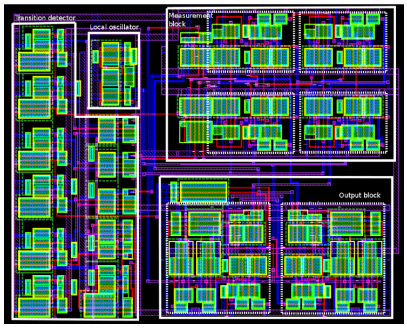
Alma Mater Studiorum · University of Bologna

VLSI (Very Large Scale Integration) design optimization problem: SAT solution



Combinatorial Decision Making and Optimization

Giuseppe Murro (giuseppe.murro@studio.unibo.it)
Giuseppe Boezio (giuseppe.boezio@studio.unibo.it)
Salvatore Pisciotta (salvatore.pisciotta2@studio.unibo.it)

August 29, 2021

Contents

1 Introduction

VLSI (Very Large Scale Integration) refers to the trend of integrating circuits into silicon chips. A typical example is the smartphone. The modern trend of shrinking transistor sizes, allowing engineers to fit more and more transistors into the same area of silicon, has pushed the integration of more and more functions of cellphone circuitry into a single silicon die (i.e. plate). This enabled the modern cellphone to mature into a powerful tool that shrank from the size of a large brick-sized unit to a device small enough to comfortably carry in a pocket or purse, with a video camera, touchscreen, and other advanced features. The aim of this project is to design a solution for the problem of put all the given input circuits in the plate optimizing its height.

2 Instances

2.1 Input

Input instances are presented using the following variables:

- w = width of the silicon plate
- n = number of circuits to insert in the plate
- x_i = horizontal dimension of i^{th} circuit
- y_i = vertical dimension of i^{th} circuit

where an instance is written in the following way:

w
n
x_0 y_0
x_1 y_1
...

2.2 Output

Starting from input instances data, once that the optimization process is finished it is given the output as:

w l
n
x_0 y_0 px_0 py_0
x_1 y_1 px_1 py_1
...

where:

- w, n, y_i, x_i are the same of the input
- l = maximum height reached by the circuits configuration
- px_i = horizontal left-down point coordinate of the i^{th} circuit block
- py_i = vertical left-down point coordinate of the i^{th} circuit block

An output instance is shown in the figure 2.1

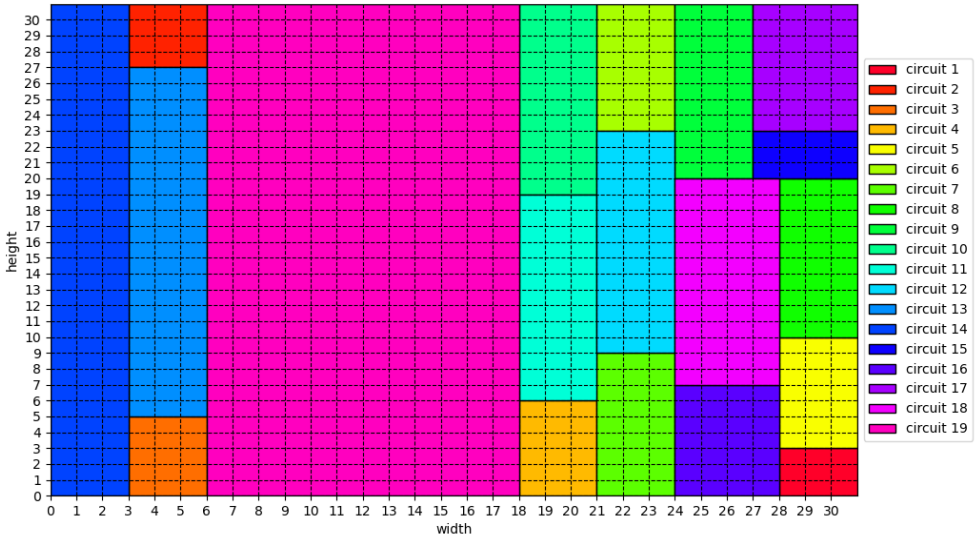


Figure 2.1: Output of an instance with w = 31 and n = 19

3 Modelling

3.1 Preliminary Reasoning

3.1.1 Modelling variables

The first part of the modelling process is the one of deciding how to model the problem and define the variables to use. In SAT, the modelling process is done using formulae in propositional logic composed of boolean variables and the logical operators that in this case are expressed using relational operators provided by Z3Py library.

