Physical design automation

automatic placement and routing for analog chip designs

van Otterdijk, P.

Award date:
2011

EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science

Physical Design Automation

Automatic Placement and Routing for Analog Chip Designs

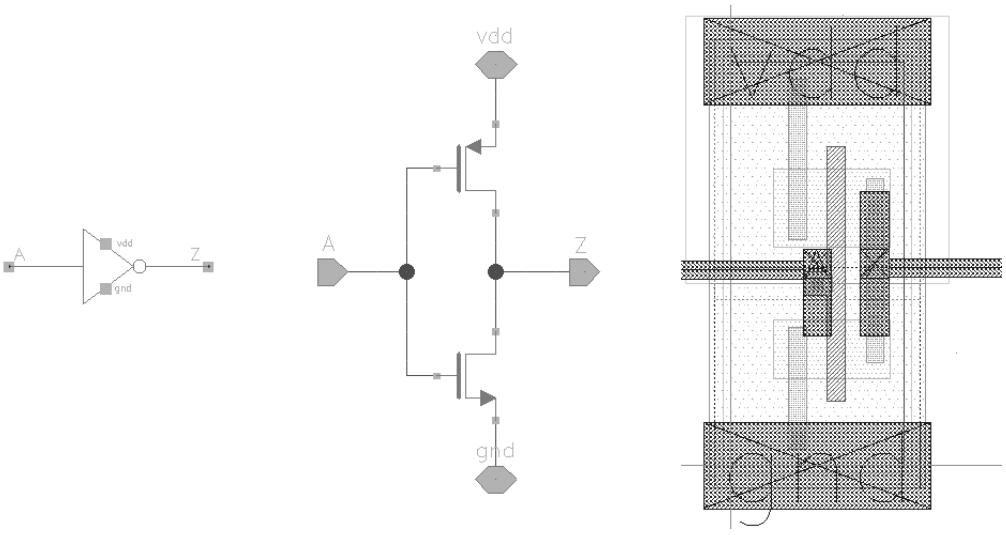
P. van Otterdijk
p.vanotterdijk@student.tue.nl
0664813

September 5, 2011

Contents

1 Introduction

In the process of designing an electrical chip, typically a schematic design of the chip is made before the layout design is made. The schematic design is an abstraction of the layout design. In a schematic design electrical components like resistors, capacitors, diodes, transistors, etc. are drawn as symbols and connected to each other in order to realize a certain electrical behaviour of the chip, see Figure 1b. This schematic design is then translated to a layout design.



(a) Symbol (b) Schematic (c) Layout

Figure 1: Three different views of an inverter.

In a layout design each component is depicted by its physical representation, consisting of a number of elementary objects, see Figure 1c. In the physical representation the position, size, shape and material of these objects together determine the electrical behaviour of the component. Note that materials are indicated by colors in a layout design. The different views (symbol, schematic, layout) of a component are captured in an abstract object, a so-called cell. In a layout design the physical representation of a component is given a position on the area of a chip. The connections between cells as specified in the schematic design are realized by thin metal strips in the layout design. These strips are called wires.

Placement and routing are two steps of the design process. When creating a layout design from a schematic design, placement refers to the process of assigning locations to the cells on a chip area, and routing refers to the process of making the required connections between the different cells. The placement and routing of a layout design have to obey certain design rules. These rules specify all sorts of requirements in order to be able to manufacture a chip correctly, e.g., the minimal distance between two components in a layout design. Besides obeying these design rules, the layout design can also be limited by for example the available chip area and total wire length.

Big differences exist between the design process of analog and that of digital chips. For digital chip designs the placement and routing steps are automated processes. However, for analog chip designs placement and routing are mostly done by hand, since there are many requirements which have to be taken into account, for example, alignment, overlap, matching, maximum distance between components, etc. These requirements make it difficult to automate the placement and routing processes.

After a design has been made simulation runs can be used to estimate the resulting electrical behaviour of the chip. Using simulation results of a schematic design or a layout design, the performance of a chip design can be evaluated and optimized. First the schematic design is simulated and optimized using those simulations. Then the layout design is created and simulated. Optimization of some aspects of the electrical behaviour of the chip can be realized by changing the layout design, whilst others involve changes in the schematic design. Changes in the schematic design (after creating the layout design) in turn lead to changes in the layout design. Every time such a change is made, the placement and routing of the layout design needs to be re-evaluated. In order to make layout designs flexible and adaptive a method is needed that can automatically adjust the placement and routing of a layout design.

Automating manual tasks like placement and routing will provide a considerable speed-up of the iterative optimization and re-design process of analog designs. If the placement and routing processes of an analog design could be automated, changes in the schematic design could automatically be translated to a new layout design and simulated. In this way, improvements of an analog design could be done much faster.

The goal of this project is to develop a method for the automatic placement and routing of analog designs, and to produce a prototype tool which can handle a subset of the placement and routing requirements found in analog designs. A new technique will be introduced that solves the placement and the routing problems simultaneously and to the best of our knowledge this has not been done before. It should be noticed that solving both problems one after another is not the same as solving both problems simultaneously.

The following sections will give an introduction to the abstract placement and routing problems.

1.1 Placement

The placement of components on a chip is typically a three dimensional problem, since there can be several layers in a chip in which components can be placed and cells consist of several layers. A simplified version of the placement problem is obtained by restricting the problem to two dimensions where cells are represented as rectangles which are to be placed in a rectangular chip area, see Figure 2. Rotating the cells by a multiple of 90 degrees is allowed. The goal is to position all cells in the chip area, such that all requirements are met. The requirements can consist of rules which specify relations between cells, such as no-overlapping, minimal spacing, relative placement, alignment or matching. These requirements may be needed for a resulting placement to obey the design rules, to improve performance and to ensure route-ability (the possibility for the routing process to create all the connections). These requirements will be explained in more detail in section 2.2.

It should be noticed that for the placement problem with a given set of cells and an area, multiple valid solutions can exist, as is illustrated in Figure 2. A solution could be chosen based on additional criteria like the spacing between cells, or the minimal area used by a given placement. However, only looking at the placement of the cells is not sufficient. The resulting placement still has to be routed. If an acceptable routing can not be created for a given placement of the cells, maybe it can when applying a different placement. Section 1.2 will give an introduction to the routing problem.

Placement has been studied in various papers. The book by Wang [14] summarizes a number of different approaches and topological representations. In this project we use a mathematical programming method where relations between cells are described using a set of constraints. Constraints

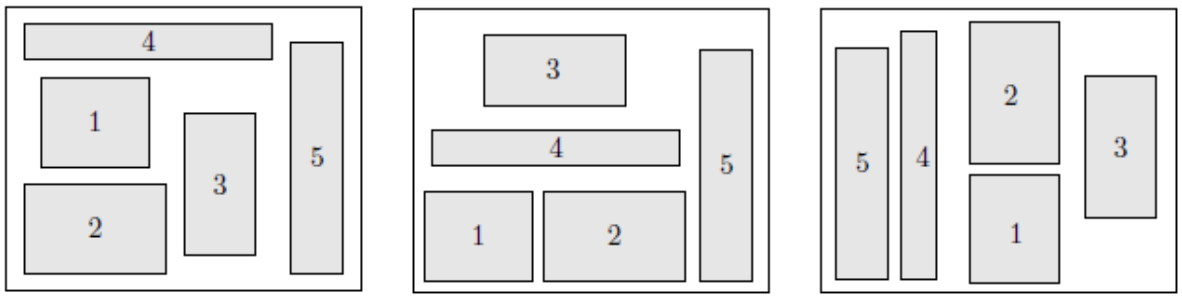


Figure 2: Example of three different solutions for placing five rectangles in a predefined area. The rectangles are only moved and/or rotated.

consist of linear inequalities in combination with logical expressions. Constraints can describe properties like *alignment* or *no overlap* between cells. Similar work has been presented in [17], where linear programming was used to describe these kinds of relations between cells.

1.2 Routing

In a schematic design cells are connected to each other. In a layout design cells have to be physically connected, corresponding to the schematic design. Routing is the process of creating the required connections between the cells, see Figure 3. As explained a layout view of a cell consists of a number of elementary objects. When a wire between cells has to be made, it can not be done just anywhere on the cell, but to a specific object in that cell, the so-called pin. Each pin corresponds with a connection to a symbol in the schematic design. Thus using the schematic design, one can determine which pins have to be connected to each other. This information is generally also available in the so-called Netlist, which describes both the pin locations and which pins are connected to each other.

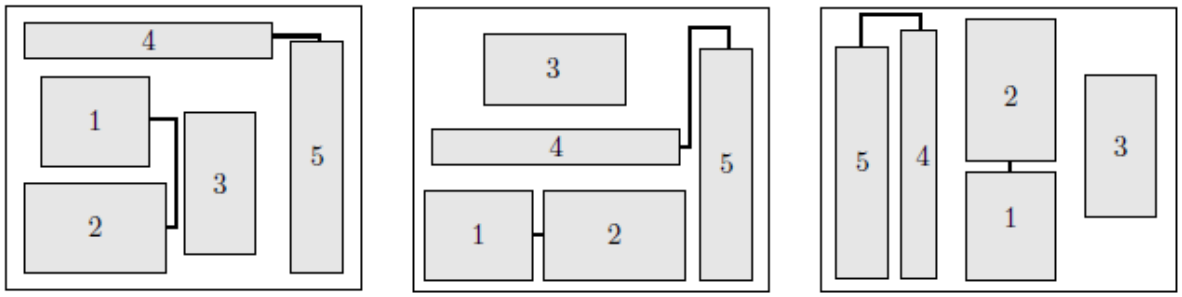


Figure 3: A connection between cells 1 and 2, and between cells 4 and 5 has been realized for three different solutions of the placement problem.

In the design process routing is usually done after placement. Routing generates the connections between the pins of different cells by creating a metal strip, called a wire. Generally the wire is a Manhattan-path, which means that only horizontal and vertical wire parts are allowed (though there are situations in chip design where a diagonal wire at the angle of 45 or 135 degrees is allowed, but these are not considered in this project).

To make a well working chip, routing should ensure that all required connections are properly realized, without creating any unintended connections (or short cuts), i.e., the wires may not cross each other. Analogously to placement, routing also has to obey the design rules. Besides that routing also needs to take into account additional requirements, with respect to aspects like the number of jogs per wire, the total capacitance of a wire or the total wire length.

As can be seen in Figure 3, the routing of the wires depends on the placement of the cells. If additional requirements on the wires are to be taken into account, then the placement of the cells should also adapt to these requirements. Solving both the placement and the routing problem simultaneously makes this possible.

2 Scope of this study

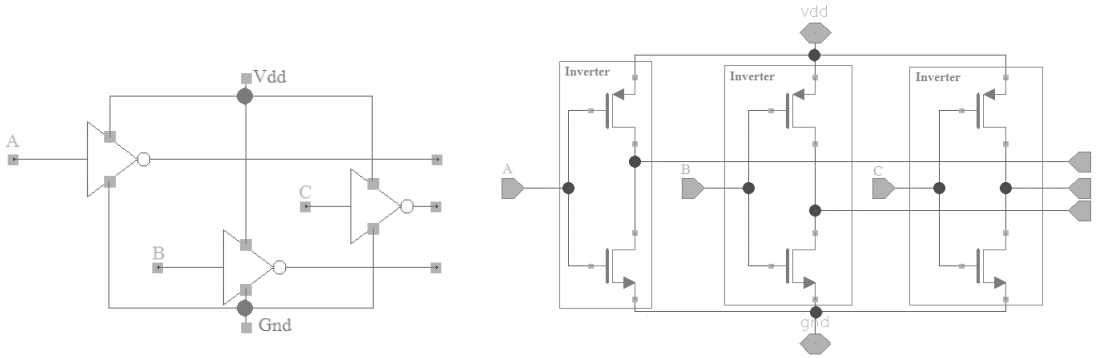
As said earlier currently in the design process of a chip, placement and routing are sequential processes. First the cells are placed, then the wires are generated (routed). This has been reported in e.g., [7] and [20]. Since the quality of the routing depends a lot on the placement of the cells, it could be more effective to do the placement and routing simultaneously, since then the placement of the cells can be made such that it favours the routing. Therefore we want to develop a method to solve the two problems as a single problem.

The following sections will describe the hierarchy in a design, give a detailed problem description of the placement and routing problems, and describe the design rules and limitations of our research.

2.1 Hierarchy in a design

In this research we look at the physical representation of a component as being a cell. Moreover a group of components could also be modelled as if it were a cell. For example, with two transistors we can create an inverter, see Figure 1. In this case the transistors are seen as cells and are placed and routed on a chip area. Three inverters together form a three-phase inverter, see Figure 4. So, the individual inverters can be seen as cells and they are placed and routed on a chip area, each containing an underlying problem of placing and routing two transistors. The three-phase inverter itself can also be modelled as a cell and be placed and routed on a chip area accordingly. In this way hierarchy is created in a design.

The use of hierarchy is common in chip design and is helpful for solving the placement and routing problems. The placement and routing problems are both NP-complete ([11] and [18]). By introducing hierarchy in a design, the size of the problem can be limited and so the run-time due to NP-completeness can be restricted. Any group of components can be modelled as a single cell, containing a underlying placement and routing problem, which can be solved as an independent problem. Note that using hierarchy in a design does not guarantee optimal solutions.



(a) inverters drawn as symbols

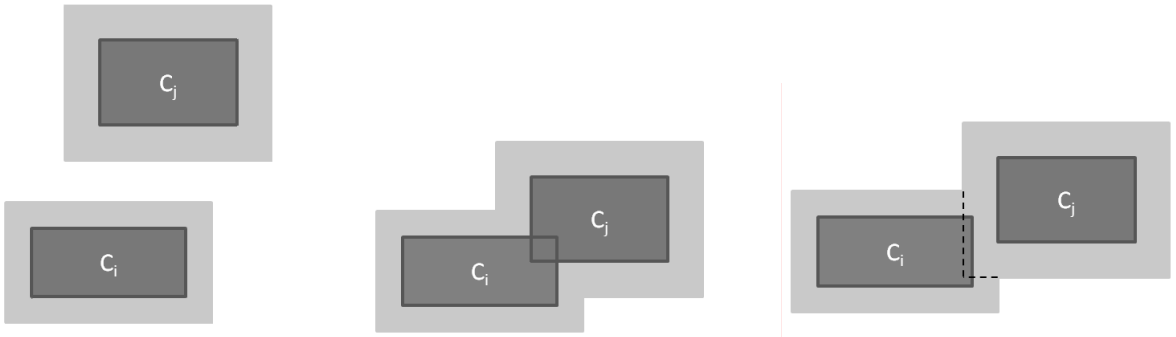
(b) inverters drawn as schematics

Figure 4: Simplified schematic of three inverters put together to form a three-phase inverter.

