The Role of *Don't care* Conditions in Synchronous Logic Optimization *

Maurizio Damiani Giovanni De Micheli

Center for Integrated Systems
Stanford University
Stanford, CA 94305

Abstract

We model synchronous circuits in terms of graphs, logic functions and *don't care* conditions induced by the external and internal interconnections. We then consider the problem of characterizing the degrees of freedom in replacing local logic functions, as done for example by Boolean optimization procedures. We show how synchronous *don't care* sets can be used for this purpose. We then present an algorithm to compute the observability *don't care* set of synchronous circuits.

1 Introduction

Most circuits of interest in digital design are synchronous multiple-level logic circuits, that are interconnections of logic gates and registers with synchronous clocking. Feedback connections are restricted to be through synchronous registers, to guarantee race-free design. We consider in the sequel this class of circuits and focus in particular on single-phase circuits with positive edge-triggered (or master-slave) clocked registers.

Techniques for sequential logic synthesis traditionally use *behavioral* circuit descriptions (in terms of state diagrams or equivalent representations) [1] [2]. In this paper we attack the problems of synchronous logic synthesis by considering a *structural* approach, *i.e.* we consider circuit specifications as interconnection of combinational gates and registers. Such a representation can support iterative improvement of a design. For example, a designer provides a synchronous circuit implementation in terms of a schematic and a tool suite optimizes the circuit while preserving its I/O behavior.

Don't care conditions play a central role in the specification and optimization of logic circuits. Indeed, they represent the degrees of freedom of transforming a network into an equivalent one. *Don't care* conditions provide important information for Boolean transformations, such as Boolean division and redundancy removal. They are also related to the circuit testability properties.

A characterization of synchronous networks by *don't care* sets was presented in [3]. We show here how the *don't care* conditions represent the degrees of freedom in optimizing a synchronous circuit by successive replacement of local logic functions. We present then an algorithm for computing the observability *don't care* set of synchronous networks.

2 Basic concepts and definitions

We consider synchronous multiple-level logic circuits. We assume that these circuits consist of an interconnection of multiple-input single-output combinational logic gates and synchronous delay-type edge-triggered registers. Single-phase clocking is assumed for the sake of simplicity. No direct combinational feedback is allowed.

These circuits are modeled by **synchronous Boolean networks**. A synchronous Boolean network is described by its **directed weighted graph** $G = (V, E, W)$. The elements of the vertex set $V = V^I \cup V^G \cup V^O = \{v\}$ are in correspondence with primary inputs, logic gates, and primary outputs, respectively. Each gate in the circuit may fan out to one or more other gates. In the latter case, for notational convenience, we represent the gate by a vertex pair joined by one edge. The first vertex models the logic function and the second, called fanout vertex, the multiple fanout connection.

The edge and weight set is defined as follows. An edge e from a vertex μ to a vertex ν with weight w models a connection between the corresponding gates through a cascade of (possibly zero) w registers. Zero weights are not represented for the sake of simplicity.

In our network model, a 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)

*The authors acknowledge the support from the Rotary Foundation and NSF, under a PYI award and grant No. MIP 8719546.

variable of a vertex ν if e_x is an edge whose tail (head) end-point is ν. The set of fanout (fanin) variables of a vertex ν is indicated by $FO(\nu)$ ($FI(\nu)$). The weight of an edge e_x is indicated by w_x.

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 the corresponding graph is acyclic.

In the case of combinational circuits, the graph is acyclic and all the weights are zero (and therefore not shown). Note that this model differs from the Boolean network model described in [4], because variables are associated to edges, as in [6], rather than to vertices.

