A Technology Agnostic Approach for Standard-cell Layout Design Automation

Tom Johansson
tom.johansson@xenergic.com

Department of Electrical and Information Technology
Lund University

Supervisor:
Hemanth Prabhu (Xenergic AB)
hemanth.prabhu@xenergic.com
Babak Mohammadi
babak.mohammadi@eit.lth.com

Examiner:
Erik Larsson
erik.larsson@eit.lth.se

August 30, 2019

Abstract

The logic scaling following Moores law has reached a level where System on Chips (SoCs) commonly contains millions of standard cells. The sheer amount implies that even small optimizations on a standard cell can have a significant effect on the SoC performance. To ensure the performance of standard cells, many of these are hand-drawn. This is a tedious task that needs to be done every time a new process technology emerges. Something which requires both many man-hours and that holds a risk of designing sub-optimal solutions or introducing human error into the design.

Presented in this thesis is a method for automatizing the complete design process of standard cell layouts, requiring only a netlist and a few fundamental design rules for the given technology. The overall procedure was divided into three parts: placement, routing, and evaluation. The tool is entirely built in Python and for functionality verification commercial EDA tools from Cadence^{TM} were used. The generated standard cells have shown to match the area requirements of typical industry-level standard cells and in some critical complex cells even outperform them.

Popular Science Summary

Moore's Law states that the number of components on a chip will double every 18 months and today, a single Integrated Circuit (IC) can contain over 20,000,000,000 transistors [1]. This astronomical number makes it clear that a fully manual IC design process isn't realistic. Instead, an increasing level of automation is being implemented in the field of IC design to minimize the workload and to keep time to market as low as possible.

Standard cells are used to implement fundamental logic, such as AND, OR, XOR and INV into compact hardware blocks that are used in typical digital ICs. Hence, standard cells are often considered as the smallest building blocks and its critical to optimize these building blocks for Power, Performance and Area (PPA).

Designing the layout of standard cells can be a tedious task that is often drawn manually. Currently, most standard cells (as well as most other analog layout designs) are manually redesigned for every new process technology that emerges because of the new sizings and design rules that follow with them.

In this thesis, a generic approach for automating the design of standard cells is presented. The approach is designed in a way such that a minimal amount of technology-specific information is needed while still producing efficient cells. This would allow digital IC designers who receive a new technology to quickly generate a library to get going with their designs, even before achieving detail knowledge about their design rules.

List of Acronyms

IC	Integrated Circuit
SAT	Boolean satisfiability
EDA	Electronic Design Automation
SoC	System on Chip
MOSFET	Metal Oxide Semiconductor Field-Effect Transistor
NMOS	n-type Metal Oxide Semiconductor
PMOS	p-type Metal Oxide Semiconductor
CMOS	Complementary Metal Oxide Semiconductor
LVS	Layout Versus Schematic
DRC	Dynamic Rule Check
GDSII	Graphic Database System Information Interchange
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration
M1	Metal 1
M2	Metal 2
MUX	Multiplexer
RTL	Register Transfer Level
MST	Minimum Spanning Tree
PUN	Pull-Up Network
PDN	Pull-Down Network
iff	If and only if
poly	Polysilicon

SPICE	Simulation Program with Integrated Circuit Emphasis
NDA	Non-Disclosure Agreement
EUV	Extreme Ultra Violet
PPA	Power, Performance and Area
HDL	Hardware Description Language

Acknowledgements

This thesis has proved to be a challenge of many twists and turns.

For more than a month, it was intended to have a reinforcement learning-based focus. After reluctantly admitting defeat on the subject, I delved into further research to come up with a way to pivot the topic of my thesis into something doable.

I found the topic of standard cell generation to be a very interesting one, so in order to not abandon it, I decided to try an algorithmic approach instead.

For this, I needed to dive deeper into the details of how layouts for small logic gates were designed and why. This led me to study heaps of industry supplied cells in attempts to find patterns in the designs, and also to reason about which ones were necessary and which were not.

Several algorithm ideas were partially or fully implemented but ultimately had to be discarded due to either poor performance or erroneous behavior. Finally, however, all tenacity and hard work paid off, and I can proudly present a tool that performs well in both time-efficiency and result. A tool that I sincerely believe could become industrially competitive with just a little bit of extra love.

I would like to thank Dr. Hemanth Prabhu for always making himself available to discuss my ideas, helping to inspire new ones when I'm stuck and for providing an endless supply of laughter and office badgering.

I would like to thank Dr. Babak Mohammadi for providing insight into the analog design of integrated circuits, for inspiring us all with his superhuman work discipline and for making this endeavor possible in the first place.

I would also like to thank Professor Joachim Rodriguez for guiding me in the industry applications of my tools, and for all the lectures he has given me to help me become a better public speaker.

Finally, I would like to thank my family for your never-ending love and support throughout my education, and my partner, Michelle, for taking care of me when I'm too busy to do it myself.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The quickly increasing complexity of ICs causes more and more development related tasks to be unfeasible for humans to execute, which is why Electronic Design Automation (EDA) is a topic which has been subject for much research over the last decades.

To implement complex digital functionality onto an IC, it is commonly described using a Register Transfer Level (RTL)-language. RTL provides a high-level abstraction of hardware, through which an engineer can code the behavior without having to put much consideration into the features of low-level components. The RTL can later be interpreted through a process typically known as synthesis that realizes the desired functionality with gates from a standard cell library.

Standard cells are predesigned logic blocks implementing very elementary functionality such as AND-gates and OR-gates. These cells can then be duplicated and reused as black boxes across the entire IC to implement more complex logic. Since a single IC can contain millions of these blocks, it is crucial that they are well optimized to avoid unnecessary energy consumption or speed loss.

Designing the layout of standard cells can be a tedious task that is generally drawn by freehand [3] [4]. Currently, most standard cells are redesigned for every new process technology that emerges. Something which is required because of the new sizings and design rules that follow with them [5]. This requires both many manhours and holds a risk of designing sub-optimal solutions or introducing human error into the design.

When designing a layout, there are heaps of considerations to keep in mind. A designer generally attempts to optimize a trade-off of efficiency in area, power consumption and performance of the circuit such that its specifications are satisfied. In addition to this, the designer must also abide by the technology-specific design rules. The vast amount of parameters and constraints to consider, forces the designer to employ heuristics achieved by experience rather than to follow an optimal algorithm.

1.1 Scope of the thesis

The idea behind this project is to automate the design of standard cells, and thus remove the thousands of work hours currently required for the manual design for each new technology. Standard cells were chosen as a topic because they are the one component in a digital flow where analog engineers are still essential. Solving standard cell generation through a tool would therefore equip digital engineers with more freedom in their work. Furthermore, it is easy to use standard cells to evaluate and benchmark their performance by comparing to existing industry standard implementations.

Questions that are attempted to answer throughout this thesis includes

- Whether it is feasible to automize the design process of standard cells.
- How well such a tool can perform compared to manually designed layouts.
- To what extent such a tool can be made technology independent.

Automation of standard cell design is bound to become a cross-field research topic to some extent. With this thesis, patterns in industry level standard cells have been studied as an initial research step to help gain insight into what can and can't be done. However, the scope of this thesis focuses mostly on the algorithmic level and how to generate area-efficient standard cell layouts in a way that requires minimum consideration for the design rules of the process technology.

This thesis is developed at Xenergic AB, a company in Lund that is developing low-power **SRAM!**s. The project it describes is realized using a 28 nm bulk process technology.

1.2 Goals achieved

The outcome of the thesis is a successful one where almost every evaluated standard cell of below 12 transistors can be generated without any human intervention such that the area requirements remain the same as industry level standard cells. Additionally, with some manual aid, a full adder could be implemented such that it required less area than the reference library being used.

1.3 Outline

In this thesis, we will first introduce some foundations and present-time methods behind physical design and EDA. Afterwards, we will dive into some graph theory algorithms that have played a significant part in the shaping of the routing algorithm.

Following, we will describe the flow of the method through placement, routing, post-processing and evaluation along with the results achieved.

Finally, to conclude the thesis, we will discuss performance, usability and future improvements that can be implemented to make the method industrially competitive.

Chapter 2

Physical Design & EDA

2.1 Inside an integrated circuit

An electronic schematic is something most people have encountered already in high school. It gives a clear and accurate overview of what is connected to what in an electrical circuit, which in turn helps to get an intuition for what functionality it implements.

In integrated circuits, however, the physical design (or layout) looks quite different. In Figure 2.1 the cross-section of an n-type Metal Oxide Semiconductor (NMOS) is illustrated. This, along with its p-type Metal Oxide Semiconductor (PMOS) equivalent, makes up the elementary building blocks that realize the logic in an IC. For IC designers, the layout is commonly seen from a topographical point of view where each material type (or metal layer for routing) is viewed as a layer. From this perspective, a transistor looks like in Figure 2.2 if we omit the bulk and dopant layers from it. Now, if voltage is applied to the polysilicon gate (often referred to as poly), the path between the source and drain will become conductive. If the gate voltage is set to ground potential, the circuit will become resistive, thus isolating the potentials on either side of the poly from each other. Similarly, its PMOS counterpart is conductive for low gate voltages and resistive for high gate voltages.

