SMT-based Placement for System-on-Chip Design

Sebastian Pointner[*] Sven Wenzek[‡] Robert Wille[*][†]
[*]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[†]Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria
[‡]EPOS GmbH & Co KG - An Infineon Company, Duisburg, Germany
Email: sebastian.pointner@jku.at sven.wenzek@epos-d.com robert.wille@jku.at

***Abstract*—The utilization of *System on Chips* (SoCs) for short-living consumer applications has become very popular over the last decades. Because of that, more and more effort has been put into the physical design of SoCs and especially into the so-called macro placement step in order to keep the final price suitable for mass production. How to guarantee that the SoC is realized based on a minimal die area remains a challenging task. Current state-of-the-art solutions for this *macro placement*-problem mostly try to tackle the problem based on meta-heuristic- and genetic algorithms. However, although such methods are commonly used for macro placement, they can not guarantee that the macro placement and, therefore the die size is optimal. In this work, we are proposing the utilization of modern satisfiability solvers in order to generate optimized macro placements. To this end, we symbolically formulate the placement problem and forward it to a solver which allows us to obtain optimized solutions for the macro placement problem. In case this is not possible, search space pruning is employed which does not allow to employ the full optimization strategy anymore but still determines feasible results. We demonstrate the approach in experiments and made the resulting tool available as open-source.**

I. Introduction

The technological progress achieved by academia as well as industry within the last decades for *Integrated Circuits* (ICs) is remarkable. Applications of ICs like smartphones or tablets have become essential parts of our modern life. To this end, the design of *Application Specific ICs* (ASICs) and especially of so-called *Systems on Chips* (SoCs) has received lots of attention. In order to stay compatible with their products, semiconductor companies have to steadily optimize their time-to-market duration. One way to do so is to not do a classical ASIC design from scratch, but to re-use existing components. This is useful since more and more systems are composed of recurring components (e.g. a CPU, etc.) which can be pre-designed, pre-verified, and eventually be used for the design of different SoCs. These components, which are also called macros, can be used in a drag and drop fashion which allows to substantially speed up the entire design process.

Leading to a design flow as sketched in Fig. 1: This simplified design flow starts with the specification as the input of the flow and ends with a tested chip that is ready to be shipped to the customers. Compared to a standard cell-based ASIC design flow, the design flow for SoCs, which is based on the usage of pre-designed and verified macros, shifts its major focus more to the physical design steps. This is because the macros which are used in the SoC have been already designed and verified, while their physical design remains to be open. The main task of physical design is to eventually map the respective components onto the chip. The result of physical design is ready to be forwarded for production.

In this work, we particularly consider the step of macro placement within this process. Macro placement as part of the physical design phase tries to place the macros on the die while trying to find a placement which minimizes a certain cost function (e.g. the needed die area for the SoC). Finding an optimal or as good as possible placement is essential in order to promote new SoC success on the market. In order to determine a good macro placement, multiple approaches have been published in the last decades. However, most of these approaches are based on *Simulated Annealing* (SA) [1] or *Genetic Algorithms* (GA) [2] which can not guarantee that the found placement is an optimal placement.

In the following, we propose the utilization of modern reasoning engines such as solvers for *Satisfiability Modulo Theories* (SMT) for the macro placement within the physical design for SoCs. To this end, we formulate the macro placement problem symbolically and forward it to a reasoning engine. In order to ensure that certain properties are satisfied, particular constraints for the placement are added (e.g. no overlapping constraints). Based on these constraints, the reasoning engine can prove whether there is a valid placement possible or not. Besides that, the consideration of a cost function on top of these constraints allows to address corresponding optimization objectives (such as reducing the required die area). By utilizing optimizing solvers for the SMT problem, we are capable to obtain optimized placements for all macros or, by additionally employing search space pruning in case the complexity gets too large, reasonably sized layouts for larger designs. We demonstrate the applicability of the proposed method based on a set of benchmarks for macro placement and made the corresponding implementation available open-source.

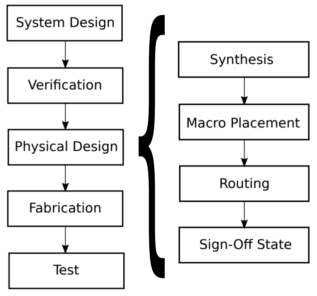

Fig. 1: Abstract System-on-Chip Design Flow.

