Synthesis of Multilevel Fault-Tolerant Combinational Circuits*

Alessandro Bogliolo

D.E.I.S.
University of Bologna
Bologna, I-40136

Maurizio Damiani

D.E.I.
University of Padova
Padova, I-35131

Abstract

In this paper we present a new approach to the design of multilevel fault-tolerant circuits. The approach is based on introducing a minimal amount of fault-masking redundancy during a multilevel logic optimization process. This is done by taking into account the degrees of freedom associated with internal don't care conditions. Experimental results obtained on several benchmark circuits compare very favourably with fault-tolerant implementations based on traditional gate-level strategies.

1 Introduction.

Digital systems are increasingly used in applications that require extremely high reliability. Some such applications include aerospace, transportation, control in harsh environments, *etc...*

Over the years, several techniques have been developed for improving the reliability of digital systems at all levels. An excellent survey of the subject is [1].

System-level techniques (such as *N-tuple Modular Redundancy* [2]) are essentially based on the addition of spare duplicate units. These units may contribute to the system's functioning at all times (static redundancy), or be used as replacements in case a failure is detected (dynamic redundancy). In particular, with static redundancy techniques, each functional part (*module*) of a given digital circuit is replicated N times so as to obtain N independent copies of each output signal. Each vector of N equivalent signals is then connected to a *restoring organ* (*i.e.*, a majority voting element). Failure of a single component is thus overrun by the other components, and the voter's output is thus a more reliable primary output.

*Supported in part by the ESPRIT III Basic Research Programme of the EC under contract No. 9072 (Project GEPPCOM)

These techniques can be employed at gate level as well. At this level, however, it is possible to take advantage of the intrinsic error masking capabilities of logic gates in order to avoid the explicit introduction of voting elements. With typical gate-level approaches to fault-tolerance (such as *quadded-logic* [3] and *interwoven-redundancy* [4]) logic gates (or subcircuits) are replicated and interconnected in a way that prevents the propagation of logic errors caused by internal faults [3, 4, 5, 6, 7].

System- and gate-level techniques preserve the topology of the original system. So they do not take into account the degrees of freedom avalaible to further optimize the fault-tolerant network. Instead, if fault-tolerance requirements are considered during logic synthesis, all the degrees of freedom available for logic optimization could ideally be used to reduce the amount of fault-masking redundancy. An alogrithm for the two-level synthesis of fault-tolerant digital circuits was proposed by Pradhan and Reddy [8], but no methods have been developed so far for the synthesis of multilevel fault-tolerant networks.

In this paper we take a step in this direction, and present a new approach to the design of fault-tolerant circuits, based on a multilevel logic synthesis paradigm. Each gate of an original arbitrary combinational network is re-synthesized so as to achieve fault tolerance with a minimum addition of hardware. This is accomplished by taking advantage of the error-masking properties of individual gates and of the functional redundancies already present in the network. In particular, in this paper we target networks that are **multilevel fault-tolerant**, *i.e.* that can confine the errors introduced by a fault in a small region. *Quadded* networks are an example of such networks.

We tested our algorithms against several benchmark combinational circuits. Substantial hardware savings (sometimes in excess of 40 %) have been obtained with respect to *quadding*.

2 Terminology.

Let $\mathcal{B}$ denote the Boolean set $\{0,1\}$. A k-dimensional Boolean vector $\mathbf{x} = (x_1, \cdots, x_k)$ is an element of the set $\mathcal{B}^k$ (bold-facing is used to denote vector quantities). A n_i-input, n_o-output Boolean function $\mathbf{f}$ is a mapping $\mathbf{f}$: $\mathcal{B}^{n_i} \rightarrow \mathcal{B}^{n_o}$. A scalar function f_1 **covers** f_2 (f_2 **implies** f_1, denoted by $f_1 \geq f_2$) if $f_1 = 1$ whenever $f_2 = 1$. An input vector $\mathbf{x}$ such that $f(\mathbf{x}) = 1$ is termed a **minterm** of f. Boolean functions can be represented symbolically by means of **Reduced Ordered Binary Decision Diagrams** (BDDs) [9, 10].