2.2 Placement requirements

The placement problem is introduced as a two dimensional problem where a number of rectangular cells needs to be placed in a rectangular area.

The width and height of the cells are defined by cell parameters. For now we assume they are given. As stated before the cells can not be placed just anywhere on the chip area. There are additional requirements which need to be respected. They result from the design rules and specified relations between cells, e.g., no-overlapping, margins, matching, alignment, merging. The following list describes these requirements on cells:



(a) No overlap (valid)

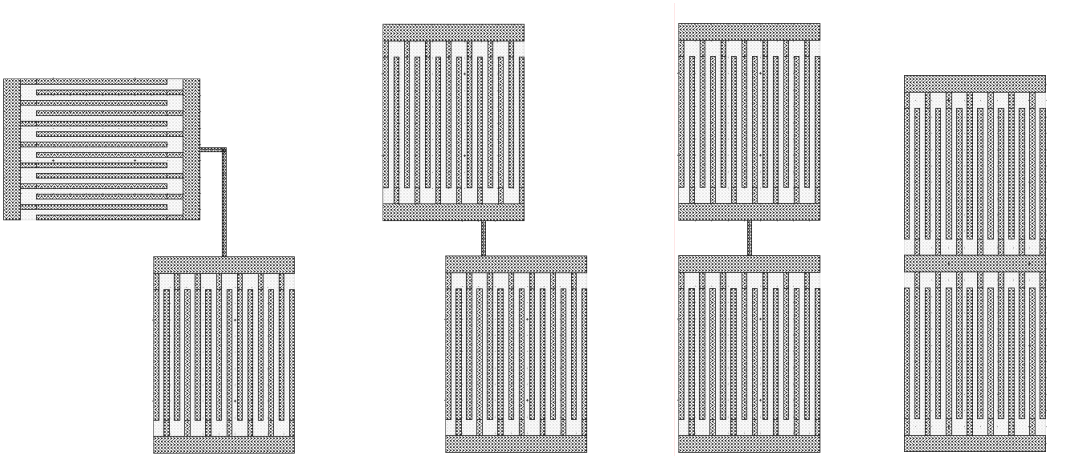
(b) Cells overlap (only valid when merging)

(c) Cell overlaps with a margin (not valid)

Figure 5: Three different ways to place two cells. Each cell is drawn with a surrounding margin.

- No-Overlap: The no-overlap requirement states that two cells can touch but not overlap each other, see Figure 5.
- Margins: To ensure that a layout design can be manufactured correctly, margins must be defined around cells, see Figure 5. Cells may not intersect the margin of other cells (except when merging two cells, see below). For simplicity we allow one single margin around each cell. This can easily be extended to cases with different margins per side of the cell.
- Matching: A set of cells is said to match if they have the same angle of rotation in the resulting layout design. A matching relationship is used to enforce similarity in a resulting layout design. This similarity can result in an even propagation of errors in a design or simply to improve the quality of the layout design. Figures 6a and 6b illustrate the difference between two non-matching and two matching capacitors.
- Alignment: Cells can be aligned either horizontally or vertically and in different ways, see Figure 7. Alignment is discussed in more detail in [5]. Similar to matching, alignment can be used to force similarity in a design. Figure 6c shows the improvement of alignment by aligning the two capacitors.
- Merging: A cell can have an area that may have overlap with an area of another cell. There are specific design rules which limit the possible situations where this can be done, but if it is possible then the amount of area occupied by those two cells can be reduced. Two cells can only overlap if these cells need to be connected to each other at the overlapping area. Figure 6d shows an example of two capacitors which can be placed such that they partially overlap.

Besides the requirement on cells, other requirements (e.g., related to chip area) can be imposed. Requirements on the chip area can involve aspect ratio, maximum perimeter or maximum surface



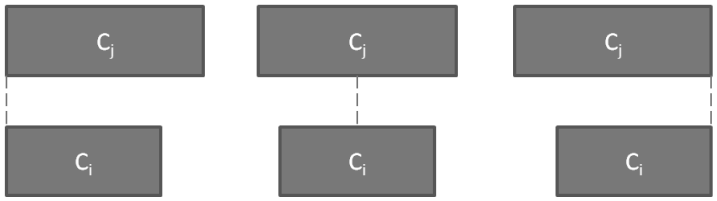
(a) Two cells, neither matched nor aligned

(b) Two cells are matched

(c) Two cells are matched and aligned

(d) Two cells are matched, aligned and merged

Figure 6: Two capacitors connected by a wire. The placement is improved by adding requirements for matching, alignment and merging.



(a) Left (b) Center (c) Right

Figure 7: Example of three different ways of aligning two cells

area. The connectivity between cells can also be seen as a different type of requirements. This topic will be discussed in section 2.3.

2.3 Routing requirements

The routing process has to create a wire for each connection between the pins of cells as defined in the schematics. This has to be done in such a way that all the requirements on all the wires are met. A requirement on the wires can result from the design rules, or it can be specified as a requirement for this specific design. For example the design rules specify the minimal wire width and the minimal distance between two wires. Additional requirements can result from relations between wires, such as, mirror-symmetry and parallel wires, see Figure 8. Two other requirements are the maximum wire length and the maximum number of jogs in a wire.

One of the difficulties in routing is that obstacles need to be avoided. Obstacles can be cells or other wires. Notice that once a wire is placed this wire becomes an obstacle for the next wire that has to be created.

The quality of the routing can be measured using a well-defined criterion (objective function). This objective function reflects a trade-off between a number of different aspects per design. The objec-

tive function can differ depending on what aspect and which wire is most important in that design.

An example of a simple routing problem is given in Figure 9, where two cells are placed and a connection between their pins is realized in two different ways. The wire length in both solutions is the same, but the solution in Figure 9b has more jogs in the wire than the solution in Figure 9a. Though both solutions are valid, one might be more desirable than the other.

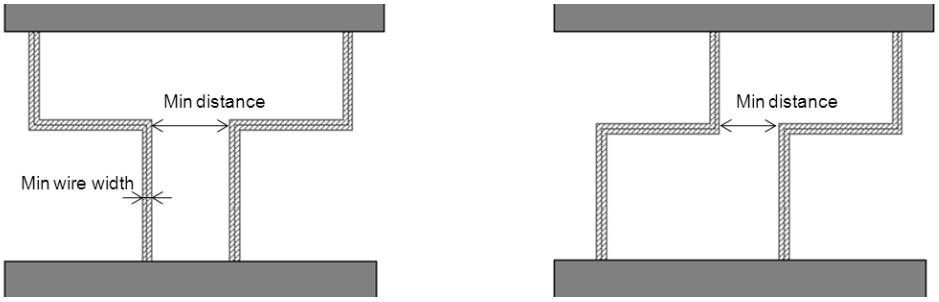


Figure 8: An example of two mirror-symmetric wires and two parallel wires

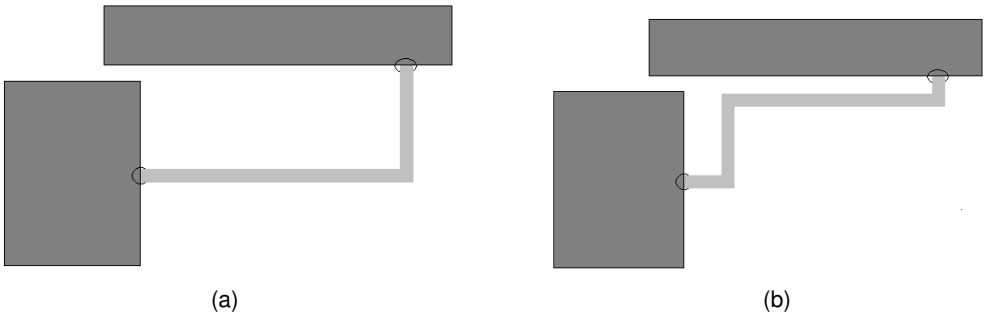


Figure 9: Two routing examples with the same wire length. (a) has fewer jogs than (b), but (b) leaves more available area than (a), assuming that the space between the cells and the wire is not available any more.

2.4 Design Rules

For a chip design certain design rules have to be taken into account. For each process technology there is a different set of design rules. These design rules specify a minimum bound and in some cases a maximum bound for almost every geometrical property of an object. Furthermore it specifies minimal distances between objects depending on the size, shape, materials, or other properties of those objects. This results in many different requirements which are all intrinsically the same. To give a flavour of what the design rules look like, Table 1 shows a quotation of a small section of the design rules of the CMOS14 technology.

The production process of a chip has a certain accuracy. Basically by obeying the design rules, we ensure that the layout design contains nothing smaller than the accuracy of the production process. Therefore a design must always obey the design rules. Beside the design rules there are additional rules which are not mandatory but obeying those rules will improve the robustness of the resulting layout design, and allow a margin of error in the production process. In our research only the requirements already mentioned in the previous sections will be considered.

Rule	Distance (in μm)	Description
METAL1S.2.1.1	0.256	Minimum METAL1S space
METAL1S.2.1.4	1	Minimum space between METAL1S (METAL1S in seal ring) on SOI
METAL1S.2.1.5	0.64	Minimum space between METAL1S in a different voltage domains, except METAL1S inside pnplESDEHV
METAL1S.2.1.6	0.49	Minimum METAL1S space inside pnplESDEHV

Table 1: Quotation of the CMOS14 design rules about the minimal distance between two metals in layer 1. Different cases are needed since certain properties influence the fault tolerance.

2.5 Limitations

As stated before the placement problem is limited in two dimensions and cells are seen as rectangular objects. Wires are considered to be a Manhattan-path, i.e., no diagonal wires are allowed (which in practise is not necessarily the case). In our research we limit ourselves to allow only two pins per wire and only, where in practice more than two pins can be connected with a single wire. As stated before we allow routing in two dimensions, where wires may not cross. We only consider pins placed on a cell, where in practice pins can also be defined as a given point on the chip area.

In general the wires are not restricted by cell boundaries, meaning that it is allowed to place a wire through a cell if it does not cause any violations with the design rules. So unused space inside the cell boundaries can still be used for routing. For simplicity we assume routing may not cross cell boundaries, and it may not intersect cell margins, except when a connection has to be made at that location. Note that this does not change the problem, since the objects in a cell can simply be modelled as if they are cells. Thus this assumption is only a limitation on the problem size, not on the problem itself.

Optimizations of the resulting layout will only be done with respect to the total number of jogs in the wires of the layout design or the total wire length of all the wires in the layout design.

3 Solving a placement and routing problem

This section will introduce the method used to solve the placement and routing problems simultaneously.

In our research we have chosen to focus on an analytical approach where a constraint programming method is used. A placement and routing problem will be expressed in mathematical inequalities combined with logical expressions. Requirements can be formulated using such expressions, e.g., for No-Overlapping we have:

$$(X_1 + W_1 \leq X_2) \vee (X_2 + W_2 \leq X_1) \tag{1}$$

Expression (1) states that a cell C_1 with corner point X_1 and width W_1 can touch but not overlap (on the X axis) with a cell C_2 specified with corner point X_2 and length W_2. Figure 10 illustrates a no-overlap relation (1) where the first inequality is true. In our application we assume the values for W_1 and W_2 are given and that a solution to the placement and routing problem generates values for X_1 and X_2, such that the expression holds.

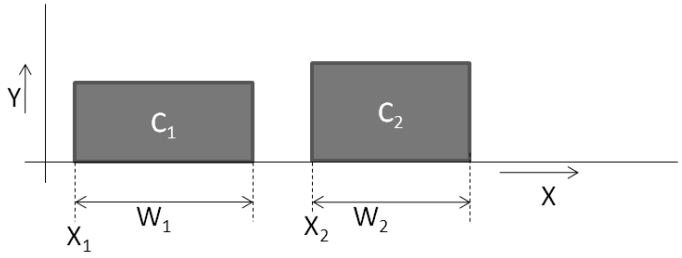


Figure 10: Example of two cells, not overlapping on the X axis.

A way to formulate mathematical inequalities combined with logical expressions is by using the language defined in the SMT library (SMT-LIB) [16]. We will use SMT-LIB, since it provides us the SMT-solver Yices [8] to solve placement and routing problems.

The following sections will give a description of SMT and an example showing how a problem is formulated using the SMT-LIB and a short description of the advantages and limitations of using SMT and Yices.

3.1 Satisfiability Modulo Theories (SMT)

Propositional satisfiability is the problem of determining if the variables in a boolean expression can be assigned in such a way that the expression evaluates to true. Satisfiability Modulo Theories (SMT) is an extension to propositional satisfiability. There are different theories that can be used. We focus on the usage of real numbers and linear inequalities, since most aspects of the placement and routing problems can easily be expressed using such linear inequalities.

The SMT-LIB [16] defines a common language (SMT) and a set of benchmarks which are used by SMT-solvers. A problem formulated in SMT is called an SMT-problem. An SMT-problem contains a number of variables and a single expression. An SMT-problem is called satisfiable if a value can be assigned, to every variable in this model, such that the expression is true. If this is not possible, then the SMT-problem is called unsatisfiable. Currently tools are already available that can prove satisfiability of an SMT-problem. These tools are called SMT-solvers, for example Yices [8]. If a

satisfying solution to an SMT-problem exists, an SMT-solver will prove this by giving an assignment for all the variables such that the expression is true. Thus if the placement and routing problem is expressed in an SMT-problem, and an solution exists, then an SMT-solver will be able to provide such a solution.

Different tools can be used to solve an SMT-problem, e.g., Yices [8], Boolector [2], MathSAT [3], OpenSMT [4] and SatEEn [13]. These tools mainly differ in the algorithms they use to solve the given problem. A study to which tool is most efficient for our application is out of the scope of this project. We have chosen to use Yices.

There are other somewhat similar languages like SMT. Some of them provide additional features. For example in the Optimization Programming Language (OPL) [19] expressions consisting of linear inequalities, logical expressions and universal quantifications can be modelled. An OPL-model can be solved using a tool called CPLEX [12]. The main additional feature of OPL and CPLEX is that it searches for an optimal solution, instead of just a solution. To determine what is meant by optimal a well-defined function has to be specified. Instead of studying with optimality issues, we have chosen to focus on expressing the combination of the placement and routing problems, by using a simple SMT approach. Adding additional optimizations could be the next step.