The remainder of this work is structured as follows: The next section briefly reviews the physical design for SoCs and the state-of-the-art in terms of macro placement, providing the motivation for this work. In the same section, also the general idea of the proposed solution is sketched. Afterwards, the implementation of the proposed idea is described in Section III. Finally, Section IV demonstrates some results obtained by the tool before this work is concluded in Section V.

II. Motivation and General Idea

This section first briefly reviews the phase of physical design for SoCs – with a particular focus on macro placement. Afterwards, we discuss the current state-of-the-art in macro placement including its limitations. Motivated by that, we introduce the general idea for the macro placement approach proposed in this work which aims to overcome the discussed limitations of the state-of-the-art.

A. System-on-Chip Physical Design

Physical design for SoCs can be seen as the design phase between functional designing/verifying a chip and getting the final layout data ready for chip fabrication. To this end, the input data for physical design can e.g. be an HDL-design based on a certain target technology in combination with particular design constraints. Therefore, the design is based on the instatitiation of pre-designed macros (i.e. functional blocks like an interrupt controller or an SPI interface). In order to obtain the final layout data, multiple design steps are conducted. These design steps, seen in the abstract design flow sketched in Fig. 1, include (1) synthesis, (2) macro placement, (3) routing, as well as the (4) final sign-off state. More precisely:

1) Synthesis: The first physical design step in the abstract design flow as shown in Fig.1 is the synthesis and technology mapping step [3] for HDL-based macros. The SoC design based on HDL-code is synthesized and eventually mapped on a target technology [4]. Therefore, the target technology directly contains the macro blocks which can be directly utilized for.

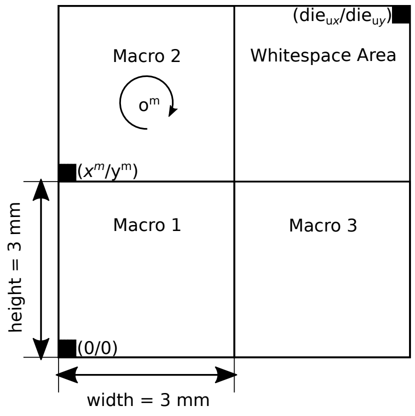

Fig. 2: Placement Problem.

2) Macro Placement: After the design has been mapped to the target technology, the macros have to be placed on the layout [5], [2], [6]. To this end, the placement underlies cost functions like minimizing the needed die area.

3) Routing: After the macros have been placed, the connections between them have to be established [7]. To this end, of course the respective placements have to be considered. At the same time, further objectives such as minimizing the wire-length used for routing are considered.

4) Sign-Off State: After placement and routing has been successfully performed, the defined constraints are finally checked again before the design can be exported and sent to the fabrication site [5].

In this work, we are focusing on macro placement. Here, a given set of macros shall be placed on a die, i.e., on a 2-dimensional area leading to a macro layout. Within the scope of this work, macros are defined as follows:

Definition 1. *Assume that all macros are stored in a set M. The shape of a rectilinear macro $m \in M$ which can be placed on a layout is defined by its width $w^m \in \mathbb{N}$ as well as its height $h^m \in \mathbb{N}$. The position of a macro within a layout is defined by the lower left corner $(x^m, y^m) \in \mathbb{N}^2$ in combination of its orientation $o^m \in \{N, W, S, E\}$. That is, the placement of the macro can be performed by setting proper values for the lower left corner (x^m, y^m) as well as the orientation o^m (c.f. Fig.2).*

The task of macro placement is to place all macros $m \in M$ on a given 2-dimensional area. This process shall result in a layout defined as follows:

Definition 2. *The* layout *of an SoC represents a placement of all macros $m \in M$ on a given area of the die. The available layout-area is a rectangular area spanned between two points. These points are defined as $(die_{lx}, die_{ly}) \in \mathbb{N}^2$ and $(die_{ux}, die_{uy}) \in \mathbb{N}^2$. Macros can be arbitrarily placed onto this area as long as they satisfy certain constraints such as that they do not overlap.*

As cost objective, several metrics such total area size, *Half-Perimeter Wirelength* (HPWL) [8], or others can be used. For sake of simplicity, we describe the approach proposed in this work considering area as cost metric (however, the proposed solution can be adjusted for other costs metrics/cost functions as well). This is straight-forwardly defined as follows:

