i) - 1$ gaps between two FETs from different classes are required. Additionally, gaps within a class may be necessary if the number of walks required to cover the configuration graph exceeds l_i. In particular, if G_i denotes the configuration graph of the FETs in $\mathcal{C}_i$ that have an odd number of fingers, then at least $\max\{0, \gamma(G_i) - l_i\}$ gaps between FETs in the class $\mathcal{C}_i$ must be introduced. Summing up, the total size of those gaps is

$$\left(\left(\sum_{i=1}^{k} l_i\right) - 1\right) \cdot d^{\mathrm{cl}}_{\mathrm{min}} + \left(\sum_{i=1}^{k} \max\{0, \gamma(G_i) - l_i\}\right) \cdot d^{+}_{\mathrm{min}}. \tag{1}$$

Because of $d^{\mathrm{cl}}_{\mathrm{min}} \geq d^{+}_{\mathrm{min}}$, which holds by definition, this term is minimized when setting $l_i := 1$ for $1 \leq i \leq k$. In this case (1) equals the value computed in line 8 of the algorithm.

The configuration graphs in line 6 can be computed even though no swap status is yet assigned to the FETs in $\mathcal{F}_u$. This is possible because only FETs with an odd number of fingers are considered, and swapping such transistors does not change the graph. It remains to show that it is allowed to omit FETs with an even number here. To see this, observe that $\gamma(G)$ never decreases when adding loops to G (the parity of all vertex degrees stays the same and the number of connected components with edges does not decrease). Thus (1) is a lower bound for the total gap size even though FETs with an even number of fingers are not considered.

Algorithm 2: UpperBoundFixedFingers

```
Data: A netlist $\mathcal{F}$ and $n_f(F)$ for all $F \in \mathcal{F}$
Output: A layout $\Lambda_{\text{best}}$
$W_{\text{best}} \leftarrow \infty$
forall the $F \in \mathcal{F}$ do
    for $c_l(F) \in \{N_s(F), N_d(F)\}$ do
        $x(F) \leftarrow 0$
        ContinuePlacementUB($\{F\}, \mathcal{F} \setminus \{F\}$)
return $\Lambda_{best}$
```

For the runtime note that the FETs can be sorted by their class in linear time using bucket sort. Moreover, the covering number of a class can be computed in a time linear in the size of the class due to Lemma 1. □

4.4.2 Upper Bound

In some cases, one is interested in good solutions that provide an upper bound for $W^{n_f}_{\text{opt}}(\mathcal{F})$ and can be computed efficiently. In particular, we propose Algorithm 2 to compute such an upper bound.

Lemma 5. *If Algorithm 2 returns Λ_{best}, then Λ_{best} is a legal layout with $W(\Lambda_{best}) \geq W^{n_f}_{opt}(\mathcal{F})$.*

Proof. It suffices to show that Λ_{best} is a legal layout. In fact, we claim that in every execution of line 4 in Algorithm 3 a legal layout is stored, and that the line is executed at least once. For the legality observe that the recursive calls guarantee that $\mathcal{F} = \mathcal{F}_l \cup \mathcal{F}_u$ always holds, and that x and c_l are set in a way that $\mathcal{F}_l$ is laid out legally every time Algorithm 3 is entered.
To see that Λ_{best} is updated at least once, which is necessary for the resulting layout to be legal, consider the point where the execution of Algorithm 3 ends. This can only happen when $\mathcal{F}_u = \emptyset$ (in this case Λ_{best} was updated) or after a recursive call of Algorithm 3 with a strictly smaller $\mathcal{F}_u$. The return statement in line 2 cannot be reached as long as W_{best} has not been set to a finite value. In other words, backtracking does not happen before line 4 was reached for the first time. □

Note that even though no optimal placement is found by this upper bound computation, it may require an exponential runtime.

Lemma 6. *There exist instances with n transistors for which Algorithm 2 requires $\Omega(2^n)$ steps.*

Proof. Let $\mathcal{F} = \{F_1, \dots, F_n\}$ and $\mathcal{N} = \{N_0, \dots, N_{n+2}\}$. We set $n_f(F) = 2$ for all $F \in \mathcal{F}$. The source and drain contacts are $N_s(F_i) = N_0$ and $N_d(F_i) = N_i$

Algorithm 3: `ContinuePlacementUB`

```
    Data: FETs $\mathcal{F}_l$ with a legal layout $\Lambda_l$, unplaced FETs $\mathcal{F}_u$, and $n_f(F)$ for
          all $F \in \mathcal{F}_u$
 1  if LowerBound($\mathcal{F}_l, \mathcal{F}_u$) $\geq W_{\text{best}}$ then
 2      return
 3  if $\mathcal{F}_u = \varnothing$ then
 4      $\Lambda_{\text{best}} \leftarrow \Lambda_l$, $W_{\text{best}} \leftarrow W(\Lambda_l)$
 5  else
 6      $F_{\text{prev}} \leftarrow$ Rightmost FET in $\mathcal{F}_l$
 7      forall the $F \in \mathcal{F}_u$ do
 8          if F can legally overlap the right contact of $F_{\text{prev}}$ then
 9              $x(F) \leftarrow x(F_{\text{prev}}) + n_f(F_{\text{prev}})$
10              Set $c_l(F)$ such that $\mathcal{F}_l \cup \{F\}$ is placed legally
11              ContinuePlacementUB($\mathcal{F}_l \cup \{F\}, \mathcal{F}_u \setminus \{F\}$)
12      if $F_{\text{prev}}$ can be swapped without losing legality then
13          Swap $F_{\text{prev}}$ and repeat steps 7–11
14      if line 11 was not yet reached then
15          Let $F \in \mathcal{F}_u$ such that $F$ can be legally placed with the smallest
            possible x-coordinate and set $x(F)$ accordingly
16          Set $c_l(F)$ such that $\mathcal{F}_l \cup \{F\}$ is placed legally
17          ContinuePlacementUB($\mathcal{F}_l \cup \{F\}, \mathcal{F}_u \setminus \{F\}$)
```

for $1 \leq i < n$ and finally $N_s(F_n) = N_{n+1}$ and $N_d(F_n) = N_{n+2}$. When no FET is swapped, the configuration graph consists of $n-1$ loops attached to v_{N_0} and one loop attached to $v_{N_{n+2}}$.

We can assume that the loop in line 7 iterates over the transistors from F_1 to F_n in the order of the indices. Then the algorithm will recurse in line 11 until F_n is reached, subsequently placing F_n with a gap to the mutually overlapping chain formed by the first $n-1$ FETs. Afterwards, W_{best} is set to $2n+2$.

As long as $F_n \in \mathcal{F}_u$, the result of the lower bound computation in line 1 is always $2n$, because FETs with an even number of fingers cannot be used by Algorithm 1 for the prediction of gaps. Consequently, the bounding in line 2 is only reached when $\mathcal{F}_u = \varnothing$.

This means that all permutations of the $n-1$ first unswapped FETs are generated by the algorithm, which only backtracks when the last remaining FET is placed. This implies a runtime of $\Omega((n-1)!)$, which exceeds $\Omega(2^n)$. □

Although upper bounds could also be determined in polynomial or even linear time, the following reasons support the method proposed in this section:

- The approach does not make any assumptions on the distance function. For specific examples of distance functions, for example $d^{0/2}$, even the

Table 4.2: Evaluation of upper bounds computed for 191 FET rows.

Number of FET rows for which the	
... upper bound is already optimal	134
... upper bound is 1 track larger than optimum	26
... upper bound is 2 tracks larger than optimum	27
... upper bound is $>$ 2 tracks larger than optimum	4
Sum of runtimes for	
... all 191 experiments (sec.)	0.159
... the 187 fastest experiments (sec.)	0.029

size of an optimal placement could be computed efficiently. However, an implementation for a general distance function is able to flexibly react to rule changes as the technology evolves.

- Experiments show that the performance of this method is good and fast enough: In many cases its results are already optimal and they rarely exceed the optimum by more than 2 tracks. The runtime of the method is mostly in the region of milliseconds, including all initialization steps. Table 4.2 summarizes the results on 191 FET rows extracted from the CLC test bed (cf. Section 7.1).
- The bounding that is performed in line 2 of Algorithm 3 can be extended to cover more complicated design rules coming from the technology. If a new rule is introduced that excludes specific configurations, the bounding function can be amended by a check of such a forbidden situation. If it is detected, `LowerBound` returns ∞, therefore preventing a further branching. All layouts that are saved to Λ_{best} in Algorithm 3 are guaranteed to be legal by that extended definition. Considering the implementation, this is even more convenient because later algorithms will employ the same bounding function and consequently satisfy the same constraints. With this technique, we gain a significant amount of flexibility that outweighs the slow runtime compared to a faster, more specialized upper bound routine.

In a setting like Algorithm 3, where a lower bound computation is performed in every node of a branch and bound tree, this computation can even be implemented to run in constant time.

Lemma 7. *Algorithm 1 can be implemented with runtime $\mathcal{O}(1)$ within the setting of Algorithm 3.*

Proof. $W(\mathcal{F}_l)$ and $\sum_{F\in\mathcal{F}_u} n_f(F)$ can easily be updated with a single addition respective subtraction before every call of Algorithm 3. Using an array that maps every FET class to the number of FETs in $\mathcal{F}_u \cup \{F_{\text{prev}}\}$ that belong to this class, it is also easy to update *num_classes* in constant runtime. Finally,

Algorithm 4: `ContinuePlacement`

```
  Data: FETs $\mathcal{F}_l$ with a legal layout $\Lambda_l$, unplaced FETs $\mathcal{F}_u$, and $n_f(F)$ for
        all $F \in \mathcal{F}_u$
1 if LowerBound($\mathcal{F}_l, \mathcal{F}_u$) $\geq W_{\mathsf{best}}$ then
2     return
3 if $\mathcal{F}_u = \varnothing$ then
4     $\Lambda_{\mathsf{best}} \leftarrow \Lambda_l$, $W_{\mathsf{best}} \leftarrow W(\Lambda_l)$
5 else
6     forall the $F \in \mathcal{F}_u$ do
7         for $c_l(F) \in \{N_s(F), N_d(F)\}$ do
8             Set $x(F)$ to the smallest legal value
9             ContinuePlacement($\mathcal{F}_l \cup \{F\}, \mathcal{F}_u \setminus \{F\}$)
```

the classes' configuration graphs and the numbers of odd vertices in those graphs also change in a constant number of places in every step. Therefore, all expressions in the return statement can be updated in constant time with every recursive call and need not be recomputed every time. □

4.4.3 Optimal Solutions

Let `OptPlacementFixedFingers` be the algorithm that results from `UpperBoundFixedFingers` by replacing all calls of Algorithm 3 by calls of Algorithm 4. In other words, the branching in the upper bound computation is replaced by another branching method. This variant now computes exact solutions rather than solutions that serve as upper bounds.

Lemma 8. *If `OptPlacementFixedFingers` returns Λ_{best}, then Λ_{best} is a legal layout with $W(\Lambda_{best}) = W_{opt}^{n_f}(\mathcal{F})$.*

Proof. Without lines 1–2 the algorithm would perform a complete enumeration of all compressed layouts, where compressed means that the distance between two neighboring FETs F and F' is exactly $d(F, F')$.
Lines 1–2 prevent a partial layout to be completed if and only if all layouts that extend the partial layout of $\mathcal{F}_l$ are at least as large as the best known complete layout Λ_{best}. Thus in the last execution of line 4 a layout of width $W_{\mathrm{opt}}^{n_f}(\mathcal{F})$ is assigned to Λ_{best}. □

4.4.4 Variants

The previous algorithms form a flexible framework that can easily be extended. The following variants will be useful tools in the generation of cell layouts that contain more than a single transistor row.

Optimal Canonical Solutions

When looking for a single minimum-width solution with no particular properties, it is not necessary to use `OptPlacementFixedFingers` as presented above: Assume that the algorithm finds a smallest layout in which $k-1$ gaps of size $d_{\max}$ occur, thereby forming k groups of FETs. Then there are a total of $k! \cdot 2^k$ layouts having the same width that can be obtained from this layout by reordering and swapping the groups. By the definition of $d_{\max}$ this is possible without losing the legality.

We can therefore restrict the search space so that only one "canonical" element from this set of $k! \cdot 2^k$ legal layouts can actually be reached by the algorithm. This motivates the following definition. Let Λ be a legal layout of $\mathcal{F}$ and let $\mathcal{F}_1, \ldots, \mathcal{F}_k$ be the FET groups, numbered from left to right, such that all gaps between these groups have size at least $d_{\max}$ and all gaps within the groups are strictly smaller. Furthermore, we denote by F_i^l the leftmost and by F_i^r the rightmost FET in $\mathcal{F}_i$. Then we call Λ a *canonical layout* if it has the following properties:

canonical

- If $|\mathcal{F}_i| > 1$, then the index of F_i^l in the array of FETs is smaller than the index of F_i^r.
- If $|\mathcal{F}_i| = 1$, then $c_l(F_i^l) = N_s(F_i^l)$.
- If $i < j$, then $W(\mathcal{F}_i) \leq W(\mathcal{F}_j)$.
- If $i < j$ and $W(\mathcal{F}_i) = W(\mathcal{F}_j)$, then the index of F_i^l in the array of FETs is smaller than the index of F_j^l.

While the first two properties prescribe swap statuses of the FETs, the remaining two properties prescribe the order of the groups.

It is now possible to modify the placement method to only look for optimal canonical layouts instead of any optimal layout. This can be achieved with a modification of the `LowerBound` function: It additionally checks if $\mathcal{F}_l$ can be extended to a canonical layout and if this is not the case, the function returns ∞, thereby enforcing a subsequent bounding. The search space is reduced drastically with this improvement and the search of a single optimal solution speeds up significantly in practice.

All Optimal Solutions

Algorithm 4 can easily be modified to enumerate *all* legal layouts of width $W_{\text{opt}}^{n_f}(\mathcal{F})$ instead of finding just a single smallest layout. This can be achieved by replacing "$\geq$" in line 1 by "$>$". Now as soon as the first optimal solution is found, all other solutions of the same width are also found because bounding only takes place if the lower bound is strictly larger than the optimal solutions.

However, when the first optimal layout is found, the algorithm does not detect this optimality. Only when the algorithm finishes the value of $W_{\text{opt}}^{n_f}(\mathcal{F})$ is

Algorithm 5: `ContinuePlacement`

```
Data: FETs $\mathcal{F}_l$ with a legal layout $\Lambda_l$, unplaced FETs $\mathcal{F}_u$, $n_f(F)$ for all
      $F \in \mathcal{F}_u$, and a number $W_{\mathsf{max}}$
if LowerBound($\mathcal{F}_l, \mathcal{F}_u$) $> W_{\mathsf{max}}$ then
    return
if $\mathcal{F}_u = \varnothing$ then
    $\Lambda_{\mathsf{best}} \leftarrow \Lambda_l$, $W_{\mathsf{best}} \leftarrow W(\Lambda_l)$
else
    forall the $F \in \mathcal{F}_u$ do
        for $c_l(F) \in \{N_s(F), N_d(F)\}$ do
            Set $x(F)$ to the smallest legal value
            while LowerBound($\mathcal{F}_l \cup \{F\}, \mathcal{F}_u \setminus \{F\}$) $\leq W_{\mathsf{max}}$ do
                ContinuePlacement($\mathcal{F}_l \cup \{F\}, \mathcal{F}_u \setminus \{F\}$)
                Increase $x(F)$
```

known. To avoid this problem, one can first look for a single optimal canonical layout of $\mathcal{F}$ to determine $W_{\text{opt}}^{n_f}(\mathcal{F})$. Then, in a second step, this knowledge is provided to the enumeration of all optimal solutions so that as soon as the first optimal solution is found the algorithm can act accordingly.

All Solutions with a Given Width

There are also cases in which not just smallest layouts need to be enumerated but all legal layouts whose width does not exceed a prescribed value, say W_{max}. This behavior can be achieved when the branching function is replaced by Algorithm 5. There are two differences compared to the branching function used to enumerate all optimal layouts:

- The lower bound is compared to W_{max} instead of W_{best}.
- $x(F)$ is not set to the smallest possible value. Instead, it is initialized with the smallest possible value and subsequently increased until the lower bound of the resulting placement exceeds W_{max}.

Algorithm 2 must be modified similarly to increase the value of $x(F)$ instead of just setting it to 0. The correctness follows from the previous discussions.

4.4.5 Enumerating Finger Numbers

Until now, the number of fingers for all FETs was assumed to be part of the input. Algorithm 6 demonstrates how BonnCell computes a globally optimal solution that does not assume fixed finger numbers.

Algorithm 6: `OptPlacement`

```
    Data: A netlist $\mathcal{F}$
    Output: A layout $\Lambda_{\text{best}}$
 1  $W_{\text{best}} \leftarrow \infty$, $\mathsf{k} \leftarrow 0$
 2  max_additional_fingers $\leftarrow \infty$
 3  while $\mathsf{k} \leq$ max_additional_fingers do
 4      forall the $n_f : \mathcal{F} \to \mathbb{N}$ with $\sum_{F \in \mathcal{F}} n_f(F) = k + \sum_{F \in \mathcal{F}} n_f^{min}(F)$ do
 5          if $L_f^F(n_f(F)) \neq \varnothing$ for all $F \in \mathcal{F}$ then
 6              $\Lambda \leftarrow$ OptPlacementFixedFingers($\mathcal{F}, n_f$)
 7              if $W(\Lambda) < W_{\text{best}}$ then
 8                  $\Lambda_{\text{best}} \leftarrow \Lambda$, $W_{\text{best}} \leftarrow W(\Lambda)$
 9      $\mathsf{k} \leftarrow \mathsf{k} + 1$
10      max_additional_fingers $\leftarrow W_{\text{best}} - \sum_{F \in \mathcal{F}} n_f^{\text{min}}(F)$
11  return $\Lambda_{best}$
```

Lemma 9. *If Algorithm 6 returns Λ_{best}, then Λ_{best} is a legal minimum-width layout, i.e. $W(\Lambda_{best}) = W_{opt}(\mathcal{F})$.*

Proof. If line 10 was omitted, the algorithm would enumerate all functions n_f that correspond to possible configurations of $\mathcal{F}$. By Lemma 8 the function `OptPlacementFixedFingers` computes smallest layouts for each of these functions, so eventually a global optimum will be found.
It remains to show that line 10 does not prevent an optimum placement to be found. So let Λ be some legal layout of $\mathcal{F}$ and $k > W(\Lambda) - \sum_{F \in \mathcal{F}} n_f^{\text{min}}(F)$. Then let $n_f : \mathcal{F} \to \mathbb{N}$ be finger numbers with $\sum_{F \in \mathcal{F}} n_f(F) = k + \sum_{F \in \mathcal{F}} n_f^{\text{min}}(F)$ as generated in line 4 of the algorithm. It follows that

$$\sum_{F \in \mathcal{F}} n_f(F) = k + \sum_{F \in \mathcal{F}} n_f^{\text{min}}(F) > W(\Lambda) \geq W_{\text{opt}}(\mathcal{F}).$$

Thus every layout with the finger numbers n_f would be strictly larger than an optimal layout. Consequently, bounding the value of k from above in line 10 does not change the set of optimum solutions found by Algorithm 6. □

Of course, the 1-dimensional placement variants mentioned above can be applied in this framework by changing line 6 accordingly. It is therefore possible to enumerate all optimal layouts, all layouts of a given size, or just to find a single canonical layout of minimum width.

Chapter 5

Cell Placement

> Das Leben kommt auf alle Fälle
>   aus einer Zelle,
> doch manchmal endets auch –
>   bei Strolchen –
>   in einer solchen.
>
> Heinz Erhardt, *Die Zelle*

Having established a toolset that can be employed to arrange 1-dimensional rows of transistors in the previous chapter, we can now turn towards the more complex task of complete cell layouts. After the definition of some terms that are unique to the 2-dimensional nature of the problem, we propose a layout flow for the most basic format of a modern CMOS cell, the arrangement of two 1-dimensional FET rows between a GND and a V_{DD} strip (cf. Section 2.3.2). Thereafter, an extended flow for the generation of multi-row layouts is presented. The chapter concludes with a discussion of several enhancements available in BONNCELL as well as an overview of the tool's parameters and how they influence the behavior of the algorithms.

5.1 Definitions

In the layout of transistor-level cells, the vertical positions of FETs are crucial, so we need to extend several definitions from Chapter 4 to incorporate the second dimension.

5.1.1 Placements and Layouts

We define a *2-dimensional placement* of a FET F as a pair $(x(F), y(F)) \in \mathbb{N}^2$ of a 1-dimensional placement and a number that encodes the vertical location of the FET. Assuming that the fins are numbered with increasing integers from bottom to top, $y(F)$ is the index of the bottommost fin intersected by

2D placement, $y(F)$

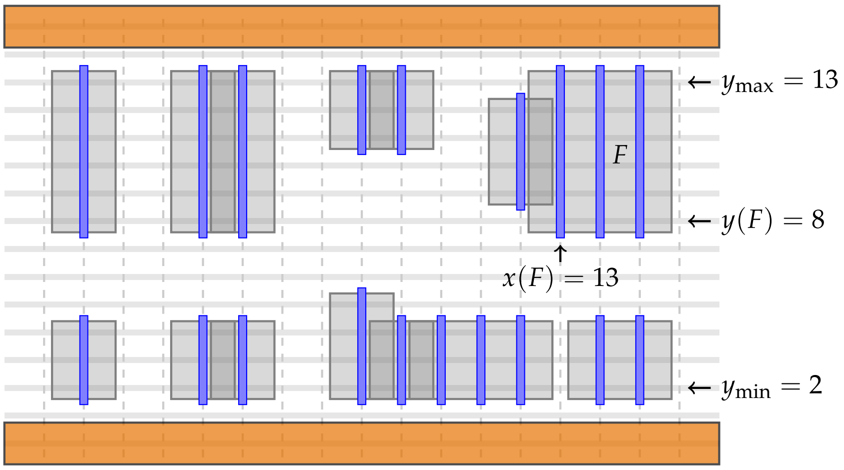


Figure 5.1: Example for a 2-dimensional standard cell layout using two FET rows within a single circuit row.

2D layout

the fingers of F. A *2-dimensional layout* $(x(F), y(F), C(F))$ is a 2-dimensional placement together with a configuration. Again we extend these notions to sets of FETs and netlists and use the short forms *placement* and *layout* if their type can be deduced from the context.

$y_{\min}, y_{\max}$

Usually not every fin can be occupied by a transistor, as the power structure near the cell borders prevents nearby fins to be used otherwise. We have thus given numbers $y_{\min}$ and $y_{\max}$ that denote the indices of the bottommost and topmost usable fins. Formally, we require $y_{\min} \leq y(F) \leq y_{\max} - l_f(F) + 1$ for every FET F. Figure 5.1 illustrates these definitions.

width, netlength

Layout widths and netlengths are defined exactly as in the 1-dimensional context, the only difference being that, in the case of vertically neighboring transistors, multiple nets can be connected to a single x-coordinate.

layout extension

In many cases a layout of a subset of $\mathcal{F}$ is given and one asks if a complete layout exists in which the subset is arranged accordingly. Given a (1-dimensional or 2-dimensional) layout Λ' of some transistors $\mathcal{F}' \subseteq \mathcal{F}$, we say that Λ' can be *extended* to a full layout if a legal 2-dimensional layout Λ of $\mathcal{F}$ exists such that Λ' and Λ are equal on the domain of Λ'.

legality oracle

Observe that we also need to augment the notion of legality. However, the criteria for a legal layout of FETs that can be arranged freely in the plane is highly technology-dependent. In addition, the rules are very complex for any recent technology. We will therefore content with the ability to *check* whether a given layout is legal. From now on, we assume that an oracle is provided that decides for a layout of $\mathcal{F}' \subseteq \mathcal{F}$ if it can be extended to a legal layout of $\mathcal{F}$. Applied to a layout of $\mathcal{F}$ the oracle thus decides if it is legal or not. The precise implementation of the oracle within BonnCell is not subject of this work due to its dependency on the specific technology node for which layouts are to be generated.

5.1.2 Target Function

Until now the only measures for the quality of 1-dimensional cell layouts have been their widths and netlengths. But these properties do not serve as good indicators for the routability of a placement, which would ideally be defined as the netlength of a shortest feasible routing (and would further take into account timing and electromigration issues, which is even more complicated). To compute good layouts, an intermediate measure is required—one that can be evaluated efficiently and is at the same time a better indication of the routability than just the netlength.

Such a compromise is meant to be provided by the following definition: For a layout $\Lambda = (x, y, (n_f, l_f, c_l))$ of $\mathcal{F}$ we denote by

$$Q(\Lambda) := \left(W(\Lambda),\ NL_g(\Lambda),\ NL(\Lambda),\ \sum_{F \in \mathcal{F}} n_f(F) l_f(F) \right)$$

the *quality* of the layout. A layout Λ is preferred over another layout Λ', which we denote by $\Lambda < \Lambda'$, if its quality is lexicographically smaller.

layout quality

The primary criterion is the layout width, which reflects the desire to create small layouts in order to have low area requirements and short wiring, thereby targeting a low power consumption and good timing properties. However, it should be noted that the probability that no feasible routing exists for a placement increases when it is generated as compact as possible. We will therefore introduce a relaxation in Section 5.4 that allows the algorithm to use a larger-than-necessary area in favor of routability.

The other criteria directly target the estimated amount of metal in the final result. The most important aspect after layout width is considered to be the gate netlength—a consequence of the technology's layer stack. Connecting opposing gates with a straight vertical wire on the PC layer uses up much less routing space than every other possible construction, so maximizing the number of such connections has very favorable consequences for the routability. The gate netlength turns out to be a close enough approximation to this preference in practice and is thus the second most important optimization goal after the layout width.

The less important criteria, the total netlength and the number of fin/gate intersections, model the total amount of wiring and the free space that is available for its routing. These numbers serve as tiebreakers for layouts that have, among all minimum-width layouts, the smallest possible gate netlength.

This lexicographically sorted multi-criteria measure of the layout quality has, although devised independently, a large resemblance of the measure employed 25 years ago by Bar-Yehuda et al. [BYFPW89]. No connection between it and the actual routability can be proven formally, yet we stress that in its practical application within the BONNCELL workflow it appears to be a good tradeoff between speed and accuracy.

5.2 Cells with Two FET Rows

We proceed with the discussion of full standard cell layouts that follow the cell model introduced in Section 2.3.2. Extensions like multi-row cells and ways to modify the tool's behavior will be discussed thereafter.

5.2.1 Algorithm

Algorithm 7 gives a high-level view on the layout flow used within BONNCELL to generate placements for cells with two transistor rows.
In the very first step, the FETs are partitioned into the groups $\mathcal{F}_b$—the FETs that are placed near the GND power rail at the bottom border of the cell—and $\mathcal{F}_t$—the FETs constituting the top transistor row near the V_{DD} power rail. By default, those groups are chosen as $\mathcal{F}_b := \{F \in \mathcal{F} \mid t(F) = \mathrm{n}\}$ and $\mathcal{F}_t := \{F \in \mathcal{F} \mid t(F) = \mathrm{p}\}$. In real-world instances, most of the cells have a comparable amount of n- and p-FETs, and n-FETs tend to be connected to GND while p-FETs tend to be connected to V_{DD}, which makes this a reasonable strategy. Section 5.2.4 describes how this default behavior can be varied in BONNCELL.

Phase 0

The zeroth phase of the layout flow is designed to yield legal 2-dimensional layouts on almost all instances within a short amount of time. Although the result may be of low quality, it provides a fallback solution in the case that the subsequent phases do not find legal layouts within the runtime limit. For cells that are hard to lay out in this sense, BONNCELL thus outputs a legal result rather than nothing at all.
In line 2 the finger lengths are bounded from above, in addition to a possible user-defined or technology-defined restriction, such that the instance has the following properties: Given legal 1-dimensional layouts (x_b, C_b) of $\mathcal{F}_b$ and (x_t, C_t) of $\mathcal{F}_t$, their union can be extended to a legal 2-dimensional layout. Consequently, both FET rows can be processed independent of each other without any consideration of the opposing FET row.
The required property is enforced by defining vertical intervals $[y_b^{\min}, y_b^{\max}]$ and $[y_t^{\min}, y_t^{\max}]$ which are reserved for the placements of both rows, that is, $y_*^{\min} \leq y(F) \leq y_*^{\max} - l_f(F) + 1$ must hold for $F \in \mathcal{F}_*$ and $* \in \{b, t\}$. The intervals are chosen sufficiently pessimistic so that no design rule can be violated when the FETs are placed accordingly.
By construction, every optimal canonical layout that is generated for both rows separately (line 3) can therefore be extended to a legal 2-dimensional cell layout. For both rows Algorithm 2 is used to quickly compute an upper bound for the width of an optimal legal layout. Then, as discussed in Section 4.4.4, an optimal canonical legal layout is determined, and W_{best} is initialized with the previously found upper bound.

