Integrated Logic Synthesis Using Simulated Annealing

PETRA FÄRM

Doctoral Thesis in Electronic System Design
Stockholm, Sweden 2007

TRITA ICT/ECS AVH 07:01
ISSN 1653-6363
ISRN KTH/ICT/ECS AVH-07/01-SE
ISBN 91-7178-516-7/978-91-7178-516-9

KTH School of Information and
Communication Technology
SE-164 40 Kista
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i elektronik fredagen den 26 januari 2007 klockan 9.00 i Sal E, Forumhuset, Kungl Tekniska högskolan, Isafjordsgatan 39, Kista.

© Petra Färm, januari 2007

Tryck: Universitetsservice US AB

Abstract

A conventional logic synthesis flow is composed of three separate phases: technology independent optimization, technology mapping, and technology dependent optimization. A fundamental problem with such a three-phased approach is that the global logic structure is decided during the first phase without any knowledge of the actual technology parameters considered during later phases. Although technology dependent optimization algorithms perform some limited logic restructuring, they cannot recover from fundamental mistakes made during the first phase, which often results in non-satisfiable solutions.

We present a global optimization approach combining technology independent optimization steps with technology dependent objectives in an annealing-based framework. We prove that, for the presented move set and selection distribution, detailed balance is satisfied and thus the annealing process asymptotically converges to an optimal solution. Furthermore, we show that the presented approach can smoothly trade-off complex, multiple-dimensional objective functions and achieve competitive results. The combination of technology independent and technology dependent objectives is handled through dynamic weighting. Dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels. The results show that, on average, the presented advanced annealing approach can improve the area and delay of circuits optimized using the Boolean optimization technique provided by SIS with 11.2% and 32.5% respectively.

Furthermore, we demonstrate how the developed logic synthesis framework can be applied to two emerging technologies, chemically assembled nanotechnology and molecule cascades. New technologies are emerging because a number of physical and economic factors threaten the continued scaling of CMOS devices. Alternatives to silicon VLSI have been proposed, including techniques based on molecular electronics, quantum mechanics, and biological processes. We are hoping that our research in how to apply our developed logic synthesis framework to two of the emerging technologies might provide useful information for other designers moving in this direction.

Acknowledgement

This dissertation could not have been written without Associate Professor Elena Dubrova who not only served as my supervisor but also encouraged and challenged me throughout my academic program. She provided the opportunities for several research visits at Cadence Berkeley Labs, as well as a semester at UC Berkeley. I thank Professor Andreas Kuehlmann for inviting me to Cadence Berkeley Labs and UC Berkeley. Professor Kuehlmann has been an excellent adviser, and helped me make many of the crucial design and implementation decisions.

I am grateful for the opportunity to be Johan Wennlund's teaching assistance. It has been a pleasure working with him and Fredrik Lundevall. The administration staff has been really helpful during my years at KTH, especially Agneta Herling, Lena Beronius and Gunnar Johansson. The thesis is reviewed by Docent Johnny Öberg, who provided constructive criticisms. The KTH library, both at Campus as well as in Kista, deserves a big thank for the excellent service provided. During my last years as a graduate student I had the fortune to share office with Delia Rodriguez De Llera Gonzal. She turned out to be not only a brilliant researcher, but also a great friend.

Finally, this thesis could not have been written without the support from my family. My partner in life, Roland Mattsson has not only helped me with technical issues, such as drawing graphs, how to use Excel and proof-reading, but also supported me at a more personal level. For example, he was a marvelous house-husband tending to our son and our rented house during my semester at UC Berkeley. The last person to be mentioned is my mother, Kristina Färm whose love and courage can not be described in words. I love you, mum.

Contents

Contents	v
List of Figures	viii
List of Tables	xi
1 Introduction	1
1.1 Contributions	4
1.2 Thesis layout	6
2 Background	9
2.1 Notation	9
2.2 Boolean algebra	10
2.3 Programmable logic array	12
2.4 Graphs	13
2.5 NPN classes	19
2.6 Finite Markov chains	20
3 Logic synthesis	23
3.1 Technology independent optimization	25
3.2 Technology mapping	28
3.3 Technology dependent optimization	30
3.4 Synthesis tools	32
3.5 OpenAccess	37
4 And/Inverter graphs	45
4.1 Graph representation	45
4.2 Structural hashing	47

4.3	Applications	50
5	A greedy local transformation-based method	51
5.1	Local transformations	51
5.2	Optimization algorithm	56
5.3	Experimental results	57
5.4	Conclusion	58
6	Simulated annealing	63
6.1	The physical analogy	64
6.2	Mathematical model	65
6.3	Annealing schedule	67
6.4	Convergence to optimum	71
6.5	Applications	72
7	Simulated annealing at And/Inverter graph level	73
7.1	Previous work	73
7.2	The annealing framework	76
7.3	Implementation details	83
7.4	Experimental results	86
7.5	Conclusion	88
8	Combining technology independent and dependent optimization objectives	93
8.1	Combined optimization algorithm	93
8.2	Experimental results	96
8.3	Conclusion	99
9	Advanced simulated annealing with integrated technology mapping	105
9.1	Previous work	106
9.2	The advanced annealing algorithm	108
9.3	Technology mapping	110
9.4	The feedback mechanisms	114
9.5	Experimental results	115
9.6	Conclusion	119

10 Applications to new emerging technologies	121
10.1 Chemically assembled electronic nanotechnology	122
10.2 Molecule cascades	125
10.3 Specifics of technology mapping for CAEN and MC techniques	128
10.4 Experimental results	132
10.5 Conclusion	137
11 Conclusion	139
Bibliography	141

List of Figures

2.1	The three-dimensional Boolean space.	11
2.2	A PLA for the multiple output function $f_1 = x\bar{y} + yz$, $f_2 = \bar{x}z + xyz$ and $f_3 = xyz + \bar{x}\bar{y}$	13
2.3	a) Undirected graph. b) Directed graph.	14
2.4	Reduced ordered binary decision diagrams for $f = (x + y)z$ with the variable order (x, y, z)	16
2.5	Binary decision diagrams for $f = (x + y)z$: a) OBDD for the vari- able order (x, y, z) . b) OBDD for the variable order (x, z, y)	17
2.6	NPN representative function realization.	20
3.1	Design flow of integrated systems.	24
3.2	Typical synthesis scenario.	25
3.3	SIS <i>script algebraic</i> which is mainly based on algebraic optimiza- tion steps.	37
3.4	SIS <i>script rugged</i> which implements Boolean optimization.	38
3.5	OpenAccess standard and reference implementation.	40
3.6	OA Gear functionality.	44
4.1	AND/INVERTER graph for the function $f = \bar{x}\bar{y} \cdot \bar{y}\bar{z}$	46
4.2	AND/INVERTER graph with arrival times.	47
4.3	AND/INVERTER graph with required times.	48
4.4	AND/INVERTER graph with level slacks.	48
4.5	Pseudo-code for the algorithm CREATEAND2, one of the AND/INVERTER graph constructor.	49
4.6	Algorithm NEWANDVERTEX for allocating a new graph vertex.	50
5.1	Two different logic expressions for NPN class 8; a) $f = xy + xz$ and b) $f = x(y + z)$	54

5.2	Two different logic expressions for NPN class 13; a) $f = x\bar{y} + x\bar{z} + \bar{x}yz$ and b) $f = x(y\bar{z} + \bar{y}z) + \bar{x}yz$	54
5.3	Two different logic expressions for a four variable NPN function; a) $f = xyz + zw$ and b) $f = z(xy + w)$	55
5.4	Pseudo-code of the NPN-based greedy optimization algorithm.	56
6.1	Pseudo-code of the Metropolis algorithm.	66
6.2	Pseudo-code of the annealing algorithm.	68
7.1	Illustration of the local replacement rules.	77
7.2	Illustration of replacement rule 1(b) of Definition 3; a) $f = x \cdot y + z \cdot y$ and b) $f = y \cdot (x + z)$	79
7.3	Illustration of replacement rule 1(e) of Definition 3; a) $f = x \cdot 1 + y$ and b) $f = x + y$	79
7.4	Illustration of a state transition graph for a small example of our annealing setup. The states are represented by AND/INVERTER graphs.	80
7.5	Pseudo-code of the AND/INVERTER graph based-annealing algorithm.	84
7.6	Convergence of the annealing algorithm for benchmark C1355.	88
8.1	Pseudo-code of the presented algorithm, combining simulated annealing, greedy optimization and technology mapping.	95
8.2	An example of how the number of vertices in an AND/INVERTER graph can change by applying rule 1(a) of Definition 3.	96
8.3	An example of how the number of levels in an AND/INVERTER graph can change by applying rule 1(a) of Definition 3. The numbers represent the level of the corresponding vertex.	97
8.4	Area/delay trade-off for different settings of variable x in the cost function given by Equation 8.1 for the presented combined optimization algorithm.	103
9.1	Pseudo-code of the advanced annealing algorithm with integrated technology mapping.	109
9.2	Pseudo-code of the optimal area cover algorithm.	111
9.3	Pseudo-code of the optimal delay cover algorithm.	112
9.4	AND/INVERTER graph with level slacks. The critical path has level slacks marked in bold.	114

9.5	AND/INVERTER graph with level slacks adjusted by dynamic weights. The critical path has level slacks marked in bold. . . .	115
9.6	Area/delay trade-off for different values of x in the objective function $\Delta \epsilon_{ij}(x) = \Delta n_{ij} + x \cdot \Delta l_{ij} + 0.4 \cdot \Delta o_{ij}$, where Δn_{ij} is the change in number of vertices, Δl_{ij} is the change in level slack, and Δo_{ij} is the change in fanout distribution.	116
9.7	Delay/execution time trade-off for different values of the mapper frequency m	118
10.1	This atomic force microscope image shows a pattern of nanoparticles that self-assembled themselves on a pre-pattern substrate. Photo credit: Prabhakaran et al. [1]	123
10.2	Logic diagram of a nanoBlock implementing an AND gate. . . .	125
10.3	A logical AND gate consists of 2 input arms (X and Y) which meet in the center of the gate. The central carbon monoxide molecule becomes part of a chevron only after both inputs have been triggered and have propagated to this molecule. Only then is the output cascade triggered. All hops in an AND gate and in the connecting wires are based on the chevron principle. Photo credit: IBM.	127
10.4	Logic AND gate in MC after input X was triggered and the resulting cascade propagated to the central molecule. Photo credit: IBM.	128
10.5	Logic diagram of a molecular block.	129
10.6	Example of (a) a network of interconnected molecular blocks; (b) the corresponding AND/INVERTER graph representation.	131
10.7	A two-level half-adder implemented in CAEN.	133

List of Tables

2.1	Some properties of Boolean algebraic systems.	11
2.2	NPN classes for three variable functions.	19
3.1	A subset of the available commands in SIS.	36
5.1	Replacement database for three variable NPN classes.	53
5.2	Comparison of the presented greedy local transformation-based system with SIS.	59
5.3	Comparison of the presented greedy local transformation-based system with SIS.	60
7.1	Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of vertices with SIS. . . .	89
7.2	Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of levels with SIS. . . .	90
7.3	Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of vertices with the previously presented NPN-based greedy optimization algorithm. . .	91
8.1	Comparison of the presented combined optimization algorithm with SIS for $x = 0$ in cost function given by Equation 8.1. . . .	100
8.2	Comparison of the presented combined optimization algorithm with SIS for $x = 0.5$ in cost function given by Equation 8.1. . . .	101
8.3	Comparison of the presented combined optimization algorithm with SIS for $x = 1$ in cost function given by Equation 8.1. . . .	102
9.1	Comparison of the presented algorithm with SIS <i>script rugged</i> for $x = 0.5$ and $m = 1,000$	117

10.1	Experimental results for chemically assembled nanotechnology (CAEN) using a greedy optimization algorithm.	134
10.2	Comparison of a greedy NPN-based optimization algorithm for CAEN and MC with SIS.	136
10.3	Comparison of the simulated annealing algorithm for CAEN and MC with SIS.	138

Chapter 1

Introduction

Electronic Design Automation (EDA) is the category of tools for designing and producing electronic systems ranging from printed circuit boards to integrated circuits. EDA has rapidly increased in importance with the continuous scaling of semiconductor technology. EDA is divided into many sub-areas. They mostly align with the path of manufacturing from design to mask generation. Some of the sub-areas of EDA are:

- Design and architecture: Design the chip's schematics, output in Verilog, VHDL, SPICE and other formats.
- Floorplanning: The preparation step of creating a basic die-map showing the expected locations for logic gates, power and ground planes, I/O pads, and hard macros.
- Logic synthesis: Translation of a chip's abstract, logical register transfer level (RTL) description into a discrete netlist of logic gate primitives.
- Formal verification, also model checking: Attempts to prove, by mathematical models, that the system has certain desired properties, and that certain undesired effects (such as deadlock) can not occur.
- Equivalence checking: Algorithmic comparison between a chip's RTL-description and synthesized gate-netlist, to ensure functional equivalence at the logical level.

- Place and route: Tool-automated placement of logic gates and other technology mapped components of the synthesized gate-netlist, then subsequent routing of the design, which adds wires to connect the components' signal and power terminals.
- Physical verification: Checking if a design is physically manufacturable, and that the resulting chips will not have any function-preventing physical defects, and will meet original specifications.
- Mask data preparation: Generation of actual lithography photomask used to physically manufacture the chip.
- Manufacturing test.

This thesis aims at improving current logic synthesis frameworks. Logic synthesis is an EDA process of automatically generating an optimized logic level representation from a high-level description. The complexity and significance of this task depends on the level of input specification, the type of logic implementation, and the criteria for an acceptable result. The level of specification can range from behavioral where only the relationship of outputs to inputs is given, to the register transfer level where the state is explicitly defined, to the structural level where the specification is given as an interconnection of hardware primitives. There are also different levels of logic implementation ranging from a set of Boolean equations, to a list of interconnected technology specific hardware primitives, to detailed mask data for manufacturing a chip.

Traditional logic synthesis is partitioned into multiple phases, such as technology independent synthesis, technology mapping, and timing optimization [2, 3, 4]. During the technology independent phase, simple literal count is applied as the primary quality metric for defining the global structure of the circuit implementation. The main reason for this is that counting literals provides a simple and straightforward measure of quality. Furthermore, certain algorithms, e.g. two-level minimization [5] or algebraic division [6], can be solved optimally or near optimally for this metric. On the other hand, the number of literals correlates only loosely with the actual design objectives such as area, timing, power, etc. Timing and power requirement are mostly handled during and after the technology mapping phase and can only perform local changes of the logic structure. As a result of this flow, bad choices made during technology independent synthesis can

only be corrected for special cases e.g., reducing the slack of a single late input by co-factoring the circuit structure [7, 8].

A possible solution to the defined problem of the partition of logic synthesis into multiple phases, is to integrate them with each other. In this thesis, we are investigating different integration approaches and eventually we suggest a simulated annealing framework with a tightly integrated technology mapper. The integrated annealing framework allows us to combine powerful technology independent optimization with technology dependent objectives.

Another issue with today’s logic synthesis tools identified above is the often very simplistic objective function e.g., literal count. The growing size and density of circuit integration has made the actual logic synthesis objective increasingly complex. Area minimization in the early days was soon complemented by performance considerations. Recently power constraints have become a key factor in highly integrated circuit designs. This multiple-dimensional optimization objective makes literal count increasingly less relevant for measuring circuit quality. More complicated objective functions, combining area, timing, routability, and power need to be applied throughout the entire synthesis flow to reflect the complex interactions of the design requirements and implementation constraints.

In this thesis, we present an objective function that can smoothly trade-off between area and delay optimization objectives. The explored framework can easily be extended to handle even more complex objective functions including objectives for power, placement, routability, etc.

A third problem with industrial logic synthesis systems is that, besides structural and ATPG-based simplifications [9, 10], they mostly apply algebraic restructuring methods during technology independent optimization [6] because general Boolean methods, including don’t care optimization, do not scale for large circuits. Algebraic methods are fast and robust. However, they are not complete and thus often result in implementations of lower quality. Furthermore, algebraic methods are primarily targeted at single-output functions—multiple-output functions are handled by complementary steps such as substitution, decomposition, and elimination [11].

In contrast, the advanced annealing-based logic optimization algorithm presented in this thesis provides a robust scalable logic synthesis framework. The quality of the solution is solely dependent on how much execution time you are willing to spend and not how you tune your tool and/or any combination of pre- or post-processing steps, which is common for other synthesis

tools.

1.1 Contributions

In the process of finding a logic synthesis framework providing fast, robust optimization and the ability to smoothly trade-off between different objectives, we explored four logic optimization algorithms:

1. A greedy local transformation-based method.
2. Simulated annealing at AND/INVERTER graph level.
3. A combined technology independent and technology dependent optimization technique.
4. Advanced simulated annealing with integrated technology mapping.

The greedy local transformation-based technique is using *Negation Permutation Negation* (NPN) classes of Boolean functions to generate the replacement rules. Naturally, a greedy algorithm is far from optimal and in the next phase we replaced it by a simulated annealing algorithm.

For the annealing-based method at AND/INVERTER graph level, we describe a framework for Boolean circuit optimization that utilizes a set of single-step moves on a simple technology-independent AND/INVERTER graph. We prove that, for the presented move set and selection distribution, detailed balance is satisfied and thus the annealing process asymptotically converges to an optimal solution. This property of the annealing algorithm is called the convergence property. For the advanced annealing-based technique, we present an approach to logic optimization using dynamic weights for the selection distribution of the move set. The weights are updated in regular intervals by technology mapping of the AND/INVERTER graph. The dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels. Furthermore, for all presented optimization methods we describe a detailed set of experiments that demonstrate the power and flexibility of the presented approaches to logic synthesis. We show that the final advanced simulated annealing approach can smoothly trade-off complex, multiple-dimensional objective functions and achieve competitive results.

The final contribution of this thesis, is the demonstration that our developed logic synthesis framework can be applied to two emerging technologies,

chemically assembled nanotechnology and *molecule cascades*. This is an interesting contribution because, a number of physical and economic factors threaten the continued scaling of CMOS devices, motivating research into computing systems based on other technologies. Alternatives to silicon VLSI have been proposed, including techniques based on molecular electronics, quantum mechanics, and biological processes. Hence, our research in how to apply our developed logic synthesis framework to two of the emerging technologies might provide useful information for other designers moving in this direction.

Complete list of publications

- **NPN-based optimization** Paper [12] presents the greedy local transformation based optimization algorithm discussed in Chapter 5. The local transformations are generated using NPN classes of Boolean functions. The work in this paper was conducted by the author of this thesis and supported by feedback from Elena Dubrova.
- **Simulated annealing** In [13, 14, 15], a simulated annealing-based framework for logic synthesis is introduced. We also present our technology mapper and the integration between technology independent optimization and technology dependent optimization. The theory of our annealing-based algorithm is explained and the convergence property is proved. The work in these papers was conducted by the author of this thesis, with important feedback regarding the theory from Elena Dubrova and regarding implementation details of both the annealing algorithm as well as the technology mapper from Andreas Kuehlmann.
- **Emerging technologies** The application of the developed logic synthesis frameworks to two emerging technologies, chemically assembled nanotechnology and molecule cascades, is described in [16, 17, 18, 19]. The work in these papers was conducted by the author of this thesis with feedback from Elena Dubrova.
- **Conjunctive decomposition** In [20, 21], a generalization of McMillan’s conjunctive decomposition of Boolean single-output functions [22] to the case of Boolean multiple-output functions was presented. The work presented in these papers, is not contained in this thesis. The majority of it was conducted by Elena Dubrova. The author of this

thesis contributed with examples, proof-reading and applications to multirate signal processing.

1.2 Thesis layout

This work contains the study of different approaches to multiple-level logic synthesis. In Chapter 2, we introduce the essential notation, and present fundamental mathematical concepts, such as the theory of Markov chains. The chapter also discusses graphs in general, and Boolean networks and binary-decision diagrams in specific. A basic understanding of graphs is vital, since the synthesis framework we propose is graph-based.

In Chapter 3, the general process of logic synthesis is described. The three phases of conventional logic synthesis

- technology independent optimization,
- technology mapping and
- technology dependent optimization

are described in details. Furthermore, some of the available synthesis tools are presented, including UC Berkeley's synthesis tool SIS. Later in the thesis, our experimental results show how our optimization algorithms compare to SIS. The industry-standard OpenAccess (OA) database provides a common electronic design automation infrastructure for physical design tools. Chapter 3 includes a brief introduction to OpenAccess and OA Gear. OA Gear extends the utility of OpenAccess with a set of common tools and applications. Our advanced annealing algorithm, presented in Chapter 9, is implemented in the OpenAccess framework taking advantage of the capabilities provided by both OpenAccess and OA Gear.

In Chapter 4, the AND/INVERTER graph representation is introduced. AND/INVERTER graphs have a number of advantages. First, by restricting the network vertices to two simple functions, i.e. two-input ANDs and inverters, much faster analysis can be achieved. Second, AND/INVERTER graphs permit quick recognition of isomorphic parts using structural hashing which allows for efficient storage. All our developed optimization frameworks are based on the AND/INVERTER graph representation.

A greedy local transformation-based method for optimizing multiple-level logic circuits with respect to area, using number of vertices in an

AND/INVERTER graph as objective is introduced in Chapter 5. The local transformations are generated using NPN classes of Boolean functions, which implies that more of the interesting cases are taken into account than for local transformation-based methods that define the set of eligible transformations directly in the graph domain.

A literature review of simulated annealing is presented in Chapter 6. The underlying mathematical model is discussed in detail and various annealing schedules are introduced. Furthermore, the requirement for convergence of the simulated annealing algorithm is stated. If an instance of simulated annealing has the convergence property, it implies that given enough execution time the algorithm will return the optimal solution according to the given objective function.

In Chapter 7, the greedy optimization algorithm from Chapter 5 is replaced by an annealing-based framework, performing technology independent optimization on an AND/INVERTER graph. The introduced objective function can target minimizing number of vertices or number of levels in the graph. A drawback with simulated annealing at AND/INVERTER graph level is that the graph characteristics, number of vertices and number of levels, used to optimize the given design do not properly reflect the actual area and delay in the resulting mapped circuit. A first attempt to overcome this problem is described in Chapter 8.

The algorithm described in Chapter 8 combines technology dependent optimization objectives with technology independent optimization. An objective function that can smoothly trade-off between number of vertices minimization and number of levels minimization is introduced. The algorithm will return the best mapped circuit found according to the objective function.

In the final and most mature logic synthesis framework, presented in this thesis, we use an advanced simulated annealing-based algorithm. This framework is presented in Chapter 9. The optimization algorithm is tightly integrated with our own technology mapper, providing powerful feedback through dynamic weights. The weights are updated on regular intervals by mapping the AND/INVERTER graph onto a selected library of gates. The dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels. Furthermore, we show that the presented advanced simulated annealing-based approach can smoothly trade-off complex, multiple-dimensional objective functions and achieve results competitive with UC

Berkeley’s publicly available synthesis tool, SIS.

In Chapter 10, we show how our developed logic synthesis frameworks can be applied to two emerging technologies: chemically assembled nanotechnology and molecule cascades. The motivation for the work presented in this chapter, is that the exponential improvement in speed and integration of silicon transistor technology is expected to slow down as devices approach nanometer dimensions. Alternatives to silicon VLSI have been proposed, including the two techniques described in this chapter.

Many nanotechnology programs are initiated by groups with the expertise in chemistry and physics. This contrasts very limited activities at design level. With the work presented here, we want to give ideas on how logic synthesis tools for chemically assembled nanotechnology and molecule cascades technologies could be constructed.

The last chapter, Chapter 11, gives a conclusion of the work presented in this thesis. In addition, some open problems and possible extensions of the presented advanced annealing-based logic synthesis framework are discussed.

Chapter 2

Background

This chapter provides the notation, definitions and theoretical foundations needed throughout the thesis. Special attention is given to graphs, since all our proposed optimization algorithms are graph-based. Furthermore, a special case of Markov chains required for simulated annealing is discussed in detail.

2.1 Notation

A *set* is a collection of *elements*. We denote sets by uppercase letters and elements by lowercase ones. The *cardinality* of a set is the number of its elements, denoted by $||$. A *cover* of a set S is a set of subsets of S whose union is S . A *partition* of S is a cover by disjoint subsets. Set membership of an element is denoted by $\in$, set inclusion by $\subset$ or $\subseteq$, set union by $\cup$ and intersection by $\cap$. The symbol $\forall$ is the *universal quantifier*; the symbol $\exists$ the *existential quantifier*. Implication is denoted by $\Rightarrow$ and co-implication by $\Leftrightarrow$. The symbol ":" means "such that".

The set of real numbers is denoted by $\mathfrak{R}$. The set of binary values $0, 1$ is denoted by B . Vectors and matrices are ordered sets. They are denoted by lower- and uppercase bold characters, respectively. For example, $\mathbf{x}$ denotes a vector and $\mathbf{A}$ a matrix. We denote a vector with all 0 entries by $\mathbf{0}$ and one with all 1 entries by $\mathbf{1}$.

The *Cartesian product* of two sets X and Y , denoted by $X \times Y$, is the set of all pairs (x, y) , such that $x \in X$ and $y \in Y$. A *relation* R between two sets X and Y is a subset of $X \times Y$. An *equivalence* relation is *reflexive*,

$(x, x) \in R$, *symmetric*, $(x, y) \in R \Rightarrow (y, x) \in R$, and *transitive*, $(x, y) \in R$ and $(y, z) \in R \Rightarrow (x, z) \in R$. A *partial order* is a relation that is reflexive and anti-symmetric, $(x, y) \in R$ and $(y, x) \in R \Rightarrow x = y$ and transitive. A *partially ordered set* is the combination of a set and a partial order relation on the set. A *totally* (or *linearly*) *ordered set* is a partially ordered set with the property that any pair of elements can be ordered such that they are members of the relation.

A *function* (or *map*) between two sets X and Y is a relation having the property that each element of X appears as the first element in one and only one pair of the relation. A function between two sets X and Y is denoted $f : X \rightarrow Y$. The sets X and Y are called the *domain* and *co-domain* of the function, respectively. The function f assigns to every element $x \in X$ a unique element $f(x) \in Y$. The set $f(X) = \{f(x) : x \in X\}$ is called the *range* of the function. A function is *onto* or *surjective* if the range is equal to the co-domain. A function is *one-to-one* or *injective* if each element of its range has a unique element of the domain that maps to it, i.e. $f(x_1) = f(x_2)$ implies $x_1 = x_2$. In this case, the function has an *inverse*, $f^{-1} : f(X) \rightarrow X$. A function is *bijective* if its both surjective and injective. Given a function $f : X \rightarrow Y$ and a subset of its domain $A \subseteq X$, the *image* of A under f is $f(A) = \{f(x) : x \in A\}$. Conversely, given a function $f : X \rightarrow Y$ and a subset of its co-domain $A \subseteq Y$, the *inverse image* of A under f is $f^{-1}(A) = \{x \in X : f(x) \in A\}$.

2.2 Boolean algebra

An *algebraic system* is the combination of a set and one or more operations. A *Boolean algebra* is defined by the set $B \supseteq \mathbf{B} \equiv \{0, 1\}$ and by two operations, denoted by $+$ and $\cdot$ which satisfy the *commutative* and *distributive* laws and whose identity elements are 0 and 1 respectively. In addition, any element $x \in B$ has a complement, denoted by $\bar{x}$, such that $x + \bar{x} = 1$ and $x \cdot \bar{x} = 0$. These axioms, which define a Boolean algebra, are often referred to as Huntington's postulates [23]. The major properties of a Boolean algebra can be derived from Huntington's postulates and are shown in Table 2.1. *De Morgan's law* is introduced in [24].

There are many examples of Boolean algebraic systems, for example set theory, propositional calculus and arithmetic Boolean algebra [25]. We consider in this thesis only the binary Boolean algebra, where $B = \mathbf{B} \equiv \{0, 1\}$ and the operations $+$ and $\cdot$ are the *disjunction* and *conjunction*, respectively,

$x + (y + z) = (x + y) + z$	Associativity
$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	Associativity
$x + x = x$	Idempotence
$x \cdot x = x$	Idempotence
$x + (x \cdot y) = x$	Absorption
$x \cdot (x + y) = x$	Absorption
$\overline{(x + y)} = \bar{x} \cdot \bar{y}$	De Morgan
$\overline{(x \cdot y)} = \bar{x} + \bar{y}$	De Morgan
$\overline{(\bar{x})} = x$	Involution

Table 2.1: Some properties of Boolean algebraic systems.

often called *sum* and *product* or OR and AND. The multiple-dimensional space spanned by n binary-valued Boolean variables is denoted by B^n . It is often referred to as the n -dimensional *cube*, because it can be graphically represented as a hypercube. A point in B^n is represented by a binary-valued vector of dimension n . The cube for the three-dimensional Boolean space is shown in Figure 2.1. The point 111 corresponds to xyz , the point 110 to $xy\bar{z}$, the point 101 to $x\bar{y}z$, etc.

Figure 2.1: The three-dimensional Boolean space.

Boolean functions

Let $f(x_1, x_2, \dots, x_n)$ be a completely specified multiple-output Boolean function of type $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$.

A point in the domain $\{0, 1\}^n$ of f is called *minterm*. The *on-set* F_f and the *off-set* R_f of f are the sets of minterms that are mapped by f to 1 and 0, correspondingly.

A *literal* is a variable x_i or its complement $\bar{x}_i$, $i \in \{1, \dots, n\}$. A *product-term* is a Boolean product (AND) of one or more literals. A *sum-of-products* is a Boolean sum (OR) of product-terms. Vice versa, a *sum-term* is a Boolean sum of literals, and a *product-of-sum* is a Boolean product of sum-terms.

The *cofactor* of f with respect to a variable x_i is defined as $f|_{x_i=j} = f(x_1, \dots, x_{i-1}, j, x_{i+1}, \dots, x_n)$, $j = \{0, 1\}$, $i \in \{1, \dots, n\}$. A function f is called *unate* in x_i , if either $f|_{x_i=1} \supseteq f|_{x_i=0}$ or $f|_{x_i=1} \subseteq f|_{x_i=0}$. Otherwise, f is *binate* in x_i [26].

A function f is *incompletely specified* if $f = (f_f, f_d, f_r) : B^n \rightarrow \{0, 1, *\}$, where "*" represents *don't cares* [27]. f_f is the on-set function, $f_f(x) = 1 \leftrightarrow f(x) = 1$. f_r is the off-set function, $f_r(x) = 1 \leftrightarrow f(x) = 0$. f_d is the don't care function, $f_d(x) = 1 \leftrightarrow f(x) = *$. (f_f, f_d, f_r) forms a partition of B^n , i.e.

- $f_f + f_d + f_r = B^n$
- $f_f f_d = f_f f_r = f_d f_r = \emptyset$ (pairwise disjoint)

2.3 Programmable logic array

A *programmable logic array* (PLA) consists in principal of a large AND/OR network. The network has several inputs leading to a cluster of AND gates and a number of outputs from OR gates. The cluster of AND gates is implemented as a programmable matrix of diodes. Every input to the circuit can be connected to any AND gate in the matrix. When programming the PLA circuit these connections are created or terminated.

In Figure 2.2, an example of a PLA implementing the multiple output function,

- $f_1 = x\bar{y} + yz$
- $f_2 = \bar{x}\bar{z} + xyz$
- $f_3 = xyz + \bar{x}\bar{y}$

is shown. A horizontal black rectangle symbolizes a connection in the AND plane, while a vertical black rectangle represents a connection in the OR plane.

Figure 2.2: A PLA for the multiple output function $f_1 = x\bar{y} + yz$, $f_2 = \bar{x}z + xyz$ and $f_3 = xyz + x\bar{y}$.

