AIG to MIG Conversion
Low Level Synthesis
Mini Task 1 Report

Lukas Rothenberger and Pallavi Gutta Ravi

01.07.2020

Table of Contents

Introduction

Nowadays, Electronic Design Automation tools are challenged by design goals of what is achievable in advanced technologies. In this scenario, recent logic synthesis works considered (slower) Boolean methods rather than (faster) algebraic methods to obtain superior circuit realizations, in terms of speed, power and area. Some data structures and algorithms have been proposed for these tasks. Most of them consider, as basis operations, inversion (INV), conjunction (AND), disjunction (OR) and if-then-else (MUX). Other Boolean operations are derived by composition.

Even though existing design automation tools, based on original optimization techniques, produce good results and handle large circuits, the possibility to push further the efficacy of logic synthesis still exists. With this aim in mind, we approach the logic optimization problem from a new angle. The underlying paper proposed a novel methodology to represent and optimize logic, by using only majority (MAJ) and inversion (INV) as basis operations. We generated the Majority-Inverter Graph (MIG), a logic representation structure consisting of three-input majority nodes and regular/ complemented edges. Majority-Inverter Graph (MIG) is a promising data structure for logic optimi zation and synthesis.

An MIG is a directed acyclic graph and MIGs include any AND/OR/Inverter Graphs (AOIGs), therefore also containing AIGs. To provide native manipulation of MIGs, we introduce a novel Boolean algebra, based exclusively on majority and inverter operations with a set of five primitive transformations forms a complete axiomatic system.Indeed, it is desirable to spend more time in logic synthesis computation to get a better final design. However, with traditional tools, it comes with limitations, for example running complex Boolean methods, does not improve a circuit quality or even requires too long runtime. But using a sequence of such primitive axioms, it is possible to explore the entire MIG representation space. This remarkable property opens great opportunities in logic optimization and synthesis. The potential of MIGs by proposing a delay-oriented optimization technique.

Experimental results, over the MCNC benchmark suite, show that MIG optimization decreases the number of logic levels by 18%, on average, with respect to AIG optimization run by ABC academic tool. Applied in a standard optimization-mapping circuit synthesis flow, MIG optimization enables a reduction in the estimated delay, area, power metrics of 22%, 14%, 11%, on average before ph ysical design, as compared to academic/commercial synthesis flows. The study of majority-inverter logic synthesis is also motivated by the design of circuits in emerging technologies.

Background and Motivation

This section gives a background on logic optimization and MIGs.

A. Logic Representation and Optimization

Early data structures and related optimization algorithms are based on two-level representation of Boolean functions in Sum Of Product form. Another representation is the Binary Decision Diagram: a canonical representation form based on nested if-then-else (MUX) formulas. Later, multi-level logic networks emerged, employing AND, OR, INV, MUX operations as basic functions. To deal with the continuous increase in logic designs complexity, a step further is enabled by, where multi-level logic networks are made homogenous, i.e., consisting of only AND nodes interconnected by regular/compl ement (INV) edges.

Logic optimization methods are usually divided into two groups: Algebraic methods, which are fast, and Boolean methods, which are slower but achieve better results. Algebraic methods treat a logic functions as a polynomial and selectively iterate over the entire logic circuits, until an improvement exists. Instead, Boolean methods handle the true nature of a logic function using Boolean identities as well as (global) don?t cares (circuit flexibilities) to get a better solution. Boolean division and substitution techniques trade off runtime for better minimization quality.

B. Notations and Definitions

1) Boolean Algebra:
In the Boolean domain, the symbol B indicates the set of binary values 0, 1.

2) Logic Network:

A logic network is a Directed Acyclic Graph (DAG) with nodes corresponding to logic functions and directed edges interconnecting the nodes. The direction of the edges follows the natural computation from inputs to outputs. The incoming edges of a node link either to other nodes, to input variables or to logic constants 0/1. Two logic networks are said equivalent when they represent the same Boolean function. A logic network is said irredundant if no node can be removed without altering the represented Boolean function. A logic network is said homogeneous if each node has an indegree (number of incoming edges, fan-in) equal to k and represents the same logic function. The depth of a node is the length of the longest path from any input variable to the node. The depth of a logic network is the largest depth of a node. The size of a logic networkis its number of nodes.

3) Majority Function:
The n-input (n odd) majority function M returns the logic value assumed by more than half of the inputs.

Majority Inverter Graph

A Majority-Inverter Graph (MIG) is a data structure for Boolean function representation and optimization. An MIG is a logic network consisting of 3-input majority nodes and regular/complemented edges. Each majority node can be reduced to a conjunction (AND) or a disjunction (OR) operator by fixing the third input to 0 or to 1, respectively. However, even better MIG representations appear by exploiting MIG nodes functionality (majority) rather than reducing it to AND/OR. To natively optimize and reach advantageous MIGs, a MIG Boolean algebra is introduced and axiomatized by five primitive transformation rules given by Omega .

