Don't Care Set Specifications in Combinational and Synchronous Logic Circuits

Maurizio Damiani and Giovanni De Micheli, *Senior Member, IEEE*

Abstract—**We present a unified framework for the specification and computation of *don't care* conditions for combinational and synchronous multiple-level digital circuits. We characterize such circuits in terms of graphs, logic functions and *don't care* conditions induced by the external and internal interconnections. We model the replacement of a gate in a synchronous logic network by a perturbation of the corresponding logic function, and show that the *don't care* conditions for the gate optimization represent the bound on this perturbation. We present algorithms to compute such *don't care* conditions in both the combinational and synchronous case. We comment on the implementation of the algorithms and on the experimental results.**

I. Introduction

LOGIC SYNTHESIS techniques are used in most digital designs. Multiple-level logic optimization techniques [1]–[4] are based on circuit transformations that preserve the circuit behavior and improve its quality. Different flavors of circuit transformations have been proposed. Optimization algorithms based on Boolean transformations, such as those used in [1], [4], and [3], have shown to be effective in reducing the circuit area and delay as well as improving its testability properties.

Don't care conditions play a central role in the specification and optimization of logic circuits, as they represent the degrees of freedom for transforming a network into a functionally equivalent one. In the case of combinational circuits, the computation of don't care sets has been the subject of extensive investigation. In particular, it was shown that observability don't care conditions could be computed by *flattening* the network [2] or by using the *chain rule* [5]. Since both methods may require a prohibitive amount of computation, algorithms for computing observability don't care subsets were proposed [3], [6]–[8]. We presented first formulas for evaluating exact observability don't care sets [9], [10] and we describe here an improved algorithm for their computation.

Don't care conditions in sequential synchronous circuits were considered mostly in connection with state minimization algorithms for finite state machines [14]–[16]. These techniques use *behavioral* descriptions of circuits, in terms of state diagrams or equivalent representations [14], [15], [18]–[20]. The major drawback of this model is its remoteness from the network implementation, that makes it difficult to evaluate some figures of merit, such as final area or performance.

We present in this paper a unified analysis of don't care conditions in combinational and sequential synchronous multiple-level circuits, and we present novel algorithms for the computation of don't care conditions in both cases. Don't care conditions are computed using a structural circuit model, and therefore they differ (cf., for example, [21], [22]) from those that were previously proposed for synchronous circuits.

Multiple-level circuits are modeled by interconnections of combinational logic gates and registers, and attention is restricted to synchronous single-clock operation. Such structural representations are common in digital design. Indeed, most designers describe (and some high-level synthesis systems automatically generate) synchronous circuit implementations in terms of a schematic or equivalent netlist. It is important to provide iterative improvement techniques of circuits having such a representation. Structural don't care specifications can be automatically extracted from a structural description without resorting to a state transition diagram (or equivalent) representation. Such don't care conditions are linked to the Boolean transformations that allow us to directly replace subnetworks with better ones in terms of area or speed.

Throughout the paper we use *perturbation analysis* as a means to investigate local optimization problems. In particular, we model the replacement of a gate in a synchronous Boolean network by a *perturbation* of the gate functionality, and we show that don't care sets represent an upper bound on the permissible perturbation. This model is first presented for single- and multiple-gate optimization in combinational networks, and then extended to the synchronous case.

The rest of this paper is organized as follows. In the next section we present the model of synchronous Boolean network. In Section III we present the perturbation theory for combinational circuits, and algorithms for the computation of observability don't care conditions of each gate in the network. In Section IV we generalize the theory to synchronous circuits. We introduce the concept of *retiming-invariant* external don't care specifications, and show that, similarly to the combinational case, under this assumption the don't care conditions on a gate can still be

Manuscript received March 27, 1991; revised July 7, 1992. This work was supported by NSF and DEC under a PYI award and by NSF/ARPA under Contract MIP-8719546. This paper was recommended by Associate Editor R.K. Brayton.

The authors are with the Center for Integrated Systems, Stanford University, Stanford, CA 94305.

IEEE Log Number 9205509.

represented as the sum of a controllability and observability component; algorithms for the computation of the observability don't care sets presented for the combinational case are then extended to the synchronous case. Eventually, in Section V, we comment on the implementation of the algorithms and on experimental results.

Appendix I contains a review of previous work in the combinational case, as well as an analysis of approximation techniques for the derivation of local don't care sets in both the combinational and synchronous case. The proofs of theorems appearing in the text are grouped in Appendix II.

II. Basic Concepts and Definitions

We consider, in this paper, both combinational and synchronous multiple-level logic circuits. We assume that these circuits consist of an interconnection of multiple-input single-output combinational logic gates and, in the case of synchronous circuits, of synchronous delay-type registers, modeled by unit-delay elements.[1]

Fanout points are modeled by single-input multiple-output *copy* gates, as shown in Fig. 1. Single clocking is assumed for the sake of simplicity. No direct combinational feedback is allowed.

We model these circuits by **synchronous, multiple-level Boolean networks.** A synchronous Boolean network is described by a **weighted multigraph** $G = (V, E, W)$. The elements of the vertex set V correspond to logic gates (including copy gates). There is an edge e from a vertex μ to a vertex ν with weight w if an output of the gate in μ is connected to an input of the gate in ν through a cascade of (possibly zero) w registers. A Boolean variable is associated to each edge, and it is denoted by a string (e.g., x, sample); the corresponding edge is indicated by a subscript (e.g., e_x, e_{sample}). A variable x is said to be a fanout (fanin) variable of a vertex ν if e_x is an edge whose tail (head) end-point is ν. The head and tail vertices of an edge e are denoted by head(e) and tail(e), respectively. The sets of fanout and fanin edges of a vertex ν are indicated by FO(ν) and FI(ν), respectively. The weight of an edge e_x is indicated by w_x. To represent primary input and output variables, a *source* and a *sink* vertex are added. An edge e_x joins the source to a vertex ν if x is a primary input variable feeding the gate corresponding to ν. Similarly, an edge e_z joins a vertex ν to the sink vertex if z is a primary output variable computed by the gate corresponding to ν. Note in particular that FI(source) = FO(sink) = ϕ. The number of primary inputs and outputs are $n_i = |\text{FO(source)}|$ and $n_o = |\text{FI(sink)}|$, respectively. The number of variables is $n_e = |E|$.

Vertices represent the computational elements of the network. In particular, each fanout variable y of a vertex ν is associated to a local function of the fanin variables of ν. For copy gates, the function reduces to identity.

[1]In so doing, we implicitly assume that the values taken by registers at power-up are deleted by some suitable reset sequence.

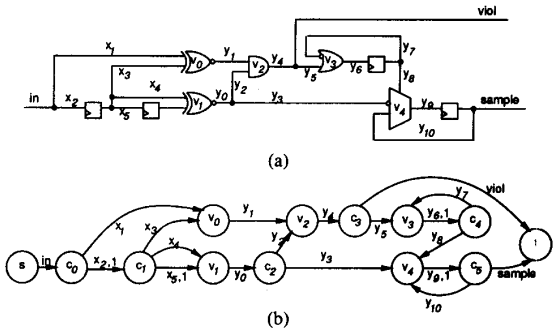


Fig. 1. (a) Synchronous Boolean network and (b) its weighted graph representation. Vertices labeled c_i correspond to *copy* gates. Vertices labeled s and t denote source and sink, respectively.

In general, a synchronous Boolean network may have cyclic dependencies, i.e., its corresponding graph may be cyclic. We assume that each cycle has strictly positive weight, to model the restriction of breaking loops of combinational logic with at least one register.

A network is called **definite** when its graph is acyclic. Combinational circuits are in particular modeled by acyclic graphs, with identically zero (and therefore not shown) edge weights. Note that this model differs slightly from the Boolean network model described in [2] because variables are associated to edges rather than to vertices, and because of the explicit accounting of copy gates.

Every synchronous network can be decomposed into a definite synchronous subnetwork, containing in particular all logic elements, and a feedback network, realizing a pure interconnection.

Example 1: A synchronous Boolean network and its graph are shown in Fig. 1. It is a portion of the phase decoder of the Digital Audio Input-Output Chip [25], that processes an input data stream with a biphase encoding (as generated by a CD player) and converts it to a stream of decoded Boolean samples or detects biphase encoding violations. The network can be represented as consisting of a definite portion plus two feedback interconnections, corresponding to variables y_7 and y_{10}. □

III. Combinational Networks

We denote by $\mathfrak{B}$ the Boolean set $\{0, 1\}$. An n-dimensional **Boolean vector** $\boldsymbol{x} = (x_1, \cdots, x_n)$ is an element of $\mathfrak{B}^n$. Underlining is used throughout the paper to denote vector quantities. A **Boolean function** is a mapping $\boldsymbol{H}: \mathfrak{B}^n \to \mathfrak{B}^m$.

The **Shannon cofactors** (or **residues**) of a function $\boldsymbol{H}$ with respect to a variable x_i are the functions $\boldsymbol{H}|_{x_i} = \boldsymbol{H}(x_1, \cdots, x_i = 1, \cdots, x_n)$ and $\boldsymbol{H}|_{x_i'} = \boldsymbol{H}(x_1, \cdots, x_i = 0, \cdots, x_n)$. We indicate by $\exists_{x_i} H$ and $\forall_{x_i} H$ the functions $H|_{x_i} + H|_{x_i'}$ and $H|_{x_i} H|_{x_i'}$, respectively. The $\exists$ and $\forall$ operators are known in the synthesis community as **smoothing** and **consensus**, respectively [7]. They are also referred to as **existential** and **universal quantifiers** [32].

In the combinational case, a Boolean network realizes in particular a function $\boldsymbol{F}: \mathfrak{B}^{n_i} \to \mathfrak{B}^{n_o}$, associating an n_o-

dimensional Boolean vector to each point of $\mathfrak{B}^{n_i}$. Due to its embedding in a larger system, or to degrees of freedom in the specification, external requirements do not impose in general a unique function to be realized, but rather a *relation* between input and output variables. The optimization of combinational networks under this type of specifications is the subject of ongoing research [17]. Traditionally, however, two specific degrees of freedom have been considered: *controllability don't cares*, defined as primary input combinations that never occur, and *observability don't cares*, defined as primary input assignments for which a network output is regarded as irrelevant [2].

Controllability don't care sets are here described by a function CDC^{ext} of the primary input variables, while observability don't cares are indicated by a n_o-dimensional vector $\boldsymbol{ODC}^{\text{ext}}$. The ith component ($i = 1, \cdots, n_o$) of $\boldsymbol{ODC}^{\text{ext}}$ is a function of the primary input variables and it represents the set of conditions for which the ith output is not observed [2]. For ease of treatment, we define an auxiliary n_o-dimensional vector $\boldsymbol{1} = (1, 1, \cdots, 1)$, and represent CDC^{ext} by a vector $\boldsymbol{CDC}^{\text{ext}} = CDC^{\text{ext}}\boldsymbol{1}$. The complete set of external don't care specifications is denoted by $\boldsymbol{DC}^{\text{ext}} = \boldsymbol{CDC}^{\text{ext}} + \boldsymbol{ODC}^{\text{ext}}$.

A function $f^y: \mathfrak{B}^{n_i} \to \mathfrak{B}$ can similarly be associated to each network variable y. This could be modeled by adding, for each fanout variable y of a vertex v, a contribution $y \oplus f^y$ to an **internal satisfiability** *don't care* **set**:

$$SDC^{\text{int}} = \sum_{e_y \in E} y \oplus f^y. \tag{1}$$

SDC^{int} is a function $\mathfrak{B}^{|E|} \to \mathfrak{B}$ describing the unfeasible assignments for the network variables, and is used during logic optimization to express each function f^y in terms of other internal variables [7].

Due to its embedding in the logic network, each internal function f^y is usually specified incompletely as well. Logic synthesis and optimization algorithms take advantage of such degrees of freedom and refine an original network iteratively by replacing f^y by others that improve some property of the network, typically area occupation or timing performance. This is accomplished by first determining which functions g^y can replace f^y, and then using logic/timing optimization tools to select an optimal realization.

Modifying f^y into a different function g^y has the effect of injecting a *perturbation* in the network, corresponding to those input combinations that result in $f^y \neq g^y$. We thus consider *perturbation analysis* as a tool for determining a description of the space of functions g^y that can replace a given function f^y. To this purpose, we introduce the following definitions.

Definition 3.1: Given a variable y of a network N, we call **perturbed network** N^y the network obtained by replacing the function f^y with the function $f^y \oplus \delta$, where δ is an added input, termed **perturbation.**

The functionality of a perturbed network N^y is described by a function $\boldsymbol{F}^y$, which in particular depends on δ: $\boldsymbol{F}^y = \boldsymbol{F}^y(\delta)$. In particular $\boldsymbol{F}^y(0) = \boldsymbol{F}$ and the functionality of any network N' obtained by replacing f^y with an arbitrary function g^y is described by $\boldsymbol{F}^y(f^y \oplus g^y)$.

Definition 3.2: Consider a network N', obtained from N by replacing a function f^y with g^y, N' is functionally equivalent to N if the vector equality

$$\boldsymbol{F}^y(g^y \oplus f^y) = \boldsymbol{F}^y(0) \tag{2}$$

is satisfied for all observable components of $\boldsymbol{F}$, for all possible primary input assignments.

Let[2]

$$\boldsymbol{E}(\delta) \stackrel{\text{def}}{=} \boldsymbol{F}^y(\delta) \,\overline{\oplus}\, \boldsymbol{F}^y(0). \tag{3}$$

From Definition 3.2 it thus follows that g^y can replace f^y if and only if for every $x \in \mathfrak{B}^{n_i}$, $\delta = f^y(x) \oplus g^y(x)$,

$$\boldsymbol{E}(\delta) + \boldsymbol{DC}^{\text{ext}} = \boldsymbol{1}. \tag{4}$$

The points of $\mathfrak{B}^{n_i}$ such that δ can be set to 1 represent the entries of f^y that can be changed (i.e., complemented) without affecting the network behavior, and thus represent the degrees of freedom for the optimization of f^y. The set of such points is termed don't care set for the function f^y. In the next section we present algorithms for its extraction.

3.1. Observability don't care Conditions

Definition 3.3: Given a perturbed network N^y, the quantity

$$\boldsymbol{ODC}^y = \boldsymbol{F}^y(1) \,\overline{\oplus}\, \boldsymbol{F}^y(0) \tag{5}$$

is termed **observability don't care vector** for the variable y. Its ith component ($i = 1, \cdots, n_o$) describes the set of network assignments for which y does not affect the ith network output.

Note that the complement of the observability don't care vector is the ordinary *Boolean difference* $\partial \boldsymbol{F}/\partial y$ of $\boldsymbol{F}$ with respect to y. We recall here that for a function $H(x)$ of a Boolean variable x [5]:

$$H(x) \,\overline{\oplus}\, H(0) = x' + H(1) \,\overline{\oplus}\, H(0). \tag{6}$$

Consequently, we have $\boldsymbol{E}(\delta) = \delta'\boldsymbol{1} + \boldsymbol{ODC}^y$ and (4) can be rewritten as

$$\delta'\boldsymbol{1} + \boldsymbol{DC}^{\text{ext}} + \boldsymbol{ODC}^y = \boldsymbol{1} \tag{7}$$

which holds if and only if

$$\delta\boldsymbol{1} \subseteq \boldsymbol{DC}^{\text{ext}} + \boldsymbol{ODC}^y \stackrel{\text{def}}{=} \boldsymbol{DC}^y. \tag{8}$$

Equation (8) shows that the functionality of N is unchanged as long as the points for which $f^y \neq g^y$ are contained in all the components of $\boldsymbol{DC}^y$, i.e., as long as the induced perturbation is *bounded* by all the components of the vector $\boldsymbol{DC}^y$.

The exact and approximate computation of $\boldsymbol{ODC}^y$ has long been recognized as a problem on its own. In the following section, we consider formulae for its computation.

[2]The function E and the perturbation δ are related to the variable y, and should be denoted by E^y and δ^y, respectively. The superscript y is dropped for the sake of conciseness.

1) Computation of observability don't care sets by local rules: In a combinational network, it is in principle possible to compute $\boldsymbol{ODC}^y$ for any variable y by flattening the network N^y and applying the Boolean difference operator to the resulting expression. This operation, however, may be very time and memory consuming and often impossible. To this regard, note also that in a logic synthesis environment it would have to be repeated every time a vertex in the transitive fanout of y is modified.

For these reasons, the computation of the ODC sets has been recently the object of intense investigation [6], [7], [3], [8]. We present here algorithms for their exact calculations, referring the reader to Appendix I for the analysis of approximate solutions and a review of previous work.