2.4 Graphs

A *graph* $G(V, E)$ is a pair (V, E) , where V is a set and E is a binary relation on V [28, 29, 30]. The elements of the set V are called *vertices* and those of E are called *edges* of the graph. In a *directed graph* the edges are ordered pairs of vertices; in an *undirected graph* the edges are unordered pairs. In Figure 2.3(a) an example of an undirected graph is displayed. An example of a directed graph is shown in Figure 2.3(b). A directed edge from vertex $v_i \in V$ to $v_j \in V$ is denoted by (v_i, v_j) and an undirected edge with the same end-points by $\{v_i, v_j\}$. We also say that an edge (directed or undirected) is *incident* to a vertex when the vertex is one of its end-points. The *degree* of a vertex is the number of edges incident to it.

Figure 2.3: a) Undirected graph. b) Directed graph.

We say that a vertex is *adjacent* to another vertex when there is an edge incident to both of them. A *walk* is an alternating sequence of vertices and edges. A *trail* is a walk with distinct edges, and a *path* is a trail with distinct vertices. A *cycle* is a closed walk (i.e., such that the two end-point vertices coincide) with distinct vertices. A graph is *connected* or *strongly connected* if all vertex pairs are joined by a path. A graph with no cycles is called an *acyclic* graph or a *forest*. A *tree* is a connected acyclic graph. A *rooted tree* is a tree with a distinguished vertex, called *root*. Vertices of a tree are also called *nodes*. In addition, they are called *leaves* when they are adjacent to only one vertex each and they are distinguished from the root.

A *subgraph* of a graph $G(V, E)$ is a graph whose vertex and edge sets are contained in the vertex and edge sets, respectively of $G(V, E)$. Two graphs are said to be *isomorphic* if there is a one-to-one correspondence between their vertex sets that preserves adjacency.

Vertex cover

A *vertex cover* of an undirected graph $G(V, E)$ is a subset of the vertices such that each edge in E has at least one end-point in that subset. The vertex covering decision problem is to determine if a given graph $G(V, E)$ has a vertex problem of cardinality smaller than (or equal to) a given integer. The corresponding optimization problem is the search for a vertex cover set of minimum cardinality. The vertex cover decision problem is intractable [31].

Heuristic and exact algorithms have been proposed to solve the minimum covering problem. Some heuristic algorithms can guarantee only that the cover is *minimal with respect to containment*, i.e., that no vertex in the cover is redundant and therefore no vertex can be removed while preserving the covering property. Such a cover is often termed *irredundant*.

Boolean networks

A *Boolean network* is a directed acyclic graph. Vertices having no incoming edges are called *primary input vertices* and represent the primary inputs (external inputs) of a circuit. Vertices having no outgoing edges are called *primary output vertices* and represent the primary outputs (external outputs) of a circuit. Other vertices, called *intermediate vertices*, represent the internal structure of a circuit. Each intermediate vertex is associated with a Boolean function that the vertex realizes. This function is called the *local function* of a vertex. All vertices are associated with Boolean variables. We treat a vertex and a variable associated with the vertex interchangeably. An edge from vertex v_i to vertex v_j implies that the local function of vertex v_j directly depends on v_i . There are several ways to represent a local function of a node:

- As a sum-of-products expression.
- On a factored form.
- As a simple gate (AND/OR/NAND/NOR etc.).

The output of a vertex may be an input to other vertices called its *fanouts*, or *fanout vertices*. The inputs of a vertex is called its *fanins* or *fanin vertices*. If there is a path from vertex v_i to vertex v_j , then v_i is in the *transitive fanin* of v_j and v_j is in the *transitive fanout* of v_i . v_i may be a primary input and v_j a primary output.

Binary-decision diagrams

A *binary-decision diagram* (BDD) represents a Boolean function as a rooted directed acyclic graph. It has two types of vertices: *terminal* and *non-terminal*. Terminal vertices are leaves in the graph corresponding to the 0 and 1 values of a function. They have no outgoing edges. All other nodes in the BDD are non-terminal, and they represent a Shannon expansion about

some variable x_i . Each non-terminal node can be viewed as a root node of some non-constant function f , and it has a $f_{\bar{x}_i}$ and a f_{x_i} child. If v is a non-terminal node with index i , then its f_v function is

$$f_v = \bar{x}_i f_{\bar{x}_i} + x_i f_{x_i}$$

Representing Boolean functions with a binary-decision diagram was originally proposed by Lee [32] and Akers [33]. However, it was only with the work of Bryant [34] in 1986 that BDDs became widely used. His work brought out the canonical nature of BDDs in representing Boolean functions. The work also introduced effective algorithms to manipulate them. Since then the use of BDDs has entered virtually every area of synthesis and verification.

Figure 2.4: Reduced ordered binary decision diagrams for $f = (x+y)z$ with the variable order (x, y, z) .

To represent a function with a *ordered binary-decision diagram* (OBDD) a total order is imposed on the BDD variables. Node variables on each root-to-terminal paths obey this order. A *reduced ordered binary-decision diagram* (ROBDD) is constructed using two reduction rules:

1. Nodes whose two edges point to the same child are deleted.

2. Isomorphic subgraphs are shared.

In Figure 2.4 a reduced ordered binary decision diagram for the function $f = (x + y)z$ is shown.

BDDs are unique for a given variable ordering and hence provide canonical forms for the representation of Boolean functions. This canonicity makes them well suited for symbolic manipulation. They are also useful for representing large combinatorial sets. A comprehensive treatment of BDDs can be found in [35].

Figure 2.5: Binary decision diagrams for $f = (x + y)z$: a) OBDD for the variable order (x, y, z) . b) OBDD for the variable order (x, z, y) .

Variable ordering is known to have a dramatic impact on the size of the BDD. In Figure 2.5 two ordered binary decision diagrams representing the function $f = (x + y)z$ for two different variable orders are shown. As seen the variable order in Figure 2.5(a) requires one node more than the variable order in Figure 2.5(b). Unfortunately, there is no known method which can quickly detect an optimal variable ordering. Most of the variable ordering techniques rely on heuristics. The earlier work on this subject couples variable ordering techniques with topological information from the circuit for which the BDD

has been constructed [36, 37, 38]. A later heuristic, based on variable sifting, was developed by Rudell [39]. In the sifting algorithm, each variable is moved up and down to greedily find its best location. Since its introduction the algorithm has been integrated into many BDD packages and applications in various flavors [40, 41, 42, 43].

The reordering techniques of any typical package are applied once by an explicit function call, or they can be evoked dynamically [39] by an implicit function call triggered by some memory consumption criteria. In many applications such asynchronous re-ordering has a profound effect on the resources consumed during the manipulation of BDDs. However, the dynamic re-ordering of variables still remains an expensive operation, and may take a significant part of a computation. Applications which are aware of variable orders implied by the intrinsic nature of the problem are therefore the most desired approach to reducing BDD sizes. For some functions however, the size of a BDD may be exponential in the numbers of inputs regardless of the variable ordering. An n -bit multiplier is an example of such a function [44].

Once the BDD for a function is constructed many operations on it have good characteristics. For example, taking function complement, or checking if the function is satisfiable can be done in constant time. In general the space and time requirements for the binary operations are proportional to the number of nodes in the two composed BDDs. Deciding if two functions are equivalent requires a graph isomorphism check, whose time complexity for the labeled *Directed Acyclic Graph* (DAG) is linear in the number of nodes. The check is even more efficient when the two given functions depend on the same set of variables. If two such functions are equivalent, a typical BDD package implementation would ensure that they reside in the same memory space. This is achieved by virtue of a *unique table* [45], which guarantees that at any time there are no isomorphic subgraphs. Thus the equivalence check takes constant time. The unique table also allows a single *multiple-rooted* DAG to represent all created functions. To reduce memory consumption, modern BDD packages also attempt to share not only isomorphic subgraphs, but subgraphs of their complement functions as well. Thus subgraphs for f and $\bar{f}$ are identical.

XBDDs [46] propose to divert from the strict functional canonicity by adding function nodes to the graph. The node function is controlled by an attribute on the referencing arc and can represent an AND or OR operation. Similar to BDDs, the functional complement is expressed by a second arc attribute and structural hashing identifies isomorphic subgraphs on the fly.

Class	Representative function
1	1
2	x_1
3	$x_1 + x_2$
4	$x_1 \oplus x_2$
5	$x_1x_2 + x_2x_3 + x_1x_3$
6	$x_1 \oplus x_2 \oplus x_3$
7	$x_1 + x_2 + x_3$
8	$x_1(x_2 + x_3)$
9	$x_1(x_2 \oplus x_3)$
10	$x_1x_2 + \bar{x}_1x_3$
11	$x_1x_2 + \bar{x}_1\bar{x}_2x_3$
12	$x_1x_2x_3 + \bar{x}_1\bar{x}_2\bar{x}_3$
13	$x_1\bar{x}_2 + x_1\bar{x}_3 + \bar{x}_1x_2x_3$
14	$x_1x_2\bar{x}_3 + x_1\bar{x}_2x_3 + \bar{x}_1x_2x_3$

Table 2.2: NPN classes for three variable functions.

The proposed tautology check is similar to a technique presented in [47] and is based on recursive inspection of all cofactors. This scheme effectively checks the corresponding BDD branching structure sequentially, resulting in exponential execution time for problems for which BDDs are excessively large.

2.5 NPN classes

In *NPN classes* (**N**egation-**P**ermutation-**N**egation) of Boolean functions [48], each class consists of all functions which differ by:

- Negation of some input variables $x_1, \dots, x_n$.
- And/or permutation of some input variables $x_1, \dots, x_n$.
- And/or negation of the function output.

For functions of four variables or less, there are in total 222 NPN classes. For functions of exactly four variables, there are 208 classes. The corresponding numbers for functions of three variables are 14 and 10. In Table 2.2, all 14 NPN classes for functions of up to three variables are listed [48].

Figure 2.6: NPN representative function realization.

The practical significance of the NPN classification is that, should we have a logic network that realizes any one of the classification entries, then all functions covered by this representative function may be realized by permuting any of the input variables, and/or by negating one or more input variables and/or by negating the overall function.

In Figure 2.6 the realization of a NPN representative function is displayed [49].

2.6 Finite Markov chains

In this section, a brief outline of the theory of discrete-time finite Markov chains is given. The foundations of a theory of general state space Markov chains are described in [50], and although the theory is much more refined now, this is still the best source of much basic material. The next generation of results is developed in [51] and more current treatments are contained in [52].

Consider some finite discrete set S of possible states, labeled $s_0, s_1, s_2, \dots, s_k$. At each of the unit time points $t = 0, 1, 2, 3, \dots$, a Markov chain process occupies one of these states. In each time step t to $t + 1$, the process either stays in the same state or moves to some other state in S . Further, it does this in a probabilistic, or stochastic, way rather than in a deterministic way. That is, if at time t the process is in state s_i , then at time $t + 1$ it either stays in this state or moves to some other state s_j according to some well-defined probabilistic rule described in more detail below. This process follows the requirements of a simple Markov chain if it has the following properties;

1. The *Markov* property. If at some time t the process is in state s_i , the probability that one time unit later it is in state s_j depends only on s_i , and not on the past history of the states it was in before time t . That is, the current state is all that matters in determining the probabilities for the states that the process will occupy in the future.
2. The *temporally homogeneous transition probabilities* property. Given that at time t the process is in state s_i , the probability that one time unit later it is in state s_j is independent of t .

More general Markov processes relax one or both requirements, but we assume throughout the section that the above properties hold.

Transition probabilities and the transition probability matrix

Suppose that at time t a Markovian random variable is in state s_i . We denote the probability that at time $t+1$ it is in state s_j by p_{ij} , called the *transition probability* from s_i to s_j . By writing the transition probability in this form we are already using the two Markov assumptions described above. First, no mention is made in the notation p_{ij} of the states that the random variable was in before time t (the memoryless property), and second, t does not occur in the notation p_{ij} (the time homogeneity property).

It is convenient to group the transition probabilities p_{ij} , into the so-called *transition probability matrix*, or more simply the transition matrix, of the Markov chain. We denote this matrix by $\mathbf{P}$, and write it as

	(to s_1)	(to s_2)	(to s_3)	$\cdots$	(to s_k)
(from s_1)	p_{11}	p_{12}	p_{13}	$\cdots$	p_{1k}
(from s_2)	p_{21}	p_{22}	p_{23}	$\cdots$	p_{2k}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
(from s_k)	p_{k1}	p_{k2}	p_{k3}	$\cdots$	p_{kk}

The rows and the columns of $\mathbf{P}$ are in correspondence with the states in $s_1, s_2, s_3, \dots, s_k$, so these states being understood, $\mathbf{P}$ is usually written in the simpler form

p_{11}	p_{12}	p_{13}	$\cdots$	p_{1k}
p_{21}	p_{22}	p_{23}	$\cdots$	p_{2k}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
p_{k1}	p_{k2}	p_{k3}	$\cdots$	p_{kk}

Any row in the matrix corresponds to the state *from* which the transition is made, and any column in the matrix corresponds to the state *to* which the transition is made. Thus the probabilities in any particular row in the transition matrix must sum to one. However, the probabilities in any given column do not have to sum to anything in particular.

It is also assumed there is some *initial* probability distribution for the various states in the Markov chain. That is, it is assumed there is some probability π_i that at the initial time point the Markovian random variable is in state s_i . A particular case of such an initial distribution arises when it is known that the random variable starts in state s_i , in which case $\pi_i = 1$, $\pi_j = 0$ for $i \neq j$. In principle the initial probability distribution and the transition matrix $\mathbf{P}$ jointly determine all the properties of the entire process. In practice, many properties are not found easily, or if found obtained by special methods.

The probability that the Markov chain process moves from state s_i to state s_j after two steps can be found by matrix multiplication. It is this fact that makes much of Markov chain theory an application of linear algebra. The argument is as follows.

Let $p_{ij}^{(2)}$ be the probability that if the Markovian random variable is in state s_i at time t , then it is in state s_j at time $t+2$. We call this a *two-step* transition probability. Since the random variable must be in some state s_k at the intermediate time $t+1$, summation over all possible states at time $t+1$ gives

$$p_{ij}^{(2)} = \sum_k p_{ik} p_{kj}$$

The right-hand side in this equation is the (i, j) element in the matrix $\mathbf{P}^2$ ($\mathbf{P} \cdot \mathbf{P}$). Thus if the matrix $\mathbf{P}^{(2)}$ is defined as the matrix whose (i, j) element is $p_{ij}^{(2)}$, then the (i, j) element in $\mathbf{P}^2$ is equal to the (i, j) element in $\mathbf{P}^{(2)}$. This leads to the identity

$$\mathbf{P}^{(2)} = \mathbf{P}^2.$$

Extension of this argument to an arbitrary number n of steps gives

$$\mathbf{P}^{(n)} = \mathbf{P}^n.$$

That is, the "n-step" transition probabilities are given by the entries in the n th power $\mathbf{P}$.

Chapter 3

Logic synthesis

Logic synthesis is the process of automatically generating an optimized logic level representation from a high-level description. The complexity and significance of this task depends on the level of input specification, the type of logic implementation, and the criteria for an acceptable result. The level of specification can range from behavioral where only the relationship of outputs to inputs is given, to the register transfer level (RTL) where the state is explicitly defined, to the structural level where the specification is given as an interconnection of hardware primitives. There are also different levels of logic implementation ranging from a set of Boolean equations, to a list of interconnected technology specific hardware primitives, to detail mask data for manufacturing a chip. In Figure 3.1 the different design levels and how they are connected are displayed.

Traditional logic synthesis systems [53, 2, 4] consists of three separate phases:

- Technology independent optimization.
- Technology mapping.
- Technology dependent optimization.

Technology independent optimization is the process of trying to reduce the cost of the representation for a given logic function. Technology mapping transforms a technology independent logic network into gates implemented in a technology library. Technology dependent optimization tries to improve the circuit characteristics (such as, area, delay, power, routing congestion,

Figure 3.1: Design flow of integrated systems.

signal integrity) by utilizing a technology library to modify the mapped circuit. In Figure 3.2, a typical synthesis scenario is displayed. Apart from the three phases enumerated above two more are added:

- The first one is to build a network representation of a Verilog/VHDL description.
- The last one is to prepare the mapped circuit for testing.

In this chapter, we will discuss technology independent optimization, technology mapping and technology dependent optimization. Furthermore, we will review a selection of available logic synthesis tools, including UC Berkeley's tool SIS.

A traditional synthesis flow.

Figure 3.2: Typical synthesis scenario.

3.1 Technology independent optimization

Technology independent optimization can be divided into two categories, two-level optimization and multiple-level optimization. Multiple-level logic means more than two-levels of logic representations and corresponds to multiple-level logic circuits.

Two-level logic optimization

Two-level logic minimization consists of finding a minimum sum-of-products expression that covers a given Boolean function f . In other words,

$$f(x_1, x_2, \dots, x_n) = P_1 + P_2 + \dots + P_k$$

with the minimal number k of products P_k . A minimal sum-of-products form has at most 2^{n-1} product-terms. The concept of sum-of-products is introduced in Section 2.2.

Two-level logic optimization is used for optimizing programmable logic arrays (PLAs). PLAs are described in Section 2.3. The size of the PLA is directly proportional to the size of the corresponding sum-of-products expression:

- The number of columns in the PLA equals to the number of products in sum-of-products form.
- The number of connections per column in the PLA equals to the number of variables in the products.

The Quine-McCluskey procedure is the classic textbook method used to derive exact minimum two-level logic circuits [54]. However, it is limited to functions up to about 15 variables [11] then it gets too computer intensive and will not finish given finite time. Instead many two-level logic optimization methods rely on heuristic. One of the most famous two-level minimization tools based on heuristics is ESPRESSO [55].

Multiple-level logic optimization

Multiple-level technology independent optimization is the process of trying to reduce the cost of a representation for a given logic function. This stage operates on the technology independent network, i.e. a network in which the gates are not bound to a particular technology cell but are generic logic gates. The optimization criteria for multiple-level logic is to minimize some function of:

- Area occupied by the logic gates and interconnect (approximated by literals in technology independent optimization).
- Critical path delay.
- Degree of testability of the circuit, measured in terms of the percentage of faults covered by a specified set of test vectors for an approximate fault model (e.g. single or multiple stuck-at faults).
- Power consumption.
- Noise immunity.
- Place-ability, wire-ability.

There are two basic techniques for manipulating Boolean networks, structural operations (change topology) and node simplification (change node functions). The structural operations can be conducted with either Boolean methods or algebraic methods. Boolean methods consider logic functions as well as their representations, whereas algebraic methods only consider logic representations. In other words, algebraic methods treat logic functions as polynomial expressions, whereas Boolean methods recognize Boolean algebra rules that can transform one logic representation to another (see Section 2.2). Generally speaking, Boolean methods are much more powerful than algebraic methods but can be computationally expensive. The basic structural operations for manipulating Boolean networks are:

1. Decomposition (single function)
 $f = abc + abd + \bar{a}\bar{c}\bar{d} + \bar{b}\bar{c}\bar{d} \Rightarrow$
 $f = xy + \bar{x}\bar{y}, \text{ where } x = ab \text{ and } y = c + d$
2. Extraction (multiple functions)
 $f = (az + b\bar{z})cd + e, g = (az + b\bar{z})\bar{e}, h = cde \Rightarrow$
 $f = xy + e, g = x\bar{e}, h = ye, \text{ where } x = az + b\bar{z} \text{ and } y = cd$
3. Factoring (series-parallel decomposition)
 $f = ac + ad + bc + bd + e \Rightarrow$
 $f = (a + b)(c + d) + e$
4. Substitution
 $g = a + b, f = a + bc \Rightarrow$
 $f = g(a + b)$
5. Collapsing (also called elimination)
 $f = ga + g\bar{b}, g = c + d \Rightarrow$
 $f = ac + ad + b\bar{c}\bar{d}$

Decomposition of Boolean functions aims at finding a representative multiple-level expression with the least number of literals. A variety of algorithms for Boolean and algebraic decomposition has been developed. A work of milestone importance is [6], where the notions of *kernels* was introduced and a method for fast algebraic decomposition based on kernels was developed. This technique with minor modifications are used in [56, 57, 58, 53, 59, 3]. Techniques for conjunctive decomposition can be found in [22, 21].

3.2 Technology mapping

In its most general form, technology mapping can be formulated as a directed acyclic graph (DAG) covering problem. Both the covering problem and DAGs are described in Section 2.4. The Boolean function to be mapped is represented as a DAG (called the *subject graph*) using a chosen set of primitive cells. Similarly, cells in the library are represented as DAGs (called *patterns*) using the same primitive functions. Each pattern is associated with a *cost* based on the optimization criterion. Then, technology mapping is the problem of finding an optimal cost covering of the vertices of the subject graph using available patterns, under the constraint that the inputs to a match must be available as the outputs of other matches.

Technology mapping is the only phase in logic synthesis that transforms each and every cell in the network. It therefore has a major impact on the global structure of the technology mapped logic, and its delay and area characteristics. Conventional technology mapping can be described as a three step procedure:

1. Decomposition.
2. Pattern matching.
3. Covering.

First, the technology independent circuit is decomposed in terms of some primitive cells to have a simple logic structure to aid the technology mapping process. Second, a pattern matcher performs analysis on the circuit and the library, either structurally or functionally, and determines a set of matches for all nodes in the circuit. The third and final phase consists of identifying the best possible matches (based on particular cost functions) for the circuit such that every node in the circuit is covered at least once and functionality is maintained. The final set of matches consists of instantiations of cells in the target technology.

Technology mapping must deal with a variety of objective functions

- area,
- delay,
- power and

- reliability.

Area is commonly calculated as the sum of the area of all individual cells in the network, ignoring wiring area. Delay is the delay calculated by a static timing analysis tool. There are several different ways to model the delay of a gate. Given a single-output gate g , let $\delta(i, g)$ denote the delay from an input pin i of g to the gate output. We will use g to denote the output of g . The load c_g refers to the cumulative capacitance seen at the output of g . It is the sum of the input pin capacitance $\gamma(p)$ of all the fanout pins p of g and the loads of the wires connected to the output pin of g . The drive capacity of g , denoted λ_g , is the maximum capacitance g can drive without compromising the signal integrity or the delay specifications. The simplest delay model is the *load-independent delay model*, in which $\delta(i, g)$ is independent of the load capacitance c_g and is given by the intrinsic delay $\alpha(i, g)$:

$$\delta(i, g) = \alpha(i, g).$$

In the more realistic *load-dependent delay model*, the delay $\delta(i, g)$ depends linearly on both intrinsic delay as well as the output load c_g :

$$\delta(i, g) = \alpha(i, g) + \beta(i, g)c_g,$$

where $\beta(i, g)$ is the load coefficient of the path from i to g . In the *input-slew dependent delay model*, $\delta(i, g)$ depends not only on the output load c_g , but also on the slew (transition time) τ at the input pin i . The slew τ , in turn, depends on the gate h that is driving the pin i as well as on the total capacitive load seen by h :

$$\tau(h) = \phi(h) + \rho(h)c_h.$$

Hence, the delay δ can be expressed as follows:

$$\delta(i, g) = \alpha(i, g) + \beta(i, g)c_g + \kappa(i, g)\tau(h) + \nu(i, g)c_g\tau(h).$$

The gate-dependent delay coefficients $\phi, \rho, \alpha, \beta, \kappa$ and ν are determined by first gathering delay and output slew values for a range of input slew and output load values (e.g. using a circuit simulator such as SPICE) and then fitting the aforementioned linear and bi-linear forms on this data (e.g. by the method of least squares).

Power and reliability calculations require additional data on the switching activity of all nodes.

Technology mapping needs to be able to deal with different objectives for different parts of the circuit:

- Fast technology cells must be used along critical paths.
- Non-critical regions may use smaller, slower, and less power-consuming cells.

Keutzer [60] proposes a mapping method based on a tree-covering method approximation. In this approach, the subject graph is partitioned into a forest of trees, and the covering problem is solved for each of the trees. This approach is motivated by the existence of efficient dynamic programming algorithms for optimum tree covering [61]. While there is no unique way of partitioning the subject graph into trees, usually the simplest solution of breaking the graph at each multiple fanout point is employed.

The covering phase of technology mapping works on a fixed subject graph. Therefore the result is largely depending on the preceding decomposition of the network. To get around this problem Lehman [62] combines decomposition with the covering itself. Despite its efficient implementation, exhaustive embedding of the decompositions lead to a significant increase of the size of the subject graph, not practical for sizes of circuits one is typically interested in.

Wavefront mapping [63] enables the embedding of multiple decompositions on-the-fly. This so-called dynamic decomposition significantly reduces the number of decompositions needed.

3.3 Technology dependent optimization

Technology dependent optimization can improve the circuit characteristics (such as, area, delay, power, routing congestion, and signal integrity) before, during, or after the layout. Such post-mapping transformations utilize a technology library to modify the mapped circuit.

Exact technology mapping is an intractable problem under most practical scenarios:

- Unconstrained area minimization [60].
- Unconstrained delay minimization under load-dependent delay model [64].
- Area-constrained delay minimization even under load-independent delay model [65].

This is due to the inherent difficult nature of the problem, the delay model, the complex interactions between the gate being mapped and the unmapped portion of the logic, and absence of layout information. There are several techniques used for technology dependent optimization. We will describe some of the most popular methods [11].

For *gate resizing* we are given a mapped circuit composed of gates from a gate library. For each gate g , several different sizes $1, \dots, k$ are available in the library, each size having the identical logic function but different area, input load γ , drive capacity λ , and delay coefficients such as α and β . The gate resizing problem is to select the size of each gate such that some objective function (e.g., circuit delay, circuit area) is minimized or reduced without violating any constraints. Gate resizing is an in-place optimization technique. It has minimal impact on the placement and routing of cells, and can be applied when more accurate wire load and delay estimates become available (e.g. during or after placement, or after placement and routing).

Fanout optimization or *buffer insertion* seeks to optimally distribute a signal from the driver gate (source, root) to the fanout gates (sinks) using buffers, without violating the drive capacity of the driver or those of the inserted buffers. The objective could be a combination of the following:

- Minimize delay through the critical parts of the net.
- Fix the overloading problem at the source of a large fanout net incurring minimum delay and area penalties.
- Reduce the slew (slope degradation) of the signal with minimum delay and area penalties.

For instance, buffering can reduce the arrival time at the output of a critical gate g by reducing the total capacitive load driven by g . The added delay due to inserted buffers is along the non-critical paths.

Gate replication, like buffer insertion, is a technique for speeding up a design by redistributing the fanout load. The idea is to make k copies $g_1, g_2, \dots, g_k$ of a gate g , partition the fanout gates of g among the k copies and replace g with g_1 through g_k . Each copy drives less capacitive load than g , potentially reducing the circuit delay. This speed-up, however, comes at a cost. Replication results in an area penalty. Also, the fanin gates of g drive a larger load now and are slowed down. The speed-up of g has to be compared against the slowing down of the fanin gates of g , before allowing

replication. Gate replication therefore typically uses only small values of k - either 2 or 3.

Simple gate decomposition transforms identify delay-critical multiple-input simple gates in the design and considers decomposing them into two or more gates to reduce the delay. A gate is simple if it is either an AND, OR, NAND, XOR or XNOR gate. For example, a 4-input AND gate g can be decomposed into a two-input AND gate g_1 followed by a 3-input AND gate g_2 , assuming both g_1 and g_2 are in the gate library. An advantage of using simple decomposition is that the circuit remains mapped after the transform and a general-purpose mapper is not needed.

In [66] heuristics procedures for performance optimization of mapped combinational logic, implemented in the system LATTIS (Logic Area-Time Trade-off for Integrated Systems) is described. LATTIS has six transform types:

1. Gate repowering.
2. Buffer insertion.
3. Down-powering of non-critical fanouts of the critical path.
4. Gate duplication.
5. De Morgan's laws.
6. Timing-directed factorization and remapping of subcircuits.

From among the transformations applicable on the critical path, LATTIS chooses the one with maximum benefit/cost ratio. Cost is increase in area, and benefit is improvement in local slack (the notion of slack is introduced in Section 4.1), weighted by the number of primary inputs and outputs that are affected.

3.4 Synthesis tools

Most synthesis tools in use today are based on the premise that the search for optimal solutions is intractable, and are designed, instead, to find acceptable sub-optimal realizations. These tools typically operate on a multiple-level representation of the functions being synthesized, continually transforming it until a satisfactory solution is found, and can be roughly classified into

two broad categories based on the granularity of transformations used. *Local* transformation methods modify the current "solution" incrementally by making appropriate changes in its immediate neighborhood. In contrast, *global* transformation methods seek good multiple-level topologies by making large-scale changes to the implementation structure while disregarding technological considerations; a second "mapping" phase insures compliance of the resulting multiple-level structure with technology constraints.

Local transformation methods

Original local optimization methods perform rule-based transformations, which are a set of *ad hoc* rules that are applied iteratively to patterns found in the network of logic gates. In the local optimization method each rule introduces a transformation by replacing a small subgraph of several gates in the network with another subgraph which is functionally equivalent but has a simpler realization according to some cost function. Initially the network consists of AND, OR, INV gates; decoders, multiplexer, adders, etc. After the simplification step these primitives are translated into an interconnection of INV to NAND gates through a sequence of transformations. Technology specific transformations are then applied as a final step in the process. Such transformations have limited optimization capability since they are local in nature, and do not have a global view on the design. Examples of systems based on this approach are LSS [2] and LORES/EX [67].

IBM's LSS (Logic Synthesis System) was the first rule-based system developed in the 1970's. LSS is an interactive system in which the user operates on a logic design at three levels of abstraction. LSS makes use of a graph-like internal representation and all transformations are performed on this graph. At every level, the implementation is a network of boxes connected by signals. The three levels of abstractions are;

- The initial AND/OR/NOT level.
- A NAND or NOR level.
- A hardware level in which the types of the boxes are books or primitives of the target technology.

The local simplifying transforms are applied repeatedly throughout the network until no more are found. Each transformation has an associated condi-

tion that determines whether the replacement simplifies the implementation by reducing boxes or connections, or not.

In later years interest in local transformations has shifted to a more rigorous theory of don't cares (see Section 2.2). They arise from the structural and external properties of a network, and are used to describe flexibility required to locally optimize a network node. In general, the use of don't cares is a difficult task since their number may become unmanageably large even for a small neighborhood of a node. Furthermore, it is not always clear from which part of a circuit the don't cares should be extracted. Various algorithms have been proposed to extract subsets of don't cares (see for example [68, 69]). Optimization with don't cares has a close relation to other optimization methods, such as transduction [70], redundancy removal [10] and global flow analysis [71, 72]. Techniques based on these methods have been successfully implemented in BooleDozer [73], a synthesis tool from IBM.

Global transformation methods

The computational limitations of the classical theory for functional decomposition motivated the development of algorithms which are effective in partitioning complex logic functions. These ideas are based on the notion of algebraic factorization applied to sum-of-products (SOP) expressions; the technique is described in [6] and [53]. For the definition of sum-of-products form, see Section 2.2. Algebraic decomposition techniques have experienced the most success to date in the field of multiple-level synthesis. They are capable of handling large combinational blocks, and produce very good results for control logic. However, representing logic of higher level abstraction with SOP forms makes it difficult to explore the structural flexibility of the original description. It can lead to the loss of a compact description of the original equations, and algebraic decomposition is too restrictive to rediscover their structure. Examples of systems which rely on algebraic techniques are MIS [53], SOCRATES [74], and SIS [3]. In more recent years much attention has also been given to AND-XOR decompositions [75, 76, 40, 77, 78].

SOCRATES is a system of programs which synthesize and optimize combinational logic circuits from Boolean equations. SOCRATES incorporates logic and circuit level manipulation and optimization techniques. Two programs, ESPRESSO [5, 79] and *weak division* [6], perform logic level manipulation on the design while the *circuit optimizer* program manipulates at the

circuit level. The circuit optimizer is a rule-based program, which improves measurable circuit characteristics by iteratively replacing and rearranging small portions of a circuit. The program uses a library which describes alternative circuit implementations in a rule form. Before a rule is selected, transformations on the circuit resulting from its application are tried. The program evaluates which effect a transformation has on other future transformations by performing a state search. The depth and breadth of the search tree determine how far, and how exhaustively the program looks forward before selecting a new rule. This look-ahead mechanism enables the circuit optimizer to choose transformations which do not immediately improve the circuit, but leads to other transformations which do. A meta-level rule-based expert system decides where to look for the next rule. It also sets the depth and breadth of the state search which is performed before choosing an alternative.