Example 1. A synchronous Boolean network and its associated graph are shown in Fig. 1. It is a portion of the phase decoder of the Digital Audio Input-Output Chip [7], 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 □. **viol**

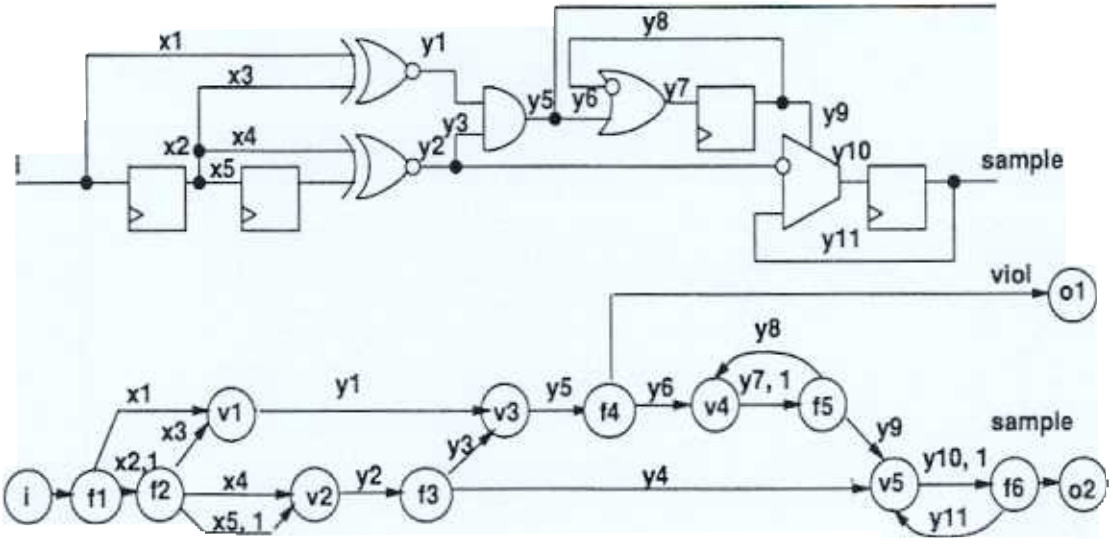

Figure 1: Synchronous Boolean Network and associated graph. Vertices labeled f_i represent fanout points. Vertices labeled i_i and o_i represent primary inputs and outputs, respectively.

We assume a discretization of time into integer time-points $\mathcal{Z} = \{\cdots, -1, 0, 1, \cdots\}$. **A synchronous literal function** is a logic variable or its complement as a function of time, *i.e.* it is a function $\mathcal{Z} \rightarrow \mathcal{B}$, where $\mathcal{B} = \{0, 1\}$. The value a literal function x takes at time n is called **literal value**, or **literal** for short, and it is denoted by $x(n)$. **A synchronous cube** or **synchronous product** or shortly a cube is a Boolean product of values of synchronous literals (e.g. $x(1)\overline{y}(3)$). **A synchronous expression** is a Boolean expression in terms of synchronous literals. In particular, it can be cast as a sum of synchronous cubes. The **retiming** $\mathcal{R}^k(y(n))$ of a literal $y(n)$ by k is the literal $y(n+k)$. The retiming $\mathcal{R}^k(expr)$ of a synchronous expression $expr$ is the retiming of all its literals by k.

The logic values of the network outputs at time n can be regarded as a function $\underline{\mathcal{F}}^n$ of all the network literals at time points $n' \leq n$. With this definition, $\underline{\mathcal{F}}^n$ is a function of an infinite number of arguments. The network structure, however, provides implicitly an expression of $\underline{\mathcal{F}}^n$ in terms of a finite number of arguments. Indeed, each vertex $\nu \in V^G$ is associated to a Boolean function over the same domain as $\underline{\mathcal{F}}^n$. This last function is described by an expression f_ν^n of past values of the fanin variables $y_1, \cdots, y_k$ of ν, *i.e.* $f_\nu^n = f_\nu^n(y_1(n - w_{y_1}), \cdots, y_k(n - w_{y_k}))$. Since each gate is modeled by a time-invariant logic function

$$f_\nu^{n+k} = \mathcal{R}^k(f_\nu^n) \quad \forall\, n, k \in \mathcal{Z} \tag{1}$$

The expression f_ν^0, denoted hereafter by f_ν, is sufficient to derive all expressions f_ν^n. Note that f_ν reduces to an identity function in the case of fanout vertices.