Consider the output variable y of a logic gate, that is not a copy gate. If $\boldsymbol{ODC}^y$ is known, then it is easy to obtain an expression of $\boldsymbol{ODC}^x$ for any fanin variable x of the gate by [5]

$$\boldsymbol{ODC}^x = \boldsymbol{ODC}^y + \left(\frac{\partial f^y}{\partial x}\right)' \boldsymbol{1}; \tag{9}$$

i.e., by adding $(\partial f^y/\partial x)'$ to all the components of the vector $\boldsymbol{ODC}^y$. The quantity $(\partial f^y/\partial x)'$ represents the *local* portion of $\boldsymbol{ODC}^y$.

If the network has a tree structure (i.e., each vertex has a single fanout variable), then (9) is sufficient to obtain all the *ODC* vectors by a backward traversal of the network. The major problem of computing *ODC* vectors occurs in presence of *copy* gates with reconvergent output variables. In this case the observability don't cares of the input variable do not necessarily coincide with those of any of its outputs, and network traversal had to be abandoned. Alternatively, the *chain rule* could be used, but its complexity has so far limited its application (the *chain rule* links the observability vector of the input variable y of a *copy* gate to those of its outputs y_1, y_2 by the equation [5]:

$$\boldsymbol{ODC}^y = \boldsymbol{ODC}^{y_1} \,\overline{\oplus}\, \boldsymbol{ODC}^{y_2} \,\overline{\oplus}\, \left(\frac{\partial^2 \boldsymbol{F}}{\partial_{y_1}\partial_{y_2}}\right)' \tag{10}$$

involving in particular higher order derivatives).

We present here an alternative technique. For the sake of simplicity, we describe first the case in which a copy gate with input y has only two fanout variables, y_1 and y_2, and later generalize the result. Let y denote the fanin variable of y, and consider the function $\boldsymbol{F}^{y_1y_2}(\delta_1, \delta_2)$ obtained by perturbing the edges corresponding to the two fanout variables with two variables δ_1, δ_2. The observability don't care vector of y is

$$\boldsymbol{ODC}^y = \boldsymbol{F}^{y_1y_2}(1, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1y_2}(0, 0). \tag{11}$$

By manipulating (11), $\boldsymbol{ODC}^y$ can be rewritten as

$$\boldsymbol{ODC}^y = (\boldsymbol{F}^{y_1,y_2}(1, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1,y_2}(0, 1)) \,\overline{\oplus}\, (\boldsymbol{F}^{y_1y_2}(0, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1y_2}(0, 0)) \tag{12}$$

where the term $\boldsymbol{F}^{y_1y_2}(0, 1)$ has been "added and subtracted" in (11). From the definition of observability don't care sets, the first term in parentheses is $\boldsymbol{ODC}^{y_1}(\delta_2 = 1)$, while the second parentheses describe $\boldsymbol{ODC}^{y_2}(\delta_1 = 0)$. Note that, for $\delta_2 = 1$ we have $y_2 = y'$ while, since $\delta_1 = 0$, for all feasible variable assignments it must be $y_1 = y$. Therefore, we finally obtain

$$\boldsymbol{ODC}^y = \boldsymbol{ODC}^{y_1}\big|_{y_2 = y'} \,\overline{\oplus}\, \boldsymbol{ODC}^{y_2}. \tag{13}$$

A different expression can be obtained by adding twice $\boldsymbol{F}^{y_1y_2}(1, 0)$ in (11):

$$\boldsymbol{ODC}^y = \boldsymbol{ODC}^{y_1} \,\overline{\oplus}\, \boldsymbol{ODC}^{y_2}\big|_{y_1 = y'}. \tag{14}$$

It follows in particular that the right-hand sides of (13) and (14) must be identical. This identity will be used extensively in Appendix I, when considering approximations to the *ODC* vectors.

Example 2: We contrast (13) with the *chain rule* and the network flattening approach using the simple multiplexer circuit of Fig. 2. We are interested in computing the ODC vector (of one component only) for the variable y. With the network flattening approach, the expressions associated to vertices A, B, and C are substituted in that of vertex O to obtain $F = x_1y' + x_2y$ (the substitution of vertex C can actually be saved if variables are associated to vertices rather than to edges. Two substitutions are thus essential to the method). We then obtain

$$\begin{aligned} ODC^y &= (x_1y' + x_2y)\big|_y \,\overline{\oplus}\, (x_1y' + x_2y)\big|_{y'} \\ &= x_1 \,\overline{\oplus}\, x_4. \end{aligned}$$

With the chain rule, the ODC expressions of the variables y_1 and y_2 are first computed:

$$ODC^{y_1} = x_1' + z_2;\ ODC^{y_2} = x_2' + z_1$$

Then,

$$\begin{aligned} ODC^y &= (x_1' + z_2) \,\overline{\oplus}\, (x_2' + z_1) \,\overline{\oplus}\, \left(\frac{\partial^2 F}{\partial y_1 \partial y_2}\right)' \\ &= (x_1 + z_2) \,\overline{\oplus}\, (x_2 + z_1) \,\overline{\oplus}\, (x_2' + x_1'). \end{aligned}$$

Notice that in this case we need only substitute the expression associated to vertex A into ODC^{y_2}, to obtain an expression of ODC^{y_2} in terms of y_1. We have, however, to perform three EXOR operations rather than one (the two appearing in the *chain rule* formula plus one needed for the higher order derivative).

Using Eq. (13), ODC^{y_1} and ODC^{y_2} are combined by substituting the expression of vertex A in ODC^{y_2}, and replacing y_1 and y':

$$ODC^y = (x_1' + z_2) \,\overline{\oplus}\, (x_2' + x_1y).$$

Thus only one substitution is performed, and only one EXOR is actually computed, comparing favorably against both the flattening approach and the *chain rule*. □

The extension of (13) to the general case of f fanout variables is provided by the following theorem.

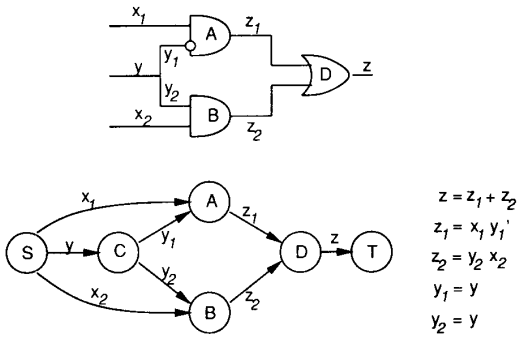


Fig. 2. Multiplexer for Example 2 and corresponding graph.

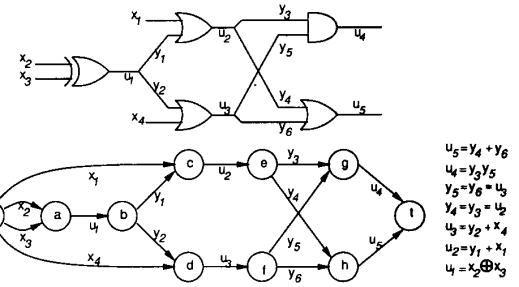


Fig. 3. Network for Example 3 and corresponding graph.

Theorem 3.1: Let y and $y_1, \cdots, y_f; f > 1$ denote the input and output variables of a copy gate; then:

$$ODC^{y} = \overline{\oplus}_{i=1}^{f} ODC^{y_i} \mid_{y_{i+1} = \cdots = y_f = y'}. \quad (15)$$

2) An Algorithm for the Computation of ODC Sets

Given Eq. (15), it is now possible to visit the Boolean network backwards from the primary outputs to its inputs and to determine the ODC sets of each vertex also in presence of reconvergent fanout.

The following algorithm performs the computation of the ODC sets. It uses the subset S of the vertices whose ODC set is known. Initially S is the set of edges associated to the primary output variables.

```
OBSERVABILITY(G);
T = {sink};
S := FI(sink);
while (T ≠ V) {
    select v ∈ {V − T} such that FO(v) ⊆ S;
    if (vertex_type(v) == gate) {
    /* y denotes the fanout variable of v, y_i the fanin variables. Equation (9) is used */
      foreach y_i ∈ FI(v) {
      ODC^{y_i} = ODC^y + (∂f^y/∂y_i)'
    }
    else {
    /* y denotes the fanin variable, y_i the fanout variables. ODC^y is computed by (15) */
      ODC^y = ⊕'_{i=1}^{n} ODC^{y_i} |_{y_{i+1} = ··· = y_f = y'}
    }
    S := S ∪ FI(v);
    T := T ∪ {v};
}
```

Theorem 3.2: Algorithm OBSERVABILITY computes the exact observability don't care set for each variable of the network.

Proof: The algorithm clearly terminates in a finite number of steps, as it traverses an acyclic graph in topological order from the primary outputs. At each **while** iteration, (T, S) is the graph of a connected subnetwork of G for which the observability don't care vectors are known. This is certainly true initially. It remains to be shown that this holds at the end of each iteration. With the *select* operation, we determine a vertex whose fanout variables already have their ODC vector computed. It is thus possible to compute the ODC vectors for all the input variables of v. The rule to be applied is either (9) or (15) depending on whether the vertex represents a logic or a *copy* gate. After the computation, $(T \cup \{v\}, S \cup \mathrm{FI}(v))$ is still a connected subnetwork, by construction, and all the ODC vectors of its variables are known. □

Example 3: We illustrate here the algorithm on the circuit shown in Fig. 3. The components of each observability vector describe the observability at outputs u_4 and u_5, respectively.

Initially, $T = \{t\}$, $S = \{u_4, u_5\}$, and

$$ODC^{u_4} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}; \quad ODC^{u_5} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

First the vertices g and h are considered, and the ODC vectors of variables y_3, y_4, y_5, y_6 are determined. By applying (13) it is then possible to determine

$$ODC^{u_2} = ODC^{y_3} \,\overline{\oplus}\, ODC^{y_4} \mid_{y_3 = u_2'} = \begin{pmatrix} y_5' \\ 1 \end{pmatrix} \overline{\oplus} \begin{pmatrix} 1 \\ y_6 \end{pmatrix} = \begin{pmatrix} y_5' \\ y_6 \end{pmatrix}$$

$$ODC^{u_3} = ODC^{y_5} \,\overline{\oplus}\, ODC^{y_6} \mid_{y_5 = u_3'} = \begin{pmatrix} y_3' \\ 1 \end{pmatrix} \overline{\oplus} \begin{pmatrix} 1 \\ y_4 \end{pmatrix} = \begin{pmatrix} y_3' \\ y_4 \end{pmatrix}.$$

From these the ODC vectors of x_1, y_1, y_2, x_4 are computed. In particular,

$$\boldsymbol{ODC}^{y_1} = \begin{pmatrix} y_5' + x_1 \\ y_6 + x_1 \end{pmatrix}; \boldsymbol{ODC}^{y_2} = \begin{pmatrix} y_3' + x_4 \\ y_4 + x_4 \end{pmatrix}$$

so that, using (13),

$$\begin{aligned} \boldsymbol{ODC}^{u_1} &= \boldsymbol{ODC}^{y_1} \overline{\oplus}\, \boldsymbol{ODC}^{y_2}\big|_{y_1 = \overline{u}_1} \\ &= \begin{pmatrix} y_5' + x_1 \\ y_6 + x_1 \end{pmatrix} \overline{\oplus} \begin{pmatrix} x_1' u_1 + x_4 \\ x_1 + u_1' + x_4 \end{pmatrix} \\ &= \begin{pmatrix} x_1 x_4 \\ x_1 + x_4 \end{pmatrix}. \end{aligned}$$

The product of all the components of $\boldsymbol{ODC}^{u_1}$ (in this case, x_1x_4) gives the conditions for which the gate in a is not observable at *any* output. □

In the end, the algorithm has computed in particular the ODC sets of the primary inputs. When two combinational networks are cascaded, the ODC sets at the input of the driven network represent the external ODC sets for the driving network.

It is important to remark that the *ODC* sets computed by algorithm OBSERVABILITY may be large in size, regardless of the representation chosen. Equation (13) allows us to construct and evaluate approximations to such sets. We refer the interested reader to Appendix I for formulae that can be used in conjunction with algorithm OBSERVABILITY to approximate the *ODC* sets.

3.2. Multiple-Gate Optimization and Compatible don't cares

A natural extension of don't care-based optimization methods consists of considering optimization problems involving more gates at a time [17]. The purpose of this section is to analyze the problem from a perturbation analysis viewpoint, and introduce some results that will be essential later when dealing with sequential circuits.

The simultaneous optimization of n gates, with output variables $y_1, \cdots, y_n$, can be modeled by introducing multiple perturbations $\delta_1, \cdots, \delta_n$, one for each variable. Let $\boldsymbol{F}^{y_1, \cdots, y_n}$ denote the function realized by the perturbed network. By introducing the function

$$\begin{aligned} \boldsymbol{E}(\delta_1, \cdots, \delta_n) = \boldsymbol{F}^{y_1, \cdots, y_n}(\delta_1, \cdots, \delta_n) \\ \overline{\oplus}\, \boldsymbol{F}^{y_1, \cdots, y_n}(0, \cdots, 0) \end{aligned} \quad (16)$$

the new network is functionally equivalent to the original one if and only if

$$\boldsymbol{E} + \boldsymbol{DC}^{\text{ext}} = \boldsymbol{1}. \quad (17)$$

Theorem (3.3) shows that, unlike (4), (17) cannot be reduced to a set of upper bounds on the individual perturbations $\delta_1, \cdots, \delta_n$.

Theorem 3.3: Perturbations $\delta_1, \cdots, \delta_n$ satisfy (17) if and only if

$$\begin{aligned} &(\boldsymbol{DC}^{\text{ext}})'(\boldsymbol{E}')\big|_{\delta_1'} \subseteq \delta_1 \boldsymbol{1} \subseteq \boldsymbol{E}\big|_{\delta_1} + \boldsymbol{DC}^{\text{ext}} \\ &(\boldsymbol{DC}^{\text{ext}})'(\forall_{\delta_1}\boldsymbol{E}')\big|_{\delta_2'} \subseteq \delta_2 \boldsymbol{1} \subseteq \exists_{\delta_1}\boldsymbol{E}\big|_{\delta_2} + \boldsymbol{DC}^{\text{ext}} \\ &\quad\vdots \\ &(\boldsymbol{DC}^{\text{ext}})'(\forall_{\delta_1, \cdots, \delta_{i-1}}\boldsymbol{E}')\big|_{\delta_i'} \\ &\quad \subseteq \delta_i \boldsymbol{1} \subseteq \exists_{\delta_1, \cdots, \delta_{i-1}}\boldsymbol{E}\big|_{\delta_i} + \boldsymbol{DC}^{\text{ext}}, \qquad i = 1, \cdots, n. \end{aligned} \quad (18)$$

Theorem (3.3) has two important consequences, that illustrate the added difficulties of dealing with multiple perturbations with respect to single perturbations: first, each individual perturbation δ_i may have a **lower** bound to satisfy, which depends on $\delta_{i+1}, \cdots, \delta_n$; second, also the upper bound on δ_i is a function of $\delta_j, j \neq i$. To remove the first such difficulty, we determine **sufficient** conditions in terms of upper bounds only, for (17) to hold.

Theorem 3.4: If perturbations $\delta_1, \cdots, \delta_n$ satisfy

$$\begin{aligned} \delta_i \boldsymbol{1} \subseteq \boldsymbol{DC}^{\text{ext}} + \boldsymbol{ODC}^{y_i}\big|_{\delta_1', \cdots, \delta_{i-1}'}(\delta_{i+1}, \cdots, \delta_n) \\ + \sum_{k \neq i} \delta_k (\boldsymbol{ODC}^{y_k}\big|_{\delta_1', \cdots, \delta_{k-1}'})' \end{aligned} \quad (19)$$

then (17) holds. Moreover, the bounds expressed by eq. (19) are **maximal**, in the sense that no minterm of the primary inputs or perturbations can be added to their right-hand side and still represent a set of solutions of (17).

Theorem (3.4) expresses sufficient conditions for (17) in terms of upper bounds on perturbations only. Such bounds still depend, however, on other perturbation signals. Elimination of such dependencies (for example, by means of the consensus operation) would result in compatible don't care sets [3], [8] for $y_1, \cdots, y_n$:

Definition 3.4: Don't care vectors $\boldsymbol{ODC}^{y_i}$; $i = 1, \cdots, n$ are termed **compatible** if:

1) none of them depends on any of $\delta_1, \cdots, \delta_n$;
2)

$$\delta_i \boldsymbol{1} \subseteq \boldsymbol{DC}^{\text{ext}} + \boldsymbol{ODC}^{y_i}, \qquad i = 1, \cdots, n \quad (20)$$

represent sufficient conditions for (17).

Compatible don't cares are said to be **maximal** if no cube in terms of the primary inputs can be added to them and (20) represent sufficient conditions for (17).

The use of compatible don't care sets allows us to optimize all gates independently with conventional minimization algorithms. Network traversal methods for extracting compatible don't care conditions are reviewed in Appendix I.

3.3. Controllability don't care Conditions

In the previous sections we analyzed the problems of determining don't care specifications (in terms of *don't*

care sets) for the internal functions of combinational and synchronous circuits.