SIS is a system for sequential and combinational circuit synthesis. It consists of a number of various programs and algorithms for synthesis and optimization of logic circuits. It includes, but is not limited to, algorithms for

- technology independent optimization, e.g.
 - constant propagation,
 - redundancy removal,
 - common subexpression sharing,
 - Boolean minimization,
- technology dependent optimization,
- technology mapping, and
- timing analysis.

Optimization is done for area, performance, and testability. Scripts are used to arrange a set of algorithms towards a common synthesis objective. The two most popular scripts are *script algebraic* (shown in Figure 3.3) and *script rugged* (shown in Figure 3.4). The former script is mainly based on algebraic optimization steps and does not use more powerful Boolean optimization. The latter script applies aggressive Boolean optimization exploiting don't cares. In contrast to algebraic methods which scale well for large circuits,

eliminate	eliminate nodes whose number of literals is less than or equal to the given value
full_simplify	simplify the two level function at each node using don't cares
fx	extract common single-cube and double-cube divisors
gkx	extract multiple-cube common divisors
resub	resubstitute each node in the node-list into all nodes
simplify -nocomp	simplify with ESPRESSO
sweep	eliminate all single-input nodes and constant nodes

Table 3.1: A subset of the available commands in SIS.

script rugged is computationally expensive and therefore it is not applicable for larger circuits. A subset of the commands available in SIS is displayed in Table 3.1. The command *decomp - g** translates to *eliminate - 1*.

SIS reads in combinational or sequential circuits on *Berkeley Logic Interchange Format* (BLIF) [3]. The goal of BLIF is to describe a logic-level hierarchical circuit in textual form.

The technology mapper in SIS uses a tree-covering algorithm to map arbitrary complex logic gates into gates specified in a technology library. The technology library has to be given on a *genlib* format [3]. The tree-covering is done by decomposing the logic to be mapped into a network of two-input NAND gates and inverters. This network is then covered by patterns that represent the possible choices of gates in the library. During the covering stage, the area or the delay of the circuit is used as an optimization criterion.

The advent of binary decision diagrams (BDDs), presented in Section 2.4, and their variants rekindled interest in classical decomposition techniques. In recent years researchers have successfully applied Roth and Karp decomposition in FPGA (Field Programmable Gate Array) synthesis [80, 81, 82, 83, 84]. These approaches decompose a function recursively until each of the generated subfunctions meets a given fanin constraint. However, since fanin count is the only notion of node complexity in these approaches, they do not extend easily to a library specific synthesis. A number of approaches have also been developed which explore the structure of the decision diagram representation of a given function [85, 77, 86, 87]. The close relation between BDDs and multiplexer circuits has also lead to several approaches to synthesis of pass transistor logic (PTL) [88, 89, 76, 90]. They are primarily based on a mapping of (decomposed) BDDs to PTL.


```

sweep
eliminate 5
simplify -m nocomp -d
resub -a

gkx -abt 30
resub -a; sweep
gcx -bt 30
resub -a; sweep

gkx -abt 10
resub -a; sweep
gcx -bt 10
resub -a; sweep

gkx -ab
resub -a; sweep
gcx -b
resub -a; sweep

eliminate 0
decomp -g *
```

Figure 3.3: SIS *script algebraic* which is mainly based on algebraic optimization steps.

3.5 OpenAccess

The physical design research community is highly fragmented [91]. Individual academic tool developers tend to implement their own infrastructure, distinct from other people's work. For example, four leading-edge academic placement tools (Capo [92, 93], Warp1 [94], Dragon [95], and Feng Shui [96]) are each implemented on their own design database.

This deep fragmentation is the root of several problems in physical design research today. First, building the infrastructure requires a significant effort from already busy researchers. At the least, each individual design database

```
sweep; eliminate -1
simplify -m nocomp
eliminate -1

sweep; eliminate 5
simplify -m nocomp
resub -a

fx
resub -a; sweep

eliminate -1; sweep
full_simplify -m nocomp
```

Figure 3.4: SIS *script rugged* which implements Boolean optimization.

requires the development of parsers and translators for processing standard file formats to that database. Second, without common infrastructure, it becomes difficult to integrate tools into larger flows, or to extend tools with additional functionality. As an example, it is difficult to tightly couple a timing engine with a placement tool to create a timing-driven placement experiment, if the timer and placer do not share a common design database. Finally, the lack of common infrastructure makes comparison of results between different tools problematic. Both [97] and [98] discuss this particular issue in detail.

The industry-standard OpenAccess (OA) database was developed to provide a common electronic design automation infrastructure for physical design tools [99, 100]. While originally indented for adoption within industry, the release of OA as a free open source makes it an ideal candidate for academic use. Moreover, adoption of a common database in the physical design community yields benefits for everyone, minimizing problems with fragmentation, easy result comparison, and the perennial problem of incompatible benchmark formats.

The OpenAccess concept

The OpenAccess approach to integrated circuit (IC) design was created to:

- Eliminate or reduce translation steps in the electronic design automation flow.
- Eliminate data loss between logic synthesis tools.
- Standardize representation semantics, thus avoiding conflicts in data representation and misinterpretation.
- Centralize name mapping support.
- Centralize accessibility of all design data, assuring completeness in the stored information.
- Enable high-performance tight loops among tools such as placement, timing, extraction, etc.
- Realize "plug'n'play" of tools from different vendors in a flow.

OpenAccess is a collaborative effort by companies in the IC design chain. The effort is built on three pillars (illustrated in Figure 3.5):

1. An openly documented API standard.
2. A reference implementation of that standard.
3. A detailed OpenEvolution process to manage on-going development, distribution, and support of the API standard and reference implementation.

The OpenAccess standard consists of several components that defines the interface. It has an information model defined by a collection of entity relationship diagrams. This model describes a conceptual perspective of the objects and their relationships, but also includes some specification level detail, such as navigability across object relationships. The OpenAccess standard also consists of a data model, defined by C++ header files, which specify class and function interface details.

Furthermore, the OpenAccess standard has an API specification, which presents the header information in a more readable format and includes additional constraints. This specification is automatically generated by the public *Doxygen* package [101] from the actual C++ headers. Class inheritance

Figure 3.5: OpenAccess standard and reference implementation.

trees, functions and their arguments, enumerated lists, and other basic syntax information, are augmented by constraint information contained in specially formatted comments, automatically producing a convenient, HTML documentation web that is extensively hyperlinked. This technique ensures centralization of the specification details directly tied to the actual implementation code.

OpenAccess Gear

The aim for OpenAccess Gear (OA Gear) [91] is to provide a supporting environment of software components with higher levels of functionality, thus realizing

- industrial-strength analysis,
- easy integration and

- visualization.

OA Gear is an open source library that extends the utility of the OA database with a set of common tools and applications. Currently OA Gear consists of five components:

- A static timing analyzer (OA Gear Timer).
- User interfaces (OA Gear Bazaar).
- Benchmarks (free and restricted cases).
- A standard cell placer (Capo [92, 93]).
- A platform for functional representation, synthesis and verification (OA Gear Functionality).

OA Gear Timer

OA Gear provides a static timing analysis tool called *OA Gear Timer*. A few of the key features of the OA Gear Timer are:

- Support for industry-standard timing libraries.
- Support for constraint file formats.
- Extensible wire delay modeling.
- Incremental timing analysis capabilities.

The OA Gear Timer follows generally accepted standard techniques for static timing analysis. Arrival and required arrival times and signal slew rates are maintained for all nodes in the circuit. Separate timing figures are kept for rising and falling signals. Internal gate delays utilize the standard interpolated two-dimensional lookup table based on output load and input signal slew rate. The OA Gear Timer has two modes of operation. *Full* timing analysis computes and stores the arrival and required arrival times and slew rates for *all* nodes in the design. Timing queries simply return the stored values for these figures. Under full analysis, if the netlist or delays are changed, the timing for all nodes is fully recomputed. In contrast, *incremental* timing uses *lazy evaluation*, and only computes timing for a

minimal subset of nodes of the design in order to satisfy any timing queries made.

Both the ability to model delays due to wires and estimation of capacitive wire loading on drivers of nets are critical for timing-driven physical design. Currently there are two wire delay/load models in OA Gear Timer. One simply ignores wire delays, and the other estimates delay and load using a half-perimeter bounding-box as an estimate of routed wire length. More sophisticated wire delay models require integration with the OpenAccess database and can be defined by users through a function callback mechanism. Such user-defined models are then automatically invoked during timing analysis. This flexibility allows arbitrary non-linear models to be added to OA Gear.

The OA Gear Timer supports the standard timing library formats offered by Cadence (.TLF) and Synopsys (.LIB "Liberty"). For timing constraints, a useful subset of the .SDC file format is supported, sufficient for use in timing-based physical design. For example, the following .SDC commands are supported:

- Set the clock period.
- Create external delays on primary inputs and outputs.
- Set the driving cell for inputs.
- Set load capacitance on outputs.

OA Gear Bazaar: User interfaces

Two visualization tools are included with OA Gear. One is a *layout viewer*, intended for visualization of placement results. This tool displays a geometrically correct physical layout of a design, where instances (gates) are shown with their proper shapes and with their assigned locations. The second tool is a *netlist browser*, which displays simple shapes for the instances, not necessarily corresponding to their actual sizes, and with locations chosen by the netlist browser. This tool gives the designer the ability to easily view and verify netlist connectivity.

The architecture of the visualization tools was designed to be flexible and easily customizable by the end developer. All user interface components are programmed using *QT*, a popular cross-platform graphics toolkit [102]. *QT* is freely available for non-commercial use of OA Gear. The user interface also

make extensive use of *OpenGL*, a standard API for graphics acceleration, for improved performance on modern graphics hardware [103].

Furthermore, OA Gear includes a command line interface component. This is intended to allow support for scripting, to provide for batch jobs and similar tasks which are amenable to text-based interaction.

OA Gear Functionality

The OA Gear Functionality package provides a platform for representing, synthesizing, and reasoning about design functionality. The components included in the OA Gear Functionality package and how they interact with OpenAccess is shown in Figure 3.6. Functional descriptions can be imported from RTL Verilog or .LIB libraries. The supported RTL Verilog format is often referred to as the *IWLS* format ¹. All functional information is synthesized into a common format, an AND/INVERTER graph extended with sequential vertices. All sequential vertices have an input, an output, and store exactly one bite of state. The sequential behavior is specified using one of three available models:

- BLACKBOX, hides non-synchronous behavior and mimics a finite state machine model.
- DLATCH, a simple D-latch.
- FEEDBACK, latches state the moment of being placed into a feedback loop.

In a design mapped with OA Gear Functionality all functionality is implemented in the leaf cells. Leaf cells are drawn from a set of library cells. Technology mapping removes pieces of the graph and replaces them with functionally equivalent library instances.

¹There is a freely available benchmark set called the IWLS benchmark set which are all on the RTL Verilog format. IWLS stands for International Workshop on Logic Synthesis.

Figure 3.6: OA Gear functionality.

Chapter 4

And/Inverter graphs

An AND/INVERTER graph is a directed acyclic graph where the vertices represent two-input AND functions. In logic synthesis, it can be used as a structural representation of a function to be reasoned about or as a subject graph used in the process of technology mapping.

AND/INVERTER graphs have a number of advantages. First, by restricting the network vertices to two simple functions, i.e. two-input ANDs and inverters, much faster analysis can be achieved. Second, AND/INVERTER graphs permit quick recognition of isomorphic parts using structural hashing which allows for efficient storage.

In this chapter, we discuss the AND/INVERTER graph representation, how to construct them and the use of structural hashing. All our optimization algorithms are based on AND/INVERTER graphs.

4.1 Graph representation

An AND/INVERTER graph is a directed acyclic graph of type $G(V, E)$. External inputs are called *primary inputs* (PI) and external outputs are called *primary outputs* (PO). The AND/INVERTER graph has three types of vertices:

- A unique terminal vertex represents the constant zero or the constant one value. The value is depending on if the vertex is referenced by a complemented edge or by a non-complemented edge.

- A second type of vertex has no incoming edges and models primary inputs.
- The third vertex type has two incoming edges and represents the Boolean AND of the vertex functions referenced by the two edges.

INVERTER attributes on the graph edges indicate Boolean implementation (see Section 2.2). Using this graph representation, a reasoning problem is expressed as an obligation to prove a particular graph vertex to be constant zero or constant one.

An example of an AND/INVERTER graph representing the function $f = \overline{xy} \cdot \overline{yz}$ is shown in Figure 4.1. A filled dot on an edge symbolizes the INVERTER attribute.

Figure 4.1: AND/INVERTER graph for the function $f = \overline{xy} \cdot \overline{yz}$.

The *level slack* of a vertex $v \in V$, $l(v)$, is defined as

Definition 1 $l(v) = l_r(v) - l_a(v)$

where

$$l_a(v) = \begin{cases} 0 & \text{if } v \in PI \\ \max_{\forall u \in fanin(v)} (l_a(u)) + 1 & \text{otherwise} \end{cases}$$

and

$$l_r(v) = \begin{cases} \max_{\forall u \in PO} (l_a(u)) & \text{if } v \in PO \\ \min_{\forall u \in fanout(v)} (l_r(u)) - 1 & \text{otherwise} \end{cases}$$

The *level* of a specific vertex in the AND/INVERTER graph is the same as $l_a(v)$ in Definition 1. The concept of $l_a(v)$, referred to as *arrival time*, is illustrated

in Figure 4.2 and $l_r(v)$, called *required time*, is illustrated in Figure 4.3. In Figure 4.4 the resulting slack is shown. Definition 1 gives that a *critical path* is a path through the AND/INVERTER graph on which all the vertices have level slack zero. The *critical path length* is the number of AND vertices on the critical path. We will often refer to the critical path length as the *number of levels* in the AND/INVERTER graph. The number of levels in the graph is given by the latest arrival time at the primary outputs, $\max_{\forall u \in PO} (l_a(u))$. For example, in Figure 4.2 the primary output f has the latest arrival time, which gives that the number of levels in the AND/INVERTER graph is three. This is the same as counting the number of AND vertices on the critical path marked with bold level slacks in Figure 4.4.

The total number of vertices in the AND/INVERTER graph is sometimes referred to as the *size* of the AND/INVERTER graph. The width of the AND/INVERTER graph is the maximum number of AND vertices with the same level. In Figure 4.2, the width of the AND/INVERTER graph would be two, since there are two AND vertices with level one (xy and zw). All other levels have fewer AND vertices.

Figure 4.2: AND/INVERTER graph with arrival times.

4.2 Structural hashing

An AND/INVERTER graph is built from the primary inputs toward the primary outputs using a set of construction operators. There are three basic constructors:

Figure 4.3: AND/INVERTER graph with required times.

Figure 4.4: AND/INVERTER graph with level slacks.

1. CREATEINPUT
2. CREATEAND2
3. CREATEINVERTER

Other operators for alternative or more complex operations are composed of these three basic constructors. Intermediate functions are passed between constructors by edge handles, which consist of a reference to the source vertex and a possible INVERTER attribute. The same handles are applied by the reasoning application to refer to functions that are stored by the graph.

The implementation of the construction operation CREATEINPUT is straight forward. It allocates and initializes a corresponding vertex data structure and returns a handle pointing to it. Similarly, the operation CREATEINVERTER simply toggles the attribute of the edge reference. Figure 4.5, shows

```

algorithm CREATEAND2( $p_1, p_2$ )
/* constant folding */
  if  $p_1 == \text{CONST\_0}$  then
    return  $\text{CONST\_0}$ ;
  if  $p_2 == \text{CONST\_0}$  then
    return  $\text{CONST\_0}$ ;
  if  $p_1 == \text{CONST\_1}$  then
    return  $p_2$ ;
  if  $p_2 == \text{CONST\_1}$  then
    return  $p_1$ ;
  if  $p_1 == p_2$  then
    return  $p_1$ ;
  if  $p_1 == \overline{p_2}$  then
    return  $\text{CONST\_0}$ ;
/* rank order inputs to catch commutativity */
  if  $\text{rank}(p_1) \geq \text{rank}(p_2)$  then
     $\text{swap}(p_1, p_2)$ ;
/* check for isomorphic entry in hash table */
  if  $(p = \text{HASHLOOKUP}(p_1, p_2)) == \text{NULL}$  then
     $p = \text{NEWANDVERTEX}(p_1, p_2)$ ;
  return  $p$ ;
end

```

Figure 4.5: Pseudo-code for the algorithm CREATEAND2, one of the AND/INVERTER graph constructor.

the pseudo-code for the operation CREATEAND2. The algorithm takes two edge handles as input parameters and returns an edge handle that represents the output of the Boolean AND operation. In the code, the notion $\overline{p_2}$ denotes Boolean complementation of operator p_2 using the procedure CREATEINVERTER. The first part of the algorithm performs constant folding, which automatically simplifies redundant and trivial expressions in the graph structure. Next a hash-lookup identifies isomorphic graph structures and eliminates them during construction. For this the procedure HASHLOOKUP checks whether an AND vertex with the requested input edges has been created before. If found the existing vertex is reused, otherwise a new ver-

```

algorithm NEWANDVERTEX( $p_1, p_2$ )
   $p = \text{ALLOCVERTEX}(p_1, p_2);$ 
  /* allocate a new vertex */
   $\text{ADDTOHashTable}(p, p_1, p_2);$ 
  /* add the new vertex to the hash table */
  return  $p$ ;
end

```

Figure 4.6: Algorithm NEWANDVERTEX for allocating a new graph vertex.

tex is created using the function NEWANDVERTEX. Before applying the hash-lookup the two operators p_1 and p_2 are ordered using a unique ranking criteria. This assures that commutative expressions, such as $p_1 \cdot p_2$ and $p_2 \cdot p_1$, are merged onto the same graph vertex.

The algorithm NEWANDVERTEX is shown in Figure 4.6. It is used to allocate a new graph vertex, ALLOCVERTEX, and add a corresponding entry to the hash table, ADDTOHASHTABLE.

4.3 Applications

AND/INVERTER graphs has been used for multiple purposes in logic synthesis since the early sixties [104].

In [62], a procedure which performs logic decomposition during technology mapping is presented. The data structure used both for the reasoning part, as well as the mapping part is an extended AND/INVERTER graph. The extended AND/INVERTER graph is called *mapping graph* and compactly encodes a set of AND/INVERTER graphs. In effect, the technology mapping algorithm applies tree-mapping to every encoded AND/INVERTER graph and retains the best result.

Kuehlmann et al. [105] use an AND/INVERTER graph data structure to perform Boolean reasoning for equivalence checking and functional property verification. They implemented a SAT¹ procedure that works directly on an AND/INVERTER graph allowing a tight interaction with BDD sweeping, local circuit graph transformations, and random simulation.

¹The question a SAT procedure is trying to answer is whether there is a variable assignment such that the given function has one as output i.e., is the function satisfiable?

Chapter 5

A greedy local transformation-based method

In this chapter a local transformation-based system for minimizing the number of AND vertices in an AND/INVERTER graph is presented. The local transformations are generated with the help of NPN classes (**N**egation-**P**ermutation-**N**egation) of Boolean functions (see Section 2.5). The use of NPN classes to generate the local transformations distinguishes our approach from previously proposed local transformation methods, which define the set of eligible transformations directly in the graph domain (see Section 3.4). In our case, we analyze the transformations in the Boolean domain, which implies that more of the interesting cases are taken into consideration.

5.1 Local transformations

The presented local transformation-based system aims at minimizing the number of vertices in an AND/INVERTER graph. In addition, it removes redundancies as well as reduces the number of locally reconvergent paths. Locally reconvergent paths are defined as paths which reconverge within three logic levels. As presented in [106] and [107], reconvergent paths are making signal probability calculations harder. The signal probability of a net in a combinational circuit is the probability that a randomly gen-

erated input vector will produce the value one on this net [108]. Signal probabilities are used for computing average switching activity in low-power optimization [106]. They are also used in testing [109] and probabilistic verification [110].

We start by presenting a theorem which is used to generate the local NPN-based transformations. The transformations are applied to an AND/INVERTER graph.

Theorem 1 *Functions belonging to the same NPN class have a minimum AND/INVERTER graph representation of the same size.*

Proof: Suppose there is a function, f in an NPN class which has a smaller AND/INVERTER graph representation compared to the other functions in the same class. We can construct an AND/INVERTER graph for any function in the NPN class by permutation of the input variables and/or by negation of one or more input variables and/or negation of the overall function, as shown in Figure 2.6, Section 2.5. Since neither of these operations affect the size of the AND/INVERTER graph, any function belonging to the same NPN class can be represented by an AND/INVERTER graph of the same size as f . Therefore, functions belonging to the same NPN class have a minimum AND/INVERTER graph of the same size.

□

Theorem 1 holds not only for AND/INVERTER graphs, but also for general Boolean circuits. The proof can be extended accordingly.

To find the minimum AND/INVERTER graph for a certain NPN class, all possible graphs for this class are considered. Redundancies are not taken into account when generating these graphs. Rules on how to transform a non-minimum AND/INVERTER graph representation into a minimum one are defined manually. A complete list of implemented replacement rules is shown in Table 5.1. The first column states which NPN class the function belongs to, the second one shows the minimal AND/INVERTER graph representation expressed as a function. The last column lists which AND/INVERTER graph representations will be identified and possible replaced by the minimal representation. An empty line means that there are no identifiable functions for the corresponding NPN-class. Both the minimal function as well as the identifiable functions hold for any negation and/or permutation of their inputs and/or negation of their output.

For example, the minimum AND/INVERTER graph representation for NPN class 8 is in a factored form, $f = x(y+z)$. It requires three AND vertices

Class	Minimal function	Identifiable functions
1	1	$x + 1$ $x + \bar{x}$ $(x + y) + \bar{x}$
2	x	$x \cdot 1$ $x \cdot x$ $(x + y)x$
3	$x + y$	$x\bar{y} + \bar{x}y + xy$ $(x + y) + x$
4	$x \oplus y$	
5	$x(y + z) + yz$	$xy + yz + xz$
6	$x \oplus y \oplus z$	
7	$x + y + z$	$(x + y + z) + x$ $x + \bar{x}(y + z)$
8	$x(y + z)$	$xy + xz$
9	$x(y \oplus z)$	$(xy) \oplus (xz)$
10	$xy + \bar{x}z$	$\bar{x}y + xz + yz$
11	$xy + \bar{x}\bar{y}z$	
12	$xyz + \bar{x}\bar{y}\bar{z}$	
13	$x(\bar{y} + \bar{z}) + \bar{x}yz$	$x\bar{y} + x\bar{z} + \bar{x}yz$
14	$x(y\bar{z} + \bar{y}z) + \bar{x}yz$	$xy\bar{z} + x\bar{y}z + \bar{x}yz$

Table 5.1: Replacement database for three variable NPN classes.

(primary inputs are not counted). This function can also be expressed as $f = xy + xz$, by the distributive law of Boolean algebra (see Section 2.2). In Figure 5.1 AND/INVERTER graphs for both expressions are given. As the figure shows, the expression, $f = xy + xz$, requires an additional AND vertex. The AND/INVERTER graph representation for the expression, $f = x(y + z)$ is the minimum AND/INVERTER representation for NPN class 8.

For some NPN classes there are larger gains. An example is NPN class 13. A minimum AND/INVERTER graph for NPN class 13 has four AND vertices. A non-minimum representation without redundancies can have up to six vertices. This is illustrated in Figure 5.2.

The replacement rules listed in Table 5.1 are all obtained from up to three variable NPN classes. If the NPN classes are extended to the four variable case, more replacement rules can be defined. We have only defined one re-

Figure 5.1: Two different logic expressions for NPN class 8; a) $f = xy + xz$ and b) $f = x(y + z)$.

Figure 5.2: Two different logic expressions for NPN class 13; a) $f = x\bar{y} + x\bar{z} + x\bar{y}\bar{z}$ and b) $f = x(y\bar{z} + \bar{y}z) + x\bar{y}z$.

Figure 5.3: Two different logic expressions for a four variable NPN function; a) $f = xyz + zw$ and b) $f = z(xy + w)$.

placement rule for four-variable functions and any negation/permutation of its inputs and/or output. It is shown in Figure 5.3, where the AND/INVERTER graph for the expression $f = xyz + zw$, is replaced by the graph for the expression $f = z(xy + w)$ implementing the same function. By applying this transform, one AND vertex is saved. It is worth noticing, that the reconverging path starting at the primary input z and converging at f , is removed by the transform. Such local transform will not be identified by the algorithm presented in [111], since its structural analysis only considers the two immediate levels preceding a vertex. In contrast, the above identified transform affects the three immediate levels preceding a vertex.

Some of the defined replacement rules removes redundancies. In Table 5.1, NPN classes 1, 2, 3 and 7 all have identifiable functions with redundancies. For example, NPN class 3 has the identifiable function $f = (x + y) + x$ which will be replaced by $f = x + y$. In case of constants, they are propagated through and any resulting disconnected parts of the AND/INVERTER graph are removed.

```

algorithm NPNSEARCH( $c_0$ )
/*  $c_0$  is the initial AND/INVERTER graph */
 $c := c_0$ ;
 $cost := |c|$ ;
/*  $|c|$  returns the number of vertices in  $c$  */
while  $|c| < cost$  do
     $cost := |c|$ ;
    for  $\forall v \in c$  do
         $matchlist[v] := MATCH(v)$ ;
        /* MATCH( $v$ ) returns the best match for  $v$  or */
        /*  $-1$  if there is none */
    /* apply all possible transforms */
    for  $\forall v \in c$  do
        if  $dirty(v) == false$  then
            if  $matchlist[v] \neq -1$  then
                 $c := APPLY(v, matchlist[v])$ ;
    return  $c$ ;
end

```

Figure 5.4: Pseudo-code of the NPN-based greedy optimization algorithm.

5.2 Optimization algorithm

This section presents the optimization algorithm for our greedy local transformation based method. It aims at minimizing the size of the AND/INVERTER graph by applying transformations from the replacement database. The pseudo-code of the presented optimization algorithm is shown in Figure 5.4.

The algorithm NPNSEARCH, takes as input parameter an AND/INVERTER graph and will return the AND/INVERTER graph with the fewest number of vertices encountered.

The AND/INVERTER graph is traversed breadth first from its primary outputs to its primary inputs. Each vertex, v , is considered a function. The procedure MATCH(v) tries to match the subgraph rooted at v to any of the patterns for which a local transformation is defined. The match which reduces the number of vertices the most is returned. In case of multiple matches with equal cost, transformations that remove local redundancies

are chosen before other transformations. Only transformations which reduce the number of AND vertices in the AND/INVERTER graph will be returned. In case there is no match, -1 is returned.

The optimization algorithm iteratively traverses the graph marking possible subgraph transformations vertices and then, after the marking, replaces as many of them as possible. The procedure APPLY applies a given transformation to a given vertex. After the AND/INVERTER graph has been locally transformed by APPLY, affected vertices are marked as dirty. The procedure APPLY is only called for a particular vertex, v , if it is not marked as dirty. The algorithm stops iterating when no further minimization can be made.

5.3 Experimental results

For evaluating the quality and performance of the local transformation-based greedy optimization, we performed a set of experiments using circuits from the IWLS 2002 set [93]. All experiments were performed on a Sun Ultra 10 with a 333 MHz Ultra SPARC-IIi CPU and 128 MB RAM, running Sun Solaris 5.6. Optimized graphs were verified using a BDD-based equivalence checker.

Our main objective was a direct comparison with the two most popular scripts used in SIS, *script algebraic* and *script rugged*. The synthesis tool SIS is presented in Section 3.4. For SIS, we used the following synthesis flow. After executing *script algebraic* or *script rugged*, the resulting network was decomposed into an AND/INVERTER graph with the command *tech_decomp -a 2*. Any redundancy was removed with the command *red_removal*. The optimization time in SIS was measured with the SIS command *time*. Both the script execution as well as the redundancy removal was accounted for. The decomposition was not timed.

The presented local transformation-based greedy optimization was timed with the C command *clock()*. The functions to be optimized were on IWLS format (see Section 3.5).

Table 5.2 and Table 5.3 summarize the results for benchmarks with more than 200 vertices after optimization with the presented NPN-based optimization algorithm. Column 1 shows the name of the benchmark, column 2 the number of primary inputs and column 3 the number of primary outputs. The next three columns show the number of AND vertices, V_a , the number of levels in the AND/INVERTER graph, L_a , and the execution time, t_a , for SIS

script algebraic, respectively. A “*” sign after the number of AND vertices indicates that the command *red_removal* could not be executed. The command *red_removal* was aborted after 200 seconds. A “-” sign indicates that the script did not terminate. The corresponding results for SIS *script rugged* are displayed in columns V_r , L_r , t_r , respectively. The last three columns show the results for the presented NPN-based greedy optimization.

Even though the presented system is quite simple, it obtains AND/INVERTER graphs of a size comparable to both SIS *script algebraic* and *script rugged* for most of the functions. For 20% of the benchmarks, it obtains smaller graphs than both scripts. Taking into account the fact that our system is more than 1000 times faster on average than *script algebraic* and more than 2000 times faster on average than *script rugged*, this demonstrates the usefulness of the presented approach.

5.4 Conclusion

This chapter discusses a greedy local transformation-based method for optimizing multiple-level logic circuits with respect to area, using literal count as objective. The replacement rules are generated using NPN classes of Boolean functions. The drawbacks of the discussed technique are:

- It is using a simplistic optimization objective.
- The optimization algorithm is greedy.

The use of a simplistic objective function often results in poor circuit structures. For example, the number of vertices in the AND/INVERTER graph is only loosely correlated with the delay of the mapped circuit. The major disadvantage with a greedy optimization algorithm is that it might quite fast get stuck in a local minimum instead of exploring all possibilities. These problems are addressed in Chapter 7, where a simulated annealing approach to logic optimization is proposed. Moreover, a more complex objective function, providing the choice of minimizing the number of vertices or number of levels, will be introduced.

