Coping with the variability of combinational logic delays

J. Cortadella
Univ. Politècnica Catalunya
Barcelona, Spain

A. Kondratyev
Cadence Berkeley Labs
Berkeley, CA

L. Lavagno
Politenico di Torino
Torino, Italy

C. Sotiriou
ICS-FORTH
Crete, Greece

***Abstract*— This paper proposes a technique for creating a combinational logic network with an output that signals when all other outputs have stabilized. The method is based on dual-rail encoding, and guarantees low timing overhead and reasonable area and power overhead.**

We discuss various scenarios in which completion detection can be used to measure the delay of a synchronous circuit at fabrication time or at run time, and of an asynchronous circuit at run time. We conclude by showing, on a large set of benchmarks, the effectiveness of the proposed technique.

I. Introduction

Variability is a growing concern to digital circuit designers. While statistical timing analysis techniques try to apply various empirically derived models to combine together all sources of variability along a critical path, designers would like to have a way to *measure* rather than just *predict* the actual delay of a circuit. The reason is that a large fraction of circuits work faster or much faster than worst-case analysis would predict. Unfortunately, delay fault testing is a costly proposition, because it requires the designer to start from a very expensive initial two-level representation, and then use only a subset of logic optimizations [1].

The goal of this paper is to propose a technique, that enables to easily measure exactly when a combinational circuit is done computing. In practical terms, we guarantee that *without almost any timing overhead* and with some area overhead, every combinational logic block has an additional completion detection output that rises a few gate delays after the last primary output has settled. This paper improves with respect to the state of the art in the following directions, by proposing:

- a technique for dual-rail network creation with much lower area, power and delay overhead than previously known techniques, and requiring only standard static CMOS gates.
- a novel method, using timing assumptions rather than completion detection, for fast reset to the spacer (inactive) state.

We show in Section VI that circuits obtained using our logic synthesis flow have a *nominal* delay which is on average within 33% of the corresponding circuit optimized using traditional techniques, with a 100% area overhead. However, by using the completion detection output their *true* delay can be measured and used to reliably latch their output values, thus reducing the *margins* that must be taken at design time.

Two key assumptions motivate the use of circuits with completion detection: (1) The difference between worst case and true delays is in the range of 60-100% (see Section II), and (2) the fraction of "truly critical" register-to-register combinational logic blocks is small (10-20% according to internal sources). We believe that, by using circuits with completion detection, one can achieve at least *25% of performance increase with 10% to 20% penalty in power and area* purely remaining in the synchronous domain.

II. Motivation

The main advantage of completion detection is the ability to run the circuit using true rather than worst-case delays. Figure 1 shows the delay penalty stemming from two main factors: a) static timing analysis technology that estimates delays for the worst case scenario and b) clocking overhead in synchronous circuits. These penalties are:

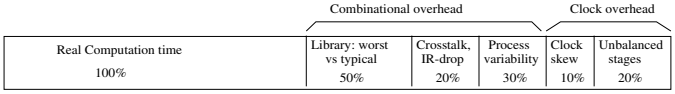

Fig. 1. Delay penalties in the modern synchronous methodology

1) The difference in the library files between typical and worst case. This may account for up to a 50% penalty.
2) Conservative estimation of the slowdown caused by signal integrity violations: namely IR drop and crosstalk. Each of them could be responsible for a 10% penalty [2], [3].
3) Variations of delays due to processing variations. We consider this to account for a 30% penalty for the latest technologies (see e.g. [4]), however even more aggressive forecasts exist.
4) Clock skew, which is about 10% of the clock cycle for the modern ASIC methodology [5].
5) Unbalancing of synchronous stages, which increases the clock cycle with respect to the mean cycle time by up to 20% [5].

The first three sources come from the combinational logic itself, while the last two are coming from the clocking scheme. This margin shows the optimization room that is available when implementing circuits with completion detection. Part of this advantage is offset by the overheads due to the dual rail conversion, the two-phase operation and the completion detection logic itself. However, as shown in Section VI, these penalties are lower than the combinational logic penalty cited above. Clock penalties, as well as exploiting data-dependent delays, may provide additional performance improvements when the circuits with completion detection are used in an asynchronous environment, as discussed below.