As a network N interacts with other networks, it limits their controllability and observability. In a logic synthesis environment, it is therefore important to determine the controllability and observability don't cares induced by a circuit on its environment.

The observability don't care conditions of a network driving N are the observability don't care sets associated to the primary inputs of N. These are computed by OBSERVABILITY upon termination. We thus divert our attention on computing the controllability don't care conditions for a network driven by N. These represent the combinations never asserted at the outputs of N:

Definition 3.5: The **output controllability** don't care **set** of a network N, realizing a function $\boldsymbol{F}$, is the set $CDC^{\text{out}} \subseteq \mathcal{B}^{n_o}$ of output combinations that cannot be asserted by the network, when presented with inputs in $\mathcal{B}^{n_i} \cap (CDC^{\text{ext}})'$.

The set $(CDC^{\text{out}})'$ is also called the **image** of $(CDC^{\text{ext}})'$ with respect to the function $\boldsymbol{F}$. Image computation techniques have, therefore, been considered for the computation of CDC^{out}.

The approach considered here consists of computing the controllability don't care sets associated to successive cutsets of the network, moving the cutset from the primary inputs to the primary outputs, as outlined in the following pseudocode.

```
CONTROLLABILITY(G, CDC_ext);
T = {source};
S = FO(source);
CDC^out = CDC^ext;
while T ≠ V) {
      select v ∈ V − T such that FI(v) ⊆ S;
      foreach fanout variable y of v {
            CDC^out = CDC^out + f_y ⊕ y
      }
      foreach fanin variable x of v {
            CDC^out = ∀_x(CDC^out);
      }
      T := T ∪ {v};
      S = S ∪ FO(v);
}
return CDC_s;
```

In CONTROLLABILITY, (T, S) form a connected subgraph of the network graph G, which is iteratively incremented by adding a suitably chosen vertex v. The following example illustrates the details of the algorithm on the circuit of Fig. 3.

Example 4: Consider the problem of computing the impossible assignments for the variables u_3 and u_4, given the external don't care set:

$$CDC^{\text{ext}} = x_1' x_4'.$$

The first step of CONTROLLABILITY consists in selecting the vertex a and computing the controllability don't care set of the variables x_1, x_4, u_1, obviously still $x_1' x_4'$.

The fanout point b is selected next. The new cutset is formed by the variables x_1, y_1, y_2, x_4, and its associated controllability don't care set is given by $x_1' x_4' + y_1 y_2' + y_2 y_1'$. By selecting the vertex c, adding the term $u_2 \oplus (y_1 + x_1)$, and computing the *consensus* w.r.t. the variables x_1 and y_1, we obtain for the cutset $\{u_2, y_2, x_4\}$,

$$CDC^{\text{out}} = \forall_{x_1, y_1}(x_1' x_4' + y_1 y_2' + y_2 y_1' + u_2 y_1' x_1' + u_2' y_1 + u_2' x_1) = u_2' x_4' + u_2' y_2. \quad (21)$$

Vertex d is then selected. For the cutset $\{u_2, u_3\}$ we thus have

$$CDC^{\text{out}} = \forall_{y_2 x_4}(u_2' x_4' + u_2' y_2 + u_3 y_2' x_4' + u_3' y_2 + u_3' x_4) = u_2' u_3'.$$

The algorithm then completes by processing vertices e, f, g, and h, thus reaching the primary outputs with the output don't care set $CDC^{\text{out}} = u_5'$. □

IV. Synchronous Networks

4.1. Terminology

In order to describe the terminal behavior of a synchronous circuit, we need to take into account the evolution of the network variables over time, and reason in terms of **sequences** of values such variables take. We consider a discretization of time in integer time points $\mathbb{Z} = \{-\infty, \cdots, -1, 0, 1, \cdots, \infty\}$, and assume the observed operation of the network to begin conventionally at time $n = 0$, after some initializing (or reset) sequence is applied. By this choice, meaningful inputs (namely, the reset inputs) are initially applied at some time point ≤ 0; consequently, in order to capture correctly the behavior of the network variables over time, we consider sequences of values over a time interval $[-r, +\infty)$ for some suitably chosen $r \geq 0$.

Given an arbitrary finite set $\mathcal{S}$, a sequence s of elements of $\mathcal{S}$ is a mapping $s: [-r, +\infty) \to \mathcal{S}$. The set of all possible sequences of elements in $\mathcal{S}$ is conventionally denoted by $\mathcal{S}^{\omega}$ [11]. For example, the set of Boolean sequences is denoted by $\mathcal{B}^{\omega}$. The set of possible input and output sequences of a n_i-input, n_o-output synchronous circuit are denoted by $(\mathcal{B}^{n_i})^{\omega}$ and $(\mathcal{B}^{n_o})^{\omega}$, respectively. The sequences of values taken by the $|E|$ internal variables of a synchronous circuit are similarly elements of $(\mathcal{B}^{|E|})^{\omega}$.

Several techniques are available to describe sets of sequences [12], [11]. They do not easily lend themselves, however, for use in a compact structural description of sequential hardware. For this reason we consider here describing such sets by means of **synchronous Boolean**

expressions. For a variable x, a **synchronous literal** (or, shortly, a **literal**) x_n (x'_n) represents the set of sequences whose value of variable x is 1 (0) *at time point n*. A **synchronous Boolean expression** S is a finite expression in terms of synchronous literals, sums and products being defined in the usual way. In particular, a **cube** of synchronous literals is their product. Hereafter, we do not distinguish between synchronous expressions and the sets they describe. As a synchronous Boolean expression contains literals with time labels in some finite time interval $[t_1, t_2]$, they can only describe particular sets of sequences, namely those whose values in $[t_1, t_2]$ satisfy the expression.

We define a **retiming** operation on sets of sequences and expressions as follows. The retiming s_k of a sequence $s \in \mathbb{S}^\omega$ is

$$s_k = \{z \in \mathbb{S}^\omega | z_{n+k} = s_n \forall n \in [r, +\infty)\}. \quad (22)$$

The retiming of a set of sequences by k is the union of the retiming of each of its elements.

The **retiming** $(x_n)_k$ of a literal x_n by k is the literal x_{n+k}. The retiming S_k of a synchronous expression S by k is the retiming of all its literals by k. It can be verified that the definition of retiming on expressions is consistent with the corresponding one for sequences: the set described by S_k is the retiming by k of the set described by S.

The operations defined on ordinary Boolean expressions and sets, such as cofactoring and quantification, carry over naturally to synchronous Boolean expressions and the corresponding sets. For notational convenience, we introduce the operations $\exists_n$ and $\forall_n$, denoting smoothing and consensus of a synchronous Boolean expression with respect to all the literals with time label n.

4.2. *Don't care Conditions in Synchronous Networks*

The terminal behavior of a synchronous circuit is entirely described by the correspondence it establishes between input and output sequences, i.e., between elements of $(\mathbb{B}^{n_i})^\omega$ and $(\mathbb{B}^{n_o})^\omega$. Due to its embedding in a larger system, (or possibly to degrees of freedom in the specification), external requirements do not impose a unique correspondence between input and output sequences, but rather a *relation* between them. Intuitively, this is due to: a) not all sequences are usually possible at the inputs of a synchronous circuit, and b) for a given input sequence, usually more responses are allowed. The task of synthesis and optimization would then consist of determining an optimal synchronous circuit whose terminal behavior satisfies that relation.

Such a broad description of don't care conditions is out of the scope of the present paper. We rather focus our attention on don't care conditions that can be described by *sets*, namely: an **external controllability** don't care set ($CDC^{\text{ext}} \subseteq (\mathbb{B}^{n_i})^\omega$), representing input *sequences* that cannot occur at the network inputs, and an **external observability** don't care set, representing conditions for which some of the output values are not observed. Similar to the combinational case, external observability don't care conditions are represented by vectors. More specifically, the ith component of the **observability don't care** vector $\boldsymbol{ODC}_n^{\text{ext}}$ represents the conditions under which the ith primary output of the network is not observed at time n. We also assume that each component of $\boldsymbol{ODC}_n^{\text{ext}}$ is a subset of $(\mathbb{B}^{n_i})^\omega$. We denote

$$\boldsymbol{DC}_n^{\text{ext}} = CDC^{\text{ext}}\boldsymbol{1} + \boldsymbol{ODC}_n^{\text{ext}}. \quad (23)$$

The following example illustrates some contexts in which external don't care conditions arise. Methods for extracting and using such information are the object of the present chapter.

Example 5: Consider the circuit of Fig. 4, initialized by $(b_{-4}, b_{-3}, b_{-2}) = (1, 0, 1)$. We thus take $r = 4$, and consider sequences in the interval $[-4, +\infty)$.

The limited controllability of the inputs of N_2 is reflected by the set of its impossible input sequences. For example, $u_n v'_{n+1}$ is an impossible input sequence for N_2: for u_n to be equal to 1 it must be $a_n = b_n = 1$; but $b_n = 1$ implies $v_{n+1} = 1$. Hence, for N_2,

$$u_n v'_{n+1} \subseteq CDC^{\text{ext}} \qquad \forall_n \in [-4, +\infty). \quad (24)$$

As a consequence of the initializing sequence, output v cannot assume the value 0 at time -3, -1:

$$v'_{-3} + v'_{-1} \subseteq CDC^{\text{ext}}.$$

Finally, it could be verified that v cannot take value 0 twice consecutively; consequently,

$$v'_n v'_{n+1} \subseteq CDC^{\text{ext}} \qquad \forall n \geq -4.$$

The interconnection of the two networks limits the observability of the primary outputs of N_1. In particular, the output of N_2 can be expressed in terms of u and v as

$$F_n = u'_{n-1} + v_{n-1} + u_n \oplus v_n.$$

The value of v_n can be observed at the output of N_2 only at time n or at time $n + 1$. In particular, v_n is observable at time n if $y_{n-1} = 0$ and $u_{n-1} = 1$. The observability don't care of v at time n can thus be described by the function:

$$ODC^v_{n,n} = u'_{n-1} + y_{n-1} = u'_{n-1} + v_{n-1}$$

while the observability at time $n + 1$ is described by

$$ODC^v_{n,n+1} = (y_{n+1}u_n)' = u_{n+1} \oplus v_{n+1} + u'_n.$$

Sufficient conditions for never observing v_n at the primary output of N_2 are described by

$$ODC^v_n = ODC^v_{n,n} ODC^v_{n,n+1}$$

in particular containing the cube $u_{n-1}u'_n$. Since $u_n = a_n b_n$, then $a_{n-1}b_{n-1}(a'_n + b'_n)$ belongs to the component of the external *ODC* set of N_1 associated to output v. □

In the combinational case, the conditions for replacing a single-output subnetwork are completely stated by (8). Namely, f^y can be replaced by g^y when the induced perturbation δ is *bounded* by a *set* DC^y, which can be ex-

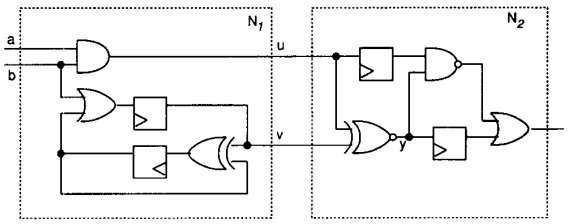


Fig. 4. Interconnected synchronous Boolean networks.

tracted from the network structure and functionality, and forms the don't care set for the variable y. Once DC^y is computed, any Boolean optimizer can be used to optimize f^y using DC^y as don't care set.

The synchronous case is more complex. In particular, the expressions g^y that can replace f^y cannot always be described by a bound on the induced perturbation. This is shown by the following simple example.

Example 6: Consider the circuit in Fig. 5. The output z_n is expressed in terms of the primary inputs by $z_n = x'_n \oplus x'_{n-1}$. It can easily be recognized that the inverter can be replaced by a simple connection, i.e., $f^y = x'$ can be replaced by $g^y = x$. In this case, $f^y \oplus g^y = 1$. Had (8) been applicable, then we should conclude that $DC^y = 1$, i.e., that the inverter can also be replaced by a constant 0 or 1, which is clearly false. □

The full analysis of such internal degrees of freedom is the subject of ongoing research [23]. In the framework of this paper, we consider don't care conditions that can be expressed by a set, as it is then possible to take advantage of efficient existing optimizers, such as ESPRESSO [26].

In the next section, we consider the case of definite networks, and present a derivation of don't care conditions associated to a single-output subnetwork in that case. In Section 4.4 we leverage upon the results for definite networks and obtain don't care conditions for networks with feedback.

4.3. *Definite Networks*

The acyclic structure of definite networks makes it possible to express their primary outputs at any time n by a synchronous Boolean expression $\boldsymbol{F}_n$ in terms of their primary inputs. Similarly, all internal variables y can be associated an expression f^y in terms of primary inputs. This could be modeled by adding, for each time point n, a contribution $y_n \oplus f^y_n$ to an **internal satisfiability** *don't care* **set** $SDC^{\text{int}} \subseteq (\mathfrak{B}^{|E|})^\omega$. Again, SDC^{int} is regarded as a tool for mapping degrees of freedom associated to a gate in terms of other internal signals, and will not intervene directly in the subsequent analysis.

The effect of modifying a local function on the network functionality is expressed by means of a perturbed network. Let $\boldsymbol{F}^y_n$ denote the function realized by the network perturbed in correspondence of an edge e_y, and P the longest path from the vertex head(e_y) to the primary output. The output $\boldsymbol{F}^y_n$ depends in general on $\delta_n, \cdots, \delta_{n-P}$,

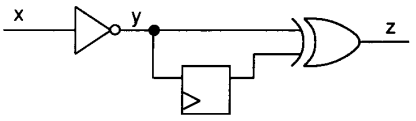


Fig. 5. Circuit example.

that is, $\boldsymbol{F}^y_n = \boldsymbol{F}^y_n(\delta_n, \cdots, \delta_{n-P})$. In particular, $\boldsymbol{F}_n = \boldsymbol{F}^y_n(0, \cdots, 0)$, and the functionality of any other network obtained by replacing f^y with some other function g^y is described by $\boldsymbol{F}^y_n(f^y_n \oplus g^y_n, \cdots, f^y_{n-P} \oplus g^y_{n-P})$, i.e., by simply replacing the perturbation δ by its expression.

Similarly to the combinational case, we introduce the function:

$$\boldsymbol{E}_n(\delta_n, \cdots, \delta_{n-P}) \stackrel{\text{def}}{=} \boldsymbol{F}^y_n(\delta_n, \cdots, \delta_{n-P}) \,\overline{\oplus}\, \boldsymbol{F}^y_n(0, \cdots, 0) \qquad (25)$$

and define g^y to be equivalent to f^y ($g^y \approx f^y$) if and only if

$$\boldsymbol{E}_n + \boldsymbol{DC}^{\text{ext}}_n = 1 \qquad \forall n \geq 0. \qquad (26)$$

Equation (26) represents the constraint equation on the perturbation δ, for the terminal behavior of the network to result unchanged. Example 6 showed that it is not possible to explicit (26) in an **equivalent** form of type $(f^y \oplus g^y)\boldsymbol{1} \subseteq \boldsymbol{DC}^y$. It is however important for logic optimization to determine a set $\boldsymbol{DC}^y$ such that a constraint of type $f^y \oplus g^y \subseteq \boldsymbol{DC}^y$ is **sufficient** to guarantee the validity of (26).

In the remainder of this section we present a formal derivation of don't care sets for logic optimization starting from the formulation of functional equivalence provided by (26), and algorithms for their extraction. In particular, first we derive bounds of type $\delta_n \subseteq \boldsymbol{DC}^y_n \; \forall n \geq 0$, where $\boldsymbol{DC}^y_n$ is some suitable set of sequences that depends on the time-point n considered. This infinite set of constraints is then transformed into a single bound $(f^y \oplus g^y)\boldsymbol{1} \subseteq \boldsymbol{DC}^y$. Eventually, algorithms for the extraction of the set $\boldsymbol{DC}^y$ are presented.

1) *Observability don't care Conditions*

The methods presented here rely upon the definition of observability don't care conditions for synchronous networks. In particular, the observability don't care set of a value y_m at time n represents the conditions for which a perturbation of y_m is not observed at the primary outputs at that time point:

Definition 4.1: We call **observability don't care function** of y the function:

$$\boldsymbol{ODC}^y_{m,n}(\delta_n, \cdots, \delta_{m+1}, \delta_{m-1}, \cdots, \delta_{n-P}) = \boldsymbol{F}^y_n(\delta_n, \cdots, \delta_{m+1}, 1, \delta_{m-1}, \cdots, \delta_{n-P}) \,\overline{\oplus}\, \boldsymbol{F}^y_n(\delta_n, \cdots, \delta_{m+1}, 0, \delta_{m-1}, \cdots, \delta_{n-P}). \qquad (27)$$

Note that, in the present case, $\boldsymbol{ODC}^y_{m,n}$ may depend on perturbations of $y_{m'}$, $m' \neq m$. Clearly, $\boldsymbol{ODC}^y_{m+k,n+k} = (\boldsymbol{ODC}^y_{m,n})_k$, so that in practice the $P + 1$ expressions $\boldsymbol{ODC}^y_{0,n}$; $n = 0, \cdots, P$ are sufficient to completely specify the observability don't care function of y.