The positive experience from our attempt at a local transformation-based logic synthesis system is that the AND/INVERTER graph representation proved to be an excellent data structure providing fast and efficient manipulation, which is key in local transformation-based system. Further-

benchmark	PI	PO	algebraic			Boolean			NPN-based		
			V_a	L_a	t_a	V_r	L_r	t_r	V_n	L_n	t_n
9symm	9	1	210	16	1.2	217	18	8.5	204	18	<0.01
C1355	41	32	526	22	7.2	430	20	13.8	428	22	0.02
C1908	33	25	413	29	9.3	428	35	17.1	399	33	0.05
C2670	233	140	650	29	25.1	779	32	116.0	675	22	0.15
C3540	50	22	1032	42	55.3	1297	46	113.9	1032	41	0.03
C432	36	7	156	27	5.4	284	42	84.4	209	43	<0.01
C499	41	32	422	24	5.3	430	20	13.5	396	21	0.01
C5315	178	123	1554	34	43.9	1530	35	23.6	1650	38	0.35
C6288	32	32	2379	94	53.3	2439	142	62.7	1916	93	0.07
C7552	207	108	1818	38	169.1	2542	42	211.9	2026	30	0.13
C880	60	26	373	28	3.6	404	40	4.6	327	25	<0.01
alu2	10	6	320	34	19.3	318	34	33.8	266	15	0.02
alu4	14	8	631	40	87.4	600	40	160.4	933	63	0.07
apex6	135	99	730	19	6.8	775	23	7.3	659	20	0.01
apex7	49	37	214	16	2.9	232	17	2.1	239	15	0.03
bigkey	262	197	3434*	12	34.4	3444*	14	51.0	5173	15	11.5
cbp_32_4	33	33	321	67	3.4	404	97	4.2	368	83	<0.01
cps	24	109	963	17	36.6	1161	19	108.8	1878	32	0.95
cse	7	7	181	13	2.2	201	14	5.9	240	25	0.02
dalu	75	16	851	23	35.8	958	23	113.2	1513	38	0.37
des	256	245	3283	20	80.3	3298	25	153.5	4065	40	11.29
dsip	228	197	2760*	14	47.8	2757*	14	42.4	2745	15	0.05
ex1	9	19	204	15	12.4	225	15	18.0	316	33	0.01
example2	85	66	280	10	2.2	307	11	2.8	288	14	0.02
exep	30	63	618	17	19.6	-	-	>500	1040	65	0.21
f51m	8	8	238	11	1.4	117	27	1.0	206	20	0.03
frg2	143	139	812	15	17.8	796	24	36.4	1480	17	0.50
i10	257	224	2063*	53	20.9	3154	55	30.3	2620	51	0.32
i2	201	1	215	15	6.1	532	14	16.0	228	47	0.01
i4	192	6	194	14	207.4	-	-	>500	246	16	<0.01
i5	133	66	198	19	5.5	198	19	1.6	313	19	<0.01

Table 5.2: Comparison of the presented greedy local transformation-based system with SIS.

benchmark	PI	PO	algebraic			Boolean			NPN-based		
			V_a	L_a	t_a	V_r	L_r	t_r	V_n	L_n	t_n
i6	138	67	448	6	8.4	448	6	4.2	548	7	0.39
i7	199	67	572	7	3.9	572	7	5.5	677	8	0.77
i8	133	81	947	15	20.9	1125	17	30.1	3306	22	0.16
i9	88	63	590	13	61.8	590	13	9.2	702	21	0.98
k2	45	45	827	22	43.1	919	20	79.3	2183	192	0.06
key	258	193	3433	13	130.3	3433	14	192.4	3278	14	0.04
keyb	7	2	216	17	6.6	283	39	26.0	406	46	0.04
kirkman	12	6	216	11	2.7	256	13	40.5	224	20	0.05
mclc	11	6	91	12	3.6	76	13	4.2	269	23	0.20
minmax10	13	10	421*	48	89.6	567*	52	171.0	436	47	0.02
minmax12	15	12	475*	66	104.8	691*	64	61.6	526	55	0.01
minmax20	23	20	803*	106	4.2	1171*	101	21.0	886	87	0.03
minmax32	35	32	1311*	149	8.0	-	-	>500	1426	135	0.05
minmax5	8	5	192	30	0.6	232	30	0.2	217	27	0.01
mult16a	17	1	189	49	3.0	197	51	3.9	220	49	0.01
mult32a	33	1	381	97	6.3	405	99	13.4	444	97	0.02
mult32b	32	1	427	5	45.4	367	7	42.7	398	5	0.01
pair	173	137	1527	22	17.3	1654	35	32.9	1565	24	0.02
planet	7	19	472	22	13.6	545	25	27.5	905	69	0.05
rot	135	107	592	27	7.1	747	30	11.6	784	34	0.23
sand	11	9	449	19	11.9	554	32	41.1	871	61	0.05
sbc	40	56	747*	16	6.0	830*	17	30.1	685	16	0.01
scf	27	56	687	19	91.9	959*	25	137.5	1233	83	0.01
styr	9	10	421	17	11.6	606	17	82.2	648	47	0.04
t481	16	1	666	20	366.6	245	19	120.3	1633	24	1.68
tbk	6	3	682*	19	25.4	-	-	>500	969	74	0.13
too_large	38	3	-	-	>500	-	-	>500	6311	91	19.38
vda	17	39	479	17	14.3	509	16	37.0	1057	60	0.01
x1	51	35	287	12	2.4	340	10	3.2	979	31	0.32
x3	135	99	682	18	6.1	807	17	10.2	997	18	0.37
x4	94	71	338	10	14.1	363	15	3.9	526	18	0.26

Table 5.3: Comparison of the presented greedy local transformation-based system with SIS.

more, the AND/INVERTER graph data structure turned out to be memory efficient and could handle cases where building a BDD would be infeasible.

Chapter 6

Simulated annealing

Simulated annealing is a generic probabilistic search algorithm for finding a good approximation of the global optimum of a given objective function in a large discrete search space. It was independently invented by S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi in 1983 [112], and by V. Cerny in 1985 [113]. During each search step the annealing algorithm replaces the current solution by a randomly selected "neighbor" solution. The neighbor is chosen with a probability that depends on the difference between the corresponding objective function values and on a global control parameter T , typically referred to as *temperature*. Starting from a high value, T is gradually decreased during the search such that the current solution changes almost randomly when T is large, but the moves become increasingly biased towards better solutions as T approaches zero. The possibility of "uphill" moves for larger values of T ensures probabilistically that the search climbs out of local minimums.

A suggestive name, *probabilistic hill climbing*, has been found for this type of algorithms. The mental picture supporting this name is the state space as a hilly landscape. States that can be reached from each other by a small number of moves are thought to be close together in this landscape. The height of each state is proportional to the cost of the state. When tracing the subsequent moves of a greedy algorithm we will never climb up a hill. Once we are at the bottom of a valley no change will occur any more. The annealing algorithm, on the contrary, has a non-zero probability on an uphill move. The name given to algorithms with that feature is therefore probabilistic hill climbing algorithms. "Probabilistic" because a random

number is involved in the acceptance decision, and “hill climbing” because a state with a higher cost than the current one might be accepted as the next state.

In this chapter, a literature review of simulated annealing is presented. The mathematical model as well as various annealing schedules are discussed. Finally, requirements for convergence of the annealing algorithm are stated. If an instance of simulated annealing has the convergence property, it implies that given enough execution time the algorithm will return the optimal solution according to the given objective function. Hence, it is a desired property.

6.1 The physical analogy

The motivation for the simulated annealing algorithm comes from an analogy between the physical annealing of solids and combinatorial optimization problems. Physical annealing refers to the process of finding low energy states of a solid by initially melting the substance, and then lowering the temperature slowly, spending a long time at temperatures close to the freezing point. An example would be producing a crystal from the molten substance. In a liquid, the particles are arranged randomly. But the ground state of the solid, which corresponds to the minimum energy configuration, will have a particular structure, such as seen in a crystal. If the cooling is not done slowly, the resulting solid will not attain the ground state, but will be frozen into a metastable, locally optimal structure, such as a glass or a crystal with several defects in the structure.

In the analogy, the different states of the substance correspond to the different feasible solutions to the combinatorial optimization problem, and the energy of the system corresponds to the function to be minimized.

Physical annealing has been successfully modeled as a Monte Carlo simulation. Back in the fifties, Metropolis et al. [114] introduced a simple algorithm to simulate a collection of atoms at a given temperature. At each iteration, an atom is given a small random displacement and the resulting change, $\Delta\epsilon$, in the energy of the system is calculated. If $\Delta\epsilon \leq 0$ then the resulting change is accepted, but if $\Delta\epsilon > 0$, the change is accepted with probability $e^{-\frac{\Delta\epsilon}{k_B T}}$, where T is the temperature and k_B a physical constant called the *Boltzmann constant*. If a large number of iterations are carried out at each temperature, the system attains thermal equilibrium at each

temperature. At thermal equilibrium, the probability distribution of the system states follows a Boltzmann distribution where the probability of the system being in state s at temperature T is

$$\frac{1}{Z(T)} \cdot e^{\frac{\epsilon_s}{k_B T}}$$

where ϵ_s is the energy of state s and $Z(T)$ is the partition function required for normalization. The acceptance function introduced by Metropolis et al. [114] ensures that the system evolves into the required Boltzmann distribution.

The procedure of repeating the basic step until thermal equilibrium is reached is called a Metropolis loop. In Figure 6.1 the Metropolis loop is embedded in an outer loop, in order to adjust the temperature. How many steps that are executed in each Metropolis loop can be controlled by the adjust functions, `ADJUST` and `READJUST`, for the *exit* variable.

In analogy, the different states of the substance correspond to different feasible solutions to the combinatorial optimization problem, and the energy of the system corresponds to the cost function to be minimized. The determination of the initial temperature, the rate at which the temperature is reduced, the number of iterations at each temperature and the criterion used for stopping is known as the annealing or cooling schedule. The choice of annealing schedule has an important bearing on the performance of the algorithm and is discussed later in this chapter.

The physical analogy has led to ideas from statistical physics being suggested as ways of determining the annealing schedule. For example, specific heat and entropy might be measured and used to determine a suitable start temperature.

6.2 Mathematical model

An annealing algorithm works on a *state space*, which is a set with a relation. The elements of the set are called *states*. Each state represents a configuration. We denote the state space set by S and its cardinality by $|S|$. A *cost function*, $\epsilon: S \rightarrow \mathfrak{R}_+$, assigns a positive real number to each state. This number is interpreted as a quality indicator, in the sense that the lower this number the better is the configuration that is encoded in that state. By defining a neighbor relation over S , $\mu \subseteq S \times S$, a *structure*, sometimes called a *topology*, is endowed to the state set S . The elements of μ are called *moves*,

```

algorithm METROPOLIS( $s_0, T$ )
/*  $s_0$  is the initial state */
/*  $T$  is the temperature */
   $exit := false$ ;
   $s := s_0$ ;
  while  $exit == false$  do
     $exit := ADJUST$ ;
     $s' := DISPLACE(s)$ ;
    if  $random < e^{-\frac{\epsilon_{s'} - \epsilon_s}{k_B T}}$  then
       $exit := READJUST$ ;
       $s := s'$ ;
  end

```

Figure 6.1: Pseudo-code of the Metropolis algorithm.

and the states $(s, s') \in \mu$ connected via a single move are called neighbors. Similarly, the states $(s, s') \in \mu^k$ are said to be connected via a set of k moves. Since we want any state to be connected to any other state by a finite number of moves, we require the transitive closure of μ to be the universal relation of S :

$$\cup_{k=1}^{\infty} \mu^k = S \times S$$

The set of states connected to a given state $s_i \in S$, is denoted by N_i , and its cardinality is called the degree of s_i .

As already mentioned, the annealing algorithm operates on a state space. At the end of the execution of a step exactly one state is the current state. The probability that a given state will be the current state depends only on its cost, the cost of the previous state and the value of the control parameter i.e., the temperature, T . The theoretical model for describing the sequences of current states generated by the annealing algorithm is known as a *Markov chain*. Markov chains are explained in more detail in Section 2.6. The essential property of Markov chains is that the next state does not depend on the states that have preceded the current state. The probability that s' will be the next state, given that s is the current state is denoted by $\tau(s, s', T)$ and is called the *transition probability*. The transition probabilities for a certain value of T can be conveniently represented by a matrix $\mathbf{P}(T)$, the *transition matrix*. The transition matrix of the Metropolis loop does not

change from step to step, because T does not change. Markov chains with constant transition matrices are called *homogeneous*. The Metropolis loop can therefore be modeled by a homogeneous Markov chain.

Transition probabilities for states that are not connected by a move is zero. For other pairs of distinct states the probability is determined by the probability that, given the first state, the second one is selected, and the probability that, once selected, the second state is accepted as the next state. The probability that the state does not change has to be such that the sum of all transition probabilities with that state as first state is one, because there is always exactly one current state. The complete Markov model for the annealing is therefore

$$\tau(s, s', T) = \begin{cases} \alpha(\varepsilon(s), \varepsilon(s'), T) \beta(s, s') & \text{if } s \neq s' \\ 1 - \sum_{s''} \alpha(\varepsilon(s), \varepsilon(s''), T) \beta(s, s'') & \text{otherwise} \end{cases}$$

where α is the *acceptance probability function*, and β is the *selection probability function*.

The selection probability function should be defined so that only pairs of states in μ can be selected

$$\begin{aligned} \forall_{(s, s') \notin \mu} [\beta(s, s') &= 0] \\ \forall_{(s, s') \in \mu} [\beta(s, s') &\neq 0] \\ \forall_{s \in S} [\sum_{s' \in N} \beta(s, s') &= 1] \end{aligned}$$

Note that the selection probability is never zero for a pair of states connected by a single move. Another function, called the acceptance function, assigns a positive probability measure to a pair of costs, and a positive real number, the temperature. Therefore,

$$\alpha: \mathfrak{R}_+^3 \rightarrow (0, 1] \subset \mathfrak{R}.$$

6.3 Annealing schedule

In simulated annealing, for each value of the temperature T , a certain number of moves is generated and accepted or discarded before T is decreased. The initial value of T , the number of moves generated at each value of T ,

```

algorithm SIMULATED ANNEALING( $s_0, T_b$ )
/*  $s_0$  is the initial state */
/*  $T_b$  is the initial temperature */
 $T = T_b$ ;
 $s = s_0$ ;
while outer-loop criterion is not satisfied do
    while inner-loop criterion is not satisfied do
         $s' = \text{SELECT}(s)$ ;
        if  $\text{ACCEPT}(\epsilon(s), \epsilon(s'), T)$ 
             $s = s'$ ;
         $T = \text{UPDATE}(T)$ ;
end

```

Figure 6.2: Pseudo-code of the annealing algorithm.

the rate of decrease of T , and the criterion used to decide when to terminate the algorithm comprise the *annealing schedule*, also known as the *cooling schedule*. Different choices have a significant impact on the algorithm's effectiveness. In Figure 6.2 pseudo-code of the annealing algorithm is shown.

The algorithm takes two input parameters; the initial state s_0 and the start temperature T_b . The procedure `SELECT` implements the selection probability function, while the procedure `ACCEPT` implements the acceptance probability function. The `UPDATE` procedure decrements T according to the selected cooling schedule.

In general, as the simulation proceeds, the temperature T is gradually reduced. Initially, T is set to a high value (infinity). Then, it is decreased at each step according to some annealing schedule and finally it becomes zero towards the end of the allocated time budget. In this way, the system is expected to initially explore a broad region of the search space containing good solutions. Then, it drifts towards low-cost regions that become narrower and narrower. Finally, at $T = 0$, it moves downhill according to the steepest descent heuristic.

Decrements in the temperature

The lowering of the temperature during the algorithm's execution has been the subject of investigation for many researches. Otten and Van Gin-

neken [115] where the first to propose to replace the simple geometric ¹ lowering of the control parameter suggested in [112]. They proposed an advanced lowering function making use of the concept “quasi equilibrium” and of concepts borrowed from physics. Their cooling of T is defined as follows

$$T_{n+1} = \gamma \frac{T_n^2}{\sigma(T_n)}$$

where $\sigma(T_n)$ is the standard deviation of the mean cost at temperature T_n . γ is a constant such that

$$\gamma T_b \ll \sigma_\infty$$

T_b is the start value of the control parameter T and σ_∞ is the standard deviation of the average cost of the entire search space.

Aarts and Van Laarhoven [117] refined the quasi equilibrium concept to develop a schedule that converged provably in polynomial time.

Hung, Romeo and Sangiovanni-Vincentelli [118] proposed a schedule using an exponential fast reduction of the control parameter

$$T_n = \frac{\gamma}{\log(n + n_0)},$$

where $n = 1, 2, \dots$ and n_0 is any positive constant. γ is a problem dependent constant. With the introduction of the extremely fast cooling schedule by Lam and Delosme [119] a point in the development of ever faster cooling schedules was reached where the complexity of the calculations needed to estimate new control parameter values could no longer be compensated by the gain in speed of the algorithm’s convergence.

Initialization

Otten et al. [115], suggested that before starting with gradually decreasing T , the annealing algorithm should go through an initialization phase to obtain initial values for T , the accessibility, the cost average E and the standard deviation σ . If the states of S are all equally likely when the control parameter has its initial value T_b , the cost average of the first chain should be equal to the average of costs over the whole state set, that is E_∞ . This is not possible, however, because the average cost $E(T)$ is lower than

¹ T is updated by the geometric law $T_{n+1} = \gamma T_n$, with $0 < \gamma < 1$ [116].

E_∞ for any finite, positive T . For a normal distribution of costs the difference between the two averages for high values of T is

$$E_\infty - E(T) = \frac{\sigma_\infty^2}{T}.$$

Otten et al. [115] continue by observing that the standard deviation for high T is hardly dependent on T . Choosing T_b much higher than σ_∞ will result in a slightly lower cost average and a virtually unchanged deviation.

$$\text{For } T_b = f\sigma_\infty, \quad f \gg 1$$

$$E(t) \approx E_\infty - \frac{\sigma_\infty}{f}, \quad \text{and}$$

$$\sigma(T) \approx \sigma_\infty.$$

Stop criterion

The object of the stopping criterion is to recognize that the algorithm has reached a good enough solution and T is so small that the probability to escape from it is negligible.

One way to determine the occurrence of such a situation, is to compare the difference between the maximum and minimum costs among the accepted states at that value of T with the maximum change in cost in any accepted move at the same value of T . If they are the same, apparently all the states accessed are of comparable costs and there is no need to continue to use simulated annealing. T is then set to zero and the algorithm becomes a standard greedy random-selection algorithm. This mechanism to terminate simulated annealing has been found to be quite successful without any negative side effects [118, 120].

Another typical criterion for termination of the annealing process is to terminate the algorithm when the average cost does not change significantly for a few consecutive values of T [112, 121].

A different approach is followed by White [122]. He proposes to stop the process when the final value of T is such that, for every local minimum $s_i \in S_{min}$,

$$\min_{s_i \in S_{min}} \min_{s_j \in N_i} e^{-\frac{\epsilon(s_j) - \epsilon(s_i)}{T}} < \gamma.$$

The condition above gives

$$T_e \leq \max_{s_i \in S_{min}} \max_{s_j \in N_i} \frac{\varepsilon(s_j) - \varepsilon(s_i)}{\log(\frac{1}{\gamma})}.$$

Notice that $\max_{s_i \in S_{min}} \max_{s_j \in N_i} \varepsilon(s_j) - \varepsilon(s_i)$ is not known and has to be estimated.

6.4 Convergence to optimum

Between 1984 and 1986, several authors independently proved that it is possible to design a simulated annealing algorithm so that the probability to be in a state with the minimum cost approaches one as the temperature approaches zero [123, 124, 125, 126, 127]. This property is called *convergence*.

For finite search spaces S , a sufficient condition for convergence is *detailed balance* [115], requiring that the probability flows between any two states s_i, s_j in the state space are equal:

$$\pi_i(T) \cdot \tau_{ij}(T) = \pi_j(T) \cdot \tau_{ji}(T) \quad (6.1)$$

where $\pi_i(T)$ is the stationary probability distribution of the state s_i at temperature T . The stationary probability distribution is a vector $\pi(T) = (\pi_1(T), \pi_2(T), \dots, \pi_{|S|}(T))$ which satisfies the equation

$$\pi^T(T) \cdot \mathbf{P}(T) = \pi^T(T)$$

where $\mathbf{P}(T)$ is the transition matrix and π^T is the transpose of π . In other words, the stationary probability distribution is a left eigenvector of the transition matrix, associated with the eigenvalue one.

Neither the existence nor the uniqueness of a stationary probability distribution is guaranteed for a general transition matrix $\mathbf{P}$. However, if the transition matrix $\mathbf{P}$ is irreducible and aperiodic, then there exists a unique stationary distribution π [128]. A transition matrix $\mathbf{P}(T)$ is *irreducible* if its underlying search space graph is strongly connected and, for all $s_i \in S$ and $s_j \in \Omega_i$, $P_{ij}(T) > 0$ [116]. The transition matrix is called *aperiodic* if its underlying search space graph has no state to which the search process will continually return with a fixed time period (greater than one). A sufficient condition for aperiodicity is that there exist a state $s_i \in S$ such that $P_{ii} \neq 0$ [116].

6.5 Applications

Simulated annealing has been applied to problems from a broad range of areas such as bioinformatics, chemistry, computer science, electrical engineering, finance, geology, mathematics, physics, missile defense, logistics, etc. A search on simulated annealing on google generates approximately 3.220.000 hits. There has been several books written about simulated annealing. For example, [128, 115, 129, 130, 131, 132, 133, 134, 135, 136, 137]. Previous attempts to apply simulated annealing to logic synthesis will be discussed in Section 7.1 and Section 9.1.

Chapter 7

Simulated annealing at And/Inverter graph level

In this chapter, we describe a logic synthesis approach based on rule-based randomized search using simulated annealing. Our work is motivated by two observations. First, traditional logic synthesis applies literal count as the primary quality metric during the technology independent optimization phase. This simplistic metric often leads to poor circuit structures as it cannot foresee the impact of early choices on the final area, delay, power consumption, etc. In this chapter, we present an objective function that can target minimizing number of vertices in the AND/INVERTER graph (literal count) or number of levels.

The second observation is that, although powerful, global Boolean optimization is not robust and corresponding algorithms cannot be used in practice without artificially restricting the application window. Other techniques, such as algebraic methods scale well but provide weaker optimization power. To address this problem, we use a randomized search that is based on a simple circuit graph representation and a complete set of local transformations that include algebraic and Boolean optimization steps.

7.1 Previous work

This section presents previous work done in the area of simulated annealing for technology independent logic optimization.

In [138], a multiple-level logic minimization technique for NMOS implementations of multiple-output functions is described. The presented technique can be decomposed into two phases:

1. A construction phase, which uses a greedy Boolean minimizer.
2. An optimization phase via simulated annealing.

During the construction phase a forest representation of multiple-output functions is constructed from the Boolean expression. Five types of nodes are used:

- Input variables (I).
- Output variables (O).
- Pull-up transistors (PU).
- Transistors in series (S).
- Transistors in parallel (P).

The children of S and P nodes represent the corresponding transistors in series or parallel respectively while the child of a PU node represents the associated pull-down network. All edges in the forest are directed. Cyclic graphs are not allowed.

Subtree isomorphisms or common subexpressions are detected and maintained through two attributes. The *label* attribute represents a logic function associated with a node and is used to identify nodes that are functionally equivalent. The labels are chained together to form a circular double-linked list. The *children* attribute is a sorted list that consists of the type of node and the labels of all its children. Initially all nodes are given distinct labels. Then children attributes are computed and inserted into a hash table. Nodes with identical children attributes are detected and reassigned to have the same label.

The final step of the construction phase invokes a greedy Boolean minimizer that reduces all pull-down networks according to the Boolean axioms. For an introduction to Boolean algebra, see Section 2.2.

In the second phase, the simulated annealing algorithm is applied using the following three sets of moves:

1. Gate separation or merging.

2. Subtree sharing or duplication. Subtree sharing is allowed only for PU or I nodes since a single pull-down network cannot be associated with more than one pull-up transistor.
3. Duality. Using De Morgan's law (see Section 2.2), a gate will be converted to its dual and its functionality preserved by adding or removing PU nodes at proper places.

A move is generated randomly, in controlled proportions, from the three sets. If a move is accepted, information for subtree isomorphism is updated by recomputing the children attributes of all modified nodes. Nodes that are deleted have their children attributes removed from the hash table. For a newly created node, a unique label is assigned and its children attributes are inserted. For a modified node, a deletion is performed, followed by an insertion. If identity between children attributes is detected during the insertion, the corresponding nodes are assigned identical labels. The children attributes of the parents of nodes whose labels have changed are also updated. Thus another round of deletion and/or insertion is required.

At the end of an annealing run, the Boolean minimizer is called upon to further minimize the final solution.

In [121], the authors showed that devising a mechanism to regulate the acceptance ratio to about 43% is key for an optimal annealing schedule. In their application of annealing to logic optimization this is achieved by biasing the probability of move selection among the three sets. For example, sharing or duplicating a subtree at high levels is undesirable at low temperatures since it typically results in a large cost change and hence is likely to be rejected. To bias the move selection probability with respect to the temperature is called *range control*. In [138], it was noted that range control did not have the large impact it was thought to have. The given reason is that only one out of three sets of moves can be properly controlled, namely subtree sharing and duplication.

A more recent attempt to apply simulated annealing to logic synthesis was presented in [139]. In this work, the Boolean circuit to be optimized is represented either as a sum-of-products (SOP) expression or a product-of-sums (POS) expression. SOP and POS forms are defined in Section 2.2. In both cases, the cost of the function is calculated as the number of literals. The move set includes two moves, *decomposition* and *extraction*. The decomposition operation consists of replacing a Boolean expression by a

set of less complex Boolean expressions. The extraction operation is applied to a function set. It identifies common subexpressions of two or more logic expressions and creates new intermediate variables corresponding to the subexpressions.

The main problem of the approach presented in [139], is that the size of the sum-of-product and product-of-sum forms grow exponentially as the number of variables of the function to be optimized increases. Therefore, such a method is not feasible for large circuits.

7.2 The annealing framework

As stated in Chapter 6, for applying simulated annealing to a specific optimization problem, one must define the search space, the neighbor selection method, the probability transition function, and the annealing schedule. Different choices have a significant impact on the algorithm's effectiveness. In this section, the overall setup of our annealing-based synthesis approach is described.

The basic iteration

We start by defining the state space, S .

Definition 2 *Let S be a set of all possible combinational acyclic Boolean circuits implementing a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$.*

At each step, the simulated annealing algorithm considers some neighbor $s' \in S$ of the current state $s \in S$, and probabilistically decides between moving the system to state s' or staying in state s . The probabilities are chosen so that the system ultimately tends to move to states of lower cost.

The set of neighbors of the state is defined by the following neighbor relation.

Definition 3 *The neighbor relation $\mu \subseteq S \times S$ is defined as follows:*

1. $(s, s') \in \mu$ if $s' \in S$ can be obtained from $s \in S$ by applying one of the following local replacement rules to a subgraph of s :

$$a) \ x \cdot (y \cdot z) = (x \cdot y) \cdot z,$$

$$b) \ x \cdot (y + z) = x \cdot y + x \cdot z,$$

Figure 7.1: Illustration of the local replacement rules.

c) $x \cdot x = x$,

d) $x \cdot \bar{x} = 0$, where $\bar{x}$ is the complement of x ,

e) $x \cdot 1 = x$,

f) $x \cdot 0 = 0$,

where x, y, z are the inputs of the subgraph. Figure 7.1 illustrates the replacement rules.

2. $(s, s') \in \mu$ if s' is obtained from s by merging two isomorphic vertices of s into one;
3. $(s, s') \in \mu$ if s' is obtained from s by making two copies of a multiple-fanout vertex of s and distributing the fanout nets between the copies;
4. if $(s, s') \in \mu$, then $(s', s) \in \mu$.

Next, we prove that the set of transformations specified by Definition 3 is complete for S .

Theorem 2 *If S is the set of all possible acyclic Boolean circuits for a given Boolean function, then the transitive closure of the neighbor relation μ specified by Definition 3 is the universal relation over S :*

$$\cup_{k=1}^{\infty} \mu^k = S \times S.$$

Proof: We can convert any acyclic Boolean circuit representing function f to a Boolean formula by repeatedly applying rule 3 of Definition 3. Any two equivalent Boolean formulas can be derived from each other by repeatedly applying the axioms of Boolean algebra. Rule 1 contains a complete set of axioms of Boolean algebra [140], except commutativity $x \cdot y = y \cdot x$ which is implemented by applying all rules for all permutations of their inputs. Since the set of moves is symmetric (rule 4), we can regenerate any acyclic Boolean circuit representing f . □

Thus, a state of S with the minimum possible cost can be reached from any initial state of S through a finite sequence of moves defined by μ .

Note that, if S is defined as above, it is countably infinite. However, in practice, our annealing algorithm only reaches a finite subset of S , containing Boolean circuits which are reachable from the initial one in a finite number of moves.

In Figure 7.1 some of the defined local replacement rules are illustrated using logic gates. The distributive law of Boolean algebra, represented by 1(b) in Definition 3, is displayed as AND/INVERTER graph representations in Figure 7.2. This rule corresponds to the replacement rule defined for NPN class 8 in Section 5.1. In Figure 7.3, rule 1(e) of Definition 3, which removes local redundancies, is displayed.

Definition 4 *Let S_k be a subset of S containing all states of S which are reachable from some initial state $s_0 \in S$ in up to k moves.*

Definition 5 *The set of neighbors of the state $s_i \in S_k$, denoted by N_i , is the set of states $s_j \in S_k$ which are reachable from s_i in a single move defined by $\mu \subseteq S_k \times S_k$.*

It is easy to see that, if S is replaced by S_k in the Definition 3, then $\mu \subseteq S_k \times S_k$ remains a universal relation (over S_k). Later, we prove that the probability that our annealing algorithm finds a state of S_k with the minimum cost approaches one as the temperature goes to zero. Since S_k is

Figure 7.2: Illustration of replacement rule 1(b) of Definition 3; a) $f = x \cdot y + z \cdot y$ and b) $f = y \cdot (x + z)$.

Figure 7.3: Illustration of replacement rule 1(e) of Definition 3; a) $f = x \cdot 1 + y$ and b) $f = x + y$.

only a subset of S , a minimum cost state for S_k is not necessarily minimum for S .

The state space of our simulated annealing algorithm can be represented by a directed graph, a *state transition graph*, where each vertex corresponds to a state of S_k . For each $s_i \in S_k$, there is an edge from s_i to each of its neighbors $s_j \in N_i$. The edges of the graph are labeled by the transition probabilities of going from one state to another, defined in the next section.

In Figure 7.4, a small example of a state transition graph for an instance of our annealing algorithm is shown. The states are represented by AND/INVERTER graphs. The moves shown in the figure is limited to a subset of the available local transformations defined by Definition 3. For example, starting in state s_0 , the neighboring state s_1 can be reached by applying rule 1(b) of Definition 3 to the initial AND/INVERTER graph while the neighboring state s_2 can be reached by applying rule 1(a) of Definition 3. Some

Figure 7.4: Illustration of a state transition graph for a small example of our annealing setup. The states are represented by AND/INVERTER graphs.

of the other local transformations applicable to the AND/INVERTER graph represented by state s_0 include rules 1(c) and 1(e) of Definition 3. These are not shown in Figure 7.4.

Transition probabilities

The probability of making the transition from some current state s_i to a neighbor state s_j is a function of the costs $\epsilon(s_i)$ and $\epsilon(s_j)$ of the two states, and the temperature T .

An essential requirement for the transition probability is that it must be non-zero for $(s_i, s_j) \in \mu$ when $\epsilon(s_j) > \epsilon(s_i)$, meaning that the system may move to the new state even when it has a higher cost than the current one. This feature prevents the method from becoming stuck in a local minimum.

On the other hand, when T approaches zero, the transition probability must approach zero for $(s_i, s_j) \in \mu$ and $\epsilon(s_j) > \epsilon(s_i)$, and be positive if $\epsilon(s_j) <$

$\epsilon(s_i)$. In this manner, for sufficiently small values of T , the system will increasingly favor moves that go "downhill", and avoid those that go "uphill". In particular, when T becomes zero, the procedure reduces to the greedy algorithm which accepts a move if and only if it reduces the cost of the current state.

Our algorithm uses the same transition probability as the algorithm of Kirkpatrick et al. [112]. It is specified as follows.

Definition 6 *The probability of transition from the state $s_i \in S_k$ to the state $s_j \in S_k$ at the temperature T is given by*

$$\tau_{ij}(T) = \begin{cases} \beta_{ij} \cdot \alpha_{ij}(T) & \text{if } s_i \neq s_j \\ 1 - \sum_{s_i \neq s_j} \beta_{ij} \cdot \alpha_{ij}(T) & \text{otherwise} \end{cases}$$

where selection probability β is given by

$$\beta_{ij} = \begin{cases} \frac{1}{|N_i|} & \text{for all } s_j \in N_i \\ 0 & \text{otherwise} \end{cases}$$

and the acceptance probability α is given by

$$\alpha_{ij}(T) = \min[1, e^{-(\epsilon(s_j) - \epsilon(s_i))/T}].$$

An essential difference between our annealing algorithm and the algorithm of Kirkpatrick et al. [112] is that, in their case, the size of the neighbor set N_i is identical for all states s_i . This implies that the selection function is symmetric, i.e., $\beta_{ij} = \beta_{ji}$ for all pairs of states s_i, s_j . In our case, the size of the neighbor set may vary for different states $s_i \in S_k$ and, thus, the selection function is not symmetric.

