Technical Paper Proposal:
Placement Algorithms for Heterogenous FPGAs

Brian B Cheng
Department of Electrical and Computer Engineering

1 Keywords

- FPGA, EDA, Synthesis, Placement, Routing, Parallel, Optimization

2 Proposal

A compiler takes a program written in high-level programming language like a Java or Python and assembles low-level machine code which is ready to be executed on a CPU. In similar fashion, an Electronic Design Automation tool (EDA) takes a high-level description of a digital system written in Verilog or VHDL and produces a bitstream which is ready to be deployed to an FPGA. In a superficial way, compilers and EDAs perform the same task for engineers but in different industries. One for software, the other for hardware.

Software compilers have evolved to become highly optimized, significantly boosting the productivity of software developers. The compilers are so robust that developers rarely need to worry about the correctness or efficiency of their machine code, and are so refined that it all happens in a matter of milliseconds to minutes, even for large projects. The speed of compilation enables rapid debugging and implementation of new features and allows developers to iterate through dozens or even hundreds of design cycles per day. Instead of tinkering with assembly code, software engineers can comfortably focus on the high-level abstractions of their design.

The compilation stages mainly consists of lexing, parsing, and optimization. The optimization stage is where most of the complexity occurs as it can deal with NP-complete problems, depending on what optimization flags the developer set when running compile. The compiler at this stage must optimize loop unrolling and code pruning as well as the ordering of the machine instructions to maximize pipelining and register allocation. While the search space for the best possible machine code is large, it is typically constrained by the size of the CPU instruction set and CPU architecture which are fixed. Furthermore, compiler makers have developed highly efficient heuristics to approximate the best machine code within an acceptable runtime.

On the other hand, the EDA stages consist mainly of synthesis, placement, and routing. All three are NP-hard optimization problems with massive search spaces. Even with heuristics and approximations to circumvent NP-hardness, the size of their search spaces make the EDA runtimes less than desirable.

In a typical FPGA design flow, the engineer creates and describes a digital design using a hardware description language (HDL) like Verilog or VHDL. Then, the FPGA engineer verifies the design by creating a testbench that wraps around the design and feeds the design input signals and observes the output signals. Then the engineer submits the design entry to an EDA like Vivado or Quartus to perform the three stages of automated design. For even modest designs that utilize a small percentage of the available resources on the FPGA, the EDA will spend a minimum two to three minutes running synthesis and implementation. The EDA run time increases exponentially with the scale of the project. Projects that utilize 80% or more of the FPGA's resources may run for hours, and for the high-end devices like Xilinx's Kintex FPGAs which have millions of logical elements, can run for days. At high utilization, there is a possibility that the EDA cannot fit the design onto the device due to routing congestion even when it has enough logical elements to implement the design. The EDA might only reach this conclusion after attempting to fit the design for several hours.

One of the most common complaints from new FPGA engineers, especially those coming from software development, is that the FPGA toolchains are slow and heavy. This is because the problems that the EDA must solve are inherently complex. Unlike in software development where an error in the program can be patched and recompiled in a matter of minutes, EDA runtimes lasting several hours means that an engineer can only go through a couple design cycles per day. An FPGA engineer must be very precise with their coding and practice thorough verification of their design before submitting it to the EDA. This overall heightens the barrier of entry into FPGA development and contributes to a shortage of qualified FPGA engineers and a limitation of their productivity.

This paper aims to study one particular pillar of this barrier: the placement stage. First, it will review prerequisite knowledge for placement strategies. This includes graph theory to represent logical elements of a digital design and the wired connections between them, and a corresponding optimization problem with a cost function which we seek to minimize.

Second, it will briefly explore the historical progress made on placement algorithms for both VLSI and FPGA [1] and then explore the current trends in open-source placement research [2], [3], [4], [5], [6], [7]. I would have liked to explore the current trends in proprietary EDAs but they are unfortunately closed source.

And third, the paper will outline my own technical work on placement strategies, whether they be reproductions of existing strategies, or a novel strategy if I manage to create one in the span of 3 months. This will include performance analysis and comparisons among different placement algorithms and strategies. To achieve this, I will use Xilinx's open-source API, RapidWright which provides backend access to Vivado which is Xilinx's EDA for FPGAs. I have already familiarized myself with the API and and able to use the Vivado + RapidWright toolchain to do basic tasks like extracting the netlist of a synthesized VHDL design or performing manual placement of logic elements. You can see my preliminary work in my Github: `https://github.com/TotoroTron/place-and-route`.