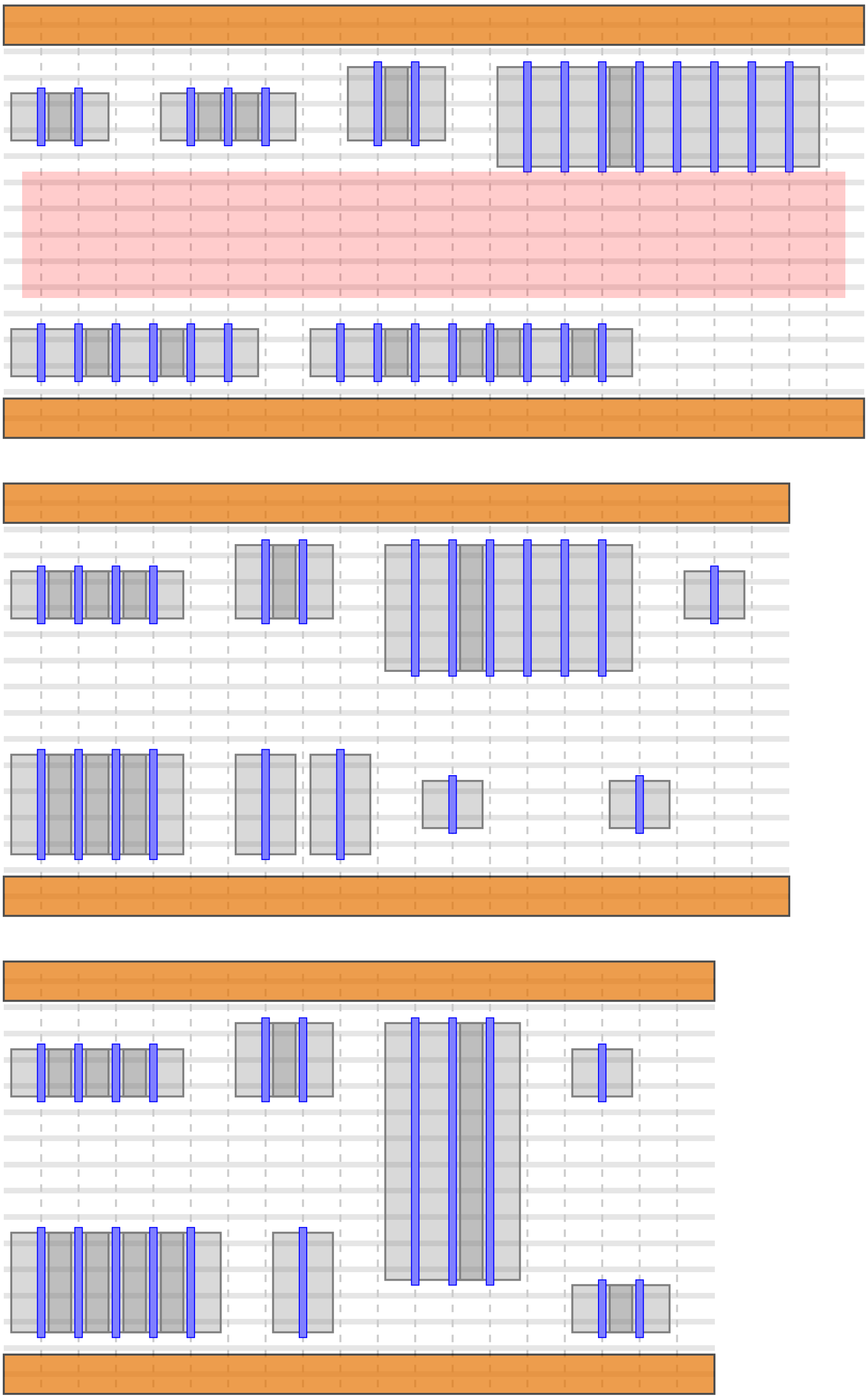


Figure 5.2: Layouts of the same netlist at the end of phase 0 (the red region is reserved for wiring), phase 1 (with a maximum finger length of 5), and phase 2 of Algorithm 7.

Algorithm 7: `CellPlacement`

```
   Data: A netlist $\mathcal{F}$
   Output: A 2-dimensional layout $\Lambda_{\text{best}}$
1  Compute partial netlists $\mathcal{F} = \mathcal{F}_b \dot{\cup} \mathcal{F}_t$
2  (Phase 0) Bound finger lengths for independent row placement
3  Compute optimal canonical legal layouts of $\mathcal{F}_b$ and $\mathcal{F}_t$
4  Extend the layouts to a legal 2-dimensional layout $\Lambda_{\text{best}}$
5  $Q_{\text{best}} \leftarrow Q(\Lambda_{\text{best}})$
6  (Phase 1) Bound finger lengths to $l_f^{\max}$
7  Compute optimal canonical legal layouts $\Lambda_b$ of $\mathcal{F}_b$ and $\Lambda_t$ of $\mathcal{F}_t$
8  Let $(\Lambda_{\text{big}}, \mathcal{F}_{\text{big}})$ be the larger row and $(\Lambda_{\text{small}}, \mathcal{F}_{\text{small}})$ the smaller row
9  $W_{\text{LB}} \leftarrow W(\Lambda_{\text{big}})$
10 while no legal 2-dimensional layout has been found in phase 1 do
11     for all legal 1-dimensional layouts $\Lambda_{big}$ of $\mathcal{F}_{big}$ with $W(\Lambda_{big}) = W_{LB}$ do
12         Find a legal 1-dimensional layout $\Lambda_{\text{small}}$ of $\mathcal{F}_{\text{small}}$ with
           $W(\Lambda_{\text{small}}) \leq W_{\text{LB}}$ such that $(\Lambda_b, \Lambda_t)$ can be extended to a legal
           2-dimensional layout $\Lambda$ of $\mathcal{F}$ and the quality of $\Lambda$ is optimal
13         if $Q(\Lambda) < Q_{best}$ then
14             $\Lambda_{\text{best}} \leftarrow \Lambda,\ Q_{\text{best}} \leftarrow Q(\Lambda_{\text{best}})$
15     $W_{\text{LB}} \leftarrow W_{\text{LB}} + 1$
16 (Phase 2) Remove restrictions on finger lengths and repeat steps 7–15
17 return $\Lambda_{\text{best}}$
```

Phase 1

Although the rigid constraints on the length of the fingers are relaxed in phase 1, the FET dimensions are again constrained for this part of the flow. Here the purpose of that restriction is not to be able to handle both FET rows independently, but to find globally optimal, or at least nearly optimal, solutions while at the same time the search space is reduced as much as possible. The motivation for this approach lies in the observation that in globally optimal 2-dimensional cell layouts very long fingers occur rarely. The value $l_f^{\max}$, i.e. the maximum length of a finger during this phase of the flow, was decided upon based on the actual distribution of finger lengths in optimal layouts. For cells that have the most common size, the data given in Figure 5.3 lead to the choice of $l_f^{\max} = 5$. For smaller and larger cell outlines, the value scales proportionally. In comparison, phase 0 has to run with $l_f^{\max} = 3$ in order to avoid errors near the center of the cell.

After constraining the finger lengths, BONNCELL generates optimal canonical layouts for both FET rows (line 7). Similar to the preceding phase those layouts are computed independent of each other, so this time these layouts

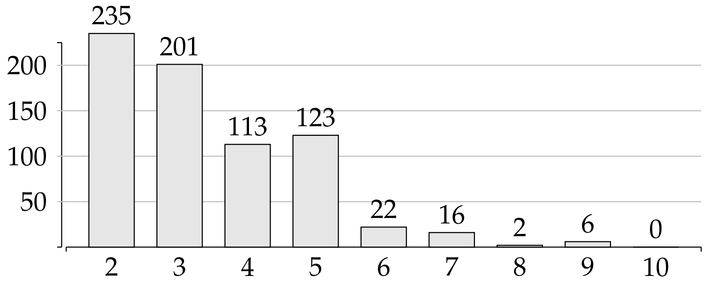


Figure 5.3: Number of FETs with k fins per finger in 63 globally optimal layouts from the CLC test bed (cf. Section 7.1). The maximum possible finger length was set to 10.

cannot, in general, be extended to a legal 2-dimensional layout. They serve two other purposes: First, the layouts indicate what the larger transistor row is, which is renamed $\mathcal{F}_{\text{big}}$. Second, the width of the larger row serves as a lower bound for the width of a legal 2-dimensional layout. This lower bound, W_{LB}, is then used to check if a legal 2-dimensional layout of that width exists. As long as this is not the case, the lower bound is increased by 1 (line 15). The process stops when the first feasible solution has been found, i.e. when W_{LB} is set to the smallest width for which a legal 2-dimensional layout exists.

The loop in line 11 is implemented by the enumeration technique described in Section 4.4.4. Each time the branching method encounters the case $\mathcal{F}_u = \varnothing$, i.e. all FETs in the large row have been placed without exceeding the prescribed width W_{LB}, another routine is called that lays out $\mathcal{F}_{\text{small}}$.

Because the program flow leaves the top-level *while* loop as soon as the first legal layout has been found, it is known in every iteration that no smaller layouts exist.

Lemma 10. *When line 11 is executed in Algorithm 7, then no legal 2-dimensional layout Λ with $W(\Lambda) < W_{LB}$ exists.*

Due to the fact that within this flow the quality of the layout is optimized rather than its width, an extended bounding possibility is gained: If for the already laid out left part of the FET row the gate netlength exceeds the gate netlength of the best known solution, which by the previous lemma cannot be more compact, then the branch and bound tree can be pruned. To achieve this, `LowerBound` returns ∞ if $NL_g(\Lambda_{\text{part}}) > NL_g(\Lambda_{\text{best}})$, where Λ_{part} is the partial layout that has been fixed in the current step of the algorithm.

Phase 1: Small Row Pruning

When $\mathcal{F}_{\text{big}}$ has been fully laid out, including the FET's y-coordinates, the algorithm must decide if $\mathcal{F}_{\text{small}}$ can also be laid out legally. Before this is done by a fully featured exact layout algorithm, a pruning method is applied that

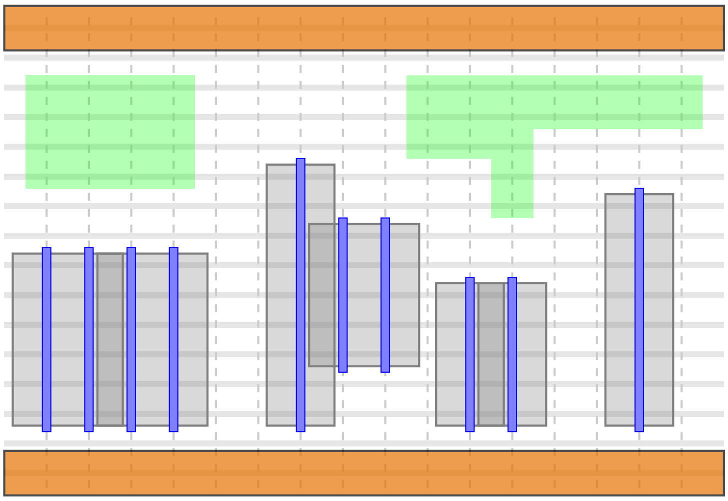


Figure 5.4: Only the fin/gate intersections in the green region can be covered by gates in the top FET row. During small row pruning, BonnCell tries to prove that the area does not suffice without calling an exact layout algorithm.

detects in many—but not all—cases when the unused part of the cell does not suffice to place the remaining FETs. The additional amount of runtime spent in this estimation is very small compared to the runtime that would be wasted in failed attempts to lay out small rows exactly.

The idea of the small row pruning is to focus on the gates or, more precisely, on all the possible fin/gate intersections (which can be identified with the elements of $\Gamma := \{0, \ldots, W_{\text{LB}} - 1\} \times \{y_{\min}, \ldots, y_{\max}\}$). On every x-coordinate in this discrete grid a set of y-coordinates is unavailable for the placement of $\mathcal{F}_{\text{small}}$. These blocked sets contain

- the indices of fins that are used by the layout of $\mathcal{F}_{\text{big}}$,
- fins that must be reserved for the wire access of $\mathcal{F}_{\text{big}}$,
- or fins lying between $\mathcal{F}_{\text{big}}$ and the adjacent power strip.

The usable points in Γ form a shape that must suffice for the layout of $\mathcal{F}_{\text{small}}$. Figure 5.4 illustrates the idea. Note that the exact definition of the usable fin/gate intersections depends on a large number of design rules affecting various layers of metal. The focus on the gates makes the swap status of FETs irrelevant, so it is not considered. Every laid out FET can be seen as a rectangle in Γ, and therefore we only need to consider, for a fixed finger length k, the realization with the fewest number of fingers that results in fingers of length k. For example, if a FET must have fingers of length 3 if configured with 4 to 6 fingers, then only the rectangular representation with width 4 and height 3 must be considered.

Similar to the 1-dimensional layout algorithm, a branch and bound method is employed that recursively generates an arrangement of the aforementioned simpler FET model from left to right. For a given initial sequence of these FETs, all possible rectangular shapes of all unplaced FETs are placed at the smallest possible x-coordinate, thereby extending the initial sequence by one element. The smallest possible x-coordinate is affected by two factors:

- If there are no common source/drain nets with the previous FET in the row, then diffusion sharing is impossible and a gap of size d^{+}_{min} is introduced.
- Several design rules regarding two opposing gates on the same x-coordinate can be checked, and if a rule is violated the corresponding placement of the FET can be forbidden.

If the layout of a subset of the FETs requires at least d_{max} tracks more space than another layout of the same subset, then no further branching must be performed on this sub-layout because the more compact solution, together with a gap of size d_{max}, can be substituted for it in every final solution. If no solution is found with this method, then this serves as a proof that no legal 2-dimensional layout of $\mathcal{F}_{\text{small}}$ can be found that extends the layout of $\mathcal{F}_{\text{big}}$. As a consequence, this layout can be pruned and no exact algorithm for the second transistor row must be called.

It should be noted that the small stack pruning itself is an exponential-time algorithm. Its implementation within BONNCELL ensures that no significant amount of runtime is spent in this method by limiting its available runtime. Moreover, the algorithm does not start if $|\mathcal{F}_{\text{small}}|$ is too large or if the number of fin/gate intersections covered by $\mathcal{F}_{\text{small}}$ is less than half of the available number of fin/gate intersections in Γ.

Phase 1: Small Row Placement

When the small row pruning method could not prove that the small row has no chance to be laid out within the remaining free space, an exact placement algorithm is called in line 12. This algorithm seeks a layout Λ_{small} of $\mathcal{F}_{\text{small}}$ with the following properties:

- $W(\Lambda_{\text{small}}) \leq W_{\text{LB}}$
- $(\Lambda_{\text{small}}, \Lambda_{\text{big}})$ can be extended to a legal 2-dimensional layout Λ of $\mathcal{F}$
- $Q(\Lambda) < Q_{\text{best}}$

The implementation of the method is again derived from the method proposed in Section 4.4.4 to enumerate layouts whose width does not exceed W_{LB}, which ensures the first item on the list. The second item is satisfied by a modification of the `LowerBound` function. Although many complicated design rules have an influence on the realizability as a 2-dimensional layout, the rules can be checked locally by a call to the legality oracle. The oracle only has to examine the direct neighborhood of a FET F, including the FETs in $\mathcal{F}_{\text{small}}$ that were placed immediately to the left of F and the FETs in $\mathcal{F}_{\text{big}}$

in the vertical proximity of F. If a non-legalizable situation is detected in `LowerBound`, it returns ∞, thereby forcing subsequent backtracking.

To aid the detection of such situations efficiently, several data structures are initialized when the layout of $\mathcal{F}_{\text{big}}$ is fixed. Among them are arrays that contain, for every x-coordinate of a gate, the information how much fins are available on that coordinate, what net must be connected to the gate from the larger FET row, and which Vt level appears in the opposing row. Another important part of the legalizability detection is a function that, given two gate contacts with the same x-coordinate, returns the minimum number of fins between the gates that must not be covered by one of the FETs, i.e. the minimum vertical distance. This is crucial because if two *different* nets must be connected to two such gates, then a large amount of free space between the gates needs to be reserved for wires. If, on the other hand, the same net is accessed, the routing can just insert a vertical wire on the PC layer and only a small amount of free space must be left between the transistors. The exact rules affecting this function heavily depend on the technology and are not discussed in detail here.

The third item in the list above provides even more bounding opportunities than in the layout of $\mathcal{F}_{\text{big}}$. By Lemma 10, the width of the layout is again irrelevant for the target function. But the netlength can be estimated much better because one half of the cell is fixed in this situation. The idea is to maintain lower bounds NL_g^{LB} and NL^{LB} of the gate netlength and total netlength of legal layouts that extend the current partial layout within the algorithm. We describe how this is achieved for the gate netlength, the total netlength is handled analogously.

$NL_g^{\text{LB}}, NL^{\text{LB}}$

When the layout of $\mathcal{F}_{\text{big}}$ is fixed, numbers b_N^- and b_N^+ are computed for every net that store the leftmost and rightmost coordinate of an *already placed* contact that accesses N, i.e. $b_N^- := \min T_g(N)$ and $b_N^+ := \max T_g(N)$. If a net does not appear in the larger row, the values are initialized with $b_N^- := \infty$ and $b_N^+ := -\infty$. During the layout of $\mathcal{F}_{\text{small}}$, the numbers are updated as FETs are placed and unplaced. Then $\sum_{N\in\mathcal{N}} \max\{0, b_N^+ - b_N^-\}$ remains a lower bound for the total netlength.

b_N^-, b_N^+

But this bound can be improved: Assume that when `LowerBound` is called there are still k fingers of FETs in $\mathcal{F}_u$, the unplaced FETs, which need to be connected to N. Then it can be anticipated that after all FETs have been laid out, b_N^+ is at least $2(W(\mathcal{F}_l) + k - 1)$, where the factor 2 is required because the unit of netlength is half-track. Hence, if k_N denotes the number of fingers connected to N in $\mathcal{F}_u$ (or equals $-\infty$ if no such fingers exist), then we can set

$$NL_g^{\text{LB}} := \sum_{N\in\mathcal{N}} \max\{0, \max\{b_N^+, 2(W(\mathcal{F}_l) + k_N - 1)\} - b_N^-\}.$$

Now `LowerBound` is modified to return ∞ if NL_g^{LB} is strictly larger than the gate netlength of the best known solution. If equality holds, then NL^{LB}, which is maintained analogously, is compared to the total netlength of the best known solution.

Phase 2

Phase 2 is identical to phase 1 with the only difference that the constraints on the finger lengths are lifted and only the technology-defined (or user-defined, cf. Section 5.4) limits are imposed. The consequence is that the algorithm has much more freedom to choose the sizes of transistors, and thus requires much more runtime in many cases. However, as a good or even optimal solution has already been found in the previous phase, this solution can be returned in case of a timeout. Figure 5.3 serves as evidence that in many cases no solutions will be found in phase 2 that have not been found in phase 1. More substantial evidence will be presented in Section 7.2.1.

Finger Enumeration

The enumeration of the finger numbers, which is part of Algorithm 6, can be improved in the context of quality optimization. Assume that for some $F \in \mathcal{F}$ we have $L_f^F(k) = L_f^F(k+2) = \{l\}$. In other words, increasing the number of fingers by 2 does not change their lengths. For a simple distance function that essentially depends on the finger lengths, which we consider in the implementation of BONNCELL, this implies that the distance requirements between F and neighboring FETs do not change.

It is easy to see that no component of the quality measure can improve, and that $\sum_{F \in \mathcal{F}} n_f(F) l_f(F)$ strictly degrades, when such a FET enlargement is performed. We can thus forbid such operations by setting $L_f^F(k+2) := \varnothing$ whenever $L_f^F(k+2) = L_f^F(k)$ holds.

Following the arguments above, we can conclude that the flow generates provably optimal layouts among all layouts with one row of n-FETs and one row of p-FETs.

Lemma 11. *When Algorithm 7 finishes, Λ_{best} is a legal 2-dimensional layout of $\mathcal{F}$ such that $\Lambda_{best} \leq \Lambda$ for every legal 2-dimensional layout Λ of $\mathcal{F}$ with $\mathcal{F}_b := \{F \in \mathcal{F} \mid t(F) = n\}$ and $\mathcal{F}_t := \{F \in \mathcal{F} \mid t(F) = p\}$.*

5.2.2 Pins

An important technological aspect that has been neglected so far are the external connections of a cell, the so-called *pins*. In a completely laid out cell those pins are regions on the wiring that are marked as access points to be used on a higher hierarchy level. On this level of the VLSI design hierarchy, cells are rectangular black boxes, devoid of internal structure, for which only the locations are specified at which the wires that have to be connected to the cell must touch their rectangular outlines. pins

$p^N_{x,\text{abs}}, p^N_{y,\text{abs}}, p^N_{x,\text{rel}}, p^N_{y,\text{rel}}, p^N_{\text{layer}}$

In the domain of transistor-level layout, this imposes additional structure on both the input and output of a layout tool: For every net N the user may specify four values $p^N_{x,\text{abs}}$, $p^N_{y,\text{abs}}$, $p^N_{x,\text{rel}}$, and $p^N_{y,\text{rel}}$. If N must be connected to the outside of the cell, a layer $p^N_{\text{layer}} \in \{\text{M1}, \text{M2}\}$ must also be specified.

The values of $p^N_{x,\text{abs}}$ and $p^N_{y,\text{abs}}$ are absolute pin locations. If $p^N_{x,\text{abs}}$ is given, then a piece of metal that belongs to N must access, on the layer p^N_{layer}, some point at the x-coordinate $p^N_{x,\text{abs}}$. If both absolute coordinates are provided by the user, they encode a single point on the layer p^N_{layer} that *must* be covered by the metal associated with N.

$p^N_{x,\text{rel}}$ and $p^N_{y,\text{rel}}$ are in a sense softer constraints. They provide relative coordinates in the range $[0, 1]$. If $p^N_{x,\text{rel}} = 0$, then N should connect a pin near the left border of the cell, $p^N_{x,\text{rel}} = 1$ encodes the right border of the cell. However, it is not enforced that the exact coordinates are connected. Rather a routing is computed without any knowledge of the pin and then the pin is assigned to the net's piece of metal that happens to be closest to the location given by $p^N_{x,\text{rel}}$ and $p^N_{y,\text{rel}}$. BONNCELL must only ensure that the net possesses *any* metal on p^N_{layer}. If both relative and absolute coordinates are prescribed, the absolute values take precedence.

During the transistor placement, the vertical pin locations, both absolute and relative, are ignored. There is nearly no freedom in that direction considering the FET placement. The horizontal pin locations, however, are taken into account. In all computations of the total netlength, the pin coordinates are considered as one of the tracks that have to be connected to the net.

While this is a simple modification for absolute pin locations, issues arise if $p^N_{x,\text{rel}}$ is defined. In many versions of the placement algorithm the layout's width is not given a priori, so the actual x-coordinate of a pin with a relative x-coordinate is unknown. The issue is solved with the introduction of some fuzziness in the pin coordinate. Lower bound computations work with a sufficiently optimistic estimate of the final location and upper bound computations use a pessimistic estimate. The impact of this fuzziness on the runtime is negligible because only the ternary optimization goal is affected by pin locations.

5.2.3 Folding

The folding of multiple FETs, i.e. interleaving the fingers of more than a single transistor, was introduced in Section 2.3. BONNCELL supports the folding of exactly two FETs. The feature is implemented as follows: Before a layout algorithm is executed, folding candidates are searched among all pairs of FETs. Such pairs (F, F') must satisfy strict constraints. Both transistors F and F' must have a source/drain contact in common and their sizes, Vt levels, and types must be identical. Under these tight restrictions, the set of candidates contains only a small number of FET pairs in practical instances rather than the theoretically possible quadratic number.

Afterwards, a layout is generated for each subset of the candidate pairs for which no FET participates in more than one pair. For that purpose the FETs that occur in these pairs are split into multiple single-finger FETs according to a number of rules defined by the technology. The best of those layouts, according to a modified quality measure, is then returned.

For reasons rooted in the technology constraints, BONNCELL enforces the selected FET pairs to be arranged in a folded fashion. It is not allowed for "fragments" of one of the original FETs to be placed arbitrarily. As a result, the complexity of the cell layout problem is reduced although the number of objects that are to be laid out increases. The enforcement of the folding scheme is implemented within the universally employed lower bound computation: If FETs have been placed contrary to the required pattern, `LowerBound` returns ∞.

If the generated layouts would be evaluated by the quality measure Q, then the default layout without any multi-FET folding would be favored. Because folding is performed to simplify the distribution of source/drain contacts and improve electrical properties of the cell at the cost of a higher gate netlength, the technique's advantages are not captured by the definition of $Q(\Lambda)$. At the same time, its disadvantages have a strong negative effect. When layouts are evaluated, we therefore compensate for this bias by counting the gate netlength of folded FET groups as if the corresponding transistors were not folded.

In case the BONNCELL user wants, for some reason, to enforce the folding of specific FETs, he may specify *folding groups*. Folding groups are transistor pairs that are arranged in a folded fashion despite any negative effect this might have. Providing this type of guidance to the tool reduces the problem complexity and increases the processing speed.

folding group

5.2.4 FET Rows with Mixed Types

In line 1 of the layout generation flow, n-FETs are assigned to the bottom transistor row and p-FETs to the top transistor row. This simple rule is a very reasonable choice in the majority of cases. On the one hand, n-FETs are usually not connected to V_{DD} in CMOS cells and p-FETs are not connected to GND (although there are exceptions). On the other hand, mixing n-FETs and p-FETs within a single row of transistors creates the necessity of additional diffusion gaps. Since FETs of different types must fulfill the strictest requirements regarding their distance, it is unlikely that rows with FETs of both types lead to more compact solutions than default rows.

However, there are a few such cases. BONNCELL provides a parameter that, if switched on, enables a limited support for mixed rows. In this mode,

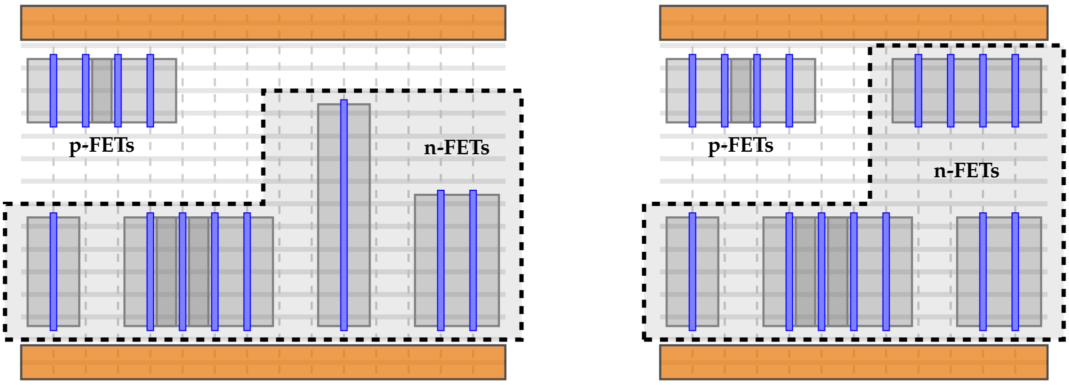


Figure 5.5: The same netlist laid out with the default row assignment and with a mixed row. In BONNCELL mixed layouts have one homogeneous FET row and one FET row with at most two type changes.

layouts with two transistor rows are generated subject to the following constraints:

- One of the two rows contains only n-FETs or only p-FETs.
- In the other row, the n-FETs or the p-FETs (or both groups) are placed in a consecutive sub-row.

An example is shown in Figure 5.5. The first condition is reasonable because the feature is most useful for cells which have a gravely uneven amount of n- and p-FETs so that some transistors from the larger group ought to be moved to the other group. Both constraints also ensure that the total number of diffusion gaps between FETs of different types is very small for mixed rows (at most 2).

To compute mixed-row layouts, we first determine if n-FETs from $\mathcal{F}_n$ should be transferred to the group $\mathcal{F}_p$ of p-FETs or vice versa. Optimal canonical layouts are computed independently for both rows, say Λ_n and Λ_p, and the larger group is chosen as the one to donate FETs to the other group. This step is only provided with a relatively small amount of runtime—if a timeout occurs, the row is selected based on the best solution found up to that point. We will henceforth assume w.l.o.g. that the larger FET group is $\mathcal{F}_n$ and that its FETs are numbered $F_1, \dots, F_k$ from left to right in the initial canonical layout. The next step is to identify a subset $\mathcal{F}_{\text{switch}} \subseteq \mathcal{F}_n$ that is transferred to $\mathcal{F}_p$. As not all subsets can be checked in detail, we identify several candidates for $\mathcal{F}_{\text{switch}}$. These are determined as follows: For every $i \in \{1, \dots, k\}$ we set

$$j_i := \operatorname*{arg\,min}_{i \le j \le k} |W(\Lambda_n) - 2W(\Lambda_n^{\{i,\dots,j\}}) - W(\Lambda_p) - d^{\max}|,$$

where Λ_n^I is the part of Λ that encompasses the FETs $\{F_i \,|\, i \in I\}$. Then $\mathcal{F}_i := \{F_i, \dots, F_{j_i}\}$ is considered to be one candidate for $\mathcal{F}_{\text{switch}}$. Observe that the indices are selected such that the difference between the transistor rows becomes as small as possible if the FETs in $\mathcal{F}_i$ move to the opposite row. The occurrence of $d^{\max}$ in the formula above comes from the fact that the minimum distance between FETs of different types is equal to that number for all distance functions of real-world technologies.

This method yields a total of k candidate sets. For each of these sets, optimal 2-row layouts are computed with $\mathcal{F}_n \setminus \mathcal{F}_{\text{switch}}$ in the bottom row and $\mathcal{F}_p \cup \mathcal{F}_{\text{switch}}$ in the top row. The bounding is modified so that at least one FET type in the inhomogeneous row forms a consecutive interval. At the end, the best of the k layouts, according to the quality measure Q, is returned.

5.2.5 Very Large Cells

The algorithms involved in the layout of transistor-level netlists are exponential and thus, beyond a certain complexity, do not yield results in an acceptable time frame. Especially when timeouts occur in the first two phases of Algorithm 7, the returned solutions are often very bad. Restricting finger lengths may ease the problem, but some cells are even too large for the easiest parameter configurations.

For this reason, BONNCELL includes a mode designed to handle very large cells. In this mode, optimality is sacrificed in favor of a fast computation of heuristic arrangements. A top-level description is given in Algorithm 8. The idea is to split the complex netlist into several less complex netlists, process each of them individually, and finally concatenate the computed layouts. The flow is visualized in Figure 5.6.

Number of Subcells

One input parameter of the algorithm is the number of subcells into which the large instance should be split. This number k is determined automatically based on various measurements of the exact algorithm's performance. In the current implementation, the number k is chosen such that the average number of FETs per subcell is at most 10 and the average sum of gate lengths is at most 80 per subcell.

Target Function

Previously, the gate netlength has been a much more important value than the total netlength. This decision was motivated by the desire to realize connections between the gates on the PC and M0 layers as compact as possible. However, other challenges become prevalent on very large cells.

One can observe that most of such cells contain so many long horizontal connections that, using the normal placement engine, some x-coordinates have to be passed by too many different nets for the cell to be routable. Hence, it is more important to reduce the horizontal cut, i.e. the largest number of nets

Algorithm 8: `BigCellPlacement`

```
  Data: A netlist $\mathcal{F}$, a number $k$
  Output: A 2-dimensional layout $\Lambda$
1 if pins are defined near the left and right cell borders then
2     Place FETs with such pins near the cell's border
3     Compute positions for other FETs by an iterative mean heuristic
4 else
5     Compute rough placement using a linear arrangement heuristic
6 Split netlist into $k$ smaller instances based on rough placement
7 Apply CellPlacement to every sub-instance
8 return Concatenation of the $k$ layouts
```

that have to cross any given x-coordinate, as much as possible. On complex cells, we therefore do not give the gate netlength any priority as we did in the definition of the layout quality Q.