Definition 3. *The* cost *of a layout is defined by the area $\in \mathbb{N}$ which is needed in order to place all macros. These costs can be easily determined by $area = die_{ux} \cdot die_{uy}$.*

Example 1. *The layout as shown in Fig. 2 consists of three macros. However, the placement is not optimal in terms of area since there is a significant percentage of whitespace (i.e. unused) die area (c.f. the upper right corner).*

B. State of the Art Macro Placement

Macro placement for SoC design has already been studied widely within the industrial as well as the academic research communities. These efforts led to multiple approaches including the application of meta-heuristic algorithms like *Simulated Annealing* (SA) [1], approaches based on *Machine Learning* (ML) [5], as well as *Genetic Algorithms* (GA) [2], or approaches based on reasoning engines [6], [9].

SA is utilized in academic macro placers like Parquet [10]. The underlying approach is inspired by the cooling process of hot metal. To this end, the mobility of the underlying atoms depends on the temperature of the metal. Transferring this approach into the domain of macro placement means, the flexibility of the macros (i.e., the flexibility to move a macro on the die) depends on the “temperature”. The hotter the temperature, the further the distances the macros can be moved. The movement (e.g., random switching macros within a certain range) of the macro should only be applied if the movement shifts the current solution closer to the global maxima of the cost function. Therefore, the number of moves in combination with the speed of the cooling process posses a significant impact on the success of the eventual placement.

Besides that, several approaches based on GAs have been proposed for the domain of macro placement. To this end, approaches like [11], [12] utilize randomness and heuristics. However, heuristic-based approaches are not capable to guarantee that the macro placement which is based on an optimal die area is found. Recently, also approaches based on ML have been published for macro placement [5]. Compared to approaches based on SA or GA, ML as applied in [5] is used in order to support the placement algorithm (e.g. as cost function). Therefore, neither SA nor GA are capable to guarantee that an optimal placement for the macros and, therefore, a solution minimizing a given cost function (e.g. area) is getting found.

The strategies of GAs as well as SA are based on identifying a global maxima for their underlying cost function. However, since both categories of algorithms are heuristic based it can not be guaranteed that the found maxima is a global maxima and not one of the local maxima. In order to overcome the uncertainties of heuristic-based approaches like GA or SA, we are now introducing our approach for macro placement based on the utilization of modern reasoning engines which allows it to generate optimized placement results for a given placement problem.

C. General Idea

In order to overcome the limitations of existing macro placement approaches and, in order to determine optimized solutions, all possibilities of placements have to be considered. In a naive fashion, this could be conducted by enumeratively generating and evaluating all possible macro placements. More precisely, for a given set of macros, all possible placements are iteratively considered. From all these placements, those are eventually discarded which do not realize the respectively given circuit and/or violate any constraints (e.g., macros which overlap). Afterwards, from the remaining placements, the placement is picked which comes up with the smallest costs (e.g. die area). An approach like that would eventually yield an optimal macro placement. However, a naive approach based on enumeration would not be capable of determining designs of appropriate size – the sheer number of possibilities would be too huge.

Hence, we are proposing to utilize the computational power of solving engines such as satisfiability solvers (see e.g., [13], [14], [15]) and optimizers (see e.g, [16], [17]) for the SMT. They are heavily optimized and additionally employ highly sophisticated deduction and learning schemes which allow them to automatically prune large parts of the search space without discarding any valid solution. In the past, this already have been proven to be very effective for many practically relevant problems such as model checking [18], stimuli generation [19], test pattern generation [20], and more [21].

The application of optimizing solvers allows to obtain an optimized macro placement. However, those approaches will be limited due to the size (i.e., the number of macros) of modern SoCs, i.e., the search space for the macro placement of modern SoCs is getting too big and would often not allow to obtain a result within a reasonable time. In order to overcome this limitation, we further propose another search space pruning by partitioning the macro placement problem into a number of sub-problems which can then be solved by the solving engine within a reasonable time. Although this will eventually lead to results for which the optimization strategy cannot be applied anymore, the respectively obtained results are expected to be close to the optimum.