III. Previous Work

One of the most popular operation modes in asynchronous design alternates data communication between set and reset phases [6], with the two-fold goal of: a) providing a clean separation between two consecutive data sets and b) ensuring monotonic behavior at circuit outputs, avoiding spurious transitions known as hazards [7].

The simplest scheme for such communication is given by dual-rail encoding, in which each signal a is represented by two wires: the 0 and 1 are encoded as 01 and 10, respectively. The spacer is usually encoded as 00. The fact that exactly one of the wires in each dual-rail pair goes high, tells that the output has settled and the set phase is over.

Two regular methods for implementing an arbitrary Boolean network with two-phase operation and dual-rail encoding were suggested in [8] (DIMS method) and [9] (NCLX approach). In both techniques every node of a network representing Boolean function f is implemented by two logic cones $f.t$ and $f.f$, where $f.t$ represents the original function f, while $f.f$ represents its inverse $\overline{f}$.

Proceedings of the IEEE International Conference on Computer Design (ICCD'04)
1063-6404/04 $ 20.00 IEEE

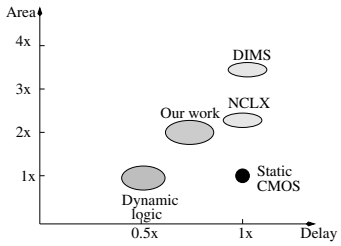

Fig. 2. Architecture-dependent area-delay trade-offs

NCLX design flow [9] is the closest approach to ours. In [9], a dual-rail network is created by simply adding duals to every gate and eliminating inversions by swapping complementary rails. Completion detection is carried out separately from the data evaluation.

Unfortunately, both approaches [8] and [9] did not manage to deliver a speed advantage, by exploiting true delays instead of worst-case approximations, because of the need to use two operational phases (set and reset) in a single computational cycle. The nearly 2x delay penalty from the two-phase operational mode is difficult to offset by reducing delays from worst-case to true.

Dual-rail networks with two-phase operation are also used in dynamic logic synthesis [10], [11]. We consider domino logic, even though the concerns we present extend to other dynamic logic styles. A major limitation of domino logic is that it can only implement non-inverting logic (other styles require strict layering of, e.g. pre-charged and pre-discharged gates). Hence, complements of internal signals need to be realized through separate cones of logic using dual functions and giving rise to partially or fully dual-rail circuits. Similar to asynchronous systems, dynamic circuits work in two phases, namely *pre-charge* and *evaluate*. This feature allows dynamic logic to achieve significant speed improvements over static CMOS, because the reset (pre-charge) phase is very short. Unfortunately, the inherent noise sensitivity and charge-sharing problems associated with this design style limit its application to small timing-critical portions of systems, which are usually handcrafted.

Our paper makes an attempt to take the best of both worlds and develop a design flow that combines performance advantages inherent in true delay measurement, with the robustness and simplicity of static standard cell CMOS, all within a standard ASIC design flow. The objective of this this work in terms of area-delay trade-offs is shown in Figure 2. The figure already takes into account the fact that circuits with completion detection (our work, DIMS and NCLX) work with true delays, and hence faster than static CMOS.

IV. Monotonic Boolean networks

A. Hazards

Glitches in combinational circuits are called *hazards* [7]. A hazard is a transient change on a signal caused by the gate propagation delays. To characterize the dynamic behavior of a combinational circuit we need to define the allowed transitions at the primary inputs of the circuit.

Definition 4.1 (Monotonic Input Transition (MIT)): A MIT consists of the change in the value of a subset of primary inputs in the same direction, i.e. all changes are either $0 \rightarrow 1$ or $1 \rightarrow 0$.

Definition 4.2 (Monotonic operation mode (MOM)): The MOM works by iteratively alternating two sub-phases, each of which is a MIT: (a) a subset of the inputs changes monotonically and (b) a subset of the outputs changes monotonically.

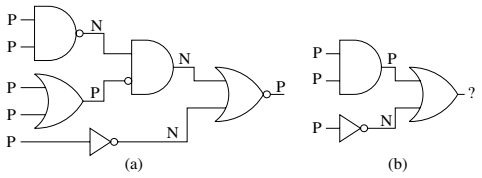

Fig. 3. (a) Monotonic network, (b) non-monotonic network.