CMOS (Complementary Metal Oxide Semiconductor) is a technology for designing integrated circuits. Its name derives from that it is built from complementary pairs of NMOS and PMOS transistors. The PMOS net connects the output to the supply voltage and the NMOS net connects the output to the ground rails. The complementary aspect of the nets is that they are designed such that only one of them can be open at a given time. Always having one of the nets completely turned off gives CMOS designs a very low static power consumption, only consuming significant power during the switching between on and off states. This power efficiency is one of the main reasons why CMOS now accounts for a vast majority of the logic in integrated circuits [6].

Figure 2.3 shows a schematic of an inverter, which is the simplest component that can be built with CMOS technology. If the input A is set to a logical '1', the NMOS transistor will open while the PMOS one will be turned off. This, in turn, will bind the output Z to ground (logical '0'). Similarly, the opposite will happen

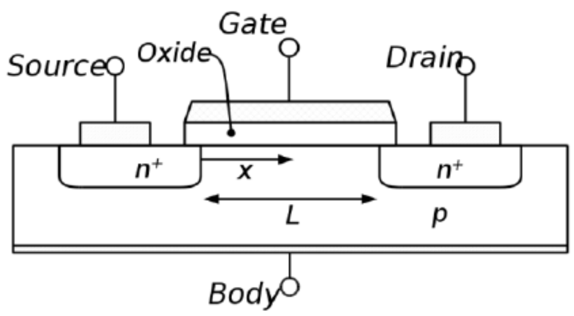


Figure 2.1: A cross-section image of a transistor. Illustration by [2]

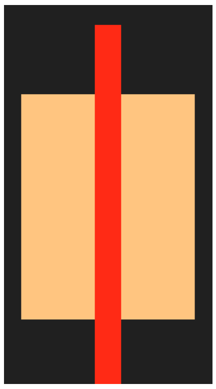


Figure 2.2: An illustration of how a transistor is viewed to a layout engineer. The yellow parts represent the source and drain while the red represents the gate.

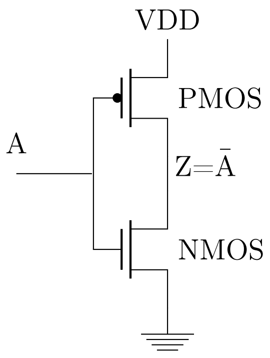


Figure 2.3: The schematic of a CMOS inverter

if A is set to '0'.

2.2 Introduction to VLSI

After the emergence of integrated circuits, the number of components on a chip grew rapidly, as mentioned in the introduction. Already in the 70s, many hardware developers started to put in extensive research into the development of tools that can automate the design effectively, efficiently and reliably [7]. Even the U.S. Department of Defense took part in developing VHSIC Hardware Description Language (VHDL) in 1983, which to this day is still among the most used Hardware Description Languages (HDLs) on the market [8] [9]. Something that gave rise to the fields of EDA and Very Large Scale Integration (VLSI).

VLSI is the process of designing an IC by combining millions of transistors into a single chip. A typical IC today can even contain upwards billions of transistors. This is obviously unfeasible to keep track of manually, which has led to that a

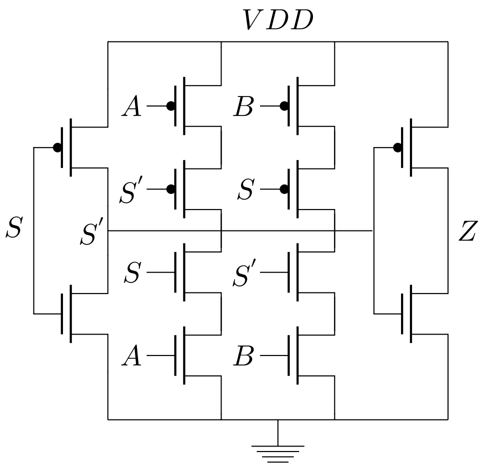


Figure 2.4: The schematic of a CMOS 2:1 MUX

large scale utilization of EDA tools has become a necessity in chip development.

RTL is an abstraction layer within digital IC design that models the flow of digital signals between hardware registers as well as the logical operations performed on those signals. RTL abstraction is used in HDLs such as Verilog and VHDL to describe the desired behavior of a circuit. From this, a lower-level circuit representation can be derived, including logic gate selection and wiring.

The process of interpreting an RTL description of a desired behavior and re-describe it in terms of logic gates is called logic synthesis. In other words, logic synthesis generates a circuit schematic from the RTL description it's fed with. During synthesis, the tool evaluates the information of each gate's propagation delay, which is described in their *.lib*-files. This information is then used to determine which gates to use to best meet the timing constraints set by the designer.

To implement the logic into an IC, global place and route tools need to be used to realize all the physical structures of the design. Firstly, all logic gates need to be placed on the IC such that the placement is routable and that related gates are located close to each other. Afterwards, the gates need to be connected with metal wires according to the schematic described by the synthesis tool. Both the placement and routing tools need to make use of the *.lef*-file of each gate, which contains information regarding the geometry of the gate along with the position of the gates pins. This is needed both to determine the size of the area that will be occupied by the gate as well as how to connect to the gate without causing any unwanted shorts.

The logic gates described above are commonly referred to as standard cells. These are small logic blocks that implements commonly used functionality, such as AND-gates, OR-gates, and MUXs. The standard cells are made to be duplicated and frequently reused across the chips, which is why it is imperative that these are well optimized. If, for example, an AND-gate would use one transistor width more than necessary and that AND-gate is reused in 100,000 instances over the

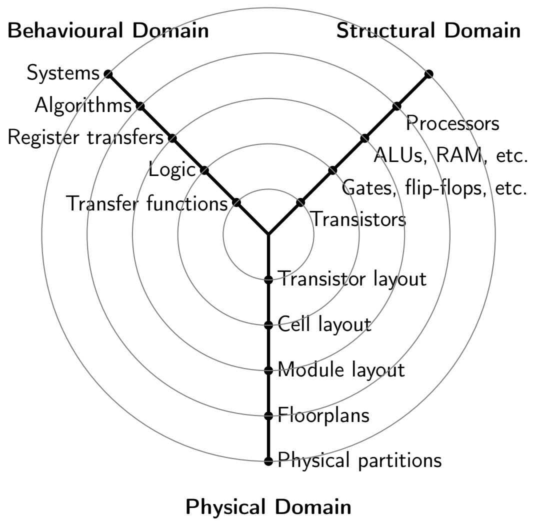


Figure 2.5: A Gajski-Kuhn Y-chart illustrating the abstraction levels from different hardware perspectives.

chip, that would imply a waste of a significant area which could have been used for more logic implementation.

An overview of a VLSI design flow is illustrated in the Gajski-Kuhn chart in Figure 2.5. Here, the hierarchical abstraction layers are depicted as circles inside each other, where the modules are divided into lower-level problems the further towards the middle you look. The three lines depict the design flow from different perspectives of hardware design, although ultimately all points on the same circle refer to equivalent modules. The tool described in this thesis will focus completely on the two innermost layers, thus creating building blocks that will be reiterated towards the outer abstraction layers.

2.3 Design goals & Techniques

2.3.1 PPA trade-off

When designing a circuit, the goal lies in implementing the desired logic such that it performs as fast as possible while requiring as little power and area as possible. These three properties often relate to one another such that if one is improved, the other two worsens. This frequently leaves hardware engineers with trade-off situations in which a component should be tweaked such that is becomes optimized for the parameter which is most crucial in the current case.

2.3.2 Drain & source sharing

A technique that can improve all three of the PPA properties simultaneously, and thus make it a trade-off free improvement, is drain or source sharing. This is

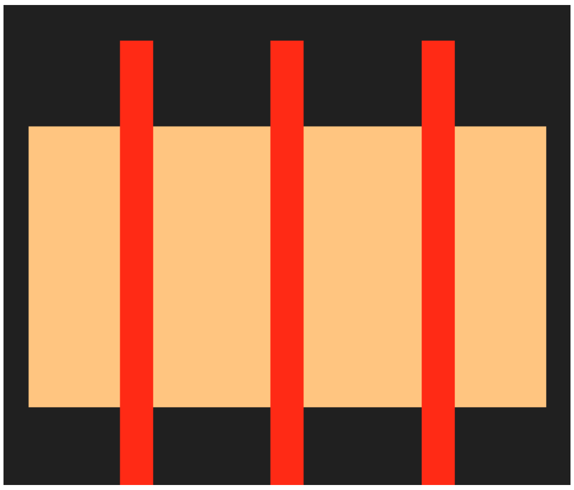


Figure 2.6: An example of source/drain sharing used to save area.

done when connected transistors, instead of being attached by a wire, are connected through sharing the same drain or source. The outcome of employing this technique is that the width of the placed layout will become one transistor width shorter for each drain/source sharing performed in the Pull-Up Network (PUN) or Pull-Down Network (PDN). Since there will be less interconnect within the cell, the parasitic capacitances will be reduced, resulting in improvements in both power consumption and performance [10] [11]. Also, since fewer routes will need to be drawn, maximizing drain/source sharing tends to make the routing step easier. An example is presented in Figure 2.6 where three consecutive gate lines over a diffusion layer make up three transistors. Furthermore, Figure 2.7 shows an example of a layout where some drain/source sharing has been used.