In order to describe the worst possible value of the height of the plate, it has been introduced a constant value l_{max}, computed as:

$$l_{max} = \begin{cases} max\_y & \left\lceil \frac{\sum_{i=1}^{n} y\_i}{\lfloor \frac{w}{max\_x} \rfloor} \right\rceil < max\_y \\ \left\lceil \frac{\sum_{i=1}^{n} y\_i}{\lfloor \frac{w}{max\_x} \rfloor} \right\rceil & \text{otherwise} \end{cases} \tag{3.1}$$

Where max_x and max_y are respectively the maximum value of lengths and heights of the input circuits. The choice of this value has been done taking in consideration the unluckiest case in which all circuits dimensions are max_x and max_y, so l_{max} is computed calculating the number of circuits that can be fitted horizontally and then how many rows of these blocks are needed vertically. But sometimes this approximation is not feasible if the result is less then max_y, so in that case could be used max_y as value for l_{max}.

Instead the variables used to model this problem are the followings:

- **plate** A 3-dimensional matrix ($w \times l_{max}$) where each cell in the plate is an array of n boolean variables where only one of them is true and represent the correspondent circuit:

$$\forall i \in \{1 \,..\, l_{max}\}\ \forall j \in \{1 \,..\, w\}\ \forall k \in \{1 \,..\, n\} : p_{i,j,k} \in \mathbb{B}$$

- **height of the plate** Maximum height reached by a circuit. It is the variable to minimize. It is represented as an array one-hot encoded where the index of the only true variable is the height reached.

$$\forall i \in \{1 \,..\, l_{max}\}\ l_i \in \mathbb{B}$$

3.1.2 Symmetries

Looking at the feasible configuration of optimal solutions it was discover that solutions with the same l value are equivalent, so there could be exponentially many equal solution and some of them are symmetries.

The possible symmetries that it has been detected are:

- Vertical flip
- Horizontal flip
- 180° rotation

3.2 Constraints

3.2.1 Main constraints

Using the variables that have been defined it was possible to impose the main constraints of the problem:

- **Overlapping**: To avoid the possible overlapping of circuits the main idea was to impose that each cell in the plate has at most one true value. In this way circuits cannot be placed in the same position. For each cell in the matrix it has been used a naive encoding of the constraint at-most-one, so called "pairwise encoding" as mentioned in the paper by Nguyen V. at al. [1]:

$$\bigwedge_{1 \leq i \leq l_{max}} \bigwedge_{1 \leq j \leq w} \bigwedge_{1 \leq k < q \leq n} \neg(p_{ijk} \wedge p_{ijq})$$

- **Circuits positioning**: Since the encoding used is in the form of a matrix, when a given circuit number is assigned to a given cell, all neighboring cells should be consistent with respect to the height and the width of that circuit. So for each circuit it has been imposed a constraint which works like a sliding windows and it encodes each possible position of each block inside the matrix.
  In order to define this constraint, the following steps are executed:
  1. iterate over all the n circuits: $\forall k \in \{1 \,..\, n\}$
  2. iterate over all the coordinates where each circuit can fit its lower-left corner: $\forall i \in \{1 \,..\, l_{max} - y_k + 1\}$ and $\forall j \in \{1 \,..\, w - x_k + 1\}$
  3. use a sliding window like algorithm to set as True all the cells containing that circuit and as False all others cells, in order to create a *circuit_positioning* array. For instance in a matrix $2x2$ a circuit $2x1$ has as *circuit_positioning*: $(p_{00} \wedge p_{10} \wedge \neg p_{01} \wedge \neg p_{11}) \vee (p_{01} \wedge p_{11} \wedge \neg p_{00} \wedge \neg p_{10})$
  4. an *exactly_one* constraint is used to make sure that each circuit is in one and only one of the *circuit_positioning*
- **One hot encoding height**: In order to obtain the exact value of the height it must be imposed that in the array length there is exactly one value true. So the combination of

at-least-one and at-most-one has been used:

$$\bigwedge_{1 \leq i < j \leq l_{max}} \neg(l_i \wedge l_j) \wedge \bigvee_{1 \leq i \leq l_{max}} l_i$$