Rough Placement

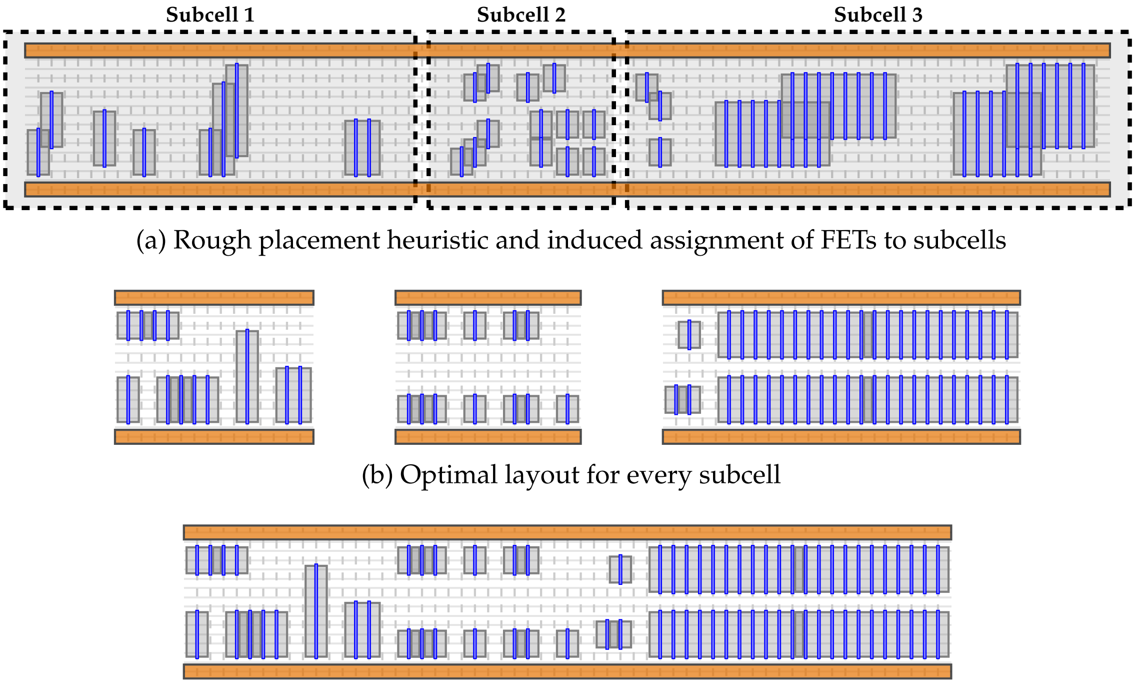
In order to find a reasonable cut at which the netlist is split, a rough placement is determined in lines 1 to 5, using one of two methods. These methods neither produce legal layouts nor determine swap statuses or finger numbers, their purpose is merely to generate horizontal locations for each FET.

If the instance contains for both the left and the right cell border at least one net with a pin defined near that border, then a heuristic is applied that takes advantage of those pins. This is the case for most of the cells in practice, as input pins are usually assigned to the left cell border and output pins to the right cell border. The first step is to place the FETs that are connected to such nets very far apart. For FETs with a connection to the left border, $x(F)$ is set to 0, for FETs connected to the right cell border, it is set to $100k$. All remaining FETs are initially placed at the coordinate $50k$.

Thereafter, all locations are updated in 10 iterations. In each iteration, the new x-coordinate for a FET F is determined as the weighted average of the FETs connected to F through at least one net. Net weights are used as weights in the computation. Gate connections are not prioritized over source or drain connections for the reason given above. After new coordinates have been assigned to all FETs, the arrangement is stretched to the interval $[0, 100k]$ in order to prevent all FETs from collapsing to a single point. In particular, $x_{\min} := \min_{F \in \mathcal{F}} x(F)$ and $x_{\max} := \max_{F \in \mathcal{F}} x(F)$ are computed and then

$$x(F) \leftarrow \frac{x(F) - x_{\min}}{x_{\max} - x_{\min}} \cdot 100k$$

is applied to each FET.



(a) Rough placement heuristic and induced assignment of FETs to subcells

(b) Optimal layout for every subcell

(c) Resulting large cell layout

Figure 5.6: Illustration of Algorithm 8.

If the pin definitions do not induce a reasonable initial assignment of varying x-coordinates, a heuristic developed to solve the MINIMUM LINEAR ARRANGEMENT problem is applied instead. In this case, we define a graph $G = (\mathcal{F} \cup \{v_l, v_r\}, E)$ in which two FETs are adjacent if a net exists that connects these two FETs. The two special vertices v_l and v_r represent the left and right cell borders and are adjacent to all FETs which are connected to a net on which a corresponding pin is defined. Note that the method is only applied when v_l or v_r is an isolated vertex.

After the graph has been constructed, the method proposed by Pantrigo et al. [PDCM11] is applied, which is an extension of the earlier heuristic by McAllister [McA99]. Algorithm C3 from [PDCM11] is used as a subroutine with β set to 0.5. The method is modified such that v_l is the first vertex to be labeled and v_r is the last vertex labeled during the algorithm. When a label $l(F)$ has been assigned to all FETs, those values are taken as x-coordinates of the rough placement in line 5 of Algorithm 8.

Splitting the Netlist

The x-coordinates of the rough placement induce a total order $\leq$ on the set of FETs. If two FETs F_1 and F_2 have the same location in the rough placement, then $F_1 \leq F_2$ holds if and only if $g_1 \leq g_2$, where g_i is the index of the net

connected to the gate of F_i. This tie-breaking criterion favors assignments in which FETs with the same gate net are contained within the same subcell. We number the FETs $F_1, \ldots, F_n$ such that $F_1 \leq \ldots \leq F_n$.
Then indices $n_1, \ldots, n_{k-1}$ are chosen and the sets $\mathcal{F}_i := \{F_{n_{i-1}+1}, \ldots, F_{n_i}\}$, where $n_0 := 0$ and $n_k := n$, are defined as the transistor sets for the subcells. The numbers are chosen such that

$$\sum_{i=1}^{k} \left| \sum_{F \in \mathcal{F}} \frac{L(F)}{k} - \sum_{F \in \mathcal{F}_i} L(F) \right|$$

is minimized, i.e. the sum of deviations from the average FET area. The instances are small enough so that $n_1, \ldots, n_{k-1}$ can be determined by enumerating all $\mathcal{O}(n^{k-1})$ possibilities.
Finally, the nets in subcell i are exactly the nets connected to FETs in $\mathcal{F}_i$. However, if a net N in subcell i is connected to some FET in $\mathcal{F}_1 \cup \ldots \cup \mathcal{F}_{i-1}$, then a pin is defined on the left border of the subcell, i.e. $p^N_{x,\mathrm{rel}} = 0$, to model the connection to the left. Analogously, a pin on the right subcell border is defined if a net is connected to another subcell further to the right.

Placing the Subcells

When the subcells have been constructed, Algorithm 7 is invoked on each of them individually. As the reduced instances are relatively small, this usually results in optimal layouts in a short runtime. These layouts are concatenated with sufficiently large spacing between them so that no design rules are violated. Figure 5.6 illustrates this method.

5.3 Cells with More than Two FET Rows

Albeit very large cells can successfully be constructed using the technique discussed above, such cells have rather long and thin outlines as long as their layout is constricted to the space between a GND and a V_{DD} power strip. These layouts have several unfavorable properties: They probably have very long connections within their interior, which negatively affects their electrical attributes, and they are more difficult to handle by higher-level VLSI tools. The natural strategy to circumvent very long and thin cell outlines is to lay out the transistors in a group of neighboring circuit rows. In BonnCell, the user gains control over the number of rows available for the cell by the parameter `numCktRows` (which defaults to 1). From an algorithmic point of view, several problems have to be overcome:

- A decision has to be made for every FET in which of the rows it is supposed to be placed. Within a circuit row, the default behavior to place n-FETs near GND and p-FETs near V_{DD} then dictates which transistor row the FET must be assigned to.

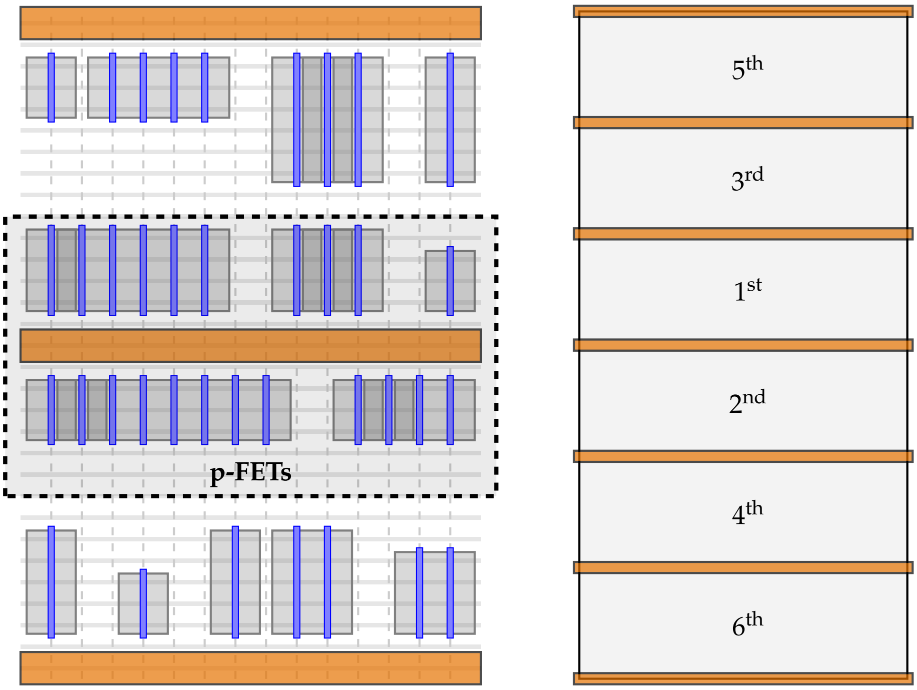


Figure 5.7: A layout with two circuit rows and the order by which the rows are processed in a cell with six circuit rows.

- A subset of the wires has to connect contacts in different circuit rows. These nets have to be routed across one or more power strips, inducing the demand of some kind of free routing space near the power supply.

Before we discuss how these issues are coped with in BONNCELL, it should be noted that the measure $Q(\Lambda)$ is insufficient to realistically assess multi-row layouts. The reason is that a large fraction of the metalized area is caused by vertical connections, and these connections are heavily influenced by the decision which FETs are put into which circuit row. Because of that we define the vertical netlength $NL_y(N)$ of a net and the vertical netlength $NL_y(\Lambda)$ of a layout in analog to the (horizontal) netlengths $NL(N)$ and $NL(\Lambda)$, replacing the horizontal track indices by the indices of the covered fins.

$NL_y(N), NL_y(\Lambda)$

Another important criterion is the cut across the row boundaries: A 2-dimensional layout is considered to be better the fewer nets need to be routed from one side of a power strip to the other side. Formally, we set $C(\Lambda, i)$, for $1 \leq i < \texttt{numCktRows}$, to the number of nets that must cross the boundary between the i-th and $(i+1)$-th circuit row. The *worst cut* $C(\Lambda)$ of a layout is then given as $C(\Lambda) := \max_{1 \leq i < \texttt{numCktRows}} C(\Lambda, i)$.

worst cut $C(\Lambda)$

The quality measure $Q_{\text{mult}}(\Lambda)$ of a 2-dimensional multi-row placement is, other than in the single-row case, an integer rather than a tuple of numbers:

quality $Q_{\text{mult}}(\Lambda)$

$$Q_{\text{mult}}(\Lambda) := \gamma_1 W(\Lambda) + \gamma_2 C(\Lambda) + NL(\Lambda) + NL_y(\Lambda)$$

Algorithm 9: `MultiRowPlacement`

```
   Data: A netlist $\mathcal{F}$
   Output: A 2-dimensional layout $\Lambda$ in numCktRows circuit rows
 1 Set initial $\rho_{\text{best}} : \mathcal{F} \rightarrow \{1, \dots, \texttt{numCktRows}\}$ by fast greedy heuristic
 2 $Q_{\text{best}} \leftarrow Q_{\text{mult}}(\Lambda)$, where $\Lambda$ contains an UpperBound for each row
 3 forall the functions $\rho : \mathcal{F} \rightarrow \{1, \dots, \textit{numCktRows}\}$ without split gates do
 4     $\Lambda \leftarrow$ UpperBound layout for each row
 5     If $Q_{\text{mult}}(\Lambda) < Q_{\text{best}}$, then update $Q_{\text{best}}$ and $\rho_{\text{best}}$
 6 forall the functions $\rho : \mathcal{F} \rightarrow \{1, \dots, \textit{numCktRows}\}$ do
 7     $\Lambda \leftarrow$ UpperBound layout for each row
 8     If $Q_{\text{mult}}(\Lambda) < Q_{\text{best}}$, then update $Q_{\text{best}}$ and $\rho_{\text{best}}$
 9 Fix row assignment $\rho_{\text{best}}$
10 Add dummy FETs for routing between rows
11 Find optimal single-row layout for $\lfloor \texttt{numCktRows}/2 \rfloor$-th row
12 Find modified optimal single-row layout for remaining rows
13 return $\Lambda$
```

The values γ_1 and γ_2 can be changed to influence the relative importance of the layout width and the worst cut compared to the netlength. A weighted sum is preferred over a lexicographically ordered measure because in this setting there is no single most important criterion, single second important criterion, and so on. Instead, it is essential to be able, for example, to spend an additional horizontal track if in turn the worst cut or netlength can be improved greatly.

Assigning FETs to Rows

Algorithm 9 provides a high-level description of BONNCELL's method to find multi-row layouts. In the first phase, it decides upon an assignment $\rho : \mathcal{F} \rightarrow \{1, \dots, \texttt{numCktRows}\}$. The first step of this phase, in line 1, is to compute an initial assignment with a fast greedy method: The FETs are sorted by their size, say $L(F_1) \geq \dots \geq L(F_n)$, and then iteratively added to the initially empty cell. In the i-th step, all possible values for $\rho(F_i)$ are checked. For each of the values, a quick `UpperBound` layout Λ is computed in every row (where only the FETs $\{F_1, \dots, F_i\}$ are placed). Then the value of $\rho(F_i)$ is fixed for which $Q_{\text{mult}}(\Lambda)$ is minimum. After n steps, and thus $n \cdot \texttt{numCktRows}$ upper bound computations, a complete row assignment ρ_{best} has been found.

The loop starting on line 3 basically enumerates all possible functions $\rho : \mathcal{F} \rightarrow \{1, \dots, \texttt{numCktRows}\}$ that do not split gate connections across multiple rows, i.e. $\rho(F) = \rho(F')$ holds if $N_g(F) = N_g(F')$. The algorithm would also work if this loop was omitted, but solutions without such gate splits are considered favorable in practice and the addition of lines 3–5 ensures that this preferred part of the search space is traversed first. Hence in case of a time-out the probability is higher that good solutions have already been found. The next loop, which starts on line 6, then enumerates all possible functions ρ. To be precise, the enumeration is again implemented as a branch and bound with frequent lower bound computations. Similar to the methods that compute single-row layouts, large parts of the branch and bound tree can be pruned with this technique. After that loop has finished, the "best" encountered row assignment is explored further. There are two details to note here:

- The assignment ρ_{best} does not necessarily belong to a 2-dimensional multi-row layout that globally optimizes Q_{mult}. Hence, the algorithm does not compute an optimal solution—it can only be seen as a heuristic that generates good layouts in practice.
- To compensate for this drawback, the algorithm does not only store the best candidate for a row assignment, as in the presentation above, but actually keeps track of the 10 best assignments that have been found. The following steps are then applied to all 10 candidates and the best of the 10 resulting layouts is returned by the algorithm. We omit this detail in favor of the presentation's clarity.

Users may prescribe row indices for an arbitrary subset of the transistors. If they do, only row assignments are enumerated that comply with this specification.

Multi-Row Layout for Fixed Row Assignment

After an assignment of FETs to rows has been fixed, *dummy FETs* are inserted into the instance to account for wires that need to cross a transistor row but are not represented in that transistor row. In particular, a new FET F_N^i is inserted into the i-th transistor row if

dummy FET

- N is connected to a transistor row with index $j < i$,
- N is connected to a transistor row with index $j > i$, and
- N is *not* connected to a source/drain contact in the i-th transistor row.

In this case the wiring of N has to pass the i-th FET row, but all the source and drain contacts within the row are accessed from the M1 layer by other nets. Hence the probability is high that other nets block the way for the wiring of N. The purpose of the dummy FETs is to occupy one or more tracks, thereby leaving some space in which the wire may pass the transistor row. Those newly introduced FETs have the following properties:

- If gaps that are reserved for the wiring of row-crossing nets must have a width of `dummySize` (default: 1), then the dummy FETs are configured with `dummySize` − 1 fingers. Note that the FET model allows the definition of FETs with 0 fingers, although such transistors do not have any electrical meaning. In this case the "transistor" consists only of a single source or drain contact without an active gate.

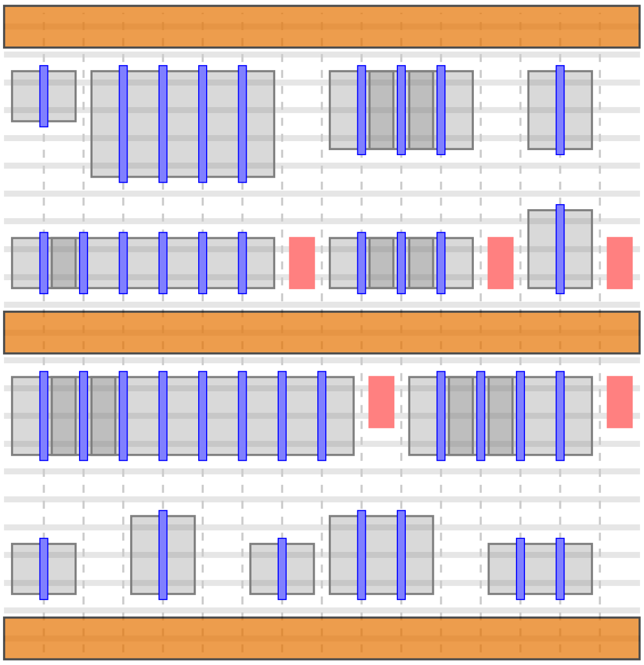


Figure 5.8: Layout with two rows and five 0-finger dummy FETs (red).

- The finger number cannot be changed, and swapping is not allowed.
- $N_s(F_N^i) = N_d(F_N^i) = N$.
- $N_g(F_N^i)$ is set to a newly inserted net that does not connect anything else, so that dummy FETs are not subject to gate netlength optimization.
- $d(F_N^i, F) = d(F, F_N^i) = 1$ for all $F \in \mathcal{F}$ and all configurations of F, i.e. dummy FETs can abut diffusion areas with all other FETs, but they cannot share diffusion.
- The type, Vt level, and length of F_N^i are irrelevant because d does not depend on these values and because the dummy FETs are removed after the layout is finished.

If sufficiently many diffusion gaps would occur without the dummy FETs, the special transistors could fill these gaps without changing the solution in any way. Otherwise, the dummy FETs enforce gaps in their row that can be used for the wiring. Moreover, they are involved in the computation of $NL(N)$ but do not affect $NL_g(N)$. An example of dummy FETs as part of a multi-row layout is shown in Figure 5.8.

After the dummy FETs have been inserted into the instance, optimal single-row placements are generated for each row individually. This is done using a variant of the methods described in Chapter 4. The first row that is processed is chosen to be near the vertical center of the available multi-row region. A default run of Algorithm 6 is applied to this sub-instance.

The remaining rows are processed with a variant of Algorithm 6 that takes the already finished rows into account. Laid out FETs are incorporated into all *total* netlength computations that occur (*gate* netlength is only measured per row because gate connections across multiple rows have to be wired on the M1 layer). The order by which the rows are laid out is chosen such that the distance from the vertical center of the multi-row region increases, i.e. the bottom and top rows are the last ones to be processed. With this strategy, there is always a neighboring row that is already laid out. Figure 5.7 illustrates the order by which the circuit rows are processed.

When placing single rows within the multi-row framework, the layout width is not necessarily minimized. This is only done for the first row near the center of the cell. All other circuit rows are provided with the number of tracks required by the widest row that has been processed thus far. If this width does not suffice to place the FETs assigned to a row, then a minimum-width layout is computed.

5.4 Extensions

This finishes the discussion of the default layout flows for single-row and multi-row cells. Beyond these defaults, BonnCell incorporates many extensions and provides many parameters with which the user can influence various aspects of the design flow.

Runtime Limit

A runtime limit can be specified by the parameter `maxRuntime`. All computations of single transistor rows first check if the runtime limit is exceeded in every node of the branch and bound tree. If this is the case, backtracking is triggered and the best solution that has been found up to that point is returned by the algorithm.

In the basic layout flow of standard cells every step is provided with the part of the runtime that has not been used by the previous steps. For example, if the user-defined runtime limit is exceeded in phase 1, the last phase is skipped. However, in flows where equally ranked computations are performed, the available runtime is distributed evenly among the tasks. For example in a multi-row cell with k subcells, the layout of the i-th subcell will be provided with a runtime limit of $\frac{R}{k-i+1}$, where R denotes the runtime that remains after the computation of the $(i-1)$-th subcell.

Users can setup BonnCell in faster modes that return, in general, suboptimal solutions, but require a shorter runtime. Using the parameter `lastPhase`, it is possible to specify the last phase (0, 1, or 2) that is executed in Algorithm 7. If set to a value smaller than 2, the remaining phases are skipped.

Table 5.1: Values of $l_f^F(k)$ for a transistor with $L(F) = 9$ and minimum finger length 2 dependent on the value of the parameter `fLenMode`.

k	$L(F)/k$	$l_f^F(k)$		
		`int`	`r-up`	`r-down`
1	9.00	9	9	9
2	4.50	–	5	4
3	3.00	3	3	3
4	2.25	–	2	2
5	1.80	–	2	2
6	1.50	–	2	–

Finger Lengths

The gate contacts of transistors cannot be arbitrarily long. In addition to the natural restriction induced by the usable fins in a cell, the technology specification provides parameters `fLenMin` and `fLenMax` that encode lower and upper bounds for $l_f(F)$ for every FET. In principle, these values could be set individually for each transistor, but the implementation of BONNCELL requires the same bound for all n-FETs and the same bound for all p-FETs. Until now we have only assumed $|L_f^F(k)| \leq 1$ but did not give a specification of that function. Because $L(F)$ is supposed to denote the number of fin/gate intersections, the ideal definition would be $L_f^F(k) = \{L(F)/k\}$. However, this might not be an integer, thereby contradicting the discrete nature of the length of a finger. The behavior of BONNCELL in this situation can be controlled by the parameter `fLenMode`, which has the three possible values `int`, `r-up`, and `r-down`. If it is set to `int`, then the exact $L(F) = n_f(F) \cdot l_f(F)$ is enforced, meaning that

$$L_f^F(k) := \begin{cases} \{L(F)/k\} & \text{if } L(F)/k \text{ is integral} \\ \varnothing & \text{otherwise.} \end{cases}$$

In the other two cases, the only element of $L_f^F(k)$ is the closest integer to $L(F)/k$, as long as it is within the interval [`fLenMin`, `fLenMax`]. If $L(F)/k$ is an odd multiple of 0.5, then in one of the modes the value is rounded up (`r-up`) and in the other mode the value is rounded down (`r-down`). An example of the parameter's effect is shown in Table 5.1.

Additional Tracks

The definition of the quality measure $Q(\Lambda)$ favors layouts that use the minimum number of tracks over every strictly larger layout. The total width as the primary optimization target is often a good choice in practice, but

sometimes the restriction to minimum-width layouts produces bad results. In these cases, providing 1 or 2 additional tracks beyond those that are required leads to layouts that need slightly more space yet much less wiring.
In BONNCELL, the issue can be addressed by the integer parameter `maxAddlTracks` (default: 0). It modifies the quality measure such that all layouts Λ with $W_{\text{opt}}(\mathcal{F}) \leq W(\Lambda) \leq W_{\text{opt}}(\Lambda) + \texttt{maxAddlTracks}$ are treated as if their width was $W_{\text{opt}}(\mathcal{F})$, i.e. only the secondary entries of $Q(\Lambda)$ influence how different layouts compare to each other. Only when the width exceeds $W_{\text{opt}}(\Lambda) + \texttt{maxAddlTracks}$, layouts are considered strictly worse than all layouts that are "small enough".
The algorithms can easily be adapted to this rule. For example, the loop in line 10 of Algorithm 7 does not stop as soon as a solution has been found, but it continues for `maxAddlTracks` iterations after that point. Moreover, lower bounds are not compared to the best known solution, but to a number which is the given number of tracks larger than that.

Boundary Condition

When several cells are placed next to each other within a circuit row, it is important that all design rules are satisfied near the gap between them. Because those rules are affected by shapes in different cells, they cannot be captured directly by the layout within a single cell. One way to solve the problem would be to always introduce k unused tracks between neighboring cells, where k is chosen pessimistic enough so that the free tracks suffice to make rule violations in such gaps impossible.
But this strategy would waste much space and a more precise technique is supported in BONNCELL. For every technology, a set of *boundary conditions* is defined that impose additional constraints near the left and right border of a cell. They are designed such that neighboring cells, both of which satisfy these constraints, do not violate any design rules. Examples for common boundary conditions are the requirement that source/drain contacts on the leftmost and rightmost track must be connected to the nearby power potential, or that fingers on the leftmost and rightmost tracks must not be longer than some technology-dependent value. The boundary condition can be switched on and off by the user parameter `boundaryCondition`.

boundary condition

The feature is implemented mainly by causing backtracking as soon as a violation is detected. But it is equally important to adjust the definition of the layout width $W(\Lambda)$, which, until now, was determined merely by the gate locations of laid out transistors. The intention of this definition is to denote the width of the smallest cell outline that can contain a given layout. With the addition of a boundary condition we thus need to augment the definition by adding a sufficiently large amount of tracks such that in a cell of width $W(\Lambda)$ the boundary condition near its right border is satisfied (the left border is already covered by the enhanced bounding).

As the extended definition of layout width does not decrease the value of $W(\Lambda)$, the lower bounds computed by the algorithms above are still lower bounds. Depending on the technology and its concrete boundary conditions, `LowerBound` can be improved to anticipate the increased value of $W(\Lambda)$ in specific situations.

Fixations

In particular cases, a user might want to prescribe a given property that should not be changed by the tool. Several options are provided for that. A trivial option is to fix the number of fingers for a given FET. In this case $L_f^F(k)$ is simply set to $\varnothing$ for all other finger numbers.

boundary fixation

A more sophisticated possibility is to set *boundary fixations*. For every FET it is possible to enforce its placement adjacent to the left respective right cell border, with a source or the drain contact facing outwards. With such a constraint, it is important to see that layouts now cease to have the property that motivated the definition of canonical layouts: It is not possible anymore to arbitrarily swap and exchange FET groups which are separated by a gap of size $d^{\max}$. We thus have to modify the definition of canonical placements to exclude the outermost FET groups if they contain a FET that has a boundary fixation and adjust the specialized bounding procedure.

Aside from this, the boundary fixations are implemented in terms of additional bounding as soon as violations are detected.

Flexible Power Strips and Odd Rows

To simplify the presentation, we have always assumed that the bottom border of a single-row cell is covered by a GND power strip and the top border by a V_{DD} power strip. This is not necessarily the case and BonnCell provides facilities to change this default power structure.

First, it is possible to define an arbitrary number of power strips with arbitrary y-coordinates. It is only required that they are disjoint. Power strips can be configured to only have metal on M2 or to be accessible from both M1 and M2. In cells with more than one circuit row, every second row has the same power structure as the bottommost row and all other have a structure which is flipped on the x-axis.

The definition of the power strips, however, does not affect the default behavior that n-FETs (on multi-row cells: in the bottommost row) are placed near the bottom cell border and p-FETs near the top border (on multi-row cells: the top border of the bottommost row). This scheme can be changed by setting the parameter `oddRow`. In this case, the cell layout is flipped vertically in every circuit row, i.e. n-FETs are assigned to the top and p-FETs to the bottom FET row. Figure 5.9 illustrates the difference.

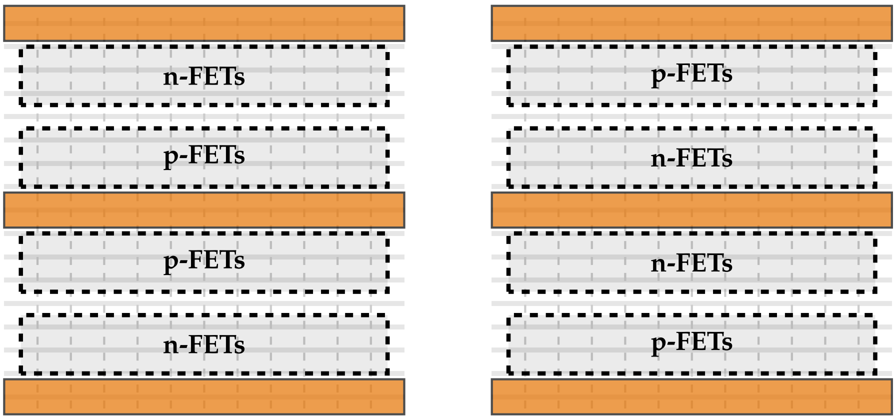


Figure 5.9: Default row assignment if `oddRow` is set to `false` (left) and if it is set to `true` (right).

10 nm Cells

The initial version of BONNCELL has been developed for the 22 nm technology node and was subsequently adapted for 14 nm cells—a major step due to the large differences in the design rules. For example, the layer stack has seen substantial changes and features like double patterning were introduced, which embeds a graph coloring problem into the layout task (cf. Chapter 7). 14 nm cells are, as of now, the primary use case of the toolset.
In addition, BONNCELL incorporates a basic support for 10 nm cells, the next major technology node after 14 nm. The number of changes introduced with this step is much smaller than the modifications required for the previous step. The changes that have been incorporated into the tool include the modification of feature sizes, gridding, and associated input/output requirements. Because design rules are still changing rapidly in the current stage, no rules specific to 10 nm cells have yet been implemented.

5.5 Synopsis of BONNCELL Parameters

`bigCellMode`: Activates the mode for very large cells (Section 5.2.5).

`boundaryCondition`: If set to `true`, additional constraints near the left and right cell border are applied (Section 5.4).

`dummySize`: Specifies the number of tracks reserved for wiring between circuit rows (Section 5.3).

`fLenMode`: Mode that determines how to compute the finger length from the FET size and the number of fingers (Section 5.4).

`fLenMin`, `fLenMax`: Minimum and maximum length of a finger (Section 5.4), can be specified independently for n-FETs and p-FETs.

γ_1, γ_2: Control the quality measure in multi-row cells (Section 5.3).

`lastPhase`: Specifies the last phase executed in the layout flow for cells with two FET rows (Section 5.4).

`maxAddlTracks`: Number of tracks exceeding the minimum number that do not affect the quality measure (Section 5.4).

`maxRuntime`: Maximum total runtime that may be used by the placement algorithms.