When designing standard cells, it is a common goal to exploit this feature as much as possible. A well-known structure in graph theory called Eulerian path is commonly used as an aid to maximize the utilization of drain and source sharing and will be described further in section 3.2.

2.3.3 Poly cutting

When trying to maximize drain/source sharing, cases may occur where an Eulerian path can be found for both the PUN and the PDN, but where matching paths do not exist. In such a case, a technique commonly known as poly cutting can be used. As the name implies, poly cutting divides the polysilicon gate into two parts. This enables opposite transistors to hold different gate signals simultaneously, thus allowing the PUN and PDN to follow different eulerian paths while still maximizing drain/source sharing.

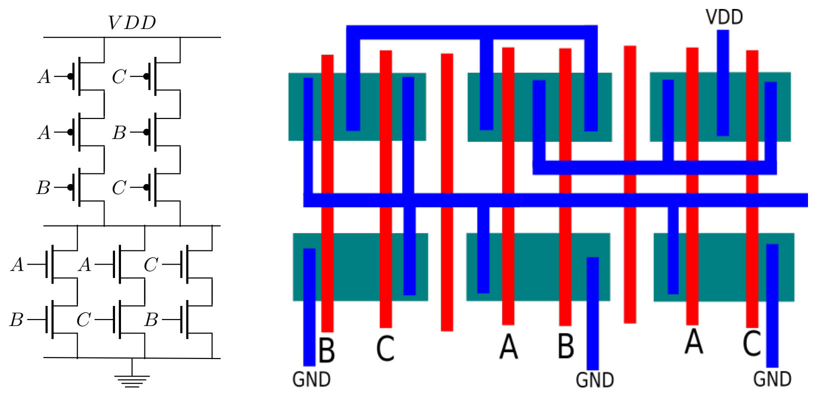


Figure 2.7: A "less than 2"-gate realized with a layout using source/drain sharing.

2.4 Previous research

As described in section 2.2, EDA problems are generally solved through many layers of divide and conquer methodology. This means that the main problem is divided into multiple smaller ones, which in turn are further divided. An example would be if a common layout placer only considers the size and pin positions of the standard cells as it places them on a chip while treating the rest of the standard cell as a black box. Designing the actual standard cell is considered a completely separate problem, which in turn can be further divided.

Since problem division is a well-employed methodology in hardware design, it is hard to find any papers on the complete design of standard cells. The authors of [12], however, published one such paper in which the goal is to generate as dense standard cells as possible. The transistor placement strategy described in the paper is performed with much focus on the consideration for routability in the subsequent step. They aim to aid this step by defining a cost function based on the total amount of subnets that are longer than minimum sized in the placement combined with some heuristic measures for how well it will be externally routable. This cost function is later minimized through the use of simulated annealing which is a stochastic method for finding approximate global extrema in discrete environments [13].

The routing method described in the paper features a greedy route-by-route depth-first search followed by an SAT[1] solving method. It uses some problem-reducing constraints, such as a maximum amount of turns of a wire. Furthermore, they implement an additional step for evaluating resource congestion. Something that can help with pushing the router towards areas used by fewer nets. It should be noted, though, that the authors of this paper have made it easier for themselves by allowing routing in diagonal directions. Something which is rarely allowed in newer technologies.

In [4] and [14], the problem has been focused towards the selection within

[1]SAT will be further explained in the next chapter.

a pre-generated set of routes. Here, the idea is to filter as many unfeasible or redundant routes as possible before solving an SAT problem which is defined to further detect routes that are incompatible with the rest of the nets. Afterwards, they solve an additional SAT-problem which searches for route combinations that are compatible.

Furthermore, [14] has put in extra effort into the evergrowing complexity of the design rules that follow from the smaller technologies. Here, SAT formulations for design rules and design robustness have been created to make sure that the environmentally sensitive lithography technology used in the Extreme Ultra Violet (EUV) process won't risk the causing of any unwanted shorts.

Chapter 3

Graph Theory & Algorithms

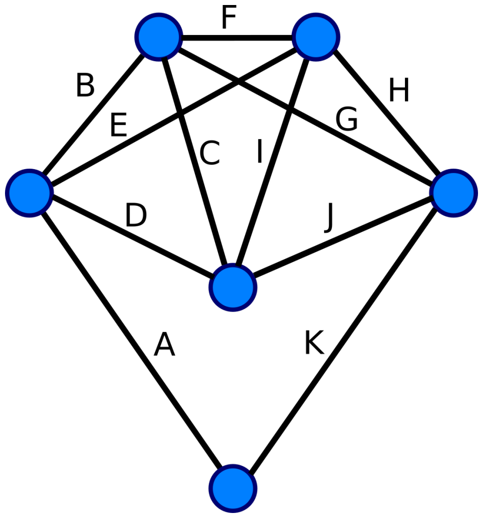


Figure 3.1: An illustration of an Eulerian path. An Euler path is achieved if following the edges in alphabetical order

Throughout the development of this thesis, the field of graph theory has proven to be very useful. It contains an extensive toolbox for efficiently analyzing and manipulating pairwise relations between nodes, which is something that can be applied to many aspects of layout design. A more formal introduction to graph theory will follow in the next subsection.

3.1 Background

In its most basic definition, a graph is defined as an ordered pair $G = (V, E)$ where

- V is a set of nodes
- $E \subseteq \{\{x, y\} | (x, y) \in V^2 \wedge x \neq y\}$ is a set of edges.

In words, this can be summarized as a structure consisting of a set of objects in which some of the objects as pairwise related. As a subset of the original graph, the edges it contains can also be paired with values to create a weighted graph.

Graph theory is used to model relations and processes in many fields including physics, biology [15], sociology [16] and computer science. A well-known example of where graph theory is used within many of the models for neural networks.

Some methods from graph theory that have proven to be very useful over the course of this thesis will be explained below.

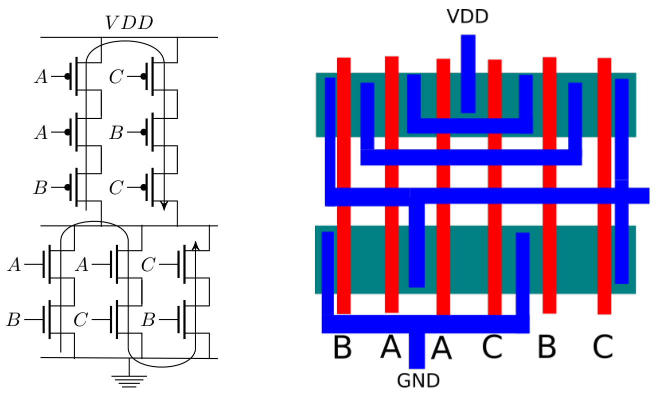


Figure 3.2: The logic from Figure 2.7 implemented in a less-than-2 gate. Illustration of a layout designed from matching Euler paths.

3.2 Relevant algorithms

3.2.1 Eulerian paths

Eulerian paths, or Euler paths, is a trail which passes through every edge in a graph exactly once. An example is shown in Figure 3.1, where an Euler path is achieved if the edges are followed in alphabetical order.

In a practical layout scenario, it is always ideal, from an area point of view, to enable source/drain sharing across all transistors in a network if possible. Additionally, the merging of the transistors reduces the need for wiring in the cell. This reduces the capacitive and resistive properties of the circuits, which in turn improves performance and power efficiency. Furthermore, the routing itself becomes an easier task by having fewer nets to route. A way to find a valid placement that maximizes the usage of source/drain sharing is by considering the schematic as a graph where the transistors are seen as edges with gate signals as identifiers. Then the logic can be realized with only one diffusion layer block if and only if it is possible to draw an Eulerian path through the PUN and PDN respectively such that the order of the identifiers is identical for both the paths.

In Figure 5.5, the circuit from Figure 2.7 has been reconstructed using complete matching Euler paths. It is clearly visible that both the required area as well as the used wiring has been reduced significantly for this case.

3.2.2 Graph traversal algorithms

There exist many methods for routing the metal layers in an integrated circuit. For larger systems, a common approach is to look at the layout from a very coarse scope and route wires semi-independently such that the routes are spread out appropriately much. After that step, the algorithm begins the detail routing by trying to connect the nets within the pre-routed areas without conflicting wires from different nets. This is a method commonly known as a global routing algorithm [17].