3.2 A simple example of an SMT-problem

Suppose (1) is extended with cell widths of 5 and 10, then we get:

$$(W_1 = 5) \wedge (W_2 = 10) \wedge ((X_1 + W_1 \leq X_2) \vee (X_2 + W_2 \leq X_1)) \tag{2}$$

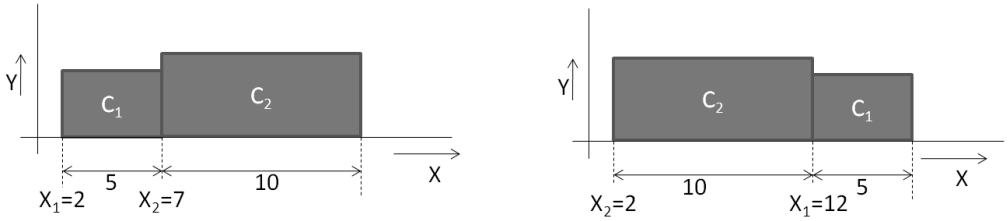
Clearly a number of satisfying solutions to this expression exists, for example $X_1 = 0$ and $X_2 = 5$. To show an SMT-solver can provide such a solution this expression in translated to an SMT-problem. Table 2 shows part of the SMT-problem, where (2) is stated according to the rules specified in SMT-LIB (using Polish notation [15]). For the full SMT-problem see Appendix A. Applying Yices to this SMT-problem gives the output as shown in the right column, indicating that $W_1 = 5$, $W_2 = 10$, $X_1 = 0$ and $X_2 = 5$ satisfies the expression. Notice that this solution is not unique.

Code snippet:	Result:
(and (= W1 5) (= W2 10) (or (<= (+ X1 W1) X2) (<= (+ X2 W2) X1)))	sat (= W1 5) (= W2 10) (= X1 0) (= X2 5)

Table 2: Example of (2) expressed in SMT (using Polish Notation), and the satisfiability provided by Yices.

We will show how a number of different requirements can easily be combined, by extending this example with a requirement specifying the region in which the cells must lie. To state that the cells must lie between 2 and 17 on the X-axis expression (2) has to be extended with: $(2 \leq X1) \wedge (2 \leq X2) \wedge (X1 + W1 \leq 17) \wedge (X2 + W2 \leq 17)$. This leaves two possible situations, as shown in

Figure 11, where C_1 lies left of C_2 ($X1 = 2$ and $X2 = 7$) or where C_2 lies left of C_1 ($X2 = 2$ and $X1 = 12$).



(a) C_1 left of C_2 (b) C_1 right of C_2

Figure 11: For the given cell widths of 5 and 10, and restriction of the cell location between 2 and 17, only 2 combinations remain for the values of X_1 and X_2.

3.3 Advantages and limitations of SMT and Yices

Most design rules and requirements of placement and routing can easily be translated to logical expressions and linear inequalities, see Table 3. Once this translation is made, a solution can be found using Yices. The advantage of SMT and Yices is that it is not necessary to study the underlying algorithms used to solve the SMT-problem. How the requirements are translated to expressions and linear inequalities is described in section 4.

The main limitation of using SMT and Yices is that non-linear expressions are not supported. For example, an aspect ratio can easily be expressed and handled by SMT and Yices (e.g., $W \times 2 \leq H$), but a limitation to the surface area can not be handled by SMT and Yices (e.g., $W \times H \leq 6$). This makes requirements with respect to surface area difficult. However it is possible to construct a linear approximation of the surface area.

We finally remark that there are variates to Yices that do support non-linearities [21].

Requirement:	Is implemented	Section
Areas to place cells in	Yes (only one)	4.1
Blocking areas	Yes	4.1
Aspect ratio of an area	Yes	4.1
Limitation on perimeter of an area	Yes	4.1
Grid size	No	
Fixed location of cells	Yes	4.2
Rotating of cells	Yes	4.2
No-overlapping cells/wires	Yes	4.3, 4.5
Matching cells	Yes	4.3
Aligned cells	Yes	4.3
Relative Placement of cells (C_i can be forced to be placed left to C_j)	No	
Mirror symmetric cells	No	
Merging	No	
Min. distance between cells (Margin)	Yes	4.3
Max. distance between cells	No	
Pins on cells	Yes	4.4
Pins in chip area	No	
Fixed wires	No	
wires between two pins	Yes	4.4
wires between three or more pins	No	
Manhattan wires	Yes	4.4
Diagonal wires	No	
Mirror symmetric wires	No	
Parallel wires	No	
Limitation on wire lengths	Yes	4.6
Limitation on number of jogs	Yes	4.6
No routing areas	No	

Table 3: A number of requirements, which can be implemented in SMT, and a selection of those which have been implemented in our prototype tool.

4 Specifications of requirements

Our purpose is to find a solution to the placement and routing problem by expressing a set of variables and requirements in SMT. If an assignment for the variables which satisfies all requirements is found, then this assignment is considered a valid solution to the placement and routing problem. This section describes different types of requirements and their translation to an expression. Since we want all the requirements to be satisfied we can combine them to a single expression using logical conjunctions. When contradicting requirements are specified, then the conjunction of the expressions will contain a contradiction and therefore the SMT-problem will become unsatisfiable, and no solution will be found.

Note that the SMT-problem is generated automatically from the requirements, by a tool created in this project, see Appendix B. The requirements have been divided into different types and will be described in the following sections: specification of the chip area, specification of a cell, relations between cells, specification of routing, wire relations and wire restrictions.

4.1 Specification of the chip area

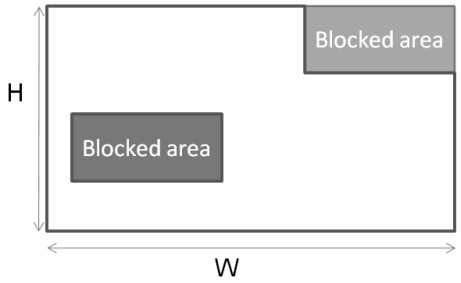


Figure 12: Illustration of the available chip area, with blocked areas.

The boundaries of a chip can be specified by a rectilinear polygon. Possibly there are blocked areas in this area, which means no objects may overlap these blocked areas. Figure 12 illustrates such an area. In our research this rectilinear polygon area is modelled as a rectangular area, with a number of blocked areas. A blocked area is also modelled as a rectangular area and can be mimicked by placing a dummy cell on this blocked area. Later we will show how to prevent other cells from overlapping with these artificial cells.

The size of the available chip area is defined by its width (W) and height (H). These can be fixed or variable. If the chip area is variable, requirements on the aspect ratio or on the perimeter of this area can easily be defined. For example:

- Aspect ratio: $0.5 \times W = H$
- Perimeter: $2 \times (W + H) \leq 15.4$

4.2 Specification of a cell

A cell is defined as a rectangular area, specified by its location, rotation angle and size. The location a of cell C_i is indicated by its lower left corner (X_i, Y_i), its rotation angle by R_i and its size by its

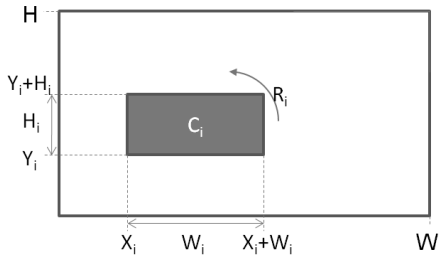


Figure 13: Example of a single cell to be placed in the chip area.

width (W_i) and its height (H_i), see Figure 13. The location and rotation of a cell can be fixed or variable. The size of a cell must always be given.

In practice rotation of cells is allowed by a multiple of 90 degrees, so four possible situations can be distinguished: $0, 90, 180$ or 270 degrees. Either one of these rotation angles can be chosen. Assume w_i and h_i are actual values for the width and height of cell C_i. In case $R_i = 0$ degrees cell C_i can be defined by $W_i = w_i \wedge H_i = h_i \wedge R_i = 0$. The situation with $R_i = 90$ degrees can be defined $W_i = h_i \wedge H_i = w_i \wedge R_i = 90$, see Figure 14. Similarly cases for 180 and 270 degrees can be defined, each time swapping w_i and h_i, and filling in the new rotation angle.

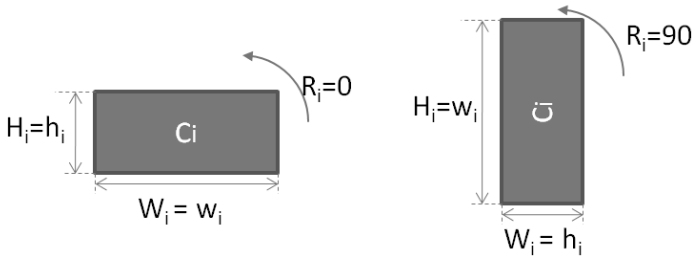


Figure 14: Specification of the Width (W_i) and Height (H_i) of a cell C_i under different rotation angles.

To ensure a cell C_i is always placed inside the chip area the constraints $X_i \geq 0, Y_i \geq 0, Y_i + H_i \leq H$ and $X_i + W_i \leq W$ have to be defined, see Figure 13.

4.3 Relations between cells

In this section we will describe the following different relations between two cells: No-Overlap, Margin, Matching and Alignment.

- No-Overlap: The No-Overlap relation states that two cells can touch but not overlap each other. Figure 15 shows two cells C_i and C_j with annotations. C_i and C_j are not overlapping if one cell C_i is above, left, right or below of the other cell C_j. For example, C_i is below C_j if (3) holds.

  $$Y_i + H_i \leq Y_j \tag{3}$$

  Similarly we have the constraint $Y_j + H_j \leq Y_i$ in case C_i is above C_j (obtained by swapping i and j in (3)). Similar constraints can be formulated by using the previous two inequalities by swapping Y and X and by swapping H and W. This results in the following four inequalities:

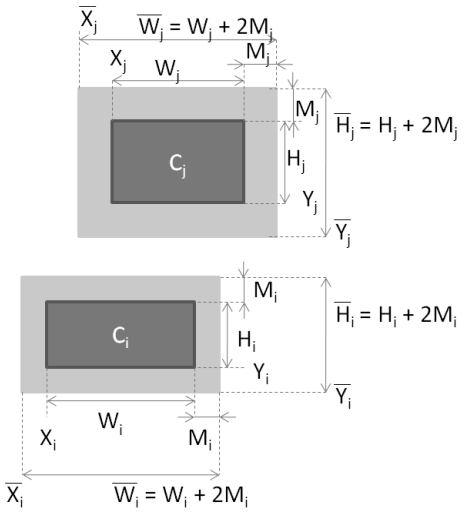


Figure 15: Variables associated to a cell.

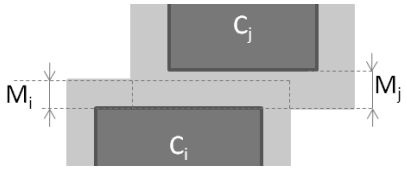


Figure 16: Two intersecting margins. Cells may not intersect with these margins.

$$\begin{aligned} Y_i + H_i &\leq Y_j \\ Y_j + H_j &\leq Y_i \\ X_i + W_i &\leq X_j \\ X_j + W_j &\leq X_i \end{aligned} \tag{4}$$

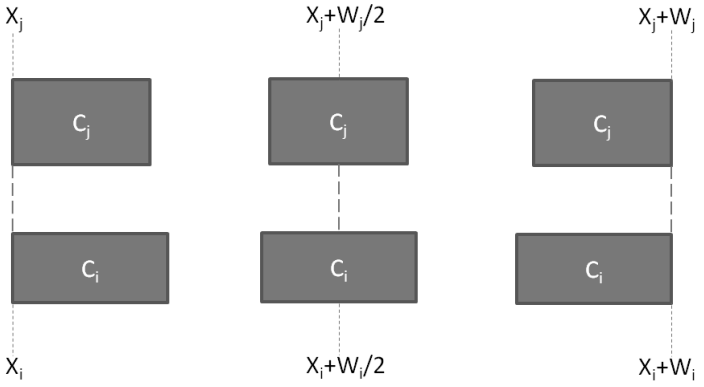
If at least one of these four inequalities holds, then the two cells are not overlapping.

- Margin: A margin specifies an area around a cell in which no (part of an) other cell may be placed. Therefore it puts a requirement on the minimal distance between two cells. Figure 15 shows two cells with margins M_i and M_j around them. To ensure these margins do not overlap, the inequalities of the No-Overlap relation can be applied to the augmented cells $\overline{C_i}$ and $\overline{C_j}$, which indicate the area of the cell plus its margin. This is done by formulating the No-Overlap inequalities with $\overline{Y}, \overline{H}, \overline{X}$ and $\overline{W}$ instead of Y, H, X and W, in (4).

  Here we will relax the no-overlap constraint by allowing overlap of the margins (but no overlap of the cells), see Figure 16. The minimal distance between two cells is defined by the maximum of the two margins. This can easily be achieved by specifying new values for M_i and M_j respectively $\overline{M_i}$ and $\overline{M_j}$, with $\overline{M_i} = \max(M_i, M_j)$ and $\overline{M_j} = 0$. In this way the size of one of the cells is expanded by the maximum of the two margins, resulting in the desired minimal distance requirement between the two cells.

- Matching: Two cells are said to match if they have the same angle of rotation. So let cell C_i and C_j match and let R_i, R_j denote their rotation. Then clearly we have the requirement $R_i = R_j$.

- Alignment: As described earlier, different types of alignment exist. For example cells can be aligned in vertical direction to the left, center or right. Similarly in horizontal direction cells can be aligned to the bottom, center or top. Figure 17 shows three different ways of aligning two cells vertically. As illustrated for vertical left alignment the requirement $X_i = X_j$ can be derived. Similarly the requirements for the other types of alignment can be derived.



(a) Left aligned (b) Center aligned (c) Right aligned

Figure 17: Two cells vertically aligned, in three different ways.

4.4 Specification of routing

In this section we will describe how routing is taken into account. To specify the routing problem we need to define the pins of cells, and we need to specify the path of the wires connecting these pins. The following sections will describe how pins are defined and how the path of a wire can be constructed using so-called Steiner points. Note that routing is discussed here apart from placement, but that in the end a single expression of the placement and routing will be formulated by using logical conjunctions.

4.4.1 Pin definition

To be able to create connections between cells, first the pins of each cell have to be defined. A pin P_j is considered a point anywhere on a cell C_i, specified by its location $X_{i,j} = X_i + e_j, Y_{i,j} = Y_i + f_j$ relative to the lower left corner (X_i, Y_i) of the cell, see Figure 18. In our implementation we chose to specify pins on the border of a cell.