- **Length construction**: In the length array, it has to be imposed that the only one assigned true value of the array must be consistent with the effective maximum height of the plate. So it has to be imposed that the correspondent height value in the array must be true if and only if in the plate matrix there is at least one true value at the correspondent height and there must not be true value in the upper elements:

$$\forall i \in \{1 \,..\, l_{max}\} \left(l_i \iff \bigvee_{1 \leq j \leq w, 1 \leq k \leq n} p_{ijk} \wedge \forall q \in \{i+1 \,..\, l_{max}\} \left(\neg \bigvee_{1 \leq j \leq w, 1 \leq k \leq n} p_{qjk} \right) \right)$$

3.2.2 Symmetry breaking constraints

In order to break the symmetries, also the ones described in 3.1.2, it was thought to put the circuit with the biggest height always on the coordinate $(0, 0)$, in this way the biggest circuit is forced to be always at the bottom left corner of the plate, rejecting all other configurations where this circuit is in another position (including 180° rotation and vertical flip of the solution).

Moreover it has been implemented an adapted version of **lex_lesseq** ordering constraint for boolean variables which works as follow:

$$z3\_lex\_lesseq([X_1, X_2, .., X_m], [Y_1, Y_2, .., Y_m]) =$$

$$z3\_less\_eq(X_1, Y_1) \wedge$$

$$\left(\bigwedge_{1 \leq k \leq n} X_{1,k} = Y_{1,k} \implies z3\_less\_eq(X_2, Y_2) \right) \wedge$$

$$\left(\left(\bigwedge_{1 \leq k \leq n} X_{1,k} = Y_{1,k} \right) \wedge \left(\bigwedge_{1 \leq k \leq n} X_{2,k} = Y_{2,k} \right) \implies z3\_less\_eq(X_3, Y_3) \right) \wedge ...$$

$$\left(\left(\bigwedge_{1 \leq k \leq n} X_{1,k} = Y_{1,k} \right) \wedge \left(\bigwedge_{1 \leq k \leq n} X_{2,k} = Y_{2,k} \right) \wedge ... \wedge \left(\bigwedge_{1 \leq k \leq n} X_{m-1,k} = Y_{m-1,k} \right) \implies z3\_less\_eq(X_m, Y_m) \right)$$

where $[X_1, X_2, .., X_m]$ is the the flattening of the matrix p, so each element is an array of boolean variables and *z3_less_eq* is a function able to compare two boolean arrays. Namely also this function is inspired by the lexicographic order pattern, but it states that an array is lower than another if in the first one a True variable appears before the other:

$$z3\_less\_eq([X_1, X_2, .., X_n], [Y_1, Y_2, .., Y_n]) =$$

$$(X_1 \vee \neg(Y_1)) \wedge$$

$$(X_1 = Y_1 \implies X_2 \vee \neg Y_2) \wedge$$

$$(X_1 = Y_1 \wedge X_2 = Y_2 \implies X_3 \vee \neg Y_3) \wedge ...$$
$$(X_1 = Y_1 \wedge X_2 = Y_2 \wedge ... \wedge X_{n-1} = Y_{n-1} \implies X_n \vee \neg Y_n)$$

The *z3_lex_lesseq* constraint has been used three times in order to break all the symmetries:

- **Horizontal Flip**

$$\forall i \in \{1 .. l_{max}\}, j \in \{w .. 1\} : lex\_lesseq([p_{00}, p_{01}, ..., p_{l_{max}w}], [p_{ij}])$$

- **Vertical Flip**

$$\forall i \in \{l_{max} .. 1\}, j \in \{1 .. w\} : lex\_lesseq([p_{00}, p_{01}, ..., p_{l_{max}w}], [p_{ij}])$$

- **180° Rotation**

$$\forall i \in \{l_{max} .. 1\}, j \in \{w .. 1\} : lex\_lesseq([p_{00}, p_{01}, ..., p_{l_{max}w}], [p_{ij}])$$

3.3 Alternative encoding of at-most-one constraint

3.3.1 Pairwise encoding

Whereas the at-least-one constraint can be easily encoded by a single clause, the encoding of the at-most-one constraint is more complicated. The *at-most-one pairwise* encoding, as showed in 3.2.1, is the traditional way of encode the at-most-one constraint into SAT. Although this encoding does not need any auxiliary variables, it requires $\binom{n}{2}$ clauses. Consequently, this method may result in large formulas on problems with large domains.

3.3.2 Bimander encoding

Alternative encodings were investigated in order to try to improve the results. The paper by Nguyen V. at al. [1] proposes a new encoding for the at-most-one constraint, the so-called *at-most-one bimander* encoding. This encoding is based on two others encodings: the at-most-one binary encoding and the at-most-one commander encoding.