For this thesis, we've decided to explore a routing algorithm that considers each point-to-point route independently and selects a combination of these such that no routes from different nets collide. Graph traversal algorithms can be used to efficiently generate these routes.

Dijkstra's algorithm

Dijkstra's algorithm is an algorithm designed to find the shortest path between two nodes in a graph. It is guaranteed to always find the best solution. The algorithm works as follows [18]:

1. Mark all nodes in the graph as unvisited.
2. Choose an origin node and set a distance score of this node to 0. Set the score of all other nodes to infinity.
3. For the current node, consider all of its unvisited neighbors and calculate their distance score through the current node. Compare with the current score and assign the smallest.
4. Mark the current node as visited.
5. Finish if the destination node has been marked as visited.
6. Otherwise, select the unvisited node with the lowest distance score and go back to step 3.

Dijkstra's algorithm is the fastest known shortest path algorithm for generic graphs, assuming no pre-existing knowledge and it runs with time complexity $\mathcal{O}(|V|^2)$, where V is the number of nodes in the graph [19].

In layout design, the grid space can become huge and the number of nodes required to get an accurate resolution can quickly make the algorithm unfeasibly slow.
Luckily, we do have some preexisting knowledge about the graph that we can use. Namely the geographical positions of all nodes. This information proves to be of huge help when looking towards the A* algorithm instead.

A* algorithm

The A* algorithm is an informed search algorithm and is considered as an extension to Dijkstra's algorithm. It works by adding a heuristic to the cost of each node, guessing the remaining distance to the final destination [20].

The algorithm works as follows:

1. Select an origin node.
2. For the current node, consider all of its neighbors and calculate their estimated cost as a sum of the traversed distance and a heuristic of the remaining distance.
3. Add all neighbors with their estimated costs to a sorted set.

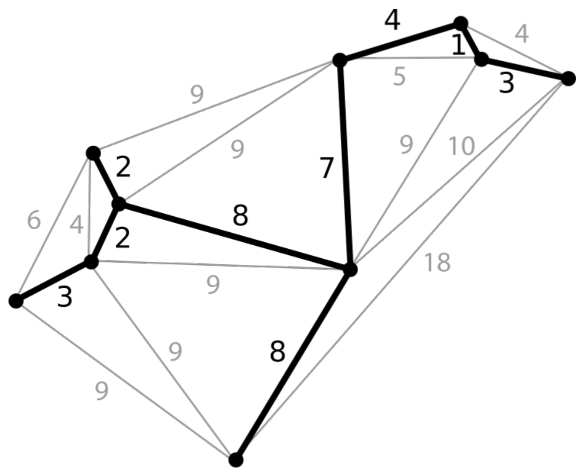


Figure 3.3: An example of a minimum spanning tree.

4. Finish if the destination node has been marked as visited or if the set is empty.
5. Otherwise, select the unvisited node with the lowest estimated cost and go back to step 2.

This algorithm has been shown to excel in many real-world applications where spatial data tends to be available at least to some extent. Examples where A* has proven to be of great use is in road navigation systems [21]. In those cases, trying all possible road combinations to get to a certain location would be unreasonably demanding, which is why some form of heuristic needs to be applied.

3.2.3 Minimum-spanning-tree algorithms

A Minimum Spanning Tree (MST) is the subset of edges in a graph that connects all of its nodes such that the total cost is as small as possible [22]. An example of an MST is illustrated in Figure 3.3.

A typical field where it is useful to find an MST is within telecommunications or power grid planning where cables need to be buried such that all houses are connected. In a use case such as this, the edges of the graph would be all areas where the cable can be buried and the nodes would be the houses that are to be connected. The MST would then represent the path that requires the least amount of wiring while connecting exactly all of the houses.

Kruskal's algorithm

Kruskal's algorithm is a greedy MST algorithm. An algorithm being greedy means that it's following a problem-solving heuristic that is only locally optimal while searching for a global optimum. In spite of this, Kruskal's algorithm guarantees an optimal solution for all cases where one exists [23]. For large problems, however, it becomes very inefficient with a time complexity of $\mathcal{O}(ElogE)$, where E is the amount of edges in the graph.

Kruskal's algorithm goes as follows:

1. Put all edges E in a list sorted by weights.
2. Pop the smallest edge e_n from the list.
3. If the nodes on each side of the edge are already connected, go to step 2.
4. Add e_n to a set of MST edges E_{MST}.
5. Finished if all nodes are connected to the same net. Otherwise, return to step 2.

Chapter 4
Tool Development & Considerations

Overview

In this chapter, we will present the components to the thesis project and the reasoning behind the choices we've made in the design. We will also relate and compare with some of the previously existing research that has inspired this work.

As with most things in the EDA world, we have deployed the divide and conquer strategy by recursively trying to chop up many of the tasks to smaller ones that are more easily solved. However, the thesis can be roughly divided into 3 main components:

1. Selection of transistor placement.
2. Routing of the nets.
3. Evaluation of the result compared to industry level standard cells.

4.1 Transistor placement

The goal of the transistor placement component is to find the placement ordering that requires the smallest possible area as well as the least amount of wiring.

While somewhat correlative, these two criteria give rise to two completely different tasks. The ordering in this approach is optimized sequentially, first on area consumption and secondly on wire requirements. This is motivated by that we have come up with an efficient method for quickly generating area optimized CMOS-logic and that it turns out that the most wire effective solution often is part of the set of area optimized solutions.

4.1.1 Area optimization

The focus of the area optimization part of the placement was on the search for Euler paths through the circuit. By treating all the transistors as edges and their interconnects as nodes, the network could be seen as graphs to traverse, thus simplifying the problem.

The initial thought for this step was that the small problem size that comes from only designing standard cells would make the problem solvable by employing the following "brute force" method. It should be noted that this algorithm was created before graph theory entered the focus of the thesis.

1. Select the drain/source of one transistor in the PUN and add the transistor to an empty set.
2. Iterate through all transistors not in the set to find those with a drain/source matching the previous one.
3. Create a new set for each such transistor extending the previous one and select the opposite drain/source of the last transistor.
4. If any transistor in a set still has an adjacent transistor, not in its set, go to step 2. Otherwise, continue.
5. Repeat from step 1 with the PDN. Then skip this step.

6. Search the sets for entries where the sequence of gate signals are identical for both the PUNs and PDNs.

This algorithm proved to have two big flaws. Firstly, it does not work for standard cells which cannot be realized in a single Euler path, and secondly, the computational complexity of this method is $\mathcal{O}(|N/2|!)$ where N is the amount of transistors in the netlist, making the computation time and memory usage explode as we're approaching larger standard cells.

To address both of these issues a second algorithm was developed. In this improved version, an empty alternative was added as a possible component in the transistor sequence for both networks. The empty entry can be matched against any other entry in the opposite network, allowing for any shape of the PUN and PDN regardless of the other. Since this algorithm allows empty alternatives, there are no longer any ways for a transistor sequence to be "locked in" and forced to continue on other combinations using the recursive method in the previous algorithm. Instead, this version performs the matching between the gate connections in the subnets on the fly and employs the A* algorithm to determine the best transistor sequence to expand.

The second version is considering the nets as graphs and goes as follows:

1. Consider each transistor as an edge with its nodes being the nets the transistor is connected to. Each edge also holds an identifier equal to the name if the transistors gate signal.
2. Generate a list holding all edges for PUN and PDN respectively.
3. Build a set $S1$ for each combination of one edge or *None* from each subnet. For each edge in $S1$, push the combination as a list of one element lists Li to a minheap H along with a cost value representing $x - distance\_traversed + min(x - distance\_remaining)$.
4. Pop Li from H and fetch all adjacent edges not yet traversed through.
5. Build a set $S2$ for each combination of one edge or *None* from each subnet.
6. Iterate through $S2$. If both edges hold the same identifier or if at least one of the edges are *None*, add the combination to a copy of Li and push it to the heap along with a cost value defined as before. Otherwise, discard this combination.
7. Repeat steps 4-6 until no unvisited edges exist in Li. Li will then hold an area optimized placement alternative.

This reduces the asymptotic time complexity to $\mathcal{O}((N+1)^2)$ which, while much better than the previous version, is further improved by the guided search provided by A*. Another good property of this algorithm is that it can be left running to find more placement alternatives. Most often, one netlist contains multiple alternatives tying for the same area usage.

In this approach, we fetch all of the best combinations and feed them to the wire optimization step.

4.1.2 Wire optimization

The wire optimization step can be seen as a preprocessing step to the routing of the standard cell. The routing is the most time-consuming part of the layout automation tool and it can become demanding to route all possible placement alternatives. Because of this, the area optimized placements are sorted based on heuristics that estimates which layouts are most likely enable efficient routing. There are two different heuristics being employed sequentially.

1. The placement which holds the smallest cumulative horizontal span of all nets.
2. The placement which holds the least maximum vertical stacking.