III. Proposed SMT-based Macro Placement

The general idea proposed above motivates the utilization of modern reasoning engines for SoC macro placement. This section describes the details of the corresponding solution. We first introduce the symbolic formulation of the problem which represents all (valid) solutions of the problem and is to be processed by a solver. Afterwards, we describe how to formulate the corresponding optimization objective and the application of optimizing solvers to determine optimized results for the placement. Since solving the resulting instance is infeasible for larger instances (as discussed above), finally our ideas for search space pruning are discussed.

A. Problem Formulation

For the realization of the proposed approach, we first have to describe the macro placement problem in terms of a symbolic formulation representing all possible placements. To this end, we first introduce the symbolical notation of the grid onto which macros should be placed:

Definition 4. *The symbolic grid (i.e. layout) used for the macro placement is described by two coordinates within a 2-dimensional area (as depicted in Fig. 2). The variables* $(die_{ux}, die_{uy}) \in \mathbb{N}^2$ *denote the upper right corner of the grid. Additionally, a lower left corner would be needed to define a rectangular grid. Without loss of generality, however, we assume this coordinate to be* $(0, 0)$ *which does not require explicit variables for representation.*

Next, given a set M of macros, each macro $m \in M$ should be placed. To symbolically represent *all* possible placements for all macros, the following formulation is used:

Definition 5. *The symbolic representation for each macro* $m \in M$ *consists out of five variables. Therefore, the variables* $x^m, y^m \in \mathbb{N}$ *define the lower left corner of the macro (c.f. Fig. 2 [22].). Together with the width and height* $w^m, h^m, \in \mathbb{N}$ *and also the macro's orientation* $o^m \in \{N, W, S, E\}$*, the shape and location of the macro is defined symbolically. To this end, the reasoning engine can now assign values to these* free-variables *and place them on the layout. However, since the macro can change its orientation, we are going to abstract the coordinates (i.e. the shape) of the macro for later constraint encoding.*

In order to abstract the shape of macro from its orientation, we are introducing the auxiliary variables $l_x^m, l_y^m \in \mathbb{N}$ *to describe the lower left corner of the macro as well as* $u_x^m, u_y^m \in \mathbb{N}$ *to describe the upper right corner of the macro. The values of the four variables are directly depending on the macro's orientation and, therefore, we have to define the values for each particular case. Moreover, we are going to introduce these auxiliary variables in order to abstract the encoding from the orientation, which means, that the reasoning engine does only know the five* free-variables *as discussed above. To this end, Fig. 3 shows the four possible macro orientations including their corresponding reference point. For the calculation of these variables, we assume the* N*-orientation as the base case and calculate the other orientations back to this case. The four auxiliary variables which are used for the eventual constraint can than be directly calculated for each of the four cases (c.f. [22]).*

Example 2. *Consider again Fig. 2 which illustrates a possible solution of the macro placement problem. Here, the reasoning engine assigned the macro* m_2 *to be orientated N, and set the values for the lower left corner of the macro to:* $x^m = 0$ *and* $y^m = 3$*. Using the formulation introduced above, the shape of the placed macro can be deduced by calculating the coordinates:* $l_x^{m_2} = x$, $l_y^{m_2} = y$, $u_x^{m_2} = x+w$ *and* $u_y^{m_2} = y+h$.

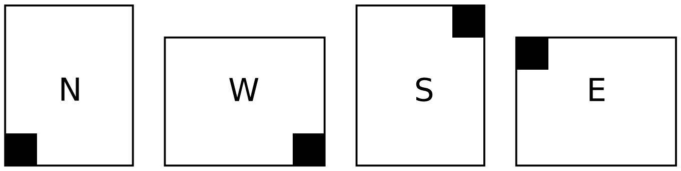

Fig. 3: Supported Orientation.

From this, it can obviously be seen, that m_2 *is placed between the coordinates* $(0, 3)$ *and* $(3, 6)$.