Example 7: Expressions of the observability don't care conditions for the XNOR gate in Fig. 4 are

$$ODC^y_{0,0} = y_{-1} + u'_{-1}; \qquad ODC^y_{0,1} = y'_1 + u'_0.$$

These expressions contain the internal variable y. Its substitution with its expression $f^y \oplus \delta$ explicits the dependencies from δ:

$$ODC^y_{0,0} = v_{-1} \oplus \delta_{-1} + u'_{-1};$$

$$ODC^y_{0,1} = u_1 \oplus v_1 \oplus \delta_1 + u'_0. \qquad \square$$

The simplest solution to (26) occurs when all paths from a gate to the primary output have the same length P. In this case, $\boldsymbol{E}_n = \boldsymbol{E}_n(\delta_{n-P})$ only, and Eq. (26) reduces essentially to the combinational case:

$$\delta_{n-P}\boldsymbol{1} \subseteq \boldsymbol{ODC}^y_{n-P,n} + \boldsymbol{DC}^{\text{ext}}_n \qquad \forall n \geq 0. \quad (28)$$

A network is said to be a *pipeline* if for each vertex all paths to a primary output have the same length. Eq. (28) shows that don't care sets fully describe the degrees of freedom for the optimization of these networks. Note also that for pipelines the computation of don't care sets is a straightforward extension to that for the combinational case, represented by $P = 0$ in (28). These don't care conditions are essentially those considered by retiming/resynthesis techniques [13].

When considering arbitrary definite networks, however, a vertex has in general multiple paths of different length to the primary output, so that the function $\boldsymbol{E}_n$ has multiple dependencies upon $\delta_n, \cdots, \delta_{n-P}$, and the associated don't care conditions expressed by (26) are correspondingly more complex. It is possible, however, to take advantage of the results of Section 3.2 on multiple perturbations. In particular,

Theorem 4.1: If a perturbation δ satisfies

$$\delta_{n-k}\boldsymbol{1} \subseteq \boldsymbol{DC}^{\text{ext}}_n + \boldsymbol{ODC}^y_{n-k,n}|_{\delta'_{n-k+1},\cdots,\delta'_n}, \qquad k = 0, \cdots, P \quad (29)$$

for every $n \geq 0$, then (26) holds.

Theorem 4.1 is a direct application of Theorems 3.3 and 3.4 to (26); its proof is, therefore, omitted.

For a given time-point $n \geq 0$, Theorem 4.1 provides $P + 1$ bounds on δ_n (there are actually only $n + P + 1$ bounds for $n = -P, \cdots, -1$). The actual bound on each δ_n is their intersection:

$$\delta_n\boldsymbol{1} \subseteq \boldsymbol{CDC}^{\text{ext}} + \prod_{k=0}^{\min(P+1,n+P+1)} \cdot (\boldsymbol{ODC}^{\text{ext}}_{n+k} + \boldsymbol{ODC}^y_{n,n+k}|_{\delta'_{n+1},\cdots,\delta'_{n+k}}) \forall n \geq -P. \quad (30)$$

Equation (30) has an intuitive interpretation: a gate output can be altered at time n (δ_n can be set to 1) corresponding to those input sequences that either do not occur (represented by CDC^{ext}), or that make the perturbation unobservable at any time in the future. These conditions are represented by the product of the observability don't care sets at different time points in (30). It is, therefore, convenient to define a function

$$\boldsymbol{ODC}^y_n(\delta_{n-1}, \cdots, \delta_{n-P}) = \prod_{k=0}^{\min(P+1,n+P+1)} (\boldsymbol{ODC}^{\text{ext}}_{n+k} + \boldsymbol{ODC}^y_{n,n+k}|_{\delta'_{n+1},\cdots,\delta'_{n+k}}). \quad (31)$$

representing such observability don't cares.

Example 8: We model the optimization of the XNOR gate of Fig. 4 by the search of feasible perturbations δ. The constraints expressed by (29) are given by

$$\delta_n \subseteq DC^{\text{ext}}_n + ODC^y_{n,n}$$

and

$$\delta_{n-1} \subseteq DC^{\text{ext}}_n + ODC^y_{n-1,n}|_{\delta'_n}$$

where $ODC^y_{0,0}$ and $ODC^y_{0,1}$ are taken from Example 7, and $DC^{\text{ext}}_n = CDC^{\text{ext}}$ is taken from Example 5. By substituting such expressions in (30),

$$\delta_n \subseteq CDC^{\text{ext}} + ODC^y_{n,n} ODC^y_{n,n+1}|_{\delta'_{n+1}} = CDC^{\text{ext}} + (v_{n-1} \oplus \delta_{n-1} + u'_{n-1}) \cdot (u_{n+1} \oplus v_{n+1} + u'_n)$$

and, in particular,

$$ODC^y_n = (v_{n-1} \oplus \delta_{n-1} + u'_{n-1}) \cdot (u_{n+1} \oplus v_{n+1} + u'_n). \qquad \square.$$

2) Retiming-Invariant don't care Conditions

Equation (30) expresses an infinite number of bounds on the signal δ, one for each time point. In order to perform logic optimization we need to reexpress such bounds in a finite form. To this regard, we need the following definition.

Definition 4.2: We call **retiming-invariant** component $\boldsymbol{DC}^{\text{ri}}$ of an external don't care specification $\boldsymbol{DC}^{\text{ext}}_n$ the set of sequences $s \in \boldsymbol{DC}^{\text{ext}}_0$ such that $s_k \subseteq \boldsymbol{DC}^{\text{ext}}_k \; \forall k \geq 0$.

Example 9: In Example 5, CDC^{ext} for N_2 was shown to contain the sets described by the cubes $v'_{-1}v'_0$, $v'_0v'_1$, $\cdots$, as well as by the literals v'_{-3} and v'_{-1}. All sequences in $v'_{-1}v'_0$ are in DC^{ri}, as the sets described by $v'_{n-1}v'_n$ are in $CDC^{\text{ext}} \subseteq DC^{\text{ext}}_n \; \forall n \geq 0$. Similarly, $u_{-1}v'_0 \subseteq DC^{\text{ri}}$. The set v'_{-1}, instead, is not, as its retiming v'_{n-1} is not always contained in DC^{ext}_n (otherwise, v would be constantly 1, which is clearly false). Similarly, the set $a_{-1}b_{-1}(a'_0 + b'_0)$ belongs to the retiming-invariant component of the external don't care set of N_1, as $a_{n-1}b_{n-1}(a'_n + b'_n) \subseteq ODC^{\text{ext}}_n \subseteq DC^{\text{ext}}_n \; \forall n \geq 0$.

The following theorem translates the infinite set of constraints represented by (26) into a finite one. As a side result, it shows that the retiming-invariant component of an external don't care specification is also the only **useful** external don't care specification for the logic optimization of a definite network.

Theorem 4.2: Two functions f^y and g^y are equivalent if and only if for $\delta_n = f_n^y \oplus g_n^y$,

$$\boldsymbol{E}_0 + \boldsymbol{DC}^{\text{ri}} = \boldsymbol{1}. \tag{32}$$

It is in practice convenient to assume that CDC^{ext} can be expressed by means of a retiming-invariant component CDC^{ri} and an *initialization* part, CDC^{init}:

$$CDC^{\text{ext}} = CDC^{\text{init}} + \sum_{k=0}^{\infty} CDC_n^{\text{ri}} \tag{33}$$

both expressed by synchronous Boolean expressions with literals in the reference interval $[-r, 0]$. CDC^{init} describes in particular the set of impossible input values over the interval $[-r, 0]$, while CDC_n^{ri} describes the set of impossible values for the input variables in some interval $[n - r, n]$, for arbitrary $n \geq 0$. We also assume external observability don't care specifications to retiming-invariant.

As a consequence of these assumptions and of Theorem 4.2, (30) can be rewritten simply as

$$\delta_0 \boldsymbol{1} = (f^y \oplus g^y)\boldsymbol{1} \subseteq \boldsymbol{CDC}^{\text{ri}} + \boldsymbol{ODC}_0^y. \tag{34}$$

The set provided by (34) is not yet directly suitable for logic optimization, because of the dependency of $\boldsymbol{ODC}_0^y$ on $\delta_{-1}, \cdots, \delta_{-P}$. Given the similarity of this problem with that of extracting compatible don't care sets in the combinational case, we present, in Appendix I, algorithms for the extraction of subsets of $\boldsymbol{ODC}_0^y$ free from such dependencies.

Alternatively, from the equation $y_n = f_n^y \oplus \delta_n$, it is possible to write $\delta_n = y_n \oplus f_n^y$, and obtain an expression of $\boldsymbol{ODC}_0^y$ in terms of $y_{-1}, \cdots, y_{-P}$. The use of this type of information would allow the insertion of feedback during the optimization process, as shown in Example 10.

Example 10: From Example 9, $CDC^{\text{ri}} = (u_{-1} + v'_{-1})v'_0$ is an appropriate choice for N_2. The observability don't care of y is given in Example 9. Its expression in terms of y_{-1} is given by

$$ODC_0^y = (u'_{-1} + y_{-1})(u_1 \oplus v_1 + u'_0).$$

The map of such don't care conditions for the XNOR gate is shown in Fig. 6(a).

Elimination of δ_{-1} from the expression of ODC_0^y by *consensus* results instead in

$$ODC_0^y = u'_{-1}(u_1 \oplus v_1 + u'_0).$$

The map of this don't care set is in Fig. 6(b), and it shows that the XNOR gate can be optimized into the simpler AND gate uv, as depicted in Fig. 7. Expressions in terms of u_{-1}, v_{-1}, y_{-1} are also possible, corresponding to situations in which registers and/or feedback paths are added. □

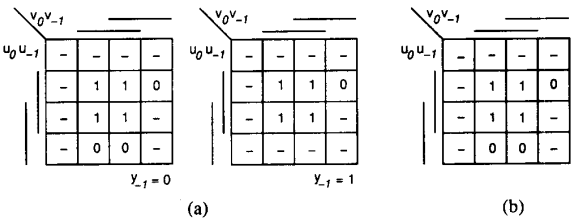


Fig. 6. (a) *Don't care* conditions for the XNOR gate of the network in Fig. 4, (b) *don't care* conditions neglecting the terms containing y_{-1}.

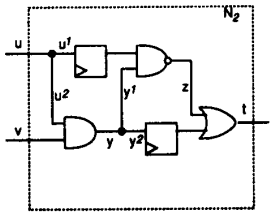


Fig. 7. An optimized version of network N_2 of Fig. 4.

3) An Algorithm for the Observability don't cares in Definite Networks

For multiple-output networks, observability don't care conditions are again expressed by vectors $\boldsymbol{ODC}_{0,k}^y$; $k = 0, \cdots P$. It is possible to extend the operations used in the extraction of the observability don't care sets to the synchronous case as follows. If $\boldsymbol{ODC}_{0,k}^y$ is known for the fanout variable y of a vertex v, then it is easy to obtain the corresponding expression for a fanin variable z of v from

$$\boldsymbol{ODC}_{0,k}^z = (\boldsymbol{ODC}_{0,k-w_z}^y)_{w_z} + \left(\frac{\partial f_{w_z}^y}{\partial z}\right)' \boldsymbol{1}. \tag{35}$$

Equation (35) is essentially identical to (9) of the combinational case, the only difference consisting in accounting for the delay w_z by appropriately retiming the *ODC* function of y. It is similarly possible to extend to the synchronous case (15) for *copy* gates. We present the extension for the case of two fanout variables, the general case being then straightforward.

Let y and v, z denote the input and output variables of a *copy* gate. The function $F^{v,z}$ describes the function of the perturbed network with the two perturbations δ^v, δ^z. If follows that

$$\boldsymbol{ODC}_{0,k}^y = \boldsymbol{F}_k^{v,z}(\delta_{w_y}^v = 1, \delta_{w_y}^z = 1) \,\overline{\oplus}\, \boldsymbol{F}_k^{v,z}(\delta_{w_y}^v = 0, \delta_{w_y}^z = 0). \tag{36}$$

By adding twice the term $\boldsymbol{F}_k^{v,z}(\delta_{w_y}^v = 0, \delta_{w_y}^z = 1)$, by manipulations similar to those leading to (13) we obtain

$$\boldsymbol{ODC}_{0,k}^y = \boldsymbol{ODC}_{w_y,k}^v \,\overline{\oplus}\, \boldsymbol{ODC}_{w_y,k}^z|_{v_{w_y} = y'_{w_y}} = \boldsymbol{ODC}_{0,k-w_y}^v \,\overline{\oplus}\, \boldsymbol{ODC}_{0,k-w_y}^z|_{v0 = y0'})_{w_y}. \tag{37}$$

The extension of the OBSERVABILITY algorithm is as follows. P denotes the longest path in the network graph G.

```
OBSERVABILITY(G);
T = {sink};
S = FI(sink);
while (T ≠ V) {
  select v ∈ V − T such that FO(v) ⊆ S;
  if (vertex_type(v) == gate) {
  foreach y ∈ FI(v) {
        for (j = 0, j < P, j ++)ODC^y_{0,j} = (∂f^μ_{wy}/∂y0)'1 + (ODC^v_{0,j−wy})_{wy}
  };
  else {
      y = FI(v)
      for (j = 0, j < P, j ++){
          ODC^y_{0,j} = (⊕ over k=1..|FO(v)| ODC^{yk}_{0,j}|_{yk+1,0 = ··· = y|FO(v)|,0 = y0'})
          }
  }
  S = S ∪ {FI(v)};
  T = T ∪ {v};
}
```

We illustrate here the algorithm on the network N_2 of Fig. 4.

Example 11: The longest path in the circuit is $P = 1$. The algorithm begins by computing $ODC^{z}_{0,0} = y^2_{-1}$, $ODC^{z}_{0,1} = 1$, $ODC^{y^2}_{0,0} = 1$ and $ODC^{y^2}_{0,1} = z_1$. The vertex corresponding to the NAND gate is then selected, to obtain $ODC^{y^1}_{0,0} = (u^1_{-1})' + y^2_{-1}$, $ODC^{y^1}_{0,1} = 1$, $ODC^{u^1}_{0,0} = 0$ and $ODC^{u^1}_{0,1} = (y^1_1)' + y^2_0$. It is now possible to compute, by (37),

$$ODC^{y}_{0,0} = ODC^{y^1}_{0,0} \,\overline{\oplus}\, ODC^{y^2}_{0,0}|_{y^1_0 = y'_0} = (u^1_{-1})' + y^2_{-1}$$

$$ODC^{y}_{0,1} = ODC^{y^1}_{0,1} \,\overline{\oplus}\, ODC^{y^2}_{0,1}|_{y^1_0 = y'_0} = y_1 + u^1_0.$$

In the end, the functions $ODC^{y^2}_{0,0} = v'_0 + (u^1_{-1})' + y^2_{-1}$ and $ODC^{u^2}_{0,1} = v'_1 + y_1 + u^1_0$ are determined, so that

$$ODC^{u}_{0,0} = ODC^{u^1}_{0,0} \,\overline{\oplus}\, ODC^{u^2}_{0,0}|_{u^1_0 = u'_0} = (v_0 u'_{-1} y^2_{-1})'$$

$$ODC^{u}_{0,1} = ODC^{u^1}_{0,1} \,\overline{\oplus}\, ODC^{u^2}_{0,1}|_{u^1_1 = u'_1} = (v_0 u'_{-1} y^2_{-1})'.$$

□.

Approximations to (36) entirely similar to those for the combinational case are also possible. Methods allowing us the computation of the final observability ODC^y, with no need for the individual observability don't care sets $ODC^y_{0,k}$, $k = 0, \cdots, P$ are in particular desirable; one such method is presented in Appendix I.

4.4. Cyclic Networks

A cyclic network N can be decomposed into a definite subnetwork N_d plus a feedback network consisting purely of interconnections (see, for example, Fig. 8(a)).

Intuitively, the simplest approach to the optimization of N, consisting of optimizing its definite portion N_d by the algorithms of Section 4.3, may neglect some additional don't care conditions, as the feedback interconnections of N_d are not directly controllable or observable. For example, some feedback sequences may be never asserted by the network and may therefore be considered as an external controllability don't care condition for N_d. Similarly, some values of the feedback input may be never observed at the primary outputs, thereby resulting in an external observability don't care condition of the feedback outputs of N_d. In this section we present techniques for extracting external observability and controllability don't care sets for the subnetwork N_d of an arbitrary cyclic network, that are able to capture the existence of feedback interconnections.