The first of these heuristics is simply intended to guess which of the placements will need the least amount of wiring. The other has the purpose of minimizing local congestions, thus making the layout easier to route. Consequently, the order of the sorting heuristics could be reversed if hard-to-route layouts are encountered.

4.2 Net drawing

One of the most important constraints of this tool is the agnosticism to process technology. Because of this, it is important that only the absolutely essential data from the technology file is being forwarded to the tool. [4] is another paper that holds the same constraints for process independence which has inspired some of the reasoning in the routing part of this project. However, since we're actually creating our own layout and routes rather than puzzling pre-existing ones, we need some extra information including minimum wire width and minimum wire spacing.

4.2.1 Initial attempts

For this part of the thesis, many different ideas have been studied and sometimes also realized and evaluated. Initially, a generative reinforcement learning approach was considered. This would, in its most general format, mean that the algorithm explores the layout space through trial and error and improves its performance from positive or negative reinforcement scores. Routing, though, proves to be a problem with exponential complexity and the chances of generating a fully connected net "by accident" are incredibly small. This means that the reinforcement learning agent will never get any positive rewards to learn from, and is therefore unlikely to improve. This is a famous problem known as a sparse reward problem which is currently undergoing lots of research.

Another approach that was considered was to place horizontal and vertical wires on different layers, thus making an easily routable grid. This idea was quickly discarded, though, due to the bad performance that can be expected from the output by having close to 50 % of all routes in Metal 2 (M2).

The last major approach that was attempted was to modify the A* algorithm to work as a multi net algorithm. This was done by defining the cost function as the total length of all current wiring and the heuristic as the sum of all minimum remaining distances between metal and endpoints. The grid steps were defined as the minimum notch of the wiring. Each iteration then had the available expansions of moving one step with either wire stub in either legal direction. It turned out, though, that the A* algorithm became greedy when dealing with multiple nets, getting stuck in local minima where the wiring can become close to fully connected, but with some nets locked in, close to a successful result.

4.2.2 Final approach

The approach that is currently employed for drawing the metal on the layout is a modular approach following multiple larger steps. The steps will be described one by one below.

Environment definition

Before the routing can begin, the layout space needs to be properly described and quantified into a three-dimensional grid. For this project, I've designed a coarse grid with intervals based on physical properties from the design rules. The grid intervals were chosen as follows:

$$dy = min(w\_metal) + min(w\_spacing) \tag{4.1}$$

$$dx = gate\_interval/2 \tag{4.2}$$

On this grid, no two objects belonging to different nets can be placed in the same point. Furthermore, one additional rule is that no two objects from different nets that are on the same y can be at less than $2dx$ distance from each other.

Support for adding weight factors for routing at a certain grid node has been implemented. This could work as a preprocessing step in aiding the routing step explained in the next section by guiding it away from congested areas. However, since no good enough method for deciding how to set the weights has been designed, this functionality is currently idle.

When the grid is in place, each net node is assigned to a grid coordinate and paired with all other nodes of the same net. Afterwards, all pairs except supply nodes are passed to the router.

Routing

The idea as a whole with this routing approach is to gather lots of possible candidate routes for each point-to-point connection, then select the combination of these that satisfies a cost function without breaking any Dynamic Rule Check (DRC)-rules.

The router at its core relies on the A* algorithm for point-to-point routing. In contrast with the normal use of the algorithm, though, this was left to keep searching for more possible routes, exploiting the data that was already gained by the previous search. The maze solver keeps searching for route alternatives until a predetermined number of routes has been found. Given the coarse grid, a fairly large portion of the solution space can be explored with a reasonable number of routes. The routing algorithm accepts user customization, primarily to let the user decide the weight of having M2 wires and vias in the layout. With these, the cost of the A* is defined as

$$cost = \sum_{n=0}^{N} l(n) \cdot cost\_factor(layer(n)) + \sum^{V} via\_cost \tag{4.3}$$

Similarly, the remaining distance heuristic is defined with the following equations:

$$dist = |x - x_{goal}| + |y - y_{goal}| \tag{4.4}$$

$$heur = \begin{cases} dist \cdot cost\_factor(layer), & \text{if } layer = layer_{goal} \\ dist \cdot cost\_factor(layer) + via\_cost, & \text{otherwise} \end{cases} \tag{4.5}$$

In order to avoid any obviously redundant routes, checks are made on the fly during the course of routing to make sure that:

- No u-shape can be made in any of the three dimensions such that the space is to small for an intermediate wire.
- No route can ever re-enter a node it's previously been in.

Selection

When a predefined number of routes have been found for each point-to-point connection, we need to match them such that no routes conflict with each other while the total cost is as small as possible. This is something that is done at the time complexity of $\mathcal{O}(R^N/2)$, where R is the number of routes per net and N is the number of nets in the netlist. This can become very computationally demanding for larger cells, which is why the first step of this approach is about minimizing the size of the route combination problem.

There are two methods being applied sequentially in order to remove unnecessary routes from the data set:

- Kruskal's algorithm
- SAT solving

Kruskal's algorithm is being used as the first step to build a minimum spanning tree for each of the nets. According to what is presented in the theory section, it works by connecting the unconnected node that is closest to a point in its connected net repeatedly until all nodes are connected. Kruskal is generally considered as a greedy algorithm, but since the number of connection points in a net is expected to be in the range of less than 10 and not in the range of thousands, its time consumption is still negligible.

After Kruskal has removed all unnecessary point-to-point connections, the routes that are incompatible with the rest of the system needs to be removed. In this approach, that is done through solving a boolean satisfiability (or SAT) problem, following an approach similar to what was presented in [4]. The SAT problem is about finding a set of variables such that a boolean formula asserts true. In this case, the formula can be defined as

$$r_{ab}^{ok} = \bigwedge_{\substack{p=0 \\ p \neq a}}^{P} \left(\bigvee_{j=0}^{R_p} (r_{ab} \wedge r_{pj}) \right) \tag{4.6}$$

where P denotes the number of point-to-point connections in the system, R_p the number of routes for a connection and r_{pj} denotes route j at point-to-point connection p. $r_{ab} \wedge r_{pj}$ is defined to assert true if both routes can be drawn without conflict. In words, this means that a route will be considered invalid if there exists a point-to-point connection with which no generated routes are compatible. Furthermore, if exactly all routes within a point-to-point connection are deemed invalid, the system is to be considered unroutable for the given placement and generated routes.

		Conn 1			Conn 2		Conn 3		Conn 4			C5
		r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
Conn 1	r_1	-			x	x		x				x
	r_2		-		x							
	r_3			-	x						x	
Conn 2	r_4	x	x	x	-							
	r_5	x				-	x		x			
Conn 3	r_6					x	-					
	r_7	x						-				
Conn 4	r_8					x			-			x
	r_9									-		
	r_{10}			x							-	
Conn 5	r_{11}	x							x			-

Table 4.1: Illustration of how the nets are filtered out in the SAT step. r_1 is completely blocking connections 2 and 5. Also r_4 and r_{11} are unfeasible.

		Conn 1		C2	Conn 3		Conn 4		C5
		r_2	r_3	r_5	r_6	r_7	r_9	r_{10}	r_{11}
Conn 1	r_2	-							
	r_3		-					x	
Conn 2	r_5			-	x				
Conn 3	r_6			x	-				
	r_7					-			
Conn 4	r_9						-		
	r_{10}		x					-	
Conn 5	r_{11}								-

Table 4.2: After filtering, r_6 completely blocks connection 2 and needs to be removed as well.

		Conn 1		C2	C3	Conn 4		C5
		r_2	r_3	r_5	r_7	r_9	r_{10}	r_{11}
Conn 1	r_2	-						
	r_3		-				x	
Conn 2	r_5			-				
Conn 3	r_7				-			
Conn 4	r_9					-		
	r_{10}		x				-	
Conn 5	r_{11}							-

Table 4.3: Filtered system with no unfeasible routes remaining.

In a net it is clearly beneficial, from all three optimization parameters point of view, if the routes coincide with each other, thus sharing metal. Because of this, a discount system has been defined, reducing the net cost corresponding to the amount of overlapping of its routes. The set of available routes that fully spans a net at the least cost has been coined as *best buddies*.

After the routes have been filtered and buddies defined, the next step is to find a combination of the remaining routes such that the layout is fully routable with the least possible wiring. The total amount of possible combinations can be written as

$$combos = \prod_{n=0}^{N} len(p_n) \tag{4.7}$$

where N is the amount of point-to-point connections in the layout and $len(p_n)$ represents the number of route alternatives remaining in a point-to-point connection.

This number will quickly become very large, even for small netlists. Something that disables us from trying all the possible combinations sequentially. Instead, a method for trying combinations in such a way that total metal cost goes from low to high while keeping unnecessary attempts at a minimum had to be designed.

For this part, we are given a list L_p of routes for each of the point-to-point connections p. A few methods were considered and discarded for this task.