Convergence to optimum

In this section, we prove that our simulated annealing algorithm will return a global minimum given enough time, i.e. it converges. As shown in Section 6.4, for finite search spaces S , a sufficient condition for convergence is detailed balance, requiring that the probability flows between any two states

s_i, s_j in the state space are equal. For convenience, we will repeat the formal definition of detail balance, Equation 6.1, from Section 6.4:

$$\pi_i(T) \cdot \tau_{ij}(T) = \pi_j(T) \cdot \tau_{ji}(T) \quad (7.1)$$

where $\pi_i(T)$ is the stationary probability distribution of the state s_i at temperature T . The stationary probability distribution is a vector $\pi(T) = (\pi_1(T), \pi_2(T), \dots, \pi_{|S|}(T))$ which satisfies the equation

$$\pi^T(T) \cdot \mathbf{P}(T) = \pi^T(T)$$

where $\mathbf{P}(T)$ is the transition matrix and π^T is the transpose of π .

As mention in Section 6.4, neither the existence nor the uniqueness of a stationary probability distribution is guaranteed for a general transition matrix. The stationary probability distribution for our instance of annealing can be derived from the result of Lundy and Mees [126]. They proved a formula for the stationary probability distribution given that a number of requirements are satisfied. The selection function β has to be specified as in Definition 6 and the acceptance function α has to be a function depending only on the temperature and the difference in cost between the two states, $\Delta\epsilon_{ij} = \epsilon(s_i) - \epsilon(s_j)$. In addition, α has to satisfy the following three conditions:

1. $\alpha_{\Delta ij} \cdot \alpha_{\Delta jk} = \alpha_{\Delta ik}$, for any i, j, k such that $\epsilon(s_i) \leq \epsilon(s_j) \leq \epsilon(s_k)$.
2. If $\Delta\epsilon_{ij} \leq 0$, then $\alpha_{\Delta ij} = 1$.
3. If $\Delta\epsilon_{ij} > 0$, then $0 < \alpha_{\Delta ij} < 1$ and $\lim_{T \rightarrow 0} \alpha_{\Delta ij} = 0$.

If the requirements listed above are fulfilled, then the stationary probability distribution of any state s_i at the temperature T is given by the following expression

$$\pi_i(T) = \frac{|N_i| \cdot \alpha_{\Delta mi}(T)}{\sum_{\forall s_j \in S_k} |N_j| \cdot \alpha_{\Delta mj}(T)}$$

where s_m is any global minimum state of the search space.

It is easy to see that our acceptance probability α satisfies the three conditions above. Therefore, we can conclude the following result.

Theorem 3 *The stationary probability distribution of any state $s_i \in S_k$ at the temperature T is given by*

$$\pi_i(T) = \frac{|N_i| \cdot e^{-\epsilon(s_i)/T}}{\sum_{\forall s_j \in S_k} |N_j| \cdot e^{-\epsilon(s_j)/T}} \quad (7.2)$$

If the size of the neighbor set N_i is constant for all $s_i \in S_k$, then the formula (7.2) reduces to the stationary probability distribution of Kirkpatrick's et al. algorithm [112]:

$$\pi_i(T) = \frac{e^{-\epsilon(s_i)/T}}{\sum_{\forall s_j \in S_k} e^{-\epsilon(s_j)/T}}$$

By substituting Theorem 7.2 into Equation 7.1, we can show that the detailed balance condition is satisfied for S_k . Thus, our annealing algorithm converges.

7.3 Implementation details

The presented simulated annealing-based logic optimization algorithm is implemented in the OpenAccess environment with the functional extensions provided by OA Gear. The data structure used is an AND/INVERTER graph.

The pseudo-code of the simulated annealing algorithm is shown in Figure 7.5. The algorithm starts from an initial state s_0 and continues until the temperature reaches the end temperature T_e . The inner **for**-loop performs p moves for each given temperature T . The outer **while**-loop decreases the value of T . A separate state variable *best_state* keeps track of the best solution seen so far. This is a modification of the annealing algorithm suggested in [141].

The input parameter x is used to set the optimization objective to either target minimizing the number of vertices ($x = 0$) or the number of levels ($x = 1$) in the AND/INVERTER graph. The algorithm will return the best AND/INVERTER graph encountered according to the objective function.

Initialization and selection of the next state

The procedure INITIALIZE sets the initial values of T , T_e , and p . The start temperature is chosen so that the initial *acceptance ratio* is 0.44 [119, 142].

```

algorithm ANNEALING( $s_0, x$ )
/*  $s_0$  is the initial state */
/*  $x \in \{0, 1\}$  specifies optimization objective to be either */
/* minimizing number of vertices or number of levels */
( $T, T_e, p$ ) = INITIALIZE( $s_0$ );
 $best\_state := s_0$ ;
 $s := s_0$ ;
while  $T > T_e$  do
    for  $p$  times do
         $s' = \text{SELECT}(s)$ ;
        /* pick a neighbor of  $s$  */
        if ACCEPT( $s, s', x, T$ ) then
            /* accept the state  $s'$  */
             $s := s'$ ;
            if  $\epsilon(s) < \epsilon(best\_state)$  then
                 $best\_state := s$ ;
         $T = \text{COOL}(T)$ ;
        /* decrease  $T$  */
    return  $best\_state$ ;
end

```

Figure 7.5: Pseudo-code of the AND/INVERTER graph based-annealing algorithm.

The acceptance ratio is the ratio between selected moves and accepted moves. For example, if 100 moves are selected we want 44 of those to be accepted. The start temperature is selected by simulation, meaning that we run the inner-loop in the annealing for a test temperature (starting with $T = 3.0$) and see what the acceptance ratio is. If it is too high the temperature is decreased, if it is too low the temperature is increased.

The stopping criterion T_e is determined by comparing the difference between the maximum and minimum costs among the accepted states at the value of T that has the maximum change in cost in any accepted move at the same value of T . If they are the same, apparently all the states accessed are of comparable costs and there is no need to continue to use simulated annealing. The inner loop criterion p , is set to $0.01 \cdot |N_{s_0}|$, where $|N_{s_0}|$ is the

number of neighbors of state s_0 .

The procedure $\text{SELECT}(s)$ picks a neighbor s' of a given state s from the set of neighboring states of s . According to the Definition 6, all neighbors of s have equal probability to be selected.

The procedure $\text{ACCEPT}(s, s', T)$ decides whether to accept s' as a next state of s or not. According to the Definition 6, the acceptance probability is determined by $\epsilon(s)$, $\epsilon(s')$ and the temperature T . The probability of accepting states with lower costs decreases exponentially with T .

Objective function

In the AND/INVERTER graph-based annealing implementation, the objective function ϵ depends on two parameters of the AND/INVERTER graph corresponding to the state s :

- Number of vertices.
- Level slack.

Since the acceptance probability α_{ij} depends only on the difference in costs between the states s_i and s_j , the objective function can be treated as:

$$\Delta\epsilon_{ij}(x) = (1-x) \cdot \Delta n_{ij} + x \cdot \Delta l_{ij}$$

where $\Delta\epsilon_{ij} = \epsilon(s_i) - \epsilon(s_j)$ is the difference in costs between s_i and s_j , $\Delta n_{ij} = |V_{s_i}| - |V_{s_j}|$ is the change in number of vertices between s_i and s_j , Δl_{ij} is the change in their level slack. The level slack is calculated in accordance to Definition 1 in Section 4.1. The values of Δn_{ij} and Δl_{ij} are scaled to $[-1, 1]$. The control parameter x is used to select whether to target minimization of the number of vertices or minimization of the number of levels. The parameter x can only have two values:

- The value 0 for number of vertices optimization.
- The value 1 for number of levels optimization.

Cooling schedule

The rate of decrease of T is defined by $\text{COOL}(T)$. In our implementation, T is cooled down according to the schedule proposed by Romeo et al. [116]:

$$T = \frac{c}{\log(r + r_0)}$$

where r starts with 0 and is incremented by 1 after each update of T , $r_0 = 2$, and $c = T * \log(r)$ is computed for the initial values of T and r . We tried many other cooling schedules, including [115, 118], but found the schedule given above to be the most advantageous for our application.

7.4 Experimental results

To evaluate the quality and performance of the presented simulated annealing approach, we performed a set of experiments using all circuits in the IWLS 2002 benchmark set [93]. All experiments were run on a Pentium 4-based PC with a single 2.0GHz CPU, 512kB cache, and 1GB RAM operated by RedHat Linux, kernel version 2.4.18.

Our main objective was a direct comparison with the two SIS scripts, *algebraic* and *rugged*.

Our simulated annealing, at AND/INVERTER graph level synthesis flow, is based on first using *script algebraic* followed by the annealing-based optimization. By this flow, we aim at synthesizing circuits of a quality that is comparable to the one produced by *script rugged* with significantly less computational effort. Moreover, our annealing-based approach is fundamentally scalable and targeted at large circuits that cannot be handled by *script rugged*.

In a first experiment we ran all benchmark circuits through the two SIS scripts, *algebraic* and *rugged*, and compared them with our approach using two different optimization objectives, one for minimizing the number of vertices and one for minimizing the number of levels. The optimization time in SIS was measured with the SIS command *time*. Only the execution time of the scripts were included in the reported numbers. The simulated annealing algorithm was timed with the *C* command *clock()*.

Table 7.1 and Table 7.2 summarize the results for benchmarks with more than 500 AND vertices after optimization with the presented annealing-based optimization approach. V_a , V_r , V_s stand for the number of AND vertices in the AND/INVERTER graphs after executing *script algebraic*, *script rugged* and simulated annealing, respectively. Similarly, L_a , L_r , L_s stand for the number of levels. The last two columns compares the total execution time for the presented annealing-based approach (i.e. time for executing script *algebraic*, t_a , followed by simulated annealing algorithm, t_s) to the execution time of *script algebraic*, t_a , and *script rugged*, t_r , respectively.

For optimizing the number of vertices, (Table 7.1) our results show that the annealing based approach can further reduce the number of vertices in circuits optimized with algebraic methods by 13.2% on average. The average CPU overhead for this is 17.2%. When compared with Boolean optimization, the results for optimizing the number of vertices can be improved by 15.3% on average using only 82.2% of the CPU time of *script rugged*. moreover, our method also improves the number of levels for these circuits by 1.2% on average.

When targeting number of levels optimization (Table 7.2), an average improvement of 16.2% over *script algebraic* is achieved with an additional number of vertices improvement of 5.2%. The corresponding number of level improvement over *script rugged* is 23.5% with an additional number of vertices improvement of 3.6%.

In a second experiment we compared our annealing algorithm with the NPN-based greedy optimization algorithm discussed in Chapter 5. The result is presented in Table 7.3. The first column shows the name of the benchmark. The second column shows the total number of vertices in the AND/INVERTER graph after running the annealing based optimization targeting minimization of the number of vertices, V_s . In the third column the corresponding number of vertices for the local transformation-based greedy optimization is shown, V_g . The last column shows the improvement of our annealing algorithm over the greedy optimization algorithm in percent. As we can see, the annealing approach allows us to further reduce the number of vertices by 37.2% on average. The price to pay is an increased execution time.

In a last experiment, we evaluated the convergence of the annealing. This was done by monitoring the number of vertices at different temperatures during the execution of the annealing algorithm. Script algebraic was not executed before the annealing as in previous experiments. In Figure 7.6 the convergence curve for *C1355* is displayed. The x-axis shows the value of the temperature and the y-axis the total number of vertices in the AND/INVERTER graph at the corresponding temperature. As the execution time increases, the temperature decreases. Hence, we are moving from the right to the left on the x-axis.

Figure 7.6: Convergence of the annealing algorithm for benchmark C1355.

7.5 Conclusion

The implementation of an annealing framework for logic optimization provides the mean to handle more complex objective functions. Our experimental results demonstrates how the algorithm can target minimizing the number of vertices in an AND/INVERTER graph or the number of levels. Furthermore, better results are achieved by simulated annealing than by the simple greedy approach presented in Chapter 5.

A drawback with simulated annealing at AND/INVERTER graph level is that the graph characteristics, i.e. the number of vertices and the number of levels, used to optimize the given design do not properly reflect the actual area and delay in the resulting mapped circuit. A first attempt to overcome this problem is described in the next chapter. Furthermore, we want to be able to trade-off area for delay in the objective function and not just be able to target one of them.

benchmark	vertices					levels					time	
	V_a	V_r	V_s	$\frac{V_a - V_s}{V_a}$	$\frac{V_r - V_s}{V_r}$	L_a	L_r	L_s	$\frac{L_a - L_s}{L_a}$	$\frac{L_r - L_s}{L_r}$	$\frac{t_a + t_s}{t_a}$	$\frac{t_a + t_s}{t_r}$
C1355	526	430	482	8.37%	-12.09%	22	20	22	0.00%	-10.00%	142.5%	33.5%
C2670	650	779	573	11.85%	26.44%	29	33	30	-3.45%	9.09%	118.8%	64.4%
C3540	1032	1297	982	4.84%	24.29%	42	47	42	0.00%	10.64%	108.8%	20.7%
C5315	1554	1530	1450	6.69%	5.23%	34	35	35	-2.94%	0.00%	120.7%	134.6%
C6288	2379	2439	1873	21.27%	23.21%	94	142	92	2.13%	35.21%	111.9%	79.7%
C7552	1818	2542	1664	8.47%	34.54%	38	42	45	-18.42%	-7.14%	106.0%	267.4%
alu4	631	600	604	4.28%	-0.67%	40	39	40	0.00%	-2.56%	103.3%	30.6%
apex6	730	775	617	15.48%	20.39%	19	23	19	0.00%	17.39%	138.5%	112.5%
dalv	851	958	745	12.46%	22.23%	23	23	25	-8.70%	-8.70%	104.7%	31.1%
des	3283	3298	2980	9.23%	9.64%	20	25	24	-20.00%	4.00%	107.2%	87.3%
frg2	812	796	709	12.68%	10.93%	15	24	15	0.00%	37.50%	111.0%	88.8%
i10	2063	3154	1731	16.09%	45.12%	51	52	53	-3.92%	-1.92%	102.5%	14.7%
i7	572	572	505	11.71%	11.71%	7	7	7	0.00%	0.00%	148.9%	121.8%
i8	947	1125	813	14.15%	27.73%	15	17	16	-6.67%	5.88%	110.8%	95.0%
i9	590	590	513	13.05%	13.05%	13	13	13	0.00%	0.00%	124.4%	117.1%
k2	827	919	733	11.37%	20.24%	22	20	28	-27.27%	-40.00%	105.5%	59.8%
pair	1527	1654	1412	7.53%	14.63%	22	35	23	-4.55%	34.29%	118.8%	81.0%
rot	592	747	497	16.05%	33.47%	27	30	29	-7.41%	3.33%	120.5%	89.3%
t481	666	245	369	44.59%	-50.61%	20	19	30	-50.00%	-57.89%	103.2%	36.8%
x3	682	807	590	13.49%	26.89%	18	17	18	0.00%	-5.88%	136.2%	77.0%
average:	1137	1263	992	13.18%	15.32%	29	33	30	-7.56%	1.16%	117.20%	82.16%

Table 7.1: Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of vertices with SIS.

benchmark	vertices					levels					time	
	V_a	V_r	V_s	$\frac{V_a-V_s}{V_a}$	$\frac{V_r-V_s}{V_r}$	L_a	L_r	L_s	$\frac{L_a-L_s}{L_a}$	$\frac{L_r-L_s}{L_r}$	$\frac{t_a+t_s}{t_a}$	$\frac{t_a+t_s}{t_r}$
C1355	526	430	530	-0.76%	-23.26%	22	20	20	9.09%	0.00%	152.50%	35.88%
C2670	650	779	606	6.77%	22.21%	29	33	26	10.34%	21.21%	118.13%	64.07%
C3540	1032	1297	1002	2.91%	22.74%	42	47	38	9.52%	19.15%	108.78%	20.74%
C5315	1554	1530	1474	5.15%	3.66%	34	35	32	5.88%	8.57%	121.21%	135.19%
C6288	2379	2439	1876	21.14%	23.08%	94	142	91	3.19%	35.92%	111.85%	79.71%
C7552	1818	2542	1704	6.27%	32.97%	38	42	36	5.26%	14.29%	105.95%	267.17%
alu4	631	600	652	-3.33%	-8.67%	40	39	29	27.50%	25.64%	103.65%	30.74%
apex6	730	775	640	12.33%	17.42%	19	23	15	21.05%	34.78%	140.00%	113.75%
dalu	851	958	830	2.47%	13.36%	23	23	18	21.74%	21.74%	104.89%	31.16%
des	3283	3298	3077	6.27%	6.70%	20	25	17	15.00%	32.00%	108.89%	88.69%
frg2	812	796	784	3.45%	1.51%	15	24	11	26.67%	54.17%	114.04%	91.23%
i10	2063	3154	1792	13.14%	43.18%	51	52	39	23.53%	25.00%	102.80%	14.74%
i7	572	572	543	5.07%	5.07%	7	7	6	14.29%	14.29%	155.56%	127.27%
i8	947	1125	894	5.60%	20.53%	15	17	12	20.00%	29.41%	113.00%	96.86%
i9	590	590	624	-5.76%	-5.76%	13	13	11	15.38%	15.38%	135.00%	127.06%
k2	827	919	856	-3.51%	6.86%	22	20	15	31.82%	25.00%	107.09%	60.72%
pair	1527	1654	1444	5.44%	12.70%	22	35	19	13.64%	45.71%	119.33%	81.36%
rot	592	747	528	10.81%	29.32%	27	30	22	18.52%	26.67%	122.00%	90.37%
t481	666	245	679	-1.95%	-177.14%	20	19	16	20.00%	15.79%	103.69%	37.03%
x3	682	807	604	11.44%	25.15%	18	17	16	11.11%	5.88%	137.69%	77.83%
average:	1137	1263	1057	5.15%	3.58%	29	33	24	16.18%	23.53%	119.30%	5.15%

Table 7.2: Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of levels with SIS.

benchmark	V_s	V_g	$\frac{V_g - V_s}{V_g}$
C1355	482	469	-2.77%
C2670	573	908	36.89%
C3540	982	1082	9.24%
C5315	1450	1828	20.68%
C6288	1873	1948	3.85%
C7552	1664	2233	25.48%
alu4	604	947	36.22%
apex6	617	794	22.29%
dalu	745	1588	53.09%
des	2980	4321	31.03%
frg2	709	1623	56.32%
i10	1731	2877	39.83%
i7	505	876	42.35%
i8	813	3439	76.36%
i9	513	790	35.06%
k2	733	2228	67.10%
pair	1412	1738	18.76%
rot	497	919	45.92%
t481	369	1649	77.62%
x3	590	1132	47.88%
average:	992	1669	37.16%

Table 7.3: Comparison of the presented simulated annealing at AND/INVERTER graph level procedure targeting number of vertices with the previously presented NPN-based greedy optimization algorithm.

Chapter 8

Combining technology independent and dependent optimization objectives

A conventional logic synthesis flow is composed of three separate phases: technology independent optimization, technology mapping, and technology dependent optimization. A fundamental problem with such a three-phased approach is that the global logic structure is decided during the first phase without any knowledge of the actual technology parameters considered during later phases. Although technology dependent optimization algorithms perform some limited logic restructuring, they cannot recover from fundamental mistakes made during the first phase, which often results in non-satisfiable solutions.

In this chapter, we present our first attempt to combine the three synthesis phases using simulated annealing as optimization engine. The problem with the simplistic objective function (mentioned in Section 7.5) is brought closer to a solution by adding the capability of a smooth trade-off between area optimization and delay optimization.

8.1 Combined optimization algorithm

The overall synthesis flow of the combined optimization approach is shown in Figure 8.1. The input parameters to the algorithm COMBINEDOPTIMIZATION, is the AND/INVERTER graph to be optimized, s_0 , and the trade-off

parameter for the objective function, x . The algorithm will return the best mapped circuit it finds according to the objective function.

Our implementation is tightly integrated into the synthesis tool SIS, taking advantage of both their technology mapper as well as their algebraic optimization procedure. The steps for algebraic optimization, ALGEBRAIC and technology mapping, TECHMAP is directly used from the SIS infrastructure by applying *script algebraic* and the SIS command *map*, respectively. The technology mapper provided by SIS is a simple tree-based technology mapper that can map for optimal area or delay.

The $\text{COST}()$ procedure returns the cost of the mapped circuit as a pair of its area and delay with respect to the input parameter x . Both the area and the delay is retrieved from SIS.

Annealing optimization

The pseudo-code for the used annealing algorithm, ANNEALING, is shown in Figure 7.5 in Section 7.3. The only difference from the AND/INVERTER graph-based annealing algorithm presented in the previous chapter is the objective function. The new definition of the objective function is as follows:

$$\Delta\epsilon_{ij}(x) = (1 - x) \cdot \Delta n_{ij} + x \cdot \Delta l_{ij} \quad (8.1)$$

where $\Delta\epsilon_{ij} = \epsilon(s_i) - \epsilon(s_j)$ is the difference in costs between s_i and s_j , $\Delta n_{ij} = |V_{s_i}| - |V_{s_j}|$ is the change in number of vertices between s_i and s_j , Δl_{ij} is the change in their level slack. The level slack is calculated in accordance with Definition 1 in Section 4.1. The values of Δn_{ij} and Δl_{ij} are scaled to $[-1, 1]$. The control parameter x is used to trade-off between minimization of the number of vertices and minimization of the number of levels. The parameter x is in the range $[0, 1]$. The value 0 results in minimization of the number of vertices without any consideration to the number of levels. The value 1 results in minimization of the number of levels without any consideration to the number of vertices.

Greedy optimization

The procedure for greedy optimization, GREEDYSEARCH exhaustively applies transformations that reduce the cost of the AND/INVERTER graph according to the given objective. It uses the same objective function as the annealing algorithm, the same trade-off parameter x and works on the same

```

algorithm COMBINEDOPTIMIZATION( $s_0, x$ )
/*  $s_0$  is the initial state represented by an AND/INVERTER graph */
/*  $x \in [0, 1]$  specifies the trade-off between area and delay optimization */
 $s := \text{ALGEBRAIC}(s_0)$ ;
/* execute SIS script algebraic */
 $best\_c := \text{TECHMAP}(s)$ ;
/* map  $s$  with SIS technology mapper */
 $best\_cost := \text{COST}(\text{AREA}(c_0), \text{DELAY}(c_0), x)$ ;
/* cost of mapped circuit in terms of area and delay */
 $s := \text{GREEDYSEARCH}(s, x)$ ;
/* perform a simple greedy optimization */
 $c := \text{TECHMAP}(s)$ ;
 $cost := \text{COST}(\text{AREA}(c), \text{DELAY}(c), x)$ ;
if  $cost < best\_cost$  then
     $best\_c := c$ ;
     $best\_cost := cost$ ;
while improvement do
     $s = \text{ANNEALING}(s, x)$ 
    /* execute the annealing algorithm defined in Chapter 6 */
     $c := \text{TECHMAP}(s)$ ;
     $cost := \text{COST}(\text{AREA}(c), \text{DELAY}(c), x)$ ;
    if  $cost < best\_cost$  then
         $best\_c := c$ ;
         $best\_cost := cost$ ;
 $s := \text{GREEDYSEARCH}(s, x)$ ;
 $c := \text{TECHMAP}(s)$ ;
 $cost := \text{COST}(\text{AREA}(c), \text{DELAY}(c), x)$ ;
if  $cost < best\_cost$  then
     $best\_c := c$ ;
     $best\_cost := cost$ ;
return  $best\_c$ ;
end

```

Figure 8.1: Pseudo-code of the presented algorithm, combining simulated annealing, greedy optimization and technology mapping.

AND/INVERTER graph. All the replacement rules available to the annealing algorithm are also available for the greedy optimization algorithm. However, the greedy algorithm will only identify and apply transforms that reduce the cost.

For example, targeting minimization of the number of vertices, rule 1(a) of Definition 3 in Section 7.2 will be applied to the AND/INVERTER graph displayed in Figure 8.2(a) resulting in the graph shown in Figure 8.2(b). Since the AND vertex representing the function $x \cdot y$ already exists and is used by output f , the vertex can be used by output g as well after applying rule 1(a) of Definition 3 to vertex f . After the transformation, the AND vertex representing the function $y \cdot z$ can be removed reducing the total number of vertices in the graph by one.

However, targeting minimization of the number of levels, the same rule will not be applied to the same vertex given the levels in Figure 8.3(a), because that would increase the level of g from 4 to 5 in the AND/INVERTER graph as shown in Figure 8.3(b).

Figure 8.2: An example of how the number of vertices in an AND/INVERTER graph can change by applying rule 1(a) of Definition 3.

8.2 Experimental results

To evaluate the quality and performance of the presented optimization approach, we performed a set of experiments using all circuits with more than 200 AND vertices in the IWLS 2002 benchmark set [93]. All experiments were run on an AMD Athlon-based PC with a single 1.3GHz CPU, 256kB

Figure 8.3: An example of how the number of levels in an AND/INVERTER graph can change by applying rule 1(a) of Definition 3. The numbers represent the level of the corresponding vertex.

cache, and 224MB RAM operated by RedHat Linux, kernel version 2.4.18. For the technology mapping, we based all experiments on the *lib2.genlib* standard cell library (see Section 3.4).

Our main objective was a comparison with SIS *script algebraic* and SIS *script rugged*.

As shown in Figure 8.1, our synthesis flow is based on first using *script algebraic* followed by the annealing-based optimization. The goal is three-fold:

- First, we aim at synthesizing circuits of a quality that is comparable to the one produced by *script rugged* with significantly less computational effort.
- Second, our combined annealing-based approach is fundamentally scalable and targeted at large circuits that cannot be handled by *script rugged*.
- Third, we aim at complex optimization functions, which can smoothly trade-off multiple objectives.

In a first experiment we run all benchmark circuits through the two SIS scripts, *algebraic* and *rugged*, and compared them with our approach using three different trade-offs for area and delay minimization. For measuring area and delay, each flow mapped the optimized circuits onto the same standard cell library using the SIS command *map*. The optimization

time in SIS was measured with the SIS command *time*. Only the execution time of the scripts are included in the reported numbers. The combined optimization algorithm was timed with the C command *clock()*.

The results of this experiment are reported in Tables 8.1, 8.2, and 8.3 for area-only optimization ($x = 0$), optimization for equally weighted area and delay ($x = 0.5$), and pure delay optimization ($x = 1$), respectively. The individual values for area and delay (in terms of the most negative slack) are reported in the columns A_i and D_i , where $i = a, r, s$ indicate the values for *script algebraic*, *script rugged*, and our simulated annealing-based algorithm, respectively. The last two columns of the area and delay sections provide a relative comparison of the simulated annealing approach with *script algebraic* and *script rugged*. A similar relative comparison is provided in the *time* section for the corresponding CPU execution time. The sign “-” indicates that *script rugged* did not terminate for this circuit.

For area-only optimization (Table 8.1) our results show that the annealing-based approach can further improve the area of circuits optimized with algebraic methods by 8.0% on average. The average execution time overhead for this is 82.6% in addition to executing *script algebraic*. When compared with *script rugged*, comparable area results can be achieved with the annealing-based approach by using only 78.7% of the execution time of *script rugged*. Interestingly, our method also improves the timing for these circuits by 8.4% on average. Moreover, the annealing method can be used robustly for large circuits for which *script rugged* is not applicable.

When targeting pure delay optimization (Table 8.3), an average slack improvement of 12.3% over *script algebraic* is achieved with an additional area improvement of 4.4%. The corresponding slack improvement over *script rugged* is 16.0% by sacrificing 3.5% of the area. Finally, the results for a balanced area and delay optimization ($x = 0.5$) are reported in Table 8.2. Here our algorithm further improves the area by 6.4% and the slack by 7.9% over *script algebraic*. The execution time overhead for doing this is 42.1%. A trade-off between area and delay increases the area obtain with *script rugged* by 1.2% while the slack is improved by 12.4%. The annealing-based approach requires only 61.4% of the execution time required by *script rugged*.

Overall, the results demonstrate clearly that our annealing-based algorithm can be easily controlled to trade-off multiple objective functions such as area and delay. They further show that the approach is applicable for large circuits that cannot be handled by traditional Boolean methods em-

ployed by *script rugged*.

In a second experiment, we evaluate the robustness of the annealing-based method. Recall from Chapter 6, that, theoretically, simulated annealing will yield the optimal solution for a given problem, provided that the control parameter T is lowered in an arbitrarily slow manner. However, this property is only of theoretical value whereas practical implementations need to compromise between the execution time and the optimization quality. A good measure for the robustness of the annealing in the given application is its capability to produce a smooth trade-off between area and delay optimization for varying values of x in the objective given by Equation 8.1. A robust algorithm should produce no or only a few dominating solutions, i.e., solutions that are better than others for both objectives, area and delay.

For this experiment we applied the algorithm for six different values of x on eight randomly selected benchmarks and plotted the area/delay trade-off curves shown in Figure 8.4. A dominated solution for a given design corresponds to a left jog in its curve, indicating that the corresponding result is better in terms of both objectives. As shown, only two out of the 48 solutions dominate others. This implies that the presented simulated annealing algorithm produces high quality results in a reliable manner. When refining the algorithm in the next chapter, a goal is to remove all dominated solutions.