1) Observability don't care Conditions

For the sake of simplicity, we consider here single-output single-feedback networks, the extensions to the more general cases being relatively straightforward. The functionality of N can be described by means of the functions realized by N_d, namely the output function F and the feedback function S. The feedback signal s_n satisfies the relation

$$s_n = S(x_n, \cdots, x_{n-P}, s_{n-1}, \cdots, s_{n-P}) \quad (38)$$

where P denotes the maximum path length in N_d. The modification of a network function f^y into a different function g^y modifies in general the functionality also of the feedback network. It is thus convenient to describe the behavior of a modified network by means of two perturbation signals, at the logic gate and feedback input, respectively, as shown in Fig. 8(b). The perturbation signals are here denoted by δ and σ, respectively. The behavior of the perturbed network is described by the output function $F^y_n(\delta, \sigma)$ and by the feedback function $S^y_n(\delta, \sigma)$. (The notation maintains implicit their dependency on $\delta_n, \cdots, \delta_{n-P}$ and $\sigma_n, \cdots, \sigma_{n-P}$.)

In particular, $F_n = F^y_n(0, 0)$ and $S_n = S^y_n(0, 0)$. As σ_n represents the perturbation of the feedback input, it obeys

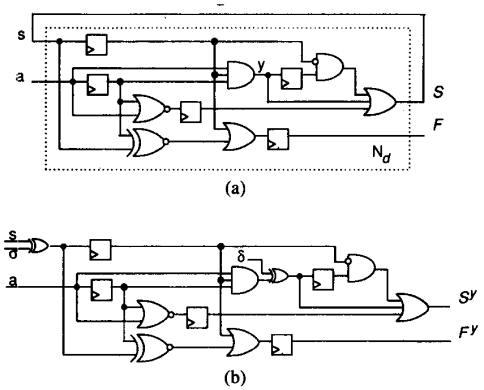


Fig. 8. (a) Decomposition of a cyclic network into a definite subnetwork and interconnection feedback. (b) Perturbed network for the optimization of the AND gate.

the recurrence equation:

$$\sigma_n = S_n^y(0, 0) \oplus S_n^y(\delta, \sigma) \stackrel{\text{def}}{=} (E_n^S)'. \tag{39}$$

The output equivalence of the original and perturbed networks is described by the function

$$E_n = F_n^y(0, 0) \,\overline{\oplus}\, F_n^y(\delta, \sigma). \tag{40}$$

The function g^y is equivalent to f^y if

$$E_n + DC_n^{\text{ext}} = 1 \;\forall n \geq 0 \tag{41}$$

where $\delta = f^y \oplus g^y$ and σ satisfies (39).

The external observability don't care set for S is constructed by determining a bound on the perturbation σ_n for (41) to hold: as σ_n models the perturbation of the feedback function at time n, its bound represents implicitly an observability don't care set for S_n. To this end, we introduce the following auxiliary equivalence functions:

$$\begin{aligned} E_{\delta,n}^F &\stackrel{\text{def}}{=} F_n^y(\delta, \sigma) \,\overline{\oplus}\, F_n^y(0, \sigma) \\ E_{\sigma,n}^F &\stackrel{\text{def}}{=} F_n^y(0, \sigma) \,\overline{\oplus}\, F_n^y(0, 0) \qquad (42) \\ E_{\delta,n}^S &\stackrel{\text{def}}{=} S_n^y(\delta, \sigma) \,\overline{\oplus}\, S_n^y(0, \sigma) \\ E_{\sigma,n}^S &\stackrel{\text{def}}{=} S_n^y(0, \sigma) \,\overline{\oplus}\, S_n^y(0, 0). \qquad (43) \end{aligned}$$

The following theorem allows us to split the problem of bounding σ and δ into two smaller subproblems, concerning σ and δ separately, and represented by (44) and (45), respectively.

Theorem 4.3: If the perturbations δ, σ, resulting from changing f^y into a different local function g^y, are such that

$$E_{\delta,n}^F + DC_n^{\text{ext}} = 1 \;\forall n \geq 0 \tag{44}$$

$$E_{\sigma,n}^F + DC_n^{\text{ext}} = 1 \;\forall n \geq 0 \tag{45}$$

then g^y can replace f^y.

We solve (45) by suitably bounding σ_n. This bound represents the extent to which the feedback input can be changed, without changing the network behavior. As the feedback input is generated by the feedback function, the bound represents an observability don't care set for that function. Equation (45) is thus ultimately reduced to an observability don't care set associated to the feedback output.

To determine this set, an iterative algorithm is used: the feedback output is initially assumed perfectly observable, and a bound on σ is determined by means of the algorithms in Section 4.3. This don't care set is then used as an external don't care condition on the feedback output to build iteratively larger bounds on the perturbation σ. More in detail, initially we set $ODC_{n,\langle 0\rangle}^s = 0 \;\forall n \geq 0$ (the symbol $\langle m\rangle$ denotes the iteration count), and then $ODC_{n,\langle m+1\rangle}^s$ is derived by solving

$$E_{\sigma,n}^F + CDC^{\text{ext}} + ODC_n^{\text{ext}} = 1 \tag{46}$$

$$E_{\sigma,n}^S + CDC^{\text{ext}} + ODC_{n,\langle m\rangle}^s = 1 \tag{47}$$

by means of the techniques presented in Section 4.3[3].

Theorem 4.4 proves the correctness of the approach.

Theorem 4.4: Suppose

$$\sigma_j \subseteq CDC^{\text{ext}} + ODC_{j,\langle m\rangle}^s \qquad j = -P, \cdots, -1 \tag{48}$$

$$E_{\sigma,n}^S + CDC^{\text{ext}} + ODC_{n,\langle m\rangle}^s = 1 \;\forall n \geq 0 \tag{49}$$

then,

$$\sigma_n \subseteq CDC^{\text{ext}} + ODC_{n,\langle m\rangle}^s \;\forall n \geq -P \tag{50}$$

and (45) holds.

From Theorem 4.4 (45) can now be replaced by (49), which represents a constraint on the perturbation δ only. Once a specification $ODC_{n,\langle m\rangle}^s$ is computed (by reaching convergence or by stopping the iteration at any arbitrary m), a don't care set ODC^y for all internal variables y can be found by solving, with the technique presented for definite networks, (44) and (49).

Equation (48) is most easily satisfied if $s_{-P}, \cdots, s_{-1}$ are determined by a reset sequence (as it is often the case), so that automatically $\sigma_j = 0$, $j = -P, \cdots, -1$. Interestingly, in this case (48) can be interpreted as providing degrees of freedom in the choice of the reset values $s_{-P}, \cdots, s_{-1}$.

We conclude by showing that each iteration improves on the size of the observability don't cares:

Theorem 4.5: For every $m \geq 0$ and $n \geq 0$, $ODC_{n,\langle m+1\rangle}^s \supseteq ODC_{n,\langle m\rangle}^s$.

Example 12: We derive here the observability don't care set ODC_n^s for the cyclic network of Fig. 8. The longest path is $P = 2$, and we assume $ODC_n^{\text{ext}} = 0 \;\forall n \geq 0$.

Initially, $ODC_{n,\langle 0\rangle}^s = 0$. Consequently,

$$ODC_{0,0}^s = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad ODC_{0,1}^s = \begin{pmatrix} 1 \\ a_0' + a_1 \oplus (a_{-1}(s_{-1} \oplus \sigma_{-1})) \end{pmatrix}; \quad ODC_{0,2}^s = \begin{pmatrix} s_1 \,\overline{\oplus}\, a_0 \\ s_1 + a_0' + a_1' \end{pmatrix}$$

[3]Assuming the retiming-invariance of external *don't care* conditions reduces the problem to that of computing ODC_n^s; $n = 0, \cdots, P$.

and, therefore,

$$ODC^s_{0,\langle 1\rangle} = (s_{-1} \oplus \sigma_{-1})\,(a'_0 + a_1 \overline{\oplus}\, a_{-1})\,(s_1 \overline{\oplus}\, a_0).$$

Note that $ODC^s_{0,\langle m\rangle}$ depends, in general, on future values of s. This dependency can be eliminated by taking advantage of (38). By working out the algebra, in this case we obtain

$$ODC^s_{0,\langle 1\rangle} = (s_{-1} \oplus \sigma_{-1})a_{-1}(a'_0 + a_1 s_{-1}).$$

At the second iteration, $ODC^s_{0,\langle 1\rangle}$ is used as external observability don't care set, and

$$ODC^s_{0,\langle 2\rangle} = (s_{-1} \oplus \sigma_{-1})a_{-1}(a'_0 + a_1(s_{-1} + a_{-2}s_{-2}))$$

is obtained. The third iteration could then be shown to result in $ODC^s_{0,\langle 3\rangle} = ODC^s_{0,\langle 2\rangle}$. Note that ODC^s depends on past values of s as well as of the perturbed signal $t = s \oplus \sigma$. As we allow the change of the feedback function during optimization, the original signal s cannot be considered available, and all internal functions must be expressed in terms of t. It follows in particular that the dependencies from s in ODC^s must be dropped, resulting in this case in $ODC^s = t_{-1}a_{-1}a'_0$. Fig. 9 shows the network of Fig. 8, optimized using this external don't care set. □.

2) Controllability don't care Conditions

In a cyclic network, the feedback interconnection can be regarded as a constraint imposed on the feedback input of the definite portion N_d. As a consequence of this constraint, even if no external don't care set is applied, in general only a subset of sequences is possible at the inputs of N_d; that is, the feedback interconnection can be regarded as inducing an external controllability don't care set at the inputs of N_d. We consider in this section the problem of determining a representation of the retiming-invariant portion of such a don't care set.

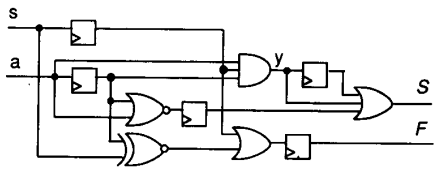


Fig. 9. The network of Fig. 8, optimized using its sequential observability *don't care* set.

We assume CDC^{ext} to be represented by an expression like (33), where CDC^{init} and CDC^{ri} have literals in the interval $[-r, 0]$. In order to describe the initialization of the feedback variables, we assume CDC^{init} to be in terms of input and feedback values, and CDC^{ri} to be in terms of inputs only.

The controllability don't care set $CDC^{\text{ext},d}$ of N_d is thus given by

$$CDC^{\text{ext},d} = CDC^{\text{init}} + \sum_{n=0}^{\infty} (CDC^{\text{ri}}_n + s_n \oplus S_n). \quad (51)$$

We derive a representation $CDC^{\text{ri},d}$, with literals in $[-r, 0]$, of a **retiming-invariant** controllability don't care set for the definite subnetwork N_d. Note that this is in particular a set of impossible sequences of inputs *and* feedback values. From Definition 4.2, $CDC^{\text{ri},d}$ must then satisfy

$$CDC^{\text{ri},d}_n \subseteq CDC^{\text{ext},d}\ \forall n \geq 0. \quad (52)$$

Since $CDC^{\text{ri},d}_n$ represents impossible values of the inputs in $[n - r, n]$, (52) can be simplified into

$$CDC^{\text{ri},d}_n \subseteq B_n \quad (53)$$

where the sets B_n, $n = 0, \cdots, \infty$ are defined by

$$B_0 = CDC^{\text{init}} + CDC^{\text{ri}} + s_0 \oplus S_0;$$

$$B_{n+1} = \forall_{n-r}(B_n + CDC^{\text{ri}}_{n+1} + s_{n+1} \oplus S_{n+1}). \quad (54)$$

One way of obtaining $CDC^{\text{ri},d}$ could then consist of starting from an initial estimate $CDC^{\text{ri},d}_{\langle 0\rangle}$, for example given by the intersection of a **finite number** of the bounds B_n, suitably retimed, and then of iteratively shrinking that estimate by intersecting it with further bounds, until convergence. The corresponding pseudocode is outline in the procedure *RET_INV*. Theorem 4.7 below shows that *RET_INV*, started with a suitable initial estimate, does indeed converge in a finite number of steps, in particular to a set satisfying (52).

```
RET_INV(CDC^init, CDC^ri, CDC^ri,d_⟨0⟩, r)
{
  TMP = CDC^ri,d_⟨0⟩;
  repeat {
      CDC^ri,d = TMP;
      TMP = CDC^ri,d ∀_{-(r+1)}(CDC^ri,d_{-1} + CDC^ri,d + s_0 ⊕ S_0);
  } until (TMP == CDC^ri,d);
  return (CDC^ri,d);
}
```

Theorem 4.6: Let $CDC^{\text{ri},d}_{\langle 0\rangle}$ be any synchronous Boolean expression, with literals in $[-r, 0]$, satisfying $CDC^{\text{ri},d}_n \subseteq B_n$; $n = 0, \cdots, r$; then *RET_INV* converges in at most $2^{(r+1)\times n_i}$ steps, and upon termination it returns a retiming-invariant portion of $CDC^{\text{ext},d}$, satisfying (52).

We illustrate the computations of *RET_INV* on the circuit of Fig. 8.

Example 13: A reset sequence $(s_{-2}, b_{-2}) = (0, 0)$; $(s_{-1}, b_{-1}) = (0, 0)$ is applied to the circuit of Fig. 8, so that

$$CDC^{\text{init}} = s_{-2} + s_{-1} + a_{-2} + a_{-1}.$$

No other external *don't care* conditions are assumed, i.e., $CDC^{\mathrm{ri}} = 0$, and a value $r = 2$ is chosen. The feedback function is $S = (a_{-1} + a_{-2})' + a_0 a_{-1} s_{-1} + a_{-1} a_{-2} s_{-2} s'_{-1}$. From (54), the bounds corresponding to $n = 0, 1, 2$ are

$$\begin{aligned} B_0 &= s_{-2} + s_{-1} + a_{-2} + a_{-1} + s_0 \oplus S_0 \\ &= s_{-2} + s_{-1} + s'_0 + a_{-2} + a_{-1} \\ B_1 &= \forall_{-2}(s_{-2} + s_{-1} + a_{-2} + a_{-1} + s'_0 + s_1 \oplus S_1) \\ &= s_{-1} + s'_0 + a_{-1} + s_1 \oplus (a'_0 + a_1) \\ B_2 &= \forall_{-1}(s_{-1} + s'_0 + a_{-1} + s_1 \oplus (a'_0 + a_1) \\ &\quad + s_2 \oplus S_2) = s'_0 + s_1 \oplus (a'_0 + a_1) \\ &\quad + s_2 \oplus (a'_1 a'_0 + a_2 s_1). \end{aligned}$$

The largest initial estimate $CDC^{\mathrm{ri},d}_{\langle 0\rangle}$ is represented by the intersection of the above upper bounds, suitably retimed:

$$\begin{aligned} CDC^{\mathrm{ri},d}_{\langle 0\rangle} &= (B_0)_0 (B_1)_{-1} (B_2)_{-2} \\ &= a_{-2}(s'_{-2} + s_{-1} a'_{-1}) + s'_{-1}(a_{-1} + s_{-2} a'_{-2}) \\ &\quad + s_0 \oplus (a'_{-1} a'_{-2} + s_{-1} a_0). \end{aligned}$$

The second estimate $CDC^{\mathrm{ri},d}_{\langle 1\rangle}$ is obtained by *RET_INV* is given by

$$\begin{aligned} CDC^{\mathrm{ri},d}_{\langle 1\rangle} &= CDC^{\mathrm{ri},d}_{\langle 0\rangle} \forall_{-3}((CDC^{\mathrm{ri},d}_{\langle 0\rangle})_{-1} + s_0 \oplus S_0) \\ &= s'_{-2} a_{-2} + (s_{-2} s'_{-1}) \oplus (a_{-2} a'_{-1}) \\ &\quad + s_0 \oplus (a'_{-1} a'_{-2} + s_{-1} a_0) \end{aligned}$$

Similarly,

$$\begin{aligned} CDC^{\mathrm{ri},d}_{\langle 2\rangle} &= CDC^{\mathrm{ri},d}_{\langle 1\rangle} \forall_{-3}((CDC^{\mathrm{ri},d}_{\langle 1\rangle})_{-1} + s_0 \oplus S_0) \\ &= s_{-2} s'_{-1}(a'_{-2} + a_{-1}) + s_{-1} b_{-2}(s_{-2} b_{-1})' \\ &\quad + s_0 \oplus (a'_{-1} a'_{-2} + s_{-1} a_0). \end{aligned}$$

It can then be verified that $CDC^{\mathrm{ri},d}_{\langle 3\rangle} = CDC^{\mathrm{ri},d}_{\langle 2\rangle}$, bringing *RET_INV* to convergence.

The network can be optimized by using $CDC^{\mathrm{ri},d}$ as an external, retiming-invariant, don't care specification for the optimization of N_d. An optimized network is in particular shown in Fig. 10. □.