The pin location needs to rotate along with the rotation of the cell. Figure 18 shows the difference between a cell rotation of 0 and 90 degrees. As is shown the pin rotates with the cell. To achieve this we distinguish four different cases, one for each rotation angle. The location of pin P_j of cell C_i is defined with a given e_j and f_j, in case of no rotation. The rotation has to be taken into account. In case of 0 degrees we get $(X_{i,j} = X_i + e_j \wedge Y_{i,j} = Y_j + f_j)$. This can be formulated using an implication as follows: $(R_i = 0) \implies (X_{i,j} = X_i + e_j \wedge Y_{i,j} = Y_j + f_j)$. Similar expressions can be generated for $90, 180$ and 270 degrees, where for each rotation angle the location of the pin is adjusted accordingly. These are shown in Equation 5. For one single pin, all four implications should hold.

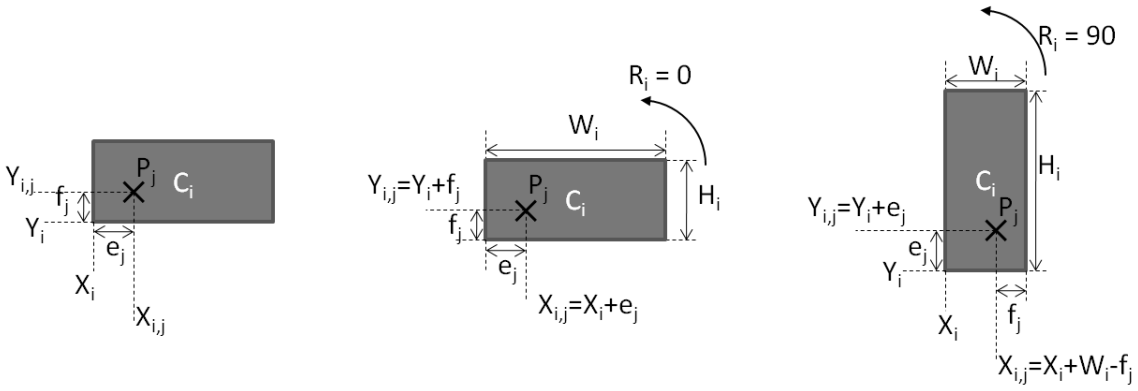


Figure 18: A pin rotates with the cell. Given a location e_j, f_j for a pin on a cell in case of no rotation, the specification $(X_{i,j}, Y_{i,j})$ needs to be adjusted for the rotation of the cell.

$$\begin{aligned}
(R_i = 0) &\implies (X_{i,j} = X_i + e_j \wedge Y_{i,j} = Y_i + f_j) \\
(R_i = 90) &\implies (X_{i,j} = X_i + W_i - f_j \wedge Y_{i,j} = Y_i + e_j) \\
(R_i = 180) &\implies (X_{i,j} = X_i + W_i - e_j \wedge Y_{i,j} = Y_i + H_i - f_j) \\
(R_i = 270) &\implies (X_{i,j} = f_j \wedge Y_{i,j} = Y_i + H_i - e_j)
\end{aligned} \tag{5}$$

4.4.2 Specification of wires by means of Steiner Points

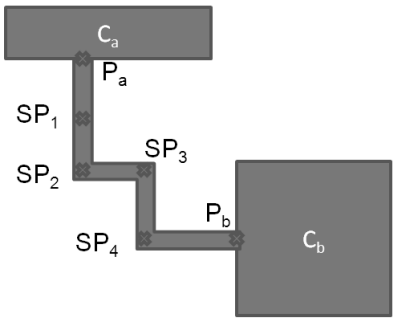


Figure 19: A wire from pin A to pin B, passing through Steiner points SP 1 up to SP 4.

A connection between two pins is realized by a wire, containing a number of jogs (≥ 0). In order to specify the locations of these jogs a number of extra intermediate points are introduced, called Steiner points [10]. A Steiner point SP_i is defined by its location (X_i^{sp}, Y_i^{sp}). The width of the wire is specified in the design rules. The wire is only allowed (but not forced) to bend over ± 90 degrees on the location of a Steiner point, see Figure 19. Allowing jogs only at the location of Steiner points implies that the number of jogs in a wire is limited by the number of Steiner points. By introducing variables for the locations of these Steiner points, we can define the path of a wire in terms of these Steiner points and the resulting SMT-problem can generate this.

In our implementation we choose to assign a fixed number of Steiner points (N) to a wire. When no jogs are allowed in a wire, no Steiner points are needed. However by assuming each wire has at least one Steiner point ($N \geq 1$), we can always define a wire from pin P_a to the first Steiner point (SP_1) and from the last Steiner point (SP_N) to pin P_b. If no Steiner points are allowed, this wire

definition becomes invalid and an additional case for creating a wire from pin P_a directly to pin P_b needs to be constructed.

Due to the introduction of Steiner points we now know how pins are connected in a wire. No additional specification has to be made for this, we just assume the wire is there and specify requirements on it to exclude undesired situations, like diagonal paths and dead ends (see below). Relations to other objects (e.g. cells or other wires) are discussed in section 4.5.

In order to specify requirements for a connection, the wire is split up in wire segments. A wire segment is a direct connection between one pin and one Steiner point, or between two Steiner points, see Figure 19. All wire segments have a rectangular shape, where either the width or the height of this shape equals the wire width. A requirement for a wire between two pins can be specified by requirements for each wire segment of that wire.

- Diagonal wires: To prevent diagonal wires, the start and end locations of a wire segment need to have either the same value for the X or the Y coordinate.
- Dead ends: A dead end can occur when a wire leaves a Steiner point in the opposite direction as it came from. This is shown in Figure 20 where a dead end is created at SP_1. A Steiner point can be modelled as a square with a pointer attached to each of the four sides (N,E,S and W). When a connection is made to an other Steiner point or pin, the pointer at the side where the connection is made, is assigned to the identifier of the connecting Steiner point or Pin, see Figure 21. Since a pointer can only be assigned to one single value (in the result of an SMT-problem), this will exclude the appearance of a dead end. Note that these pointers will be used again later on, to describe additional requirements.

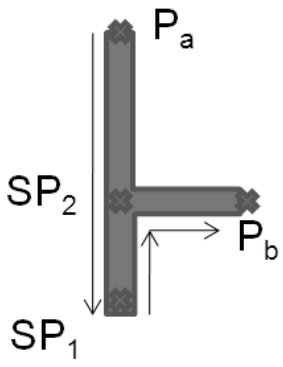


Figure 20: A path between P_a and P_b, going from P_a, via SP_1, and SP_2, to P_b. There a dead end at SP_1 is created.

4.5 Wire relations

A wire is split up in segments specified by pins and Steiner points, see section 4.4. Since these wire segments have a rectangular shape, requirements (like No-Overlap and margins) can be defined similarly as they are defined for cells, see section 4.3. Note that two wire segments of a single wire can overlap. Three types of relations to a wire can be distinguished: relations to chip area, relations to other wires and relations to cells.

- Wire related to chip area: The path generated for a wire should always stay within the chip

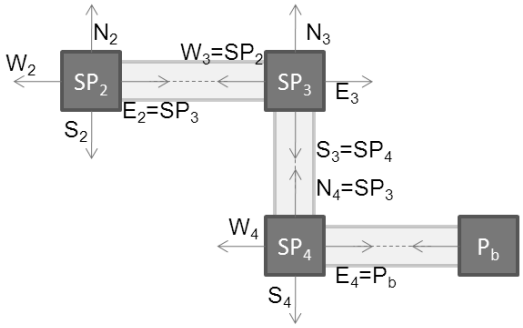
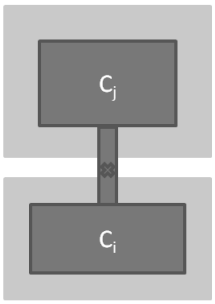


Figure 21: Three Steiner points, depicted as squares, with a pointer attached to each of the four sides. When connected to another Steiner point or pin the pointer in the direction of the connection is assigned with the corresponding identifier.

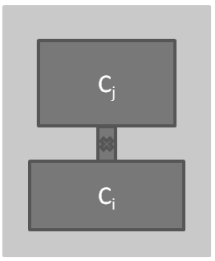
area. This can be done by specifying similar constraints as is done in section 4.2, where the location and size of the wire segments are used, instead of the location and size of the cell.

- Wire related to other wires:
  - No-Overlap between two wires: Similar to the No-Overlap relation between cells, a No-Overlap relation for two wire segments of two different wires can be constructed. Notice that the number of times this requirement needs to be specified, for all wire segments not to overlap each other, grows quadratically.
  - Margin between two wires: Adding margins between two wires can be done by extending the No-Overlap relation similar to the margins on cells.
- Wire related to a cell: For relations between a wire and cells we need to distinguish two kinds of cells. The cells connected by this wire, and cells not connected by this wire.
  - No-Overlap between a wire and an unconnected cell: Similar to the No-Overlap relation between two cells, a no overlap relation can be constructed for each wire segment to that cell. As stated before, in practice this is generally not required, since wires are not restricted by cell boundaries.
  - No-Overlap between a wire and a connected cell: Since we assumed pin locations are located on the border of a cell, a No-Overlap relation for each wire segment to a cell can be constructed similarly to the No-Overlap between a wire and an unconnected cell. Notice that as a result of this requirement a wire leaves a connected cell perpendicular to the cell border.
  - Margin between a wire and an unconnected cell: Adding a margin between a wire and an unconnected cell can be done in a similar way as has been done with margins on cells, by extending the No-Overlap relation for each wire segment.
  - Margin between a wire and a connected cell: Notice that the no-overlap relation between a wire and a connected cell can not be extended with a margin just as easily. If a connection to a cell is realized by a wire, a margin between the wire and the cell it to be connect would prevent the connection from being made. Leaving out the margin entirely for cells connected by a wire is also an undesired situation, as shown in Figure 22c.

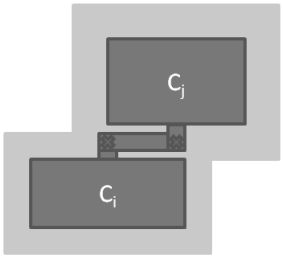
    This problem can be solved by distinguishing wire segments between a pin and a Stiener



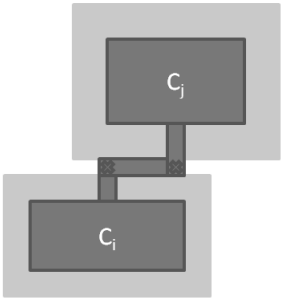
(a) Undesired situation when there are no jogs in the wire.



(b) Desired situation when there are no jogs in the wire.



(c) Undesired situation when jogs are present in the wire.



(d) Desired situation when jogs are present in the wire.

Figure 22: The difference between the desired situations and undesired situations involving wire segments with respect to the margins around cells.

point from wire segments between two Steiner points. The wire segments between a pin and a Steiner point should ignore the margin of the connected cell. The wire segments connecting two Steiner points may only ignore the margins of the connected cells if there are no jogs in the wire. This is shown in Figure 22b. If there are jogs in the path then the No-Overlap relation should be extended to include the margin around the cells, see Figure 22d.

4.6 Wire restrictions

Wires can have additional restrictions. For example one could limit the length of the resulting wire, or the number of jogs in the wire. To count the number of jogs in a wire, we can look at the Steiner points. We know that the path of a wire can only (but is not forced to) bend at the location of a Steiner point. The pointers assigned to the four sides of a Steiner point are assigned to a unique value when a connection is realized in that direction. Per Steiner point one could state that if a connection is realized to either the north or the south and a second connection is realized to either the east or the west, then the path contains a jog at this Steiner point, see Figure 21. Since we know the number of Steiner points, a summation of the number of jogs per wire can easily be made and a restriction on this number can be defined. Notice that when allowing three dimensional routing, then changing direction to go up or down is in principle also a jog.

It should be noticed that the total number of jogs in the wires is also limited to the total number

of Steiner points in the wires. Limiting the total number of jogs could also be done by reducing the total number of Steiner points. To do so the number of number of Steiner points for each wire has to be made variable in the SMT-problem. As a result the additional requirements, which are specified using a fixed number of wire segments, should be changed to allow a variable number of wire segments. Since this makes specifying additional requirements much more difficult we chose not to reduce the total number of Steiner points, but only limit the total number of jogs.

A limitation on the total wire length of all wires in a layout design can easily be realised after determining the total wire length of the wires. The total wire length is determined by a summation of the length of the individual wires. The length of one wire between two pins can easily be determined, by summing the length of all the wire segments of that wire. The length of one wire segment can be determined by the absolute value of difference between the start and the end location of the wire segment.

4.7 Two optimization approaches

A layout design can be optimized with respect to different aspects. We present an approach to minimize the wire length and an approach to minimize the total number of jogs in the wires.

Minimizing the total wire length can be done by iteratively decreasing the specified maximum total wire length. An initial wire length $n(0)$ can be obtained by running a model without any limitations on the wire length. Then the maximum wire length $N(k+1)$ for iteration $k+1$ (with $0 \leq k < K$) can be specified by $N(k+1) = \alpha * n(k)$, where $\alpha (0 < \alpha < 1)$ is a reduction factor and $n(k)$ equals the wire length of iteration k. This process can be repeated as long as $k < K$ and a solution with a wire length $n(k) \leq N(k)$ exists. For practical considerations we used $\alpha = 1 - P/100$ with $P = 5$ and $K = 20$. Notice that this method does not guarantee an optimal wire length.

An approach for minimizing the total number of jogs in the wires was also implemented. This is similar to that for the wire length. An initial value for the total number of jogs $j(0)$ is obtained by running the model without imposing any restrictions on the number of jogs. Then the allowed maximum total number of jogs $J(k+1)$ can be iteratively decreased by specifying $J(k+1) = j(k)-1$ for iteration $k+1$ (with $k \geq 0$), where $j(k)$ equals the total number of jogs found in iteration k. This can be repeated as long as $j(k) > 0$ and a solution with $j(k) \leq J(k)$ can be found.

Similar requirements can be made for minimizing the number of jogs per wire. When doing this a method for prioritizing wires should also be considered, since the order in which the wires are minimized can influence the result.