One involved simply increasing the index for one of the lists, trying the combination, then increasing an index again if no successful route combination was found. This method would skip past many alternatives, though, making it an incomplete solver. An illustration of the problem is shown in Figure 4.1.

A second procedure was a recursive solution where each index increment was added to a heap as a (cost, indices) structure. This, however, caused problems in that the same index combination can be reached in many different ways, leading to large overheads in re-investigating the same combination multiple times. Particularly for high indices, the number of possible ways is so many that the same combination could be checked billions of times. An example of how the same index combination can be reached multiple times with this algorithm is shown in 4.2

The final solution builds on the second one. By adding the previously tried index combinations as a tuple to a set the program can abort re-evaluations of these as they appear. It's important that the data structure is made to be a set

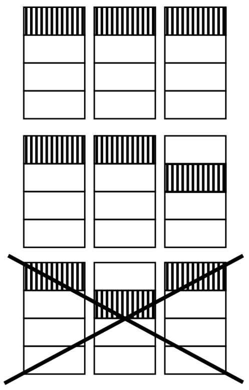


Figure 4.1: Illustration of the problem with constant increment. If this algorithm is used, the combination in step 3 may never be tried.

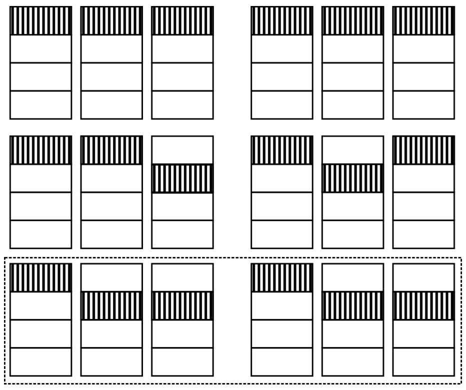


Figure 4.2: Illustration of the problem with recursive increment. If this algorithm is used, the combination in step 3 may be tried multiple times.

of tuples as the hashing operation of the set enables the search to be performed at $\mathcal{O}(log(n))$ as opposed to $\mathcal{O}(n)$ which normal list comparison would operate. For millions of attempts, this makes a very large difference. Additionally, the expansion of an index doesn't happen until the route it represents is involved in a conflict. This doesn't remove any of the possible combinations, but it does allow the program to focus on faster iterating through the more problematic routes.

The method currently being used adheres to the following algorithm:

1. Sort all the lists L_p on the cost of their routes.
2. Initiate a tuple of indexes for the lists L_p, starting at 0 for each list.
3. Add the tuple to a minheap H along with the total cost value for the routes represented.
4. Pop from H and investigate if any nets are colliding.
5. If yes, increase create a new tuple for each colliding net where that index has been incremented by 1.
6. For each new tuple, look in a set of tried combinations S if such a combination has been tried. If not, push the tuple to the heap along with a new associated cost and add the tuple to S.
7. Repeat from step 4 until a successful net combination has been found.

Genericness of the routing method

While the grid system alone produces very high generalizability in the route creation, there are a few fundamental rules that are not covered by it alone. One such rule is the available directions for routing. The technology used in developing this method allows for routing in the x and y directions, which is why that is the routing style featured in this thesis. Other technologies, however, such as the 0.6 μm technology allows for diagonal routing as well. Furthermore, it is not unreasonable to assume that other technologies may prefer certain directions of routing, such as is the case for most global routing scenarios. To address this constraint, the method presented in this thesis allows for easy configuration of which directions are allowed for each metal layer. It also supports soft breaks in certain directions by increasing their cost if that should become relevant.

Via placement

When nets and transistor layouts have been chosen, it's time to place the vias. Vias add a few extra constraints to its surroundings such as the required encapsulation of the layers it connects. This implies a need for some extra information from the technology. In order to maintain as high indifference to the process technology as possible, I've built a decision tree that considers the surroundings of the layout and builds a metal structure around the via such that it makes the via legal but without intruding on other metal drawing spacing constraints. The inherent margin in the x-axis that comes from the grid, has been shown to give the room needed for metal expansion in that direction. The only extra information needed from the technology for this problem is the required extra sizing of the layers adjacent to the vias.

A fundamental flaw with this approach would be if it encountered a technology that required more extra space than is provided by the margin. In this case, it is reasonable to assume that a less coarse post-processing step or manual intervention could aid in finding the needed space.

Additional layout features

As a final step, after all the components are in place, some required layers need to be put in place. Examples are the dopant masks and the body contacts. This step is performed in a rather hardcoded manner, simply depending on the size of the finalized layout. Since not much consideration is placed on these layers, we won't elaborate on their implementation any further. A demonstration of the extra layers can be seen in Figure 5.1 in the Results section.

4.3 Evaluation

4.3.1 Preprocessing

Since this project has been built entirely with Python, a first step before being able to properly evaluate the results is to export the data to a format recognized by the simulation environment. Graphic Database System Information Interchange (GDSII) is a file format that has become the industry standard for data exchange of IC layouts. In GDSII all layout data is stored as polygons or labels along with an associated layer and datatype.

There can be tons of different layers for each process technology including both metal types and abstract design rules that should be applied for a given area. Furthermore, layer and datatype id may vary for different technologies. Because of this, a mapping function was built to map the internal naming of all layers to their corresponding counterparts in a given technology.

4.3.2 Procedure

The evaluation was made using Cadence Virtuoso, which is an analog design environment that aids the user with drawing, modifying and verifying layouts. With the help of Virtuoso, a proper schematic implementing the logic could be built. Afterwards, a Layout Versus Schematic (LVS) test could be run to verify that the layout does indeed implement the desired logic. Also, a DRC check was run to verify that the layout was built in a manner that does not break any of the design rules.

Since the method is intended to be technology independent, the area of the cells is measured by gate count rather than absolute measurements. Considering the size of the cells, this simply became a manual counting task for the purposes of this evaluation.

It should be noted that the cells are never evaluated on performance and power consumption. This is motivated by that the cells capacitive values aren't expected to vary significantly from the industry level cells. Furthermore, it would be too time-consuming to set up the capacitive extraction and simulation test benches.

Chapter 5

Generation Performance

5.1 Results

The final outcome of the tool presented in this thesis is one which is verified to feature quick transistor placement and metal route generation for most standard cells consisting of 12 transistors or less. The largest cell generated purely through the tool has been one containing 18 transistors. As can be seen in the figures 5.1, 5.3 and 5.4, the outer metal wires for most cells overlaps with the outermost poly. This is something that, during implementation, would increase the required area of the standard cell by one transistor width. However, it has been both practically shown and studied from industry level standard cells that these wires can be "squeezed" to fit within the margin required to leave space for an adjacent cell. As is shown in Figure 5.2, the tool supports adding any extra peripheral layers, such as dopant areas, that are needed to make the design immediately integrable.

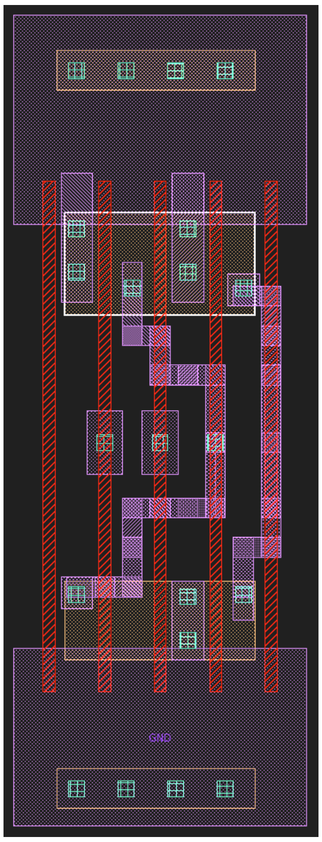
The industry level standard cells being compared with in this thesis are protected by a Non-Disclosure Agreement (NDA) which is why no comparative layouts or supplier names are given throughout the thesis.

Cell type	# transistors	Routed connections	Area [gate count][1]	Industry area	Generation time
BUFF	4	3	2 (3)	2	4 s
2-AND	6	3	3 (4)	3	5 s
4-AND	10	3	5 (6)	5	12 s
Full adder part 1	10	5	5 (6)	-	25 s
MUX	12	9	7 (8)	6	24 s
XOR	12	12	7 (8)	7	2 min 5 s
Full adder part 2	18	10	9 (10)	-	3 min 13 s
Full adder	28	20	15	16	$\sim$ 4 min + puzzle time[2]

Table 5.1: A table showing the performance and generation time for some of the most common standard cells.

[1]The number in the parenthesis refers to the area of the output of the tool. The number outside of the parenthesis is the area required for the same cell design after "wire squeezing".