`maxRuntimeRouting`: Maximum total runtime that may be used by the routing algorithms.

`mixTypes`: Specifies if FETs of different types may be assigned to the same FET row (Section 5.2.4).

`oddRow`: If set to `true`, the mirroring of circuit rows changes parity, i.e. the bottommost circuit row acts as if it was the second circuit row in a multi-row layout (Section 5.4).

$p^N_{x,\mathbf{abs}}$, $p^N_{y,\mathbf{abs}}$, $p^N_{x,\mathbf{rel}}$, $p^N_{y,\mathbf{rel}}$, $p^N_{\mathbf{layer}}$: Specification of pin locations (Section 5.2.2).

`numCktRows`: Number of available circuit rows (Section 5.3).

`tech`: One of the values `22nm`, `14nm`, or `10nm`. Specifies the technology node (Section 5.4).

Chapter 6

Routing

> Im Gebirge ist der nächste Weg von Gipfel zu Gipfel: aber dazu musst du lange Beine haben.
>
> Friedrich Nietzsche, *Also sprach Zarathustra*

After a layout of the transistors in a CMOS cell has been found, the second major task in physical design is routing: All FET contacts that belong to the same net must be electrically connected by a contiguous 3-dimensional structure within the layer stack given in Table 2.1. This completes the layout of CMOS cells in the sense that it yields functional circuits to be used in a larger context.

The original version of BONNCELL's routing algorithm targeted the 22 nm technology and was implemented by Nieberg and Kölker [Köl12], an overview was published in [HNS13]. The current implementation for the 14 nm version is due to Weyd [Wey13] and Silvanus [Sil13]. In this chapter, we summarize these works and amend them by several further aspects related to the practical implementation.

6.1 Overview

Rather informally, the routing problem can be stated as follows.

CELL ROUTING PROBLEM

Instance: A netlist $(\mathcal{F},\mathcal{N})$, a legal 2-dimensional layout for $\mathcal{F}$, and technology-defined design rules R.

Task: Find disjoint legal wires for each $N \in \mathcal{N}$ satisfying R such that the sum of the wires' lengths is minimum.

In BONNCELL, there are two models by which the desired metalized areas, called *wires*, are described. The *exact model* describes wires as a set of 2-dimensional rectilinear polygons, each of which is associated with one of the metal layers. While from the tool's point of view this shape-based description is more complex to handle algorithmically, real-life design rules, and thus the notion of legality, is defined according to these polygonal wire outlines.

wire

6.1.1 Routing Grid

A simpler, yet less exact model considers the "centerlines" of the wires as collections of 1-dimensional horizontal or vertical line segments in the 2-dimensional plane, again assigned to one of the metal layers. The exact shapes of the wires can then be deduced as the Minkowski sum of these centerlines and, in the simplest case, a rectangle encoding the "width" and "height" of the wires. The centerlines that constitute the wires form subgraphs of a certain technology-defined grid graph, thereby providing a combinatorial structure as a basis on which algorithms can operate more efficiently than on free-form polygonal shapes.

routing grid Ξ

Henceforth, we call this grid graph the cell's *virtual routing grid* and denote it by $\Xi = (V, E)$, where implicit embeddings of the edges are given as axis-parallel straight line segments. We further denote $n := |V|$ and $m := |E|$, a visualization is shown in Figure 6.1. The routing grid consists of a 2-dimensional (possibly irregular) grid graph for every metal layer as well as vertical edges, called *vias*, at locations where neighboring layers may be electrically connected.

via

6.1.2 Legality

The legality of a routing is determined by a large number of shape-based rules that are specified separately for each technology. In the 14 nm node, about 700 rules affect the layers generated within BONNCELL. These range from general grid-related constraints over simple distance or area requirements (e.g. shapes on layer x must have a minimum area of a) to very complex specifications involving more than two polygons on different layers that meet several conditions. In addition, "special constructs" are defined—specific multi-layer configurations of metalized and unmetalized area that are feasible even though they would violate one or more of the aforementioned 700 geometric rules.

In view of the large number and complexity of these constraints, it is beyond the scope of BONNCELL to generate completely legal solutions. Rather, a large fraction of the constraints is satisfied by construction, while some of them are only addressed in a later stage of the algorithm. Very few rules are not implemented at all. The goal is to generate as few violations as possible, as those have to be fixed manually by human layouters.

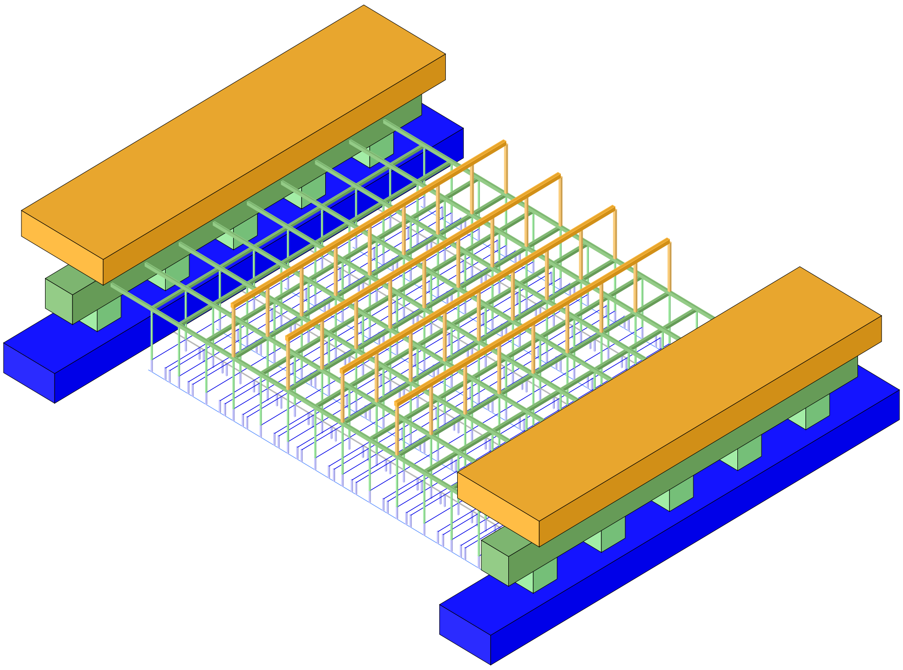


Figure 6.1: Virtual routing grid between two power strips.

Algorithms

The routing implementation in BONNCELL runs in three phases:

- *Combinatorial routing:* Starting with an empty cell, vertex-disjoint Steiner trees are computed in Ξ that satisfy a reduced set of constraints.
- *MIP routing:* Taking the Steiner trees from the first phase as an initial solution, subgraphs of Ξ are rerouted by a mixed integer programming (MIP) formulation with an extended set of design rules.
- *Shape-based post-optimization:* Finally, various shape-based post-optimization steps are applied to the polygonal wire shapes.

The first two modules operate on the virtual routing grid, while only the last module modifies the actual wire shapes. In the following, we will give an overview of the methods used by these three phases.

6.2 Combinatorial Routing

The combinatorial router employs a dynamic programming scheme to efficiently generate Steiner trees in the routing grid. It takes the routing task as a Steiner tree packing problem, i.e. the problem of computing vertex-disjoint Steiner trees, subject to several additional constraints that relate to the design rules, such that the sum of their lengths is minimized. Even highly restricted versions of this problem are NP-hard. For example, it is NP-complete to decide for k pairs of vertices $(s_1, t_1), \ldots, (s_k, t_k)$ if there are vertex-disjoint shortest s_i-t_i-paths even if the graph is planar and all edge lengths are 1 [ET98].

To solve the problem in practice, BONNCELL implements a rip-up-and-reroute approach roughly structured as follows: In each iteration, an unrouted net is chosen and a shortest Steiner tree is generated for this net such that no conflicts with already routed nets are introduced. Moreover, the trees are constructed subject to several additional constraints that are deduced from particular design rules. However, in order to be able to incorporate these constraints into the Steiner tree generation algorithm, only a subset of the design rules are considered in the combinatorial router. Many rules are left to later stages of the layout flow.

The method used to compute exact shortest Steiner trees is an improvent of the Dijkstra-Steiner approach and is presented by Hougardy et al. in [HSV]. For the computation of a Steiner tree with terminals $T \subset V$ it requires a theoretical running time of $\mathcal{O}(3^k n + 2^k(n \log n + m))$, where $k := |T|$. Its idea is to choose some $t \in T$ and compute labels $l(v, I)$ for $v \in V$ and $I \subseteq T \setminus \{t\}$ that encode the length of a shortest subgraph of Ξ connecting v with all elements of I. Similar to Dijkstra's approach to the shortest path problem, the algorithm traverses the graph, thereby merging and generating labels appropriately, until $l(t, T \setminus \{t\})$ is determined.

In addition to its efficient implementation, the Steiner tree generation procedure employed by BONNCELL computes effective lower bounds for the costs required to complete a partial tree. Both the improved Dijkstra-Steiner algorithm as well as its embedding into BONNCELL's routing flow is described in great detail by Silvanus [Sil13]. The second part of Figure 6.2 provides an example of the initial routing of a CMOS cell found by this method.

6.3 MIP-Based Routing

Because many design rules cannot be supported efficiently by the combinatorial router, its output usually contains many violations. The aim of the second phase, the MIP-based router, is to locally modify the combinatorial router's Steiner tree packing, thereby fixing as many rule violations as possible. The fundamental tool used to achieve this goal is the formulation of the Steiner tree packing problem as a MIP as proposed by Grötschel et al. [GMW97]. For each $e \in E$ and each $N \in \mathcal{N}$ a binary variable x_e^N is introduced that encodes if the given edge is occupied by net N, and additional variables x_e, $e \in E$, represent the usage of e independent of a specific net. If $T_N \subseteq V$ denotes the terminals of $N \in \mathcal{N}$, then the solutions of

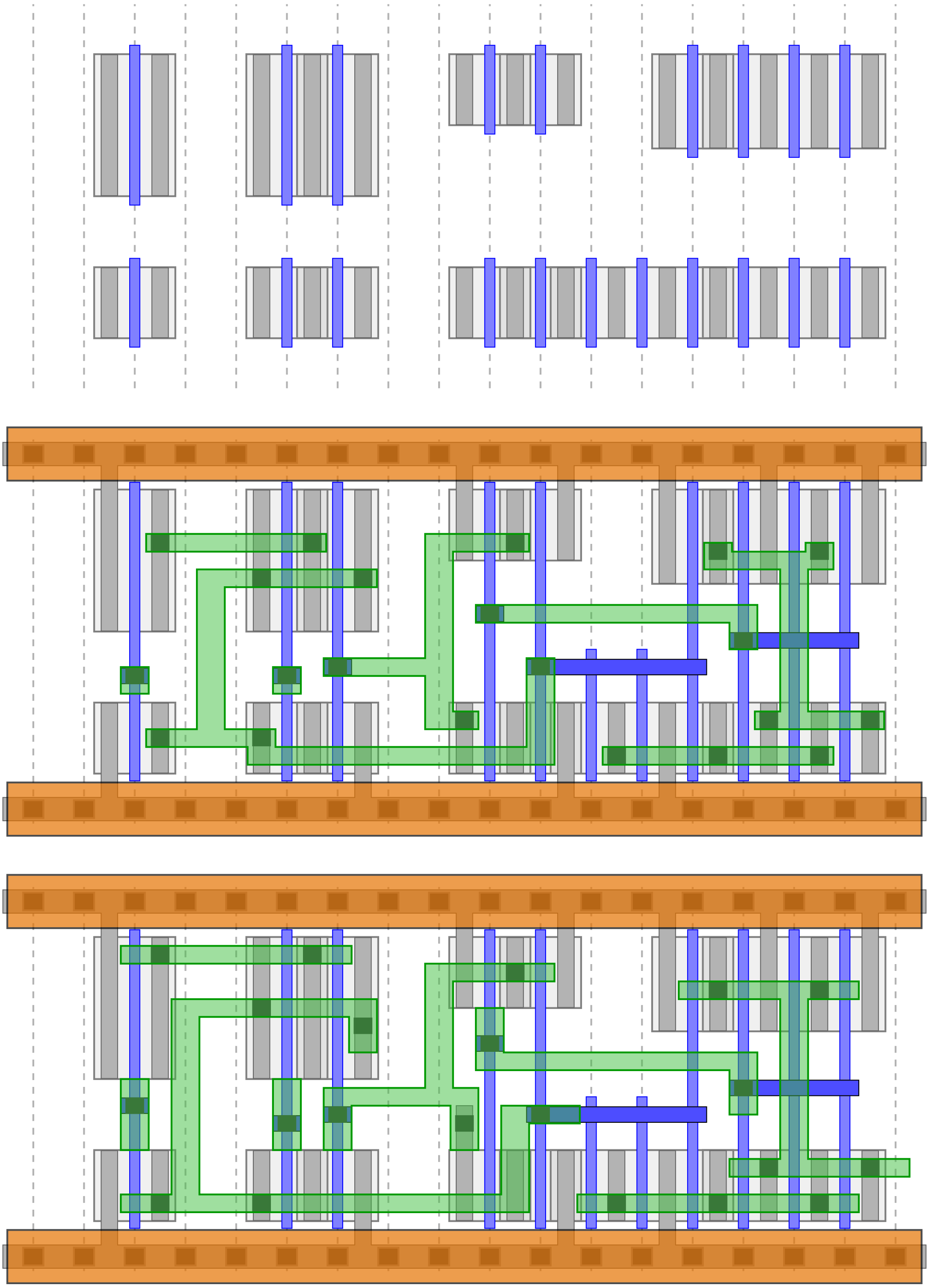


Figure 6.2: Placement, combinatorial routing, and MIP routing.

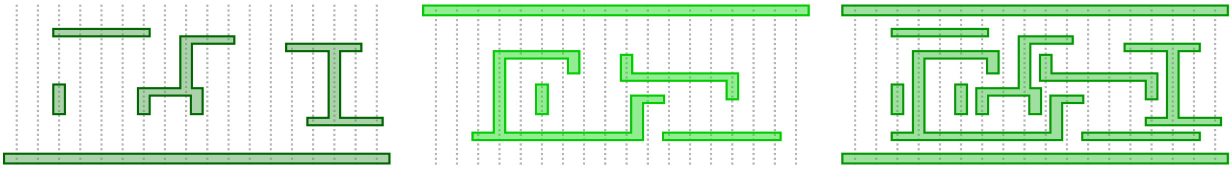
$$
\begin{array}{lll}
\min & \sum\limits_{e \in E} c_e x_e & \\
\text{s.t.} & x_e - \sum\limits_{N \in \mathcal{N}} x_e^N \leq 1 & \text{for all } e \in E \\
& x_e \leq 1 & \text{for all } e \in E \\
& \sum\limits_{e \in \delta(W)} x_e^N \geq 1 & \text{for all } N \in \mathcal{N} \text{ and all } W \subseteq V \\
& & \qquad \text{with } W \cap T_N \neq \varnothing \text{ and } (V \setminus W) \cap T_N \neq \varnothing \\
& x_e^N \in \{0, 1\} & \text{for all } N \in \mathcal{N} \text{ and all } e \in E
\end{array}
$$

are exactly the edge-disjoint Steiner tree packings of Ξ (vertex-disjointness can be achieved with further constraints). Unfortunately, the third set of constraints, which ensures the connectivity of the terminal-connecting subgraphs, is exponentially large. Hence, we instead use flow-based constraints to obtain this property, an idea initially proposed by Beasley [Bea84]. Essentially, new variables model a flow that is sent from one terminal of a net to the net's other terminals, where edges may only carry flow if they are occupied by a net.

This very general framework lays the foundation for a highly flexible router. The following list names several examples for types of constraints that are captured by the BONNCELL MIP:

- Nearly all of the geometric design rules can be converted to linear inequalities in the MIP, sometimes with the aid of additional variables. This includes, but is not limited to constraints on the distance between shapes, their minimum area, their edge lengths, and so on, often involving shapes on different metal layers.
- In the 14 nm technology, a graph coloring problem became part of the physical layout: The structures on some layers are too small to be manufacturable by traditional means. To manufacture these structures nevertheless, multiple masks are produced, each of which contains a subset of the shapes on the given layer. Then both masks, as illustrated in Figure 6.3, are superimposed to create the desired patterns. For the routing problem, this means that each part of a wire that forms a connected component on one of the layers must be assigned one of two "colors" (representing the mask to which the shape is assigned), and that wires with the same color must satisfy stricter constraints.

  In the MIP, these rules can be addressed by splitting the variables for each edge into two variables. One of these encodes if the edge is used by a piece of metal with color 1, the other encodes the same for color 2. All design rules involving shapes of different or identical colors then incorporate the new variables accordingly.



(a) Mask 1 (b) Mask 2 (c) Metalized area

Figure 6.3: Coloring problem as part of the physical design. All shapes in (a) and (b) have a sufficiently large distance to be manufacturable on the same mask.

- It is possible to process only a part of the cell by adding equalities to the MIP that fix the value of an arbitrary set of variables. For example, BonnCell respects pre-wiring, user-defined metalized areas that may be incorporated into any (or any given) net. Moreover, rather than allowing the MIP router to change all nets arbitrarily, only parts of the full cell are usually processed at once. In these cases, all variables outside the considered part of the cell are fixed, which is much easier to implement than generating only the part of the graph within that region.

- Secondary optimization targets can be worked into the target function. For example, as long as it is possible without sacrificing netlength, bends in the wires should be avoided. This can be modeled by counting the number of 90 degree corners in the wires using additional variables and penalizing these variables with a properly weighted term in the target function.

It is worth to note that in this phase the generated subgraphs are not necessarily trees. In some situations, optimal solutions do contain cycles in the wires to avoid rule violations.

When all features of the tool are incorporated into the MIP, its size is extremely large. The cell in Figure 6.2 for instance would require a MIP with 1 842 011 variables and 4 957 847 constraints. In the default flow, the MIP is thus not solved in one step. Rather, several strategies are applied to find feasible solutions for parts of the chip: Routing single nets, sometimes restricted to a tunnel of increasing radius around the combinatorial router's solution, routing multiple nets within small boxes, and so on. The previous solutions are given to the MIP solver as a starting solution. Then, parts thereof are updated every time a feasible solution is found for one of the processed subinstances. The commercial tool CPLEX 12.6 is used to solve the MIPs.

BonnCell's MIP-based router was implemented and described by Weyd [Wey13]. Figure 6.2 illustrates typical changes applied by the algorithm to wires generated in the first routing phase.

6.4 Shape-Based Routing

Until this point in the flow, metal has only been represented by edges in the virtual routing grid. However, some geometric design rules cannot be modeled like this without decreasing the edge lengths to single nanometers, which would drastically inflate the size of the grid. In addition, several layers of metal, which are not directly part of the wires, are not considered at all by the previous steps. Hence, a final shape-based routing post-optimization prepares the cell's layout for the output and subsequent injection into the higher-level VLSI toolset.

The first step in this phase is to generate the actual metal shapes from the previously computed subgraphs of the routing grid. Those are obtained by expanding the 1-dimensional embeddings of the grid's edges in all directions according to the assigned layer's parameters. Thereafter, the following post-optimization steps are applied to the metalized polygons.

6.4.1 Coloring

The MIP-based router implicitly computes a coloring of the wires, but its purpose is merely to guarantee the *existence* of a feasible coloring. In the post-optimization, a new coloring is determined that additionally aims at several soft constraints. For that purpose, BONNCELL generates an auxiliary graph with one vertex for each connected metalized region and edges between vertices if the corresponding pieces must not have the same color. Subsequently, all possible 2-colorings of the graph are evaluated for their conformance with the soft constraints, for example:

- The number of coloring rule violations should be as small as possible (such violations may occur if the previous routing algorithms didn't find a colorable solution).
- The color of a wiring layer should match the color of connected vias on the adjacent layers.
- Both colors should be assigned to roughly the same amount of metal.

BONNCELL also supports the fixation of colors in the pre-wiring: Especially the power rails usually have a user-defined color that must not be changed.

6.4.2 Via Pads

Metal on the wiring layers must overlap connected vias by some amount according to several design rules that depend on the involved layers. To avoid violations, metal is added in the post-optimization to satisfy these rules such that the number of distance violations introduced by this modification does not exceed the number of violations resolved by it. If the algorithm can choose between several variants of these "via pads", it prefers the variant that adds the fewest amount of metal.

6.4.3 Cut Shapes

In the 14 nm technology, the PC layer constitutes a special case. While so far the polysilicon has been treated as an ordinary wiring layer on which the FET gates are located, it is, in the data model, completely covered by vertical PC wires. In places where these must be interrupted, cut shapes have to be defined on a separate layer. These shapes must follow unique design rules by themselves. Using the parameter `ctMode`, BONNCELL gives the user the ability to choose between several strategies:

- *Small rectangular:* The cut shapes consist of a set of disjoint rectangles that are generated as small as possible, and as close to the cell's vertical center as possible. The restriction to rectangles is useful because inner corners of cut shapes need to comply with complicated rules that are not encoded in BONNCELL.
- *Large rectangular:* The cut shapes consist of a set of disjoint rectangles that are generated as large as possible.
- *Non-rectangular:* The cut shapes may be arbitrary rectilinear polygons.

The shapes are determined by selecting for every vertical PC wire the maximum interval which can be covered by cut shapes without violating design rules. Then, depending on the value of `ctMode`, the resulting polygons are modified according to the selected strategy. In the current implementation, the placement algorithms, which assume one of the first two modes, do not change their behavior in dependence on the parameter. Hence, the placement implementation overlooks layouts that are only possible by exploiting the specific non-rectangular cut shape rules. As the mode is rarely used in practice, the restriction is usually without consequences for the optimality.

6.4.4 Additional Layers

The post-optimization also generates several layers of metal indicating particularly processed regions of the underlying substrate. Those are associated with the types of the transistors or their Vt levels. Although they are not part of the wires, they must comply with a number of design rules. The majority of these rules can be satisfied by simple operations on sets of rectangles without any influence on the placement or routing. Some, however, do forbid specific FET arrangements. These are avoided within the BONNCELL flow by incorporating them into the legality oracle employed by the placement algorithms.

All in all, BONNCELL outputs rectilinear polygons on 16 layers of metal, polysilicon, and other materials in the form of an XML file. The VLSI toolchain calling and providing the input for BONNCELL then reads this XML for further processing. This usually includes some manual work to remove remaining errors and finally the usage of the generated layouts in a higher level of hierarchy.

Chapter 7

Results

> To know but not to do is not to know.
>
> Chinese proverb

In this chapter, we describe the experiments that have been conducted to evaluate BONNCELL. We characterize a large set of CMOS cells on which the tool is tested, examine the influence of several parameter settings on the results, compare our layouts—wherever possible—with their industry counterparts, and finally document two scenarios in which BONNCELL has been used by the industry with great benefit.

7.1 Test Beds

BONNCELL was tested on several hundred real-life 14 nm CMOS cells provided by IBM. The entirety of these cells was divided into 3 test beds with varying characteristics and applications. Before the results are presented, we elaborate on the properties of these groups. Table 7.1 summarizes the cells' key data.

CLC: Custom Leaf Cells

The first test bed, henceforth abbreviated CLC, consists of custom leaf cells that have been flagged by our industry partner IBM as appropriate test cases for BONNCELL. Most of these cells have in fact been subject to at least some kind of optimization with the help of our toolchain, ranging from placement runs used to obtain quick estimates on the total size of the final layout up to complete layout generation. This set of cells offers a very large diversity: It contains a small buffer with 4 transistors and some very complex cells having a width of almost 80 tracks.

Table 7.1: Characteristics of the test beds.

	CLC	LIB	LAT
Number of cells	102	198	36
Total number of FETs	1 621	1 284	1 276
$\lvert\mathcal{F}\rvert$ (min/median/max)	4/12/51	2/6/11	16/36/48
Total number of nets	1 630	1 732	948
$\lvert\mathcal{N}\rvert$ (min/median/max)	6/13/46	5/9/11	14/27/36
Sum of optimum widths[1]	2 898	3 543	1 172
$W_{\text{opt}}(\mathcal{F})$ (min/median/max)[1]	4/23/79	2/13/77	12/31/75
Sum of FET sizes $L(F)$	15 140	29 328	4 710
Avg. FET size (min/median/max)	2.1/8/62.7	2/12/200	2.3/2.4/10.7

LIB: Standard Library

The test bed LIB represents the majority of a 14 nm standard library that is in use in server processors at IBM. A standard library is a collection of basic logic functions like AND or NOR circuits or just inverters. They are used as basic building blocks to compose the logic functionality of a processor and make up the largest part of the total chip area.

For the test bed, we selected only the subset of the library consisting of the "regular" Vt level cells, because the cells associated with other Vt levels are identical copies from the physical layout's perspective (the only difference concerns layers that are not subject to our presentation). Thus, unlike the other test beds, all cells in LIB contain only FETs of a single Vt level.

In general, this test bed is the easiest to lay out because the sizes of the netlists are comparatively small. Although the layout of standard cells is not the prime application BONNCELL was designed for—such cells can also be generated by less complex systems due to their simple structure—there are, as discussed in Section 7.4.2, means by which our tool can be used to great avail in the generation of a standard library. Table 7.2 contains a detailed overview of the cell types occurring in LIB. Cells of the same type match in their FET and net numbers. However, the sizes of transistors, and thus of the layouts, vary broadly.

LAT: Latches

Finally, the third test bed LAT is situated at the upper end of the difficulty range. It is a collection of latches and related cells, circuits storing information, and strictly speaking part of the same standard library that constitutes the second test bed. But in contrast to the basic logic functions, these memory elements are highly complex in their structure and, on average, significantly

[1] The smallest known width is used where the optimum is not known.

Table 7.2: Number of occurrences and characteristics of all cell types in the LIB test bed.

Cell type	Count	$\|\mathcal{F}\|$	$\|\mathcal{N}\|$	$W_{\text{opt}}(\mathcal{F})$
Inverter	19	2	5	2 to 38
2-input AND	3	6	8	4 to 6
2-input OR	3	6	8	4 to 6
2-input NAND	19	4	7	3 to 68
3-input NAND	13	6	9	4 to 46
4-input NAND	7	8	11	5 to 31
2-input NOR	16	4	7	3 to 68
3-input NOR	10	6	9	4 to 42
2-1 AND-OR-Invert	17	6	9	4 to 58
2-2 AND-OR-Invert	17	8	11	5 to 77
2-1 OR-AND-Invert	17	6	9	4 to 58
2-2 OR-AND-Invert	17	8	11	5 to 77
2-input XNOR	10	9	9	6 to 24
2-input XNOR + buffer	10	11	10	7 to 29
2-input XOR	10	9	9	6 to 24
2-input XOR + buffer	10	11	10	7 to 29

harder to lay out than the set of custom leaf cells. To account for this difference, the standard library was split into LIB and LAT.

As Table 7.1 shows, transistors in this test bed are usually small yet numerous, leading to a large number of possible permutations and swap configurations. In addition, the netlists are larger (with more than 35 FETs on average) and have more complex nets (with over 4 FETs, on average, connected by each net).

General Test Setup

Unless otherwise noted, all of the following experiments have been conducted on a an Intel E5-2667 CPU at 3.3 GHz. Moreover, BONNCELL was set up with default parameters (cf. Chapter 5) and runtime limits of 2 hours for placement and 6 hours for routing.

7.2 Evaluation of BONNCELL Features

We begin the discussion of the results by stating several properties of the placement algorithms and analyzing the effect of a number of parameters.

7.2.1 Placement with Default Settings

Figure 7.1a shows the runtime of the cell placement (Algorithm 7) required for the CLC cells sorted by total runtime. The colors indicate the time spent in the three phases of the algorithm. A timeout occurs for 13 cells in phase 2, for 18 cells in phase 1 and for 8 cells already in phase 0. Global optima are found for the remaining 63 cells. It may have happened that global optima were also found in some timed out runs, but the optimality could not be proven in these cases.

The chart shows that, in some cases, a phase can run even faster than the preceding phase, although the phases were designed to have an increasing complexity. This behavior can be observed when the additional restrictions imposed during earlier phases enforce a layout to have more diffusion gaps—and thus allows more permutations of the FETs—while the later phase allows very compact placements for which good or even exact upper bounds are discovered early.

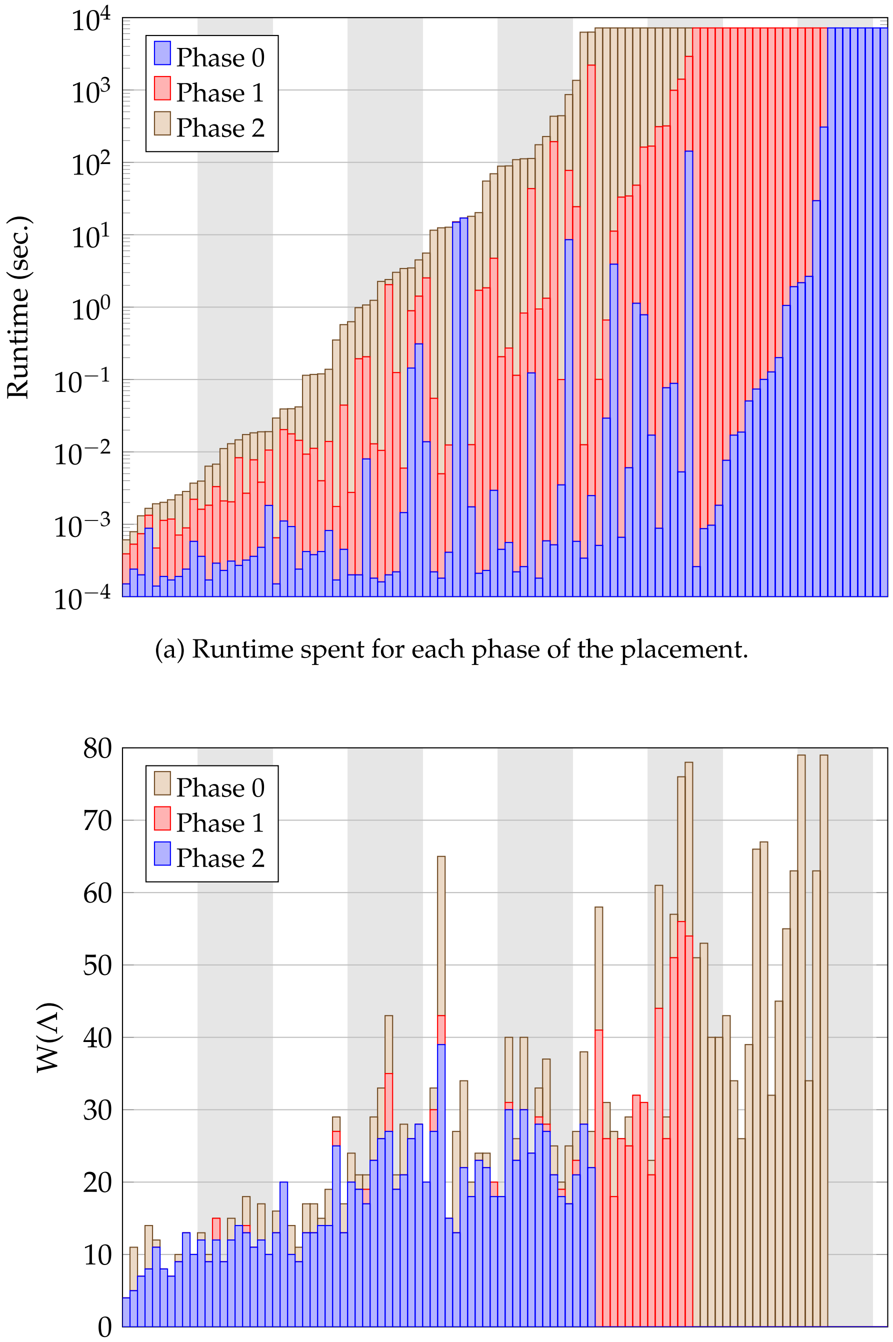
The widths of the generated layouts are illustrated in Figure 7.1b. The columns of the chart correspond to the columns of the chart above. Colors again correspond to the three phases of the algorithm: The blue bar shows the result of phase 2, i.e. $W_{\text{opt}}(\mathcal{F})$. The red bar shows the width at the end of phase 1 and the beige bar the width after the initial phase in which both stacks are placed independently. Only 13 of the 63 cells without a timeout can be placed into fewer tracks after phase 2 than after phase 1, suggesting that in most cases results of phase 1 are already global optima.

