TEXAS A&M UNIVERSITY LIBRARY

DETECTION AND REMOVAL OF FUNCTIONAL REDUNDANCY IN MULTI-LEVEL LOGIC CIRCUITS

FOR UNIVERSITY UNDERGRADUATE

RESEARCH FELLOWS

A Senior Honors Thesis

By

DAVID MICHAEL DORSEY

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

April 2002

Group: Computer Science

DETECTION AND REMOVAL OF FUNCTIONAL REDUNDANCY

IN MULTI-LEVEL LOGIC CIRCUITS

FOR UNIVERSITY UNDERGRADUATE

RESEARCH FELLOWS

A Senior Honors Thesis

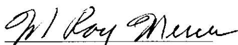
By

DAVID MICHAEL DORSEY

Submitted to the Office of Honors Programs
& Academic Scholarships
Texas A&M University
in partial fulfillment of the requirements of the

UNIVERSITY UNDERGRADUATE
RESEARCH FELLOWS

Approved as to style and content by:



M. Ray Mercer
(Fellows Advisor)

Edward A. Funkhouser
(Executive Director)

April 2002

Group: Computer Science

Abstract

Detection and Removal of Functional Redundancy

in Multi-level Logic Circuits. (April 2002)

David Michael Dorsey
Department of Electrical Engineering
Texas A&M University

Fellows Advisor: Dr. M. Ray Mercer
Department of Electrical Engineering

Whenever digital designs are created, they may contain many logic redundancies. Minimization tools are then used to remove these redundancies. The minimized circuit should be smaller, faster, and cheaper while still behaving like the original circuit. This research will focus on finding non-traditional methods for minimizing multi-level logic circuits.

Acknowledgements

Much thanks goes to Dr. M. Ray Mercer, Jennifer Dworak, and Nathan Mickler for all their valuable contributions

Table of Contents

Page

Abstract ... iii

Acknowledgements ... iv

Table of Contents ... v

Table of Figures ... vi

Introduction ... 1

Procedure ... 5

Results ... 8

Analysis ... 8

Conclusions & Future Work ... 10

References ... 11

Table of Figures

Page

Figure 1: Different Implementations of Equivalent Logic Functions 2

Figure 2: Two Isomorphic Functions 5

Figure 3: Two Isomorphic Structures 6

Figure 4: An Example of Structural Redundancy 9

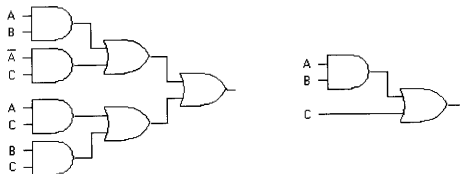
Introduction

There are many different ways to construct a multi-level digital logic function. Some implementations of a particular logic function are more efficient than other implementations. The goal of multi-level logic minimization is to transform a logic circuit C1 into an identical functioning circuit C2, circuit C2 being less expensive than C1 according to a cost function. A cost function typically includes area, propagation delay, power consumption, and testability as the primary goals of the optimization procedure [Kunz97].

Logic minimization is important because current synthesis tools inefficiently translate a hardware description language to its gate level equivalent. After this translation, synthesis tools then perform logic minimization. The current method to minimize these circuits is through manipulation of Boolean expressions. However, as circuits grow more complex, memory limitations severely hinder this method as it minimizes the circuit [Mehler99].

Since there exist several different implementations of a logic function, the problem is moving from a complex circuit C1 to a less complex, but identically functioning circuit C2. For a two level logic circuit with less than six inputs, the transformation is relatively simple - all that is required is a Karnaugh map. However, above six variables the Karnaugh map loses its effectiveness [Katz94]. The two circuits shown in Figure 1 on the following page are equivalent circuits, yet the circuit on the left is more complex than the circuit on the right.

Figure 1: Different Implementations of Equivalent Logic Functions



Another limitation of the Karnaugh map is that it always produces a two level circuit. This can lead to a large circuit that is efficient in terms of propagation delay, but possibly extremely inefficient in terms of area. Quinn-McCluskey is a computer-based technique for performing two-level logic minimization. It works well for circuits up to about 20 inputs, but the complexity of the approach grows exponentially with the number of circuit inputs [McCluskey86].

Reducing a multi-level logic circuit to a more efficient implementation is much more complex than two-level logic minimization. Even constructing a truth table for a moderate size circuit can prove overwhelming. If a circuit has n inputs, there are 2^n input combinations. A circuit with forty inputs has over one trillion input combinations. There are several algorithms to reduce multi-level logic circuits. Boolean algebra is a commonly used tool. A circuit is translated to its Boolean algebraic equivalent and then is minimized through the properties of Boolean algebra. The major limitation of this approach is the memory requirements needed to perform these calculations on a computer [Katz94].