[2]Puzzle time refers to the time used to attach two separately generated layouts. This is further explained in section 5.1.1



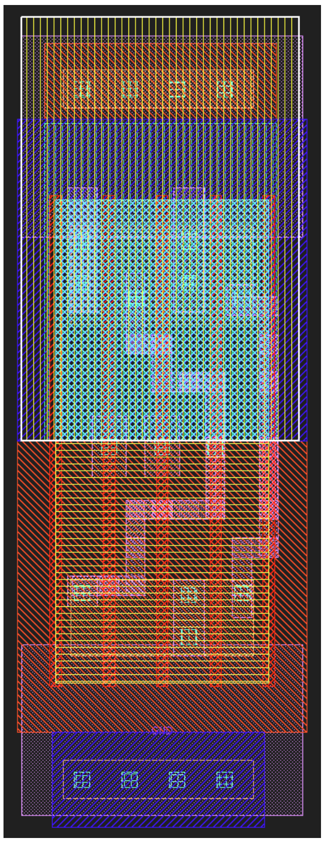


Figure 5.1: Generated layout for an AND-gate.

Figure 5.2: Same layout showing all of the layers used to generate the complete cell.

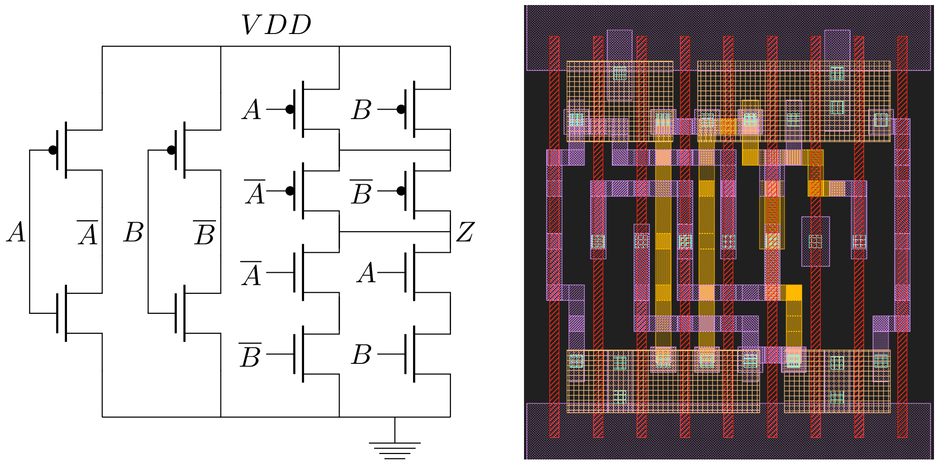


Figure 5.3: The schematic of a CMOS 2:1 MUX

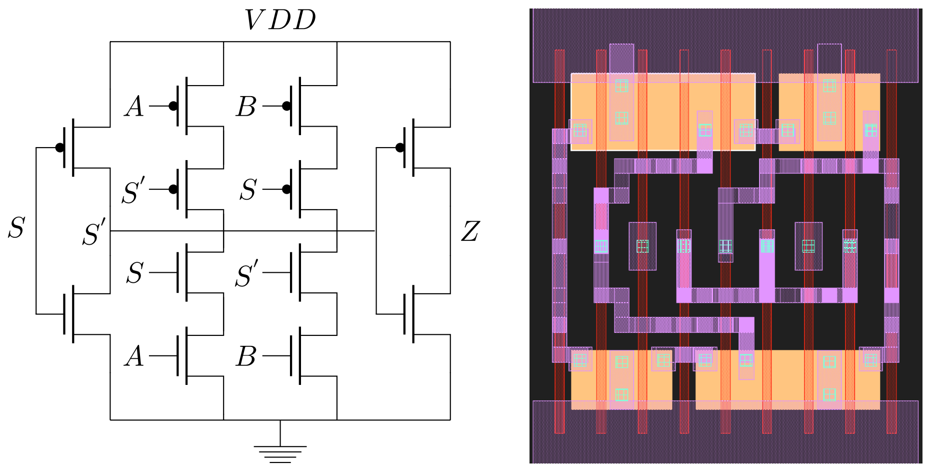


Figure 5.4: The schematic of a CMOS 2:1 MUX

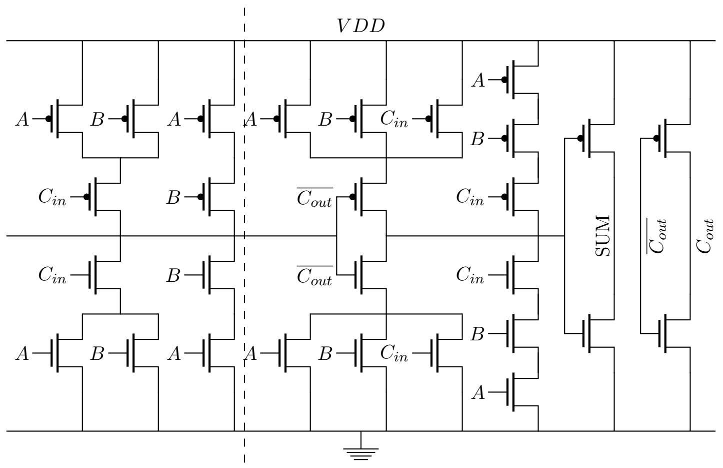


Figure 5.5: A schematic of a full adder. The dashed line shows where the problem was divided into the generation procedure.

5.1.1 Complex Standard cells

The tool has some trouble with generating very large standard cells in its current state. That problem is shown to be solvable by dividing the input netlist into multiple parts and puzzling them together. The full adder which is shown in Figure 5.6 and referenced in table 5.1 is built using such a methodology with great result. As the table shows, this standard cell was implemented using only 94 % of the area required by the industry supplied cell. There hasn't been any such puzzling functionality implemented in the tool, though, meaning the coupling of the two parts had to be done manually. It should be noted that even though I'm an inexperienced layout drawer, the connecting of the parts took less than 30 minutes and is thereby a feasible solution until a better one is added.

5.2 Comments on the results

The method presented in this thesis is shown to be able to generate most cells in a library. Its success rate decreases with larger sizes, but the tool has been shown to generate 18 transistor cells as well. It is shaped such that the routes are constrained to occupy at most the area required to fit the transistors in the logic. This, in turn, constrains the generated cell itself to always occupy the area required to place the transistors.

For cases where the cell becomes too complex for the tool, it is possible to divide the schematic of the cell into two parts for which the layouts are generated independently. These layouts can then be merged as a post-processing step to further push the boundaries of which cells can be generated. The aforementioned strategy was used to implement the logic of a full adder, which is something that requires 28 transistors. The result ended up being a great success story, allowing for a circuit using only 94 % of the industry supplied cells.

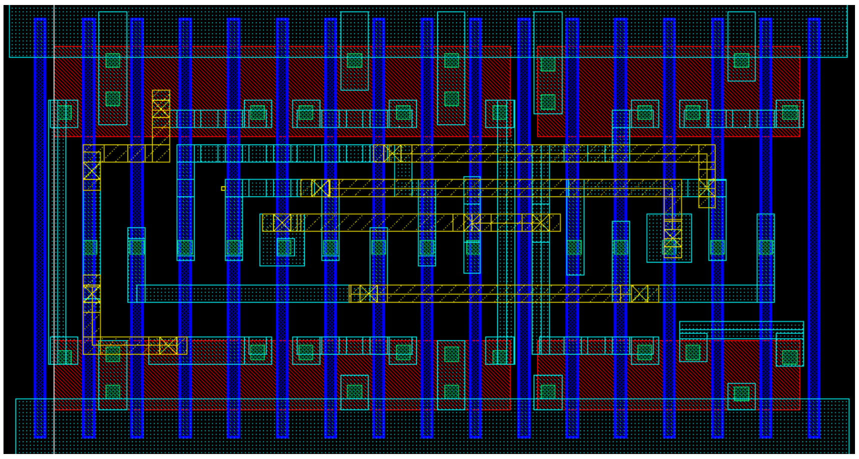


Figure 5.6: Full adder generated by manually connecting two subparts of the schematic. This layout is shown to require less area than the industry level cell supplied to us.

Some cells generated through this method, such as the XOR, requires more area than the industry standard. The reason for this is that it was found out too late in the project that cutting of the poly-layer was an alternative in layout design. By cutting the Polysilicon (poly), it is made possible to keep some euler paths in the PUN and PDN even though they're not matching. Something which would save area in the placement of the IC. It should be noted, though, that the implementation of this only would affect the placement module of the method. Because of this, it is reasonable to update it accordingly as future work without compromising compatibility with subsequent modules.

It should be acknowledged that only one process technology has been evaluated in this thesis. The design methodology is made such that it should be resilient to changes in the design rules, including sizings and routable directions. Before having actually verified it, though, there is no guarantee that the tool will work for a given process technology.

5.3 Comparison with similar research

Despite only containing a few components of the thesis, [4] is a paper that has inspired much of its final design. After a few unsuccessful initial attempts with more globally focused routing methods and fine level resolutions, this paper sparked the idea of both using a coarse grid definition and to route each point-to-point connection independently.