A **logic network** ($\mathcal{N}$) is a directed, acyclic graph (V, E). Vertices of the graph represent primary inputs/outputs or logic gates, while edges denote interconnections: an edge from a gate g_1 to a gate g_2 indicates that g_1 is used as input to g_2. We denote by $FI(g)$ and $FO(g)$ the sets of immediate fanin and fanout gates of a gate g, respectively. Each gate g of a logic network realizes a function of the primary inputs, denoted by $g(\mathbf{x})$.

In principle, the Boolean operation realized at each vertex is arbitrary. For the sake of simplicity, however, hereafter we assume that each vertex represents a NOR gate. Hence, for a gate g, the output function can be expressed as

$$g(\mathbf{x}) = (\sum_{g_i \in FI(g)} g_i(\mathbf{x}))'. \qquad (1)$$

2.1 Multilevel Optimization.

Multilevel logic optimization of a given Boolean network consists on the application of a suitable set of network tranformations that improve a target parameter (area, delay, power consumption) while preserving the network functionality [11, 12].

In this paper we consider an approach based on a sequence of local optimizations of internal gates. The optimization of a gate is carried out based on the following observations.

Eq. (1) indicates that the function $g(\mathbf{x})$ realized by a gate is the NOR of functions $g_i(\mathbf{x})$, each of which must be contained in $g'(\mathbf{x})$: $g_i(\mathbf{x}) \leq g'(\mathbf{x})$. Any function $f(\mathbf{x})$ such that $f \leq g'$ is an **implicant** of g'.

In general, a function g can be realized not only as the NOR of given functions g_i, but also as the NOR of other functions, maybe already available in the network. Local resynthesis attempts the optimization of each gate precisely by trying to re-express each gate function as a NOR of fewer and / or simpler functions.

3 Fault-Tolerance.

3.1 Faults and Fault-Tolerance.

As mentioned, we refer in this paper to NOR-only networks, for the sake of convenience. The results of the paper, however, can be extended to networks of arbitrary elementary gates.

We consider in this paper single stuck-at type faults, occurring either at a gate input or at a gate output. The following terminology will be used extensively afterwards to classify the impact of a fault in a network:

Definition 3.1 *Let $\mathcal{N}$ and $\mathcal{N}^*$ denote a fault-free and a faulty network, respectively. Let also $g^*(\mathbf{x})$ denote the function realized by gate g in the faulty network. For a given input configuration $\mathbf{x}$, we say that an* **error** *is present at the output of g if $g(\mathbf{x}) \neq g^*(\mathbf{x})$.*

Definition 3.2 *An error at a gate input is* **critical** *if it causes the input to take incorrectly the dominant value. It is* **subcritical** *otherwise. A fault is likewise named critical if it can cause at least a critical error to be present at the inputs of some gate.*

Fault-tolerance targets the design of networks that preserve correct behavior even in presence of a fault. Some faults, however, such as stuck-at's on primary inputs or outputs, are clearly inherently intolerable and cannot be targeted by any approach to fault-tolerance.

Definition 3.3 *We call* **internal faults** *the faults that can affect gate inputs and outputs, except those on primary inputs, those on interconnections from primary inputs, those on the primary outputs and critical faults at the inputs of the output-driving gates. For our purposes, a network if termed* **fault-tolerant** *if its functionality is not affected by any single internal fault.*

Hereafter, we refer implicitly **only** to internal faults.

3.2 Multilevel Fault-Tolerant Networks.

All gate-level techniques for constructing fault-tolerant logic networks take advantage of the masking properties of logic gates to compensate, as soon as possible, the errors produced by internal faults. Standard techniques, such as *quadding*, actually grant that **the effects of a single fault never propagate through more than one logic level**. This property motivates our definition:

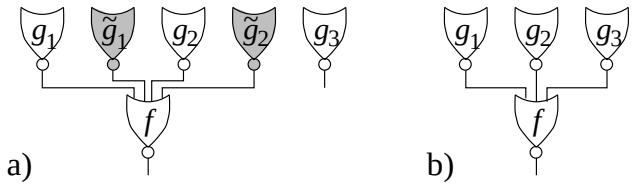

Figure 1: Two fault-tolerant implementations of the same Boolean function $f(\mathbf{x})$, obtained a) by *quadding* and b) by a minimum double-cover. The maps of the functions are shown in Fig. (2).