In order to use this mode of operation for completion detection, the set of changing outputs must be known in advance. In particular, with dual-rail encoded signals exactly one input and one output in every pair changes value in each phase.

B. Monotonic Boolean networks

We assume the reader to be familiar with the basic concepts on *Boolean functions* and *Boolean networks*. Each non-input node n_i of a Boolean network has an associated Boolean function f_i in terms of its local fanin.

Now we introduce the concept of monotonicity of the nodes of a Boolean network.

Definition 4.3 (Monotonicity): A node n_i with local function f_i is *positive* if for each input x_i in its local fanin it holds that: – x_i is positive (negative) and f_i is increasing (decreasing) in x_i A node n_i with local function f_i is *negative* if for each input x_i in its local fanin it holds that: – x_i is negative (positive) and f_i is increasing (decreasing) in x_i A node is *monotonic* if it is either positive or negative.

While this definition works for any network which admits a consistent labeling of each node (including primary inputs and primary outputs) as positive or negative, for the sake of simplicity in the following we assume that all primary inputs are dual-rail encoded with spacer 00, and hence are defined to be *positive*.

Definition 4.4 (Monotonic and positive Boolean networks): A Boolean network is monotonic (MBN) if all its nodes are monotonic. An MBN is positive (PBN) if all its nodes are positive (e.g. a domino logic network is positive).

Theorem 4.1 (Hazard-free behavior of an MBN): An MBN is hazard-free under a monotonic input transition.

Proof: See [12] ■

Figure 3 depicts a monotonic and a non-monotonic network. The labels `P` and `N` indicate the positive and negative nodes, respectively. The network in Figure 3(b) is not monotonic since it is not possible to assign a phase to the output node. Note that the input transition $000 \rightarrow 111$ applied to all inputs may produce a glitch at the output, if the inverter and the OR gate switch before the AND gate.

C. Use cases

Combinational logic with completion detection can be used in two different environments: synchronous and asynchronous (see Figures 4(a) and 4(b)).

In the synchronous environment, the output of the completion detector (denoted by CD in Figure 4(a)) is used to check whether a circuit can work at a given clock frequency. If a change at the output of a completion detector happens *before* the clock edge, then data inputs of receiving flip-flops have settled before the clock rises and no synchronization fault occurs. If, however, the completion detector makes a transition *after* the clock edge, then there are chances that erroneous values have been stored in flip-flops, and the *error* signal must be raised. The *error* signal may be used: (1) during production test to bin chips according to their performance or, (2) to provide an

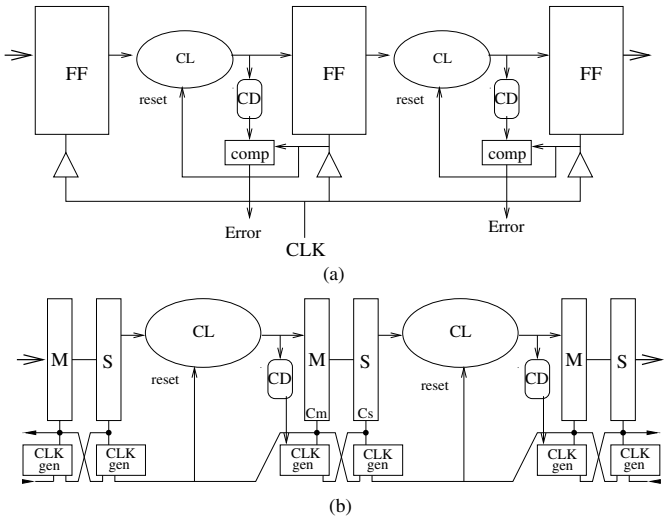

Fig. 4. Architectures with completion detection

on-line testing capability, assuming that the system may roll back and repeat the computation cycle or is capable of stretching the clock.

In order to give a qualitative estimation of the potential performance benefits from using circuits with completion detection, let us compare their cycle time with the cycle time of conventional synchronous designs.

The cycle time in a synchronous circuit must satisfy the following timing constraints:

$CL_{max} + T_{skew} + T_{setup} + T_{CQ\_max} \leq T_{cycle}$ *setup constraint*

$CL_{min} + T_{CQ\_min} \geq T_{hold} + T_{skew}$ *hold constraint*

In these inequalities CL_{max} and CL_{min} stand for worst and best case propagation delays through combinational logic, while T_{CQ} is the clock-to-output delay of a flip-flop.