[12] was unfortunately found too late to have much influence on the architecture of the thesis. It is notable, though, that many of the considerations were similar to the ones had throughout this project. However, they had some interesting ideas for how to implement the placement step of their solution. The current cost function being used in this approach is very coarse, meaning that many of the

objects in the minheap hold the same priority level. This even can become a show stopper for large cells. By adding expected routability to the cost function as a runtime heuristic, the cost landscape can become much more nuanced. Furthermore, the usage of simulated annealing to rank placement strategies would reduce both the time and memory consumption for large cells compared to the minheap solution being used now. Left to evaluate is if simulated annealing also can be expected to always generate area optimized results even though it is a stochastic method. Additionally, for the routing component of their solution, they used a congestion evaluation step to modify the cost of routing at different edges. In an early version of this thesis, a congestion grid was implemented as a part of the routing component. At the time we did not manage to get desired functionality from it, which led us to comment that block away. Looking at this paper, though, we feel there might be reason to revisit that work as a continuation project.

As we look towards the latest technologies such as the 7 nm EUV process technology, the design rules will grow significantly more complex. A reason for this is that shapes produced through the EUV process can be heavily dependent on the layout around it. [14] proposes a sophisticated set of SAT-rules that not only looks at the distance between wires, but also at specific shapes in the layout that are more prone to cause unwanted behaviors.

In order for this tool to perform well for the process technologies all the way down to EUV levels, it could be good to implement such an SAT-step to verify the compatibility of the proposed routing combination here as well.

5.4 Limitations & Improvements

Despite promising results, there are still some shortcomings of the tool that should be fixed. Furthermore, some features that would improve the success rate of the tool and the quality of its generated layouts have been identified.

Automation

In order to consider the tool fully automized, the layout it generates should be ready to be implemented straight away with the promised quality. It should also be reliable in creating DRC-clean output. In its current embodiment, however, both of these factors might require manual efforts because the routes may be drawn outside the cell borders and the via structures currently aren't built in a robust enough manner. In order to mitigate the boundary issue, a future desired feature of the tool is to add a post-processing step to squeeze the outer wires into the cell such that they fit inside the cell boundaries. This may require some pushing of some other wires as well to follow the DRC-rules. Something which can be fairly intuitive for a human. However, this task might prove quite challenging to generalize in a program, meaning this may become a small project of its own. To address the via structure issue, the primary intention is simply to further improve the decision tree that decides which structure to place around the via.

Furthermore, for the sakes of automation, functionality for separating the schematic of difficult cells into multiple components and reattaching the resulting layouts should be implemented into the tool. This could be designed as an immediate fall back step as soon as the primary algorithm fails. Given that the layouts will have continuous connection points, the routing problem it creates may require another routing algorithm.

Success rate

One more goal is to increase the number of cells that can be designed. Here, we primarily encounter problems in the routing part of the tool. The main reason for failure is that the amount of possible routes increases exponentially with the distance being routed over. For long connections, this results in that most generated routes are going to be very similar to each other, thus reducing the flexibility of the connections when combining it with others. For this problem, two combinable solutions have been identified. One is to add an intermediary node for distant connections before initiating the routing. By routing to the intermediary node from both sides, the variability in the routing suggestions should be increased significantly. The other idea is to make the tool congestion aware. By adding a cost function of how many nets are trying to route in a given point, the routes can be iteratively updated to spread out away from each other. The congestion information can also be used to intentionally place the intermediary nodes in less congested areas.

Quality

To improve the quality of the outputs, two proposals have been made. One is to add support for poly cutting already in the placement phase of the design. This would enable the tool to further minimize the area further when there is a mismatch in gate sequences of the PUN and PDN. The existing algorithm for transistor placement is already very dynamic in its architecture, so implementing this is not expected to be a very demanding task. Additionally, automizing the sizing of the transistors depending on the stacking in the schematic would make the layout further improved. Also, this is expected to be a fairly easily integrated improvement.

EUV

As we look towards the smaller process technologies, the DRC-rules will become increasingly complex. Particularly the EUV process holds a sensitivity to how the layout looks in the vicinity to the structure it's currently drawing [14]. The tool in its current state does not support consideration for this type of effects. Due to the modular nature of the tool, however, this could seamlessly be integrated as an additional SAT definition in the route selection step, not unlike what is done in [14].

Chapter 6

Conclusion & Future Work

6.1 Final words

In this thesis we have presented a method for the complete layout design of a standard cell, using only a netlist and a few elementary design rules. The method is shown to generate layouts with similar areas to industry level designs and have even been shown to outperform some industry-level cells. In its current state, the layouts produced by the tool still require some manual oversight. However, the features that are needed to make the tool truly independent have been identified and are intended to be implemented as a continuation of the project.

The speed of which this tool generates standard cells along with the quality of its results leads me to believe that it has the potential to become industrially competitive and that it can reduce manual workload within the field significantly.

6.2 Future work

This is a project that can be grown to a very large extent. Because of this, the list of future work and improvements can grow long as well. However, some of the most important points for making this tool competitive in the industrial domain have been identified and are listed below.

- Build a post-processing step to squeeze the outer wires towards the center. While fairly easy to edit manually afterwards, this step is essential to make the tool fully automatic without unnecessary area usage.
- Add support for poly cutting in the placement component of the tool.
  Poly cutting enables an extra level of area optimization that would not be possible if there is a mismatch in the Euler paths of the PUN and PDN.
- Improve routability for long connections by adding intermediary nodes.
  By cutting the long routes in half we can counteract the route complexity that grows exponentially with distance.
- Automate transistor sizing.
  By automizing transistor size based on the stacking in the schematic, the standard cell quality can be even further improved.
- Improve via decision tree.
  The via algorithm needs to be more considerate to its environment when placing its structures to avoid causing any DRC-violations.
- Make routing step congestion aware.
  By making the routing algorithm congestion aware, we can get more placeable routes already from the beginning.

References

[1] Gordon E. Moore. Cramming more components onto integrated circuits. 4 1965.

[2] Mohammad Maadi. Evaluation of spice p-n junction's, bjt's and mosfet's model parameters. 10 2013.

[3] Anil Bahuman. An evolutionary approach to standard cell design automation. 09 2001.

[4] N. Ryzhenko and S. Burns. Standard cell routing via boolean satisfiability. In *DAC Design Automation Conference 2012*, pages 603–612, June 2012.

[5] Cristiano Lazzari, Lorena Anghel, and Ricardo Reis. *A Transistor Placement Technique Using Genetic Algorithm And Analytical Programming*, volume 240. 01 2005.

[6] R. Jacob Baker. *CMOS: circuit design, layout, and simulation.* Wiley-IEEE, 2 edition, 2008.

[7] Daniel Nenni. A brief history of eda. 5 2012.

[8] IEEE. 1076-1987 – ieee standard vhdl language reference manual. 1988.

[9] Department of Defense. Military standard, standard general requirements for electronic equipment. 1992.

[10] Ming-Bo Lin. *Introduction to VLSI systems: A logic, circuit and system perspective.* CRC Press, 11 2011.

[11] Kuntal Roy. Optimum gate ordering of cmos logic gates using euler path approach: Some insights and explanations. *Journal of Computing and Information Technology*, pages 85–92, 2007.

[12] Yi-Wei Lin, Malgorzata Marek-Sadowska, and Wojciech Maly. Transistor-level layout of high-density regular circuits. In *Proceedings of the 2009 International Symposium on Physical Design*, ISPD '09, pages 83–90, New York, NY, USA, 2009. ACM.

[13] A. Khachaturyan, S. Semenovsovskaya, and B. Vainshtein. The thermodynamic approach to the structure analysis of crystals. *Acta Crystallographica Section A*, 37(5):742–754, Sep 1981.

[14] Nikolay Ryzhenko, Steven Burns, Anton Sorokin, and Mikhail Talalay. Pin access-driven design rule clean and dfm optimized routing of standard cells under boolean constraints. In *Proceedings of the 2019 International Symposium on Physical Design*, ISPD '19, pages 41–47, New York, NY, USA, 2019. ACM.

[15] O. Mason and M. Verwoerd. Graph Theory and Networks in Biology. *Hamilton*, Mar 2006.

[16] Frank Harary and Robert Z Norman. *Graph theory as a mathematical model in social science.* University of Michigan, Institute for Social Research Ann Arbor, 1953.

[17] Luciano Lavagno, Igor L. Markov, Grant Martin, and Louis K. Scheffer. *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology.* CRC Press, 2016.

[18] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, Dec 1959.

[19] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *25th Annual Symposium onFoundations of Computer Science, 1984.*, pages 338–346, Oct 1984.

[20] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[21] Dustin Ostrowski, Iwona Pozniak-Koszalka, Leszek Koszalka, and Andrzej Kasprzak. Comparative analysis of the algorithms for pathfinding in gps systems. *ICN 2015*, page 114, 2015.

[22] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401.

[23] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.