A Comparison of Algorithms for the Generation of Layouts based on Reconfigurable Slots on FPGAs

Department of Computer Science
Paderborn University

January 5, 2023

Supervisors: Prof. Dr. Marco Platzner
Student: Yashwanth Tadakamalla (6896890)

1 Background

Executing real-time tasks on dynamically reconfigurable FPGAs requires one to solve the challenges of scheduling and placement. In the past, many approaches have been presented to address these challenges, but most of them rely on idealized assumptions about the reconfigurability of FPGAs and the capabilities of commercial tool flows. In [4], an approach was presented that aims at solving these problems leveraging a practically useful 2D slot-based FPGA area model. Under this model, hardware tasks are grouped together and placed into reconfigurable slots on the FPGA. A reconfigurable slot consists of a set of so-called micro slots, which are the smallest units of FPGA resource management. On one hand, this model enables the use of previous real-time scheduling techniques and lends itself to a practical realization. On the other hand, the model also poses new problems of reconfigurable slot creation and layout generation. For layout generation, heuristic [4] and optimal [5] solutions have been presented. The heuristic is a rather simple, rule-based construction technique for the layout, that is very efficient in terms of runtime. The optimal technique is based on formulating and solving a quadratic constraint program (QCP), that is feasible only for small sets of reconfigurable slots.

2 Problem Definition

The objective of this master project is to study, adapt, develop and evaluate algorithms for the problem of layout generation in the context of executing real-time tasks on FPGAs [4][5]. Based on a literature survey and study of related packing problems, algorithms for the layout generation problem should be proposed. A few algorithms should be selected and implemented. Good candidates are probably Simulated Annealing and Genetic Algorithms. For the implementation, existing code for heuristic and optimal solutions can serve as starting point. Based on the implementations and test data sets, the algorithms should be compared. Metrics of interest are the runtime and the quality of the result, i.e., percentage of mappable slot sets or FPGA utilization. The algoritms should be integrated in a software tool. The tool must be able to receive inputs via appropriate command line

parameters to be run in batch mode. A graphical user iterface, possibly with a visulization of the results and intermediate steps is an optional task.

3 Literature Review

One of the crucial phases in the physical design of VLSI (Very-large-scaled-Integration) is floorplanning. This floorplan can be considered as NP-hard ((Non-Deterministic Polynomial-time Hard). Floorplanning can be classified into two categories that are 1) Slicing structure and 2) non-slicing structure. The floorplan can be repeatedly chopped horizontally or vertically to create a slicing floorplan, which is impossible in non-slicing structures. The technique of putting circuit modules of random sizes and dimensions on a predetermined layout area with the aim of minimizing the area and wire length between the modules is known as floor planning in the context of circuit design. There has been a lot of research going on in this phase. Researchers have used iterative methods like Simulated Annealing, Genetic Algorithms, Particle Swarm Optimisation, and also hybrid methods to solve this problem

In [1] the author has studied two types of modern floor planning problems which are 1) fixed outline floor planning and 2) bus-driven floor planning. In order to address the multi-objective VLSI floor planning problem, this floor planner used a B*-tree floor plan representation based on the fast three-stage simulated annealing (SA) scheme known as Fast-SA.

In [6] in order to express the topology of non-slicing floorplans, the author uses the sequence-pair proposed by Murata et al and then presented two methods to obtain floor planning from sequence pair. One Is a construction method and the other one is based on a linear programming model. The author have used the Simulated Annealing technique to find the near-optimal floorplanning solutions

A memetic algorithm (MA) was put forth by Maolin Tang et al [9]] for a hard-module, non-slicing VLSI floor planning problem. This MA is a hybrid genetic algorithm that explores the search space effectively through genetic search and effectively accesses information in the search region through local search.

In [14] the cause of dead space is examined by the author first, and then a simple and quick method is suggested to choose the proper orientation for each module. The best floorplan is then found using the simulated annealing process after a perturbing operator of the normalized polish expression is adjusted to obtain the neighborhood solution. Experiment findings have shown that these techniques can promisingly deliver a solution by minimizing area utilization.

In [11] the author has used a genetic algorithm to for local search on each iteration. For the arrangement of rectangle modules, a non-slicing B* tree is used. Ordered binary trees are the foundation of B*-trees. The implementation of B*-trees is fairly simple, and they can execute each of the elementary tree operations—such as search, insertion, and deletion—in only O(1), O(1), and O(n) times, respectively. This GA has been used and tested on well-known benchmark issues. According to experimental findings, GA can swiftly generate ideal solutions for every benchmark problem that has been tried.