Passing this formulation to an SMT solver would lead to arbitrary placements of all the macros. To this end, the solving engine may not only place the macros arbitrarily, but also would allow for layouts of arbitrary size (while the objective is to keep the layout size as small as possible). In fact, since none of the introduced variables are restricted (all can be assigned to any value of $\mathbb{N}$), macros may overlap or even be placed outside of the given layout boundaries. To prevent that, constraints are added which enforce that only assignments representing *valid* macro placements are allowed. First, we have to ensure that all macros are placed within the given layout boundaries. This is accomplished through:

$$\bigwedge_{m \in M} \quad \bigwedge_{\substack{l_x^m \geq die_{lx} \\ l_y^m \geq die_{ly}}} \quad \bigwedge_{\substack{u_x^m \leq die_{ux} \\ u_y^m \leq die_{uy}}} \tag{1}$$

Next, all macros should be placed so that no macro overlaps with another. This is accomplished through:

$$\bigwedge_{\substack{m_1 \in M \\ m_2 \in M}} \quad \bigvee_{\substack{l_x^{m_1} \geq u_x^{m_2} \\ u_x^{m_1} \leq l_x^{m_2}}} \quad \bigvee_{\substack{l_y^{m_1} \geq u_y^{m_2} \\ u_y^{m_1} \leq u_y^{m_2}}} \tag{2}$$

Passing these formulations over to a solving engine now either gives an assignment of the variables defining a valid placement or proves that a valid placement cannot be realized.

B. Cost Optimization

The placement determined by the reasoning engine allows it so far to place non-overlapping macros on the layout. Reasoning engines like Z3 [13] or Boolector [14] can be utilized for this purpose. However, whether the solution determined by these SMT engines indeed is minimal with respect to the costs can not be deduced from the found solution. In order to overcome this problem, we utilize solving engines for the *Optimization Module Theories* (OMT) problem like vZ [17] or OptiMathSAT [16]. Compared to SMT, OMT commands over an extended set of instructions and introduces the possibility to minimize/maximize certain variables of an existing encoding.

In order to invoke an OMT solving engine, we have to formulate a cost function to be minimized/maximized. Since we aim for minimize the total area of the layout, a cost function based on the product of die_{ux} and die_{uy} would be suitable for our approach. However, solving engines like vZ do not support non-linear cost functions for the optimization. Hence, adding the following optimization target to the proposed problem formulation ensures that the reasoning engine determines an optimized macro placement result:

Definition 6. *The cost function is based on the minimization of the two coordinates* die_{ux} *and* die_{uy}*. By utilizing an OMT solving engine, we can add these two optimization targets to the encoding in order to obtain an optimized macro placement.*

$$\underset{die_{ux}, die_{uy} \in \mathbb{N}}{\text{minimize}} \quad die_{ux}, die_{uy} \tag{3}$$

Modern OMT solvers like vZ [17] support different optimization strategies. To this end, a box optimizer could be utilized for single object optimizations. Therefore, we are going to invoke vZ's Pareto optimizer which is capable to handle multi-object optimization. In case of the Pareto optimization, the solver generates a number of optimized solutions. In order

TABLE I: Results comparing our approach based on the MCNC benchmark suite [23].

			SMT (ISED 2016 [6])		Parquet Placer [10]		*Proposed*		
Circuit	Size	Min. Area	Area	Whitespace	Area	Whitespace	Area	Whitespace	Partitioning
apte	9	46.56	48.05	3.10	46.94	0.79	46.92	0.77	
hp	11	8.83	9.26	4.64	9.66	8.59	8.94	1.3	
xerox	9	19.35	20.01	3.30	19.91	3.21	19.8	2.25	
ami33	33	1.16	1.28	9.38	1.23	6.03	1.23	5.69	✓
ami49	49	35.45	40.04	11.46	37.76	3.59	38.76	8.54	✓

Note that the units for area are mm^2 and that whitespace is given percentage compared to the minimum area.

to obtain the best possible placement result we search the set of optimized results for the best solution. Therefore we are using the product of the generated solution for die_{ux} and die_{uy} to classify the best result. The optimized placement result therefore depends on the underlying reasoning engine.

C. Search Space Pruning: Partitioning

The problem encoding as introduced above is capable to find optimized macro placements. However, due to the size of modern SoCs, this approach ends up with scalability issues due to the exponential growth of the search space. In order to overcome these scalability issues, we use a further approach for search space pruning, namely partitioning. Search space pruning tries to break down the placement problem into smaller partitions, which contain macros themselves and can be solved in a reasonable time utilizing the reasoning engine. Eventually, the partitions are used for final placement instead of the macros directly. Therefore, a partition is defined as follows.