Definition 3.4 *A Boolean network $\mathcal{N}$ is said to be* **multilevel fault-tolerant** *if and only if no internal fault can cause an error that propagates through more than one level of logic.*

Existing design rules for fault-tolerance, however, are of an essentially **topological** nature. The following rules are employed, for instance, by *quadding*. They are trivially sufficient (but **not necessary**) to grant multilevel fault-tolerance:

Rule 1. *Logic levels alternate in such a way that the propagation of a critical error produces sub-critical errors at the next logic level.* This happens, for example, in NOR-only or NAND-only representations.

Rule 2. Two distinct gates realizing the same internal logic function are termed **equivalent**. *The fanin of each gate must contain only primary inputs or pairs of equivalent gates.* A sub-critical error on a gate input can thus be compensated by an error-free equivalent input.

Rule 3. Two gates g_1, g_2 with no common fanin are termed **independent**. *Equivalent gates feeding a common gate must be independent.* Otherwise a fault on a common input could produce multiple sub-critical errors at the inputs of another gate, which may go uncompensated.

It is worth noting that, in order to satisfy the above rules, *quadding* must actually quadruple all internal gates. Rules (1-3) do not take into account the functionality realized at the various gates. For instance, Rule (2) is too resctrictive: A sub-critical error affecting a gate input may also be masked by another, non-equivalent input.

Example 3.1 *Suppose the function $f(\mathbf{x})$, mapped in Fig. (2), is to be synthesized, using the implicant functions $g_1(\mathbf{x}), g_2(\mathbf{x}), g_3(\mathbf{x})$, realized by gates already present in the network. A minimal NOR realization of*

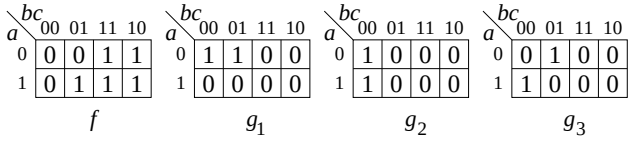

Figure 2: Maps of the completely specified Boolean functions $f(\mathbf{x}), g_1(\mathbf{x}), g_2(\mathbf{x}), g_3(\mathbf{x})$ realized by the gates of Fig. (1). $g_1(\mathbf{x}), g_2(\mathbf{x})$ and $g_3(\mathbf{x})$ are implicants of $f'(\mathbf{x})$.

function $f(\mathbf{x})$ is, for instance, $f(\mathbf{x}) = (g_1(\mathbf{x}) + g_2(\mathbf{x}))'$. In order to achieve fault-tolerance, Rule (2) would require the duplication of gates g_1 and g_2, as shown in Fig. (1a). Rule (2') allows us, instead, to find a double-cover with only one copy of g_1, g_2, g_3: $f(\mathbf{x}) = (g_1(\mathbf{x}) + g_2(\mathbf{x}) + g_3(\mathbf{x}))'$. With this solution, shown in Fig. (1b), we achieve the same degree of fault-tolerance without doubling any gate. □

In the next paragraph we replace Rules (1-3) with rules of **functional** nature, and we show that these rules are **necessary and sufficient** for a network to be multilevel fault-tolerant.

3.3 Functional Rules for Multilevel Fault-Tolerance.

As we consider NOR-only networks, Rule (1) needs not be modified. The following Theorem provides a functional criterion replacing Rule (2).

Theorem 3.1 *A NOR-gate g tolerates all single sub-critical errors affecting its inputs if and only if each minterm $\mathbf{x}$ of $g'(\mathbf{x})$ is covered at least twice, (i.e.there are at least two input gates $g_1, g_2 \in FI(g)$ such that $g_1(\mathbf{x}) = g_2(\mathbf{x}) = 1$).*

Proof 3.1 *(If part): Consider a subcritical error affecting an input signal g_1 of g. This error turns a logic 1 on the input into a logic 0. Since each minterm $\mathbf{x}$ of $g'(\mathbf{x})$ covered by g_1 is also covered by an error-free input signal g_2, the sub-critical error is masked for each relevant input configuration.*

(Only if part): Suppose, by contradiction, that there exists a minterm $\mathbf{x}$ of $g'(\mathbf{x})$ covered only by g_1. If a s-a-0 fault occurs on g_1 the gate output would take the incorrect logic value 1 for input vector $\mathbf{x}$. Hence the fault would not be tolerated, a contradiction. □

By Theorem (3.1) Rule (2) can then be replaced by the following:

Rule 2'. *Each minterm of each internal function must be covered by a primary input or by at least two implicants.*

Rule (2') no longer requires the pairwise equivalence of inputs. Hence, already existing, different signals can be used to mask subcritical errors. This actually was already shown in Example (3.1).