3 Foundational Knowledge (so far)

A graph is modeled as $G = (V, E)$, where V represents the set of vertices and E represents the set of edges between them. A hypergraph is denoted as $G_H = (V_H, E_H)$, where V_H represents the set of vertices, and E_H represents the set of hyperedges, which are edges that can connect more than two vertices. Electronic circuits are typically represented as hypergraphs because a voltage source pin can have one or many sink pins. In FPGA and VLSI design, the hypergraph is transformed into something called a "netlist", which is a hypergraph that has been flattened down into a graph. For every hyperedge in the hypergraph, transform the hyperedge into a set of 2-pin edges, each having one source to one sink. The resulting graph can be represented by $G = (V, E)$, where V represensts the ports (pins) of all modules (logic gates,) There are different strategies for this reduction - unoriented star model, clique model, etc.. [8]

The vast majority of existing placers use the Half Perimeter Wire Length (HPWL) cost function in one of its various flavors - manhattan distance, euclidian distance, bounding box, etc.. The manhattan and euclidian HPWL models the cost function as the summation of the lengths of all nets in the netlist. The manhattan HPWL is convex while the euclidian HPWL is strictly convex. We can use smooth methods to minimize the euclidian, however, it produces worse Quality of Result (QoR) than the manhattan [9]. The bounding box HPWL models the cost function as simply the longest net in the netlist. Shown below in (1) and (2) are the manhattan and euclidian distance HPWL objective functions respectively. Shown in (3) is the bounding box version.

$$\Phi(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j} \left(|x_i - x_j| + |y_i - y_j| \right) \tag{1}$$

$$\Phi(\vec{x}, \vec{y}) = \sum_{i,j} w_{i,j} \left[(x_i - x_j)^2 + (y_i - y_j)^2 \right] \tag{2}$$

$$\Phi(\vec{x}, \vec{y}) = \sum_{i=1}^{n} \left(\max_{j \in e_i}(x_j) - \min_{j \in e_i}(x_j) + \max_{j \in e_i}(y_j) - \min_{j \in e_i}(y_j) \right) \tag{3}$$

$x = \{x_1, x_2, ..., x_N\}$ represents the set of physical x-coordinates of the logic block pins on the chip and $y = \{y_1, y_2, ..., y_N\}$ represents the set of y-coordinates. The units for these coordinates can be nanometers in VLSI design, but for FPGAs where the logic blocks are uniform size and arranged in a grid, the coordinates can simply be positive integers. N is the total number of logic block pins in the design. $E = \{e_1, e_2, ..., e_M\}$ represents the netlist with M total nets.

We want to minimize the total wirelength in our design for three reasons: power, clock skew, and routing. FPGAs are commonly used for mission critical applications and edge computing devices that are required to run on low power. Minimizing the wirelength, or more accurately, minimizing the signal path length can help save power because wires have a non-zero resisitance. This is particularly important for FPGAs because signal paths must propagate not only through wires but also switchboxes and programmable interconnects which can also dissipate power.

The second and more important reason is to minimize clock skew. Signal propagation through wires and transistors takes time. The clock signal is crucial for state transition in a finite state machine (FSM). If a clock signal drives two logic blocks that were placed too far apart, they may become desynchronized as the same clock edge may reach the two flip-flops at different clock periods. All flip-flops operating on the same clock domain must strike at the same nanosecond.

The third, equally important reason is to simplify the routing stage that follows placement. If logic blocks are placed close together in a way that creates chaotic, spaghetti-like wiring between them, certain routing points may become over-congested and risk failure. To avoid this, the placer should group related logic blocks into semi-isolated neighborhoods, minimizing inter-neighborhood connections and ensuring more efficient and manageable routing.

4 Abbreviations

- **FPGA**: Field Programmable Gate Array
- **VLSI**: Very Large Scale Integration
- **EDA**: Electronic Design Automation
- **VHSIC**: Very High Speed Integrated Circuits
- **HDL**: Hardware Description Language
- **VHDL**: VHSIC HDL
- **HLS**: High Level Synthesis: Generating synthesizable HDL from high-level software languages. A company might want to have a software engineer write C or C++ code and have a program translate it into synthesizable Verilog. This can can boost productivity and save the company the need to hire a hardware engineer.
- **IP**: Intellectual Property: In FPGA context, this means pre-built modules or subsystems like a hardened microprocessor or Ethernet controller. These are usually proprietary.