The cycle time in a circuit with a completion detection also needs to satisfy hold and setup constraints which are:

$CL_{true} + T_{skew} + T_{setup} + T_{CQ\_max} + T_{reset\_max} + T_{CD} \leq T_{cycle}$

$CL_{min} + T_{CQ\_min} \geq T_{hold} + T_{skew}$

where CL_{true} is the true delay of combinational logic, while T_{CD} and T_{reset_max} are the delay overheads of the architecture with completion detection.

A circuit with completion detection must satisfy an additional timing assumption in order to function correctly in the reset phase:

$T_{reset\_min} \geq T_{hold}$

where T_{reset_min} is the minimum delay for propagating spacer values to the outputs of the combinational logic. This constraint is similar to the hold constraint and is needed to ensure that spacer does not overwrite data values before they are stored in registers.

One can see that a synchronous architecture with completion detection provides a performance advantage if

$CL_{true} + T_{reset\_max} + T_{CD} < CL_{max}$

Experimental results give a quantitative estimation of T_{reset_max} and T_{CD}, showing that the conditions of the above inequality are indeed met very often.

In order to use circuits with completion detection in an asynchronous environment, one can exploit standard micropipeline-based architectures [13]. For example, they are suitable for desynchronized circuits [14], which are derived from synchronous synthesizable specifications. The only difference is that the request signals triggering controllers are derived from completion detectors rather than from matched delays, as shown in Figure 4(b).

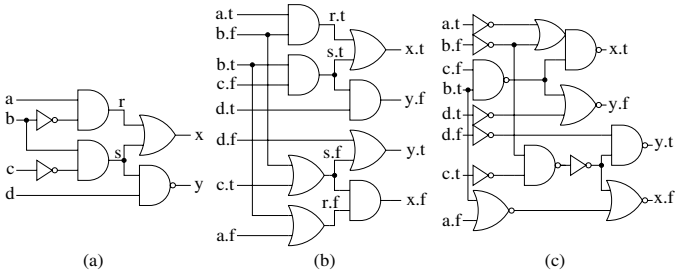

Fig. 5. From single-rail (a) to dual-rail (b), and technology mapping (c).

V. Transformations to obtain and preserve a Monotonic Boolean Network

A. Conversion to Monotonic Boolean Network

We can generate an MBN from a generic Boolean network by using the dual-rail encoding of inputs and outputs.

We now present a technology-independent conversion to a MBN[1]:

1) From each primary input x, two primary inputs x^t and x^f are created to represent the *true* and *false* evaluations of x.
2) From each node implementing the function $y_i = f_i(x_1, \ldots, x_n)$, two nodes are created with the functions

$$y_i^t = \mathsf{DR}\left(f_i(x_1, \ldots, x_n)\right) \qquad y_i^f = \mathsf{DR}\left(\overline{f_i}(x_1, \ldots, x_n)\right)$$

where DR denotes the transformation of the function into positive unate, changing also its input signals from x_i to x_i^t and x_i^f as appropriate. Formally, the transformation DR can be recursively defined as follows, using Shannon's expansion:

$$\mathsf{DR}(0) = 0 \qquad \mathsf{DR}(1) = 1$$
$$\mathsf{DR}(x \cdot f_x + \overline{x} \cdot f_{\overline{x}}) = x^t \cdot \mathsf{DR}(f_x) + x^f \cdot \mathsf{DR}(f_{\overline{x}})$$

As an example, the function $y = a\overline{b} + b(c + \overline{d})$ would be converted into

$$y^t = \mathsf{DR}\left(a\overline{b} + b(c + \overline{d})\right) = a^t b^f + b^t(c^t + d^f)$$
$$y^f = \mathsf{DR}\left(\overline{a\overline{b} + b(c + \overline{d})}\right) = (a^f + b^t)(b^f + c^f\, d^t)$$

Figure 5 depicts a complete example of how a single-rail circuit (a) is converted into a dual-rail circuit (b). After technology mapping, the circuit in Figure 5(c) can be obtained.