5 Results

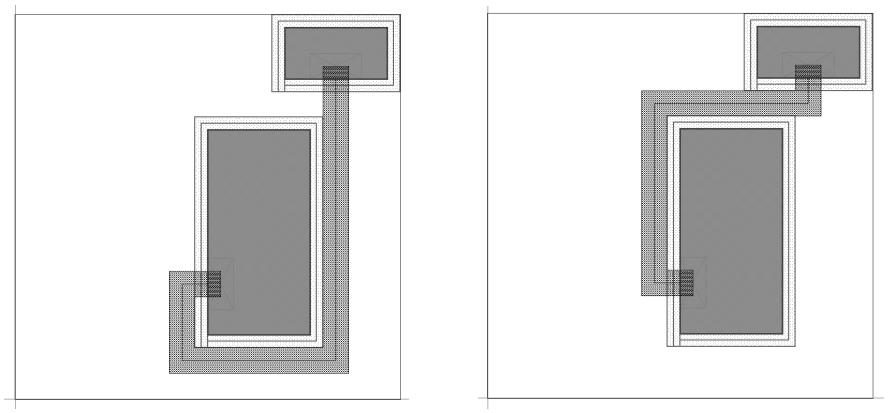
In this section we will demonstrate some results obtained by using the prototype tool created for this project. First an artificial example will demonstrate how placement and routing is combined and will show that routing influences the resulting placement. Then two realistic problems are used to demonstrate what can already be achieved with the limited functionality in the prototype tool. Afterwards we will discuss the scalability and the resulting design flow.

5.1 Demonstration of placement and routing using an artificial example

In this section we will show a number of examples where a number of cells are placed and routed on a chip area. We will demonstrate how an undesired solution can be transformed to a feasible solution with little to no effort of a designer.

5.1.1 Create and improve routing of placed cells

An artificial test case was created where two cells, each having the same margin, are placed on a fixed location in a chip area, see Figure 23. A wire between the two cells is realized by the solver, see Figure 23a. Clearly this solution is valid. However, we would like the wire to be routed along the top of the bigger cell to achieve a shorter wire, see Figure 23b. We could simply place a blockage below the bigger cell, such that there is no other option but to route the wire along the top. A more generic way to achieve the result in Figure 23b is by adding a requirement on the total wire length. Suppose the wire length in the solution of Figure 23a is L. Then we rerun the SMT-solver with an additional requirement of a wire length less than L. Indeed, a new result with a total wire length less than L is found, as shown in Figure 23b.



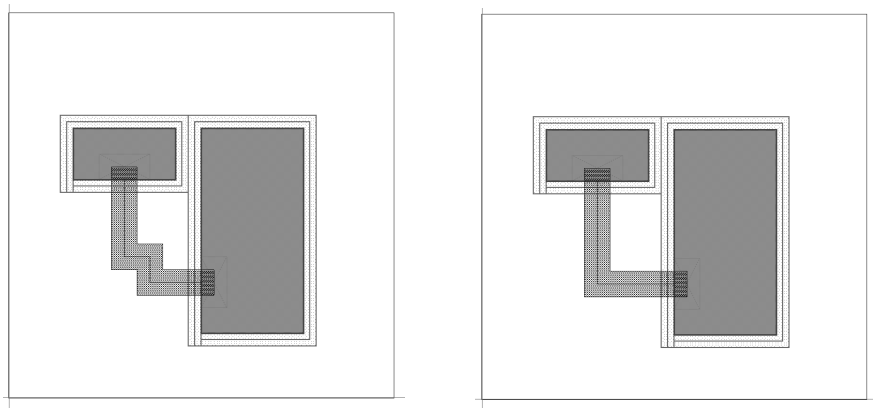
(a) Undesired situation (has wire length L)

(b) Added a requirement limiting the wire length (has wire length $L' < L$)

Figure 23: An artificial test case where two cells are placed on a fixed location and a wire is realized between the two cells by the solver. The left case has no additional requirements on the wire. The solution in the right figure with wire length L' was obtained by requiring a total wire length of less than L.

Limitations on the wire length can improve a layout design. However, requirements on the wire

length are not always helpful. For example, consider the situation as in Figure Figure 23 and move the smaller cell to the left of the big cell, see Figure 24. Clearly, the wire length of Figure 24a and 24b are equal, but the wire in Figure 24a has 3 jogs. We need to look at more routing aspects, than just the wire length, if we would have a minimum number of jogs in the wire. For example, by adding an additional requirement, limiting the number of jogs in the wire to two, the result shown in Figure 24b was obtained.



(a) No additional requirements (has wire length 7 and 3 jogs)

(b) Added a requirement limiting the number of jogs (has wire length 7 and 1 jog)

Figure 24: An artificial test case where two cells are placed on a fixed location and a wire is realized between the two cells by the solver. The left situation has no additional requirements, resulting in a wire with 3 jogs. The right situation is obtained by specifying a maximum number of jogs in the wire (< 3). The wire length in both solutions is the same.

5.1.2 Influence of variable cell location on placement and routing

The placement of the cells can be improved by specifying additional routing constraints. This will be shown using the same test case as in Figure 24. There the location of the smaller cell was fixed, now we will make the location of this cell variable. So, the solver has to assign a location to this cell. The initial result found by the solver is shown in Figure 25. No additional requirements on the wire length or the number of jogs have been made.

As done before, we can add a requirement limiting the wire length or the number of jogs in the wire. In this case we even like to minimize the wire length. A straight forward way of doing this is by iteratively decreasing the specified maximum wire length until the formula becomes unsatisfiable, see Figure 26. Notice that for every iteration the location and rotation of the lightly coloured cell can change, in order to meet the requirement on the wire length.

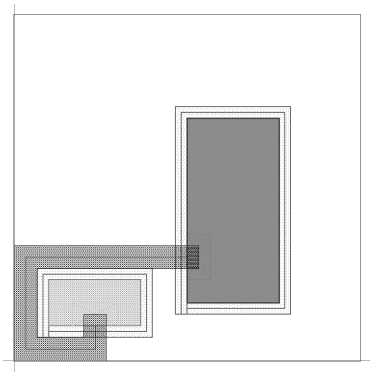
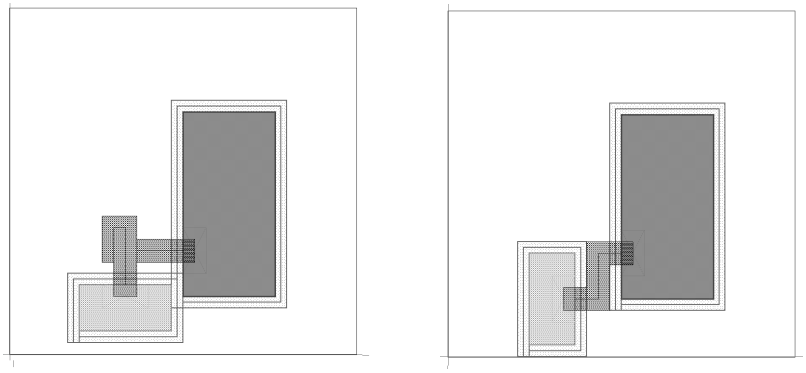


Figure 25: The location of light coloured cell has been made variable inside the chip area, the dark cell has a fixed location. No additional requirements on the routing have been made. (Wire length L_0 and 3 Jogs)



(a) Wire Length: L_1 (3 jogs)

(b) Wire Length: L_2 (2 jogs)

(c) Wire Length: L_3 (4 jogs)

(d) Wire Length: L_4 (0 jogs)

Figure 26: The location of the light coloured cell has been made variable inside the chip area, the dark cells have a fixed location. To improve the placement as well as the routing of the resulting layout design, the maximum allowed wire length was decreased iteratively. Notice that $L_1 > L_2 > L_3 > L_4$

In Figure 26a a self intersecting wire is created. In practice self intersecting wires are not allowed in a design. A requirement can be constructed which prevents the creation of self intersecting wires. However this is not necessary, since by simply continuing with reducing the maximum number of jogs in the wires or the maximum total wire length, the problem will eventually be solved. It is evident

that reducing the total wire length and reducing the number of jogs is always possible in a solution containing a self intersecting wire, since the loop can be removed.

5.2 Realistic Problems

We can apply our approach to simplified versions of realistic problems. In the following sections we will consider a communication channel and an LNA.

5.2.1 Communication channel

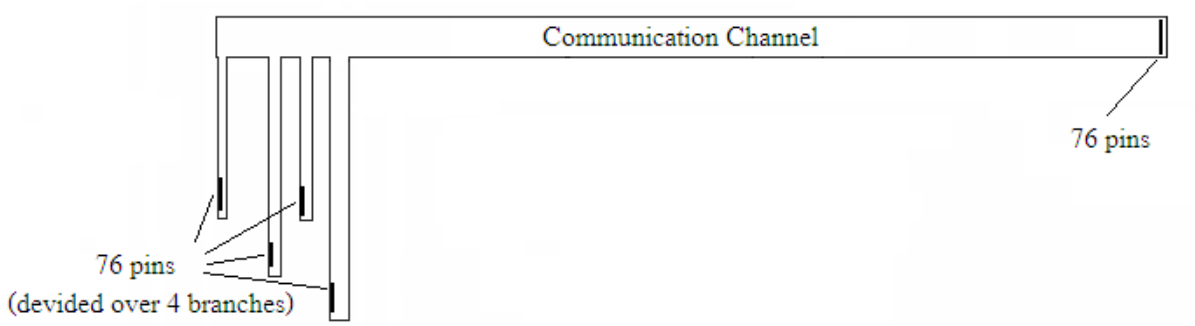
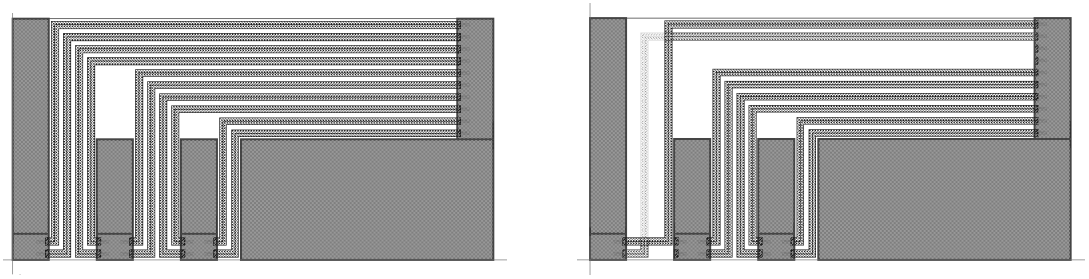


Figure 27: Example of the area of a real communication channel, and an indication of the pins of the 76 wires in this channel.

A communication channel is used to group a number of wires in a chip, and can be seen as a separated chip area in which only the specified wires may be placed. Figure 27 illustrates such an area. A communication channel can contain many wires, easily 50 to 100 wires. The wires in such a channel can automatically be routed, we will demonstrate this by an example with ten wires in the channel, see Figure 28a: All wires start from one cell in the upper right corner of the chip area, and follow the channel horizontally. These wires are then split in three branches towards the bottom of the chip area, where they are connected to three other cells. Notice that problems can occur in the 2D-routing of a communication channel. For example, one wire cuts off the path for three other wires, see Figure 28b. The lighter color wire illustrates the part of one of the wires that was cut off. Notice that this is not a valid solution.

In real life the start and end locations of the wires in a communication channel are not sorted on both ends, thus three dimensional routing will be required to solve the problem. So far we only did two dimensional routing, thus wires are not allowed to cross each other. As a result the wires of each branch all turn one after another. When the width of the communication channel and that of its branches is chosen to just fit the selected number of wires, then the wires are forced to turn in a nicely diagonally aligned fashion, see Figure 29. These diagonally aligned corners are preferred by designers. Therefore, sorting and reordering the start and end locations of the wires would result in a preferred routing solution for a communication channel.

Suppose each pin (at the end of the wires) in the upper right area of Figure 28a was placed on a separate cell. Then by only making the position on the Y axis variable, the solver is able to reorder the cells and by doing so, it is able to reorder the pins. Since the pin locations in the bottom three cells are still fixed and wires are not allowed to cross, there exists only one ordering in which all the



(a) Correct routing

(b) Path for three other wires is cut off, one of these wires is shown with a lighter color for the part of the wire that was cut off (not a valid solution).

Figure 28: A communication channel with three branches in the top and left [art of the chip area. The cells have a fixed location and are used to connect the wires and to close the routing channel.

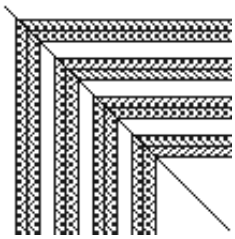


Figure 29: Zoomed in on the branching point of one of the branches in Figure 28. The shown line indicates the diagonal over which the corners are aligned.

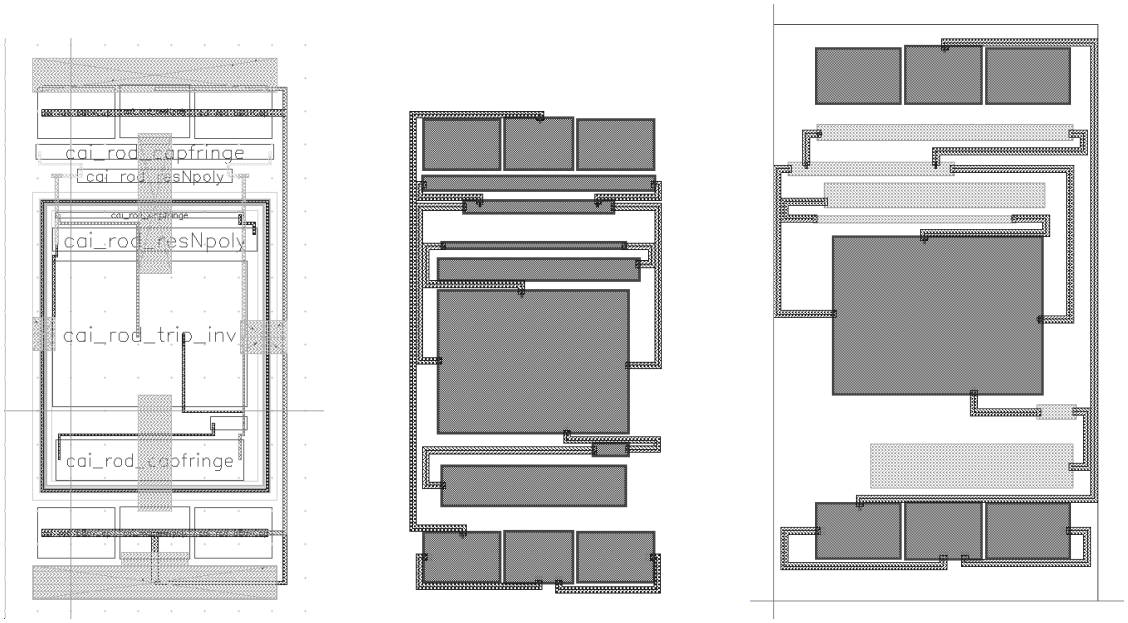
wires can be created. This results in an automatically ordering of the pins. Notice that reordering the pin locations in a communication channel influences the routing inside the connected cells. The new pin locations could be used as feedback to the designer, stating how the layout design could be improved by swapping pin locations.