Rule (3) was required to grant that no critical fault could produce multiple subcritical errors at the inputs of some other gate. We can replace this need with the following functional rule:

Rule 3'. *For each minterm* $\mathbf{x}$ *of a function* $f(\mathbf{x})$*, at least two of the functions covering* $\mathbf{x}$ *must correspond to independent gates.*

Theorem (3.2) below shows that Rules (2') and (3') are indeed **necessary and sufficient** for a network to be multilevel fault-tolerant:

Theorem 3.2 *A NOR-only network* $\mathcal{N}$ *is multilevel fault-tolerant if and only if for each gate* $g \in \mathcal{N}$ *and for each minterm* $\mathbf{x}$ *of* $g'(\mathbf{x})$*, there exist at least two independent gates* $g_j, g_k \in FI(g)$ *such that* $g_j(\mathbf{x})$ *and* $g_k(\mathbf{x})$ *cover* $\mathbf{x}$*.*

Proof 3.2 *(If part): We need to prove that no single fault causes an error to propagate through more than one logic level. To this regard, notice that:*

1) Theorem (3.1) insures that, because of the double covering of each internal function, all single subcritical faults are masked inside the network.

2) A critical fault affecting the output of a gate f *can only produce subcritical errors at the output of the gates in* $FO(f)$*. Consider a gate* g *with an input* $g_j \in FO(f)$*. Each minterm* $\mathbf{x}$ *of* $g'(\mathbf{x})$ *covered by* $g_j(\mathbf{x})$ *is also covered by an independent signal* $g_k(\mathbf{x})$*. Hence, gates* g_j *and* g_k *share no inputs and* $g_k \notin FO(f)$*, i.e., it is not affected by the fault of* f*. Gate* g_k *thus masks the error on* g_j *for the input configuration* $\mathbf{x}$*. Since such an error-free signal exists for each minterm covered by each gate* $g_j \in FO(f)$*, the effects of the critical fault are always masked at the next logic level.*

3) The case of a critical fault on a gate input can be handled just as the previous case.

(Only if part): Suppose, by contradiction, that there exists a minterm $\mathbf{x}$ *of some internal signal* $g'(\mathbf{x})$ *covered only by a set* $\mathcal{S}$ *of dependent gates, i.e.sharing at least an input* f*. A critical fault at the output of* f *would cause all the signals of* $\mathcal{S}$ *to take the incorrect logic value 0. This multiple subcritical error would then propagates through gate* g*, for input vector* $\mathbf{x}$*. Therefore, the critical fault of the internal signal* f *would cause an error that propagate through two logic levels. Hence, the network is not multilevel fault-tolerant, a contradiction.* □

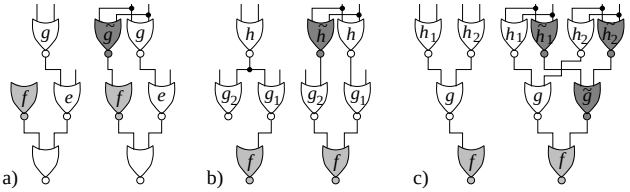

Figure 3: There are three main cases in wich the resynthesis of a gate f possibly needs the duplication of some portion of the network. Light shading denotes the gate f under optimization, while dark shading denote the duplicated gates.

4 Synthesis of Multilevel Fault-Tolerant Networks.

In this section we illustrate the basic algorithms involved in the construction of a minimal multilevel fault-tolerant network. This construction entails two-level re-synthesis of portions of the network. The next paragraph describes how the classical two-level synthesis paradigm must be modified to account for Rules (2') and (3').

4.1 Two-Level Synthesis

The two-level synthesis engine takes as input a single-output, two-level NOR-NOR network, realizing a function f. The flow of two-level synthesis follows the two-step paradigm of implicant extraction and minimum-cost covering [13]. Since we are working in a multiple-level synthesis environment, implicants of f are selected among other, already existing internal functions and their logic NORs [11].

The output is a two-level, NOR-NOR network that satisfies Rules (2') and (3'). In particular, it finds a cover of a function $f'(\mathbf{x})$ such that:

1) each minterm $\mathbf{x}$ is covered at least twice;

2) for each minterm $\mathbf{x}$, at least two of the implicants covering $\mathbf{x}$ must be generated by independent gates.