Figures 7.2 and 7.3 depict the same data for the test beds LIB and LAT. In both of these groups, pairs of instances are noticeable that display a very similar behavior both in runtime and layout width. These pairs consist of very similar but not identical netlists, e.g. a 2-input `NAND` and a 2-input `NOR` with FETs of the same size.

7.2.2 Additional Tracks

The parameter `maxAddlTracks`, as outlined in Section 5.4, allows the layout to span additional tracks beyond those that would be required for a minimum-width solution. The motivation for this feature is the idea that such non-minimal layouts occupy a slightly larger area, but are in turn “more routable” in the sense that placements are possible with a shorter netlength. These would require less metal for the necessary interconnections and can even cause otherwise unroutable placements to become routable, either due to the reduced netlength or due to the larger amount of available routing tracks.

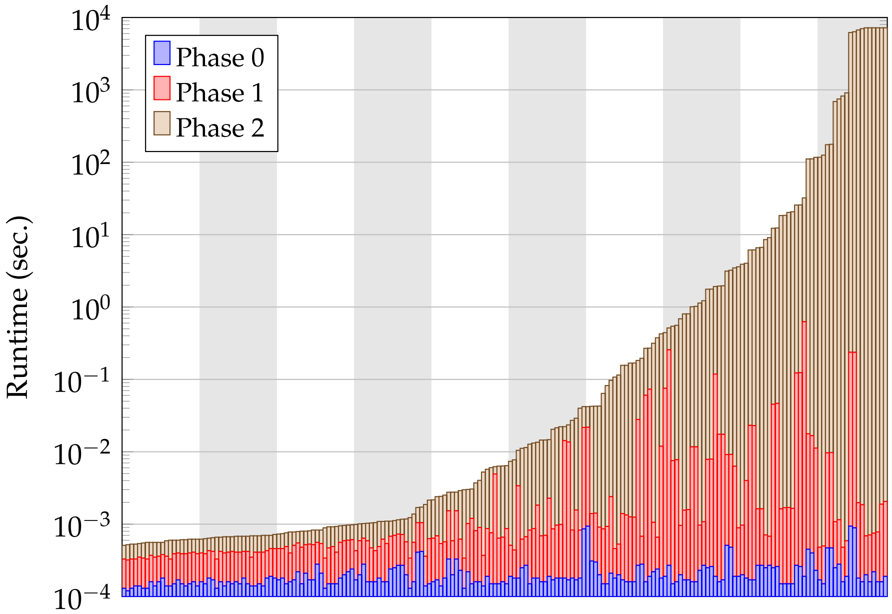
To evaluate the parameter, BONNCELL was run with `maxAddlTracks` set to 0, 1, 2, and 3 on all CLC cells. The routability is estimated by applying the combinatorial router to the generated placements. Although the success of the routing algorithm neither implies nor is implied by the existence of a



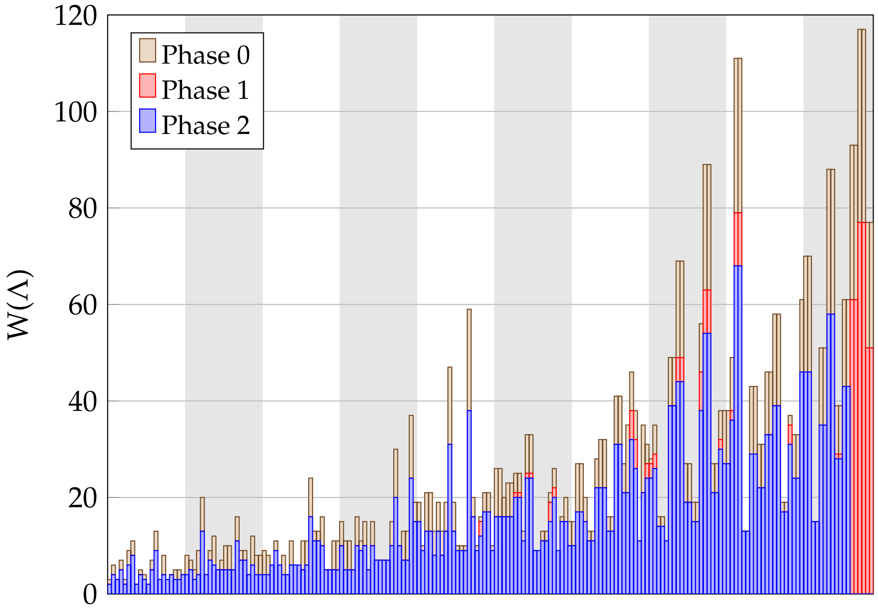
(a) Runtime spent for each phase of the placement.

(b) Layout widths after each of the 3 phases. The chart only shows bars for phases without a timeout. The ordering is the same as in (a).

Figure 7.1: Placement results on the 102 CLC instances with default parameters and a runtime limit of 2 hours.

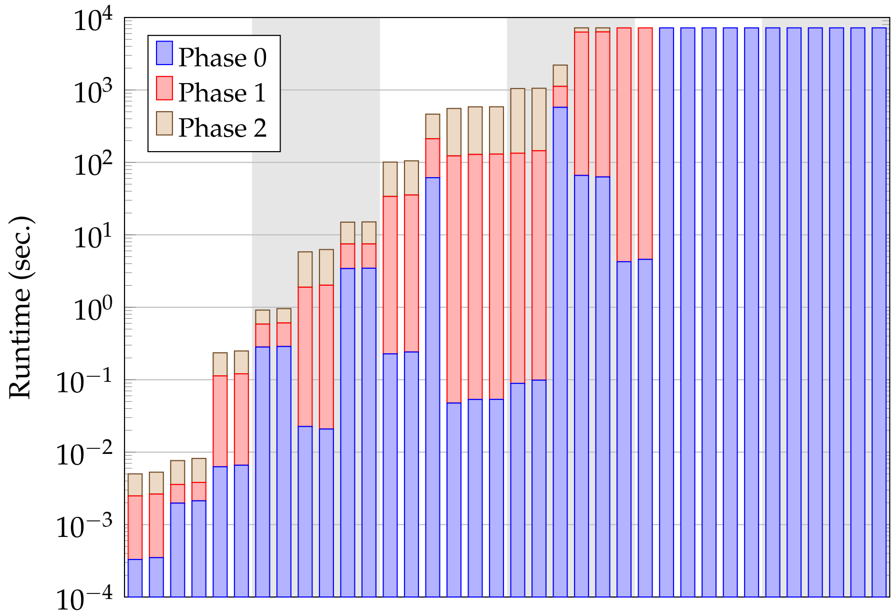


(a) Runtime spent for each phase of the placement.

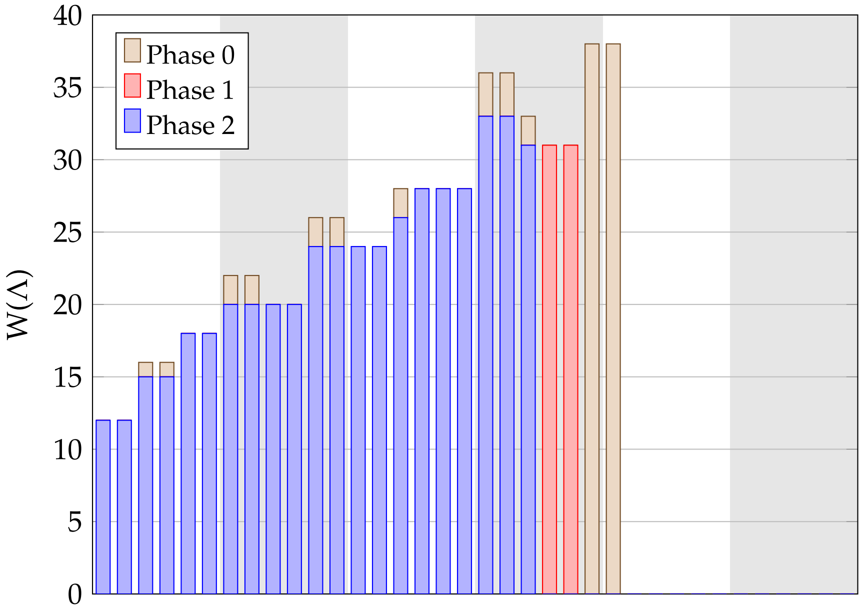


(b) Layout widths after each of the 3 phases. The chart only shows bars for phases without a timeout. The ordering is the same as in (a).

Figure 7.2: Placement results on the 198 LIB instances with default parameters and a runtime limit of 2 hours.



(a) Runtime spent for each phase of the placement.



(b) Layout widths after each of the 3 phases. The chart only shows bars for phases without a timeout. The ordering is the same as in (a).

Figure 7.3: Placement results on the 36 LAT instances with default parameters and a runtime limit of 2 hours.

Table 7.3: Influence of `maxAddlTracks` parameter on layout runtime and quality. The numbers refer to the 50 CLC cells for which globally optimal solutions were found in all cases.

	`maxAddlTracks`			
	0	1	2	3
Sum of placement runtimes (sec.)	736	1 970	7 433	24 109
Average slowdown factor	—	4.7	30.4	195.9
Sum of gate netlengths (half-tracks)	1 070	962	916	890
Sum of total netlengths (half-tracks)	6 157	5 782	6 072	6 194
Number of routings found	47	47	46	46
Number of M2 tracks used by wires[2]	43	35	32	30
Sum of routing runtimes[2] (sec.)	38 295	32 976	27 947	32 259

routing that satisfies all design rules, it is the closest approximation available to us (and, as our experience suggests, a good approximation).

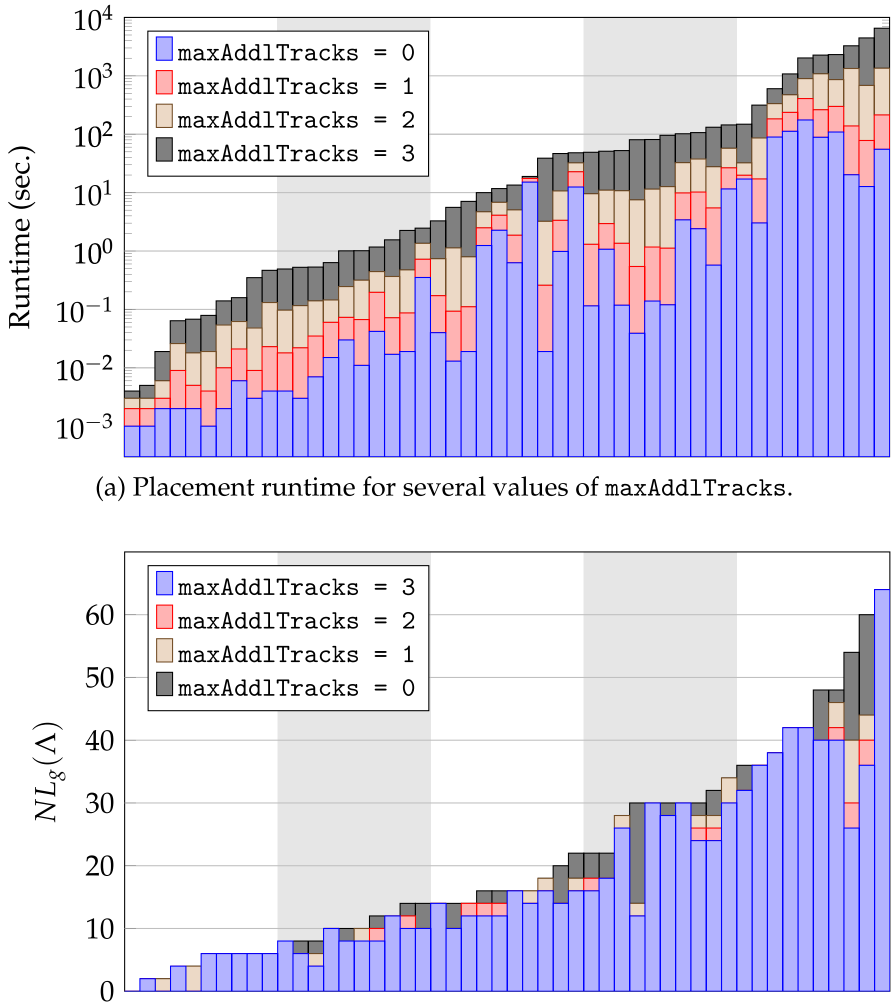
For 50 of the 102 cells, globally optimal layouts were found in all four experiments. Table 7.3 summarizes the results on these 50 cells. The placement runtime increases, as expected, significantly for larger values of `maxAddlTracks`. The behavior is caused by the new degrees of freedom that emerge when the placement becomes less tight. New opportunities of flipping, shifting, or even permuting the transistors are gained, thereby making the problem more complex. In most cases, increasing the number of available tracks by 1 increases the placement runtime by a factor of 2 to 10. The exact numbers for all 50 cells are illustrated in Figure 7.4a. The second row of Table 7.3 gives the average value of the slowdown factor, which is defined as the ratio between the runtime with the according number of additional tracks and the runtime when no additional tracks may be used.

slowdown factor

The next two rows show how the netlengths develop. The gate netlength is the secondary optimization target and thus cannot get larger when additional tracks may be used at no cost. The total netlength, however, can change in both directions because a better gate netlength may come at the cost of an increased total netlength. The table confirms that by allowing larger-than-necessary layouts, the gate netlength can be improved significantly. This is also illustrated in Figure 7.4b in greater detail. The total netlength changes in both directions, with a tendency towards decreasing, but the effect is smaller than for the gates.

The question if a design can be routed at all is hardly affected: While routings are found for 70 layouts with default settings, the number changes to 69 for all tested values of `maxAddlTracks`. Among the 50 cells examined

[2] The numbers only refer to the 45 cells for which routings were successfully found in all four settings.



(a) Placement runtime for several values of `maxAddlTracks`.

(b) Gate netlength depending on the available number of additional tracks.

Figure 7.4: Influence of `maxAddlTracks` parameter on runtime and gate netlength. The bars represent the cells summarized in Table 7.3.

above, 47 are routable when `maxAddlTracks` is at most 1, and one fewer routing is found for larger values. In fact, two cells become unroutable at that point while one further cell that could not be routed for small values of `maxAddlTracks` becomes routable for larger values.

The last two rows in the table support the claim that the routing becomes easier in some sense when more area is available: On the one hand, less tracks on M2 are needed as the cell area increases. This is important because those tracks can also be used by wires on a higher hierarchy level, hence blocking them within the cells is disadvantageous. On the other hand, the total routing runtime tends to decrease, although that number is only loosely related to the “hardness” of the routing problem.

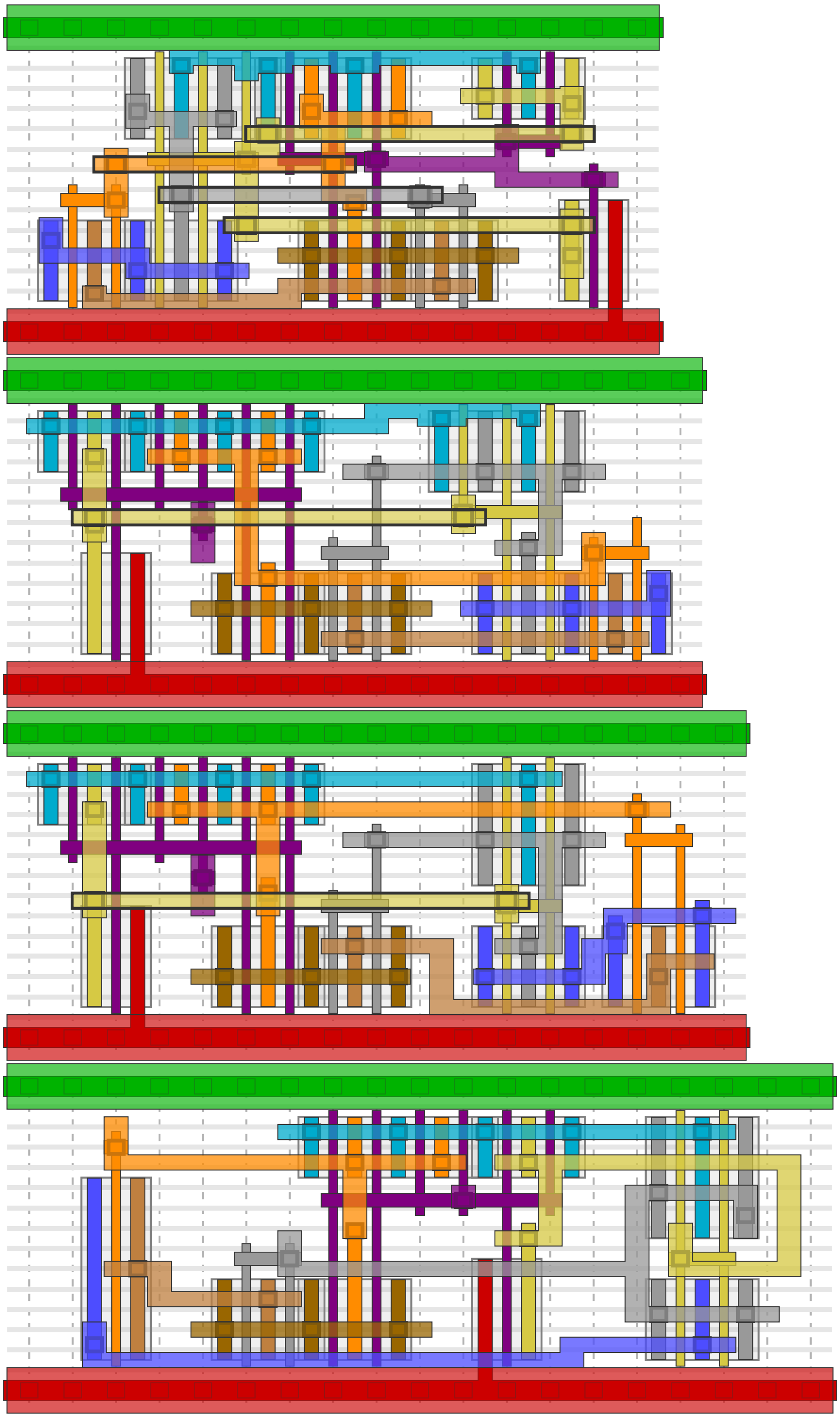


Figure 7.5: Layouts of the same cell with `maxAddlTracks` set to 0, 1, 2, and 4. Thicker outlines indicate M2 metal, shapes are colored by net.

Table 7.4: Comparison of a default run of all CLC cells with layouts where all finger lengths are restricted to 3.

	maxFingerLength	
	∞	3
Sum of placement runtimes (sec.)	297 670	151 605
... for cells w/o timeout in both runs	16 870	1 606
Placement timeouts in phase 0/1/2	8/18/13	8/10/2
Sum of layout widths	2 885	3 376
Sum of gate netlengths	12 232	10 732
... for cells w/o timeout in both runs	1 538	2 376
Successful routings	70	76

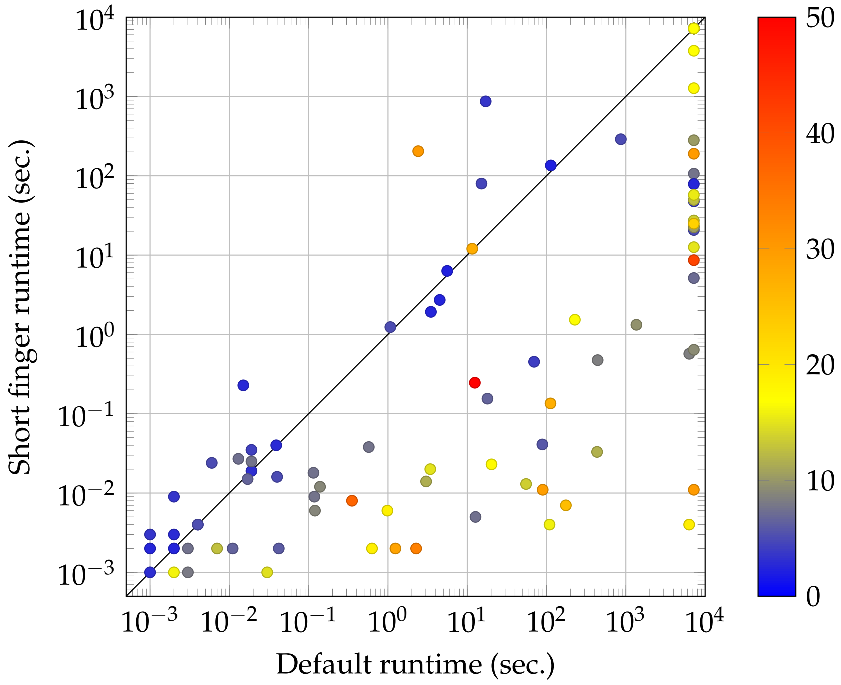
We can conclude that the parameter `maxAddlTracks` can be used to simplify the routing by reducing the netlength and the probability that an M2 track needs to be used at the cost of a significant runtime increase. But taking the routability as a binary yes/no property, no large effect can be demonstrated by these experiments. To cancel the negative effect of the runtime increase, BONNCELL is configured to run in a loop with an increasing value of `maxAddlTracks`. If the user sets the parameter to k, then at first a placement is generated for the value 0, then for 1, and so on until k is reached. Finally, the result from the last iteration without a timeout is returned.

An example of how a concrete cell layout is affected by changing the number of available tracks can be seen in Figure 7.5. It depicts four layouts of the same instance, the only difference being the value of `maxAddlTracks`.

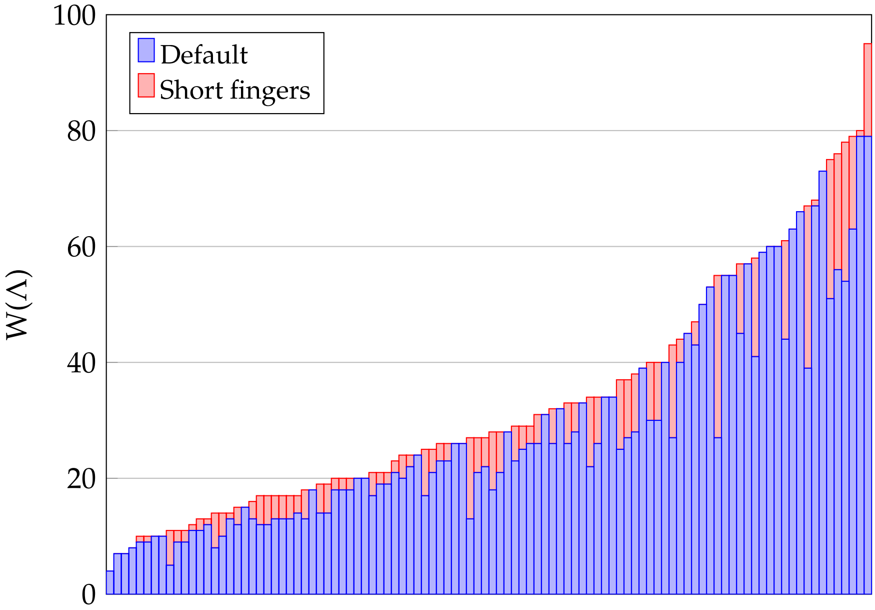
7.2.3 Restricting Finger Lengths

BONNCELL provides control over the maximum number of fins a single finger may overlap, i.e. the maximum finger lengths. This reduces the number of ways by which each FET can be configured, may thus greatly improve the runtime, but also leads to larger layouts. The feature was tested by comparing all CLC layouts with default parameters to corresponding layouts where all finger lengths were restricted to 3.

Table 7.4 contains a summary of the results. The total runtime roughly halves, which is due to 8 fewer timeouts and many other cells that can be laid out much faster. Figure 7.6a, which compares the runtimes of the two experiments, allows a more detailed view on that aspect. The improvements are much more numerous and also larger in their magnitude than the degradations. However, some cells also need more time in the restricted version because less diffusion sharing is possible. The color of the dots indicate the average FET size on the associated cell, suggesting that bounded finger lengths



(a) Placement runtime with default settings vs. placement runtime if finger lengths are restricted to 3. The colors indicate the average value of $L(F)$ on the corresponding cell.



(b) Layout widths with default parameters and with a maximum finger length of 3.

Figure 7.6: Effect of enforced short finger lengths for all CLC cells.

are especially effective in speeding the algorithm up on cells with many large transistors.

The sum of the layout widths increases by about 17%. Figure 7.6b shows, for all 102 cells, the width of the default layout and the width of the restricted layout (which is never smaller). The gate netlengths, however, improve especially when the algorithm with default settings hits the runtime limit. Moreover, some layouts become routable by restricting the finger length. Again this tends to happen when the default run timed out and a better placement can be found by truncating the search space.

7.2.4 Multi-Row Cells

In Section 5.3, we describe BONNCELL's method to generate transistor-level layouts spanning several circuit rows. In practice, this feature is most often activated by the user when the surroundings of the cell dictate another form factor. However, it can also be used to handle very large cells and, in general, produces variants of the default layouts, thereby increasing the chance to find feasible routings. To evaluate the feature, we configured each of the 102 CLC cells to be laid out within 1, 2, and 3 circuit rows.

The Tables 7.5 and 7.6 show the results of these experiments. The first table summarizes the results of the 63 cells on which no timeout occurred in the single-row case, i.e. provably optimal layouts are known for this case. As it turns out, the multi-row runs did not end with a timeout either on these 63 cells. The second table provides the same statistics for the 39 remaining cells with a timeout. Using 2 rows, only 11 of those encountered a timeout, and with 3 rows only a single cell did not finish within the time limit of 2 hours.

The first table suggests that it rarely pays off to generate multi-row layouts when an optimal single-row version is available. The area covered by the cell's rectangular outline increases, on average, by 29% for 2 rows and by 56% for 3 rows. Not a single multi-row cell is smaller than the minimum-width solution in a single row. In principle this would be possible, because arranging the layouts of the individual rows side by side might require additional empty tracks between those blocks due to minimum distance rules. However, considering those 63 CLC cells, the single-row variants are always the most compact ones. When the single-row placement fails, this behavior changes. Table 7.6 shows that in some cases the cell area decreases. If for every cell the smallest variant is chosen, netlists can be laid out with 10% less area on average.

The netlength values in the tables are, in this case, the sum of vertical and horizontal total netlengths. In contrast to the single-row problem, where the vertical netlength is largely fixed and not subject to optimization, we now have a 2-dimensional problem in which both dimensions of the netlengths are relevant. Comparing against non-optimal single-row placements, the multi-row counterparts, which are also non-optimal, can have a significantly

Table 7.5: Multi-row results on the 63 CLC cells that were successfully placed within one circuit row without a timeout. The last column shows the results obtained by taking for each cell the number of rows that produces the best value for the given measure.

	Number of rows			
	1 row	2 rows	3 rows	Best
Change in cell area (avg)	—	+29%	+56%	+0%
Change in total netlength (avg)	—	+17%	+33%	-2%
Sum of total netlengths	15 280	16 900	19 088	—
$NL_g(\Lambda)$ (min/median/max)	0/20/82	0/18/76	0/16/64	—
Sum of gate netlengths	1 538	1 422	1 298	1 188
Sum of placement runtimes	16 870	147	2 335	112
Number of timeouts	0	0	0	0
Successful routings	89%	81%	62%	90%

Table 7.6: Same as Table 7.5, but for the 39 CLC cells for which a timeout occurred in the single-row placement.

	Number of rows			
	1 row	2 rows	3 rows	Best
Change in cell area (avg)	—	-2%	+19%	-10%
Change in total netlength (avg)	—	-28%	-22%	-28%
Sum of total netlengths	37 973	24 458	26 889	—
$NL_g(\Lambda)$ (min/median/max)	24/190/938	18/70/262	18/66/152	—
Sum of gate netlengths	10 694	3 686	2 674	2 536
Sum of placement runtimes	280 800	168 774	60 006	54 536
Number of timeouts	39	11	1	1
Successful routings	36%	49%	46%	64%

better netlength. Increasing the number of rows also tends to improve the gate netlength, which is clear because now more FETs may share the same x-coordinate (however, the transistors are usually assigned so that all FETs sharing a gate contact lie within the same circuit row).

As the multi-row placement does not perform an exhaustive search over all possible assignments of FETs to circuit rows, it does not guarantee to find a global optimum. Only after such an assignment has been fixed, optimal layouts are computed within the individual circuit rows. These can be found much faster than in the single-row case because much fewer FETs are contained in each given row. For this reason, multi-row placements are usually finished much faster, which also reflects in the tables. Using two rows, all instances in the first group can be laid out in a total runtime of 147 seconds.