8.3 Conclusion

A framework combining technology dependent optimization objectives with technology independent optimization with the help of a technology mapper was presented. The goal was to be able to optimize for actual technology parameters, such as area and delay of the mapped circuit, instead of graph characteristics e.g., vertex count. However, since the technology mapping is done using SIS technology mapper we have little control over how our technology independent optimization is affecting the mapped circuit. A stronger integration would require more insight in the technology mapping. In order to achieve this we implemented our own technology mapper and introduced the notion of dynamic weights. The resulting advanced annealing algorithm with integrated technology mapping is presented in the next chapter.

benchmark	area					delay					time	
	A_a	A_r	A_s	$\frac{A_a-A_s}{A_a}$	$\frac{A_r-A_s}{A_r}$	D_a	D_r	D_s	$\frac{D_a-D_s}{D_a}$	$\frac{D_r-D_s}{D_r}$	$\frac{t_a+t_s}{t_a}$	$\frac{t_a+t_s}{t_r}$
9symml	189776	157760	190704	-0.49%	-20.88%	-17.11	-22.31	-16.72	2.28%	25.06%	230.0%	19.2%
C1355	493696	424096	499264	-1.13%	-17.72%	-26.7	-25.42	-27.61	-3.41%	-8.62%	343.3%	33.2%
C1908	445904	443584	425024	4.68%	4.18%	-38.5	-39.42	-37.23	3.30%	5.56%	212.0%	29.4%
C2670	656560	618976	583248	11.17%	5.77%	-29.16	-33.88	-30.05	-3.05%	11.30%	213.8%	57.0%
C3540	1047248	1055600	986464	5.80%	6.55%	-49.36	-51.93	-49.61	-0.51%	4.47%	179.5%	39.9%
C432	212976	177712	166112	22.00%	6.53%	-44.53	-38.77	-35.23	20.88%	9.13%	200.0%	0.8%
C499	438016	424096	443584	-1.27%	-4.60%	-30.94	-25.42	-32.62	-5.43%	-28.32%	290.0%	27.2%
C5315	1514960	1379472	1479696	2.33%	-7.27%	-38.23	-38.43	-36.52	4.47%	4.97%	174.1%	123.8%
C6288	2307472	2755232	2401664	-4.08%	12.83%	-105	-143.43	-101.34	3.49%	29.35%	176.1%	63.9%
C7552	2004016	1868992	1881984	6.09%	-0.70%	-70.97	-77.92	-85.18	-20.02%	-9.32%	146.8%	26.6%
C880	372128	358672	328048	11.85%	8.54%	-28.71	-44.45	-32.22	-12.23%	27.51%	196.0%	108.9%
alu2	363776	301600	320624	11.86%	-6.31%	-44.42	-45.95	-39.47	11.14%	14.10%	168.3%	9.8%
alu4	705280	592064	625008	11.38%	-5.56%	-50.19	-46.26	-45.7	8.95%	1.21%	128.1%	8.7%
apex6	687184	650064	660736	3.85%	-1.64%	-19.13	-26.74	-19.22	-0.47%	28.12%	270.0%	162.0%
apex7	228288	206480	212048	7.11%	-2.70%	-15.94	-16.88	-15.02	5.77%	11.02%	240.0%	160.0%
dalv	1139120	751216	710848	37.60%	5.37%	-64.27	-47.63	-46.93	26.98%	1.47%	132.0%	14.6%
des	2980736	2871232	2886544	3.16%	-0.53%	-120.35	-126.36	-88.49	26.47%	29.97%	126.1%	83.7%
example2	324800	294176	268192	17.43%	8.83%	-17.41	-20.47	-15.19	12.75%	25.79%	172.0%	172.0%
f51m	118784	82128	116000	2.34%	-41.24%	-11.64	-29.21	-12.06	-3.61%	58.71%	170.0%	170.0%
frg1	116000	125280	111824	3.60%	10.74%	-14.29	-15.12	-12.15	14.98%	19.64%	136.7%	37.3%
frg2	800400	672336	747040	6.67%	-11.11%	-35.62	-33.61	-34.51	3.12%	-2.68%	154.5%	54.8%
i10	2112592	1944624	2012832	4.72%	-3.51%	-72.09	-65.92	-63.81	11.49%	3.20%	135.9%	7.9%
i2	174464	171680	172144	1.33%	-0.27%	-9.1	-10.54	-9	1.10%	14.61%	182.5%	38.4%
i4	182816	-	182816	0.00%	-	-7.3	-	-8.31	-13.84%	-	100.4%	-
i5	210656	210656	200448	4.85%	4.85%	-15.01	-15.01	-16.07	-7.06%	-7.06%	205.0%	136.7%
i6	435232	345680	404608	7.04%	-17.05%	-33.4	-33.78	-32.1	3.89%	4.97%	220.0%	157.1%
i7	542416	532672	443120	18.31%	16.81%	-34.88	-31.99	-32.92	5.62%	-2.91%	217.1%	152.0%
i8	840304	887632	824528	1.88%	7.11%	-61.76	-40.77	-62.58	-1.33%	-53.50%	152.0%	65.5%
i9	516896	505760	485808	6.01%	3.94%	-41.99	-45.91	-37.37	11.00%	18.60%	200.0%	94.1%
k2	910832	1011056	876960	3.72%	13.26%	-30.14	-30.88	-28.84	4.31%	6.61%	116.3%	52.8%
pair	1513104	1385504	1501504	0.77%	-8.37%	-30.79	-39.34	-28.26	8.22%	28.16%	174.7%	98.1%
rot	629184	610624	602736	4.20%	1.29%	-30.55	-28.68	-30.38	0.56%	-5.93%	183.0%	91.5%
t481	665840	612016	408784	38.61%	33.21%	-24.72	-26.36	-23.73	4.00%	9.98%	106.4%	34.9%
term1	208336	150800	160080	23.16%	-6.15%	-18.32	-11.8	-14.68	19.87%	-24.41%	147.5%	39.3%
ttt2	182816	194416	175856	3.81%	9.55%	-16.74	-17.63	-17.17	-2.57%	2.61%	195.0%	97.5%
x1	282576	266800	266800	5.58%	0.00%	-11.05	-14.96	-11.28	-2.08%	24.60%	190.0%	126.7%
x3	649136	659344	603200	7.08%	8.52%	-16.59	-22.63	-15.65	5.67%	30.84%	180.8%	144.7%
x4	347072	346144	303920	12.43%	12.20%	-18.49	-18.04	-17.72	4.16%	1.77%	173.3%	173.3%
average:	725037	703951	675547	8.04%	0.66%	-35.41	-37.94	-33.24	3.92%	8.40%	182.6%	78.7%

Table 8.1: Comparison of the presented combined optimization algorithm with SIS for $x = 0$ in cost function given by Equation 8.1.

benchmark	area					delay					time	
	A_a	A_r	A_s	$\frac{A_a - A_s}{A_a}$	$\frac{A_r - A_s}{A_r}$	D_a	D_r	D_s	$\frac{D_a - D_s}{D_a}$	$\frac{D_r - D_s}{D_r}$	$\frac{t_a + t_s}{t_a}$	$\frac{t_a + t_s}{t_r}$
9symml	189776	157760	194880	-2.69%	-23.53%	-17.11	-22.31	-15.85	7.36%	28.96%	155.0%	12.9%
C1355	493696	424096	505760	-2.44%	-19.26%	-26.7	-25.42	-25.73	3.63%	-1.22%	230.0%	22.3%
C1908	445904	443584	434304	2.60%	2.09%	-38.5	-39.42	-35.17	8.65%	10.78%	152.0%	21.1%
C2670	656560	618976	593456	9.61%	4.12%	-29.16	-33.88	-28.52	2.19%	15.82%	157.5%	42.0%
C3540	1047248	1055600	1011520	3.41%	4.18%	-49.36	-51.93	-48.12	2.51%	7.34%	137.0%	30.4%
C432	212976	177712	166112	22.00%	6.53%	-44.53	-38.77	-35.23	20.88%	9.13%	145.0%	0.6%
C499	438016	424096	449152	-2.54%	-5.91%	-30.94	-25.42	-28.61	7.53%	-12.55%	200.0%	18.8%
C5315	1514960	1379472	1492688	1.47%	-8.21%	-38.23	-38.43	-35.77	6.43%	6.92%	135.9%	96.7%
C6288	2307472	2755232	2401664	-4.08%	12.83%	-105	-143.43	-101.34	3.49%	29.35%	129.8%	47.1%
C7552	2004016	1868992	1901472	5.12%	-1.74%	-70.97	-77.92	-80.28	-13.12%	-3.03%	122.6%	22.2%
C880	372128	358672	367952	1.12%	-2.59%	-28.71	-44.45	-27.84	3.03%	37.37%	146.0%	81.1%
alu2	363776	301600	335936	7.65%	-11.38%	-44.42	-45.95	-40.46	8.91%	11.95%	135.0%	7.9%
alu4	705280	592064	630112	10.66%	-6.43%	-50.19	-46.26	-44.92	10.50%	2.90%	113.4%	7.7%
apex6	687184	650064	660736	3.85%	-1.64%	-19.13	-26.74	-19.22	-0.47%	28.12%	183.3%	110.0%
apex7	228288	206480	214368	6.10%	-3.82%	-15.94	-16.88	-14.19	10.98%	15.94%	160.0%	106.7%
dalv	1139120	751216	718272	36.95%	4.39%	-64.27	-47.63	-44.81	30.28%	5.92%	117.7%	13.0%
des	2980736	2871232	2886544	3.16%	-0.53%	-120.35	-126.36	-88.49	26.47%	29.97%	118.3%	78.6%
example2	324800	294176	268192	17.43%	8.83%	-17.41	-20.47	-15.19	12.75%	25.79%	136.0%	136.0%
f51m	118784	82128	118320	0.39%	-44.07%	-11.64	-29.21	-11.31	2.84%	61.28%	130.0%	130.0%
frg1	116000	125280	114608	1.20%	8.52%	-14.29	-15.12	-11.42	20.08%	24.47%	120.0%	32.7%
frg2	800400	672336	747040	6.67%	-11.11%	-35.62	-33.61	-34.51	3.12%	-2.68%	130.9%	46.5%
i10	2112592	1944624	2042992	3.29%	-5.06%	-72.09	-65.92	-59.46	17.52%	9.80%	119.8%	6.9%
i2	174464	171680	172144	1.33%	-0.27%	-9.1	-10.54	-9	1.10%	14.61%	137.5%	28.9%
i4	182816	-	182816	0.00%	-	-7.3	-	-8.31	-13.84%	-	100.2%	-
i5	210656	210656	200448	4.85%	4.85%	-15.01	-15.01	-16.07	-7.06%	-7.06%	155.0%	103.3%
i6	435232	345680	413424	5.01%	-19.60%	-33.4	-33.78	-31.68	5.15%	6.22%	176.0%	125.7%
i7	542416	532672	443120	18.31%	16.81%	-34.88	-31.99	-32.92	5.62%	-2.91%	161.4%	113.0%
i8	840304	887632	843552	-0.39%	4.97%	-61.76	-40.77	-53.71	13.03%	-31.74%	129.2%	55.7%
i9	516896	505760	517360	-0.09%	-2.29%	-41.99	-45.91	-32.34	22.98%	29.56%	171.3%	80.6%
k2	910832	1011056	952592	-4.58%	5.78%	-30.14	-30.88	-25.52	15.33%	17.36%	108.7%	49.4%
pair	1513104	1385504	1514496	-0.09%	-9.31%	-30.79	-39.34	-26.28	14.65%	33.20%	137.5%	77.2%
rot	629184	610624	623616	0.88%	-2.13%	-30.55	-28.68	-26.84	12.14%	6.42%	140.0%	70.0%
t481	665840	612016	408784	38.61%	33.21%	-24.72	-26.36	-23.73	4.00%	9.98%	105.3%	34.6%
term1	208336	150800	165648	20.49%	-9.85%	-18.32	-11.8	-14.67	19.92%	-24.32%	125.0%	33.3%
ttt2	182816	194416	177248	3.05%	8.83%	-16.74	-17.63	-16.5	1.43%	6.41%	150.0%	75.0%
x1	282576	266800	270512	4.27%	-1.39%	-11.05	-14.96	-11.28	-2.08%	24.60%	140.0%	93.3%
x3	649136	659344	605984	6.65%	8.09%	-16.59	-22.63	-15.55	6.27%	31.29%	139.2%	111.3%
x4	347072	346144	303920	12.43%	12.20%	-18.49	-18.04	-17.72	4.16%	1.77%	150.0%	150.0%
average:	725037	703951	685572	6.36%	-1.19%	-35.41	-37.94	-31.80	7.85%	12.37%	142.1%	61.4%

Table 8.2: Comparison of the presented combined optimization algorithm with SIS for $x = 0.5$ in cost function given by Equation 8.1.

benchmark	area					delay					time	
	A_a	A_r	A_s	$\frac{A_a-A_s}{A_a}$	$\frac{A_r-A_s}{A_r}$	D_a	D_r	D_s	$\frac{D_a-D_s}{D_a}$	$\frac{D_r-D_s}{D_r}$	$\frac{t_a+t_s}{t_a}$	$\frac{t_a+t_s}{t_r}$
9symml	189776	157760	194880	-2.69%	-23.53%	-17.11	-22.31	-15.85	7.36%	28.96%	275.0%	22.9%
C1355	493696	424096	527104	-6.77%	-24.29%	-26.7	-25.42	-24.98	6.44%	1.73%	483.3%	46.8%
C1908	445904	443584	434304	2.60%	2.09%	-38.5	-39.42	-35.17	8.65%	10.78%	258.0%	35.8%
C2670	656560	618976	631040	3.89%	-1.95%	-29.16	-33.88	-28.06	3.77%	17.18%	265.0%	70.7%
C3540	1047248	1055600	1022192	2.39%	3.16%	-49.36	-51.93	-44.04	10.78%	15.19%	214.0%	47.6%
C432	212976	177712	183744	13.73%	-3.39%	-44.53	-38.77	-33.43	24.93%	13.77%	250.0%	1.1%
C499	438016	424096	449152	-2.54%	-5.91%	-30.94	-25.42	-28.61	7.53%	-12.55%	396.7%	37.2%
C5315	1514960	1379472	1492688	1.47%	-8.21%	-38.23	-38.43	-35.77	6.43%	6.92%	209.1%	148.7%
C6288	2307472	2755232	2401664	-4.08%	12.83%	-105	-143.43	-101.34	3.49%	29.35%	187.8%	68.1%
C7552	2004016	1868992	1918176	4.28%	-2.63%	-70.97	-77.92	-71.36	-0.55%	8.42%	168.9%	30.6%
C880	372128	358672	373984	-0.50%	-4.27%	-28.71	-44.45	-26.92	6.23%	39.44%	236.0%	131.1%
alu2	363776	301600	320624	11.86%	-6.31%	-44.42	-45.95	-39.47	11.14%	14.10%	200.0%	11.7%
alu4	705280	592064	665804	5.60%	-12.45%	-50.19	-46.26	-38.59	23.11%	16.58%	142.5%	9.7%
apex6	687184	650064	703424	-2.36%	-8.21%	-19.13	-26.74	-17.19	10.14%	35.71%	343.3%	206.0%
apex7	228288	206480	214368	6.10%	-3.82%	-15.94	-16.88	-14.19	10.98%	15.94%	285.0%	190.0%
dalv	1139120	751216	736368	35.36%	1.98%	-64.27	-47.63	-39.96	37.82%	16.10%	151.4%	16.7%
des	2980736	2871232	2886544	3.16%	-0.53%	-120.35	-126.36	-88.49	26.47%	29.97%	150.8%	100.1%
example2	324800	294176	268192	17.43%	8.83%	-17.41	-20.47	-15.19	12.75%	25.79%	206.0%	206.0%
f51m	118784	82128	126208	-6.25%	-53.67%	-11.64	-29.21	-10.93	6.10%	62.58%	180.0%	180.0%
frg1	116000	125280	114608	1.20%	8.52%	-14.29	-15.12	-11.42	20.08%	24.47%	156.7%	42.7%
frg2	800400	672336	799472	0.12%	-18.91%	-35.62	-33.61	-31.42	11.79%	6.52%	196.4%	69.7%
i10	2112592	1944624	2055984	2.68%	-5.73%	-72.09	-65.92	-51.99	27.88%	21.13%	156.8%	9.1%
i2	174464	171680	172144	1.33%	-0.27%	-9.1	-10.54	-9	1.10%	14.61%	225.0%	47.4%
i4	182816	-	184672	-1.02%	-	-7.3	-	-5.98	18.08%	-	100.6%	-
i5	210656	210656	200448	4.85%	4.85%	-15.01	-15.01	-16.07	-7.06%	-7.06%	255.0%	170.0%
i6	435232	345680	431056	0.96%	-24.70%	-33.4	-33.78	-31.66	5.21%	6.28%	334.0%	238.6%
i7	542416	532672	443120	18.31%	16.81%	-34.88	-31.99	-32.92	5.62%	-2.91%	287.1%	201.0%
i8	840304	887632	862576	-2.65%	2.82%	-61.76	-40.77	-52.7	14.67%	-29.26%	189.2%	81.6%
i9	516896	505760	517360	-0.09%	-2.29%	-41.99	-45.91	-32.34	22.98%	29.56%	316.3%	148.8%
k2	910832	1011056	1018016	-11.77%	-0.69%	-30.14	-30.88	-24.84	17.58%	19.56%	126.8%	57.6%
pair	1513104	1385504	1514496	-0.09%	-9.31%	-30.79	-39.34	-26.28	14.65%	33.20%	211.3%	118.6%
rot	629184	610624	623616	0.88%	-2.13%	-30.55	-28.68	-26.84	12.14%	6.42%	219.0%	109.5%
t481	665840	612016	439872	33.94%	28.13%	-24.72	-26.36	-20.14	18.53%	23.60%	111.0%	36.5%
term1	208336	150800	170752	18.04%	-13.23%	-18.32	-11.8	-13.86	24.34%	-17.46%	170.0%	45.3%
ttt2	182816	194416	184672	-1.02%	5.01%	-16.74	-17.63	-13.68	18.28%	22.40%	245.0%	122.5%
x1	282576	266800	274688	2.79%	-2.96%	-11.05	-14.96	-10.48	5.16%	29.95%	225.0%	150.0%
x3	649136	659344	605984	6.65%	8.09%	-16.59	-22.63	-15.55	6.27%	31.29%	215.0%	172.0%
x4	347072	346144	321088	7.49%	7.24%	-18.49	-18.04	-17.46	5.57%	3.22%	243.3%	243.3%
average:	725037	703951	696976	4.35%	-3.49%	-35.41	-37.94	-30.37	12.28%	15.99%	226.0%	98.0%

Table 8.3: Comparison of the presented combined optimization algorithm with SIS for $x = 1$ in cost function given by Equation 8.1.

Figure 8.4: Area/delay trade-off for different settings of variable x in the cost function given by Equation 8.1 for the presented combined optimization algorithm.

Chapter 9

Advanced simulated annealing with integrated technology mapping

The algorithms presented in Chapters 5, 7, and 8 use technology-independent objective functions, such as the number of literals or the length of the critical path in terms of number of vertices in the AND/INVERTER graph, without considering actual technology parameters. A fundamental problem with such an approach is that the objective is too crude and does not include any feedback from realistic values for delay, area, and power and their strong dependency on the circuit structure. In Chapter 8, we were trying to combine technology dependent optimization objectives with technology independent optimization by letting the algorithm return the circuit with best cost *after* technology mapping. However, there were no feedback from the mapped circuit to the technology independent objective function.

In this chapter, we present a simulated annealing-based approach to logic optimization using dynamic weights for the objective function. The weights are updated on regular intervals by mapping the AND/INVERTER graph onto a selected library of gates. The dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels.

9.1 Previous work

In [143] simulated annealing is applied to a NAND/NOR network and the subsequent technology mapping. The search space consists of all functionally equivalent NAND or NOR logic representations at the technology independent level. Technology dependent logic representations consisting of gates from a technology library in addition to all primitive and hybrid representations compromise the search space at the technology mapping.

A number of NAND/NOR transformations are identified and implemented. For example, one transformation removes an even number of inverters and makes the necessary logic adjustments if the signal at the output of an odd numbered inverter feeds additional locations in the logic. Another transformation removes “pre-anding” logic and moves the source signals directly to the sink primitives. An additional transformation removes specified redundancies. Similarly, transformations exist to map from a NAND or a NOR to each gate of the target technology and from each gate to its equivalent NAND or NOR representation.

Each transformation in the process computes its effect on a set of weighted logic variables which compromise the objective function. The objective function’s terms pertaining to the NAND/NOR transformations are as follows:

- Node count change.
- The algebraic maximum path length change within the operating area.
- Path length summation change within the operating area.
- Connection count change.
- The change in a fanout balance factor within the operating area.
- The change in total fanout within the operating area.
- The change in a fanin balance factor within the operating area.
- The change in total fanin within the operating area.

The balance factors refer to the average deviation from the mean of the quantity in the transformation’s operating area. Intermediate tables exist to store cost information required to evaluate the terms listed above. A fanout table holds source and target fanout data for all signals involved in

the transformation. Other tables which are completed by the cost routines in a manner similar to the fanout table, collect intermediate data pertaining to logic levels and fanin.

The terms in the objective function used in the technology mapping phase appear below:

- Node count change.
- The summation of added path delay which exceeds the maximum permitted delay.
- The algebraic maximum path delay change within the operating area.
- Path delay summation change within the operating area.
- Connection count change.
- The summation of added fanout which exceeds the maximum permitted fanout.
- The summation of added fanin which exceeds the maximum permitted fanin.

Since many members of the search space contain both technology independent and technology dependent gates, routines which predict the future mapping of blocks to technology cells compute technology dependent values of technology independent logic blocks in order to measure the cost of a particular transform with either a hybrid source or a hybrid target representation.

A restriction of the simulated annealing method gave comparable results to the full annealing with a substantial performance improvement when applied to the technology mapping phase. This method executes mapping transformations in the best order. The best order is experimentally determined. Each transformation initially loop through the logic and measures the cost of replacing a group of the technology independent logic blocks with a gate available in the target technology. These proposed changes along with the associated reverse transformation expansion are then randomly selected and probabilistically accepted according to simulated annealing theory.

The main differences between the above described approach and the integrated advanced annealing-based algorithm that we will present are twofold:

1. Our objective function is much simpler and therefore much faster to evaluate.

2. Our technology mapper is integrated into the technology independent optimization and hence, the optimization is not done in two separated phases.

9.2 The advanced annealing algorithm

There are two differences between the simulated annealing algorithm presented in Chapter 7 and the advanced annealing algorithm introduced here:

- The advanced annealing provides a tight integration with the technology mapper.
- It supports a more complex objective function that can trade-off between different optimization objectives.

The pseudo-code of the presented advanced simulated annealing is shown in Figure 9.1. The input parameter x is used to control the desired trade-off between area and delay. The value $x = 0$ is for area optimization; the value $x = 1$ is for delay optimization. The input parameter m determines how often the technology mapper will be called. s_0 is the AND/INVERTER graph to be optimized. The ANNEALING algorithm will return the AND/INVERTER graph which corresponds to the best mapped circuit according to the objective function.

For an explanation of the procedures INITIALIZE, SELECT and ACCEPT see Section 7.3.

Objective function

In the advanced annealing implementation, the objective function ϵ depends on three parameters of the AND/INVERTER graph corresponding to the state s :

- Number of vertices.
- Level slack.
- Fanout distribution.

Definition 1 in Section 4.1, defining the level slack of a vertex $v \in V$, $l(v)$, is enhanced with dynamic weights $w(v)$, extracted from the mapped


```

algorithm ANNEALING( $s_0, x, m$ )
/*  $s_0$  is the initial state */
/*  $x \in [0, 1]$  specifies trade-off between area and delay optimization */
/*  $m$  specifies how often the mapper is called */
( $T, T_e, p$ ) = INITIALIZE( $s_0$ );
 $c_0$  = MAPPING( $s_0, x$ );
/*  $c_0$  is the mapped circuit for  $s_0$  */
FEEDBACK( $s_0, c_0$ );
/* extract feedback from mapped circuit */
 $best\_state := s_0$ ;
 $best\_cost := COST(AREA(c_0), DELAY(c_0), x)$ ;
/* cost of the mapped circuit in terms of area and delay */
 $map\_frequency := m$ ;
 $s := s_0$ ;
while  $T > T_e$  do
  for  $p$  times do
     $s' = SELECT(s)$ ;
    /* pick a neighbor of  $s$  */
    if ACCEPT( $s, s', x, T$ ) then
      /* accept the state  $s'$  */
       $s := s'$ ;
    if  $map\_frequency = 0$  then
       $c = MAPPING(s, x)$ ;
      FEEDBACK( $s, c$ );
       $cost := COST(AREA(c), DELAY(c), x)$ ;
      if  $cost < best\_cost$  then
         $best\_state := s$ ;
         $best\_cost := cost$ ;
       $map\_frequency := m$ ;
    else
       $map\_frequency --$ ;
   $T = COOL(T)$ ;
  /* decrease  $T$  */
return  $best\_state$ ;
end

```

Figure 9.1: Pseudo-code of the advanced annealing algorithm with integrated technology mapping.

circuit by the procedure FEEDBACK for each primary output. The extracted weights reflect the timing slack at each primary output $v \in PO$ expressed in terms of number of inverters and will be described in more detail in Section 9.4. The enhanced level slack is defined as follows:

$$l(v) = l_r(v) - l_a(v) \quad (9.1)$$

where

$$l_a(v) = \begin{cases} 1 & \text{if } v \in PI \\ \max_{\forall u \in \text{fanin}(v)} (l_a(u)) + 1 & \text{otherwise} \end{cases}$$

and

$$l_r(v) = \begin{cases} \max_{\forall u \in PO} (l_a(u)) + w(v) & \text{if } v \in PO \\ \min_{\forall u \in \text{fanout}(v)} (l_r(u)) - 1 & \text{otherwise} \end{cases}$$

Since the acceptance probability, α_{ij} , depends only on the difference in costs between the states s_i and s_j , the objective function can be handled as:

$$\Delta \epsilon_{ij}(x) = \Delta n_{ij} + x \cdot \Delta l_{ij} + 0.4 \cdot \Delta o_{ij}$$

where $\Delta \epsilon_{ij} = \epsilon(s_i) - \epsilon(s_j)$ is the difference in costs between s_i and s_j , $\Delta n_{ij} = |V_{s_i}| - |V_{s_j}|$ is the change in number of vertices between s_i and s_j , Δl_{ij} is the change in their level slack according to Equation 9.1, and Δo_{ij} the change in their fanout distribution. The values of Δn_{ij} , Δl_{ij} and Δo_{ij} are scaled to $[-1, 1]$. The control parameter x is used to trade-off delay versus area in the objective function.

The change in fanout distribution, Δo , is a function which penalizes moves that increase the number of vertices with multiple fanout and rewards moves that decrease the number of vertices with multiple fanouts. It is defined as

$$\Delta o_{ij} = |V_{s_i}^{fo>1}| - |V_{s_j}^{fo>1}|$$

where $V_{s_k}^{fo>1} = \{v \in V_{s_k} \mid \text{fanout of } v \text{ is greater than } 1\}$ for any state s_k .

9.3 Technology mapping

The procedure MAPPING() in Figure 9.1 performs technology mapping. We have implemented a simple technology mapper which does tree covering by

```

algorithm OPTIMALAREACOVER( $v, P$ )
/*  $v$  is a graph vertex,  $P$  is set of library patterns */
  for  $u \in \text{fanin}(v)$  do
    OPTIMALAREACOVER( $u, P$ );
   $\text{cost}(v) := \infty$ ;
  for  $m \in \text{match}(v, P)$  do
    /*  $\text{match}(v, P)$  the set of library patterns that match at  $v$  */
     $\text{cost}(v, m) := \text{area}(m) + \sum_{v_i \in \text{inputs}(m)} \text{cost}(v_i)$ ;
    if  $\text{cost}(v, m) < \text{cost}(v)$  then
      /*  $\text{bestMatch}(v)$  stores the best match at  $v$ , and */
      /*  $\text{cost}(v)$  its area */
       $\text{bestMatch}(v) := m$ ;
       $\text{cost}(v) := \text{cost}(v, m)$ ;
  end

```

Figure 9.2: Pseudo-code of the optimal area cover algorithm.

breaking the circuit at each multiple fanout point. The motivation for the tree-based approach is that we rely on the simulated annealing framework to handle the fanout optimization.

As already mention in Section 3.2, the technology mapping step in the synthesis flow converts a subject graph composed of logic primitives into an implementation composed of library cells. The mapping typically consists of two phases:

1. Enumerating subgraphs and *matching* them against library cells.
2. Performing a *covering* of the graph with the found matches.

Structural matching [60] applies pattern graphs for the library to match against the enumerated subgraphs, but require storing many patterns for each library cell. Instead, the authors of [144] proposed the use of Boolean matching in the first mapping phase. This technique compares parts of the subject graph with the library by their associated functions, thus providing a general technique for matching arbitrary subgraphs. For a given part of the subject graph, a Boolean matcher finds all library cells that are equivalent. A generalized matching criteria considers, besides permutating function inputs and outputs, also their negations. Recall from Section 2.5, that two functions

```

algorithm OPTIMALDELAYCOVER( $v, P$ )
/*  $v$  is a graph vertex,  $P$  is set of library patterns */
  for  $u \in \text{fanin}(v)$  do
    OPTIMALDELAYCOVER( $u, P$ )
  for  $\gamma \in \Gamma(v)$  do
/*  $\Gamma(v)$  is the set of possible load intervals at  $v$  */
     $\text{cost}(v, \gamma) := \infty$ ;
    for  $m \in \text{match}(v, P), \gamma \in \Gamma(v)$  do
      if  $\text{cost}(v, \gamma, m) < \text{cost}(v, \gamma)$  then
        /*  $\text{match}(v, \gamma)$  stores the best match at  $v$  for  $\gamma$  */
        /*  $\text{cost}(v, \gamma)$  stores propagation delay up to  $v$  */
         $\text{match}(v, \gamma) := m$ ;
         $\text{cost}(v, \gamma) := \text{cost}(v, \gamma, m)$ ;
    end
  end

```

Figure 9.3: Pseudo-code of the optimal delay cover algorithm.

are NPN-equivalent if one can be obtained from the other by negation and/or permutation of the inputs and outputs. The dominant approach for Boolean matching is to compute for each library cell a canonical (or semi-canonical) form that is an invariant with respect to input/output and store it in a hash table [145, 146, 147, 148, 149]. To attempt a match for a subgraph, the matcher computes the canonical form for the associated function and looks it up in a hash table. Matchers based on canonical forms are particularly suited to large cell libraries, because the execution time is independent of the size of the library. The pattern matching technique used by our technology mapper is the fast Boolean matcher based on canonical forms presented in [150].

The tree covering algorithms for area and delay are implemented in a standard way [11]. The key property that makes optimal tree covering for area algorithms efficient is that given a match m at a vertex in the AND/INVERTER graph, the costs of the best matches at the inputs to m are independent of each other. In addition, the best match at a vertex is independent of the best matches at the vertices in its transitive fanout.

It follows that the minimum cost match m at a root of a tree is the match that minimizes $\text{area}(m) + \sum_{v_i \in \text{inputs}(m)} \text{minArea}(v_i)$, where the v_i are vertices in

the AND/INVERTER graph which are input to the match m , and $\text{minArea}(v_i)$ is the cost of the minimum cost match at the node v_i . $\text{area}(m)$ is the area of the gate corresponding to match m as given by the cell library. A dynamic programming algorithm to accomplish this traverses the tree from the leaves to the root, determining the best match at each node. When a node is visited, all the matches at that node are enumerated and from among those, the minimum cost match is stored. A recursive formulation of this idea is displayed in Figure 9.2. The input parameter v is the root of the tree to be covered.

While tree covering for delay is similar to tree covering for area in that the costs of the best matches at the inputs to a match are independent of each other, there is a key difference. In fact, the cost of a match can not be determined until matches have been obtained for all its fanout vertices. This is needed to determine the load γ (see Section 3.2).

To make delay mapping amenable to dynamic programming, the choice of the match has to be made independently of the actual output load. This can be accomplished by a two pass approach. In the first pass through the tree from the leaves to the root, each vertex in the AND/INVERTER graph is visited once and the minimum cost match at the vertex is stored for each possible load interval. We use intervals of load values rather than single load values [151]. The cost in this case is the delay at the vertex, corresponding to the input pin with the latest arrival time. That is, given a load γ , the cost of a match m for load γ is given by $\max_{v_i \in \text{inputs}(m)} (\alpha_{i,m} + \beta_{i,m}\gamma + \text{cost}(v_i, \gamma_{i,m}))$, where i is the pin of the gate corresponding to match m that connects to v_i . $\gamma_{i,m}$ is the load presented to v_i by m , and $\text{cost}(v_i, \gamma_{i,m})$ is the propagation delay of the best match at node v_i for load $\gamma_{i,m}$.

In the second pass from the root to the leaves, the best match is picked for each node visited, based on the matches stored in the first pass. A recursive formulation of this idea is shown Figure 9.3.

The implemented mapper can only map for optimal delay or optimal area, not for a combination of the two. It can be extended to do optimal delay covering with area recovery, as in [63]. Since mapping for area often has a smaller impact on delay than vice versa the impact of the input parameter x on the optimization objective is as follows:

$$\begin{aligned} 0 \leq x \leq 0.6 & \quad \text{map for optimal area,} \\ 0.6 < x \leq 1.0 & \quad \text{map for optimal delay.} \end{aligned}$$

9.4 The feedback mechanisms

The procedure `FEEDBACK()` extracts parameters from the mapped circuit and uses them to update the information at the AND/INVERTER graph level. This dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters. For each primary output, a weight corresponding to how many second fastest inverters in the cell library can be inserted at this primary output without increasing the delay of the mapped circuit is calculated. It is defined for all primary output vertices $v \in PO$ of the AND/INVERTER graph as:

$$w(v) = \frac{\max_{\forall u \in PO} (t_{arrival}(\text{MAP}(u)) - t_{arrival}(\text{MAP}(v)))}{\text{DELAY}(g_{inv})}$$

where $t_{arrival}(g)$ is the arrival time computed by the OA Gear timer for gate g (OA Gear is introduced in Section 3.5). The procedure `MAP(v)` returns the mapped gate for vertex v of the AND/INVERTER graph, and $\text{DELAY}(g_{inv})$ is the delay of the second fastest inverter in the used cell library. In essence, the weight $w(v)$ reflects the slack of the vertex v expressed in terms of number of inverters. Since this is only a crude estimate, the selection of the inverter type does not greatly impact the results. In our implementation, we chose the second fastest inverter of the library to convert actual delay numbers into number of graph levels.