5.2.2 Low-Noise Amplifier (LNA)

Figure 30a shows the layout design of an LNA. Notice that the cells in this design are no elementary components, but each cell may be the solution of an underlying placement and routing problem. We simplified this design, such that it can be handled by our tool (by removing e.g., crossing wires, wires with more than two pins, and the greyed input and output connections). To have a reference to the original layout design the locations of the cells have been made fixed, and the wires were generated automatically, see Figure 30b. Figure 30c shows the situation where the lightly coloured cells have been given a variable location, which is then assigned by the solver. These layout designs have been optimized for the total number of jogs in the wires. This is done straightforwardly, by using an additional requirement, restricting the total number of jogs, which is then iteratively decreased until the SMT-problem becomes unsatisfiable.

The original LNA example (shown in Figure 30a) is a parametrized design. The size of the cells in this design can vary by choosing different parameter values, so the placement and routing processes have to deal with these changes. Hence an automated solution is needed. Since the original design was three dimensional, the results in Figures 30b and 30c can not directly be compared. However, the resulting placement of Figure 30c resembles the placement made by the designer,

while only the location of the seven dark coloured cells is given as input from a designer.



(a) Original manual design (three dimensional)

(b) Simplified model, where the cell locations are fixed, automatically routed (30 jogs, generated in: $<$ 1 minute)

(c) Simplified model, automatically placed and routed (26 jogs, generated in: 25 minutes)

Figure 30: Three layout designs of an LNA module.

The simplification of the LNA module was used two more times, with each time, more cells been given a variable location, see Figure 31. The top three and bottom three cells in the left image are specified to be horizontally bottom aligned, while their location was made variable for the right image. It should be noted that we still only look for the minimal total number of jogs in the wires. As a result the placement is adjusted and optimized such that the minimal number of jogs in the wires is achieved.

(a) big block free (20 jogs; generated in: 70 minutes)

(b) all blocks free (16 jogs; generated in: 20 hours)

Figure 31: Two Layout designs of an LNA module, where the locations of more cells are made variable. Optimized for the total number of jogs in the wires.

5.3 Scalability

As stated before both the individual placement and routing problems are NP-complete. Though not proven, it is reasonable to assume that a combination of the placement and routing problems is also NP-complete. The run-time for any known method to solve an NP-complete problem grows more than polynomially to the problem size. In our case the number of requirements will grow exponentially to the number of cells, the number of wires, and the number of Steiner points per wire. The quickly growing number of requirements results in an equally fast growing SMT-problem.

The way Yices solves an SMT-problem is not part of this research. But it is clear that as soon as Yices finds a solution it stops. So detecting an SMT-problem is unsatisfiable will generally take much longer than finding a satisfiable solution.

Two test cases were constructed to show the run-time of Yices for a growing number of cells. The basic test case only requires that cells may not overlap. The second test case used three additional requirements namely: a margin around the cells, a matching relation between all the cells and alignment of every two cells. No wires are created between the cells. The run-times for 6 to 33 cells is shown in Figure 32. In case of 33 cells the run-time reaches 1.5 minute, after which we stopped. Clearly, the run-time grows exponentially. A reason for why test cases with additional requirements are generally faster than the basic test is found in the way Yices finds a solution, but this is out of the scope of this report.

The LNA example in section 5.2.2 also serves as a good illustration to show the scalability of this method. Table 4 shows the run-time of the four LNA models, as shown in Figures 30b, 30c and 31. Notice that the only difference between the input for Figure 30c and Figure 31a is that the location of the biggest cell is made variable, which results in a more than twice larger run-time. This shows that

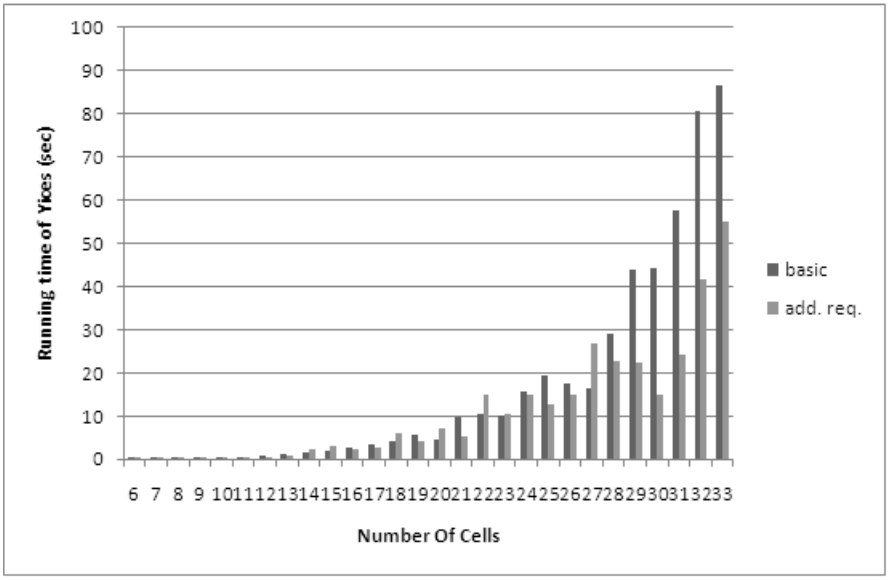


Figure 32: Run-time of Yices as a function of the number of cells. Requirements of the basic case only include no overlapping. Additional requirements include margins, matching and alignment requirements.

Figure:	Run-time
30b	$<$1 minute
30c	$\pm$ 25 minutes
31a	$\pm$ 70 minutes
31b	$\pm$ 16 hours

Table 4: Run-times of the LNA example.

this method will only work if the problem size is kept small (e.g., by using hierarchy in the design) and that specifying additional requirements, can help in finding a solution faster.

5.4 Design flow

This section describes the automated process of generating a layout design from a schematic design. There are different ways our method could be used. The following sections will describe four different usages of our method.

5.4.1 Automated layout generation

A layout design can automatically be generated from a schematic design by the following procedure illustrated in Figure 33: The design environment we used is called Cadence [6]. Cadence supports a scripting language called SKILL [1]. Using such a SKILL-script the input configuration file containing cell information and the netlist can be generated from a schematic design. This configuration file, along with the additional requirements provided by a designer, are automatically translated to an SMT-problem. The SMT-problem is solved by Yices. The solution provided by Yices, combined with the input configuration are translated to a results file. This file contains the size and locations of

the cells and wires. A SKILL-script can then interpret this file and draw the actual layout design in Cadence. This process can be executed without needing any input from a layout designer (except for specifying additional requirements).

Without input from a designer there is no specific information about what aspects of the design are most important, thus the design will probably not be optimal. However the layout design can be generated fully automatically and be simulated afterwards. Even though the layout design is not optimized, a first impression of the performance of the layout design can already be obtained.

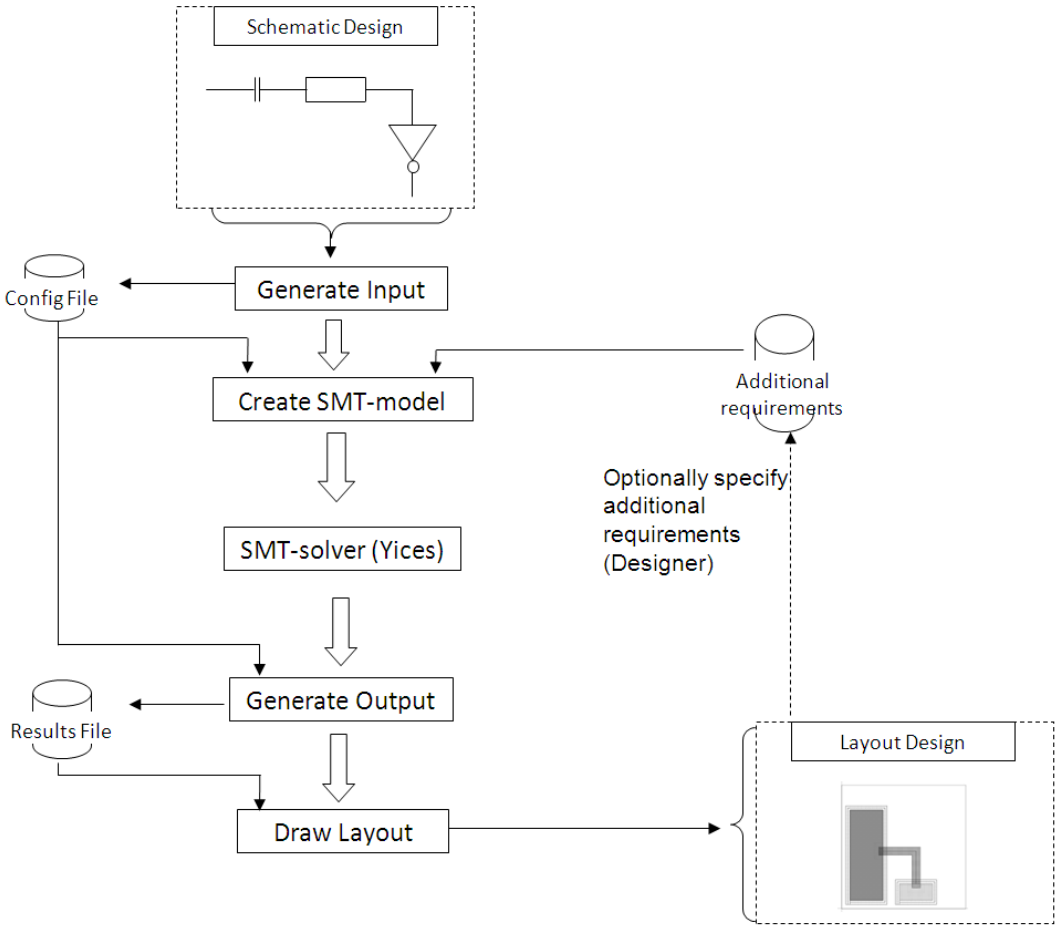


Figure 33: The design flow from schematic design to a layout design.

5.4.2 Interactive layout generation

Our approach can also be used interactively. To this end, an implementation has to be made where additional requirements can be specified interactively and the result of the current specification is shown. Depending on the result one could choose to change or to add requirements, shaping the resulting layout design as desired.

5.4.3 Relative Object Design (ROD)

A new way to design layouts is to use the ROD library [5]. The ROD library enables a designer to create parametrized designs, including layout designs. Our method can easily be extended to generate ROD code. The advantage of having ROD-code is that it is more flexible and open to changes.

5.4.4 Automated completion of a partial layout

When only a part of the layout design has been made, this layout design can be finished using our method. A SKILL-script can extract the cells and pins from a layout view. Together with the schematic design, the missing cells and wires can be detected and added. This has been done with the LNA example in section 5.2.2, where in Figure 30c a partial placement of the layout design has been given as input.

6 Conclusion

We presented a constraint driven approach which uses Satisfiability Modulo Theories (SMT) and Yices to solve the placement and routing problems simultaneously. To the best of our knowledge this is the first time the placement and routing problems have been addressed as a single problem. It has been shown that by specifying requirements on the wires, both the placement of the cells and the wires are adjusted such that these requirements are met. Currently we are able to find a solution to a placement and routing problem, which satisfies a given set of requirements. A straightforward implementation is made to minimize the wire length or the number of jogs in the wires of a layout design. Finding some form of optimality is still an open topic.

There is still room for further improvements since we limited ourselves to a simplification of the actual problem. However many additional requirements (Table 3) can be implemented analogously to the already defined requirements.

In our opinion the presented method has the potential to automate the step from a schematic design to a layout design for analog chip designs. One of the advantages of a fully automated approach is that simulation results of the layout design can be obtained very quickly. Our method is applicable in different levels of the hierarchy of a design, and can be used interactively to improve the resulting layout. It can also be applied to finish partial layout designs. An extension to our method can be made to generate ROD code, this would improve the flexibility of the resulting layout design.

The biggest challenge is to overcome the scalability problems. Both the placement and the routing problems are proven NP-complete. As expected, the time needed to solve the placement and routing problems simultaneously also grows exponentially with the problem size.

The advantage of our method is that a complex problem can be cut into small parts, where each part can be translated to a constraint. These constraints can then be put together and solved by existing tools. The problem of how to solve a placement and routing problem, has changed into the problem of defining what is required for the solution to a placement and routing problem.

7 Future Work

In this report we demonstrated the feasibility of applying SMT and Yices to the placement and routing problem as found in a chip design process. Clearly not all aspects and issues have been solved, so several open questions remain to be answered. This section will mention a number of these open questions and issues.

- Implement more features: In order to get a fully working tool our prototype tool (created for this research) should be extended to handle more requirements for placement and routing, design rules and three dimensions.
- Optimisation of the layout design: The layout designs generated by the current implementation can only be optimized with respect to the total wire length or the total number of jogs in all the wires, by using a straight forward approach. A smart optimization method is needed when different aspects like individual wire length, number of jogs in a wire and perimeter of chip area have to be taken into account. This can be done using an objective function, or by an automated method of specifying additional requirements. Notice that the criterion for determining what is optimal, can vary for different designs.
- Optimise the SMT-problem: Research can be done, how an SMT-problem should be constructed such that it can be solved as efficient as possible. Currently in the generated SMT-problem all objects are simplified to rectangular shaped areas. In particular each wire is modelled as a number of rectangular areas which causes a dramatic increase of requirements (No-Overlap, etc.). Defining objects as rectilinear polygon shaped areas might be more efficient, since then each wire can be defined as a single polygon area. The expression generated from the requirements can be formulated in various ways. This can also influence the execution time of the SMT-solver. Even the order in which the requirements are specified in the SMT-problem can influence the run-time.
- Robustness of a solution for small changes: When using the current method in an interactive way, a designer can specify an additional constraint to improve a small aspect of a generated layout design. This could result in a completely different layout, since the SMT-solver is only concerned with finding a satisfying solution to the given SMT-problem. This is unacceptable for the designer, since only the areas in which the changes were intended should be changed. When improving a given layout design it could be effective to only allow moving cells over a given distance from their previous location to avoid generating a completely different layout.
- Unsatisfiable SMT-problems: When a SMT-problem is unsatisfiable the SMT-solver used (Yices) only states that no solution can be found. It can be difficult to find out what requirement causes the model to become unsatisfiable. An automated way to find contradicting requirements or to relax requirements would make our approach more user-friendly and robust. A partial result might be generated which helps a designer to identify possible mistakes.
- Automatically generated requirements: In [9] a method is proposed to automatically generate additional requirements from a schematic design. When including this method to our approach, this can significantly improve our results and reduce the amount of input to be given by a designer.