Definition 7. *A partition $p \in P$ commands over a subset of macros $M_i \subset M$. By utilizing partitioning, all macros have to be part of any partition $\cup_{M_i} = M$. In order to apply the encoding as shown above, every partition has to command over the same variables as a macro. These variables, namely $x^{M_i}, y^{M_i} \in \mathbb{N}$ which denotes the lower left corner of the partition, the partition's width and height $w^{M_i}, h^{M_i} \in \mathbb{N}$ and it's orientation $o^{M_i} \in \{N, W, S, E\}$.*

The approach of partitioning is capable to break down the macro placement problem into sub-problems. However, identifying the macros which should be clustered together as a partition still remains an open issue. The partition should consist of a significant number of macros in order to break down the eventual partition placement on the die as much as possible. At the same time, the partitions, same as the eventual layout on the die, should be realized with minimal costs (i.e. the partitions should be as small as possible) while still all constraints (e.g. non overlapping) are getting fulfilled.

In order to identify the macros to be clustered within a partition, we are going to utilize the similarity of the macros shapes (i.e. the width and the height). Moreover, we are going to utilize *K-means clustering* for this application [24]. Thereby the algorithm tries to identify K cluster in an unknown set of data. The algorithm starts by picking a random start point and iteratively approximates the optimal clusters for the given number of clusters (i.e. K cluster).

For the creation of a partition, the partitions size has to be determined first. For this, the partition is treated like a layout in order to find the values for its width and height. To this end, the encoding as introduced for finding the global macro placement is getting applied for partitioning as well.

IV. Experimental Evaluation

In order to evaluate the performance of the proposed approach, we implemented the solution as described in Section III in form of an open source tool called *SMT_MacroPlacer* [1]. In this section, we summarize the obtained results based on benchmarks which are widely used for macro placement. To this end, we first describe the setup as well as the considered test cases. Afterwards, results are provided and discussed.

[1]The tool is available at http://github.com/gledr/SMT_MacroPlacer.

A. Setup and Benchmarks

For our experiments, we considered the MCNC benchmark suite. We have decided to work based on these benchmarks in order to allow a comparison of our approach with other existing approaches, namely [6], [10]. Based on these benchmarks, we applied our approach as realized by the tool in order to determine a macro placement. Afterwards, we compared our results with the results as stated in [6] which is to the best of our knowledge the only other macro placement approach based on reasoning engines and the results of the local execution of the Parquet placer proposed as in [10].

B. Results and Discussion

The results of the conducted experiments are shown in Table I. The table compares the performance of two existing approaches for macro placement, namely [6], [10], with the method proposed in this work. More precisely, the first columns provide the name of the benchmark circuit, the number of macros utilized by the circuits, and the minimal possible area needed for a legal placement (i.e. the sum of the area of all macros $m \in M$). Afterwards, results of the considered macro placement methods are reported, including the needed die area (placement result), the percentage of whitespace (i.e. unused area) compared to the theoretical minimum. If the problem size was still feasible without applying search space pruning, the "pure" SMT-based approach has been applied. To this end, we applied a timeout of 3000 CPU seconds. If it was possible to determine a solution within that time, no partitioning had been utilized. Otherwise, search space pruning as described in Section III-C was additionally invoked.

Table I shows that the results obtained by the proposed approach for small circuits (c.f. apte, hp and xerox) outperforms the results shown by both other approaches. Moreover, for the first three benchmarks, no partitioning had been utilized and the reasoning engine could command over the entire search space which ended up in an optimized macro placement. For the other two benchmarks (ami33, ami49), the obtained results are non optimal since partitioning had to be utilized. Even though the obtained results could not approximate an optimal placement, we could still outperform the results generated by [6]. Overall, this demonstrates the applicability of modern reasoning engines for the application of macro placement.

V. Conclusion

This paper demonstrated the application of satisfiability solving engines for the macro placement step during the physical design of modern SoCs. While other approaches for macro placement are e.g. based on meta-heuristic algorithms which are not capable to guarantee an optimal solution, our approach utilizes the power of modern optimization engines to overcome this existing limitation. To this end, we proposed a symbolic formulation of the constraints as well as the symbolic formulation of the optimization target in order to approximate an optimal placement for given benchmark circuits. In case the complexity of an instance can still not been tackled by this, also proposed a method for state space pruning which does not guarantee optimal results anymore, but still allows to determine reasonable layouts. To this end, a partitioning strategy is employed. We demonstrate the applicability of the proposed method based on a set of benchmarks for macro placement and made the corresponding implementation available online in terms of an open-source tool.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria, the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems, as well as the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] S. A. Kravitz and R. A. Rutenbar, "Placement by simulated annealing on a multiprocessor," *IEEE Trans. on CAD of ICs and Systems*, 1987.