The idea is to split a set of propositional variables $[x_1, x_2, .., x_n]$ into m (between 1 and n) disjoint subsets $[G_1, G_2, .., G_m]$ such that each subset G_i consists of $g = \lceil \frac{n}{m} \rceil$ variables. It is also required to introduce a set of auxiliary propositional variables $[b_1, b_2, .., b_{\lceil \log_2 m \rceil}]$ which play a direct role in the clauses.

The at-most-one bimander encoding is the conjunction of the following clauses:

1. At most one variable in each subset can be True. One must encode this constraint for each subset G_i, $1 \leq i \leq m$, by using the at-most-one pairwise encoding:

$$\bigwedge_{i=1}^{n} amo\_pairwise(G_i)$$

2. The following clauses are generated by the constraints between each variable and auxiliary variables in a subset:

$$\bigwedge_{i=1}^{m} \bigwedge_{h=1}^{g} \bigwedge_{j=1}^{\lceil \log_2 m \rceil} \neg x_{i,h} \vee \phi(i,j)$$

where $\phi(i,j)$ denotes b_j (or $\neg b_j$) if the bit j of $i-1$ represented by a unique binary string is 1 (or 0).

In the actual implementation $m = \lceil \frac{n}{2} \rceil$ was chosen as suggested by the paper. This kind of encoding uses less clauses than pairwise encoding, exactly $n \log_2 n - \frac{n}{2}$, however it requires $\lceil \log_2 n \rceil - 1$ auxiliary variables each time that this at-most-one constraint is used.

3.4 Rotation

3.4.1 Introduction

A variation of the problem consists of allowing a rotation of circuits. This means that the final height and length of each circuit is not necessary equals to dimensions provided in input but they could be swapped.

3.4.2 Constraints

Starting from this assumption, a new array of boolean **rotation** has been added to the previous model to express the fact that original dimensions of the i^{th} circuit have been swapped or not:

$$\forall i \in \{1 \,..\, n\} \; r_i \in \mathbb{B}$$

From a logical point of view, constraints are the same of the previous model since dimension of each circuits are not taken into account as provided in input but how they are actually used.

The only one constraint that changes is the one about circuits positioning in order to take into account the possible rotation of each piece. Therefore, we duplicated it such that the first one deals with not rotated pieces while the second one deals with pieces that are rotated. Finally, an *exactly one constraint* ensures that each circuit is rotated or not rotated.

3.4.3 Output

Output is the same of the initial model but it was added to each row a letter R whether the circuit has been rotated. Following an example:

w l
n
x_0 y_0 px_0 py_0
x_1 y_1 px_1 py_1 R
...

4 Search

To compare the performance of the models, we have used the standard search method provided by Z3Py which is a class called *Solver*. This is suitable for our problem because it allows to get an assignment for each variable such that it provides a solution that satisfies the given constraints. In order to obtain the optimal solution, it was implemented a branch and bound strategy, using a cycle comparing the last founded solution and accept it only if it is better with respect to the best solution supplied until that moment. In order to do this, the following constraint has been implemented:

$$\bigvee_{i=1}^{length\_sol} l_i$$

where *length_sol* is the actual length of the plate during each iteration.

5 Results

5.0.1 Hardware used

Our experiments were conducted on a machine with those specifications:

- Intel® Core™ i7-10750H (6 core)
- RAM DDR4 16 GB
- NVIDIA® GeForce RTX™ 2060 6 GB
- Windows 10 Home 64

The version of the interpreter used is *Python 3.8.10* with *z3-solver 4.8.12.0*.

5.0.2 Execution

Different models have been tested:

- final: the simplest model which implements the constraints listed in section 3.2.1
- bimander: the one with the alternative encoding of the at-most-one constraint (see section 3.3)
- symmetries: the model which contains also the symmetry breaking constraints (see section 3.2.2)

In the figure 5.1 it can be seen the execution of the implemented models applied on all the instances. The ones for those models don't provide a solution within 300 seconds, they were not shown. As it can be noticed the final model is the one that solves the highest number of them and it provides the solution always faster with respect to the others. Therefore even if bimander model is more sophisticated than the simplest one, it does not speed up the execution as expected. Also the model with the symmetries performs worse than the others due to the high number of complex constraints that it must satisfy.