8 Acknowledgements

I would like to thank Edwin van der Heijden, Frank Leong and Gerben de Jong for their input and support as layout designers on several design topics. Thanks to Wybe Boelders for his input on exiting routing tools available in the design environment (Cadence) and the flaws in these tools. Last but not least I would like to thank Joost Rommes and Theo Beelen for their support and guidance as supervisors of this project.

References

[1] T. J. Barnes. Skill: a cad system extension language. *27th ACM/IEEE Design Automation Conference*, pages 266–271, 1990.

[2] Robert Brummayer and Armin Biere. Boolector: An efficient smt solver for bit-vectors and arrays, 2009. http://fmv.jku.at/boolector/ (accessed 14 July 2011).

[3] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzen, Alberto Griggio, and Roberto Sebastiani. The mathsat 4 smt solver. http://mathsat.fbk.eu/ (accessed 14 July 2011).

[4] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015, chapter 12, pages 150–153. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2010. http://verify.inf.unisi.ch/opensmt.

[5] Cadence Design Systems, Inc. Virtuoso relative object design user guide. Product Version 6.1.2 (company restricted document).

[6] Cadence Design Systems, Inc. . Virtuoso analog design environment, 2011. http://www.cadence.com/products/cic/analog_design_environment/pages/default.aspx (accessed 19 July 2011).

[7] J.M. Cohn, D.J. Garrod, R.A. Rutenbar, and L.R. Carley. Koan/anagram ii: new tools for device-level analog placement and routing. *IEEE Journal of Solid-State Circuits*, 26(3):330 –342, mar 1991.

[8] Bruno Dutertre and Leonardo De Moura. The yices smt solver, 2006. http://yices.csl.sri.com (accessed 19 July 2011).

[9] M. Eick, M. Strasser, Kun Lu, U. Schlichtmann, and H.E. Graeb. Comprehensive generation of hierarchical placement rules for analog integrated circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(2):180 –193, feb. 2011.

[10] P. Winter F.K. Hwang, D.S. Richards. *The Steiner Tree Problem*. Elsevier, North-Holland, 1992.

[11] S.L. Hakimi. A problem on rectangular floorplans. In *IEEE International Symposium on Circuits and Systems*, volume 2, pages 1533–1536, jun 1988.

[12] Ilog, Inc. Cplex solver, 2011. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ (accessed 14 July 2011).

[13] Hyondeuk Kim, HoonSang Jin, and Fabio Somenzi. Sateen: Sat enumeration engine for smtcomp'08. http://vlsi.colorado.edu/~hhkim/sateen/ (accessed 14 July 2011).

[14] Yao-Wen Chang Laung-Terng Wang and Kwang-Ting Cheng. *Electronic Desing Automation: Synthesis, Verification and Test*. Morgan kaufmann, 2009.

[15] H. A. Pogorzelski. Reviewed work(s): Remarks on nicod's axiom and on "generalizing deduction". *The Journal of Symbolic Logic*, 30(3):376–377, september 1965.

[16] Silvio Ranise, Loria, and Cesare Tinelli. The smt-lib standard: Version 1.2, 2006. http://www.SMT-LIB.org (accessed 19 July 2011).

[17] S. Sutanthavibul, E. Shragowitz, and J.B. Rosen. An analytical approach to floorplan design and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and*

Systems, 10(6):761 –769, jun 1991.

[18] Chao Chi Tong and Chuan-Lin Wu. Routing in a three-dimensional chip. *IEEE Transactions on Computers*, 44(1):106 –117, jan 1995.

[19] Pascal Van Hentenryck. *The OPL optimization programming language*. MIT Press, Cambridge, MA, USA, October 2006.

[20] Linfu Xiao, E.F.Y. Young, Xiaoyong He, and K.P. Pun. Practical placement and routing techniques for analog circuit designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 675 –679, nov. 2010.

[21] Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning*, LPAR'10, pages 481–500, Berlin, Heidelberg, 2010. Springer-Verlag.

A SMT-problem

In section 3.2 a code snippet from an SMT-problem is given. The full SMT-problem is shown below. We will explain this SMT-problem now. The word benchmark in line 1 indicates the start of the SMT-problem, followed by a name, which has no influence on the SMT-problem itself. On line 2 variables are defined using the ":extrafuns". Variables can be of different types (e.g., real or int). On line 3 the keyword ":formula" indicates the start of a single expression, which starts with the opening bracket of the first operand (line 4) and ends with the closing bracket of this operand (line 11). The end of the SMT-problem is indicated by a closing bracket on line 12.

$$(W_1 = 5) \wedge (W_2 = 10) \wedge ((X_1 + W_1 \leq X_2) \vee (X_2 + W_2 \leq X_1)) \tag{6}$$

The SMT-problem below is the translation of (6). The opening bracket of the "and" on line 4 is closed by the bracket on line 11. There are three expressions encapsulated in these brackets, which are all conjuncted. So by only writing one "and" operator multiple expressions can be conjuncted. One of these expressions is the "or" (disjunction), which itself contains two expression.

```
1  (benchmark name
2    :extrafuns ( (X1 int)(X2 int)(W1 int)(W2 int) )
3    :formula
4    (and
5          (= W1 5)
6          (= W2 10)
7          (or
8                (<= (+ X1 W1) X2)
9                (<= (+ X2 W2) X1)
10         )
11   )
12 )
```

B Tool description

To create a layout design from a schematic design the steps 1 up to 5 of the design flow illustrated in Figure 34 have to be executed. For this project a tool was created which executes steps 2, 3 and 4. It generates an SMT-problem from a set of input requirements (step 2). Next it automatically calls Yices (step 3), waits for Yices being finished and translates the output (if the SMT-problem was satisfiable) to a format easily readable by a SKILL-script (step 4). A SKILL-script that can generate the input file (step 1) has not been created yet. However, an alternative to this step has been provided to us (by Joost Rommes). It generates the configuration file from an layout design. Thus instead of having a schematic design as input, currently the cells in the schematic design need to be initialized in some temporary layout design, after which we can automatically generate the required input for step 2. In step 5 a SKILL-script is used to draw the layout from the this output. Notice that the resulting layout is only used to illustrate the outcome of the placement and routing problem, i.e., cells are drawn in layers chosen for their color, not for their representation of a layer in a chip.

The syntax for the input and output files will be described in the following sections.

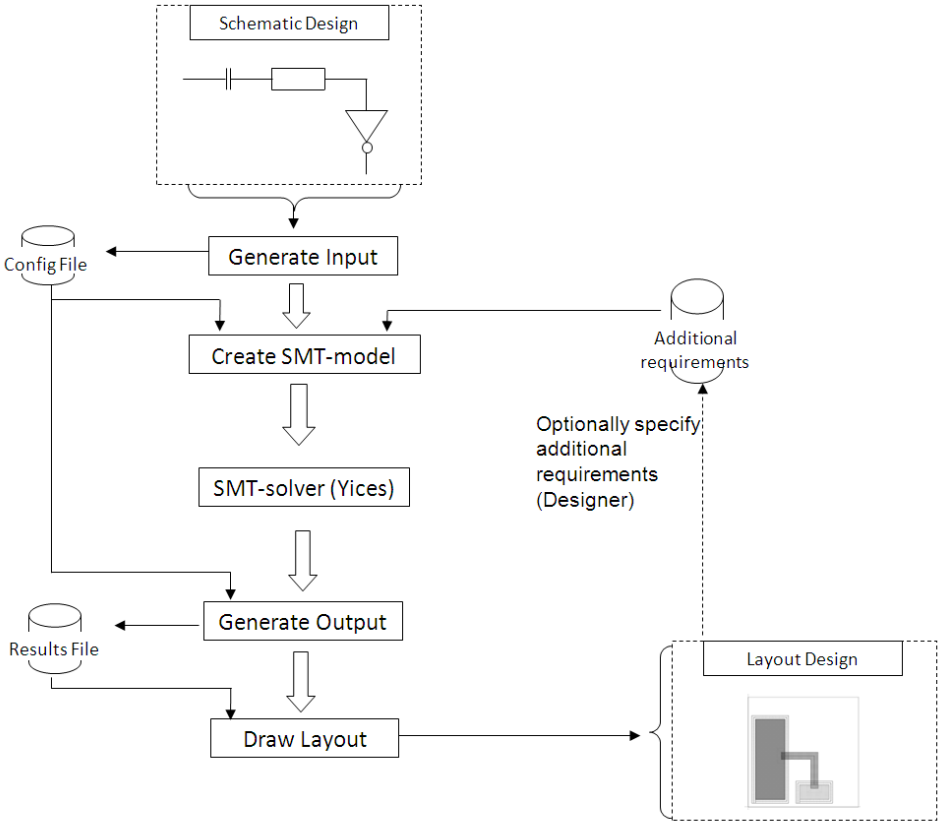


Figure 34: Illustration of the design flow from schematic design to a layout design.

B.1 Input file

The input created in step 1, uses the following syntax. Lines starting with a '.' indicate a section. Accepted sections are: GlobalRoutingSettings, Blocks, Area, Nets, Matching, Align. Table 5 shows

an example of a input file using these sections. The syntax used for each of these sections is described below. No overlap constraints will automatically be defined for all cells, and for all wires, such that no overlap is allowed. Note that the loop for reading the input file will stop after detecting an empty line in the file.

- .GlobalRoutingSettings: This identifier can be followed by two settings:
  1. "SteinerPoints" to set the number of Steiner points per wire (default is 1).
  2. "Jogs" to set the maximum total number of jogs of all wires (default is unlimited).
- .Area: This identifier is followed by the coordinates of the lower left and upper right corners of the chip area. If this identifier is not set, no limitation is imposed on the chip area. If set it must be: "0 0 W H", where W is the width of the chip, and H the height of the chip.
- .Blocks: This identifier is followed by block definitions, until an other section is encountered. The syntax must be: "*Name* $L_X L_Y U_X U_Y$ MM ['F']". The *Name* must be a unique name. (L_X, L_Y) are the coordinates of the lower left corner and (U_X, U_Y) coordinates of the upper right corner. The width and height of a cell are derived from these two points. The margin around the cell is specified by M and optionally an 'F' can be added to make the location of a cell fixed. The locations of where the cells are initialized are used to see which pin lies in which cell. So initializing cells on top of each other causes problems when defining the pin locations.
- .Nets: The pin definitions, and Nets (wires) are derived from this section. The syntax is as follows: "*NetName* $L_X L_Y U_X U_Y$ *Layer*". *NetName* indicates the name of the net. All pins with the same *NetName* are to be connected to each other. (L_X, L_Y) are the coordinates of the lower left corner and (U_X, U_Y) coordinates of the upper right corner of the pin. If the pin lies within the boundary of a cell, the pin is considered to be part of the cell, otherwise its just a pin located on the chip area (of which routing is not yet supported). *Layer* indicates the layer in which the pin is located. However this parameter is not of importance for our implementation.
- .Matching: This identifier is followed by a number of matching relations, with syntax: "$QN_1N_2 \ldots N_i$", with $(i \geq 1)$. Q can be 'A', '0', '1', '2' or '3' and is multiplied by 90 to obtain the angle of rotation, where 'A' allows any of the four rotation angles. N_1 to N_i are the names of cells that are matched.
- .Align: This identifier is followed by a number of alignment relations. The syntax of alignment is as follows: "$QN_1N_2 \ldots N_i$", with $i \geq 2$. In this case Q can be either 'A', 'X' or 'Y', which specifies the axis on which the cells must be aligned ('A' being Any, 'X' being vertically aligned, and 'Y' being horizontally aligned). N_1 to N_i $(i \geq 2)$ indicate the names of the cells that must be aligned, all in the same direction. Only alignment with respect to the lower left corner is implemented.

B.2 Output file

The output file has a similar syntax as the input file. Again '#' indicates comment, and '.' indicates a section. Notice that outputting and drawing the margins is only done to illustrate their existence. A margin is not an object in the layout design.

- .Blocks: A cell is specified as follows: "*Name* M $L_X L_Y U_X U_Y$ *Layer*". *Name* is preserved from the input, and must be unique. The margin M must be greater or equal to zero. It

```
1  #comment
2  .GlobalRoutingSettings
3  SteinerPoints 3
4  Jogs 3
5  .Area
6  0 0 15 15
7  .Blocks
8  block1 1 1 5 10 M0.5 F
9  block3 11 3 15 5 M0.0
10 .Nets
11 B 4 6 5 8 metal1
12 B 12 4 14 5 metal1
13 .Matching
14 0 block3
15 .Align
16 A block1 block3
```

Table 5: example of an input file

indicates the width of the margin that is drawn around the cell. The bounding box of the cell is specified by the lower left corner (L_X, L_Y) and the upper right corner (U_X, U_Y), similar as before. *Layer* indicates the Layer to draw the cell in. In our case the layer is chosen just for it color in Cadence and does not serve any other purpose.

- .Nets: This section indicates the locations of the pins, not the wires. The pins are specified by "*NetName* $L_X L_Y U_X U_Y$ *Layer*". The *NetName* and *Layer* are preserved from the input. If the cell in which this pin is located is moved, then the bounding box of the pin is adjusted according to the movement of the cell. The bounding box, in this case, is specified with respect to the chip area (0,0).
- .Wires: This section specifies which wires need to be drawn. The wires are specified by "*NetName* W M $X_1, Y_1 \ldots X_i, Y_i$". The *NetName* is the name of the wire, as specified in the input. The with is specified by W, with $(W \geq 0)$. M specifies the width of the margin that should be drawn around the wire $(M \geq 0)$. $X_1, Y_1 \ldots X_i, Y_i$ indicates a list of coordinates. In this list every two consecutive numbers are the coordinates of a point on the chip, through which the wire needs to be drawn. (these are the locations of the pins and Steiner points)
- .Align: When cells are aligned we draw a line indicating the alignment relation, which must be disregarded in an actual layout design. It is specified similar to a wire.
- .Boundary: A boundary is drawn to indicate the size of the chip . This is specified by a value for the line width followed by the coordinates of the four corners of the cell adjusted to the width of the line.

```
#
#Blocks (name margin boundingBox layer)
#
.Blocks
block1 0.50000 1.00000 1.00000 5.00000 10.00000 metal5
block3 0.50000 5.50000 3.00000 9.50000 5.00000 metal2
#
#Nets (NetName boundingBox Layer)
#
.Nets
B 4.00000 6.00000 5.00000 8.00000 metal1
B 6.50000 4.00000 8.50000 5.00000 metal1
#
#wires (name width margin ListOfCoordinates)
#
.Wires
B 1.00000 0.00000 5.00000 7.00000 6.00000 7.00000 6.50000 7.00000
    6.50000 6.50000 7.50000 6.50000 7.50000 6.00000 7.50000 5.00000
#
#Alignments (width pathCentreOfCorners)
#
.Align
#actual align
#
#Boundary
#
.Boundary
0.020000 -0.010000 -0.010000 15.010000 -0.010000 15.010000 15.010000
    -0.010000 15.010000
```

Table 6: example of an output file

C Input of the used examples

In order to reproduce the results shown section 5 of this document, the input configuration files used to obtain the different images have been included below. Some results were obtained by automatically minimizing the wire length or the total number of jogs others were not optimized at all.