In [15], a set of transformations that do not introduce new hazards in Boolean networks was presented. They extend the set originally given in [7] and cover, among others, the conventional algebraic optimizations performed during technology-independent logic synthesis [16] and during technology mapping.

Thus, logic synthesis and technology mapping can be performed on MBNs as long as the set of transformations fall into the category of hazard-non-increasing.

B. Fast reset and completion detection

An MBN would operate at half the speed of its original counterpart, due to the need for resetting all primary inputs before another monotonic phase can begin. We can speed up this reset phase by inserting signals that force gate outputs to their "quiescent" value (*i.e.* to the value that they assume when all dual-rail inputs are at 0, in the spacer condition). Of course there is a trade-off here between the number of gates which are reset in this manner (the more numerous, the faster becomes the reset phase) and the large

[1] A conversion for technology-mapped circuits is also presented in [12]

IWLS circuit	SR delay (ns)	SR area (μm^2)	DR delay increase	DR area increase	DR_R delay increase	DR_R area increase	DR_CD delay increase	DR_CD area increase
C1908	2.07	9709	-4%	79%	0%	91%	12%	99%
C2670	1.53	13592	3%	79%	9%	93%	34%	121%
C3540	2.59	16665	7%	79%	14%	90%	23%	94%
C5315	2.08	33737	6%	77%	40%	90%	58%	103%
C7552	2.16	35294	26%	81%	32%	100%	50%	111%
alu4	1.85	9079	0%	80%	4%	90%	13%	93%
apex6	0.89	11006	11%	91%	15%	107%	57%	139%
dalu	1.34	10998	-2%	84%	2%	94%	18%	99%
des	1.46	51910	34%	73%	35%	99%	63%	108%
frg2	0.94	10965	3%	67%	5%	82%	45%	116%
i10	2.34	32489	5%	80%	41%	96%	59%	113%
i2	0.73	3062	9%	94%	16%	102%	21%	105%
i7	0.71	8754	-40%	62%	-39%	72%	8%	96%
i8	1.05	13149	-11%	86%	-7%	95%	25%	115%
i9	1.09	9612	-5%	77%	0%	85%	29%	109%
k2	1.18	14942	0%	87%	4%	102%	30%	112%
pair	1.49	21544	6%	86%	12%	98%	37%	116%
rot	1.27	9132	9%	79%	13%	95%	42%	138%
vda	0.96	9059	0%	90%	6%	99%	35%	108%
x3	0.72	10669	-2%	91%	4%	203%	55%	236%
x4	0.61	5749	-3%	72%	3%	86%	59%	116%
average			3%	77%	11%	95%	33%	109%

TABLE I

EXPERIMENTAL RESULTS FOR IWLS BENCHMARKS.

DLX Circuits	SR delay (ns)	SR area (um^2)	DR delay (ns)	DR area (um^2)	DR_R delay (ns)	DR_R area (um^2)
IF	1.174	36465.7	1.371	57715.3	1.404	63786.6
ID	5.561	387083	7.63	691559	7.939	765135
EX	4.106	242852	6.697	450832	6.835	484527
MEM	0.55	4591.42	0.576	8537.76	0.638	10966.3
average	100%	100%	130%	177%	136%	203%

TABLE II

EXPERIMENTAL RESULTS FOR P&R DLX.

capacitance connected to the reset signal (which may end up slowing down the circuit too much).

Furthermore, the network that detects when all outputs have stabilized can be built simply by using an `or` gate for each dual-rail pair, whose output rises when one of the signals rises, and a tree of `and` gates.

VI. RESULTS

The method described in Sect. V has been automated and a design flow has been created. The flow is compatible with commercial EDA tools and, in particular, has been integrated with the Synopsys Design Compiler^{TM}. The steps of the flow are the following: (1) conversion from single-rail to dual-rail, (2) technology mapping and (3) buffer optimization.

The flow was applied to the largest IWLS combinational benchmarks (more than 20 examples) and to a set of larger Verilog RTL circuits, *i.e.* the pipeline stages of a DLX CPU. All circuits were mapped to the UMC 0.18μm technology library. To evaluate the impact of the physical part of the flow, the synthesised and DR-transformed stages of the DLX CPU were P&R using Cadence SOC Encounter^{TM}.