The network model described above implicitly assumes that the content of every register is always the result of the operation performed by its driving gate, *i.e.* that the network is always started in a reachable state. Preset signals therefore need to be accounted explicitly.

3 *Don't care* conditions in synchronous networks

Don't care conditions are introduced by the network interconnection. As in the combinational case [5], they can be classified as external/internal satisfiability/observability *don't care* conditions. The external satisfiability *don't care* set represents input sequences that cannot occur at the network inputs. This set has also been called *controllability don't care* set in [3]. Here it is denoted by SDC_{ext}. Similarly, external observability *don't care* sets represent conditions for which some of the output values are not observed, at some time n. The following examples illustrate some contexts in which such *don't care* sequences arise.

Example 2. Consider the circuit of Fig.(2), representing the cascade interconnection of two simple synchronous networks, and assume that the registers of $M1$ are initially reset to 0. then the sum $z1(0)+z2(0)+\overline{z}3(0)$ cannot occur at the inputs of $M2$. Therefore, $SDC_{ext} \supseteq z_1(0)+z_2(0)+\overline{z}_3(0)$. At time $n = 1$, $z_1(1) = 1$ is impossible, so that $SDC_{ext} \supseteq z_1(1)$. Note then that the cube $z_1(n+1)\overline{z}_2(n)\overline{z}_3(n)$; $n \geq 1$ is a ***don't care*** condition for the network $M2$, *i.e.* this condition cannot occur when $n \geq 1$. This can be shown by looking at network $M1$ and noticing that when, at some time $n \geq 1$, $z_3(n) = 0$, then necessarily $x_1(n)x_2(n) + \overline{x}_1(n-1) = 0$. Similarly, $z_2(n) = 0$, together with the just derived condition $\overline{x}_1(n-1) = 0$, implies $x_2(n-1) = 0$. Hence, $z_1(n+1) = \overline{x}_1(n+1)x_2(n-1) = 0$ and the cube $z_1(n+1)\overline{z}_2(n)\overline{z}_3(n)$ cannot be an input to the network $M2$, for $n \geq 1$. It is easy to verify, however, that $z_1(1)\overline{z}_2(0)\overline{z}_3(0)$ is, in general, a *care* condition for $M2$ because the corresponding inputs to $M2$ can occur, depending on the initial conditions of the registers in $M1$. Hence,

$$SDC_{ext} \supseteq z_1(0) + z_2(0) + \overline{z}_3(0) + z_1(1) + \sum_{n=1}^{\infty} z_1(n+1)\overline{z}_2(n)\overline{z}_3(n) \qquad \square$$

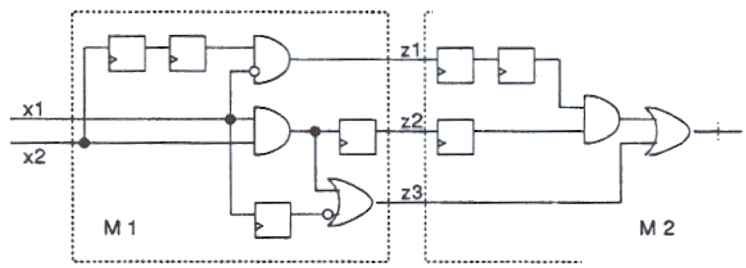

Figure 2: Cascaded Synchronous Boolean Networks

External observability *don't care* conditions are represented here by vectors. More specifically, the i^{th} component of the observability don't care vector $\underline{ODC}_{ext}(n)$ represents the conditions under which the i^{th} primary output of the network is not observed at time n.