The finite-state machine model of a synchronous network decomposes the network into a special definite portion (namely, such that $P = 1$), and a set of feedback lines, whose associated variables are termed state variables. This model can thus be regarded as a special case of the one considered here. *RET_INV* can similarly be regarded as a generalization of state traversal algorithms for finite-state machines, with the set of possible feedback values in an arbitrary interval $[-r, 0]$ (contained in $(CDC^{\mathrm{ri},d})'$) replacing the set of reachable states.

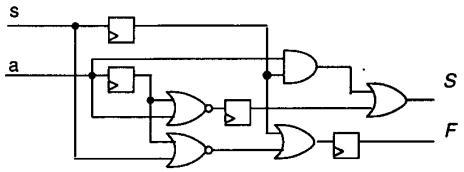


Fig. 10. The network of Fig. 8, optimized using its sequential controllability *don't care* set.

Two aspects that distinguish *RET_INV* from finite-state machine traversal are, however, worth remarking. First, *RET_INV* considers values on the feedback wires over predefined, but otherwise *arbitrarily long* intervals, rather than values at individual time-points. Second, it extracts a set of impossible sequences of input **and** feedback values, as opposed to focusing on impossible feedback combinations only.

We conclude by observing that the simultaneous use of controllability and observability don't care information, derived from the existence of feedback, may lead to optimization errors, as evidenced by Example (14) below: intuitively, using observability don't care conditions at the feedback output may change the set of feedback sequences, thereby possibly invalidating the controllability don't care set.

Example 14: The observability and controllability don't care sets associated to the feedback wire in the circuit of Fig. 8 were derived in Examples 12 and 13, respectively. As optimized realization, obtained by using both don't care specifications, is shown in Fig. 11, and is not equivalent to the original machine. For example, corresponding to the input sequence $(a_0, a_1, a_2, a_3, a_4) = (1, 0, 0, 0, 0)$, the network output is (1, 0, 1, 1, 1), while the correct output would be (1, 0, 1, 1, 0).

4.5. Controllability don't care Sets in Synchronous Networks

The controllability constraints induced by a network on its environment can be represented by the set of sequences that cannot be asserted by the network outputs:

Definition 4.3: The **output controllability** don't care **set** of a synchronous network N is the set $CDC^{\mathrm{out}} \subseteq (\mathfrak{B}^{n_o})^\omega$ of sequences over the interval $[0, +\infty)$ that cannot be asserted by the network, when driven by an input sequence in $(\mathfrak{B}^{n_i})^\omega \cap (CDC^{\mathrm{ext}})'$ spanning the interval $[r, +\infty)$.

In Section 4.5 external controllability don't care sets were expressed by the sum of a retiming-invariant component, expressed by CDC^{ri}, and an initializing portion CDC^{init}. Following that framework, we are interested in deriving (possibly incomplete) representations of CDC^{out} by means of an initializing portion $CDC^{\mathrm{init,out}}$ and a retiming-invariant component $CDC^{\mathrm{ri,out}}$. In particular, we consider expressions in the reference interval $[-r, 0]$. We restrict our attention to definite networks, as the case of cyclic networks is captured essentially by (52), reducing the feedback interconnection to an external controllability *don't care* set for the definite subnetwork N_d.

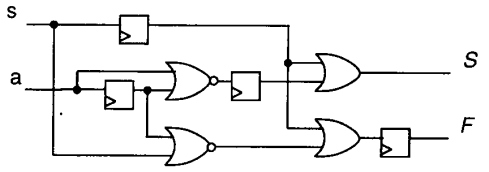


Fig. 11. Incorrectly optimized version of the network of Fig. 8.

The following algorithm is a direct extension of *CONTROLLABILITY* to the present case, and computes a set $CDC^{\text{out, ri}}$ of impossible output combinations in a reference interval $[-r, 0]$.

CONTROLLABILITY $(G, P, CDC^{\text{ri}}, r)$;
$T = \{\text{source}\}$
$S = \text{FO}(\text{source})$
$CDC^{\text{ri, out}} = \Sigma_{k=-P}^{0} CDC_k^{\text{ri}}$;
while $T \neq V$ {
  select $v \in V - T$ such that $\text{FI}(v) \subseteq S$;
  $T := T \cup \{v\}$;
  $S = S \cup \text{FO}(v)$;
  $P_v = \text{longest\_path}(v)$;
  foreach fanout variable y of v {
    for $(n = -(P_v + r);\ n \leq 0;\ n{+}{+})\, CDC^{\text{ri, out}} = CDC^{\text{ri, out}} + f_n^y \oplus y_n$;
  }
  foreach fanin variable x of v $CDC^{\text{ri, out}} = \forall_{x_i; i=-(r+P_v), \cdots, 0}\, (CDC^{\text{ri, out}})$;
  $T := T \cup \{v\}$;
  $S = S \cup \text{FO}(v)$;
}
return $(CDC^{\text{ri, out}})$;

Similar to the combinational case, CONTROLLABILITY traverses the network from the primary inputs, iteratively moving an initial cutset (defined by the connected subgraph (S, T)) to the primary outputs, and computing the controllability don't care set associated to each cutset. As the outputs in an arbitrary interval $[n - r, n]$ are functions of the primary inputs in the interval $[n - r - P, n]$, the impossible input combinations in this interval are determined first, represented by the initial estimate:

$$CDC^{\text{ri, out}} = \sum_{k=-(r+P)}^{0} CDC_k^{\text{ri}}. \tag{55}$$

For each selected vertex, first its longest distance P_v from the primary outputs is determined. As the outputs in the interval $[-r, 0]$ can depend on the vertex outputs only in the interval $[-(r + P_v), 0]$, for each fanout variable y of v only the contributions $y_n \oplus f_n^y$; $n \in [-(r + P_v), 0]$ are necessary.

The *consensus* rule is then applied, similarly to the combinational case. We illustrate CONTROLLABILITY on the circuit of Fig. 7.

Example 15: We take $r = 1$, and assume an input don't care set as determined in Example 5;

$$CDC^{\text{ri}} = (u_{-1} + v'_{-1})\, v'_0.$$

Initially, $T = \{\text{source}\}$, $S = \{e_u, e_v\}$. The longest path is $P = 1$, so that the initial estimate of $CDC^{\text{ri, out}}$ is

$$CDC^{\text{ri, out}} = \sum_{k=-1}^{0} CDC_k^{\text{ri}} = (u_{-2} + v'_{-2})\, v'_{-1} + (u_{-1} + v'_{-1})\, v'_0.$$

The fanout point of variable u is selected first. Its longest path from the primary outputs is *NEW* $P = 1$; consequently, first

$$CDC^{\text{ri, out}} = (u_{-2} + v'_{-2})\, v'_{-1} + (u_{-1} + v'_{-1})\, v'_0 + \sum_{k=-2}^{0} (u_k \oplus u_k^1 + u_k \oplus u_k^2)$$

is constructed, and then the dependency from u is eliminated by *consensus* in u_{-2}, u_{-1}, u_0. This results in

$$CDC^{\text{ri, out}} = (u_{-2}^1 + v'_{-2})\, v'_{-1} + (u_{-1}^1 + v'_{-1})\, v'_0 + \sum_{k=-2}^{0} u_k^1 \oplus u_k^2.$$

The AND gate is then selected. After adding the terms $y_k \oplus (v_k u_k^2)$; $k = -2, -1, 0$ and removing by *consensus* all dependencies from the fanin variables of v (namely, v and u^2),

$$CDC^{\text{ri, out}} = y_{-2}(u_{-2}^1)' + y_{-1}(u_{-1}^1)' + y_0(u_0^1)' + y'_0 u_0^1 u_{-1}^1 + y'_{-1} u_{-1}^1 u_{-2}^1.$$

Variables u and y are then similarly processed, to obtain

$$CDC^{\text{ri, out}} = z'_{-1}(y_{-1}^2)' + z'_0(y_0^2)' + y_{-2}^2 y_{-1}^2 z_{-1} + y_{-1}^2 z_0 y_0^2 + y_{-2}^2 (y_{-1}^2)' z'_0.$$

Eventually, the output gate is selected. Its longest path to the primary output is trivially 0, so that $t_k \oplus (z_k + y_k)$; $k = -1, 0$ are added. The elimination of the dependencies from y and z produces the final estimate

$$CDC^{\text{ri, out}} = t'_{-1} t'_0 \qquad (\square)$$

V. Implementation and Results

The algorithms presented in this paper have been implemented in C and tested on several synchronous bench-

TABLE I
BENCHMARK STATISTICS

Circuit	inputs	outputs	literals	registers	feedback	longest path
S208	11	2	166	8	8	1
S298	3	6	244	4	14	2
S244	9	11	269	15	15	1
S444	3	6	352	21	15	2
S526	3	6	445	21	21	1
S641	35	24	537	19	11	2
S820	18	19	757	5	5	3
S832	18	19	769	5	5	3
S1196	14	14	1009	18	0	3
S1238	14	14	1041	18	0	3
S1494	8	19	1393	6	6	1
S9234.1	36	39	7900	211	90	6

TABLE II
COMPUTATION OF THE OBSERVABILITY *don't care* SETS. THE APPROXIMATIONS PRESENTED IN APPENDIX I WERE USED FOR THE CIRCUITS LABELED (*)

Circuit	CPU time	BDD Nodes
S208	0.9	1280
S298	2.1	889
S344	2.1	2015
S444	4.1	4547
S526	5.6	3289
S641	10.1	2645
S820	19.0	11788
S832	18	6679
S1196(*)	23.4	305622
S1238(*)	17.9	398591
S1494(*)	19.6	12623
S9234.1(*)	151.4	456071

TABLE III
OPTIMIZATION RESULTS

	$r = P$			$r = P + 1$			$r = P + 2$		
Circuit	literals	registers	CPU time	literals	registers	CPU time	literals	registers	CPU time
S208	72	8	16	58	8	21	52	8	24
S298	109	12	27	102	12	44	99	12	51
S344	131	15	31	127	16	41	122	15	49
S444	144	19	29	131	18	41	127	17	51
S526	216	20	31	188	21	34	149	21	41
S641	209	14	53	187	15	64	150	14	88
S820	260	5	59	255	5	69	243	5	73
S832	261	5	65	245	5	98	245	5	188
S1196	531	15	194	531	15	194	531	15	194
S1238	609	15	238	609	15	238	609	15	238
S1492	582	6	91	569	6	191	565	6	236
S9234.1	747	176	785	462	174	987	398	177	1686

mark circuits. The benchmark statistics are shown in Table I.

All functions were represented internally by their BDD's [30], [31]. This choice renders in particular trivial the convergence test for the fixed-point algorithms, used for deriving external don't care conditions in cyclic networks.

The choice of the feedback interconnections to be cut affects the internal observability of all the internal nodes, and therefore in general the synthesis results. A minimum feedback set algorithm [34] was used for the choice of the feedback variables. The number of cuts and the longest paths (in terms of register counts) that resulted by this choice are reported in the columns labeled *feedbacks* and *longest path P*, respectively, in Table I. These parameters are obviously affected by choice of the feedback variables: for example, for the benchmark s344, a cut based on a depth-first network traversal [35] resulted in $P = 10$.

We present results on the computation of the observability don't care sets as well as on the overall logic optimization, in Tables II and III, respectively. Table II reports the peak size of the observability don't care sets in

terms of BDD node count as well as the global CPU time spent in their computation. The results of logic optimization, in terms of literals and CPU time, are shown in Table II. As no information about reset sequences or reset states is available, a reset sequence consisting of r consecutive zeros was selected for each circuit. The parameter r was then assigned the values P, $P + 1$, $P + 2$; feedback don't care conditions and logic optimization were performed for each of these values. Delay elements were assigned finite cost, equivalent to 4 literals. It was thus in principle possible to trade off combinational complexity by the addition of delay elements.

VI. Summary

In this paper we presented an analysis of don't care conditions in combinational and synchronous networks. The target of the analysis was the derivation of don't care conditions that could be used in particular for the optimization of synchronous circuits at the *structural* (i.e., netlist) level.

We thus assumed circuit specifications directly in terms of networks (e.g., interconnections of gates and delay elements), rather than representations with the explicit notion of state, such as state diagrams. We think that this model can support better optimization techniques by iterative improvement, where local combinational or sequential gates are iteratively replaced, because of the possibility of evaluating directly area and performance variation due to optimization.

In order to derive such don't care conditions, we have developed a *perturbation analysis* of the network, where the replacement of a local logic function is modeled by a perturbation of its functionality. Don't care conditions that can be used by current logic optimizers are only *upper bounds* on the possible perturbations.

In the case of combinational networks, it was shown [2] that a single upper bound on the perturbation of a single gate describes all the don't care conditions for that gate. For sequential networks, this is in general not the case. Describing don't care conditions entails being able to reason in terms of sequences of values taken by inputs and internal variables, and upper bounds on the perturbing sequences (as can be used by ordinary logic optimizers) represent some (but not all) the degrees of freedom in replacing the gates.

Algorithms for deriving and approximating such bounds have been derived and tested on several large sequential benchmark circuits, and used for their logic optimization. Unlike classic sequential logic optimization, where the number of registers was *a priori* minimized, and then the resulting combinational logic iteratively optimized, the present approach offers a greater degree of flexibility, by allowing the insertion of delay elements where justified by the benefit in the overall complexity of the logic. This property makes it possible to explore different tradeoffs in the distribution of the logic between combinational and sequential elements.

Appendix I

In this Appendix we analyze the problem of approximating observability don't care sets by formulas that can replace (15) in the OBSERVABILITY algorithm. Approximations to ODC sets are needed in order to reduce the size of their representation and possibly avoid the flattening operations discussed in Sect. (3.2). Some of such approximations essentially coincide with methods for extracting *compatible don't care* sets, which are considered in Section (8.2). In Sect. (8.3) approximation methods are extended to the case of synchronous circuits.

I.1 Previous Work

Several methods for approximating the observability don't care sets by a single backward traversal of the network have been proposed in the past [1], [3], [6], [7].

We use here (13) for examining the quality of these approximations. For simplicity, we restrict our attention to single-output networks (so that the ODC vectors have only one component) and to a *copy* vertex with only two output variables, as shown in Fig. 12.

We define the auxiliary quantities.

$$a_0 = ODC^{y_1}|_{y=0},\ a_1 = ODC^{y_1}|_{y=1},$$
$$b_0 = ODC^{y_2}|_{y=0},\ b_1 = ODC^{y_2}|_{y=0} \tag{56}$$

so that

$$ODC^{y_1} = a_0 y' + a_1 y;\ ODC^{y_2} = b_0 y' + b_1 y. \tag{57}$$

By substituting these expressions in (13) and (15) the following identity must hold:

$$ODC^{y} = (a_0 y' + a_1 y) \,\overline{\oplus}\, (b_0 y + b_1 y')$$
$$= (a_0 y + a_1 y') \,\overline{\oplus}\, (b_0 y' + b_1 y). \tag{58}$$

Any assignment a_0, a_1, b_0, b_1 violating identity (58) is, therefore, contained in the *satisfiability don't care* set of the network. Namely,

$$a_0 \oplus a_1 \oplus b_0 \oplus b_1 \subseteq SDC^{\text{int}}. \tag{59}$$

The Karnaugh map of ODC^y in terms of a_0, a_1, b_0, b_1, and y is shown in Fig. 13, where 1's denote the observability don't care, 0's the observability care, and $-$ denotes the impossible assignments given by (59), i.e., the satisfiability don't cares.

All previous approaches approximate ODC^y by some other function of a_0, a_1, b_0, b_1, and possibly, y. The quality of any such approximation can thus be measured by the number of covered 1's in the Karnaugh map.

We contrast various approaches referring to the graph of Fig. 12, and denote an approximation to ODC^y by $\widetilde{ODC}^y$. The most "conservative" simplification is taken in the program MIS-II, as reported in [1]. There, ODC^y is computed by assuming that the variables z_1, z_2 are fully observable. The observability $\widetilde{ODC}^{y_i}$ of each variable y_1, y_2 is thus obtained from (5) by neglecting the first term of the sum. In our case, each $\widetilde{ODC}^{y_i}$ is independent from any other variable y_j, i.e., $\widetilde{ODC}^{y_1} \subseteq a_0 a_1$ and $\widetilde{ODC}^{y_2} \subseteq b_0 b_1$.

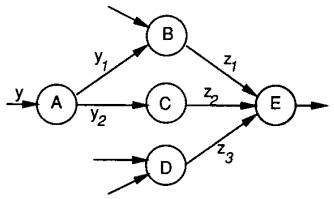


Fig. 12. Model of a circuit with reconvergent fanout.

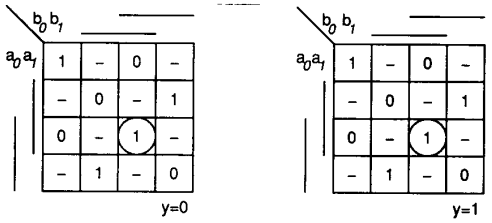


Fig. 13. Model of ODC^y in terms of the variables a_0, a_1, b_0, b_1. Circles represent the approximation given by (60).