Requirements (1) and (2) do not add to the complexity of implicant extraction. Instead, they make the construction of a minimum-cost cover more complex. The classical covering step is thus modified as follows:

1) Whenever an implicant is included in a partial cover, it is not removed from the list of candidate implicants. This means that it is still available for inclusion a second time.

2) Whenever an implicant is included in a partial cover, its **cost** is dynamically evaluated as: i) the cost of its implementation, plus ii) the cost possibly induced by the necessity of duplicating portions of hardware in order to achieve independence.

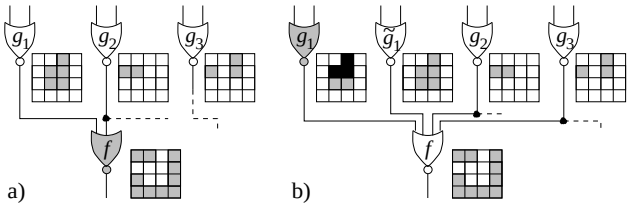

Figure 4: A simple realization of a Boolean function f and its fault-tolerant version. In the maps of the functions 1's are shaded. Notice that the double covering requirements causes some minterms of $f'(\mathbf{x})$ to be covered three times. These minterm can be regarded as *observability don't cares* for implicant g_1.

There are three cases that require duplication. They are reported in Fig. (3).

Case 1. This case is illustrated in Fig. (3a). Gate f, under optimization, must be independent from another gate (in this case, gate e). Hence, any implicants of f' that are also inputs to e must be duplicated if they are to be included in a cover of f'.

Case 2. To grant satisfaction of Rule (3'), implicants covering a common minterm of a function f' must be made independent by duplicating common fanin, as shown in Fig. (3b).

Case 3. If an implicant g_1 is essential to f' (*i.e.*, it is the unique implicant that can cover some minterm of f'), then it must necessarily be selected twice. Hence, gate g_1 has to be duplicated as well as its fanin, as shown in Fig. (3c).

It is worth noting that *quadding* can be regarded as a systematic use of the duplications of Case (3), while Cases (1) and (2) are not acknowledged. Hence, *quadding* represents a **worst-case bound** to our method.

4.2 Multiple-Level Synthesis.

The input of the synthesis algorithm is an **arbitrary** NOR-only network. The network may or may not have redundancies or fault-tolerance properties. Gates are topologically sorted and then visited in order, starting from primary outputs. Each gate is regarded as the output of a two-level network, which is re-synthesized using the algorithms of Section (4.1). In this way, we are sure that at every time the already-synthesized portion of the network is multilevel fault-tolerant. The program terminates when all gates have been visited.

4.3 Observability *don't cares*.

The notion of **observability** *don't cares* has been proficuos in the development of accurate multiple-level logic optimization algorithms [11, 12].

It is possible to port this notion also in the synthesis of fault-tolerant networks. Consider, to this regard, the situation of Fig. (4b). Function $f'(\mathbf{x})$ is double-covered by functions $g_1(\mathbf{x}), \tilde{g}_1(\mathbf{x}), g_2(\mathbf{x}), g_3(\mathbf{x})$. Inspection of the covering shows that some minterms of f' are actually covered three times. Corresponding to these minterms, the value of an implicant function (say, g_1) can be changed without affecting the double-covering of f. Hence, the value of g_1, is irrelevant (**don't care**) for these minterms. These *don't cares*can be spent for the optimization of g_1.

It could be shown that these observability *don't care* conditions can be computed and propagated essentially by the traditional means [14]. The formal proof, however, is out of the scope of this paper.

5 Implementation and Experimental Results.

We have implemented in C the algorithms described in this paper. We have used a standard BDD package to implement the basic manipulation routines for Boolean functions. The algorithms were tested against a set of well-known logic synthesis benchmarks [15, 16]. The circuits were initially optimized by running the optimization program SIS [17].

Table (1) reports the initial circuit statistics, in terms of input, output, NOR gate and interconnection counts. The data refer to the optimized circuits. The second column refers to the gate and interconnection counts of the *quadded* versions of the networks. The last column reports the gate and interconnection counts of the circuits obtained by our algorithm (named FT_SYN), as well as the CPU time required. Time was taken on a SUN SPARCstation IPX.

The average improvement with respect to quadding is of 18%.