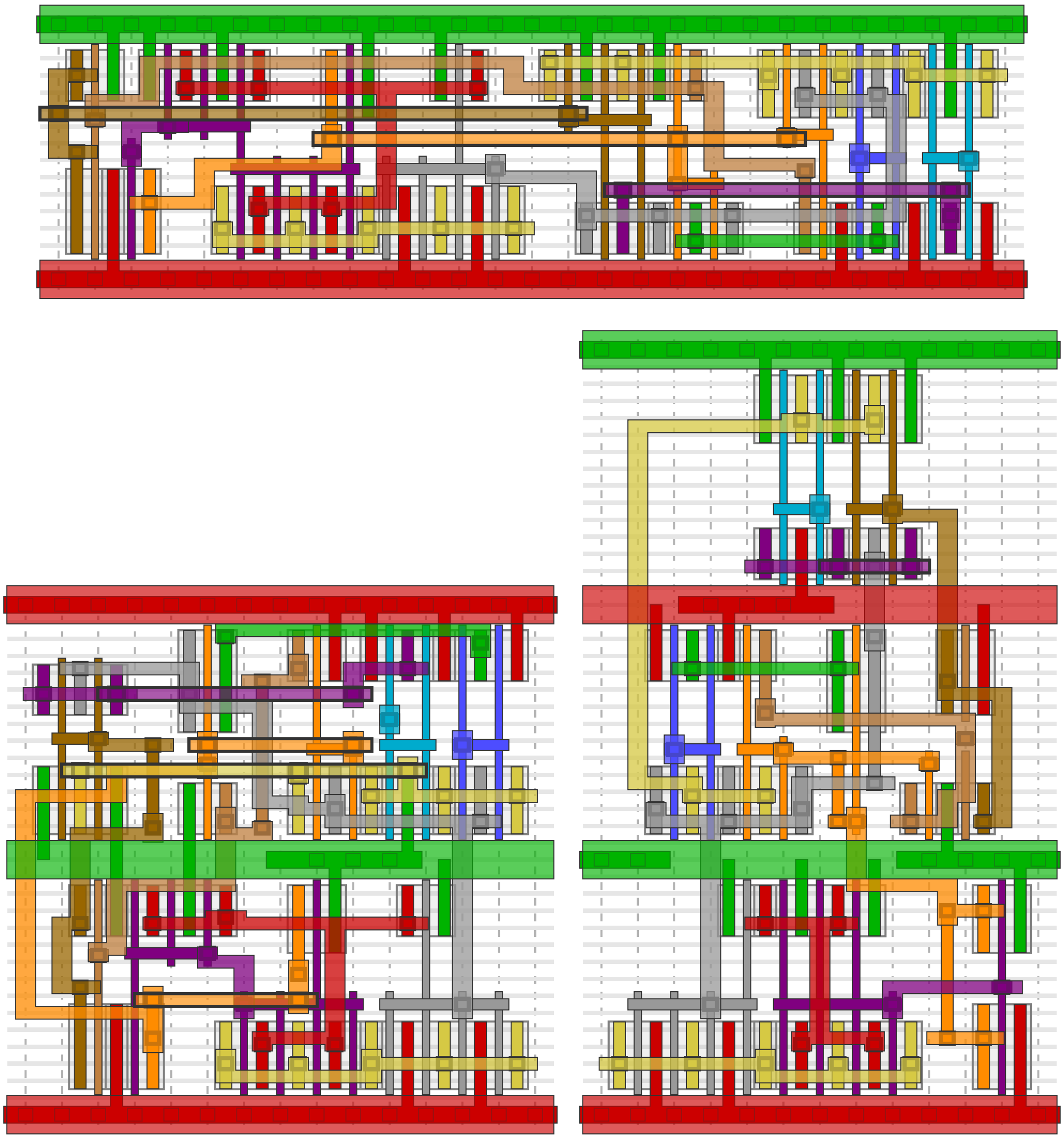


Figure 7.7: The same netlist laid out in 1, 2, and 3 circuit rows.

In addition to the placement's behavior, we evaluated the routability of the layouts. The experiments confirm the trend that with an increasing number of rows the probability to find a routing decreases, which is explained by the added difficulty to route nets through the border regions between circuit rows. When no optimal single-row placement could be found, the multi-row variants—especially those without a timeout—have a higher probability to be routable. For 25 out of the 39 cells in this group one of the three variants was successfully routed. In total, 82 out of all 102 CLC cells were routable in at least one of the variants.

Table 7.7: Placement runtime, layout width, gate netlength, total netlength, and routability with default settings and with mixed FET rows activated. The last column group shows the differences to the default results. If no runtime is given, a timeout occurred.

	FET Areas		Default layout					Mixed FET rows				
Cell	n	p	Runt.	W	NL_g	NL	Rtb.[3]	Runt.	W	NL_g	NL	Rtb.[3]
clc26	11	34	0.0	13	6	51	yes	0.1	-2	+14	-2	yes
clc46	28	10	0.0	14	34	114	no	1.5	-1	-6	-14	no
clc59	288	112	—	79	530	1 774	no	—	0	0	0	no
clc62	94	34	—	26	50	242	yes	—	+3	+74	+84	yes
clc64	120	41	—	40	158	541	yes	—	0	0	0	yes
clc65	26	9	0.0	9	10	51	yes	0.0	0	0	0	yes
clc80	12	5	0.0	7	0	52	yes	0.0	0	0	0	yes
clc85	180	60	2.4	27	42	301	no	—	0	0	0	no
clc89	6	22	0.0	8	6	29	yes	0.1	0	0	0	yes
clc94	219	9	—	54	74	1 048	yes	—	+24	+52	+577	yes

7.2.5 Mixed FET Rows

The strategy to arrange transistors in one row of n-FETs and one row of p-FETs is usually reasonable, but especially netlists with a very unbalanced ratio between n-FETs and p-FETs—more precisely: an unbalanced area covered by these groups—are better placed with mixed rows as described in Section 5.2.4. Figure 7.8 illustrates the ratio between the size of the n-FETs and the size $\sum_{F\in\mathcal{F}} L(F)$ of all FETs for each of the CLC instances. While most of the cells have a comparable amount of n-FETs and p-FETs, some are extremely unbalanced. The instances in LIB and LAT are essentially balanced and not relevant for the evaluation of this feature.

To test the effect of mixed circuit rows, BONNCELL's placement algorithm was applied, with `mixTypes` set to `true`, to the 10 most unbalanced cells in CLC. Detailed results on these cells are given in Table 7.7. Only 2 of these 10 cells were placed with mixed circuit rows. For the other 8 cells two reasons prevented mixed rows from being laid out:

- In some cases the best layouts have very high p-FETs where every finger covers many fins and comparatively flat n-FETs (or vice versa). This scheme is often better than mixed rows to handle unbalanced netlists.
- When timeouts occur in the phase where the assignment of FETs to FET rows should be determined, which happens on the large cells, the trivial assignment separating n-FETs from p-FETs is chosen as the one for which the final layout is computed.

[3] Indicates if the combinatorial router found a routing within 6 hours.

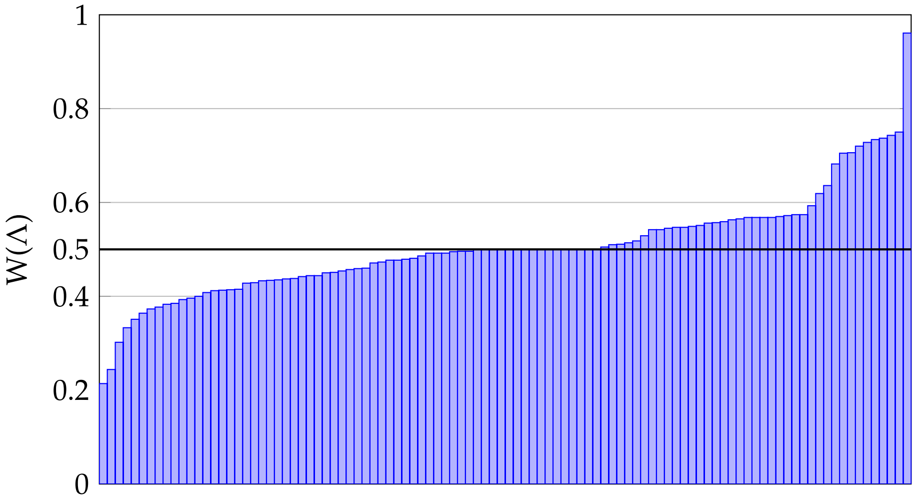
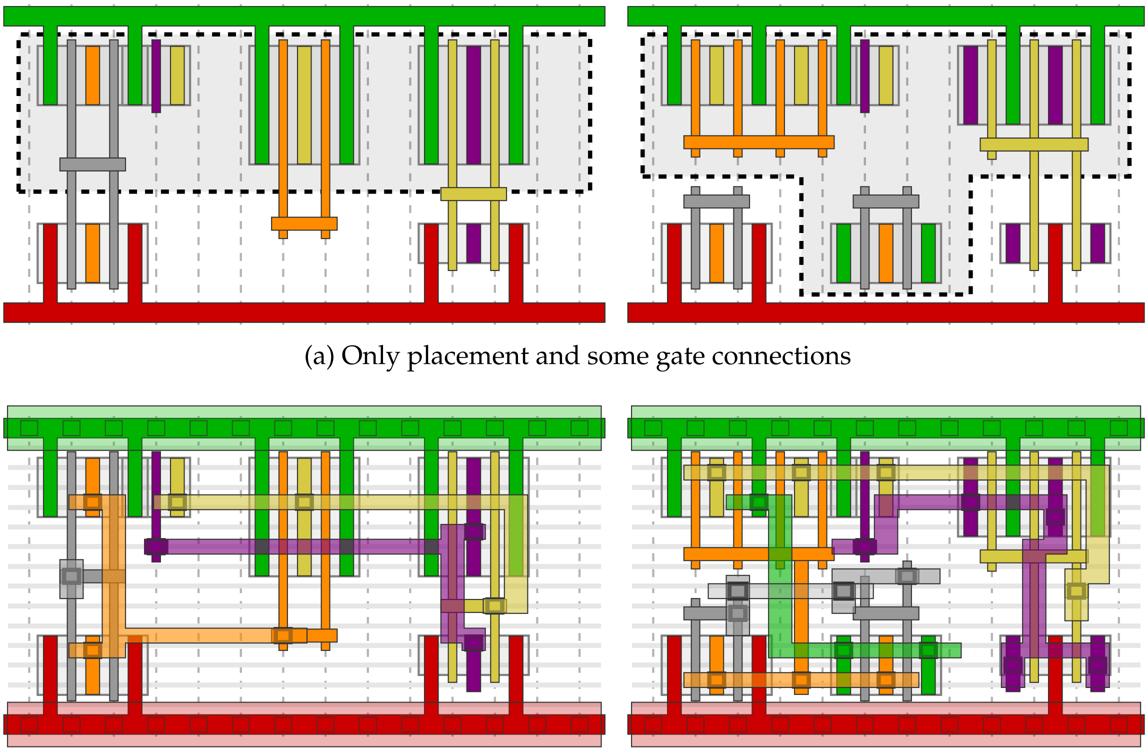


Figure 7.8: Ratio between the area covered by n-FETs (i.e. the number of fin/gate intersections) and the total area covered by FETs.



(a) Only placement and some gate connections

(b) Full layouts with routing

Figure 7.9: Default layout (left) and layout with a mixed circuit row. The gray regions indicate the p-FETs. The empty track on the right side of the default layout is due to the boundary condition.

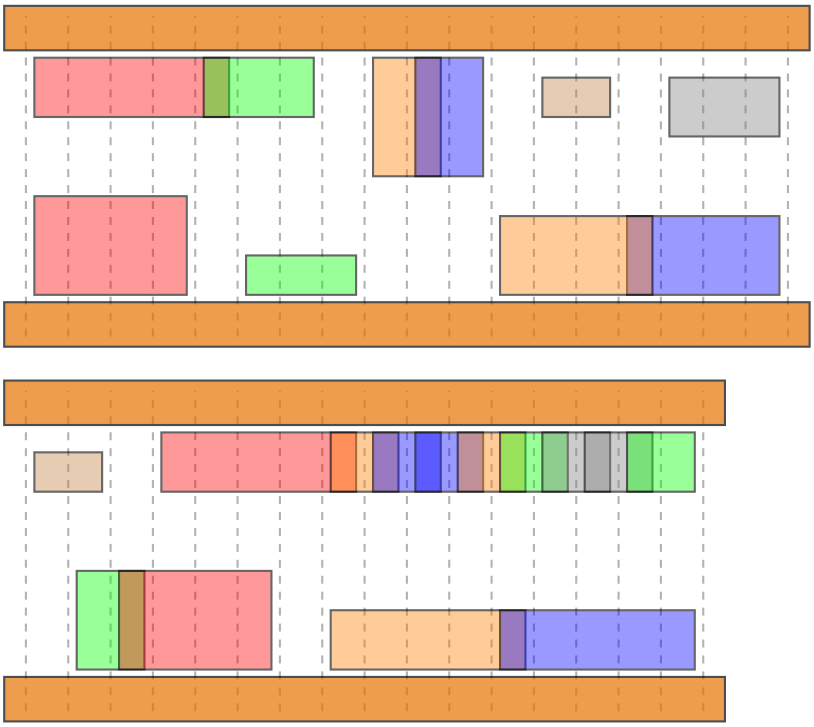


Figure 7.10: Placement of the same instance without and with multi-FET folding. By interleaving the two large transistors near the middle the amount of diffusion sharing can be increased, saving 2 tracks. Corresponding FETs in the two figures have the same color.

The two instances that do not encounter a timeout and, at the same time, are laid out with a mixed FET row do improve the target function. One can be placed with 1 fewer track, the other one even saves 2 tracks. The latter is depicted in Figure 7.9.

7.2.6 Folding of Multiple FETs

Another feature supported by BonnCell is the folding of multiple FETs, i.e. splitting pairs of large transistors with more than 1 finger into smaller single-finger transistors and arranging them in an interleaving fashion. The method is presented in detail in Section 5.2.3. To test this technique, all Clc cells were processed with default parameters and subsequently with multi-FET folding activated. Out of all 102 instances, BonnCell chose to apply folding on 20 cells. The effect on those 20 cells is documented in detail in Table 7.8.

The columns associated with the experiments where multi-FET folding was activated show the differences relative to the default run, and #*F* stands for the number of FET pairs that BonnCell split in order to arrange them in an interleaving pattern. In most cases, 1 pair of FETs was folded.

Table 7.8: Placement runtime, layout width, gate netlength, total netlength, and routability with default settings and with activated multi-FET folding. The results for the folding experiments show the differences to the default results, the column #*F* contains the number of FET pairs to which folding was applied. If no runtime is given, a timeout occurred.

	Default layout					Multi-FET folding					
Cell	Runt.	W	NL_g	NL	Rtb.[4]	#F	Runt.	W	NL_g	NL	Rtb.[4]
clc02	12.7	15	14	119	yes	3	—	-2	14	-26	yes
clc15	—	33	308	557	no	2	—	-4	-260	-351	no
clc31	3.0	19	54	200	yes	1	—	0	0	+17	yes
clc40	441.1	18	30	131	yes	2	2 492.9	-2	+22	-5	yes
clc41	0.0	12	16	68	yes	1	2.4	0	0	0	yes
clc49	0.0	9	6	55	yes	1	1.6	-1	+6	-6	yes
clc54	0.1	14	22	87	yes	1	211.1	0	0	0	yes
clc55	18.0	18	18	113	yes	2	—	0	0	+110	yes
clc58	0.1	13	16	73	yes	1	31.9	-1	+4	-6	yes
clc69	0.1	13	16	73	yes	1	29.7	-1	+4	-6	yes
clc70	0.1	14	22	111	yes	2	70.9	-1	+16	+3	no
clc71	0.0	9	8	65	yes	1	0.4	0	0	+10	yes
clc72	0.0	9	6	55	yes	1	1.3	-1	+6	-6	yes
clc87	0.0	13	14	73	yes	1	104.7	0	0	0	yes
clc90	0.0	12	14	60	yes	1	10.9	0	0	+4	yes
clc91	0.0	13	30	106	yes	1	2.6	0	0	0	yes
clc92	0.6	13	18	114	yes	1	35.9	0	0	0	yes
clc93	0.0	12	12	82	yes	1	6.6	0	0	0	yes
clc95	88.6	18	12	113	yes	1	382.4	0	0	0	yes
clc99	1.1	17	20	210	yes	1	6 109.3	0	+4	0	yes

The algorithm's target function, which essentially is a lexicographically ordered tuple of the layout width (W), the gate netlength (NL_g), and the total netlength (NL), changed on 13 out of 20 cells. From these 13 cells, 8 improved and 5 degraded. These degradations of the target functions can be explained by a timeout (as for clc15) or by the fact that the target function is modified in the multi-FET folding mode to favor the variants where folding is performed. This behavior was implemented because one assumes that when the user activates this mode interleaving patterns are desired for electrical reasons that are not captured by the standard measure.

The improvements of the target function are usually much more substantial than the degradations. All of the 8 improved cells could be laid out with a

[4] Indicates if the combinatorial router found a routing within 6 hours.

smaller number of tracks, while the degraded instances are mostly affected in the less relevant total netlength measure.

As most of the 20 instances represented in the table are routable with default settings and stay routable when activating the folding feature, the routability does not appear to be affected significantly. This is expected because the complexity of the netlist structure roughly stays the same. The drastic effect on the runtime can be attributed to the still experimental implementation of the feature that did not undergo optimization so far.

An example for the effect of multi-FET folding is depicted in Figure 7.10. Effects on the standard library will be discussed in Section 7.4.2. In the LAT test bed, almost all FETs are too small to be candidates for this technique, hence such tests are expendable.

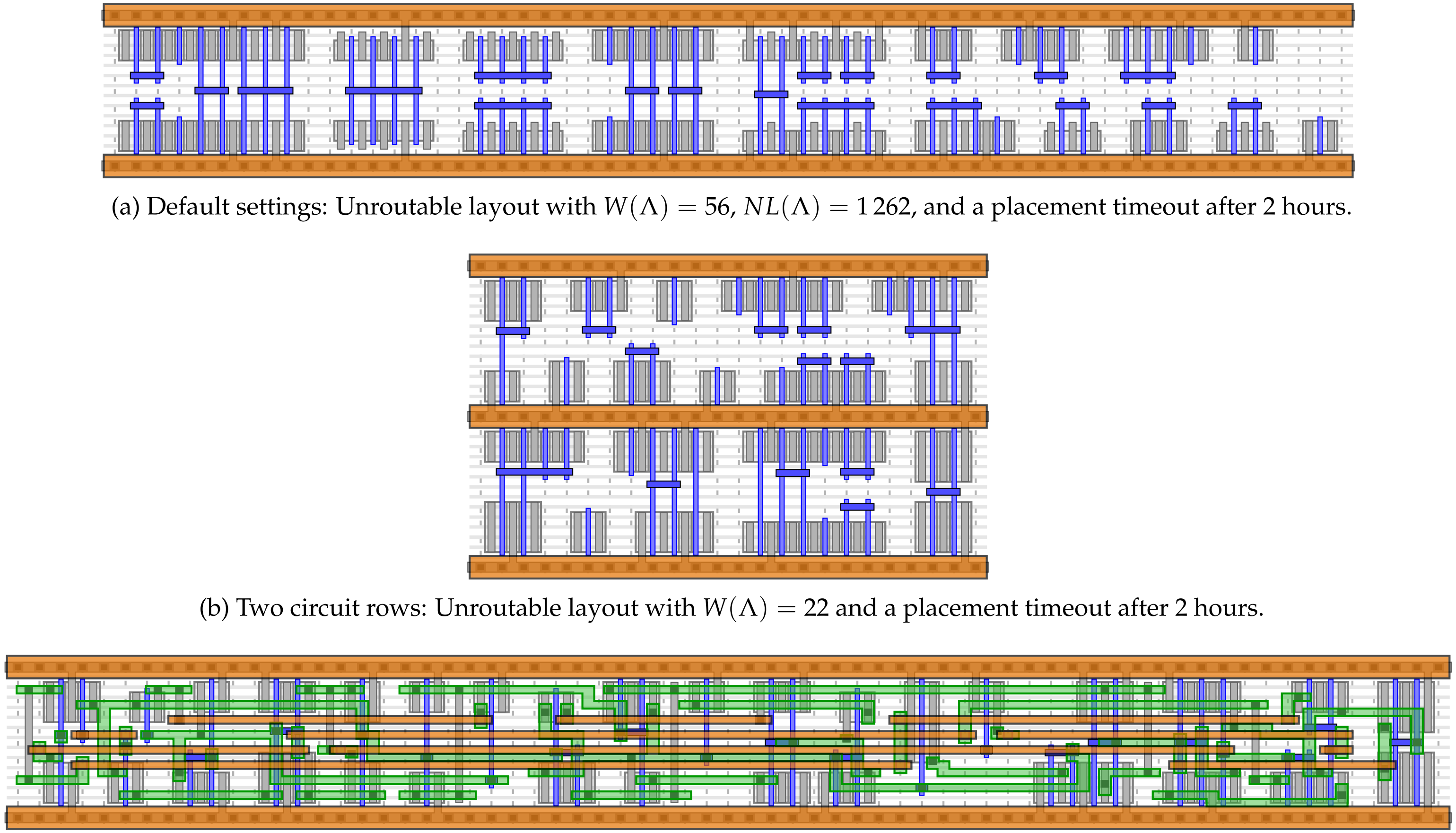
7.2.7 Large Cell Placement

After a timeout, especially in the first two phases, the generated layouts are often too bad to be routable. The mode introduced in Section 5.2.5 to handle very large cells promises to alleviate this problem by several means. Table 7.9 lists results of experiments conducted with the CLC test bed.

On the one hand, the reduction of total netlength is targeted rather than gate netlength, thereby reducing the amount of wiring that has to be packed into the cell outline. Table 7.9 shows that on CLC cells running into a timeout when using default settings, the total netlengh decreases by almost 12%. On the other hand, the sum of cell areas increases by about 8% on the same set of cells. Thus, by moving the focus away from the overall layout compactness, more routing area is available for less wiring. Consequently, out of the 26 cells with a timeout in the first two phases, 20 can be routed successfully after generating a layout with the alternative flow. With default settings, only 4 of these cells would be routable.

Although the greatly improved routability comes at the cost of larger cell outlines, the layouts are generated in a vastly reduced runtime. Only a single subcell out of 206 that were generated during the large cell heuristic encounters a timeout. All other BONNCELL runs finish much quicker than the maximum of 2 hours. To be precise, the 101 other cells produce a result after 81 seconds on average. The parameters involved in splitting the large instance into smaller instances are chosen sufficiently small so that almost all of the smaller instances can be processed in less than a second.

The same data for the LAT test bed is shown in Table 7.10. However, on these cells the heuristic for large cells was comparably unsuccessful as the default run. Even on cells which encounter an early timeout when run with default parameters, the netlengths increased and still none of these examples can be routed after `bigCellMode` is activated.



(a) Default settings: Unroutable layout with $W(\Lambda) = 56$, $NL(\Lambda) = 1\,262$, and a placement timeout after 2 hours.

(b) Two circuit rows: Unroutable layout with $W(\Lambda) = 22$ and a placement timeout after 2 hours.

(c) Large cell mode: Routable layout with $W(\Lambda) = 65$, $NL(\Lambda) = 786$, and a placement runtime of 0.2 seconds.

Figure 7.11: Three variants of a very complex cell. Routability could only be achieved with `bigCellMode` **activated.**

Table 7.9: Results for the CLC test bed when activating `bigCellMode`.

	Default	Large cells	Diff.
Total number of subcells	—	206	—
Number of subcell timeouts	—	1	—
Sum of placement runtimes (sec.)	299 561	15 423	-95%
Sum of layout widths	2 885	3 309	+15%
... for cells with phase 0/1 timeout	1 344	1 456	+8%
Sum of total netlengths	38 444	40 133	+4%
... for cells with phase 0/1 timeout	25 032	22 081	-12%
Successful routings	70	72	+3%
... for cells with phase 0/1 timeout	4	20	+400%

Table 7.10: Results for the LAT test bed when activating `bigCellMode`.

	Default	Large cells	Diff.
Total number of subcells	—	144	—
Number of subcell timeouts	—	2	—
Sum of placement runtimes (sec.)	113 557	7 236	-94%
Sum of layout widths	1 210	1 826	+51%
... for cells with phase 0/1 timeout	706	949	+34%
Sum of total netlengths	15 750	28 013	+78%
... for cells with phase 0/1 timeout	11 468	16 266	+42%
Successful routings	6	4	-33%
... for cells with phase 0/1 timeout	0	0	+0%

7.2.8 Routability

An essential aspect of the generated layouts is their routability. Arranging transistors as compact as possible is futile if their arrangement cannot be augmented by a feasible routing. Since it is not viable to actually conceive wires satisfying all design rules (about 700) for all generated cells, the term *routable* indicates, for the purpose of this chapter, if the combinatorial BONNCELL router finds an electrically correct wiring of all contacts in the design.

Table 7.11 condenses the routability results of all experiments presented in this chapter. It shows that while 70 out of 102 CLC cells can be routed using default settings, 22 additional cells have known feasible routings in at least one of the experiments. The ratio of successful routings is even larger for the simple library cells, yet rather low for the complicated LAT test bed. For all 198 LIB instances at least one routing was found.

The last two rows of the table show the number of cells which were routable

Table 7.11: Routable number of layouts for each of the presented experiments. "Total" counts the cells that were routable in at least one experiment, "proposed flow" summarizes the 4 experiments with `maxAddlTracks` set to 2, short fingers, 2 circuit rows, and `bigCellMode`.

	CLC	LIB	LAT
Default settings	70	182	6
Short fingers	76	190	7
1 additional track	69	193	7
2 additional track	69	196	6
3 additional track	69	196	6
2 circuit rows	70	178	5
3 circuit rows	57	135	5
Multi-FET folding	59	157	6
Large cell heuristic	72	194	4
Total	92	198	9
Total (relative)	90%	100%	25%
Proposed flow	90	198	8
Proposed flow (relative)	88%	100%	22%

in one of four variants, namely the ones where 2 additional tracks were allowed, with restricted finger lengths, with 2 circuit rows, and with activated large cell heuristic. The numbers demonstrate that these four variants suffice to find routable layouts for almost all cells for which the placement was routable in the remaining tests. We therefore propose to use BONNCELL in a flow where these four settings are executed in parallel before returning, among the routed layouts, the most compact one. According to the table, this flow would be able to generate layouts for nearly 90% of all runs.

When solutions are not routable, in many cases the reason seems to be that too much metal has to be routed across too few tracks. To support this claim, we define the average cut as follows: For a layout Λ let $NL^*(\Lambda)$, the *horizontal metal length*, the same as $NL(\Lambda)$ except that pins are ignored and net weights are considered to be 1. Then the average cut is defined as $C_{\text{avg}}(\Lambda) := \frac{NL^*(\Lambda)}{2W(\Lambda)}$. This number can be seen as a lower bound for the average number of horizontal wires that have to cross every x-coordinate within the cell.

horizontal metal length

average cut $C_{\text{avg}}(\Lambda)$

Figure 7.12 illustrates the average cut and the routability of every CLC layout that was generated with default settings. The figure shows that most solutions below a certain threshold of around 3.5 are routable, while most layouts exceeding this average cut cannot be routed. Hence, a strong correlation between these two properties appears to exist. For LAT instances, Figure 7.13 indicates the same behavior, just that the threshold seems to be smaller. In Section 7.3.2, we elaborate on the additional difficulties in this test bed.

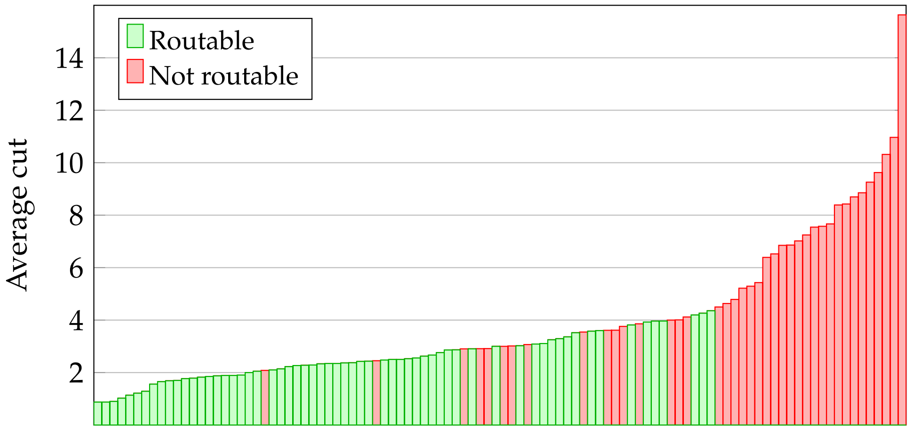


Figure 7.12: Average cut value $C_{\mathrm{avg}}(\Lambda)$ for all CLC layouts generated with default parameters. Green bars represent routable layouts.

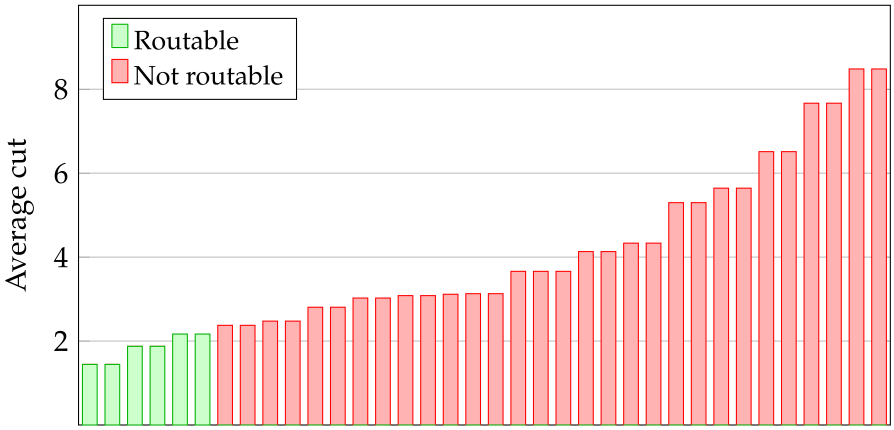


Figure 7.13: Average cut value $C_{\mathrm{avg}}(\Lambda)$ for all LAT layouts generated with default parameters. Green bars represent routable layouts.

7.3 Comparisons with Industry Layouts

Since there are, to the best of our knowledge, no other tools that are comparable to BONNCELL in their specification and generality, it is impossible to directly compare BONNCELL with other automatic layout generation programs. Moreover, there is neither a standardized data format for CMOS schematics nor publicly available test instances. It would in principle be possible to reproduce some isolated instances specified in the academic literature, but due to the strongly varying design rules it would not be possible to draw meaningful conclusions from such experiments.

The only direct comparison we can make is thus with the actual implementation of the cells that are currently in use at IBM. Those layouts have either been generated by scripts, which is the case for the simply structured standard cells in LIB, or were manually drawn by skilled designers, which is true for the complex memory elements in LAT. No finished layouts are available for CLC cells to compare against: On the one hand, the technology is still in the process of being conceived, so that finished layouts may not yet be available. On the other hand, BONNCELL is actively used in many cases, so purely manual constructed versions will never exist.