$\widetilde{ODC}^y$ is computed as the product of $\widetilde{ODC}^{y1}$ and $\widetilde{ODC}^{y2}$; therefore, $\widetilde{ODC}^y \subseteq a_0a_1b_0b_1$.

In [7], the following approximation is proposed:

$$\widetilde{ODC}^{y1} = \forall_{y2}(ODC^{y1}) = a_0a_1$$

$$\widetilde{ODC}^{y2} = \forall_{y1}(ODC^{y2}) = b_0b_1$$

$$\widetilde{ODC}_y = \widetilde{ODC}^{y1}\widetilde{ODC}^{y2} = a_0a_1b_0b_1. \quad (60)$$

According to this approximation, first the portions of ODC^{yi} independent from y_j, $j \neq i$ are computed, i.e., the observability don't care vectors of the fanout variables are *decoupled*. It is then possible to form their product to obtain a correct approximation. Note that, with this approach, it is necessary to have explicitly ODC^{yi} in terms of all variables y_j and, therefore, flattening operations may be required. These approximations capture at most the circled minterms in the map of Fig. 13, i.e., only two 1's out of the possible eight.

Muroga proposes in [3] an apparently better approximation. It consists in computing only $\widetilde{ODC}^{y2} = \forall_{y1}ODC^{y2} = b_0b_1$, and to compute $\widetilde{ODC}^y$ according to

$$\widetilde{ODC}^y = ODC^{y1}\widetilde{ODC}^{y2} = (a_0y' + a_1y)b_0b_1. \quad (61)$$

The portion of ODC^y covered with this second approach is shown in Fig. 14. Interestingly, the accuracy is not greater than that of (60). Note, however, that (61) requires fewer computations: only ODC^{y2} needs to be independent from y_1. To this purpose, a further simplification is used at the vertex where the actual reconvergence occurs (for example, in Fig. 12, it would be used at vertex E). It consists in computing the portion of ODC^{z2} that is independent from z_1, namely

$$\widetilde{ODC}^{z2} = \forall_{z1}ODC^{z2}. \quad (62)$$

The approximation of ODC^{y2} that can be obtained in this way is thus automatically independent from y_1, and can be used directly in (61).

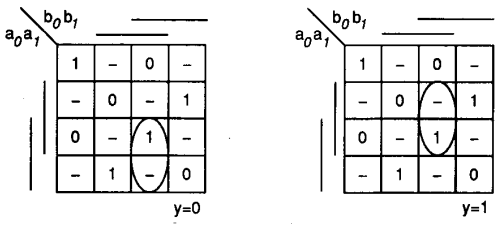


Fig. 14. Approximation of ODC^y provided by (65).

Finally, Savoj *et al.* in [8] propose a better approximation. In the case of two fanout variables it reduces to computing first

$$\widetilde{ODC}^{y2} = \forall_{y1}ODC^{y2} + ODC^{y2}(ODC^{y1})'. \quad (63)$$

and then

$$\widetilde{ODC}^y = ODC^{y1}\widetilde{ODC}^{y2}. \quad (64)$$

Muroga's method is in practice used to avoid the flattening operations in (63), that is,

$$\widetilde{ODC}^{z2} = \forall_{z1}ODC^{z2} + ODC^{z2}(ODC^{z1})' \quad (65)$$

is computed. Although the set computed by (63) or (65) are larger than what provided by (61) and (62), still when (64) is applied (61) is obtained again. All methods proposed so far, therefore, capture essentially the same portion of the *ODC* sets, although with different degrees of computational efficiency. The common idea is that of approximating the observability don't care sets of ODC^{y1}, ODC^{y2} by subsets that can then be multiplied. For this reason, we introduce the following definition.

Definition 8.1: For a copy gate with input variable y and output variables $y_1, y, \cdots, y_n$ the subsets $\widetilde{ODC}^{yi} \subseteq ODC^{yi}$ are said to be **decoupled** if

$$\widetilde{ODC}^y = \prod_{i=1}^{n} \widetilde{ODC}^{yi} \subseteq ODC^y. \quad (66)$$

The methods reviewed in this section can then be regarded as strategies for obtaining decoupled *ODC* subsets.

Several other approximation strategies can be derived from (13). They can readily be obtained by expanding the EXOR's appearing in (13):

$$\begin{aligned} ODC^y = {} & ODC^{y1}ODC^{y2}|_{y1=y'} \\ & + (ODC^{y1})'(ODC^{y2})'|_{y1=y'} \\ & + ODC^{y2}ODC^{y1}|_{y2=y'} \\ & + (ODC^{y2})'(ODC^{y1})'|_{y2=y'}. \end{aligned} \quad (67)$$

If subsets of ODC^{yi} and $(ODC^{yi})'$ are available, then (67) automatically provides a subset of the true ODC^y [9]. (67) also shows that

$$\widetilde{ODC}^y = ODC^{y1}ODC^{y2}|_{y1=y'} + ODC^{y2}ODC^{y1}|_{y2=y'} \quad (68)$$

is also a subset of ODC^y. In order to compare this latter

approximation with the previous ones, we rewrite it in terms of a_0, a_1, b_0, b_1:

$$\widetilde{ODC}^{y} = (a_0 y' + a_1 y)(b_0 y + b_1 y') + (a_0 y + a_1 y')(b_0 y' + b_1 y). \quad (69)$$

The maximum coverage provided by (69) is shown in Fig. 15. Solid circles correspond to the maximum coverage provided by the first product term in (68), while dashed circles correspond to the second one. Notice that both of them cover strictly more of the map of ODC^{y} than the original approximations. Although flattening operations may be required to explicit the dependencies on y_1, y_2, these can be stopped at any time and be replaced by *consensus* operations to eliminate any undesired variables. It is thus possible to consider different CPU time tradeoffs.

I.2. Compatible don't care Sets by Local Rules

Equation (19) provides maximal bounds on the perturbations $\delta_1, \cdots, \delta_n$. Such bounds, however, depend on some of the perturbations δ_i themselves. A maximal compatible don't care set $\widetilde{ODC}^{y_i}$ can be obtained from (19) by observing that, since $\delta_j \subseteq \widetilde{ODC}^{y_j}$ for $j = i+1, \cdots, n$, the expressions $\delta_j(\widetilde{ODC}^{y_j})'; j = i+1, \cdots, n$, represent external don't care conditions that can be added to ODC^{y_i}. Consequently,

$$\widetilde{ODC}^{y_i} = \forall_{\delta_{i+1}, \cdots, \delta_n}\left(ODC^{y_i}|_{\delta_1', \cdots, \delta_{i-1}'} + \sum_{j=i+1}^{n} \delta_j(\widetilde{ODC}^{y_j})' \right), \quad i = 1, \cdots, n \quad (70)$$

is a maximal compatible don't care set [8].

In the case of two perturbations δ_1, δ_2, (70) would in particular result in

$$\widetilde{ODC}^{y_2} = ODC^{y_2}|_{\delta_1'}$$
$$\widetilde{ODC}^{y_1} = ODC^{y_1}|_{\delta_2'}(ODC^{y_1}|_{\delta_2'} + (\widetilde{ODC}^{y_2})'). \quad (71)$$

We analyze here the possibility of computing *maximal compatible* don't care sets for all gates in a network by a single backward traversal. Similarly to the case of full don't care sets, we need different rules for logic and *copy* gates. Consider a logic gate with output y and inputs $y_1, \cdots, y_n$. Given a don't care set ODC^{y}, maximal compatible don't care sets associated with the input variables $y_1, \cdots, y_n$ can be found by applying directly Eq. (70) to $ODC^{y_i} = ODC^{y} + (\partial f^{y}/\partial y_i)'$.

In particular, for two input variables only, $ODC^{y_2}|_{\delta_1'}$ and $ODC^{y_1}|_{\delta_2'}$ are just the *ODC* sets of y_1, y_2, respectively, and

$$ODC^{y_1}|_{\delta_2} ODC^{y_1}|_{\delta_2'} = ODC^{y_1}|_{y_2'} ODC^{y_1}|_{y_2} = \forall_{y_2} ODC^{y_1}.$$

Equation (71) then **coincides** with the approximations (62) and (65).

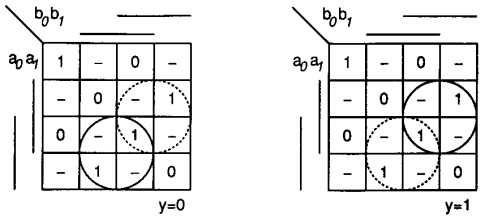


Fig. 15. Our approximation for the *ODC* sets, as provided by (67). Solid circles represent the coverage provided by the first product, dotted circles the coverage provided by the third one.

It is important, however, to distinguish the use of (70) (or (71)) in the context of *approximating ODC* sets and of *computing compatible don't care* sets. For example, in the circuit of Fig. 12, when using (70) for approximating don't care conditions, we consider the portion of the *ODC* set of z_2 which is independent from z_1, and the full *ODC* set of z_3 can be computed. When computing compatible don't care conditions, instead, the portion of ODC_{z_3} actually compatible with those of z_1 and z_2 needs be computed. The compatibility requirement, therefore, shrinks the *ODC* set that could be associated to z_3.

In the case of *copy* gates, the general rule linking maximal compatible don't care set of the input variable to those of its fanout variables is complex. For simplicity, we consider here only the case of a *copy* gate with only two fanout variables.

Let $\delta_1, \cdots, \delta_n$ denote n arbitrary perturbations of the network, corresponding to n network variables $z_1, \cdots, z_n$. The observability don't care set for a variable y, compatible with those of $z_1, \cdots, z_n$ is provided by (70):

$$\widetilde{ODC}^{y} = \forall_{\delta_1, \cdots, \delta_n}\left(ODC^{y} + \sum_{j=1}^{n} \delta_j(\widetilde{ODC}^{z_j})' \right). \quad (72)$$

Suppose y is the input variable of a *copy* gate, with outputs y_1, y_2. From (13), it then follows that

$$ODC^{y} = \forall_{\delta_1, \cdots, \delta_n}\left[\left(ODC^{y_1} + \sum_{j=1}^{n} \delta_j(\widetilde{ODC}^{z_j})' \right) \overline{\oplus} \left(ODC^{y_2}|_{y_1 = y'} + \sum_{j=1}^{n} \delta_j(\widetilde{ODC}^{z_j})' \right) \right]. \quad (73)$$

The two expressions within the parentheses in (73) resemble the *ODC* sets of y_1, y_2, compatible w.r.t. $z_1, \cdots, z_n$. Unfortunately, the *consensus* operation is not distributive w.r.t. the XNOR operation; consequently, the maximal compatible don't care set of y is not in general derivable from those of y_1, y_2, and (73) shows in particular that the full don't care sets ODC^{y_1}, ODC^{y_2} are necessary.

Following the approach outlined in Section (8.1), by taking decoupled subsets of ODC^{y_1} and ODC^{y_2} it is possible to approximate the XNOR operation in (73) by a product. Since *consensus* is distributive w.r.t. the AND oper-

ation, it is possible to write

$$\widetilde{ODC}^y = \forall_{\delta_1,\cdots,\delta_n}\left(ODC^{y_1} + \sum_{j=1}^{n}\delta_j(\widetilde{ODC}^{z_j})'\right) \cdot \forall_{\delta_1,\cdots,\delta_n} \cdot \left(ODC^{y_2}|_{y_1=y'} + \sum_{j=1}^{n}\delta_j(\widetilde{ODC}^{z_j})'\right) = \widetilde{ODC}^{y_1}\widetilde{ODC}^{y_2}. \quad (74)$$

Equation (74) is the one considered in [3], [8]. The following example shows that the set provided by (74) is not, in general, maximal.

Example 16: Consider applying (70) and (74) to derive compatible don't care sets for the gates of the network shown in Fig. 16, by backward network traversal. Clearly, $ODC^w = 0$. The observability don't care set of z is computed according to

$$\widetilde{ODC}^z = \left(\frac{\partial F}{\partial z}\right)' = 0. \quad (75)$$

Next, in order to compute the don't care set of a, first the compatible don't care sets of x and y are determined:

$$\widetilde{ODC}^x = 0; \quad \widetilde{ODC}^y = 0 \quad (76)$$

and eventually their product is formed:

$$\widetilde{ODC}^a = \widetilde{ODC}^y\widetilde{ODC}^x = 0. \quad (77)$$

Using (73) would have resulted in $\widetilde{ODC}^a = 1$. This result is intuitively correct: since the don't cares associated to each of w, z are identically 0, no gate can be changed, and the compatible don't care set associated to a coincides with the full don't care set. □.

I.3. Approximations for Synchronous Networks

In the previous sections it was in particular shown that by using *decoupled* don't cares it is possible to simplify (13) into (61).

This property carries over to synchronous networks. Moreover, we show below that by this approach it is possible to avoid computing the individual sets $ODC^y_{0,k}$ for each network variable and each time point $k = 0, \cdots, P$ and instead compute directly approximations $\widetilde{ODC}^y_0$ of the final observability don't cares.

Different rules are given for logic and *copy* gates. For a logic gate with output variable y, by combining (35) and (31) we obtain for each input variable z:

$$ODC^z_0 = \prod_{k=0}^{P}\left(ODC^y_{w_z,k} + \left(\frac{\partial f^y_{w_z}}{\partial z}\right)'\right) = \left(\frac{\partial f^y_{w_z}}{\partial z}\right)' + \prod_{k=0}^{P} ODC^y_{w_z,k} = \left(\frac{\partial f^y_{w_z}}{\partial z}\right)' + ODC^y_{w_z} \quad (78)$$

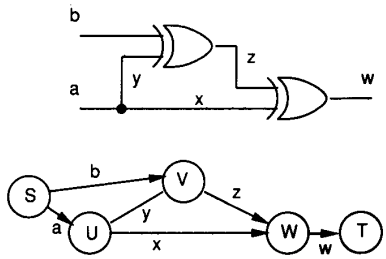


Fig. 16. Circuit topology for the analysis of Compatible observability *don't care*.

which is, except for the retiming of ODC^y by w_z, identical to (9) of the combinational case.

For a *copy* gate with input variable y and outputs v, w, by combining (36) and (31):

$$ODC^y_0 = \prod_{k=0}^{P}(ODC^{\text{ext}} + ODC^v_{0,k} \,\overline{\oplus}\, ODC^w_{0,k}|_{v_0=y'_0}). \quad (79)$$

If $ODC^v_{0,k}$ and $ODC^w|_{0,k}$ are approximated by decoupled subsets, then,

$$\widetilde{ODC}^y_0 = \prod_{k=0}^{P}(ODC^{\text{ext}}_k + \widetilde{ODC}^v_{w_y,k}\widetilde{ODC}^w_{w_y,k}) = \prod_{k=0}^{P}(ODC^{\text{ext}}_k + \widetilde{ODC}^v_{w_y,k}) \cdot \prod_{k=0}^{P}(ODC^{\text{ext}}_k + \widetilde{ODC}^w_{w_y,k}) = (ODC^v_0 ODC^w_0)_{w_y}. \quad (80)$$

Equations (78) and (80) identical to the corresponding rules for combinational circuits (except for a simple retiming operation) and allow us to compute directly $\widetilde{ODC}^y_0$ without considering the observability at different time points.

Appendix II

Proof of Theorem (3.1): Let $\delta_1, \cdots, \delta_f$ denote the perturbations associated to the f fanout variables. The following identity can be easily verified:

$$\begin{aligned}\boldsymbol{ODC}^y &= \boldsymbol{F}^{y_1 y_2 \cdots y_f}(1, \cdots, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, \cdots, 0) \\ &= (\boldsymbol{F}^{y_1 y_2 \cdots y_f}(1, \cdots, 1) \oplus \boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, 1, \cdots, 1)) \,\overline{\oplus}\, (\boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, 1, \cdots, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, 0, 1, \cdots, 1)) \,\overline{\oplus}\, \cdots \,\overline{\oplus}\, (\boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, \cdots, 0, 1) \,\overline{\oplus}\, \boldsymbol{F}^{y_1 y_2 \cdots y_f}(0, \cdots, 0)). \quad (81)\end{aligned}$$

Equation (81) can be rewritten as:

$$\boldsymbol{ODC}^y = \overline{\oplus}_{i=1}^{f} \boldsymbol{F}^{y_1 y_2 \cdots y_f}|_{y_i = \cdots = y_f = y'} \,\overline{\oplus}\, \boldsymbol{F}^{y_1 y_2 \cdots y_f}|_{y_{i+1} = \cdots = y_f = y'}. \quad (82)$$

Equation (15) then follows by observing that each term of the sum in (82) is precisely $ODC^{y_i}|_{y_{i+1}=\cdots=y_f=y'}$ □.