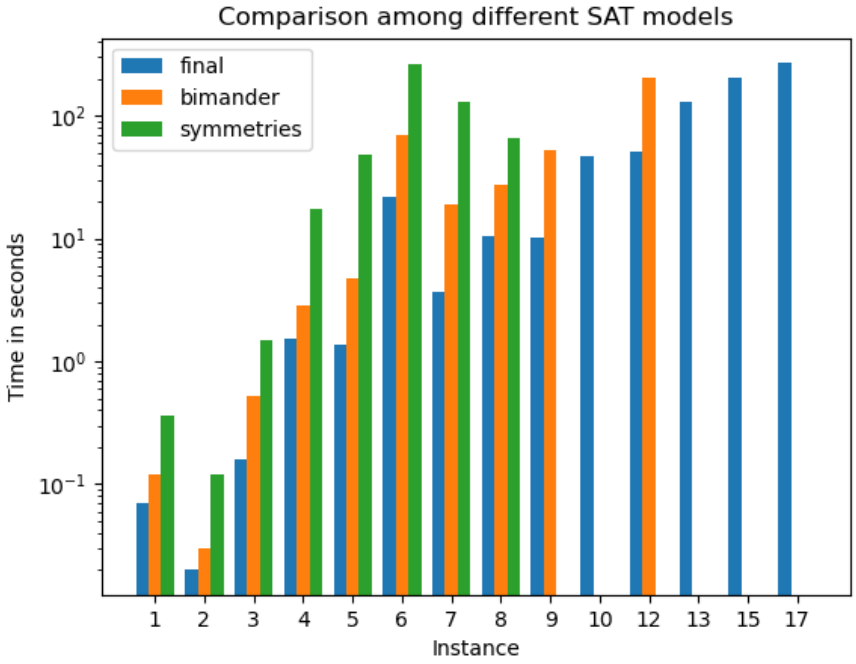


Figure 5.1

In figure 5.2 it can be seen the execution of the rotation model on the instances. Despite it does not provide a solution for a large number of instance, it can be highlighted that the behaviour of it is almost the same of the bimander and symmetries models.

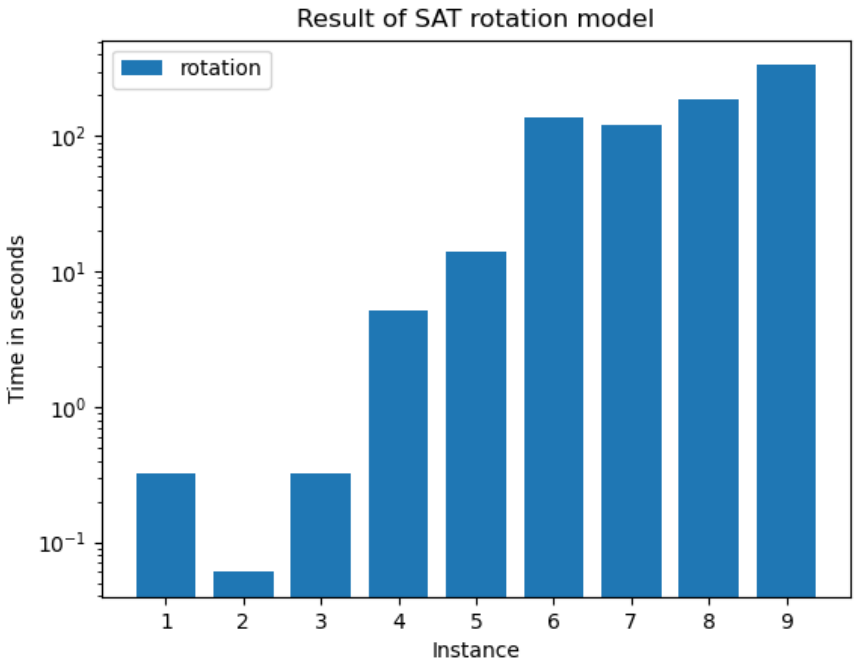

Result of SAT rotation model
rotation
Time in seconds
10^2
10^1
10^0
10^{-1}
1
2
3
4
5
6
7
8
9
Instance

Figure 5.2

Bibliography

[1] Van-Hau Nguyen and Sn Mai Thái. "A New Method to Encode the At-Most-One Constraint into SAT". In: Dec. 2015, pp. 1–8. DOI: 10.1145/2833258.2833293.