[2] A. Kaur and S. S. Gill, "Hybrid swarm intelligence for vlsi floorplan," in *Int'l Conf. on Computing, Communication and Automation*, Noida, India, 2016.

[3] W. Clifford and J. Glaser, "Yosys - a free verilog synthesis suite," in *2018 Austrochip Workshop on Microelectronics (Austrochip)*, Linz, Austria, 2013.

[4] X. Xu, N. Shah, A. Evans, S. Sinha, B. Cline, and G. Yeric, "Standard cell library design and optimization methodology for asap7 pdk: (invited paper)," in *Int'l Conf. on CAD*, Irvine, USA, 2017.

[5] Y. Huang, Z. Xie, G. Fang, T. Yu, H. Ren, S. Fang, Y. Chen, and J. Hu, "Routability-driven macro placement with embedded cnn-based prediction model," in *Design, Automation and Test in Europe*, Florence, Italy, 2019.

[6] S. Banerjee, A. Ratna, and S. Roy, "Satisfiability modulo theory based methodology for Floorplanning in VLSI circuits," in *International Symposium on Embedded Computing and System Design*, Patna, India, 2016.

[7] A. B. Kahng, L. Wang, and B. Xu, "Tritonroute: An initial detailed router for advanced vlsi technologies," in *Int'l Conf. on CAD*, San Diego, USA, 2018.

[8] C. Chu, "Flute: fast lookup table based wirelength estimation technique," in *Int'l Conf. on CAD*, San Jose, USA, 2004.

[9] A. Grimmer, Q. Wang, H. Yao, T. Ho, and R. Wille, "Close-to-optimal placement and routing for continuous-flow microfluidic biochips," in *ASP Design Automation Conf.*, Chiba, Japan, 2017.

[10] S. N. Adya and I. L. Markov, "Fixed-outline Floorplanning : Enabling Hierarchical Design," *IEEE Trans. on VLSI Systems*, 2003.

[11] M. Tang and X. Yao, "A memetic algorithm for vlsi floorplanning," *IEEE Trans. on Systems, Man, and Cybernetics*, 2007.

[12] P. Fernando and S. Katkoori, "An elitist non-dominated sorting based genetic algorithm for simultaneous area and wirelength minimization in vlsi floorplanning," in *VLSI Design*, Hyderabad, India, 2008.

[13] L. De Moura and N. Bjørner, "Z3: An efficient SMT Solver," *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[14] A. Niemetz, M. Preiner, and A. Biere, "Boolector 2.0 system description," *Journal on Satisfiability, Boolean Modeling and Computation*, 2014.

[15] R. Wille, G. Fey, D. Große, S. Eggersglüß, and R. Drechsler, "SWORD: A SAT like prover using word level information," in *VLSI-SoC: Advanced Topics on Systems on a Chip: A Selection of Extended Versions of the Best Papers of the Fourteenth International Conference on Very Large Scale Integration of System on Chip*, 2009.

[16] R. Sebastiani and P. Trentin, "OptiMathSAT: A Tool for Optimization Modulo Theories," *Journal of Automated Reasoning*, 2018.

[17] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νz - an optimizing smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, Heidelberg, Germany, 2015.

[18] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 1579. Springer Verlag, 1999, pp. 193–207.

[19] R. Wille, D. Große, F. Haedicke, and R. Drechsler, "SMT-based stimuli generation in the SystemC verification library," in *Forum on Specification and Design Languages*, 2009, pp. 1–6.

[20] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved sat-based ATPG: more constraints, better compaction," in *Int'l Conf. on CAD*, 2013.

[21] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*. IOS Press, 2009.

[22] "LEF/DEF Language Reference," Cadence Design Systems, Inc., Tech. Rep. Version 5.7, 2009.

[23] C. J. Alpert, "The ISPD98 Circuit Benchmark Suite," in *Int'l Symp. on Physical Design*, New York, USA, 1998.

[24] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society.*, 1979.