7.3.1 Simple Library Cells

Table 7.12 presents the widths of layouts generated by BONNCELL and the corresponding cells in the actual IBM library. For each functional group the table lists the sum of the original layouts' widths as well as the area gain in percent. In most cases the automatic layouts are more compact yet still routable. Experiments concerning the routability of the placements follow in Section 7.4.2. In the last row the sums are only taken over the "routable" solutions, i.e. layouts that could be successfully routed by the combinatorial router. Even for the XOR and XNOR circuits, which are the most complex instances in LIB and which could be realized with about a quarter fewer tracks, 26 out of 40 cells were routable despite their compactness. Counting only cells for which routings were found, the total area of the full library could be decreased by about 5%.
A total of 29 cells are slightly larger (each by 2 tracks) in the BONNCELL result than in the original IBM layout. The positive percentages in the column *BC*, which corresponds to a run with default parameters, are caused by these cells. The effect seems to contradict the claimed optimality of the tool. However, these 29 IBM circuits employ the folding of multiple transistors, which is not performed by default in BONNCELL. The column *BC+F* lists the results of runs where multi-FET folding was activated and the last column always counts the better of the two variants generated by BONNCELL. While some functional groups are not affected at all by the folding technique, like inverters, AND, and OR circuits, others benefit from it at the cost of increased runtime. An example is presented in Figure 7.14. The total runtime to process all 198 cells increased from ca. 21 hours to 188 hours, which includes many timeouts after 2 hours. As mentioned in Section 7.2.6, the feature's implementation is experimental and not optimized for runtime.

7.3.2 Complex Memory Elements

For the LAT circuits, finished layouts are available as well. In the experiments visualized in Figure 7.3, 23 out of 36 cells could be placed optimally or, in 2 cases, such that the algorithm successfully reached phase 2. The BONNCELL layouts of these cells amount to a total of 535 tracks, which is 13.1% less than the handcrafted IBM layouts (616 tracks). However, the caveat is that the

Table 7.12: Comparison of $W(\Lambda)$ between actual 14 nm library layouts and BONNCELL layouts for LIB test bed, without and with folding of multiple FETs. The last column takes the better of the two layouts. The last row only counts cells for which a routing was found in the layout without multi-FET folding.

Cell type	IBM	BC	BC+F	Best
Inverter	213	-4.7%	-4.7%	-4.7%
2-input AND	15	-0.0%	-0.0%	-0.0%
2-input OR	15	-0.0%	-0.0%	-0.0%
2-input NAND	397	-6.8%	-6.8%	-6.8%
3-input NAND	238	-1.3%	-2.5%	-2.5%
4-input NAND	127	-3.9%	-1.6%	-4.7%
2-input NOR	382	-7.1%	-7.1%	-7.1%
3-input NOR	217	-12.0%	-12.4%	-12.4%
2-1 AND-OR-Invert	314	+1.6%	-0.3%	-0.3%
2-2 AND-OR-Invert	413	+0.5%	-1.0%	-1.0%
2-1 OR-AND-Invert	314	+1.6%	-0.3%	-0.3%
2-2 OR-AND-Invert	413	+0.5%	-1.0%	-1.0%
2-input XNOR	160	-23.8%	-27.5%	-27.5%
2-input XNOR + buffer	200	-25.0%	-26.5%	-27.0 %
2-input XOR	160	-23.8%	-26.3%	-26.3%
2-input XOR + buffer	200	-25.0%	-27.5%	-28.0%
Sum	3 778	-6.9%	-8.0%	-8.2%
Sum (only routed)	3 596	-5.1%	-6.3%	-6.4%

very compact automatic layouts appear to be either very hard to route or completely unroutable: For just 6 of the 23 aforementioned cells a routing could be found by the combinatorial router. The question arises why these netlists seem to be much harder to route. Several reasons can be given:

- Nets in LAT instances connect an average number of 4.04 different FETs, much more than in the other test beds (2.98 for CLC and 2.22 for LIB). Consequently, the wires tend to form complicated tree-like structures rather than paths between two FETs.

- On average, the FETs are much smaller in the LAT test bed than in the other test beds (cf. Table 7.1). About 82% of all FETs in the memory elements have a total length of 4 or less, 64% even have the smallest possible length of 2. Hence, the cell area is fragmented by many small FETs rather than large multi-finger blocks with the same gate contact and alternating source/drain contacts. Taking the sum of layout widths given in Table 7.1, the number of layout tracks per transistor is only 0.92, whereas the ratio is 1.79 for CLC and 2.76 for LIB. This high FET

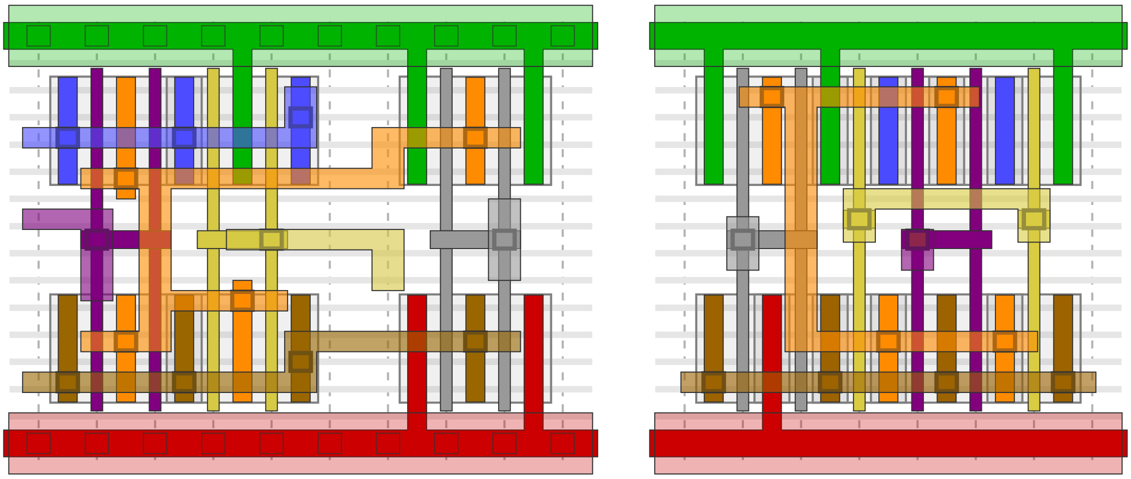


Figure 7.14: Optimal layouts with default parameters (left) and with folding of multiple FETs activated: $W_{\text{opt}}(\mathcal{F})$ decreases from 8 to 6. Corresponding nets in the two pictures are shown in the same color. Note that after the folding the blue contacts need not be connected.

density suggests that minimizing the cell width generates overly compact solutions and large values of `maxAddlTracks` would be required to generate routable versions.

- The average cut, as defined above, is very high on most LAT layouts. With default settings, most designs have an average cut significantly higher than 3.5, as shown in Figure 7.13.
- The IBM layouts of the LAT cells follow a fundamentally different strategy for the vertical FET positions $y(F)$. While BONNCELL places FET groups connected to the neighboring power strip as close to the power strip as possible and all other FET groups sufficiently close to the power such that the gate contacts can still be accessed from the cell center, the IBM placements arrange almost all FETs as close to the cell center as possible, meaning that gates, in many cases, have to be accessed from the other side. The two different placement strategies are illustrated in Figure 7.15.

  At the same time, the horizontal placement of the two transistor rows aims to horizontally align PC and M0 contacts of the same net alike, while our tool strongly prioritizes the PC contacts. Although both approaches have advantages, the BONNCELL routing method makes several assumptions related to the vertical placement and thus may fail if those assumptions are false.
- Finally, one has to keep in mind that the existence of a routing without design rule violations cannot be disproven by a failing combinatorial router. Even on the original IBM placements, for which such routings

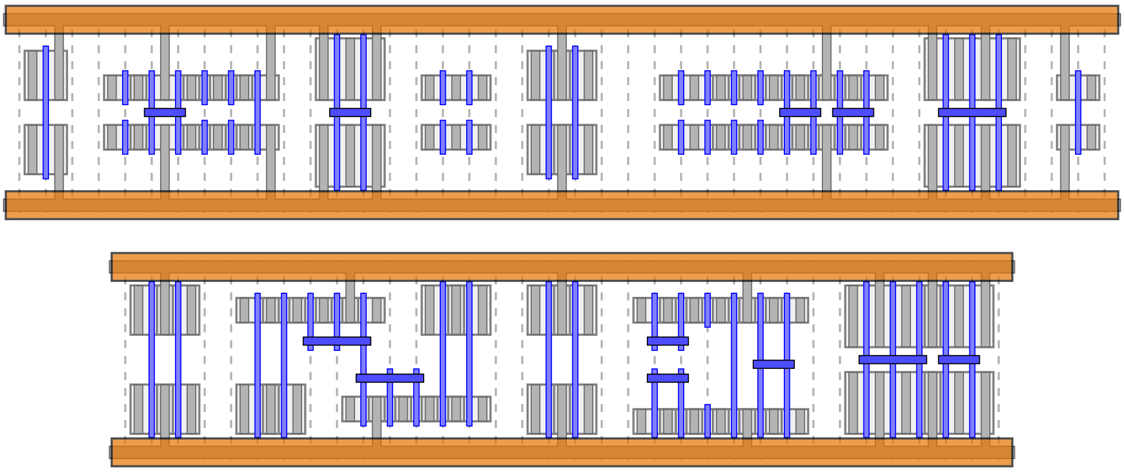


Figure 7.15: Different strategies for the vertical placement of IBM layout (top) and BONNCELL result (bottom).

demonstrably exist, BONNCELL routing fails on 10 out of 36 instances. Some of these failed attempts might be due to a timeout after 6 hours, but some detect unroutability—in the sense of the tool—after only a few seconds. In these cases, BONNCELL overconstrains the problem to reduce the search space (e.g. by regarding only a subgrid of the actual technology's grid), hoping that these additional constraints do not make routable layouts unroutable. Unfortunately, this apparently happens.

7.4 Application in the Industry

Thanks to our close cooperation with IBM during the development of BONNCELL, the toolchain has not just been used to process isolated test cases, but has been frequently applied during major VLSI projects in the industry. In this section, we elaborate on two different applications for which it was employed in practice.

7.4.1 Array Design

The first larger unit for which BONNCELL has been used extensively was an SRAM block—a 2-dimensional arrangement of memory elements together with a set of control logic—that was used several times in each core of a major server processor. The design work on the unit encompassed many different steps, one of them being the layout of transistor-level netlists. In some cases, only size estimations were obtained by a placement run so that the floorplanning on a higher level could proceed, in other cases full routings were generated.

Based on the experience with earlier comparable projects, it was estimated that after 3 months the first important milestone would be reached. At this point, the unit's timing properties can be determined for the first time on the basis of individual FETs. Instead of 3 months, the milestone was reached after only 7 weeks, mainly due to the automation of transistor-level layout:

> "Usually [such units] are completed relatively late. We finished [this unit] so quickly because we never had to wait for transistor-level layouts. Whether routed or just placed, we received immediate results with which we could continue work."
>
> Raphael Polig, IBM [Pol12]

7.4.2 Library Design

For basic logic functions in the standard library, including the ones listed in Table 7.2, scripts are available to generate, depending on parameters like FET sizes, full layouts of the corresponding netlists. This is possible because the complexity of these layouts is low and good solutions can be obtained by following a relatively small number of rules.

However, BONNCELL can still be applied profitably during the concept phase of a new standard cell library. Although the tool does not compute immediately manufacturable layouts, it does provide a fast way to evaluate properties of the technology in an early stage of development when some aspects of the cell image have not yet been decided upon. In this section, we outline one scenario that has been practically implemented during the development of the 14 nm technology at IBM.

This scenario is concerned with the height of the cells: In general, it is desirable to have a library of cells which all have the same vertical extension. By this method, all cells can be arranged in a uniform circuit row structure without losses between cells of different heights. Hence, a decision has to be made early in the process on the number of horizontal tracks contained in each cell. It is easy to see that a narrow cell outline with few tracks is favorable for netlists with small FETs (otherwise only a small fraction of the chip area could be covered with transistors), while big FETs tend to fit better into images with a large number of tracks. But the complete library contains extremes on both ends of that spectrum, so the choice of the common cell height is not a trivial one.

As manual experiments, which basically involve drawing rectilinear shapes into a specialized CAD software, are time-consuming, an automatic solution helps to generate evidence in favor of one choice or the other. Table 7.13 illustrates the results on three experiments, each involving all 198 cells from the LIB test bed. For these experiments, each cell has been configured with a height of 10, 14, and 18 horizontal tracks.

Table 7.13: Evaluation of all 198 LIB cells for vertical cell sizes of 10 tracks, 14 tracks, and 18 tracks.

	Height of circuit row		
	10 tracks	14 tracks	18 tracks
Number of optimal placements	192	198	198
Placement timeouts in phase 0/1/2	0/0/6	0/0/0	0/0/0
Sum of placement runtimes (sec.)	73 881	445	1
Sum of cell areas (normalized)	100.0%	100.3%	115.4%
Successful routings	182	191	194
Sum of routing runtimes (sec.)[5]	257 782	52 456	49 185
Total number of used M2 tracks[6]	11	9	0

The results are very helpful for the decision on the future cell image. On the one hand, they show that with increasing circuit row height the netlists become "easier" to lay out: The 10 track image requires by far the longest runtime both for placement and routing, and the only timeouts occur in these experiments. In fact, by using 14 and 18 tracks all cells can be placed optimally and the combinatorial router is successful on more than 96% of the cells. With the largest image, all 198 cells can even be placed optimally in a total runtime of 1.4 seconds. The reason is that in this case almost all FETs can be configured with their minimum finger numbers and hardly any gaps occur in the final layouts, so that very good lower bounds can be computed during the algorithms. All experiments—nearly 600 full layouts—can be computed in about 5 hours on a computer with 24 cores.

The data also shows that M2, which should be employed as sparingly as possible, is not used at all in cells with 18 tracks. On the downside, these cells require about 15% more area than the other variants. The conclusions that can be drawn from these facts depend on the the priorities of the given VLSI project, other experiments, and further technological constraints. But still the availability of an automatic layout tool like BONNCELL can—and did—provide valuable, profound data aiding in central decisions during the early conceptual phase of a new technology.

[5] Only combinatorial router.

[6] Only the 173 layouts are considered for which routings were successfully found in all three experiments.

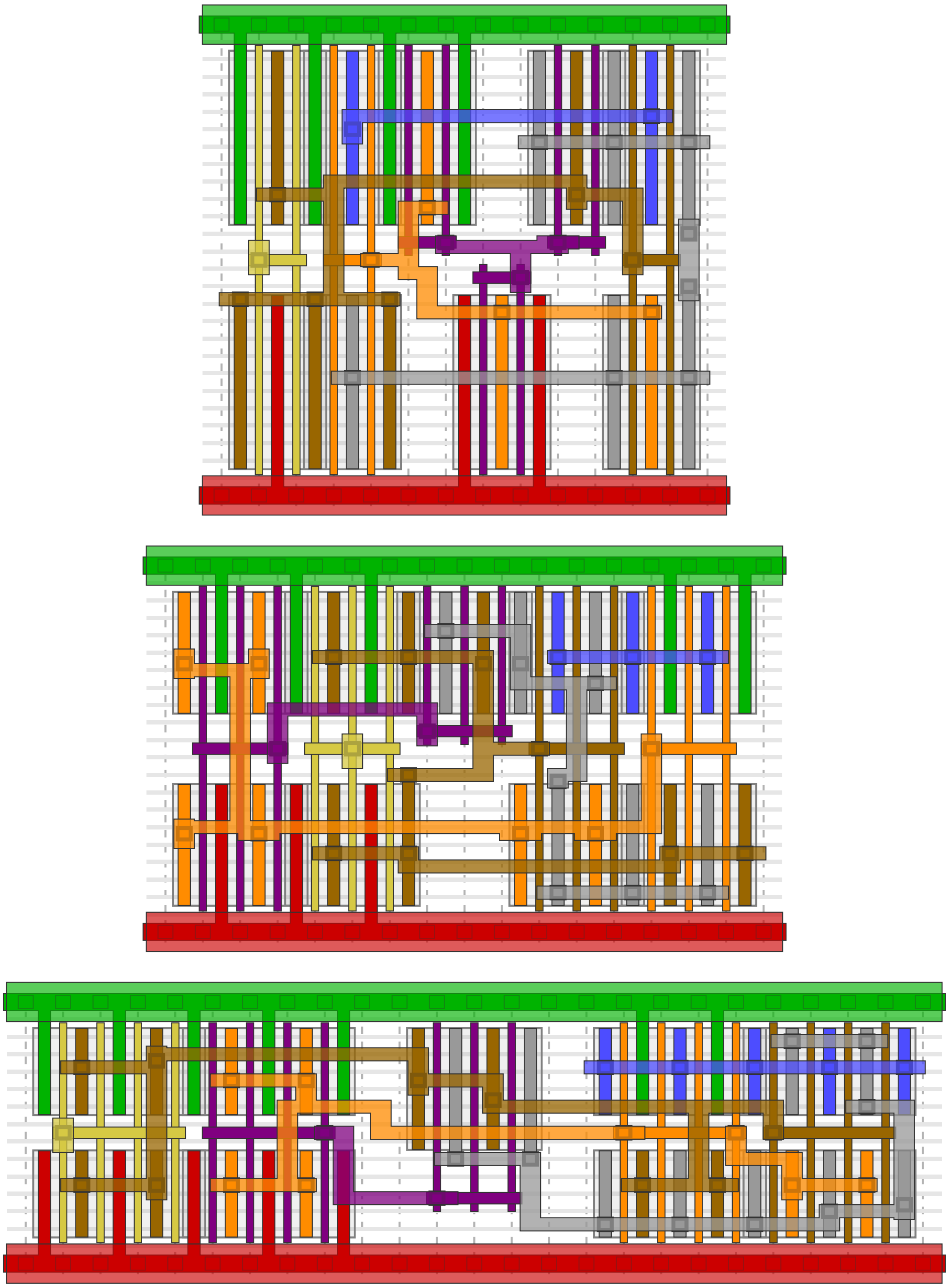


Figure 7.16: Two-input NOR laid out with 18, 14, and 10 vertical tracks.

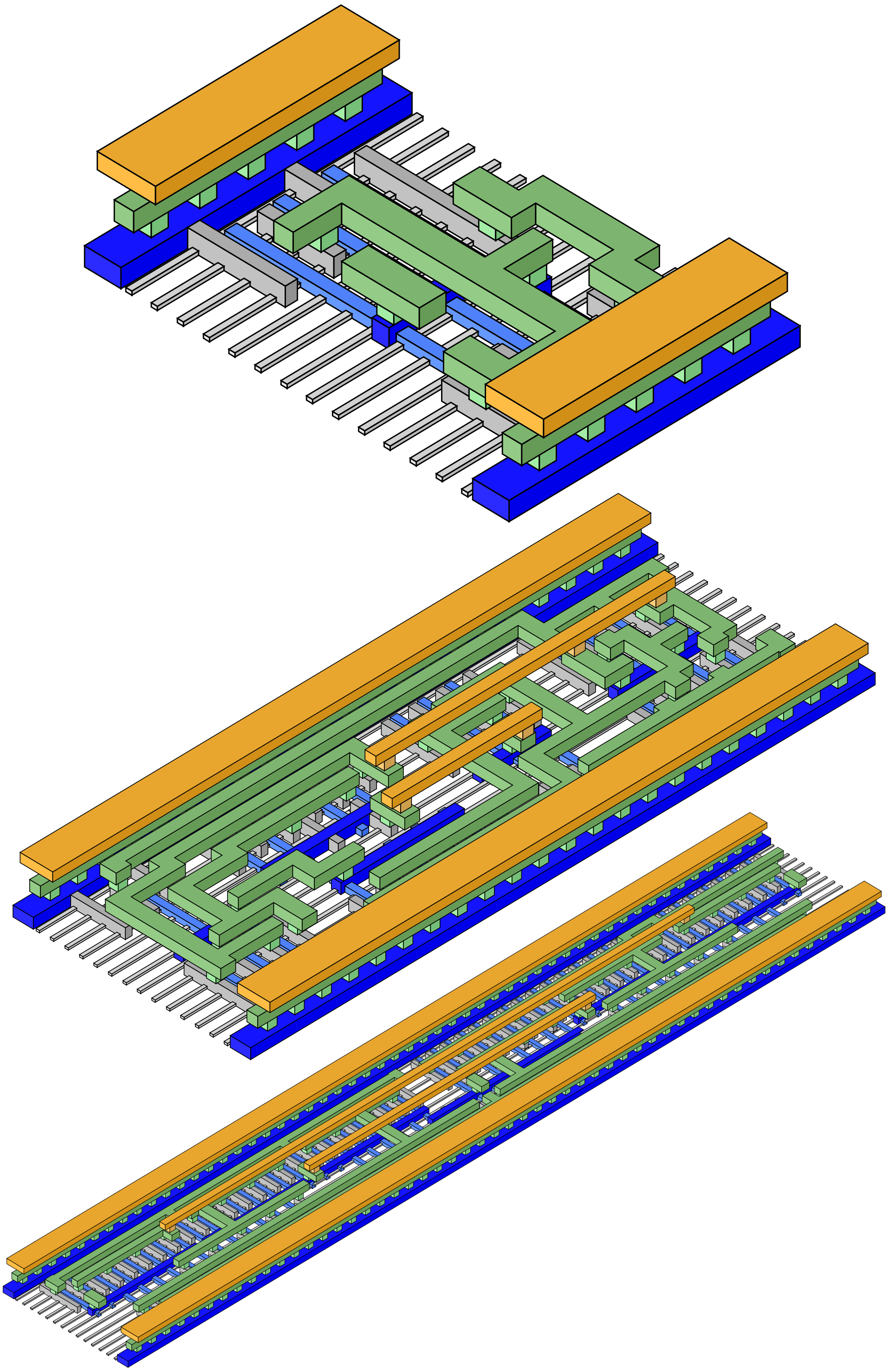


Figure 7.17: 3-dimensional illustrations of a small, medium, and large BonnCell layout.

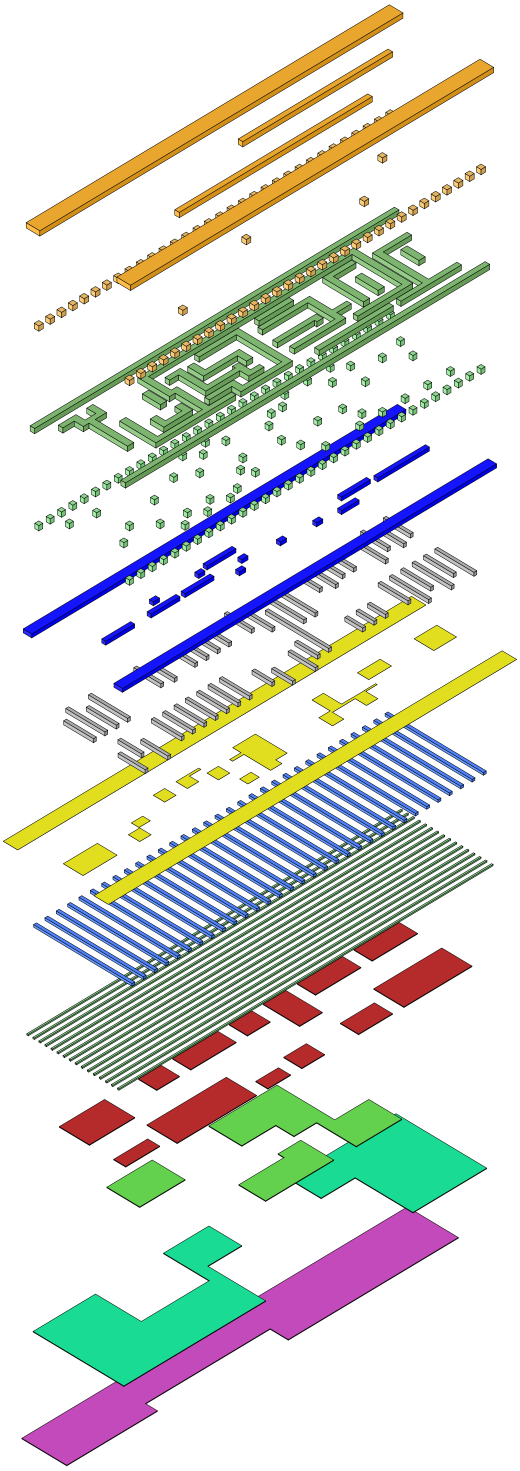


Figure 7.18: Example of the shapes in a BonnCell layout. We generate shapes on a total of 16 layers, 3 of which are not used here.

Chapter 8

Conclusions

> We're all stories, in the end. Just make it a good one, eh?
>
> The Doctor, *Doctor Who*

In this dissertation, we have presented and analyzed algorithms to automatically generate transistor-level layouts as well as the associated software system BONNCELL. To conclude, we summarize our results and provide an outlook on the future development.

8.1 Summary

Our main contribution is the development of an algorithmic framework and an associated software system, BONNCELL, that manages to successfully lay out a wide range of real-life transistor-level netlists. Out of all 336 cells that were provided by our industry partner IBM, all of which are in use in the current 14 nm technology node, 276 can be placed such that the target function is provably minimized. For a total of 299 cells, layouts could be found that were routable by the tool's own routing engine.

While a large number of tools have been presented in academic and technical literature, BONNCELL surpasses all published systems in the number of supported features. Features implemented in our system include:

- Generation of layouts with two transistor rows where FETs may be configured with multiple fingers, swapped, and reordered.
- No restrictions are imposed on the netlist structure.
- Provably optimal solutions with dynamic placement and dynamic folding in the terminology of [GH98].
- Support for complex rules affecting the spatial relations of different FETs based on type, Vt levels, FET configuration and connected nets.

- Pairs of FETs can be folded and arranged in an interleaving pattern.
- n-FETs and p-FETs may mix in a single FET row.
- Cells can be generated that span multiple circuit rows.
- Complex cells can be processed with a heuristic approach that splits instances into smaller sub-instances, thereby generating less compact layouts with an improved routability.
- Cell layouts can be flexibly prescribed with a large number of input parameters. For example, all aspects of the power supply can be defined arbitrarily.
- Support for 22 nm (old version), 14 nm, and 10 nm (initial implementation) cells.
- Complete workflow with placement, routing and post-optimization.
- The algorithm's behavior can be controlled with many user-definable parameters.

An integration of BONNCELL into the chip design software in use at IBM, including data conversion and an easy-to-use GUI, was contributed by our cooperation partners at IBM. Thanks to the simple availability and high flexibility of the tool, BONNCELL is in wide use within the company's groups that are concerned with transistor-level layout.

Beyond the processing of isolated test cases, two large-scale examples for applications of BONNCELL in the industry have been presented in Section 7.4: On the one hand the initial design phase of a large SRAM unit required only half of the expected 3 month period due to the frequent use of our tool, on the other hand BONNCELL could provide valuable input aiding central decisions in the early concept phase of a new technology generation.

The algorithms underlying the layout program have been presented and analyzed in Chapters 4 and 5. This includes the generalization of previous graph-theoretic approaches to formulate the transistor row placement problem, which implies polynomial algorithms and hardness results of several problem variants, and a discussion of the NP-hardness of the corresponding netlength optimization problem.

The presented algorithms, building upon a simple branch and bound core, constitute a flexible framework for the computation of cell layouts. Even though many of them display an exponential runtime behavior, powerful bounding techniques as well as suitable combination and nesting of the individual methods result in a viable toolset. Moreover, due to the highly flexible concept, technological changes or new rules can be incorporated easily into the existing framework.

8.2 Future Development

In the future, more work on the subject can be done both in terms of theoretical exploration as well as practical improvements of BONNCELL. Regarding the theoretical aspects, the most notable open question remains the hardness of the TRANSISTOR ROW PLACEMENT PROBLEM for the distance function $d^{0/2}$. This distance function is close to the actual technology's behavior and, at the same time, lies between functions for which the problem is easily solvable and functions for which it becomes NP-hard.

Further room for improvement lies in the consideration of other target functions than the netlength. Most notably, electrical requirements make it necessary to use thicker wires for certain segments of particular nets. The old 22 nm version of BONNCELL included a post-optimization method that increased the thickness of such segments, but the recent implementation for current technologies does not yet have this capability. It is also desirable to directly address this issue during the routing phase rather than as a post-optimization.

As for some cells, especially from the LAT group, BONNCELL could not generate routable layouts, this remains an aspect that can be improved. One way to address the topic is to change the vertical arrangement of transistors. While the horizontal locations are subject to most of the optimization steps, the y-coordinates are determined by a heuristic based on few simple rules. However, some experiments, mostly the observations related to the LAT instances, point to other, more successful strategies at least on appropriately structured cells.

The technique for the placement of very large cells also bears much potential. The already successful method presented in Section 5.2.5 has not yet been tuned a lot, although many parameters like the size of the subcells or various aspects of the rough placement heuristics are involved. Moreover, it may prove worthwhile to reduce the maximum cut rather than the average cut or to modify the target function. It is also possible to improve the way by which finished subcells are combined, e.g. by swapping or rearranging subcells as clusters of FETs.

Until now, all of the presented methods are implemented to run in a single thread. While a parallelization of the algorithm can only achieve speedup factors linear in the number of processor cores, which hardly compensates for the exponential runtime, this scheme may be used to improve the routability: If several layout variants are generated and routed in parallel, the probability that one of the variants leads to a success is larger (cf. Section 7.2.8).

Glossary

3D transistor → FinFET

Average cut Measure for the amount of metal that has to pass every x-coordinate on average. → p. 129

Buffer Electronic circuit realizing the identity $x_{\text{out}} \leftarrow x_{\text{in}}$.

Canonical layout Legal layout with a set of additional properties. → p. 66

Cell Functional implementation of a transistor-level electronic circuit, in our context using the CMOS scheme. → p. 19

Chip → Integrated circuit

Circuit row Region between two neighboring power rails that usually prescribes the height of the generated cell layouts. → p. 24

Closed walk Walk in a graph that starts and ends in the same vertex. → p. 15

CMOS (Complementary Metal-oxide-semiconductor) Implementation paradigm for integrated circuits using n-FETs and p-FETs, which are often arranged in pairs with the same gate connection. → p. 19

Compaction → Layout compaction

Configuration (of a FET) Tuple specifying how a transistor is implemented. Contains the number of fingers, the length of the fingers, and the swap status. → p. 50

Configuration graph Graph modeling the the connectivity of a set of FETs and their configurations. → p. 53

Covering number Minimum size of a walk cover. → p. 54

Delayed Binding Algorithmic technique that does not assume the topology of the netlist to be fixed [Mad89].