Since it is not always possible to check every input combination (pattern) for a circuit, methods have been developed to model a circuit's behavior based on a relatively small amount of input patterns. Automatic Test Pattern Generation (ATPG) is the tool that is used to thoroughly test a logic circuit using only a small amount of input patterns.

When a point internal to a logic circuit affects the output of that circuit, then that point is said to be observable. If two points are at exactly the same value every time those points are observable, the points are weakly equivalent. A fault simulation tool can be used to identify when points on the circuit are observable and when values at those points do not matter. If two circuits have the possibility of being compatible, a miter circuit can be used to validate their equivalence. A miter circuit uses an exclusive-or gate to compare the output of the current circuit and the output of the possible new circuit. An exclusive-or gate is used because it will output logic one if the outputs of the circuits differ. If the output of the miter circuit is both observable and logic one, these two circuits are not compatible and the possible new circuit cannot be considered to be an appropriate transformation from the original circuit. In order to make this determination using an ATPG tool, the output of the added exclusive-or gate in the miter circuit is considered to be stuck-at-zero. ATPG is then used to try to detect the stuck-at zero at the output of the exclusive-or gate. If a full test by an ATPG detects this fault, it means the two circuits were not equivalent when the point was observable. In such a case, the potential transformation from one of the circuits to the other is then discarded because the two circuits are not compatible. While this approach has the advantage of using any one of a set of commercially available logic tools that regularly run on circuits

with millions of gate, a full test by the ATPG in some cases can be a time consuming task.

An alternative method to Boolean manipulation is to reduce gates in a gate netlist. Previous efforts in this area have met with success. The Texas Aggie Logic Optimizing Netlister (TALON) was shown to be comparable to commercial tools and could be used in addition to commercial tools. TALON uses single stuck-at fault simulation for a limited number of input vectors to portray the full set of input vectors. For each point in the circuit, there are three values

1. Stuck-at zero
2. Stuck-at one
3. No stuck-at fault detected.

If there is no fault detected, the value at that point does not affect the output. TALON then checks to see if points in the circuit are compatible with other points in the same circuit. Two points do not have to be the same for all input vectors to be exchangeable; they only have to be the same when that point in the circuit has an effect on the output. TALON minimizes a circuit in a step-by-step manner. TALON investigates one potential transformation and verifies the validity of that transformation. It then tests the newly created circuit again for any possible redundancies [Mehler99].

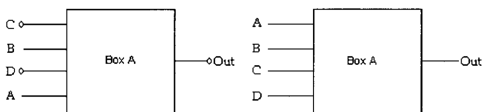
This research seeks to enhance this process by finding new rules to minimize logic circuits.

Procedure

To find rules for minimization, we had to decide how many switching variables to use. Three switching variables create 256 different functions while four switching variables create 65,536 different possible functions. Three switching variables create a manageable number of functions, but we decided it was too few functions to be interesting. However, four variables gave us so many functions we had to find a way to bridge the gap to a more manageable number of functions.

This gap was bridged by functional isomorphism. This takes advantage of the similarities between many functions. Constructing a logic function can be viewed as a two-step procedure. The first step is to define the structure of the circuit and assign gate types. The circuit itself is then considered "inside the box." In our research, this box was limited to four inputs and one output.

Figure 2: Two Isomorphic Functions



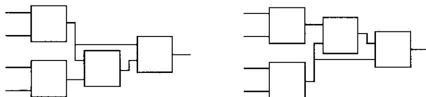
The second step assigns variables to inputs and possibly inverts any input and the output. This is done "outside the box." It has no affect on the internal structure of the box. Figure 2 shows two isometric functions; they only differ outside the box.

Inversions are shown through "bubbles", as shown inputs of C and D and the output of the function on the left. Depending on how step two is performed, different functions can be generated. However, all the functions created during step two are isomorphic to each other. Two functions, A and B, are said to be isomorphic to each other if function A can be transformed into function B by inverting the output, inverting one or more of the inputs, reconnecting the inputs in a different order, or any combination of these.

When we applied this idea to all functions of four variables, we reduced the effective number of functions from 65,536 to 222 – a reduction of over 99%. In addition, fourteen of these are degenerate functions. A degenerate function is a function that is not dependent on all the input variables. This leaves us with only 208 fundamentally different functions in our set of interest.

With this reduction, we set out to find rules to minimize the structure inside the box. To do this, we started by looking at all possible non-isomorphic structures of three and four gates. Isomorphic structures are structures that are laid out differently but are fundamentally the same. Figure 3 gives an example of two structures isomorphic to each other. As you can see, the bottom one is merely the mirror image of the top one.