Example 3. With reference to Fig. (2), let us consider the situation at the outputs of $M1$. The interconnection of the two networks limits the observability of the primary outputs of $M1$. In particular, a value $z_1(n)$ is not observable at the output of $M2$ when $\overline{z}_2(n+1) + z_3(n+2)$ is satisfied, $\forall n \geq 0$.

The observability of the value $z_2(0)$, however, depends on the initial conditions in M_2: $z_2(0)$ is not observable if $\overline{y}_1(-1) + z_3(1)$ is satisfied. If, for example, M_2 starts in a reset state, then $\overline{y}_1(-1) = 1$ and $z_2(0)$ is certainly not observed. The values $z_2(n)$, $n \geq 1$ are not observed when $\overline{z}_1(n-1) + z_3(n+1)$ is satisfied. Again, assuming all registers initially reset, the variables $z_3(0), z_3(1)$ are always observable, while $z_3(n)$, $n \geq 2$ is observable only if $\overline{z}_2(n-1) + \overline{z}_1(n-2)$ is satisfied. Therefore, the external ODC set for M_1 is completely described by

$$\underline{ODC}_{ext}(0) = \begin{pmatrix} z_3(2) + \overline{z}_2(1) \\ 1 \\ 0 \end{pmatrix}; \underline{ODC}_{ext}(1) = \begin{pmatrix} z_3(3) + \overline{z}_2(2) \\ \overline{z}_1(0) + z_3(1) \\ 0 \end{pmatrix}$$

$$\underline{ODC}_{ext}(n), n \geq 2 = \begin{pmatrix} z_3(n+2)) + \overline{z}_2(n+1) \\ \overline{z}_1(n-1) + z_3(n+1) \\ \overline{z}_2(n-1) + \overline{z}_1(n-2) \end{pmatrix} \qquad \square$$

The internal satisfiability *don't care* set is induced by the internal network interconnection, defined by the equations $y(n) = f_y^n$. Therefore

$$SDC = \sum_{n=-\infty}^{+\infty} \sum_{y \in Y} y(n) \oplus f_y^n \qquad (2)$$

Note in particular that

$$\mathcal{R}^k(SDC) = SDC. \tag{3}$$

For ease of treatment, satisfiability *don't care* sets are represented in the sequel by vectors $\underline{SDC}$ and $\underline{SDC}_{ext}$ with the same dimensions as $\underline{\mathcal{F}}^n$, *i.e.* $|V^O|$, and whose elements are SDC and SDC_{ext} respectively.

3.1 Synchronous logic optimization using *don't care* sets

Boolean optimization techniques of combinational circuits [11] [8] [6] refine a network by replacing iteratively the expressions f_ν by more convenient ones. The replacement of an expression f_ν with g_ν can be achieved when

$$(f_\nu \oplus g_\nu)\underline{1} \subseteq \underline{ODC}_\nu + \underline{ODC}_{ext} + \underline{SDC}_\nu + \underline{SDC}_{ext} \stackrel{def}{=} \underline{DC}_\nu \tag{4}$$

(Here, $\underline{1}$ denotes a vector of all 1's). In other words, the only requirement on g_ν is that the input values for which f_ν differs from g_ν (*i.e.* satisfying $f_\nu \oplus g_\nu$) must be contained in all the components of some vector $\underline{DC}_\nu$, that can be constructed from the internal network structure and/or external specifications. Once $\underline{DC}_\nu$ is computed, any Boolean optimizer can be used to optimize f_ν using the intersection of the components of $\underline{DC}_\nu$ as *don't care* set.

The synchronous case is more complex. In particular, the expressions g_ν that can replace f_ν cannot always be described by an equation of the type of Eq. (4). This can be shown by the following simple example:

Example 4. Consider the circuit in Fig. (3). It realizes the function $z(n) = \overline{x}(n) \oplus \overline{x}(n-1)$. It can easily be recognized that the inverter can be replaced by a simple wire, *i.e.* the function $f_\nu = \overline{x}$ can be replaced by $g_\nu = x$. In this case, $f_\nu \oplus g_\nu = 1$. If Eq. (4) were applicable, then we should conclude that $DC_\nu = 1$, *i.e.* that the vertex is redundant, which is clearly false □.