  - **SoC**: System on Chip: An FPGA device (chip) that features hardened IP in addition to the programmable logic fabric.
  - **PL-PS**: Programmable Logic - Processing System: A design that utilizes the on-chip hard microprocessor in conjunction with the programmable logic fabric.
  - **EDIF**: Electronic Design Interchange Format
  - **HPWL**: Half Perimeter Wire Length

5 Ideas

- **FPGA**: Field Programmable Gate Array
  - FPGA Vendors:
    - AMD-Xilinx (∼50% FPGA vendor market share)
    - Intel-Altera (∼35% share)
    - Lattice
    - Microsemi
- **EDA**: Electronic Design Automation
  - Proprietary software for FPGA and VLSI development:
    - Xilinx - Vivado (Design + Simulation) + Vitis (HLS + PL-PS codesign)
    - Altera - Quartus (Design) + ModelSim (Simulation)
    - Synopsis (VLSI)
    - Cadence (VLSI)
  - Open source software for FPGA development:
    - **VTR**: Simulated Annealing placer for FPGAs. Popular among researchers who study placement techniques. Commonly referred to as an ”academic placer”.
    - **OSS-CAD**: a full-flow software suite that includes ABC synthesis, Yosys synthesis, Yosys nextpnr.
    - **AMF-Placer**: Analytical Placer for FPGAs
    - **RapidWright**: Semi-open source Java API that provides backend access to Xilinx Vivado EDA using design checkpoints.
    - **RapidLayout**: Hard Block Placer for Systolic Arrays. Built with RapidWright.
    - **RapidStream**: HLS Placer. Built with RapidWright.
    - **DREAMPlace**: GPU-powered deep learning placement for VLSI.
    - **DREAMPlaceFPGA**: DREAMPlace, adapted to FPGAs via the RapidWright API.

- **Synthesis**
  - Takes a design written in a high-level HDL like VHDL or Verilog and ”synthesizes” a **logical netlist** out of it.
  - The logical netlist is usually generated as an EDIF, JSON, or a low-level Verilog file.
  - The netlist describes the necessary basic elements of logic (BELs) and the wired connections between them that are necessary to implement the design.
- **Placement**
  - Takes the **logical netlist** and produces a **physical netlist**.
  - For each BEL in the netlist, assign the BEL to a Cell, Site, and Tile on the physical FPGA device.
- **Routing**
  - Takes the **physical netlist** and maps the connections between BELs onto wires, interconnects, and switchboxes on the FPGA.

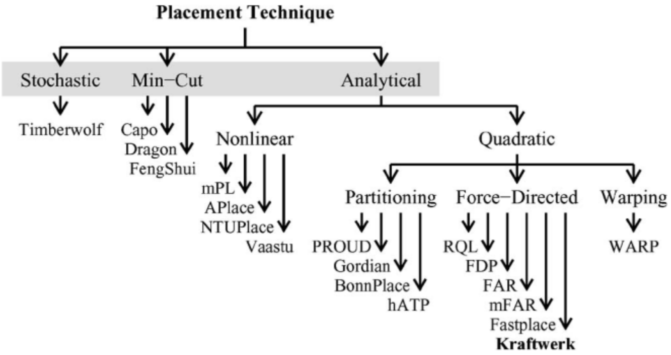

Figure 1: Landscape of VLSI placement techniques (Spindler) [10]

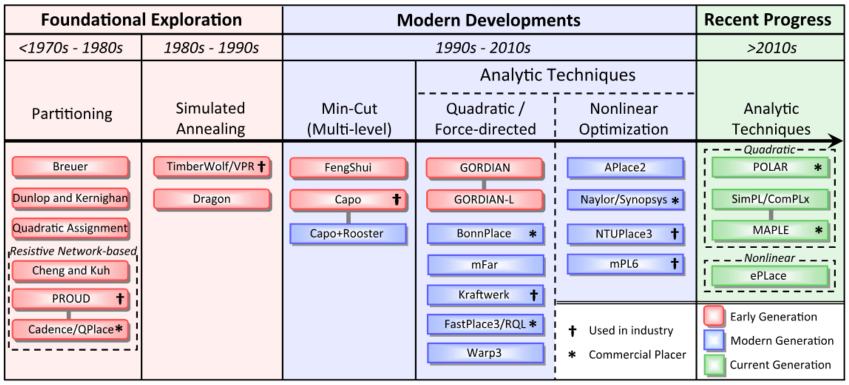

Figure 2: Historical timeline of VLSI placement techniques (Markov) [1]

This is a citation for AMFPlacer. [11]

References

[1] I. L. Markov, J. Hu, and M.-C. Kim, "Progress and challenges in vlsi placement research," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 1985–2003, 2015.