Figure 3: Two Isomorphic Structures



We created all non-isometric structures of three and four gates. For three gates, there are only two different structures. We used successive extension to get from the simple three gate structures to the four gate structures. This technique adds a gate at every unique point of the circuit. It then adds a fan-out from every other unique point of the circuit that would maintain the circuit as a combinational circuit. We did not want a sequential circuit because ths would fundamentally change the nature of the logic function being realized. When we applied this to the three gate structures, we found thirty unique structures of four gates.

We also limited the gate types that went inside the box to five different two-input gates. They are the AND gate, the AND gate with one input inverted, the AND gate with the other input inverted, the AND gate with both inputs inverted, and the XOR (exclusive-or) gate. This set is one minimal set of gates that can realize every possible switching function for any given structure.

With these results, we were then able to make circuits and analyze them to find rules to move from a one structure to a more nearly minimal structure. Our procedure was to create all possible circuits and then identify what function they produced. Once we identified the function, we transformed it into the representative function for that isometric family and stored the structure. We called the process of identifying and transforming to the representative function "spinning." A by-product of spinning was the creation of identical circuits. We used a concept called "bubble pushing" to keep only one copy of identical circuits. Since spinning can invert inputs, we push these

bubbles (inversions) inside the box. Once the bubbles are inside the box, the circuit is changed depending on the gate type and the location of a bubble.

After creating and classifying every possible circuit for all three and four gate structures, we had some interesting results.

Results

There are 250 different possible circuits of three gates divided evenly between the two non-isomorphic structures. These circuits realized only 18 of the 222 different isometric families. With three gates, we could realize less than 9% of the total number of non-isometric logic functions. In addition, after eliminating all redundant circuits from the original set of 250, only 22 unique circuits remain.

There are 18,750 different possible circuits formed of four two-input gates, and they are divided evenly among thirty non-isomorphic structures. However, these circuits only realized 59 of the 222 different non-isomorphic logic functions. This is still less than one-third of the total number of functions. All but one of the functions realized using three two-input gates were also realized by at least one of the circuits formed using four two-input gates.

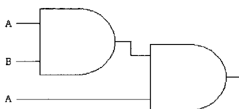
Analysis

It is surprising that all possible circuits of three and four gates create only 27% of the 222 potential non-isomorphic functions of four variables. This means that the

majority of the functions minimize to five gates at best. This is important because we need to know when we arrive at the minimal circuit for any given function.

Despite only creating 27% of the functions, we were able to find some simple rules for minimization. For the four two-input gate structures and three two-input gate structures that realized the same function, every one of the four two-input gate structures contained structural redundancy. Figure 4 shows a simple example of a structural redundancy. In this example, the logic operation "A and B" is performed twice. Only once is necessary; the second gate does not accomplish anything. We can detect structural redundancy in this case by determining that if the upper "A" input is fixed to the value of logic 1, the realized circuit function is not changed. This suggests replacement of the first AND gate by the primary input "B," and this step results in the desired minimum realization.

Figure 4: An Example of Structural Redundancy



Using isometric functions and isometric structures to reduce the amount of data to analyze was an important achievement of this research. As indicated in the Results

section, the 250 possible circuits of three gates were reduced to merely 22 – a reduction of 91.2%. The level of reduction for the four gate structures is similar.

Conclusions & Future Work

Eliminating redundant circuits by using spinning and bubble pushing is an important step in our logic minimization research. Spinning allows us to easily identify and transform a function into the representative function of that isometric family. Bubble pushing allows us to eliminate most of the circuits created by spinning because they are identical.

Much future work is needed in this area. One area will focus on finding circuits that realize the 162 functions that we have not yet created. We do not know the minimum number of gates needed for most of the functions out there and finding them is an important step. Another area, and in my opinion the most important area, is finding reductions or transformations to move from a complex circuit to a more minimal circuit. Lastly, work is needed in the creation of isomorphic structures of five or more gates. Creating the isomorphic structures of three and four gates was a simple task to do by hand, but creating all the five gate structures will be too time consuming because of sheer numbers. This process will need to be automated by a computer program.

References

[Katz94] Randy H. Katz, *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994

[Kunz97] W. Kunz, D. Stoffel, and P. R. Menon, "Logic Optimization and Equivalence Checking by Implication Analysis," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 16, 1997, pp. 266-281

[McCluskey86] Edward J. McCluskey, *Logic Design Principles*, Prentice – Hall, 1986.

[Mehler97] R. Mehler and M. R. Mercer, "Multi-level Logic Minimization Through Fault Dictionary Analysis," *Proceedings of the 1999 International Conference on Computer Design*, 1999, pp. 315-318.