In [2] Two algorithms for the floorplan design issue are presented in this work. In essence,

the algorithms are pretty similar. They both use the simulated annealing search method and Polish expressions to depict floorplans. The first approach is used when all modules are rectangular, and the second algorithm is used when either rectangular or L-shaped modules are present.
The author Wong and Liu [13] has used a normalized polish expression to depict the floorplan and also made use of Simulated Annealing to find out the optimal floorplan. Based on this algorithm [3] have extended this algorithm to the algorithm to deal with situations when some blocks need to be positioned inside or close to certain areas of a chip. And also to handle the non-slicing floorplans there is so much research that used sequence pair to represent the floorplanning topology
In [8] authors suggest using genetic algorithms to floorplanning in order to give novice designers better initial circumstances as a place to start their design work. and In [10] author suggests a GA that adopts the sequence-pair representation as the coding scheme for each chromosome to tackle the floorplanning problem in VLSI layout design. To effectively explore the search area, new genetic operators for the issue are proposed. The suggested GA has an adaptive strategy that, based on the state of an individual, dynamically chooses the best genetic operator during GA execution.
In [7] for non-slicing VLSI floorplanning, authors have described a hybrid simulated annealing technique (HSA) in this study. The HSA employs a novel bias search technique to balance global exploration with local exploitation, a new greedy method to build an initial B*-tree, a new operation on the B*-tree to explore the search space.

4 Main Subtasks

- Familiarize yourself with existing literature on packing problems, in particular in electronic design automation, e.g., [12] Chapter 10, and the concrete layout generation problem [4][5].
- Analyze and select a few algorithms for the layout generation problem.
- Create appropriate test data sets and define relevant metrics for evaluation.
- Implementation of layout generation code using Simulated Annealing algorithm in python.
- Implementation of layout generation code using Genetic algorithm in python.
- Run experiments with existing and newly implemented algorithms for layout generation.
- Compare the algorithms using selected metrics that are runtime and the total number of tasks allocated.
- (Optional:) Develop a graphical user interface for the developed tool.

5 Planned Schedule

The following chart shows the tentative work plan that we intend to follow through this thesis work.

Task Description	December		January				February				March				April				May	
	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
Literature review																				
Analyse and Select Algorithms																				
Create test data																				
Implement Simulated Annealing (in python)																				
Implement Genetic Algorithm (in python)																				
Experiments with existing and newly implemented algorithms (with selected metrics)																				
(Optional:) Graphical User Interface																				
Documentation and Report																				

6 Organizational Matters

- **Schedule:** Before registering your thesis project, perform a project planning, i.e., extend this task description and add a time schedule. Discuss this with your advisor. Keep a continuous record of the progress of your work.
- **Regular meetings:** To keep track of your progress and to discuss difficulties or next steps, you should meet regularly (i.e., every two weeks) with your advisor.
- **Introductory presentation:** After you have familiarized yourself with the project and its objectives, you are expected to give a short (10–15 minutes) introductory presentation of your task, the proposed solution and approaches, and your time schedule.
- **Written report:** At the end of your work, you need to submit a written thesis. As a rule of thumb, the thesis should be between 80 and 120 pages long. Before writing the thesis you should draft an outline and discuss it with your supervisor.

Furthermore, it is highly recommended to discuss the thesis with your supervisor before submitting, in order to receive feedback and suggestions for improvement.

- **Final presentation:** After submitting the written thesis, you are expected to give a final presentation covering the task and the objectives, the taken approaches and the achieved results.

References

[1] Chang Chen, T.C. Modern floor planning based on b*-tree and fast simulated annealing. 2006.

[2] C. L. Liu D. F. Wong. Floorplan design of vlsi circuits.

[3] D. F. Wong F. Y. Young and H. H. Yang. Slicing floorplans with boundary constraints. 1999.

[4] Zakarya Guettatfi, Marco Platzner, Omar Kermia, and Abdelhakim Khouas. An approach for mapping periodic real-time tasks to reconfigurable hardware. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2019.

[5] Zakarya Guetttatfi, Paul Kaufmann, and Marco Platzner. Optimal and greedy heuristic approaches for scheduling and mapping of hardware tasks to reconfigurable computing devices. In *Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC)*, 2020.

[6] Yeong-Dae Kim Jae-Gon Kim. A linear programming-based algorithm for floorplanning in vlsi design.

[7] Wenxing Zhu Jianli Chen and M. M. Ali. A hybrid simulated annealing algorithm for nonslicing vlsi floorplanning. 2011.

[8] Satoshi Yamane Kenji Oshima Kazuhiko Eguchi, Junya Suzuki. An application of genetic algorithms to floorplanning of vlsi.

[9] Xin Yao Maolin Tang. A memetic algorithm for vlsi floor planning.

[10] Shin'ichi WAKABAYASHJ Shingo NAKAYA, Tetsushi KOIDEZ. An adaptive genetic algorithm for vlsi floorplanning based on sequence-pair. 2000.

[11] M. Dec T. Singhaa, H. S. Duttab. Optimization of floor-planning using genetic algorithm. 2012.

[12] Laung-Terng Wang Wanf, Yao-Wen Chang Chang, and Kwang-Ting Cheng. *Electronic Design Automation*. Morgan Kaufmann, 2009.

[13] D. F. Wong and C. L. Liu. new algorithm for floorplan design. 1986.

[14] Chen Meixue Jia Xiaomin Li Xuanxiang Du Shimin Zhu Lichen, Yang Runping. An efficient simulated annealing based vlsi floorplanning algorithm for slicing structure. 2012.