Diffusion sharing Technique that allows neighboring FETs to overlap in the sense that one source or drain contact is used jointly by both transistors. → p. 21

Distance function A specification of the required distance between pairs of FETs. → p. 51

Drain contact One of the contacts of a transistor. → p. 16

Dual netlist A netlist topology is considered dual if the configuration graph of the n-FETs is the series-parallel dual of the configuration graph of the p-FETs (when all finger numbers are considered to be 1).

Eulerian graph Graph that possesses a closed Eulerian walk. → p. 15

Eulerian walk Walk in a graph that covers all of its edges. → p. 15

FET (Field effect transistor) Concrete technical realization of a transistor on an integrated circuit. Most transistors on modern chips are implemented as FETs. → pp. 16, 49

FET class Set of FETs for which all elements have the same Vt level and the same type. → p. 59

FinFET Implementation of a FET that uses the third dimension to reduce the FET's area requirement. → p. 17

Finger One of the gate contacts of a folded transistor. All fingers of a FET have to be connected electrically. → p. 22

Folding When a large transistor is folded it is replaced by several smaller transistors which are connected in parallel. This technique allows to control the aspect ratio of otherwise elongated transistors. → p. 21

Gate contact The switching contact of a transistor. → p. 16

Gate netlength Restricted netlength definition that only considers gate contacts. → p. 53

GND One of the power potentials on a chip (ground), represents the logic 0 in our context. → p. 24

Half-track Unit in which the netlengths are measured: Half of the distance between two neighboring gate centers.

IC → Integrated circuit

Integrated circuit An electronic circuit that is realized by employing a layer of semiconducting material. Integrated circuits are commonly known as chips. → p. 16

Inverter Electronic circuit realizing the negation $x_{\text{out}} \leftarrow \neg x_{\text{in}}$. → p. 20

ILP (Integer linear program) Optimization problem in which a target function has to be optimized in a solution space specified by a set of linear inequalities. → [KV13]

Latch Electronic circuit that stores information.

Layer stack Specification of the layers of metal (and other materials) on which shapes can be manufactured for a given VLSI technology. → p. 23

Layout (of a FET) A pair that contains the configuration and the placement of a FET. → pp. 50, 70

Layout compaction The process that transforms a symbolic layout and technology-dependent rules into a set of geometric shapes that require the least amount of area.

Layout extension The process of completing a partial layout to a complete 2-dimensional layout. → p. 70

Layout quality → Quality

Layout width → Width

Leaf cell → Cell

Leg Sometimes used in the literature to denote the fingers of a folded FET.

Legal layout A layout of a FET or a set of FETs that does not violate any design rules. → p. 51

Legality oracle Algorithm that decides for a given partial layout if it can be extended to a legal layout. → p. 70

Length (of a FET) The required length of the FET's gate contact. In the 14 nm technology, the parameter is given as the number of fin/gate intersections a transistor must have. → p. 49

MIP (Mixed integer linear program) Optimization problem in which a target function has to be optimized in a solution space specified by a set of linear inequalities. → [KV13]

MOS (Metal-oxide-semiconductor) One possible technological realization of an integrated circuit, usually involving a layer of SiO_2 above a silicon substrate.

MOSFET (Metal-oxide-semiconductor FET) → MOS, FET

Multi-FET folding Technique in which multiple FETs are split into smaller FETs and arranged with an interleaving pattern. → p. 22

n-FET N-doted FET, forms a conducting channel if the V_{DD} power level is applied to the gate. → p. 16

Net Refers either to a set of contacts in the netlist that have to be connected, or to the metalized region in the layout that realizes such a connection. → p. 49

Net weight Factor for the contribution of a net to the netlength. Can be used to put a larger emphasis on the length of particular nets. → p. 49

Netlength Lower bound for the horizontal length of the metalized area that realizes a net. → pp. 53, 70

Netlist Specification of a cell's topology, including the FETs and the nets. → p. 50

p-FET P-doted FET, forms a conducting channel if the GND power level is applied to the gate. → p. 16

Pin Distinguished point on a wire on which the net is accessed from outside the cell. → p. 79

Placement (of a FET) The location of a transistor given as an x- and a y-coordinate. → pp. 50, 69

Quality A multi-dimensional measure for the expected quality of a layout. → p. 71

Routing Collection of all wires that realize a given net, or all nets respectively.

Routing grid 3-dimensional grid graph on which most of the routing algorithms operate → p. 98

Semiconductor A material that can act both as a conducting material as well as an insulator.

Semi-Eulerian graph Graph that possesses a Eulerian walk. → p. 15

Series-parallel dual The planar dual of an embedding of $G + \{s,t\}$, where G is a series parallel graph and $s,t \in V(G)$ its distinguished vertices.

Series-parallel graph A graph that can be constructed from a single edge by iteratively duplicating and subdividing edges. → [KV13]

Simple distance function A class of distance functions that obey several restricting conditions. → p. 51

Slowdown factor Ratio between the runtime of a particular experiment and the runtime when using default settings. → p. 114

Source contact One of the contacts of a transistor. → p. 16

Stacked device → Folding

Steiner tree Connected acyclic subgraph of a graph that contains a given set of vertices (the "terminals"). → [KV13]

Swapped FET When FETs are swapped, their source and drain contacts exchange their places. → p. 21

Symbolic layout A technology-independent geometric representation of a cell that prescribes spatial relations of transistors and wires but no exact coordinates. The latter are often determined in a layout compaction step.

Symmetric netlist A netlist topology is considered symmetric if there is a bijection between the inputs of the n-FETs and the p-FETs. This property is frequently exploited in literature in order to determine pairs of n-FETs and p-FETs in algorithms that arrange arrays of such pairs.

Total netlength → Netlength

Track Denotes either the grid lines on which the gate contacts are routed or, in the context of M1/M2 metal, the grid lines on which horizontal M1/M2 wires can be routed.

Transistor An electronic switch without mechanical parts with essentially three contacts: source, drain, and gate. The voltage applied to the gate contact controls if a current can flow from the source to the drain contact. → p. 16

Transistor Row Netlength Minimization Problem Formal definition of the problem to arrange a set of transistors such that the netlength is minimized among all minimum-width layouts. → p. 56

Transistor Row Placement Problem Formal definition of the problem to arrange a set of transistors in a compact 1-dimensional row. → p. 53

Tri-gate transistor → FinFET

V_{DD} One of the power potentials on a chip, represents the logic 1 in our context. → p. 24

VeSFETs (Vertical Slit FETs) Alternative geometry for a FET. → p. 46

Via Vertical connection between wires on two adjacent metal layers. → pp. 24, 98

Virtual routing grid → Routing grid

VLSI (Very-large-scale Integration) Laying out a "very large" number of transistors as an integrated circuit. The term is in use since the 1970s when several thousand transistors had to be handled.

Vt level (Voltage threshold) A property of FETs that influences their power demand as well as their performance. → p. 17

Walk Edge progression in a graph without duplicate edges. → p. 15

Walk cover Set of edge-disjoint walks covering a graph. → p. 54

Width (of a layout) Horizontal space occupied by a given layout. → pp. 52, 70

Wire Geometric implementation of a net within a cell. → p. 98

Worst cut Largest number of nets that have to cross the boundary between two neighboring circuit rows. → p. 87

Copyright Permissions

Figure 1.3a: *New POWER7+ microprocessor* by ibmphoto24 is licensed under CC BY-NC-ND 2.0 (`https://creativecommons.org/licenses/by-nc-nd/2.0`), reproduced without changes.

Figure 1.3b: *SecureDigital Micro memory card drawing* by Alban75 is licensed under CC BY-SA 3.0 (`https://creativecommons.org/licenses/by-sa/3.0`), reproduced without changes.

Figure 1.1: Courtesy of Texas Instruments. Reprinted in compliance with the "additional permission" granted on `http://www.ti.com/corp/docs/kilbyctr/downloadphotos.shtml`.

Figure 2.1: Photo released to the public domain by NASA.

Figure 2.2: From *Intel Announces New 22nm 3D Tri-gate Transistors*, 2011, © Intel Corporation. Reprinted by permission of Intel Copyright Permission Department.

Figure 2.3: From *Intel Announces New 22nm 3D Tri-gate Transistors*, 2011, © Intel Corporation. Reprinted by permission of Intel Copyright Permission Department.

IBM and **CPLEX** are trademarks of IBM Corporation.

Bibliography

[AMW82] D. Aubert, J. J. Monbaron, and J. P. Wattenhofer. Computer Aided Layout of Distributed CMOS Static Decoders. In *8th European Solid-State Circuits Conference*, ESSCIRC 1982, pages 90–93. IEEE, September 1982.

[AT78] Tetsuo Asano and Kokichi Tanaka. A Gate Placement Algorithm for One-Dimensional Arrays. *Journal of Information Processing*, 1(1):47–52, April 1978.

[Bak11] R. Jacob Baker. *CMOS: Circuit Design, Layout, and Simulation*. IEEE Series on Microelectronic Systems. Wiley, 2011.

[BBC$^+$85] R. K. Brayton, N. L. Brenner, C. L. Chen, G. DeMicheli, McMullen C. T., and R. H. J. M. Otten. The YORKTOWN Silicon Compiler. In *Proceedings of the 34th IEEE International Symposium on Circuits and Systems*, ISCAS 1985, pages 391–394. IEEE Press, 1985.

[Bea84] John E. Beasley. An Algorithm for the Steiner Problem in Graphs. *Networks*, 14(1):147–159, 1984.

[Ber01] Krzysztof S. Berezowski. Transistor Chaining with Integrated Dynamic Folding for 1-D Leaf Cell Synthesis. In *Proceedings of the 2001 Euromicro Symposium on Digital Systems Design*, pages 422–429. IEEE, 2001.

[BR96] Bulent Basaran and Rob A. Rutenbar. An O(N) Algorithm for Transistor Stacking with Performance Constraints. In *Proceedings of the 33rd Design Automation Conference*, DAC 1996, pages 221–226. ACM, 1996.

[BYFPW89] R. Bar-Yehuda, J.A. Feldman, Ron Y. Pinter, and S. Wimer. Depth-first-search and Dynamic Programming Algorithms for Efficient CMOS Cell Generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):737–743, July 1989.

[CC89] C. C. Chen and S.-L. Chow. The Layout Synthesizer: An Automatic Netlist-to-Layout System. In *Proceedings of the 26th Design Automation Conference*, DAC 1989, pages 232–238. ACM, 1989.

[CCM95] Bradley S. Carlson, C. Y. Roger Chen, and Dikran S. Meliksetian. Dual Eulerian Properties of Plane Multigraphs. *SIAM Journal on Discrete Mathematics*, 8(1):33–50, February 1995.

[CHH99] Zhi-Zhong Chen, Xin He, and Chun-Hsi Huang. Finding Double Euler Trails of Planar Graphs in Linear Time. In *40th Annual Symposium on Foundations of Computer Science*, pages 319–329, 1999.

[CHR91] Sreejit Chakravarty, Xin He, and SS Ravi. Minimum Area Layout of Series-parallel Transistor Networks is NP-hard. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(7):943–949, 1991.

[Coo71] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, STOC 1971, pages 151–158. ACM, 1971.

[Cor13] Jordi Cortadella. Area-Optimal Transistor Folding for 1-D Gridded Cell Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(11):1708–1721, November 2013.

[EMP73] W. L. Engl, D. A. Mlynski, and P. Pernards. Computer-Aided Topological Design for Integrated Circuits. *IEEE Transactions on Circuit Theory*, 20(6):717–725, 1973.

[ET98] Tali Eilam-Tzoreff. The Disjoint Shortest Paths Problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998.

[Fel76] A. Feller. Automatic layout of low-cost quick-turnaround random-logic custom LSI devices. In *Proceedings of the 13th Design Automation Conference*, DAC 1976, pages 79–85. ACM, 1976.

[FSA95] Masahiro Fukui, Noriko Shinomiya, and Toshiro Akino. A New Layout Synthesis for Leaf Cell Design. In *Proceedings of the 10th Asia and South Pacific Design Automation Conference*, ASP-DAC 1995. ACM, 1995.

[GH96] Avaneendra Gupta and John P. Hayes. Width Minimization of Two-dimensional CMOS Cells Using Integer Programming. In *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD 1996, pages 660–667. IEEE Computer Society, 1996.

[GH97a] Avaneendra Gupta and John P. Hayes. A Hierarchical Technique for Minimum-Width Layout of Two-Dimensional CMOS Cells. In *Proceedings of the 10th International Conference on VLSI Design: VLSI in Multimedia Applications*, VLSID 1997, pages 15–20. IEEE Computer Society, 1997.

[GH97b] Avaneendra Gupta and John P. Hayes. CLIP: An Optimizing Layout Generator for Two-dimensional CMOS Cells. In *Proceedings of the 34th Design Automation Conference*, DAC 1997, pages 452–455. ACM, 1997.

[GH98] Avaneendra Gupta and John P. Hayes. Optimal 2-D Cell Layout with Integrated Transistor Folding. In *IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers*, ICCAD 1988, pages 128–135. ACM, 1998.

[GMD$^+$97] Mohan Guruswamy, Robert L. Maziasz, Daniel Dulitz, Srilata Raman, Venkat Chiluvuri, Andrea Fernandez, and Larry G. Jones. CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries. In *Proceedings of the 34th Design Automation Conference*, DAC 1997, pages 327–332. ACM, 1997.

[GML89] U. Gatti, F. Maloberti, and V. Liberali. Full Stacked Layout of Analogue Cells. In *Proceedings of the 38th IEEE International Symposium on Circuits and Systems*, ISCAS 1989, pages 1123–1126, 1989.

[GMW97] Martin Grötschel, Alexander Martin, and Robert Weismantel. The Steiner Tree Packing Problem in VLSI Design. *Mathematical Programming*, 78(2):265–281, 1997.

[GN76] Dave Gibson and Scott Nance. SLIC – Symbolic Layout of Integrated Circuits. In *Proceedings of the 13th Design Automation Conference*, DAC 1976, pages 434–440. ACM, 1976.

[GTH96] Avaneendra Gupta, Siang-Chun The, and John P. Hayes. XPRESS: A Cell Layout Generator with Integrated Transistor Folding. In *Proceedings of the European Design and Test Conference (ED&TC)*, pages 393–400. IEEE Press, March 1996.

[HF87] Hansruedi Heeb and Wolfgang Fichtner. GRAPES: A Module Generator Based on Graph Planarity. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, ICCAD 1987, pages 428–431. IEEE, 1987.

[HF92] Hansruedi Heeb and Wolfgang Fichtner. A Module Generator Based on the PQ-tree Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):876–884, 1992.

[HHLH89] Chi-Yi Hwang, Yung-Ching Hsieh, Youn-Long Lin, and Yu-Chin Hsu. An Optimal Transistor-chaining Algorithm for CMOS Cell Layout. In *IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, ICCAD 1989, pages 344–347. IEEE, 1989.

[HHLH91] Yung-Ching Hsieh, Chi-Yi Hwang, Youn-Long Lin, and Yu-Chin Hsu. LiB: a CMOS cell compiler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 10(8):994–1005, August 1991.

[Hil85] Dwight D. Hill. Sc2: A Hybrid Automatic Layout System. In *IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, ICCAD 1985, pages 172–174. IEEE, 1985.

[HNS13] Stefan Hougardy, Tim Nieberg, and Jan Schneider. BCell: Automatic Layout of Leaf Cells. In *Proceedings of the 18th Asia and South Pacific Design Automation Conference*, ASP-DAC 2013, pages 453–460. IEEE, 2013.

[HS91a] King C. Ho and Sarma Sastry. Flexible Transistor Matrix (FTM). In *Proceedings of the 28th Design Automation Conference*, DAC 1991, pages 475–480. ACM, 1991.

[HS91b] Ying-Min Huang and M. Sarrafzadeh. A Parallel Algorithm for Minimum Dual-Cover with Application to CMOS Layout. *Journal of Circuits, Systems, and Computers*, 1(2):177–204, 1991.

[HSV] Stefan Hougardy, Jannik Silvanus, and Jens Vygen. Dijkstra Meets Steiner: Fast Algorithm for Optimum Steiner Trees. Reasearch Institute for Discrete Mathematics, University of Bonn, March 2014.

[HT74] John Hopcroft and Robert Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, October 1974.

[HW73] Carl Hierholzer and Chr. Wiener. Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[HW89] S. Huang and O. Wing. Improved Gate Matrix Layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):875–889, 1989.

[HW93] T. W. Her and D. F. Wong. Cell Area Minimization by Transistor Folding. In *Proceedings of the European Design Automation Conference, with EURO-VHDL '93*, EURO-DAC 1993, pages 172–177. IEEE, 1993.

[IIA04a] Tetsuya Iizuka, Makoto Ikeda, and Kunihiro Asada. Exact Wiring Fault Minimization via Comprehensive Layout Synthesis for CMOS Logic Cells. In *Proceedings of the 5th International Symposium on Quality Electronic Design*, ISQED 2004, pages 377–380. IEEE Computer Society, 2004.

[IIA04b] Tetsuya Iizuka, Makoto Ikeda, and Kunihiro Asada. High Speed Layout Synthesis for Minimum-width CMOS Logic Cells via Boolean Satisfiability. In *Proceedings of the 19th Asia and South Pacific Design Automation Conference*, ASP-DAC 2004, pages 149–154. IEEE Press, 2004.

[Jam11] Dave James. Intel predicts 1,200 quintillion transistors in the world by 2015. `http://www.techradar.com/news/computing-components/processors/intel-predicts-1-200-quintillion-transistors-in-the-world-by-2015-1025851`, September 13, 2011.

[Kar53] Maurice Karnaugh. The Map Method for Synthesis of Combinational Logic Circuits. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 72(5):593–599, November 1953.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[Kil64] Jack S. Kilby. Miniaturized electronic circuits, 1964. U.S. Patent 3,138,743, issued June 23, 1964.

[KK97] Jaewon Kim and S.-M. Kang. An Efficient Transistor Folding Algorithm For Row-based Cmos Layout Design. In *Proceedings of the 34th Design Automation Conference*, DAC 1997, pages 456–459. ACM, June 1997.

[Köl12] Katrin Kölker. Leaf Cell Routing im VLSI-Design. Diploma thesis, University of Bonn, Germany, July 2012.

[KV13] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Algorithms and Combinatorics. Springer, 5th edition, 2013.

[KW85] P. W. Kollaritsch and N. H. E. Weste. TOPOLOGIZER: An Expert System Translator of Transistor Connectivity to Symbolic Cell Layout. *IEEE Journal of Solid-State Circuits*, 20(3):799–804, June 1985.

[Lar71] Robert P. Larsen. Computer-Aided Preliminary Layout Design of Customized MOS Arrays. *IEEE Transactions on Computers*, C-20(5):512–523, May 1971.

[Lee61] Chin Yang Lee. An Algorithm for Path Connections and its Applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[LK82] J. Luhukay and W. J. Kubitz. A Layout Synthesis System for NMOS Gate-Cells. In *Proceedings of the 19th Design Automation Conference*, DAC 1982, pages 307–314. IEEE Press, June 1982.

[LL80] Alexander D. Lopez and Hung-Fai S. Law. A Dense Gate Matrix Layout Method for MOS VLSI. *IEEE Transactions on Electron Devices*, 27(8):1671–1675, 1980.

[LM88] Thomas Lengauer and Rolf Müller. Linear Algorithms for Optimizing the Layout of Dynamic CMOS Cells. *IEEE Transactions on Circuits and Systems*, 35(3):279–285, March 1988.

[LMSM10] Yi-Wei Lin, Malgorzata Marek-Sadowska, and Wojciech P. Maly. Layout Generator for Transistor-Level High-Density Regular Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(2):197–210, February 2010.

[LSR06] C. Lazzari, C. Santos, and R. Reis. A New Transistor-Level Layout Generation Strategy for Static CMOS Circuits. In *Proceedings of the 13th IEEE International Conference on Electronics, Circuits and Systems*, ICECS 2006, pages 660–663, December 2006.

[Mad89] Jan Madsen. A New Approach to Optimal Cell Synthesis. In *IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, ICCAD 1989, pages 336–339. IEEE, 1989.

[MAU86] Hiroshi Miyashita, Tohru Adachi, and Kazuhiro Ueda. An Automatic Cell Pattern Generation System for CMOS Transistor-pair Array LSI. *Integration, the VLSI Journal*, 4(2):115–133, 1986.

[McA99] A. J. McAllister. A New Heuristic Algorithm for the Linear Arrangement Problem. Technical Report Technical Report TR-99-126a, Faculty of Computer Science, University of New Brunswick, 1999.

[MD88] Frédéric Mailhot and Giovanni DeMicheli. Automatic layout and optimization of static CMOS cells. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors*, ICCD 1988, pages 180–185. IEEE, 1988.

[MFNK95] Hiroshi Murata, Kunihiro Fujiyoshi, Shigetoshi Nakatake, and Yoji Kajitani. Rectangle-packing-based Module Placement. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD 1995, pages 472–479. IEEE Computer Society, 1995.

[MH87] Robert L. Maziasz and John P. Hayes. Layout Optimization of CMOS Functional Cells. In *Proceedings of the 24th Design Automation Conference*, DAC 1987, pages 544–551, June 1987.

[MH91] Robert L. Maziasz and John P. Hayes. Exact Width and Height Minimization of CMOS Cells. In *Proceedings of the 28th Design Automation Conference*, DAC 1991, pages 487–493. ACM, 1991.

[MH92] Robert L. Maziasz and John P. Hayes. *Layout Minimization of CMOS Cells.* VLSI, Computer Architecture and Digital Signal Processing. Kluwer Academic Publishers, Boston/Dodrecht/London, 1992.

[MH96] Brian T. Murray and John P. Hayes. Testing ICs: Getting to the Core of the Problem. *Computer*, 29(11):32–38, November 1996.

[ML86] Rolf Müller and Thomas Lengauer. Linear Algorithms for Two CMOS Layout Problems. In Filia Makedon, Kurt Mehlhorn, T. Papatheodorou, and P. Spirakis, editors, *VLSI Algorithms and Architectures*, volume 227 of *Lecture Notes in Computer Science*, pages 121–132. Springer, 1986.

[MLMS07] Wojciech P. Maly, Yi-Wei Lin, and Malgorzata Marek-Sadowska. OPC-free and Minimally Irregular IC Design Style. In *Proceedings of the 44th Design Automation Conference*, DAC 2007, pages 954–957. ACM, 2007.

[MO88] C. T. McMullen and R. H. J. M. Otten. Minimum Length Linear Transistor Arrays in MOS. In *Proceedings of the 37th IEEE International Symposium on Circuits and Systems*, ISCAS 1988, pages 1783–1786. IEEE, June 1988.

[Moo65] Gordon E. Moore. Cramming More Components onto Integrated Circuits. *Electronics*, 38(8):4ff, April 1965.

[MP95] Enrico Malavasi and Davide Pandini. Optimum CMOS Stack Generation with Analog Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):107–122, 1995.

[NBR85] Ravi Nair, Anni Bruss, and John Reif. Linear time algorithms for optimal CMOS layout. In P. Bertolazzi and F. Luccio, editors, *VLSI: Algorithms and Architectures: Proceedings of the International Workshop on Parallel Computing and VLSI*, pages 327–338. Elsevier Science Publishers, North, 1985.

[NJ85] Tak-Kwong Ng and S. Lennart Johnson. Generation of Layouts from MOS Circuit Schematics: A Graph Theoretic Approach. In *Proceedings of the 22nd Design Automation Conference*, DAC 1985, pages 39–45. IEEE Press, 1985.

[NR96] L. S. Nyland and J. H. Reif. An Algebraic Technique for Generating Optimal CMOS Circuitry in Linear Time. *Computers & Mathematics with Applications*, 31(1):85–108, 1996.

[OLL89] Chong-Leong Ong, Jeong-Tyng Li, and Chi-Yuan Lo. GENAC: An Automatic Cell Synthesis Tool. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, DAC 1989, pages 239–244. ACM, 1989.

[OMK$^+$79] T. Ohtsuki, H. Mori, E.S. Kuh, T. Kashiwabara, and T. Fujisawa. One-Dimensional Logic Gate Assignment and Interval Graphs. *IEEE Transactions on Circuits and Systems*, 26(9):675–684, September 1979.

[PB83] C. Piguet and M. Bertarionne. Automatic Generation of Metal Oriented Layout for CMOS Logic. In *9th European Solid-State Circuits Conference*, ESSCIRC 1983, pages 167–170. IEEE, September 1983.

[PDCM11] Juan-José Pantrigo, Abraham Duarte, Vincente Campos, and Rafael Martí. Heuristics for the Minimum Linear Arrangement Problem. 2011.

[Poi89] Charles J. Poirier. Excellerator: Custom CMOS Leaf Cell Layout Generator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(7):744–755, July 1989.

[Pol12] Raphael Polig. Personal communication, February 2012.

[PZSB84] C. Piguet, J. Zahnd, A. Stauffer, and M. Bertarionne. A Metal-oriented Layout Structure for CMOS Logic. *IEEE Journal of Solid-State Circuits*, 19(3):425–436, June 1984.

[RB11] Nikolai Ryzhenko and Steven Burns. Physical Synthesis onto a Layout Fabric with Regular Diffusion and Polysilicon Geometries. In *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference*, DAC 2011, pages 83–88, June 2011.

[RB12] Nikolai Ryzhenko and Steven Burns. Standard Cell Routing via Boolean Satisfiability. In *Proceedings of the 49th Design Automation Conference*, DAC 2012, pages 603–612. ACM, 2012.

[RO71] N. A. Rose and J. V. Oldfield. Printed-wiring-board layout by computer. *Electronics and Power*, 17(10):376–379, 1971.

[RS99] Michael A. Riepe and Karem A. Sakallah. Transistor Level Micro-placement and Routing for Two-dimensional Digital VLSI Cell Synthesis. In *Proceedings of the 1999 International Symposium on Physical Design*, ISPD 1999, pages 74–81. ACM, 1999.

[RS03] Michael A. Riepe and Karem A. Sakallah. Transistor Placement for Noncomplementary Digital VLSI Cell Synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 8(1):81–107, January 2003.

[RTL95] Sanjay Rekhi, J. Donald Trotter, and Daniel H. Linder. Automatic Layout Synthesis of Leaf Cells. In *Proceedings of the 32nd Design Automation Conference*, DAC 1995, pages 267–272, 1995.

[SC90] Uminder Singh and C. Y. Roger Chen. A Transistor Reordering Technique for Gate Matrix Layout. In *Proceedings of the 27th Design Automation Conference*, DAC 1990, pages 462–467. ACM, 1990.

[Sil13] Jannik Silvanus. Fast Exact Steiner Tree Generation Using Dynamic Programming. Master's thesis, University of Bonn, Germany, October 2013.

[SOH$^+$81] I. Shirakawa, N. Okuda, T. Harada, S. Tani, and H. Ozaki. A Layout System for the Random Logic Portion of an MOS LSI Chip. *IEEE Transactions on Computers*, C-30(8):572–581, August 1981.

[SS99] Tatjana Serdar and Carl Sechen. AKORD: Transistor Level and Mixed Transistor/Gate Level Placement Tool for Digital Data Paths. In *IEEE/ACM International Conference on Computer-Aided Design. Digest of Technical Papers*, ICCAD 1999, pages 91–97. IEEE, November 1999.

[SSK$^+$88] Yoichi Shiraishi, Jun'ya Sakemi, Makoto Kutsuwada, Akira Tsukizoe, and Takashi Satoh. A High Packing Density Module Generator for CMOS Logic Cells. In *Proceedings of the 25th Design Automation Conference*, DAC 1988, pages 439–444. IEEE Computer Society Press, 1988.

[TP07] Brian Taylor and Larry Pileggi. Exact Combinatorial Optimization Methods for Physical Design of Regular Logic Bricks. In *Proceedings of the 44th Design Automation Conference*, DAC 2007, pages 344–349. ACM, 2007.

[Ull76] Julian R. Ullmann. An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[UTK88] S. Ueno, K. Tsuji, and Y. Kajitani. On Dual Eulerian Paths and Circuits in Plane Graphs. In *Proceedings of the 37th IEEE International Symposium on Circuits and Systems*, ISCAS 1988, pages 1835–1838, June 1988.

[Uv79] Takao Uehara and William M. vanCleemput. Optimal Layout of CMOS Functional Arrays. In *Proceedings of the 16th Design Automation Conference*, DAC 1979, pages 287–289. IEEE Press, June 1979.

[Uv81] Takao Uehara and William M. vanCleemput. Optimal Layout of CMOS Functional Arrays. *IEEE Transactions on Computers*, C-30(5):305–312, May 1981.

[Wan67] Frank M. Wanlass. Low stand-by power complementary field effect circuitry, 1967. U.S. Patent 3,356,858, issued December 5, 1967.

[Wei67] Arnold Weinberger. Large Scale Integration of MOS Complex Logic: A Layout Method. *IEEE Journal of Solid-State Circuits*, 2(4):182–190, 1967.

[Wey11] Thomas Weyd. Leaf Cell Layout. Bachelor's thesis, University of Bonn, Germany, August 2011.

[Wey13] Thomas Weyd. Mixed Integer Programming Approach for Leaf Cell Routing. Master's thesis, University of Bonn, Germany, September 2013.

[WHW85] O. Wing, Shuo Huang, and Rui Wang. Gate Matrix Layout. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(3):220–231, July 1985.

[WLC$^+$13] Po-Hsun Wu, M. P. Lin, Tung-Chieh Chen, Tsung-Yi Ho, Yu-Chuan Chen, Shun-Ren Siao, and Shu-Hung Lin. 1-D Cell Generation With Printability Enhancement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(3):419–432, March 2013.

[WLL88] D. F. Wong, H. W. Leong, and C. L. Liu. *Simulated Annealing for VLSI Design*. Kluwer Academic Publishers, 1988.

[WNMD83] Wayne Wolf, John Newkirk, Robert Mathews, and Robert Dutton. Dumbo, A Schematic-to-Layout Compiler. In Randal Bryant, editor, *3rd Caltech Conference on Very Large Scale Integration*, pages 379–393. Springer Berlin Heidelberg, 1983.

[WPF87] Shmuel Wimer, Ron Y. Pinter, and Jack A. Feldman. Optimal Chaining of CMOS Transistors in a Functional Cell. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):795–801, September 1987.

[YKK75] H. Yoshizawa, H. Kawanishi, and K. Kani. A Heuristic Procedure for Ordering MOS Arrays. In *Proceedings of the 12th Design Automation Conference*, DAC 1975, pages 384–393. IEEE, 1975.