Omega comprises of:
? Commutativity C : M(x, y, z) = M(y, x, z) = M(z, y, x)
? Majority M : if(x = y): M(x, y, z) = x = y, If(x = y): M(x, y, z) = z
? Associativity A : M (x, u, M(y, u, z)) = M(z, u, M(y, u, x))
? Distributivity D : M(x, y, M(u, v, z)) = M(M(x, y, u),M(x, y, v), z)
? Inverter Propagation I : M(x, y, z) = M(x, y, z)

From a theoretical perspective, it is possible to traverse the entire MIG representation space just by using a sequence of transformations. We call the MIG optimization techniques algebraic because they locally use MIG algebra transformations. The paper proposed alternative techniques, focusing on global properties of MIGs such as voting resilience and don?t care conditions. Due to their global and general nature, the proposed MIG optimization methods are ?Boolean?. Desirable properties for a logic system are soundness and completeness. Soundness ensures that if a formula is derivable from the system, then it is valid. Completeness guarantees that each valid formula is derivable from the system. However, the length of the exact transformation sequence might be impractical for modern computers.

To alleviate this problem, we derive from omega three powerful transformations, referred to as PSI, that facilitate the MIG manipulation task. The first, relevance (psi.R), replaces and simplifies re-convergent variables. The second, complementary associativity (psi.C), deals with variables appearing in both polarities. The third and last, substitution (psi.S), extends variable replacement also in the non-re-convergent case. We represent a general variable replacement operation, say replace x with y in all its appearance in z, with the symbol z(x/y').

Psi comprises of :
? Relevance R: M(x, y, z) = M(x, y, z x/y')
? Complementary Associativity C: M(x, u, M(y, u', z)) = M(x, u, M(y, x, z))
? Substitution S: M(x, y, z) = M(v, M(v', Mv/u(x, y, z), u), M(v', Mv/u'(x, y, z), u'))

Boolean Logic Optimization in MIG

The first step in the optimization of the MIG is the to convert all the AND gates to MAJ gate. To do this we replace and gates by MAJ gates and add an additional input with a constant Boolean 0 or 1. This creates homogenous node and hence easy to generate the Majority graph from it which is again Homogenous.

Optimizing the Size of an MIG:

To optimize the size of an MIG, we aim at reducing its number of nodes. Node reduction can be done, at first instance, by applying the majority rule. In the Boolean algebra domain, that is the ground to operate on MIGs, this corresponds to the evaluation of the majority axiom (M) from Left to Right (L ? R), as M(x, x, z) = x. A different node elimination opportunity arises from the distributivity axiom (D), evaluated from Right to Left (R ? L), as M(x, y, M(u, v, z)) = M(M(x, y, u),M(x, y, v), z). By applying repeatedly M(L?R) and D(R?L) over an entire MIG, we can actually eliminate nodes and thus reduce its size.

There may be MIGs where no direct node elimination is evident. This is because
(i) the optimal size is reached
(ii) we have a local minima.

In the latter case, we want to reshape the MIG to enforce new reduction opportunities.

The reshaping process is to locally increase the number of common inputs/variables to MIG nodes. For this purpose, the associativity axioms (PSI.A, PSI.C) allow us to move variables between adjacent levels and the relevance axiom (PSI.R) to exchange re-convergent variables. When a more radical transformation is beneficial, the substitution axiom (PSI.S) replaces pairs of independent variables, temporarily inflating the MIG. Once the reshaping process created new reduction opportunities, majority M(L?R) and distributivity D(R?L) run again over the MIG simplifying it. Reshape and elimination processes can be iterated over a user-defined number of cycles, called effort. Algorithm used is shown below:

```
-------------------------------------------------------------------------------
INPUT: MIG ?  OUTPUT: Optimized MIG ?
-------------------------------------------------------------------------------
for (cycles=0; cycles<effort; cycles++) do
?.ML?R(?); ?.DR?L(?);
?.A(?); ?.C(?);
?.R(?); ?.S(?);
?.ML?R(?); ?.DR?L(?);
end for
-------------------------------------------------------------------------------
```

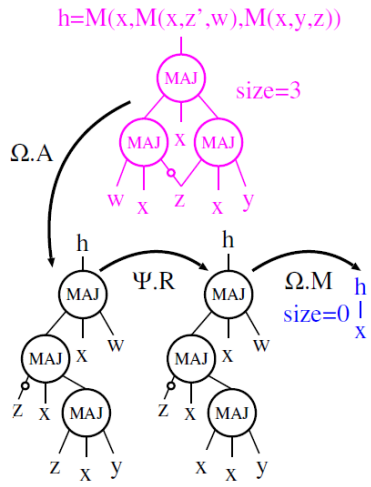

Figure 1: Examples of MIG optimization of size The input MIG is equivalent to the formula M(x, M(x, z', w),M(x, y, z)), which has no evident simplification by majority and distributivity axioms. Consequently, the reshape process is invoked to locally increase the number of common inputs. Associativity A swaps w and M(x, y, z) in the original formula obtaining M(x, M(x, z', M(x, y, z)),w), where variables x and z are close to the each other. Later, relevance R applies to the inner formula M(x, z', M(x, y, z)), exchanging variable z with x and obtaining M(x, M(x, z', M(x, y, x)),w). At this point, the final elimination process runs, simplifying the reshaped representation as M(x ,M(x, z', M(x, y, x)),w) = M(x, M(x, z', x),w) = M(x, x, w) = x by using M(L?R). The obtained result is optimal.

Design decisions and Implemenatation

Results and Evaluation

Conclusion and Reference