The proof of Theorem (3.3) requires the following lemma ([32, pp. 158–161]):

Lemma 8.1: For a function $H(\delta_1, \cdots, \delta_n)$, the equation

$$H = 1 \tag{83}$$

holds if and only if

$$H'|_{\delta_1'} \subseteq \delta_1 1 \subseteq H|_{\delta_1} \tag{84}$$

and

$$\exists_{\delta_1} H = 1. \tag{85}$$

Proof of Theorem 3.3: It follows immediately by taking $H = E + DC^{\text{ext}}$ and applying iteratively Lemma (8.1) to the functions

$$\exists_{\delta_1, \cdots, \delta_i} E + DC^{\text{ext}}. \tag{86}$$

□

Theorem (3.4) can similarly be proved using the following lemma.

Lemma 8.2: For a function $H(\delta_1, \cdots, \delta_n)$, if

$$\delta_1 1 \subseteq H|_{\delta_1} \tag{87}$$

and

$$H|_{\delta_1'} = 1 \tag{88}$$

then (83) holds. Moreover, the upper bound in (87) is maximal.

Proof of Lemma 8.2: Equations (87) and (88) represent sufficient conditions for (84) and (85), and therefore for (83). This proves the first assertion. To prove the maximality property, suppose by contradiction that the upper bound (87) can be replaced by a bound

$$\delta_1 1 \subseteq \mathcal{G} \tag{89}$$

for some function $\mathcal{G}$ such that, for some assignment of its arguments, all components $\mathcal{G}$ take value 1, while there is a component of $H|_{\delta_1}$ taking value 0. Corresponding to that assignment, (89) would indicate that one can choose $\delta_1 = 1$ and still satisfy Eq. (83). On the other hand, corresponding to that assignment, $H = H|_{\delta_1}$ and therefore at least one component of H takes value 0, a contradiction. The bound expressed by (87) must therefore by maximal. □

Proof of Theorem 3.4: By taking $H = E + DC^{\text{ext}}$ and by iteratively applying Lemma 8.2 to the functions

$$E|_{\delta_1', \cdots, \delta_i'} + DC^{\text{ext}}; \quad i = 1, \cdots, n \tag{90}$$

it follows that

$$\delta_i 1 \subseteq E|_{\delta_1', \cdots, \delta_{i-1}', \delta_i} + DC^{\text{ext}}; \quad i = 1, \cdots, n \tag{91}$$

and

$$E|_{\delta_1', \cdots, \delta_i'} + DC^{\text{ext}} = 1 \tag{92}$$

represent sufficient conditions for (17) and

$$E|_{\delta_1', \cdots, \delta_i'} + DC^{\text{ext}} = 1; \quad i = 1, \cdots, n. \tag{93}$$

If the perturbations δ_i satisfy (91), the terms $\delta_k(E|_{\delta_1', \cdots, \delta_{k-1}', \delta_k})'$ can be regarded as external don't care conditions and therefore added to the r.h.s. of Eq. (91). By arguments similar to those used in Lemma 8.2, such bounds are also maximal.

Since $E|_{\delta_1', \cdots, \delta_n'} = F^y(0, \cdots, 0) \,\overline{\oplus}\, F^y(0, \cdots, 0) = 1$, (92) certainly holds. To complete the proof, it is then sufficient to show that the right-hand side of (91) is identical to that of eq. (??). On the other hand,

$$\begin{aligned} E|_{\delta_1', \cdots, \delta_{i-1}', \delta_i} + DC^{\text{ext}} &= (F^y(0, \cdots, 0, 1, \delta_{i+1}, \cdots, \delta_n) + DC^{\text{ext}}) \\ &\quad \overline{\oplus}\, (F^y(0, \cdots, 0) + DC^{\text{ext}}) \\ &= ((F^y(0, \cdots, 0, 1, \delta_{i+1}, \cdots, \delta_n) \\ &\quad + DC^{\text{ext}}) \,\overline{\oplus}\, (F^y(0, \cdots, 0, \delta_{i+1}, \cdots, \delta_n) \\ &\quad + DC^{\text{ext}})) \,\overline{\oplus}\, (E|_{\delta_1', \cdots, \delta_{i-1}', \delta_i'} + DC^{\text{ext}}). \end{aligned}$$

By using (93) and by recognizing that

$$\begin{aligned} &((F^y(0, \cdots, 0, 1, \delta_{i+1}, \cdots, \delta_n) + DC^{\text{ext}}) \\ &\quad \overline{\oplus}\, (F^y(0, \cdots, 0), \delta_{i+1}, \cdots, \delta_n) + DC^{\text{ext}}) \\ &= ODC^{y_i}|_{\delta_1', \cdots, \delta_{i-1}'} + DC^{\text{ext}} \end{aligned} \tag{94}$$

it then follows

$$\begin{aligned} E|_{\delta_1', \cdots, \delta_{i-1}', \delta_i} + DC^{\text{ext}} \\ = (ODC^{y_i}|_{\delta_1', \cdots, \delta_{i-1}', \delta_i'} + DC^{\text{ext}}) \,\overline{\oplus}\, 1 \end{aligned} \tag{95}$$

thus proving the required identity. □

Proof of Theorem 4.2: Suppose, by contradiction, that (32) holds, but that there are a value n^* and an input sequence s such that (26) does not hold.

It then follows that, for that particular assignment, $E_{n^*} = 0$ and $DC^{\text{ext}}_{n^*} = 0$. Corresponding to the retiming by $-n^*$ of s, $E_0 = 0$. Moreover, it must be $DC^{\text{ri}} = 0$ or otherwise, by the definition of DC^{ri}, the set of sequences $(s_{-n^*})_{n^*}$ (containing in particular s) would be in $DC^{\text{ext}}_{n^*}$, thereby contradicting $s \in DC^{\text{ext}}_{n^*}$. Consequently, corresponding to s_{-n^*} the l.h.s. of (32) takes value 0, a contradiction.

Only if: Suppose, by contradiction, that there exists an input sequence s such that the l.h.s. of (32) takes value 0, but such that (26) holds for every n. For that sequence, $E_0 = 0$ and $DC^{\text{ext}} = 0$. Consider the retiming by an arbitrary k of s. Again, by the time-invariance of the network, corresponding to $s_k E_k = 0$, and therefore in order to satisfy (26) it must be DC^{ext}_k. Therefore, $s_k \subseteq DC^{\text{ext}}_k$ $\forall k \geq 0$, and by the definition of DC^{ext} it should be $s \in DC^{\text{ext}}$, contradicting the assumption that $s \notin DC^{\text{ext}}$. □

Proof of Theorem 4.3: It is sufficient to observe that, from (42),

$$E_n = E^F_{\delta,n} \,\overline{\oplus}\, E^F_{\sigma,n} \supseteq E^F_{\delta,n} E^F_{\sigma,n}.$$

Consequently,

$$E_n + DC_n^{\text{ext}} \supseteq (E_{\delta,n}^F + DC_n^{\text{ext}})(E_{\sigma,n}^F + DC_n^{\text{ext}}) \quad (96)$$

and, by hypothesis, both factors in the right-hand side of (96) are 1. □

As the proof of Theorem 4.4 requires that of Theorem 4.5, the latter is proven first.

Proof of Theorem 4.5: By induction on $\langle m \rangle$. The assertion is trivially true for $m = 0$. To prove the inductive step, recall that the expressions of $ODC^s_{n,\langle m+1\rangle}$ and $ODC^s_{n,\langle m\rangle}$ are derived applying (31) to the definite subnetwork N_d. Each factor appearing in (31) has the form:

$$ODC^s_{n+k,\langle m\rangle} + ODC^s_{n,n+k}|_{\sigma'_{n+1},\cdots,\sigma'_{n+k}} \quad (97)$$

for the expression of $ODC^s_{n,\langle m\rangle}$, while it has the form:

$$ODC^s_{n+k,\langle m-1\rangle} + ODC^s_{n,n+k}|_{\sigma'_{n+1},\cdots,\sigma'_{n+k}} \quad (98)$$

for $ODC^s_{n,\langle m\rangle}$. Since, by the inductive assumption, $ODC^s_{n+k,\langle m\rangle} \supseteq ODC^s_{n+k,\langle m-1\rangle}$, each factor of $ODC^s_{n,\langle m+1\rangle}$ is not smaller than the corresponding of $ODC^s_{n,\langle m\rangle}$, and consequently $ODC^s_{n,\langle m+1\rangle} \supseteq ODC^s_{n,\langle m\rangle}$. □

Proof of Theorem 4.4: We first prove (50) by induction on n. To this regard, observe that (50) holds by hypothesis for $n = -P, \cdots, -1$. As for the inductive step note that, by (39), proving (50) is equivalent to showing that $E_n^S + CDC^{\text{ext}} + ODC^s_{n,\langle m\rangle} = 1$.

Since $E_n^S = E_{\delta,n}^S \,\overline{\oplus}\, E_{\sigma,n}^S$, by recalling (49) it suffices to prove

$$E_{\sigma,n}^S + CDC^{\text{ext}} + ODC^s_{n,\langle m\rangle} = 1. \quad (99)$$

On the other hand, from Theorem 4.1, having $\sigma_{n-k} \subseteq CDC^{\text{ext}} + ODC^s_{n,\langle m\rangle}$ is a sufficient condition to ensure

$$E_{\sigma,n}^S + CDC^{\text{ext}} + ODC^s_{n,\langle m-1\rangle} = 1. \quad (100)$$

Since, by Theorem 4.5, $ODC^s_{n,\langle m\rangle} \supseteq ODC^s_{n,\langle m-1\rangle}$, (100) implies (99). We finally recall that ODC^s_n represents an observability don't care condition for the perturbation σ; consequently, (45) holds by construction. □.

Proof of Theorem 5.1: To prove convergence, it suffices to observe that

1) $CDC^{\text{ri},d}$ represents a subset of combinations of values for the input variables in the interval $[-r, 0]$, of which there is only a finite number, namely $2^{n_i \times (r+1)}$;

2) at each iteration, $CDC^d_{\langle m+1\rangle} \subseteq CDC^d_{\langle m\rangle}$, i.e., the sequence of sets $CDC^d_{\langle m\rangle}$ is monotonically nonincreasing.

In order to prove (52), it is sufficient to observe that by construction

$$CDC_t^{\text{ri},d} \subseteq CDC^{\text{init}} + \sum_{n=0}^{t} (CDC_n^{\text{ri}} + s_n \oplus S_n)$$

$$\supseteq CDC^{\text{ext},d} \forall t \geq 0. \quad (101)$$

□

Acknowledgment

The authors are grateful to Prof. David Dill for the many helpful discussions on sequential synthesis issues, and to Jerry Yang for his help in the implementation of the program.

References

[1] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1062–1081, Nov. 1987.

[2] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multilevel logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, pp. 723–739, June 1988.

[3] S. Muroga, Y. Kambayashi, H. Lai and J. Culliney, "The transduction method—design of logic networks based on permissible functions," *IEEE Trans. Computers*, vol. 38, pp. 1404–1424, 1989.

[4] D. Bostick, G. D. Hachtel, R. M. Jacoby, M. R. Lightner, P. Moceyunas, C. R. Morrison, and D. Ravenscroft, "The Boulder optimal logic design system," in *Proc. ICCAD 1987*, pp. 62–65, Santa Clara CA, Nov. 1987.

[5] A. C. L. Chang, I. S. Reed, A. V. Banes, "Path sensitization, partial Boolean difference and automated fault diagnosis," *IEEE Trans. Computers*, vol. C-21, pp. 189–194, Feb. 1972.

[6] G. D. Hachtel, R. M. Jacoby, P. H. Moceyunas, "On computing and approximating the observability don't care set," in *Proc. Int. Workshop on Logic Synthesis*, Research Triangle Park, May 1989.

[7] P. C. McGeer and R. K. Brayton, "The observability don't care set and its approximations," in *Proc. ICCD 1990*, Cambridge, UK, Sept. 1990.

[8] H. Savoj and R. K. Brayton, "The use of observability and external don't cares for the simplification of multiple-level networks," in *Proc. DAC 1990*, Orlando, FL, June 1990.

[9] M. Damiani and G. De Micheli, "Efficient computation of exact and simplified observability don't care sets for multiple-level combinational networks," in *Proc. IFIP Workshop on Logic and Architecture Synthesis*, Grenoble, France, Apr. 1990.

[10] —, "Observability don't care sets and Boolean relations," in *Proc. ICCAD 1990*, Santa Clara, CA, Nov. 1990.

[11] B. Trakhtenbrot and Y. Barzdine, *Finite Automata: Behavior and Synthesis*. Amsterdam, The Netherlands: North Holland, 1973.

[12] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Menlo Park, CA: Addison-Wesley, 1988.

[13] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: optimizing sequential networks with combinational techniques," in *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 74–84, 1991.

[14] S. Devadas, T. Ma, A. Newton, and A. Sangiovanni-Vincentelli, "A synthesis and optimization procedure for fully and easily testable sequential machines," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1100–1109, Oct. 1989.

[15] S. Devadas, T. Ma, and A. R. Newton, "Redundancies and don't cares in sequential logic synthesis," in *Proc. Int. Test Conf.*, Washington, DC, Aug. 1989.

[16] G. Saucier, M. Crastes de Paulet and P. Sicard, "ASYL: A rule-based system for controller synthesis," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 1088–1097, Nov. 1987.

[17] R. K. Brayton and F. Somenzi, "Boolean relations and the incomplete specification of logic networks," in *Proc. VLSI '89*, pp. 231–240, Munich, Germany, Aug. 1989.

[18] J. Hartmanis and H. Stearns, *Algebraic Structure Theory of Sequential Machines*. Englewood Cliffs, N.J.: Prentice-Hall, 1966.

[19] Z. Kohavi, *Switching and Finite Automata Theory*, Second ed., New York, McGraw-Hill, 1978.

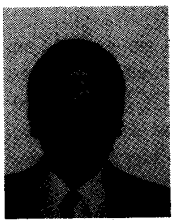
[20] K. T. Cheng and V. D. Agrawal, "State assignment for initializable synthesis," in *Proc. ICCAD 1989*, Santa Clara, CA, Nov. 1989.

[21] M. Damiani and G. De Micheli, "Synchronous logic synthesis; Circuit specifications and optimization algorithms," in *Proc. ISCAS 1990*, pp. 1566–1570.

[22] —, "The role of don't care conditions in synchronous logic optimization," in *Proc. Synthesis and Simulation Meeting and International Interchange (SASIMI)*, pp. 55–62, Kyoto, Japan, 1990.

[23] —, "Synthesis and optimization of synchronous logic circuits from recurrence equations," in *Proc. EDAC 1992*, pp. 226–231, Brussels, Belgium, March, 1992.

[24] ——, "Recurrence equations and the optimization of synchronous circuits," in *Proc. DAC 1992*, Anaheim, CA June 1992.
[25] M. Ligthart, A. Bechtolsheim, G. De Micheli, and A. El Gamal, "Design of a digital audio input output chip," in *Proc. Custom Integrated Circuit Conf.*, pp. 15.1.1–15.1.6, San Diego, CA, May 1989.
[26] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimizations Algorithms for VLSI Synthesis*. Boston, MA: Kluwer Academic, 1984.
[27] B. Lin, H. J. Touati, and A. R. Newton, "Don't care minimization of sequential logic networks," in *Proc. ICCAD 1990*, pp. 414–417, Santa Clara, CA, Nov. 1990.
[28] P. C. McGeer and R. K. Brayton, "Consistency and observability invariance in multi-level logic synthesis," in *Proc. ICCAD 1989*, pp. 426–429, Santa Clara, CA, Nov. 1989.
[29] J. Kim and M. M. Newborn, "The simplification of sequential machines with input restrictions," *IEEE Trans. Computers*, vol. C-21, pp. 1440–1443, Dec. 1972.
[30] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Computer-Aided Design*, vol. 35, pp. 677–691, Aug. 1988.
[31] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient implementation of a BDD package," in *Proc. DAC 1990*, pp. 40–45, June 1990.
[32] F. M. Brown, *Boolean Reasoning*. Boston, MA: Kluwer Academic, 1990.
[33] M. Garey and D. Johnson, *Computers and intractability*. New York, 1979.
[34] G. W. Smith and R. B. Walford, "The identification of a minimal feedback vertex set of a directed graph," *IEEE Trans. Circuits Syst.*, vol. CAS-22, pp. 9–15, Jan. 1975.
[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.



Maurizio Damiani received the Dr.Eng. degree (summa cum laude) from the University of Bologna, Italy, in 1987, and the M.S. degree from Stanford University, Stanford, CA. In 1992, he joined the faculty of the University of Padova, Padova, Italy, as Associate Professor.

His research interests are in the area of sequential logic synthesis and testing, including design for testability and built-in self-test, high-level synthesis, and coding theory.

Dr. Damiani received an AEI Scholarship and a Rotary International Fellowship in 1988 and 1989, respectively.

Giovanni De Micheli (S'79–SM'89), for photograph and biography please see page 46 of the January 1993 issue of this TRANSACTIONS.