Figure 9.4: AND/INVERTER graph with level slacks. The critical path has level slacks marked in bold.

In Figure 9.4 an AND/INVERTER graph with level slacks according to Definition 1, Section 4.1 is shown. The critical path is from primary output

Figure 9.5: AND/INVERTER graph with level slacks adjusted by dynamic weights. The critical path has level slacks marked in bold.

f to inputs x and y . After technology mapping the required times of the primary outputs are updated according to Equation 9.1. The resulting level slacks for the AND/INVERTER graph in Figure 9.4 are shown in Figure 9.5. The critical path is now from the primary output g to inputs z and w .

The $\text{COST}()$ procedure returns the cost of the mapped circuit as a pair of its area and delay with respect to the input parameter x . The area of the mapped circuit is calculated as a sum of areas of all individual cells in the network, ignoring wire area. The delay of the mapped circuit is computed using the timing analysis tool OA Gear Timer.

9.5 Experimental results

To evaluate the quality and performance of the presented advanced annealing algorithm, we performed a set of experiments using the IWLS'02 benchmark set [93]. All experiments were run on a Pentium 4-based PC with a single 2.0GHz CPU, 512kB cache, and 1GB RAM operated by RedHat Linux, kernel version 2.4.18.

First, we compared our results for area/delay trade-off ($x = 0.5$) to the results of *script rugged* of SIS. For this experiment, we used all IWLS'02 benchmarks with more than 500 vertices. The frequency of the calls to the mapper, m , was set to 1000. In order to avoid the optimization results to be influenced by differences in SIS technology mapper and our technology mapper, we used our mapper for both cases.

Figure 9.6: Area/delay trade-off for different values of x in the objective function $\Delta \epsilon_{ij}(x) = \Delta n_{ij} + x \cdot \Delta l_{ij} + 0.4 \cdot \Delta o_{ij}$, where Δn_{ij} is the change in number of vertices, Δl_{ij} is the change in level slack, and Δo_{ij} is the change in fanout distribution.

Table 9.1 summarizes the results. Columns labeled D_r , A_r , L_r , V_r and t_r stand for delay, area, number of levels, number of vertices and execution time for *script rugged*, respectively. The delay is computed using the timing analysis tool OA Gear Timer. The area is calculated as the sum of areas of individual cells in the mapped circuit, ignoring wiring area. SIS *script rugged* is timed using SIS command *time*. The corresponding columns for the advanced simulated annealing algorithm is labeled D_s , A_s , L_s , V_s and t_s . The annealing algorithm is timed using the standard library function *clock()*.

The last three columns show the improvement for delay and area and the increase in execution time, respectively. Table 9.1 show that, on average, circuits optimized by the presented advanced annealing approach are 32.5% faster and 11.2% smaller than the ones optimized by SIS. However, simulated annealing requires 11.8 times more execution time than *script rugged*.

benchmark	SIS <i>script rugged</i>					advanced annealing					$\frac{D_r - D_s}{D_r} \cdot 100\%$	$\frac{A_r - A_s}{A_r} \cdot 100\%$	time t_s/t_r
	D_r	A_r	L_r	V_r	t_r , sec	D_s	A_s	L_s	V_s	t_s , sec			
C2670	4.87243	18824.4	38	732	4.11	3.81743	16608.2	27	600	39.8	21.65%	11.77%	9.68
C3540	7.74114	42781	64	1860	10.1	6.05482	25933	41	992	105.26	21.78%	39.38%	10.42
C5315	4.98544	36748.1	35	1454	6.09	4.84326	40869.4	33	1464	204.19	2.85%	-11.22%	33.53
C6288	21.4463	78997.7	142	2415	16.03	14.7477	61644.2	93	1925	348.64	31.23%	21.97%	21.75
C7552	7.23733	64294.6	57	2561	36.58	6.98681	50550.5	37	1689	289.66	3.46%	21.38%	7.92
alu4	6.41646	15111.4	55	624	47.36	4.52553	16835	33	613	45.32	29.47%	-11.41%	0.96
apex6	2.38512	17010	25	667	1.79	1.97321	16750.8	18	626	38.81	17.27%	1.52%	21.68
dalu	5.1737	30261.6	26	1313	32.61	3.40549	19219.7	23	807	78.34	34.18%	36.49%	2.40
frg2	3.15653	17755.2	23	670	7.24	2.20026	21902.4	14	737	61.87	30.29%	-23.36%	8.55
i10	8.58305	61916.4	69	2466	153.51	6.36488	51354	44	1755	308.81	25.84%	17.06%	2.01
i2	12.3654	9933.84	184	533	2.42	1.09961	3512.16	14	214	7.1	91.11%	64.64%	2.93
i7	1.82331	14307.8	6	514	1.72	1.27235	14340.2	7	538	39.61	30.22%	-0.23%	23.03
i9	3.59733	12817.4	14	525	2.3	1.8421	13808.9	13	556	40.7	48.79%	-7.74%	17.70
k2	4.94984	22116.2	56	871	18.2	2.61238	19433.5	21	710	57.9	47.22%	12.13%	3.18
pair	5.34324	39748.3	36	1510	7.4	2.96608	38711.5	21	1434	178.33	44.49%	2.61%	24.10
rot	3.95737	17444.2	31	658	2.81	3.0565	14657.8	24	497	25.57	22.76%	15.97%	9.10
t481	5.19488	15377	61	660	27.65	2.61216	15558.5	19	641	48.7	49.72%	-1.18%	1.76
average:											32.49%	11.17%	11.81

Table 9.1: Comparison of the presented algorithm with SIS *script rugged* for $x = 0.5$ and $m = 1,000$.

Figure 9.7: Delay/execution time trade-off for different values of the mapper frequency m .

In the second experiment, we applied the advanced annealing algorithm for six different values of x on five randomly selected big-C benchmarks and plotted the area/delay trade-off curves to check the presence of dominating solutions. Figure 9.6 shows the results. The input parameter m was set to 1000. As one can see, none of 30 solutions dominate others. This attests to the overall robustness of the presented method and provides an indication that our algorithm produces high quality results in a reliable manner.

The third experiment evaluates the integration between the implemented technology mapper and the annealing algorithm. In Figure 9.7, the trade-off curves between delay improvement and execution time for four different values of the mapper frequency m are displayed for five randomly selected big-C benchmarks. A smaller value of m indicates that the mapper is called more often. The input parameter x was set to 1 i.e., the objective was pure delay optimization. Clearly, the tighter the integration with the mapper, the better delay is achieved. However, as our results show, this is only true down to a certain value of m , after which a relatively small gain in delay

causes a significant increase in execution time.

9.6 Conclusion

In this chapter, we presented an optimization approach integrating technology independent optimization steps with technology dependent objectives in an annealing-based framework. Furthermore, we showed that the presented advanced simulated annealing-based approach can smoothly trade-off complex, multiple-dimensional objective functions and achieve results competitive with results obtain from UC Berkeley's publicly available synthesis tool, SIS.

The combination of technology independent and technology dependent objectives is handled through dynamic weighting. Dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels.

Chapter 10

Applications to new emerging technologies

In 1959, the late Nobel Prize-winning physicist Richard P. Feynman presented a talk entitled “There’s Plenty of Room at the Bottom” at the annual meeting of the American Physical Society. Feynman proposed using machine tools to make smaller machine tools, which, in turn, would be used to make still smaller machine tools, and so on all the way down to the molecular level. He suggested that such nanomachines, nanodevices and nanorobots ultimately could be used to develop a wide range of atomically precise microscopic instrumentation and manufacturing tools. He concluded that this is “a development which I think cannot be avoided”. The vision of nanotechnology was born.

Fifty years ago, this talk was greeted with astonishment and skepticism. However since then, there have been a remarkable progress toward Feynman’s vision. From the dawn of the microcomputer era several decades ago, there has been a significant increase in speed and power of computers. This is due, in large measure, to the ever-decreasing size of the electronic components that can be packed at ever-increasing densities onto a single silicon chip. Transistor density has doubled every 18 months, an observation that has come to be known as Moore’s law. The size of features on computer chips has shrunk from a fraction of a millimeter in the first microprocessor chip to 0.1 micrometers in the latest chips. However, the exponential improvement in speed and integration of silicon transistor technology is expected to slow down as devices approach nanometer dimensions [152].

A number of devices based on gating the flow of electrons have been proposed, including quantum dots [153], organic molecules [154], carbon nanotubes [155], nanowires [156], and the motion of single atoms or molecules [157]. Other computation schemes, operating on different principles, include electrons confined in quantum dot cellular automata [158, 159], magnetic dot cellular automata [160], and solutions of interacting DNA molecules [161, 162]. Computation can also be performed by purely mechanical means [163, 164], as in the calculating engine of Babbage [165].

This chapter describes how our developed logic synthesis framework can be applied to two different emerging technologies. The first one is chemically assembled electronic nanotechnology [166], which is expected to offer extremely dense and low-power computational elements built with inexpensive chemical self-assembly. The second one, introduced in [163], performs computation by mechanical means, similarly to the toppling of a row of standing domino.

10.1 Chemically assembled electronic nanotechnology

Chemically Assembled Electronic Nanotechnology (CAEN) devices are on the order of a few nanometers in size [166]. The fabrication process is hierarchical, proceeding from basic components (e.g., wires and switches), through self-assembled arrays of components, to complete systems. In the first step, wires of different types are constructed through chemical self-assembly [167]. One of the newest types of nanoparticle self-assembly is *chemical lithography* [1]. The process can effectively form periodic arrays of very stable nanoparticles that do not succumb to many of the defects and limitations of previous lithography techniques. Chemical lithography is a combination of techniques, where particle arrangement is controlled by differences in reactivity - a characteristic determined by exposing particles and surfaces to an assortment of chemical treatments.

In [1] luminescent yttrium aluminum garnet (YAG) nanoparticles assembled on a silicon wafer was used, synthesizing the particles through doping and crystallization to determine their shape and composition. Before placing the particles on silicon wafers, the wafers were pre-patterned using etching techniques based on a phenomena called “atomic step movement”. Because atom-high steps innately exist on silicon surfaces, these steps could

be moved during high-temperature treatments to fabricate a desired pattern. Chemical reactions (between the silicon, nitrogen and oxygen) caused very thin nitride linings to form in accordance with the atomic step boundaries, thereby pre-patterning the wafers. Then, in order to align the particles along the pattern on the wafer, the samples were annealed in an ultra-high vacuum chamber for several hours. An example of the resulting precisely aligned nanoparticles is displayed in Figure 10.1.

Figure 10.1: This atomic force microscope image shows a pattern of nanoparticles that self-assembled themselves on a pre-pattern substrate. Photo credit: Prabhakaran et al. [1]

The next step of the CAEN fabrication process aligns groups of wires. Also through self-assembly, two planes of aligned wires will be combined to form a two-dimensional grid with configurable molecular switches at the cross-points. The resulting grids will be in the order of a few microns. A separate process will create a silicon-based die using standard lithography.

The circuits on this die will provide power, clock lines, an I/O interface, and support logic for the grids of switches. The die will contain “holes” in which the grids are placed, aligned, and connected with wires on the die.

Using only self-assembly and self-alignment restrict the manufacturing to simple, regular structures, e.g. rafts of parallel wires or grids composed of orthogonal rafts.

NanoFabric

The architecture of the *nanoFabric* [166] is designed to overcome the constraints associated with directed assembly of nanometer-scale components and exploit the advantages of molecular electronics. Because the fabrication process is fundamentally non-deterministic and prone to introducing defects, the nanoFabric must be reconfigurable and amenable to self-testing.

The nanoFabric is a 2-D mesh of interconnected *nanoBlocks*. The nanoBlocks are logic blocks that can be programmed to implement a three-bit input to a three-bit output Boolean function and its complement. It can also be used as switches to route signals. The nanoBlock is composed of three sections:

- The molecular logic array, where the functionality of the block is located.
- The latches, used for signal restoration and signal latching for sequential circuit implementation.
- The I/O area, used to connect the nanoBlock to its neighbors through the switch block.

Figure 10.2 shows a nanoBlock realizing the logic AND function.

The nanoFabric is defect tolerant because it is regular, highly configurable, fine-grained, and has a rich interconnect. The regularity allows choosing where a particular function is implemented. The configurability allows selecting which nanowires, nanoBlocks, or parts of a nanoBlock will implement a particular circuit. The fine-grained nature of the device combined with the local nature of the interconnect reduces the impact of a defect to only a small portion of the fabric (or even a small portion of a single nanoBlock). Finally, the rich interconnect allows choosing among many paths in implementing a circuit. Thus, with a defect map working circuits can be created on a defective fabric. Defect discovery relies on the fact that

Figure 10.2: Logic diagram of a nanoBlock implementing an AND gate.

the nanoFabric can be configured to implement any circuit, which implies that the nanoFabric can be configured to test its own resources.

10.2 Molecule cascades

In [163] carbon monoxide (CO) molecules were arranged in atomically precise configurations, called *molecule cascades* (MC) where the motion of one molecule causes the subsequent motion of another, and so on in a cascade of motion similar to a row of toppling dominoes. This hopping motion is due to quantum tunneling of the molecule between neighboring binding sites on the surface. The tunneling rate is engineered in molecule cascades by controlling the hopping direction and interactions with neighboring molecules.

Two CO molecules positioned at nearest neighbor sites appear as dips centered at copper (Cu) lattice sites with a bright area directly between them. This arrangement is called *dimer*. The CO dimer is stable at 5 K although the molecules could be easily separated with the tip of the scanning tunneling microscope (STM). There are three possible configurations of a CO trimeron on the Cu(111) surface:

1. A three-fold symmetric, close-packed arrangement in which the molecules are all nearest neighbors to each other.
2. A bent-line configuration.
3. A straight-line configuration.

The trimers exhibit a hierarchy of stability. The close-packed structure is entirely stable at 5 K. The bent-line configuration is stable enough to image but decays spontaneously in a few minutes by the indicated hop of the central molecule. The straight-line configuration decays on a time-scale on the order of 1s. In [163] the bent-line arrangement of three molecules is used and is referred to as a *chevron*.

A cascade of hops is produced by arranging CO molecules in staggered chains of dimers. The cascade is initiated by moving a "trigger" CO molecule with the STM tip to form an initial chevron. This newly formed chevron then spontaneously decays and forms yet another chevron, etc., for a cascade of any length. Such a linked-chevron cascade propagates forward reliably because the energy of the system is lowered each time a molecule hops to a new site on the surface. The hopping rate is equal to the number of hops in the cascade divided by the average measured propagation time.

A molecule cascade serves as a wire to communicate one bit of information across the surface. A molecule in its initial or final position represents binary 0 or 1, respectively. The cascades can also serve as logic gates. The operation of a logic AND gate, whose inputs and outputs are linked-chevron cascades, were demonstrated in [163]. The core element of the AND gate is a chevron trimer. The central CO molecule becomes part of a chevron only if cascades have propagated along both input arms. Once this molecule becomes part of a chevron, its hop triggers the output arm of the AND gate. A schematic diagram illustrating this principle is shown in Figure 10.3. Figure 10.4 shows a molecule cascade implementing a two input logic AND after input *X* was triggered and the resulting cascade propagated to the central molecule. The remaining cascade is now a simple linked chevron cascade. Green circles represent CO's that have hopped to their final position.

An implementation of a two-input sorter, which computes both the logic AND and the logic OR of the inputs, was also presented in [163]. The structure contains fanouts to duplicate the input signals for distribution to both gates and a crossover to pass the two signals through each other. The OR gate is based on a chevron that is already assembled in the initial configuration. This chevron is stabilized by two molecules blocking the normal hop of the decaying chevron. However, when either of the input cascades appends a fourth molecule to this chevron, the augmented chevron decays as shown to activate the OR gate's output cascade. The fanout is also based on a stabilized chevron in the initial configuration.

The most attractive feature of molecule cascades is their size. The three-

Figure 10.3: A logical AND gate consists of 2 input arms (X and Y) which meet in the center of the gate. The central carbon monoxide molecule becomes part of a chevron only after both inputs have been triggered and have propagated to this molecule. Only then is the output cascade triggered. All hops in an AND gate and in the connecting wires are based on the chevron principle. Photo credit: IBM.

input sorter reported in [163], implemented in CMOS's technology, requires an area of $53\mu\text{m}^2$, whereas the molecule cascade implementation uses only 200nm^2 , a factor of 260,000 difference. Even if CMOS density were to continue to double every 2.5 years, it would still take 45 years to shrink to the size of the molecule cascades [163].

Figure 10.4: Logic AND gate in MC after input X was triggered and the resulting cascade propagated to the central molecule. Photo credit: IBM.

10.3 Specifics of technology mapping for CAEN and MC techniques

Common problem

Conventional logic design relies on the use of an inverter gate. However, in neither of the discussed technologies, CAEN and MC, inverters can be built. In CAEN the restriction to implement logic functions using two-terminal devices only is due to the difficulties associated with precise collocation of three wires at a nanometer-scale device. For molecule cascades, implementing an inverter would require that all molecules in the inverter's output untopple (in analogy to dominoes) when the input cascade topples. This is not possible because the untoppled state has higher energy than the toppled state. As a solution, two possibilities were proposed in [163]. One is to compute

the inverter function with a two-input gate called a CLOCKED-NOT, in which an input, the CLOCK signal (logical 1) is known to arrive at the gate after the other input, the DATA signal. A logical 1 DATA signal functions as an interrupter of the logical 1 CLOCK signal, and a logical 0 DATA signal allows the CLOCK signal to propagate to the output unchanged. Thus, the output is the complement of the DATA input.

Another option, that works for both CAEN as well as MC, is to avoid the need for an inverter by representing signals by the dual-rail convention that is employed in some self-timed logic designs [168]. This option is investigated here. As a basic building block we use a *molecular block*, shown in Figure 10.5. A molecular block has 4 inputs $x_1, \dots, x_4$ and two outputs $f_1 = x_1 \cdot x_2$ and $f_2 = x_3 + x_4$. The inputs x_3 and x_4 are the complements of the inputs x_1 and x_2 , respectively. As a result, $f_2 = x'_1 + x'_2 = (x_1 \cdot x_2)' = f'_1$ i.e., the outputs of the block are complements of each other. The sign "x'" represents Boolean complementation according to Boolean algebra as described in Section 2.2. Since all signals are present in both, complemented and non-complemented form, and the logic AND operation is available, any Boolean function can be implemented by a composition of such molecular blocks.

Figure 10.5: Logic diagram of a molecular block.

Multiple-level implementation style

Both the AND and the OR operation can be implemented both by CAEN and by linked-chevron cascades as shown in [166] and [163], respectively. Since the set of operations AND and NOT is functionally complete for Boolean

functions, and all signals are available in both non-complemented and complemented form, we can compute any Boolean function by interconnecting the molecular blocks in Figure 10.5.

Next, we show how a network of 2-input ANDs and inverters, which we call an AND/INVERTER graph (see Chapter 4), can be mapped into a network of interconnected molecular blocks.

Theorem 4 *There exists a one-to-one mapping between an AND/INVERTER graph and a network of interconnected molecular blocks shown in Figure 10.5.*

Proof: By construction. Let $x_1, \dots, x_n$ be a set of primary inputs, M be a set of molecular blocks, and $C \subseteq M \times M$ a set of wires connecting the blocks. Let $in(b), in'(b), out(b), out'(b)$ be defined as follows:

- $in(b)$ is one of the inputs $\{1, 2\}$ of a molecular block $b \in M$,
- $in'(b)$ is the input 3 of b if $in(b)$ is the input 1 of b ; otherwise, $in'(b)$ is the input 4 of b ,
- $out(b)$ is the output 5 of b ;
- $out'(b)$ is the output 6 of b .

An AND/INVERTER graph $G = (V, E)$ is mapped into a network of interconnected molecular blocks $A = (M, C)$ in the following way:

1. Every AND vertex $v \in V$ corresponds to a molecular block $b_v \in M$.
2. Every edge $(u, v) \in E$ between a pair of AND vertices, u and v , corresponds to two connections between the molecular blocks b_u and b_v , depending on the inverter attribute $i(u, v) \in \{0, 1\}$:

$$((u, v) \in E) \wedge (i(u, v) = 1) \Leftrightarrow ((out'(b_u), in(b_v)) \in C) \wedge ((out(b_u), in'(b_v)) \in C).$$

$$((u, v) \in E) \wedge (i(u, v) = 0) \Leftrightarrow ((out(b_u), in(b_v)) \in C) \wedge ((out'(b_u), in'(b_v)) \in C).$$

3. Every edge $(x_i, v) \in E$ between a primary input of the AND/INVERTER graph, x_i , $i \in \{1, \dots, n\}$, and an AND vertex v corresponds to two connections between the primary input x_i of the molecular network and the molecular block b_v , depending on the inverter attribute $i(x_i, v) \in \{0, 1\}$:

$$((x_i, v) \in E) \wedge (i(x_i, v) = 1) \Leftrightarrow ((x'_i, in(b_v)) \in C) \wedge ((x_i, in'(b_v)) \in C),$$

$$((x_i, v) \in E) \wedge (i(x_i, v) = 0) \Leftrightarrow ((x_i, in(b_v)) \in C) \wedge ((x'_i, in'(b_v)) \in C).$$

Figure 10.6: Example of (a) a network of interconnected molecular blocks; (b) the corresponding AND/INVERTER graph representation.

This mapping is unique up to permutation of the inputs 1 and 2 of the molecular block.

□

Figure 10.6(a) shows an example of a network of molecular blocks implementing the function $f(x_1, x_2, x_3) = (x_1 x_2)' x_3$. The corresponding AND/INVERTER graph representation is shown in Figure 10.6(b).

Two-level implementation style

For CAEN we also investigated a two-level implementation style. The CAEN molecular switches used in [166] are passive devices that cannot provide gain. Therefore, the signal is degraded every time it passes a switch. The need for signal restoration limits the possible size of the basic logic block. The study of [169] shows that this can be helped by using nanoscale FET devices, which provide for both signal restoration and programming support for the switches. This opens a possibility for building macro-cells similar to PLAs using CAEN (for more information regarding PLAs, see Section 2.3). Macro-cells of 50-100 columns/rows are predicted to be feasible to be implemented in this manner [169].

In Lemma 1, a formula for the upper bound on the size of a two-level CAEN implementation is defined. The formula is assuming that both the function and its complement is realized.

Suppose the on-set of a Boolean function f is given as a sum-of-products

with k products. Applying De Morgan's law we can get the product-of-sums expression for the complement of f with k sums (see Section 2.2). If the n inputs are placed on the west and north sides of the molecular block and the m outputs are taken from the east and south sides, then the resulting two-level implementation is square shaped.

Lemma 1 *An upper bound on the size of an implementation of a function $f: \{0,1\}^n \rightarrow \{0,1\}^m$ and its complement in CAEN is*

$$S_{2L} \leq N_r \times N_c,$$

where N_r and N_c are the numbers of rows and columns in the macro-cell given by

$$N_r = N_c = 2n - n_{unate} + m + \min(|F_f|, |R_f|).$$

n_{unate} is number of unate variables, F_f represents the on-set of function f and R_f the off-set (see Section 2.2). An example of a two-level implementation of a half-adder is shown in Figure 10.7. Logic functions are implemented by programming the configurable molecular switches which lie at each intersection of two orthogonal wires. A CAEN switch is a two-terminal switch behaving like a diode. Compared to a CMOS transistor, a CAEN switch uses much less power, since only a few electrons are required for switching.

The formula given by Lemma 1 is only an upper bound, since it is possible to implement a function in two-level form with less columns/rows. For example, the two-level implementation of the logic function AND shown in Figure 10.5 has one row and one column less than the N_r , N_c estimated by Lemma 1.

10.4 Experimental results

We performed a set of experiments to evaluate the effectiveness of the presented algorithms for CAEN and MC optimization.

Simple greedy optimization

The simple greedy optimization aims at minimizing the area of the resulting MC or CAEN implementation. The area is estimated as the rectangle of the length and width of the network of molecular blocks required to realize the function. The unit is square molecule wires. According to Theorem 4 there

Figure 10.7: A two-level half-adder implemented in CAEN.

is a one-to-one mapping between an AND/INVERTER graph representation and a network of interconnected molecular blocks. Hence, the optimization objective can be translated to length and width of the AND/INVERTER graph, where the length is the number of levels in the AND/INVERTER graph and the width is the number of AND vertices on the level containing most vertices.

Table 10.1 summarizes the result. The first column states the name of the benchmark. The second column shows the area A_{2L} for the two-level implementation, computed using Lemma 1, under the assumption that $|F_f| \leq |R_f|$. The area is in square molecule wires. Most of the benchmarks were too large to be represented in a sum-of-products form (indicated by the sign "-"). Column 3 shows the area A_{ML}^{opt} of the optimized network of molecular blocks using multiple-level implementation style.

The two last columns show the improvement of A_{ML}^{opt} with respect to two-level and non-optimized multiple-level implementation, respectively. As shown implementing and optimizing CAEN in a multiple-level style yields on average 64.7% smaller area than implementing it in a two-level style. Furthermore, the results show that optimizing the multiple-level implementations with a simple greedy algorithm yields on average circuits with 20.1% smaller area than no optimization at all.

benchmark	A_{2L}	A_{ML}^{opt}	$\frac{A_{2L} - A_{ML}^{opt}}{A_{2L}}$	$\frac{A_{ML} - A_{ML}^{opt}}{A_{ML}}$
C1355	-	27972	-	42.2%
C1908	-	20880	-	35.6%
C2670	-	45360	-	12.5%
C3540	-	67500	-	6.3%
C432	-	9720	-	0%
C499	-	18792	-	54.7%
C5315	-	127872	-	28.4%
C6288	-	157680	-	12.0%
C7552	-	135792	-	7.9%
C880	-	17640	-	23.9%
al2	31684	9504	70.0%	56.0%
apex6	889249	45864	94.8%	46.8%
apex7	400689	18648	95.3%	33.9%
apla	27556	23940	13.1%	13.6%
count	64009	11520	82.0%	37.5%
cps	2146225	149940	93.0%	5.6%
gary	65025	53352	18.0%	9.3%
i6	163216	28512	82.5%	23.3%
ibm	62001	27864	55.1%	14.0%
jbp	212521	43524	79.5%	16.2%
mark1	40000	20088	49.8%	10.0%
minmax32	-	299700	-	13.3%
mult16a	-	45360	-	36.2%
mult16b	-	7200	-	16.7%
opa	619369	51120	91.7%	9.0%
rot	-	94752	-	20.3%
shift	19044	9072	52.4%	9.7%
soar	594441	355752	40.2%	9.0%
t1	861184	294624	65.8%	17.5%
t2	121801	34560	71.6%	9.1%
t3	33856	21168	37.5%	12.5%
t4	300304	20736	93.1%	5.3%
term1	110224	44712	59.4%	9.2%
ti	419904	327060	22.1%	8.0%
x2dn	86436	31752	63.3%	26.3%
x4	538756	52632	90.2%	33.3%

Table 10.1: Experimental results for chemically assembled nanotechnology (CAEN) using a greedy optimization algorithm.

Greedy NPN-based optimization

The next iteration of experiments was done using the greedy NPN-based method described in Chapter 5.

Two scripts were used when optimizing the functions with SIS, *algebraic* and *rugged*. The resulting network was decomposed into an AND/INVERTER graph with the command *tech_decomp -a 2*. Any redundancy was removed with the command *red_removal*. The sign "*" indicates that the command *red_removal* could not be executed. The command *red_removal* was aborted after 200 seconds. The sign "-" means that the script did not terminate. The functions were read in using blif format (see Section 3.4).

The results of our tool compared to SIS are shown in Table 10.2. The first column states the name of the benchmark. The next column shows the area of the network of molecular blocks optimized by the greedy NPN-based algorithm, A_n . Column 3 the resulting area after optimizing with SIS script *algebraic*, A_a and column 4 the size after optimizing with SIS script *rugged*, A_r . The area is reported in square molecule wires.

The last two columns shows the improvement in percent of the greedy NPN-based optimization algorithm over SIS script *algebraic* and SIS script *rugged*, respectively.

The presented greedy NPN-based optimization algorithm, obtains a smaller AND/INVERTER graph than any of the SIS scripts *algebraic* and *rugged* for 19.4 % of the benchmarks. For 26.9 % of the benchmarks the presented optimization algorithm finds a smaller AND/INVERTER graph than one of the executed SIS scripts.

Simulated annealing optimization

At last we applied our fully developed simulated annealing framework to the problem of optimizing designs implemented in molecule cascades or CAEN. A difference from traditional logic synthesis is the one-to-one mapping between the AND/INVERTER graph representation and the mapped design (Theorem 4) which means there is no need to integrate a technology mapper. In Chapter 8, we presented the integrated random search mixing annealing with a greedy algorithm and an algebraic decomposition step that we are using here.

Table 10.3 summarizes the results. The first column states the name of the benchmark. The next column shows the area of the network of molecular blocks optimized by the simulated annealing algorithm, A_s . Column 3 shows the area after optimizing with SIS *script algebraic*, A_a , and column 4 the area after optimizing with SIS *script rugged*, A_r . The area is reported as square molecule wires.

benchmark	A_n	A_a	A_r	$\frac{A_a - A_n}{A_a}$	$\frac{A_r - A_n}{A_r}$
9symml	816	840	868	-2.86%	-5.99%
C1355	1712	2104	1720	-18.63%	-0.47%
C1908	1596	1652	1712	-3.39%	-6.78%
C2670	2700	2600	3116	3.85%	-13.35%
C3540	4128	4128	5188	0.00%	-20.43%
C432	836	624	1136	33.97%	-26.41%
C499	1584	1688	1720	-6.16%	-7.91%
C5315	6600	6216	6120	6.18%	7.84%
C6288	7664	9516	9756	-19.46%	-21.44%
C7552	8104	7272	10168	11.44%	-20.30%
C880	1308	1492	1616	-12.33%	-19.06%
alu4	3732	2524	2400	47.86%	55.50%
apex6	2636	2920	3100	-9.73%	-14.97%
apex7	956	856	928	11.68%	3.02%
cps	7512	3852	4644	95.02%	61.76%
cse	960	724	804	32.60%	19.40%
ex1	1264	816	900	54.90%	40.44%
f51m	824	952	468	-13.45%	76.07%
frg1	1548	444	792	248.65%	95.45%
i2	912	860	2128	6.05%	-57.14%
i4	984	776	-	26.80%	-
i5	1252	792	792	58.08%	58.08%
ibm	1244	704	844	76.70%	47.39%
k2	8732	3308	3676	163.97%	137.54%
key	13112	13732	13732	-4.52%	-4.52%
keyb	1624	864	1132	87.96%	43.46%
kirkman	896	864	1024	3.70%	-12.50%
mclc	1076	364	304	195.60%	253.95%
minmax20	3544	3212*	4684*	10.34%	-24.34%
minmax32	5704	5244*	-	8.77%	-
mult32a	1776	1524	1620	16.54%	9.63%
pair	6260	6108	6616	2.49%	-5.38%
planet	3620	1888	2180	91.74%	66.06%
sbc	2740	2988*	3320*	-8.30%	-17.47%
styr	2592	1684	2424	53.92%	6.93%
t481	6532	2664	980	145.20%	566.53%
vda	4228	1916	2036	120.67%	107.66%
x1	3916	1148	1360	241.11%	187.94%
average:	3348	2681	2935	46.24%	61.63%

Table 10.2: Comparison of a greedy NPN-based optimization algorithm for CAEN and MC with SIS.

The last two columns show the improvement in percent of the presented annealing-based algorithm over *script algebraic* and *script rugged*, respectively. The results show that we can improve the area achieved by SIS *script algebraic* by 13.9% on average and the area achieved by SIS *script rugged* by 19.3%.

