MS Technical Paper: Placement Algorithms for Heterogenous FPGAs

Brian B Cheng
Rutgers University Department of Electrical and Computer Engineering

1 Keywords

- FPGA, EDA, Placement, Simulated Annealing, Optimization, RapidWright

2 Abstract

fdsafdsafdsa.

3 Introduction

Field-Programmable Gate Arrays (FPGAs) have witnessed unprecedented growth in capacity and versatility, driving significant advances in computer-aided design (CAD) and electronic design automation (EDA) methodologies. Since the early-to-mid 2000s, the stagnation of single-processor performance relative to the rapid increase in integrated circuit sizes has led to a design productivity gap, where the computational effort for designing complex chips continues to rise. In FPGA CAD flows—which traditionally encompass logic synthesis, placement, routing, and bitstream generation—the placement stage has emerged as one of the most time-consuming processes. Inefficiencies in placement not only extend design times from hours to days, thereby elevating cost and reducing engineering productivity, but also limit the broader adoption of FPGAs by software engineers who expect compile times akin to those of conventional software compilers.

In this paper, we recreate and experiment with established placement algorithms by leveraging the Xilinx's RapidWright, which is a semi-open-source API that offers backend access to Xilinx's industry-standard FPGA environment, Vivado. We implement a simulated annealing placer for Xilinx's 7-series FPGAs, with an emphasis on minimizing total wirelength while mitigating runtime. Our implementation is organized into three consecutive substages. The **prepacking** stage involves traversing a raw EDIF netlist to identify recurring cell patterns—such as CARRY chains, DSP cascades, and LUT-FF pairs—that are critical for efficient mapping and legalization. In the subsequent **packing** stage, these identified patterns, along with any remaining loose cells, are consolidated into SiteInst objects that encapsulate the FPGA's discrete resource constraints and architectural nuances. Finally, the **placement** stage employs a simulated annealing (SA) algorithm to optimally assign SiteInst objects to physical sites, aiming to minimize total wirelength while adhering to the constraints of the 7-series architecture.

Simulated annealing iteratively swaps placement objects, in our case, SiteInsts, guided by a cost function that decides which swaps should be accepted or rejected. Hill climbing is permitted by occasionally accepting moves that increase cost, in hope that such swaps may later lead to a better final solution. SA remains a popular approach in FPGA placement research due to its simplicity and robustness in handling the discrete architectural constraints of FPGA devices. While SA yields surprisingly good results given relatively simple rules, it is ultimately a heuristic and stochastic approach that explores the vast placement space by making random moves. Most of these moves will be rejected, meaning that SA must run many iterations, usually hundreds to thousands, to arrive at a desirable solution.

In the ASIC domain, where placers must handle designs with millions of cells, the SA approach has largely been abandoned in favor of analytical techniques, owing to SA's runtime and poor scalability. Modern FPGA placers have also followed suit, as new legalization strategies allow FPGA placers to leverage traditionally ASIC placement algorithms and adapt them to the discrete constraints of FPGA architectures. While this paper does not present a working analytical placer, it will explore ways to build upon our existing infrastructure (prepacker and packer) to replace SA with AP.

4 Xilinx 7-Series Architecture

Before any work can be done on a placer, one must consider what is being placed and upon what it is being placed.
(Square peg square hole analogy)
(Show the arch hierarchy).

how to organize this?

Device hierarchical objects:
Super Logic Region, Clock Region, Tile, Site, BEL

Device connection objects:
Switchbox, PIP, BELPin, SitePin

Device object instances:
SitePinInst, SiteInst, CellPin, Cell
EDIF objects:
EDIFNet, EDIFCell, EDIFPort
EDIFHierNet, EDIFHierCell, EDIFHierPort
EDIFHierCellInst, EDIFHierPortInst

5 RapidWright API

RapidWright represents the architecture constructs as faithfully as possible

The RapidWright API offers a user-friendly toolkit to take pre-implemented designs, that is, designs that have already been placed and/or routed by Vivado, and make custom optimizations to fit complex design criteria. RapidWright has several "BlockPlacers", a general Router, but no general Placer. We use this API to implement a crude general placer, in our case, a simulated annealer. (Show what parts of the FPGA toolchain that RapidWright already has and how our placer fits into that toolchain).

6 Simulated Annealing

(Describe the algorithm, then in context of 7-Series arhictecture).

7 Analytical Placement

Introduce placement with legalization methods.
Many AP approaches use a Global placement, Legalization, then Detailed placement approach.
Our SA only considers legal moves via bookkeeping structures, so it has no concept of global placement or legalization.
It is a detailed placement only approach.

- Talk about HeAP as an example.
- Global Placement, Legalization, then Detailed Placement approach.
- Global placement using the Bound2Bound net model from SimPL.
- Legalization Step 1:
  Find an area of the device that is overutilized (illegal) for twhich the blocks contained within must spread to a larger area.
  To obtain this overutilized area, adjacent locations (Sites, Tiles, etc.) on the device that are occupied by more than one block (SiteInst, ModuleInst, etc.) are repeatedly clustered together, until all clusters are bordered on all sides by non-overutilized locations.
  Then, the area (?) is expanded in both the x and y dimensions (?) until it is large enough to accommodate all blocks contained within.
  Specifically, the area is expanded until its "occupance" O_A divided by its "capacity" C_A is less than a maximum "utilization factor" β, where β is less than or equal to 1. HeAP uses 0.9.
- Legalization Step 2:
  Two cuts are generated: a "source cut" and a "target cut".
  The source cut pertains to the blocks being placed (SiteInsts in home buffer).
  The target cut pertains to the area into which the blocks are placed (away buffer).
  The source cut splits the blocks into 2 partitions, while the target cut splits the area into 2 sub-areas, into which the blocks in each partition are spread.
  Two objectives are minimized during this process: The imbalance between the number of blocks in each partition, and the difference in the utilization of each sub-area.
  Utilization defined as the occupancy divided by the capacity of the sub-area.
  $U_{sub-area} = \frac{O_{sub-area}}{C_{sub-area}}$
  To generate the source cut, the cells are first sorted by their x or y location, depending on the orientation of the desired cut.
  Once the cells are sorted, the source cut generation is akin to choosing a pivot in a sorted list, where all blocks to the left of the pivot are assigned to the left/bottom partition and all blocks to the right of the pivot are assigned to the right/top partition.
  The target cut is an x or y cut of the area such that all blocks in each partition fit in their respective sub-areas, and such that $|U_{sub-area_1} - U_{sub-area_2}|$ is minimized.