The results for IWLS combinational benchmarks and DLX are shown in Tables VI and VI respectively. The shorthands SR, DR, DR_R and DR_CD in the column labels correspond to Single-Rail, Dual-Rail, Dual-Rail with Fast Reset, and Dual-Rail with Fast Reset and Completion Detection respectively.

The delay penalty for the *data phase* of the dual-rail circuits, indicated by columns DR and DR_R, is relatively small. The penalty due to completion detection, which is currently larger, can be reduced by considering it only for critical outputs. The area penalty is about two-fold.

Note, that according to Table VI area and delay penalties after P&R are consistent to those obtained after logic synthesis.

As dual-rail circuits operate using the two-phase discipline, the *reset phase* delay must also be taken into account. For the circuits presented here, with reset employed every 6 circuit levels, the reset time was measure to be in the range of 0.16ns and 0.22ns.

VII. CONCLUSIONS

This paper proposes a novel technique for creating circuits with completion detection, based on dual-rail encoding. We describe both the theory for ensuring correct hazard-free operation, and implementations both at the technology-independent and at the technology-dependent levels. The experimental results show that a 100% area and power increase and a 30% delay increase are sufficient to obtain a circuit that is fully able to signal when its outputs have stabilized.

Bearing in mind the margin of 60-100% between worst case and true delays, one can conclude that circuits with completion detection could operate at 25-65% higher clock frequencies than traditional ones.

This enables for the first time a trade-off between applying completion detection to the whole design, and achieving the 25-65% speedup with 100% area and power penalty, versus applying it only to the most critical stages of logic, and gaining less speed with significantly smaller penalty in area and power. Exploring this trade-off is left to future work.

Acknowledgments. This work has been partially funded by CICYT TIC2001-2476 and ACiD-WG (IST-1999-29119).

REFERENCES

[1] S. Devadas and K. Keutzer, "Synthesis of robust delay-fault testable circuits: Theory," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 1, pp. 87–101, Jan. 1992.

[2] "User guide," in *Celtic User Manual, Cadence Design Systems, Inc.*, 2004.

[3] "Datasheet," in *Physical Studio Datasheet, Sequence Design, Inc.*, 2003.

[4] S. Nassif, "Modeling and analysis of manufacturing variations," in *Proc. of Asia and South Pacific Design Automation Conference*, May 2001.

[5] D. Chinnery and K. Keutzer, "Reducing the timing overhead," in *Closing the Gap between ASIC and Custom: Tools and Techniques for High-Performance ASIC design*. Kluwer Academic Publishers, 2002, ch. 3.

[6] C. L. Seitz, "System timing," in *Introduction to VLSI Systems*, C. A. Mead and L. A. Conway, Eds. Addison-Wesley, 1980, ch. 7.

[7] S. H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, John Wiley & Sons, Inc., 1969.

[8] J. Sparsø and J. Staunstrup, "Delay-insensitive multi-ring structures," *Integration, the VLSI journal*, vol. 15, no. 3, pp. 313–340, Oct. 1993.

[9] A. Kondratyev and K. Lwin, "Design of asynchronous circuits by synchronous CAD tools," in *Proc. ACM/IEEE Design Automation Conference*, June 2002.

[10] M. Prasad, D. Kirkpatrick, R.Brayton, and A. sangiovanni Vincentelli, "Domino logic synthesis and technology mapping," in *Proc. International Workshop on Logic Synthesis*, vol. 1, 1997.

[11] R. Puri, A. Bjorksten, and T. Rosser, "Logic optimization by output phase assignment in dynamic logic synthesis," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1996, pp. 2–8.

[12] J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Coping with the variability of combinational logic delays," `www.lsi.upc.es/~jordicf/publications/pdf/iccd04rep.pdf`, July 2004.

[13] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[14] I. Blunno, J. Cortadella, A. Kondratyev, L. Lavagno, K. Lwin, and C. Sotiriou, "Handshake protocols for de-synchronization," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. IEEE Computer Society Press, Apr. 2004, pp. 149–158.

[15] D. S. Kung, "Hazard-non-increasing gate-level optimization algorithms," in *Proc. International Conf. Computer-Aided Design (ICCAD)*, 1992, pp. 631–634.

[16] R. Brayton, G. Hachtel, and A. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 264–300, 1990.