[2] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: An open source framework from verilog to bitstream for commercial fpgas," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 1–4, 2019.

[3] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "Amf-placer 2.0: Open source timing-driven analytical mixed-size placer for large-scale heterogeneous fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[4] N. Zhang, X. Chen, and N. Kapre, "Rapidlayout: Fast hard block placement of fpga-optimized systolic arrays using evolutionary algorithms," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 145–152, 2020.

[5] L. Guo, P. Maidee, Y. Zhou, C. Lavin, J. Wang, Y. Chi, W. Qiao, A. Kaviani, Z. Zhang, and J. Cong, "Rapidstream: Parallel physical implementation of fpga hls designs," in *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '22, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2022.

[6] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Z. Pan, "Dreampiace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2019.

[7] R. S. Rajarathnam, M. B. Alawieh, Z. Jiang, M. Iyer, and D. Z. Pan, "Dreamplacefpga: An open-source analytical placer for large scale heterogeneous fpgas using deep-learning toolkit," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 300–306, 2022.

[8] A. A. Kennings and I. L. Markov, "Analytical minimization of half-perimeter wirelength," in *Proceedings of the 2000 Asia and South Pacific Design Automation Conference*, ASP-DAC '00, (New York, NY, USA), p. 179–184, Association for Computing Machinery, 2000.

[9] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous fpgas," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 143–150, 2012.

[10] P. Spindler, U. Schlichtmann, and F. M. Johannes, "Kraftwerk2—a fast force-directed quadratic placement approach using an accurate net model," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 27, p. 1398–1411, aug 2008.

[11] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "Amf-placer: High-performance analytical mixed-size placer for fpga," in *2021 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–6, 2021.

[12] A. Mishra, N. Rao, G. Gore, and X. Tang, "Architectural exploration of heterogeneous fpgas for performance enhancement of ml benchmarks," in *2023 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 232–235, 2023.

[13] U. Farooq, H. Parvez, H. Mehrez, and Z. Marrakchi, "Exploration of heterogeneous fpga architectures," *Int. J. Reconfig. Comput.*, vol. 2011, jan 2011.

[14] C. Lavin, M. Padilla, S. Ghosh, B. Nelson, B. Hutchings, and M. Wirthlin, "Using hard macros to reduce fpga compilation time," in *2010 International Conference on Field Programmable Logic and Applications*, pp. 438–441, 2010.

[15] C. Lavin and A. Kaviani, "Rapidwright: Enabling custom crafted implementations for fpgas," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 133–140, 2018.

[16] Y. Zhou, P. Maidee, C. Lavin, A. Kaviani, and D. Stroobandt, "Rwroute: An open-source timing-driven router for commercial fpgas," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, nov 2021.

[17] C. Lavin and E. Hung, "Invited paper: Rapidwright: Unleashing the full power of fpga technology with domain-specific tooling," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–7, 2023.

[18] S. Areibi, G. Grewal, D. Banerji, and P. Du, "Hierarchical fpga placement," *Canadian Journal of Electrical and Computer Engineering*, vol. 32, no. 1, pp. 53–64, 2007.

[19] J. Chen, W. Zhu, J. Yu, L. He, and Y.-W. Chang, "Analytical placement with 3d poisson's equation and admm based optimization for large-scale 2.5d heterogeneous fpgas," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019.

[20] S. Dhar, L. Singhal, M. Iyer, and D. Pan, "Fpga accelerated fpga placement," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 404–410, 2019.

[21] H. Long, Y. Bai, Y. Li, J. Wang, and J. Lai, "Optimizing wirelength and delay of fpga tile through floorplanning based on simulated annealing algorithm," in *2023 IEEE 15th International Conference on ASIC (ASICON)*, pp. 1–4, 2023.

[22] B. Ray, A. R. Tripathy, P. Samal, M. Das, and P. Mallik, "Half-perimeter wirelength model for vlsi analytical placement," in *2014 International Conference on Information Technology*, pp. 287–292, 2014.

[23] B. N. B. Ray, S. K. Mohanty, D. Sethy, and R. B. Ray, "Hpwl formulation for analytical placement using gaussian error function," in *2017 International Conference on Information Technology (ICIT)*, pp. 56–61, 2017.