DAG-Map:

Graph-Based FPGA Technology Mapping for Delay Optimization

KUANG-CHIEN CHEN
Fujitsu America
JASON CONG
YUZHENG DING
ANDREW B. KAHNG
PETER TRAJMAR
University of California,
Los Angeles

This article presents a graph-based technology-mapping package for delay optimization in lookup-table-based FPGA designs. In contrast to most previous techniques, DAG-Map carries out technology mapping and delay optimization on the entire Boolean network. The package consists of three main parts: transformation of an arbitrary network into a two-input network, technology mapping for delay minimization, and area optimization in the mapping solution. The authors show DAG-Map's effectiveness in comparative benchmark tests.

THE FIELD-PROGRAMMABLE gate array is a relatively new technology that allows circuit designers to produce ASIC chips without going through the fabrication process. An FPGA chip usually consists of three components: programmable logic blocks, programmable interconnections, and programmable I/O blocks. Current technology implements programmable logic blocks with either *K*-input RAM/ROM lookup tables (*K*-LUTs)[1] or programmable multiplexers.[2] Programmable interconnections consist of one-dimensional, segmented channels or two-dimensional routing grids with switch matrices. Programmable I/O blocks provide a user-configurable interface between internal logic blocks and I/O pads.

The design process for FPGAs is similar to that for conventional gate arrays and standard cells. Starting from a high-level design specification, the process includes logic synthesis, technology mapping, placement, and routing. However, the end result is not a set of masks for fabrication, but rather a configuration matrix that sets the values of all the programmable elements in an FPGA chip.

Fast turnaround and low manufacturing cost have made FPGA technology popular for system prototyping and low- or medium-volume production. Moreover, lookup-table-based FPGAs (such as those developed by Xilinx[1]) allow dynamic reconfiguration of chip functions, leading to many interesting applications. But field-programmable components usually introduce longer delays than conventional devices, so performance is a major consideration in many applications.

In this article, we study the technology-mapping problem for delay optimization of lookup-table-based FPGAs. The technology-mapping problem is to implement a synthesized Boolean network with logic cells from a prescribed cell family. Researchers have done much work on the technology-mapping problem for conventional gate array or standard cell designs.[3,4] In particular, Keutzer showed that a Boolean network can be decomposed into a set of fan-out-free trees and that optimal technology mapping can be performed for each tree independently with a dynamic programming approach.[4] However, the conventional methods do not apply directly to the technology-mapping problem for FPGAs; a *K*-LUT can implement any one of 2^{2^K} *K*-input logic gates, and consequently the equivalent cell family is too large to be manipulated efficiently.

Recently, researchers have proposed a number of technology-mapping algorithms for area optimization in lookup-table-based FPGA designs, with the objective of minimizing the number of programmable logic blocks in the mapping solution. The MIS-pga program, developed by Murgai et al.,[5] first decomposes a Boolean net-

0740-7475/92/0900-007$03.00 © 1992 IEEE

On average, DAG-Map reduces both network delay and number of lookup tables.

work into a feasible network, using Roth-Karp decomposition and kernel extraction to bound the number of inputs at each node. Then the program enumerates all the possible realizations of each network node and solves the binate covering problem to get a mapping solution using the fewest lookup tables. An improved MIS-pga program[6] incorporates more decomposition techniques, including bin packing, cofactoring, and AND-OR decomposition. It solves the covering problem more efficiently via certain preprocessing operations.

The Chortle program and its successor Chortle-crf, developed by Francis et al.,[7,8] decompose a Boolean network into a set of fan-out-free trees and then carry out technology mapping on each tree, using dynamic programming. Chortle-crf uses bin-packing heuristics for gate-level decomposition, achieving significant improvement over its predecessor in solution quality and running time.

The Xmap program, developed by Karplus,[9] transforms a Boolean network into an if-then-else DAG (directed acyclic graph) representation and then goes through a simple marking process to determine the final mapping.

Another algorithm, proposed by Woo,[10] introduces the notion of invisible edges to denote edges that do not appear in the resulting network after mapping. The algorithm first partitions a network into subgraphs of reasonable size and then exhaustively determines the invisible edges in each subgraph.

Previous work on FPGA mapping for delay optimization includes Chortle-d, developed by Francis et al.,[11] and an extension of MIS-pga, developed by Murgai et al.[12] Chortle-d's basic approach is similar to that of Chortle-crf. It decomposes the network into fan-out-free trees and then uses dynamic programming and bin-packing heuristics to map each tree independently, at each step minimizing the depth of the node being processed. The method indeed reduces the depths of mapping solutions considerably, but it has two drawbacks. First, it decomposes the network into a set of fan-out-free trees. Although it guarantees the optimal depth for each tree (when the input limit of each lookup table is no more than six), this *prior* decomposition usually results in suboptimal depth for the overall network. Second, Chortle-d uses many more lookup tables than area optimization algorithms (MIS-pga and Chortle-crf) use.

The MIS-pga extension contains two phases: mapping, and placement and routing. The mapping phase computes a delay-optimized two-input network and then traverses the network from the primary inputs, collapsing nodes in the longest paths into their fan-outs to reduce network depth. During this procedure, various decomposition techniques dynamically resynthesize the network, so the program uses a reduced number of lookup tables. The advantage of this approach is that it takes layout information into consideration at the technology-mapping stage. However, on average it yields a greater network depth than Chortle-d, especially for large networks, and requires much more computation time.

In this article, we present DAG-Map, a graph-based technology-mapping algorithm for delay optimization in lookup-table-based FPGA designs. Our algorithm carries out technology mapping and delay optimization on the entire Boolean network, instead of decomposing it into fan-out-free trees as Chortle-d does. DAG-Map is optimal for trees for any K-LUTs, while Chortle-d is optimal for trees only when K is no more than six.[11] As a preprocessing phase of DAG-Map, we introduce a general algorithm called DMIG, which transforms an arbitrary n-node network into a two-input network with only an $O(1)$ factor increase in network depth. In contrast, previous transformation procedures may result in an $\Omega(\log n)$ factor increase in network depth. Finally, we present a matching-based technique that minimizes area without increasing network delay, which we use in the postprocessing phase of DAG-Map.

We have compared DAG-Map with previous FPGA mapping algorithms on a set of MCNC (Microelectronics Center of North Carolina) logic synthesis benchmarks. Our experimental results show that on average, DAG-Map reduces both network delay and the number of lookup tables, compared with either Chortle-d or the mapping phase of the MIS-pga extension for delay optimization.

Problem formulation

We can represent a Boolean network as a directed acyclic graph in which each node represents a logic gate, and in which a directed edge (i, j) exists if the output of gate i is an input of gate j. A primary input (PI) node has no incoming edge, and a primary output (PO) node has no outgoing edge. We use $input(v)$ to denote the set of nodes that supply inputs to gate v. Given a subgraph H of the Boolean network, $input(H)$ denotes the set of distinct nodes that supply inputs to the gates in H. For a node v in the network, a K-feasible cone at v, denoted C_v, is a subgraph consisting of v and predecessors of v such that any path connecting a node in C_v and v lies entirely in C_v, and $|input(C_v)| \leq K$. (Node u is a predecessor of node v if there is a directed path from u to v.) The *level* of a node v is the length of the longest path from any PI node to v. The level of a PI node is zero. The depth of a network is the largest node level in the network.

We assume that each programmable