10.5 Conclusion

In this chapter we showed that our developed logic synthesis frameworks can be used for two emerging technologies, chemically assembled nanotechnology and molecule cascades. The motivation for the work presented in this chapter, was that the exponential improvement in speed and integration of silicon transistor technology is expected to slow down as devices approach nanometer devices. Alternatives to silicon VLSI have been proposed, including the two techniques described in this chapter. Many nanotechnology programs are initiated by groups with the expertise in chemistry and physics. This contrasts very limited activities at design level. With the work presented here, we wanted to give ideas on how logic synthesis for CAEN and MC technologies could be constructed.

benchmark	A_s	A_a	A_r	$\frac{A_a - A_s}{A_a}$	$\frac{A_r - A_s}{A_r}$
9symml	776	840	868	7.62%	10.60%
C1355	1928	2104	1720	8.37%	-12.09%
C1908	1484	1652	1712	10.17%	13.32%
C2670	2292	2600	3116	11.85%	26.44%
C3540	3928	4128	5188	4.84%	24.29%
C432	588	624	1136	5.77%	48.24%
C499	1544	1688	1720	8.53%	10.23%
C5315	5800	6216	6120	6.69%	5.23%
C6288	7492	9516	9756	21.27%	23.21%
C7552	6656	7272	10168	8.47%	34.54%
C880	1320	1492	1616	11.53%	18.32%
alu2	1212	1280	1272	5.31%	4.72%
alu4	2416	2524	2400	4.28%	-0.67%
apex6	2468	2920	3100	15.48%	20.39%
apex7	716	856	928	16.36%	22.84%
dal	2980	3404	3832	12.46%	22.23%
des	11920	13132	13192	9.23%	9.64%
example2	920	1120	1228	17.86%	25.08%
f51m	416	952	468	56.30%	11.11%
frg1	416	444	792	6.31%	47.47%
frg2	2836	3248	3184	12.68%	10.93%
i10	6924	8252	12616	16.09%	45.12%
i2	840	860	2128	2.33%	60.53%
i4	760	776	0	2.06%	
i5	528	792	792	33.33%	33.33%
i6	1524	1792	1792	14.96%	14.96%
i7	2020	2288	2288	11.71%	11.71%
i8	3252	3788	4500	14.15%	27.73%
i9	2052	2360	2360	13.05%	13.05%
k2	2932	3308	3676	11.37%	20.24%
pair	5648	6108	6616	7.53%	14.63%
rot	1988	2368	2988	16.05%	33.47%
t481	1476	2664	980	44.59%	-50.61%
term1	608	772	764	21.24%	20.42%
ttt2	616	700	896	12.00%	31.25%
x1	976	1148	1360	14.98%	28.24%
x3	2360	2728	3228	13.49%	26.89%
x4	1120	1352	1452	17.16%	22.87%
average:	2519	2897	3122	13.88%	19.34%

Table 10.3: Comparison of the simulated annealing algorithm for CAEN and MC with SIS.

Chapter 11

Conclusion

This thesis presented a global optimization approach combining technology independent optimization steps with technology dependent objectives in an annealing-based framework. We proved that, for the presented move set and selection distribution, detailed balance is satisfied and thus the annealing process asymptotically converges to an optimal solution. Furthermore, we showed that the presented approach can smoothly trade-off complex, multiple-dimensional objective functions and achieve competitive results.

The combination of technology independent and technology dependent objectives is handled through dynamic weighting. Dynamic weighting reflects the sensitivity of the local graph structures with respect to the actual technology parameters such as gate sizes, delays, and power levels.

The results show that, on average, the presented advanced annealing approach can improve the area and delay of circuits optimized with script rugged of SIS by 11.2% and 32.5% respectively.

In the process of developing our advanced annealing framework, we explored several other optimization techniques, including a greedy optimization algorithm based on local transformations. The local transformations are generated using NPN classes of Boolean functions, which implies that more of the interesting cases are taken into account than for other local transformation methods that define the set of eligible transformations directly in the graph domain.

We also showed how our developed logic synthesis framework can be applied to two emerging technologies: chemically assembled nanotechnology and molecule cascades. The reason for our interest in emerging technologies

is that the exponential improvement in speed and integration of silicon transistor technology is expected to slow down as devices approach the nanometer dimensions. The search for functional nanometer-scale structures led to the exploration of alternative computing schemes. Many nanotechnology programs are initiated by groups with the expertise in chemistry and physics. This contrasts very limited activities at design level.

The main contribution of this thesis is the integration of the different phases of logic synthesis with each other. The result of the integration is a robust optimization framework driven by a powerful simulated annealing-based engine.

Open problems

The presented annealing-based optimization framework considers only delay and area, however, the general framework easily supports other objectives such as power, placement, etc.

After the enhancement of the objective function, the next natural step for the presented advanced annealing framework would be to integrate placement into the flow. There are already several placement tools implemented using simulated annealing [170, 171, 172, 173, 95]. Integrating placement into the framework would give the benefit of an even more complex objective function, considering technology parameters regarding area, delay, placeability, routability, power consumption, etc.

Regarding the OpenAccess environment, in which we implemented our advanced annealing framework, the integration could be made tighter by using the OA Gear technology mapper. Currently, the OA Gear only provide an extremely simple technology mapper, mapping an AND/INVERTER graph to two predefined gates.

Bibliography

- [1] K. Prabhakaran, S. Gotzinger, K. Shafi, A. Mazzei, S. Schietinger, and O. Benson, “Ultrafine luminescent structures through nanoparticle self-assembly,” *Nanotechnology*, vol. 17, pp. 3802–3805, 2006.
- [2] J. Darringer, W. Joyner, L. Berman, and L. Trevillyan, “Logic synthesis through local transformations,” *IBM Journal on Research and Development*, vol. 25, pp. 272–280, July 1981.
- [3] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, “SIS: A system for sequential circuit synthesis,” in *Technical report*, (U.C. Berkley), May 1992.
- [4] L. Stok, D. S. Kung, D. Brand, A. D. Drumm, A. J. Sullivan, L. N. Reddy, N. Hieter, D. J. Geiger, H. H. Chao, and P. J. Osler, “Boole-Dozer: Logic synthesis for ASICs,” *IBM Journal on Research and Development*, vol. 40, pp. 407–430, July 1996.
- [5] R. K. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*. Boston, MA: Kluwer Academic Publishers, 1984.
- [6] R. K. Brayton and C. McMullen, “The decomposition and factorization of Boolean expression,” in *Proceedings of the IEEE International Symposium of Circuits and Systems*, pp. 49–54, May 1982.
- [7] C. L. Berman, D. J. Hathaway, A. S. LaPaugh, and L. H. Trevillyan, “Efficient techniques for timing correction,” in *Proceedings of the International Symposium on Circuits and Systems*, pp. 415–419, May 1990.

- [8] P. C. McGeer, R. K. Brayton, A. L. Sangiovanni-Vincentelli, and S. K. Sahni, "Performance enhancement through the generalized bypass transform," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 184–187, November 1991.
- [9] L. Berman and L. H. Trevillyan, "Improved logic optimization using global-flow analysis," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 102–105, November 1988.
- [10] K.-T. Cheng and L. Entrena, "Multi-level logic optimization by redundancy addition and removal," in *Proceedings of the European Design Automation Conference*, pp. 373–377, February 1993.
- [11] S. Hassoun and T. Sasao, eds., *Logic synthesis and verification*. Kluwer academic publishers, 2002.
- [12] P. Färm and Dubrova, "Fast multi-level logic optimization using local transformations," in *International Workshop on Logic & Synthesis*, pp. 120–125, May 2003.
- [13] P. Färm, E. Dubrova, and A. Kuehlmann, "Integrated logic synthesis using simulated annealing," *Submitted to IEE Circuits, Devices and Systems Journal*, 2006.
- [14] P. Färm, E. Dubrova, and A. Kuehlmann, "Integrated logic synthesis using simulated annealing," in *International Workshop on Logic & Synthesis*, pp. 7–14, June 2006.
- [15] P. Färm, E. Dubrova, and A. Kuehlmann, "Logic optimization using rule-based randomized search," in *Proceedings of the Asia South Pacific Design Automation Conference*, pp. 998–1001, January 2005.
- [16] P. Färm, E. Dubrova, and H. Tenhunen, "Logic optimization technique for molecular nanoelectronics," in *Proceedings of SPIE Nanotechnology II*, pp. 95–104, 2005.
- [17] P. Färm, E. Dubrova, and H. Tenhunen, "NanoEDA: Architecture and design methodology for nano-scale electronic systems," in *Swedish system-on-chip conference*, 2003.

- [18] P. Färm, E. Dubrova, and H. Tenhunen, “Logic optimization for technology mapping for CAEN,” in *Technical Proceedings of the Nanotechnology Conference and Trade Show*, pp. 145–148, 2003.
- [19] P. Färm and E. Dubrova, “Technology mapping for chemically assembled electronic nanotechnology,” in *International Workshop on Logic & Synthesis*, pp. 121–124, June 2002.
- [20] P. Färm, E. Dubrova, R. Stankovic, and J. Astola, “Conjunctive decomposition for multiple-valued input binary-valued output functions,” in *Proceedings of International TICSP Workshop on Spectral Methods and Multirate Signal Processing*, pp. 227–234, September 2002.
- [21] E. Dubrova and P. Färm, “A conjunctive canonical expansion of multiple-valued functions,” in *Proceedings of the 32nd International Symposium on Multiple-Valued Logic*, pp. 35–38, May 2002.
- [22] K. McMillan, “A conjunctively decomposed Boolean representation for symbolic model checking,” in *Proceedings of the 8th international Computer aided verification conference*, (New Brunswick, NJ), pp. 13–25, July 1996.
- [23] F. Hill and G. Peterson, *Introduction to switching theory and logical design*. New York, NY: Wiley, 1981.
- [24] H. Flegg, *Boolean algebra and its application including Boolean matrix algebra*. London: Blackie, 1964.
- [25] F. Brown, *Boolean reasoning*. Boston, MA: Kluwer Academic Publisher, 1990.
- [26] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, 1994.
- [27] K. Bartlett, R. Brayton, G. Hatchel, R. Jacoby, C. Morrison, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, “Multi-level logic minimization using implicit don’t cares,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 6, pp. 723–740, June 1988.

- [28] J. Bondy and U. Murty, *Graph theory with applications*. New York, NY: North-Holland, 1976.
- [29] N. Deo, *Graph theory with applications to engineering and computer science*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
- [30] F. Harary, *Graph theory*. Reading, MA: Addison-Wesley, 1972.
- [31] M. Garey and D. Johnson, eds., *Computers and Intractability*. New York, NY: Freeman, 1979.
- [32] C. Lee, "Representation of switching circuits by binary decision diagrams," *Bell systems technical journal*, vol. 38, pp. 985–999, 1959.
- [33] S. Akers, "Binary decision diagrams," *Transactions on Computers*, vol. 27, pp. 509–516, June 1978.
- [34] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, pp. 677–691, August 1986.
- [35] R. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *Computing surveys*, vol. 24, pp. 293–318, September 1992.
- [36] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and improvements of Boolean comparison method based on binary decision diagrams," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 2–5, November 1988.
- [37] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic verification using binary decision diagrams in logic synthesis environment," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 6–9, November 1988.
- [38] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," in *Proceedings of the 27th Design Automation Conference*, pp. 52–57, June 1990.
- [39] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 42–47, November 1993.

- [40] H. Higuchi and F. Somenzi, "Lazy group sifting for efficient symbolic state traversal of FSMs," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 45–49, November 1999.
- [41] G. Kamhi and L. Fix, "Adaptive variable ordering for ordered binary decision diagrams," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 359–365, November 1998.
- [42] C. Meinel and A. Slobodova, "Speeding up variable reordering of OBDDs," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 338–343, October 1997.
- [43] S. Panda and F. Somenzi, "Who are the variables in your neighborhood," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 74–77, November 1995.
- [44] R. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication," *IEEE Transactions on Computers*, vol. C-40, pp. 205–213, February 1991.
- [45] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in *Proceedings of the 27th Design Automation Conference*, pp. 40–45, June 1990.
- [46] S. Jeong, B. Plessier, G. Hachtel, and F. Somenzi, "Extended BDD's: Trading off canonicity for structure in verification algorithms," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 464–467, November 1991.
- [47] G. Smith, R. Bahnsen, and H. Halliwell, "Boolean comparison of hardware and flowcharts," *IBM Journal on Research and Development*, vol. 26, pp. 106–116, January 1982.
- [48] S. Hurst, *The Logical Processing of Digital Signals*. Crane, Russak and Company, Inc., 1978.
- [49] S. Hurst, D. Miller, and J. Muzio, *Spectral Techniques in Digital Logic*. Academic Press, 1985.

- [50] J. Doob, *Stochastic processes*. New York, NY: John Wiley & Sons, 1953.
- [51] S. Orey, *Limit theorems for Markov chain transistion probabilities*. London,UK: Van Nostrand Reinhold, 1971.
- [52] E. Nummelin, *General irreducible Markov chains and non-negative operators*. Cambridge,UK: Cambridge university press, 1984.
- [53] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *Transactions on Computer-Aided Design*, vol. 6, pp. 1062–1081, November 1987.
- [54] E. McCluskey, "Minimization of Boolean functions," *Bell Systems Technical Journal*, vol. 35, pp. 1417–1444, April 1959.
- [55] P. C. McGeer, R. Brayton, J. Sanghavi, and A. L. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," *IEEE Transactions on VLSI*, vol. 1, pp. 432–440, December 1993.
- [56] K. Bartlett, W. Coehn, A. de Geus, and G. Hatchel, "Synthesis and optimization of multilevel logic under timing constrains," *IEEE Transactions on Computer-Aided Design*, vol. 5, pp. 582–596, April 1986.
- [57] D. Bostick and G. D. Hachtel, "The Boulder optimal logic design system," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 62–65, November 1987.
- [58] R. K. Brayton and C. McMullen, "The Yorktown silicon compiler," in *Proceedings of the IEEE International Symposium of Circuits and Systems*, pp. 391–394, 1985.
- [59] H. Mathony and U. G. Baitinger, "CARLOS: An automated multilevel logic design system for CMOS semi-custom integrated circuits," *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 3, pp. 346–355, 1988.
- [60] K. Keutzer, "Dagon: technology binding and local optimization by DAG matching," in *Proceedings of the 24th Design Automation Conference*, pp. 341–347, 1987.

- [61] A. Aho and S. Johnson, "Optimal code generation for expression trees," *J. ACM*, vol. 23, pp. 488–501, 1976.
- [62] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 264–271, November 1995.
- [63] L. Stok, M. Iyer, and A. Sullivan, "Wavefront technology mapping," in *Proceedings of Design Automation and Test in Europe*, pp. 531–536, March 1999.
- [64] R. Murgai, "On the complexity of minimum-delay gate resizing/technology mapping under load-dependent gate model," in *Proceedings of International Workshop of Logic & Synthesis*, pp. 209–211, June 1999.
- [65] W. Li, A. Lim, P. Agarwal, and S. Sahni, "On the circuit implementation problem," in *Proceedings of the 29th Design Automation Conference*, pp. 478–483, June 1992.
- [66] J. Fishburn, "LATTIS: an iterative speedup heuristic for mapped logic," in *Proceedings of the 29th Design Automation Conference*, pp. 488–491, June 1992.
- [67] J. Ishikawa, H. Sato, M. Hiramane, K. Ishida, S. Oguri, Y. Kazuma, and S. Murai, "A rule-based reorganization system LORES/EX," in *Digest of Technical Papers of the IEEE International Conference on Computer Design*, pp. 262–266, October 1988.
- [68] M. Damiani and G. D. Micheli, "Observability don't care sets and Boolean relations," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 502–505, November 1990.
- [69] H. Savoj, H. Touati, and R. Brayton, "Extracting local don't cares for network optimization," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 514–517, November 1991.

- [70] S. Muroga, Y. Kambayashi, H. Lai, and J. Culliney, "The transduction method - design of logic networks based on permissible functions," *IEEE Transactions on Computers*, vol. 10, pp. 1404–1424, October 1989.
- [71] L. Berman and L. H. Trevillyan, "A global approach to circuit size reduction," in *Advanced research in VLSI, 5th MIT Conference*, pp. 66–69, November 1988.
- [72] L. Berman and L. H. Trevillyan, "Global flow optimization in automatic logic design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 5, pp. 557–562, May 1991.
- [73] L. Stok, D. Kung, D. Brand, A. Drumm, A. Sullivan, L. Reddy, N. Hieter, D. Gieger, H. Chao, and P. Osler, "BooleDozer: Logic synthesis for ASICs," *IBM Journal of Research and Development*, vol. 4, pp. 407–430, July 1986.
- [74] D. Gregory, K. Bartlett, A. de Geus, and G. Hachtel, "SOCRATES: A system for automatically synthesizing and optimizing combinational logic," in *Proceedings of the 23rd Design Automation Conference*, pp. 79–85, June 1986.
- [75] C. Tsai and M. Marek-Sadowska, "Multilevel logic synthesis for arithmetic functions," in *Proceedings of the 33rd Design Automation Conference*, pp. 242–247, June 1996.
- [76] S. Chattopadhyay, S. Roy, and P. Chaudhuri, "KGPMIN: an efficient multilevel multioutput AND-OR-XOR minimizer," *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, vol. 3, pp. 257–265, March 1997.
- [77] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M. Perkowski, "Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams," in *Proceedings of the 31st Design Automation Conference*, pp. 415–419, June 1994.
- [78] E. Dubrova, D. Miller, and J. Muzio, "Upper bound on number of products in AND-OR-XOR expansion of logic functions," *Electronics Letters*, vol. 31, pp. 541–542, March 1995.

- [79] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *Transactions on Computer-Aided Design*, vol. 6, pp. 727–750, September 1987.
- [80] S. Chang and M. Marek-Sadowska, "Technology mapping via transformations of functional graphs," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 159–162, November 1992.
- [81] Y. Lai, M. Pedram, and S. Vrudhula, "BDD based decomposition of logic functions with application to FPGA synthesis," in *Proceedings of the 30th Design Automation Conference*, pp. 642–647, June 1993.
- [82] R. Murgai, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimum functional decompositions using encoding," in *Proceedings of the 31st Design Automation Conference*, pp. 408–414, June 1994.
- [83] H. Sawada, T. Suyama, and A. Nagoya, "Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 353–358, November 1995.
- [84] B. Wurth, K. Eckl, and K. Antreich, "Functional multiple-output decomposition: theory and an implicit algorithm," in *Proceedings of the 32nd Design Automation Conference*, pp. 54–59, June 1995.
- [85] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 78–82, November 1997.
- [86] C. Yang, M. Ciesielski, and V. Singhal, "BDS: A BDD-based logic optimization system," in *Proceedings of the 37th Design Automation Conference*, pp. 92–97, June 2000.
- [87] Y. Ye and K. Roy, "A graph-based synthesis algorithm for AND/XOR networks," in *Proceedings of the 34th Design Automation Conference*, pp. 107–112, June 1997.
- [88] V. Bertacco, S. Minato, P. Verplaeste, L. Benini, and G. D. Micheli, "Decision diagrams and pass transistor logic synthesis," in *International Workshop on Logic & Synthesis*, May 1997.

- [89] P. Buch, A. Narayan, A. Newton, and A. Sangiovanni-Vincentelli, "Logic synthesis for large pass transistor circuits," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 663–670, November 1997.
- [90] T. Liu, A. Aziz, and J. Burns, "Performance driven synthesis for pass-transistor logic," in *International Workshop on Logic & Synthesis*, pp. 255–259, June 1998.
- [91] Z. Xiu, D. A. Papa, P. Chong, C. Albrecht, A. Kuehlmann, R. A. Rutenbar, and I. L. Markov, "Early research experience with OpenAccess Gear: An open source development environment for physical design," in *International Symposium on Physical Design*, pp. 94–100, April 2005.
- [92] A. Caldwell, A. Kahng, and I. Markov, "Can recursive bisection alone produce routable placements?," in *Proceedings of the 37th Design Automation Conference*, pp. 477–482, June 2000.
- [93] S. Adya, S. Chaturvedi, J. Roy, D. Papa, and I. Markov, "Unification of partitioning, placement and floorplanning," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 550–557, November 2004.
- [94] Z. Xiu, J. Ma, S. Fowler, and R. Rutenbar, "Large-scale placement by grid-warping," in *Proceedings of the 41st Design Automation Conference*, pp. 351–356, June 2004.
- [95] M. Wang, X. Yang, and M. Sarrafzadeh, "Dragon2000: Standard-cell placement tool for large industry circuits," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 260–264, November 2000.
- [96] M. Yildiz and P. Madden, "Global objectives for standard cell placement," in *Proceedings of the 11th ACM Great Lakes Symposium on VLSI*, pp. 68–72, March 2001.
- [97] S. Adya, M. Yildiz, I. Markov, P. Villarrubia, P. Parakh, and P. Madden, "Benchmarking for large-scale placement and beyond," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, pp. 472–487, April 2004.

- [98] P. Madden, "Reporting of standard cell placement results," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 240–247, February 2002.
- [99] <http://www.openeda.si2.org/>
- [100] T. Blanchard, R. Ferreri, and J. Wilmore, "The OpenAccess coalition: The drive to an open industry standard information model, API, and reference implementation for IC design data," in *International symposium on quality electronic design*, pp. 69–74, March 2002.
- [101] <http://www.doxygen.org/>
- [102] <http://www.trolltech.no/>
- [103] <http://www.opengl.org/>
- [104] L. Hellerman, "A catalog of three-variable or-inverter and and-inverter logical circuits," *Transactions on Electronical Computation*, vol. 12, pp. 198–223, June 1963.
- [105] A. Kuehlmann, M. Ganai, and V. Paruthi, "Robust Boolean reasoning for equivalence checking and functional property verification," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, pp. 1377–1394, December 2002.
- [106] J. Monteiro, S. Devadas, A. Ghosh, K. Keutzer, and J. White, "Estimation of average switching activity in combinational logic circuits using symbolic simulation," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 121–127, January 1997.
- [107] J. Costa, J. Monteiro, and S. Devadas, "Switching activity estimation using limited depth reconvergent path analysis," in *International Symposium on Low Power Electronics and Design*, pp. 184–189, August 1997.
- [108] K. Parker and E. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Transactions on Computers*, vol. C-24, pp. 668–670, June 1975.
- [109] J. Savir, G. Ditlow, and P. Bardell, "Random pattern testability," *Transactions on Computers*, vol. C-33, pp. 79–90, January 1984.

- [110] R. Krenz, E. Dubrova, and A. Kuehlmann, "Circuit-based evaluation of the arithmetic transform of Boolean functions," in *International Workshop on Logic & Synthesis*, pp. 321–326, June 2002.
- [111] M. Ganai and A. Kuehlmann, "On-the-fly compression of logical circuits," in *International Workshop on Logic & Synthesis*, May 2000.
- [112] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.
- [113] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.
- [114] A. Metropolis, N. and Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *Journal of chemical physics*, no. 21, pp. 1087–1092, 1953.
- [115] R. Otten and L. van Ginneken, *The Annealing Algorithm*. Boston/Dordrecht/London: Kluwer Academic Publishers, 1989.
- [116] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica*, vol. 6, no. 3, pp. 302–345, 1991.
- [117] E. Aarts and P. van Laarhoven, "A new polynomial-time cooling schedule," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 206–208, 1985.
- [118] M. D. Huang, F. Romeo, and A. L. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 381–384, November 1986.
- [119] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in *Proceedings of the 25th Design Automation Conference*, pp. 306–311, 1988.
- [120] F. Romeo, *Simulated Annealing: Theory and Applications to Layout Problems*. Ph.D. Thesis, University of California, Berkeley, 1989.

- [121] J. Lam and J. Delosme, "An adaptive annealing schedule," in *Technical report 8608*, (University of Yale), 1987.
- [122] S. White, "Concept of scale in simulated annealing," in *Proceedings of International Conference on Computer Design*, pp. 646–651, 1984.
- [123] S. German and D. German, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Transactions of Pattern Analysis and Machine Intelligence*, vol. 6, pp. 721–741, 1984.
- [124] B. Gidas, "Non-stational Markov chains and convergence of the annealing algorithm," *Journal of Statistical Physics*, vol. 39, pp. 73–131, 1984.
- [125] S. Gelfand and S. Mitter, "Analysis of simulated annealing for optimization," in *Proceedings of 24th Conference on Decision and Control*, pp. 779–786, December 1985.
- [126] M. Lundy and A. Mees, "Convergence of the annealing algorithm," *Mathematical Programming*, vol. 34, pp. 111–124, 1986.
- [127] D. Mitra, F. Romeo, and A. L. Sangiovanni-Vincentelli, "Convergence and finite-time behaviour of simulated annealing," *Advanced Applied Probability*, vol. 18, pp. 747–771, 1986.
- [128] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995.
- [129] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural network*. John Wiley & Sons, 1989.
- [130] P. van Laarhoven and E. Haarts, *Simulated annealing: theory and applications (Mathematics and its applications)*. Kluwer academic publisher, 1987.
- [131] L. Davis, *Genetic algorithms and simulated annealing (Research notes in artificial intelligence)*. Morgan Kaufmann publisher, 1987.
- [132] P. Salamon, *Facts, conjectures, and improvements for simulated annealing (SIAM monographs on mathematical modeling and computation)*. Society for industrial and applied mathematics, 2002.

- [133] J. E. Kalivas, *Adaption of simulated annealing to chemical optimization problems (Data handling in science and technology)*. Elsevier publishing company, 1995.
- [134] D. Wong, *Simulated annealing for VLSI design (The international series in engineering and computer science)*. Springer, 1988.
- [135] C. Sechen, *VLSI placement and global routing using simulated annealing (The international series in engineering and computer science)*. Springer, 1988.
- [136] M. E. Johnson, *Simulated annealing (SA and optimization: modern algorithms with VLSI, optimal design, and missile defense applications)*. American sciences press, 1989.
- [137] R. Azencott, *Simulated annealing: parallelization techniques (Wiley-Interscience series in discrete mathematics)*. John Wiley & Sons, 1992.
- [138] J. Lam and J.-M. Delosme, "Logic mimization using simulated annealing," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 348–352, November 1986.
- [139] J. Sanchez and J. Lanchares, "Multilevel logic synthesis using algorithms based on natural processes," in *Proceedings of the international conference on microelectronics*, (Nis,Serbia), pp. 823–828, September 1995.
- [140] G. Birkhoff and S. MacLane, *Brief Survey of Modern Algebra*. New York: Macmillan, 1977.
- [141] R. Eglese, "Simulated annealing: A tool for operational research," *European Journal of Operational Research*, vol. 46, pp. 271–281, 1990.
- [142] K. Binder, *Monte Carlo methods in statistical physics*. Springer-Verlag, 1986.
- [143] G. Gonsalves, "Logic synthesis using simulated annealing," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 561–564, October 1986.
- [144] F. Malihot and G. Micheli, "Technology mapping using Boolean matching and don't care sets," in *Proceedings of the European Design Automation Conference*, pp. 212–216, March 1990.

- [145] J. Burch and D. Long, "Efficient Boolean function matching," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 408–411, November 1992.
- [146] Q. Wu, C. Chen, and J. Acken, "Efficient Boolean matching algorithm for cell libraries," in *Proceedings of the IEEE International Conference on Computer Design*, pp. 36–39, October 1994.
- [147] U. Hinsberger and R. Kolla, "Boolean matching for large libraries," in *Proceedings of the 35th Design Automation Conference*, pp. 206–211, June 1998.
- [148] J. Ciric and C. Sechen, "Efficient canonical form for Boolean matching of complex functions in large libraries," in *Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 535–544, May 2003.
- [149] D. Debnath and T. Sasao, "Efficient computation of canonical form for Boolean matching in large libraries," in *Proceedings of the Asia South Pacific Design Automation Conference*, pp. 591–596, January 2004.
- [150] D. Chai and A. Kuehlmann, "Building a better Boolean matcher and symmetry detector," in *Proceedings of Design Automation and Test in Europe*, pp. 1–6, March 2006.
- [151] H. Touati, *Performance Oriented Technology Mapping*. Ph.D. Thesis, University of California, Berkeley, 1990.
- [152] J. Meindl, Q. Chen, and J. Davis, "Limits on silicon nanoelectronics for terascale integration," *Science*, vol. 293, pp. 2044–2049, 2001.
- [153] D. Goldhaber-Gordon, M. Montemerlo, J. Love, G. Opiteck, and J. Ellenbogen, "Overview of nanoelectronic devices," *Proceedings of the IEEE*, vol. 85, pp. 521–540, April 1997.
- [154] G. Tseng and J. Ellenbogen, "Nanotechnology: Enhanced: Toward nanocomputers," *Science*, vol. 294, pp. 1293–1294, 2001.
- [155] S. Wind, J. Appenzeller, R. Martel, and V. Derycke, "Vertical scaling of single-wall carbon nanotube CMOS field effect transistors using top gate electrodes," *Applied Physics Letter*, vol. 80, p. 3817, 2002.

- [156] Y. Huang, X. Duan, Y. Cui, L. Lauhon, K. Kim, and C. Lieber, "Logic gates and computation from assembled nanowire building blocks," *Science*, vol. 294, pp. 1313–1317, 2001.
- [157] D. Eigler and C. L. and W. Rudge, "An atomic switch realized with the scanning tunnelling microscope," *Nature*, vol. 352, pp. 600–602, 1991.
- [158] C. Lent and P. Tougaw, "A device architecture for computing with quantum dots," *Proceedings of the IEEE*, vol. 85, pp. 541–557, April 1997.
- [159] I. Amlani, A. Orlov, G. Toth, G. Bernstein, C. Lent, and G. Snider, "Digital logic gate using quantum-dot cellular automata," *Science*, vol. 284, pp. 289–291, 1999.
- [160] R. Cowburn and M. Welland, "Room temperature magnetic quantum cellular automata," *Science*, vol. 287, pp. 1466–1468, 2000.
- [161] R. Lipton, "DNA solution of hard computational problem," *Science*, vol. 268, pp. 542–545, 1995.
- [162] R. Braich, N. Chelyapov, C. Johnson, P. Rothmund, L. Adleman, R. Cowburn, and M. Welland, "Solution of a 20-variable 3-SAT problem on a DNA computer," *Science*, vol. 296, pp. 499–502, 2002.
- [163] A. Heinrich, C. Lutz, J. Gupta, and D. Eigler, "Molecule cascades," *Science*, vol. 298, pp. 1381–1387, 2002.
- [164] K. Drexler, *Nanosystems*. New York, NY: Wiley, 1992.
- [165] D. Swade, "Redeeming Charles Babbage's mechanical computer," *Scientific American*, vol. 268, no. 2, pp. 62–68, 1993.
- [166] S. Goldstein and M. Badiu, "Nanofabrics: Spatial computing using molecular electronics," in *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pp. 178–191, June 2001.
- [167] T. Mallouk, "Nanomaterials: Synthesis and assembly," in *Foresight Conference Tutorial*, November 2000.
- [168] C. Mead and L. Conway, *Introduction to VLSI systems*. UK: Addison-Wesley, 1980.

- [169] A. DeHon, "Array-based architecture for molecular electronics," in *Notes of the 1st Workshop on Non-Silicon Computation*, pp. 70–77, February 2002.
- [170] C. Sechen and A. Sangiovanni-Vincentelli, "The TimberWolf placement and routing package," *Journal for solid State Circuits*, vol. SC-20, pp. 510–522, April 1985.
- [171] P. Banerjee and M. Jones, "A parallel simulated annealing algorithm for standard cell placement on a hypercube computer," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 34–37, November 1986.
- [172] L. K. Grover, "A new simulated annealing algorithm for standard cell placement," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 387–380, November 1986.
- [173] A. Casotto, F. Romeo, and A. L. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement on macro-cells," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, (Santa Clara, California), pp. 30–33, November 1986.