Figure 23a was created with:

```
.GlobalRoutingSettings
SteinerPoints 7
WireLength 21
Jogs 3
.Area
0 0 15 15
.Blocks
block1 7.5 2.5 11.5 10.5 M0.5 F
block3 10.5 12.5 14.5 14.5 M0.5 F
```

```
10 #b 9 11 10 12 M0.5 F
11 .Nets
12 B 7.5 3.5 8.5 5.5 metal1
13 B 11.5 12.5 13.5 13.5 metal1
```

Figure 23b was created with:

```
 1 .GlobalRoutingSettings
 2 SteinerPoints 7
 3 WireLength 20.9
 4 .Area
 5 0 0 15 15
 6 .Blocks
 7 block1 7.5 2.5 11.5 10.5 M0.5 F
 8 block3 10.5 12.5 14.5 14.5 M0.5 F
 9 #b 9 11 10 12 M0.5 F
10 .Nets
11 B 7.5 3.5 8.5 5.5 metal1
12 B 11.5 12.5 13.5 13.5 metal1
```

Figure 24a and 24b were created with (minimizing # jogs):

```
 1 .GlobalRoutingSettings
 2 SteinerPoints 3
 3 WireLength 7
 4 .Area
 5 0 0 15 15
 6 .Blocks
 7 block1 7.5 2.5 11.5 10.5 M0.5 F
 8 block3  2.5   8.5   6.5   10.5 M0.5 F
 9 #b 9 11 10 12 M0.5 F
10 .Nets
11 B 7.5 3.5 8.5 5.5 metal1
12 B     3.5   8.5   5.5   9.5   metal1
```

Figure 25 and 26a - 26d were created by minimizing the wire length in:

```
 1 .GlobalRoutingSettings
 2 SteinerPoints 5
 3 .Area
 4 0 0 15 15
 5 .Blocks
 6 block1 7.5 2.5 11.5 10.5 M0.5 F
 7 block3 10.5 12.5 14.5 14.5 M0.5
 8 #b 9 11 10 12 M0.5 F
 9 .Nets
10 B 7.5 3.5 8.5 5.5 metal1
11 B 11.5 12.5 13.5 13.5 metal1
```

Figure 28a was created with:

```
1 .GlobalRoutingSettings
```

```
 2  SteinerPoints 2
 3  .Area
 4  0 0 40 20
 5  .Blocks
 6  C1 37 0 40 20 M0.0 F
 7  C2 14 0 17 10 M0.0 F
 8  C3 7 0 10 10 M0.0 F
 9  C4 0 0 3 20 M0.0 F
10  C5 19 0 37 10 M0.0 F
11  .Nets
12  P9      37 10.4 38 10.6 metal1
13  P8 37 11.4 38 11.6 metal1
14  P7 37 12.4 38 12.6 metal1
15  P6 37 13.4 38 13.6 metal1
16  P5 37 14.4 38 14.6 metal1
17  P4 37 15.4 38 15.6 metal1
18  P3 37 16.4 38 16.6 metal1
19  P2 37 17.4 38 17.6 metal1
20  P1 37 18.4 38 18.6 metal1
21  P0 37 19.4 38 19.6 metal1
22  P9 16 0.4 17 0.6 metal1
23  P8 16 1.4 17 1.6 metal1
24  P7      14      1.4     15      1.6     metal1
25  P6      14      0.4     15      0.6     metal1
26  P5      9       0.4     10      0.6     metal1
27  P4      9       1.4     10      1.6     metal1
28  P3      7       1.4     8       1.6     metal1
29  P2      7       0.4     8       0.6     metal1
30  P1      2       0.4     3       0.6     metal1
31  P0      2       1.4     3       1.6     metal1
```

Figure 30b was created by (minimized for # jogs):

```
 1  .GlobalRoutingSettings
 2  SteinerPoints 4
 3  .Area
 4  0 0 100 100
 5  #Instances (name boundingbox)
 6  #
 7  .Blocks
 8  tripInv 8.244 28.676 37.27 50.506 M0.0 F
 9  Res1 8.244 52.026 38.816 55.442 M0.0 F
10  Res2 31.8 25.24 37.27 27.156 M0.0 F
11  C1 8.744 56.962 36.77 57.922 M0.0 F
12  C2 8.744 17.64 36.77 23.72 M0.0 F
13  fbkRes 12.006 62.38 35.054 64.296 M0.0 F
14  fbkCap 5.806 65.816 41.254 68.056 M0.0 F
15  serResB1 18.272 5.776 28.788 13.702 M0.0 F
16  serResT1 18.272 69.056 28.788 76.982 M0.0 F
17  decoup4 29.468 68.976 41.074 76.66 M0.0 F
```

```
18 decoup3 5.986 68.976 17.592 76.66 M0.0 F
19 decoup2 29.468 5.896 41.074 13.58 M0.0 F
20 decoup1 5.986 5.896 17.592 13.58 M0.0 F
21 #
22 #Nets (name pin_boundingbox layer)
23 #
24 .Nets
25 P1 5.806 66.602 6.478 66.914 metal4
26 P1 14.39 63.338 14.71 64.296 metal4
27 P2 32.35 63.34 32.67 64.296 metal4
28 P2 40.59 66.624 41.254 66.936 metal4
29 P3 11.806 12.58 12.106 13.58 metal2
30 P3 23.584 76.098 23.884 76.982 metal2
31 P4 8.744 57.24 9.416 57.442 metal3
32 P4 20.782 49.506 21.102 50.506 metal3
33 P5 27.65 28.676 27.97 29.676 metal1
34 P5 36.638 26.206 37.27 26.408 metal1
35 P6 31.8 26.01 32.45 26.212 metal1
36 P6 8.744 20.7 9.41 20.902 metal1
37 P7 36.1 57.25 36.77 57.44 metal1
38 P7 38.172 54.484 38.816 54.674 metal1
39 P8l 5.986 9.236 7.166 10.236 metal1
40 P8r 39.896 9.236 41.074 10.236 metal1
41 P8l 23.042 5.776 24.042 6.776 metal1
42 P8r 26.042 5.776 27.042 6.776 metal1
43 P10 12.006 63.084 12.282 63.596 metal4
44 P10 8.244 39.572 8.474 40.084 metal5
45 P11 34.824 63.084 35.054 63.596 metal4
46 P11 36.472 38.842 37.27 39.354 metal5
```

Figure 30c was created by (minimized for # jogs):

```
 1 .GlobalRoutingSettings
 2 SteinerPoints 4
 3 .Area
 4 0 0 45 80
 5 #Instances (name boundingbox)
 6 #
 7 .Blocks
 8 tripInv 8.244 28.676 37.27 50.506 M0.0 F
 9 Res1 8.244 52.026 38.816 55.442 M1.0
10 Res2 31.8 25.24 37.27 27.156 M1.0
11 C1 8.744 56.962 36.77 57.922 M1.0
12 C2 8.744 17.64 36.77 23.72 M1.0
13 fbkRes 12.006 62.38 35.054 64.296 M1.0
14 fbkCap 5.806 65.816 41.254 68.056 M1.0
15 serResB1 18.272 5.776 28.788 13.702 M0.0 F
16 serResT1 18.272 69.056 28.788 76.982 M0.0 F
17 decoup4 29.468 68.976 41.074 76.66 M0.0 F
18 decoup3 5.986 68.976 17.592 76.66 M0.0 F
```

```
19 decoup2 29.468 5.896 41.074 13.58 M0.0 F
20 decoup1 5.986 5.896 17.592 13.58 M0.0 F
21 #
22 #Nets (name pin_boundingbox layer)
23 #
24 .Nets
25 P1 5.806 66.602 6.478 66.914 metal4
26 P1 14.39 63.338 14.71 64.296 metal4
27 P2 32.35 63.34 32.67 64.296 metal4
28 P2 40.59 66.624 41.254 66.936 metal4
29 P3 11.806 12.58 12.106 13.58 metal2
30 P3 23.584 76.098 23.884 76.982 metal2
31 P4 8.744 57.24 9.416 57.442 metal3
32 P4 20.782 49.506 21.102 50.506 metal3
33 P5 27.65 28.676 27.97 29.676 metal1
34 P5 36.638 26.206 37.27 26.408 metal1
35 P6 31.8 26.01 32.45 26.212 metal1
36 P6 8.744 20.7 9.41 20.902 metal1
37 P7 36.1 57.25 36.77 57.44 metal1
38 P7 38.172 54.484 38.816 54.674 metal1
39 P8l 5.986 9.236 7.166 10.236 metal1
40 P8r 39.896 9.236 41.074 10.236 metal1
41 P8l 23.042 5.776 24.042 6.776 metal1
42 P8r 26.042 5.776 27.042 6.776 metal1
43 P10 12.006 63.084 12.282 63.596 metal4
44 P10 8.244 39.572 8.474 40.084 metal5
45 P11 34.824 63.084 35.054 63.596 metal4
46 P11 36.472 38.842 37.27 39.354 metal5
```

Figure 31a was created by (minimized for # jogs):

```
 1 .GlobalRoutingSettings
 2 SteinerPoints 4
 3 .Area
 4 0 0 45 80
 5 #Instances (name boundingbox)
 6 #
 7 .Blocks
 8 tripInv 8.244 28.676 37.27 50.506 M1.0
 9 Res1 8.244 52.026 38.816 55.442 M1.0
10 Res2 31.8 25.24 37.27 27.156 M1.0
11 C1 8.744 56.962 36.77 57.922 M1.0
12 C2 8.744 17.64 36.77 23.72 M1.0
13 fbkRes 12.006 62.38 35.054 64.296 M1.0
14 fbkCap 5.806 65.816 41.254 68.056 M1.0
15 serResB1 18.272 5.776 28.788 13.702 M0.0 F
16 serResT1 18.272 69.056 28.788 76.982 M0.0 F
17 decoup4 29.468 68.976 41.074 76.66 M0.0 F
18 decoup3 5.986 68.976 17.592 76.66 M0.0 F
19 decoup2 29.468 5.896 41.074 13.58 M0.0 F
```

```
20 decoup1 5.986 5.896 17.592 13.58 M0.0 F
21 #
22 #Nets (name pin_boundingbox layer)
23 #
24 .Nets
25 P1 5.806 66.602 6.478 66.914 metal4
26 P1 14.39 63.338 14.71 64.296 metal4
27 P2 32.35 63.34 32.67 64.296 metal4
28 P2 40.59 66.624 41.254 66.936 metal4
29 P3 11.806 12.58 12.106 13.58 metal2
30 P3 23.584 76.098 23.884 76.982 metal2
31 P4 8.744 57.24 9.416 57.442 metal3
32 P4 20.782 49.506 21.102 50.506 metal3
33 P5 27.65 28.676 27.97 29.676 metal1
34 P5 36.638 26.206 37.27 26.408 metal1
35 P6 31.8 26.01 32.45 26.212 metal1
36 P6 8.744 20.7 9.41 20.902 metal1
37 P7 36.1 57.25 36.77 57.44 metal1
38 P7 38.172 54.484 38.816 54.674 metal1
39 P8l 5.986 9.236 7.166 10.236 metal1
40 P8r 39.896 9.236 41.074 10.236 metal1
41 P8l 23.042 5.776 24.042 6.776 metal1
42 P8r 26.042 5.776 27.042 6.776 metal1
43 P10 12.006 63.084 12.282 63.596 metal4
44 P10 8.244 39.572 8.474 40.084 metal5
45 P11 34.824 63.084 35.054 63.596 metal4
46 P11 36.472 38.842 37.27 39.354 metal5
```

Figure 31b was created by (minimized for # jogs):

```
1 .GlobalRoutingSettings
2 SteinerPoints 4
3 Jogs 27
4 .Area
5 0 0 45 80
6 #Instances (name boundingbox)
7 #
8 .Blocks
9 tripInv 8.244 28.676 37.27 50.506 M1.0
10 Res1 8.244 52.026 38.816 55.442 M1.0
11 Res2 31.8 25.24 37.27 27.156 M1.0
12 C1 8.744 56.962 36.77 57.922 M1.0
13 C2 8.744 17.64 36.77 23.72 M1.0
14 fbkRes 12.006 62.38 35.054 64.296 M1.0
15 fbkCap 5.806 65.816 41.254 68.056 M1.0
16 serResB1 18.272 5.776 28.788 13.702 M1.0
17 serResT1 18.272 69.056 28.788 76.982 M1.0
18 decoup4 29.468 68.976 41.074 76.66 M1.0
19 decoup3 5.986 68.976 17.592 76.66 M1.0
20 decoup2 29.468 5.896 41.074 13.58 M1.0
```

```
21 decoup1 5.986 5.896 17.592 13.58 M1.0
22 #
23 #Nets (name pin_boundingbox layer)
24 #
25 .Nets
26 P1 5.806 66.602 6.478 66.914 metal4
27 P1 14.39 63.338 14.71 64.296 metal4
28 P2 32.35 63.34 32.67 64.296 metal4
29 P2 40.59 66.624 41.254 66.936 metal4
30 P3 11.806 12.58 12.106 13.58 metal2
31 P3 23.584 76.098 23.884 76.982 metal2
32 P4 8.744 57.24 9.416 57.442 metal3
33 P4 20.782 49.506 21.102 50.506 metal3
34 P5 27.65 28.676 27.97 29.676 metal1
35 P5 36.638 26.206 37.27 26.408 metal1
36 P6 31.8 26.01 32.45 26.212 metal1
37 P6 8.744 20.7 9.41 20.902 metal1
38 P7 36.1 57.25 36.77 57.44 metal1
39 P7 38.172 54.484 38.816 54.674 metal1
40 P8l 5.986 9.236 7.166 10.236 metal1
41 P8r 39.896 9.236 41.074 10.236 metal1
42 P8l 23.042 5.776 24.042 6.776 metal1
43 P8r 26.042 5.776 27.042 6.776 metal1
44 P10 12.006 63.084 12.282 63.596 metal4
45 P10 8.244 39.572 8.474 40.084 metal5
46 P11 34.824 63.084 35.054 63.596 metal4
47 P11 36.472 38.842 37.27 39.354 metal5
48 .Align
49 Y decoup1 serResB1 decoup2
50 Y decoup3 serResT1 decoup4
```