6 Conclusions and Future Work.

Traditional means of designing fault-tolerant logic networks used topology-based replication techniques, that are inefficient in terms of area and may add excessive redundancies.

In this paper we took a more global **synthesis** approach. We showed that it is possible to take into account internal *don't care* conditions and synthesize a fault-tolerant network by locally adding just the minimal amount of extra hardware needed. The experimental results in this sense are extremely encouraging.

Circuit	Original Network				*Quadding*		FT_SYN		
	Ins	Outs	Gates	Conns	Gates	Conns	Gates	Conns	CPU
cm42a	4	10	20	41	**38**	101	**29**	91	1
cm163a	16	5	56	103	**175**	490	**131**	372	3
decod	5	16	41	67	**100**	264	**68**	218	1
x2	10	7	30	72	**95**	313	**82**	270	2
cc	21	13	61	102	**269**	410	**150**	397	7
f51m	8	7	72	165	**275**	1112	**252**	1056	33
apex7	49	37	198	393	**576**	1823	**534**	1805	101
alu2	10	6	232	570	**850**	3720	**794**	3613	358
alu4	14	8	488	1128	**1820**	7662	**1730**	7643	5301
s208	19	10	37	81	**119**	382	**95**	319	6
s444	24	27	108	226	**368**	1280	**304**	1117	14
s526	24	27	121	239	**365**	1141	**284**	940	21
s953	45	52	266	644	**795**	2883	**704**	2783	579
s1196	32	32	383	956	**1232**	5068	**1128**	4851	1116

Table 1: Experimental results on several benchmark circuits. Data refer to the combinational portion of the sequential circuits.

In this paper, we targeted the synthesis of multi-level fault-tolerant networks, having in addition strong fault-confinement properties. In the future we plan to generalize the ideas and algorithms to tackle more general classes of fault-tolerant networks.

References

[1] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design.* Digital Press, 1982.

[2] R. E. Lyions and W. Vanderkulk, "The Use of Triple Modular Redundancy to Improve Computer Reliability," *IBM J. Res. Dev.*, pp. 200–209, 1962.

[3] J. G. Tryon, "Quadded logic," in *Redundancy Techniques for Computing Systems* (R. H. Wilcox and W. C. Mann, eds.), pp. 205–228, Spartan, 1962.

[4] W. H. Pierce, *Failure Tolerant Computer Design.* New York: Academic, 1965.

[5] T. F. Schwab and S. S. Yau, "An Algebraic Model of Fault-Masking Logic Circuits," *IEEE Transactions on Computers*, vol. C-32, pp. 809–825, Sept. 1983.

[6] A. E. Barbour and A. S. Wojcik, "A General, Constructive Approach to Fault-Tolerant Design Using Redundancy," *IEEE Transactions on Computers*, vol. 38, pp. 15–29, Jan. 1989.

[7] A. E. Barbour, "Solution to the Minimization Problem of Fault-Tolerant Logic Circuits," *IEEE Transactions on Computers*, vol. 41, pp. 429–443, Apr. 1992.

[8] C. K. Pradhan and S. M. Reddy, "Design of Two-Level Fault-Tolerant Networks," *IEEE Transactions on Computers*, vol. C-23, pp. 41–47, Jan. 1974.

[9] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 33, pp. 677–691, Aug. 1986.

[10] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Proc. of Design Automation Conf.*, pp. 40–45, June 1990.

[11] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Transactions on CAD*, vol. 6, pp. 1062–1081, Nov. 1987.

[12] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, C. R. Morrison, R. L. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "Multilevel Logic Minimization Using Implicit Don't Cares," *IEEE Transactions on CAD*, vol. 7, pp. 723–740, June 1988.

[13] E. J. McCluskey, *Logic Design Principles: with Emphasis on Testable Semicustom Circuits.* Prentice Hall Int., 1986.

[14] M. Damiani and G. DeMicheli, "Don't Care Set Specifications in Combinational and Synchronous Logic Circuits," *IEEE Transactions on CAD*, vol. 12, pp. 365–388, Mar. 1993.

[15] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. of IEEE Int. Symp. on Circuit And Systems*, pp. 1929–1934, 1989.

[16] R. Lisanke, "Logic Synthesis and Optimization Benchmarks – User Guide," tech. rep., Microelectronics Center of North Carolina, 1988.

[17] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," tech. rep., University of California, Berkeley, 1992.