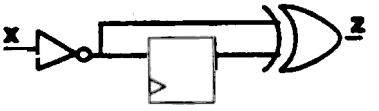

Figure 3: Circuit example

It is possible, however, to obtain descriptions of *don't care* sets for a vertex in a synchronous network. Although these expressions do not capture all the degrees of freedom for circuit optimization, they can advantageously be used in a standard logic optimization framework.

Let us consider the effect of modifying the expression associated to a vertex ν on the network functionality. This modification can be described by considering a **cut network** N_ν, obtained from N by cutting the fanout edge of ν and adding the corresponding variable to the primary inputs. The functionality of the cut network is described by a function $\mathcal{F}_\nu^n$. Clearly, the functionality of the original network can be recovered from that of the cut network. Let y denote the variable corresponding to the cut edge. Then:

$$\mathcal{F}^n = \mathcal{F}_\nu^n|_{y(m)=f_\nu^m;m\le n} \tag{5}$$

and the functionality of any other network obtained by replacing f_ν with some other function g_ν is described by

$$\mathcal{F}_\nu^n|_{y(m)=g_\nu^m;m\le n} \tag{6}$$

The satisfiability *don't care* set associated to the cut network is

$$SDC_\nu = \sum_{n=-\infty}^{+\infty} \sum_{\mu\in V,\mu\neq\nu} y(n) \oplus f_\nu^n \tag{7}$$

Definition 3.1 *Consider a network N', obtained from N by modifying an expression f_ν. The expression g_ν is said to be equivalent to f_ν at time n, and denoted by $g_\nu \stackrel{n}{\approx} f_\nu$, if the equality*

$$\mathcal{F}_\nu^n|_{y(m)=g_\nu^m;m\le n} = \mathcal{F}_\nu^n|_{y(m)=f_\nu^m;m\le n} \tag{8}$$

holds for all the observable components of $\underline{\mathcal{F}}^n$ and for all combinations of literal values that are possible in the cut network. We say that g_ν is equivalent to f_ν, and we denote it by $g_\nu \approx f_\nu$, if $g_\nu \stackrel{n}{\approx} f_\nu$, $\forall n \ge 0$.

From the definition, $g_\nu \approx f_\nu$ if and only if

$$\mathcal{F}_\nu^n|_{y(m)=g_\nu^m;m\le n} \overline{\oplus} \mathcal{F}_\nu^n|_{y(m)=f_\nu^m;m\le n} + \underline{ODC}_{ext}(n) + \underline{SDC}_\nu + \underline{SDC}_{ext} \equiv \underline{1} \tag{9}$$

holds for every n.

Definition 3.2 *Consider a network $\mathcal{N}_y$, obtained from $\mathcal{N}$ by cutting e_y. We call observability don't care function of a value $y(m)$ the function*

$$\underline{ODC}_{y(m)}(n) = \underline{\mathcal{F}}_y^n|_{y(m)=0} \overline{+} \underline{\mathcal{F}}_y^n|_{y(m)=1} \tag{10}$$

The quantity $\underline{ODC}_{y(m)}(n)$ specifies the conditions for which $y(m)$ does not affect the network outputs *at time n*. It represents the complement of the Boolean difference [9] of $\underline{\mathcal{F}}^n$ w.r.t. $y(m)$. In general, $\underline{ODC}_{y(m)}(n)$ may depend on $y(m')$; $m' \neq m$. From the time-invariance properties of the circuit, $\underline{ODC}_{y(m+k)}(n+k) = \mathcal{R}^k(\underline{ODC}_{y(m)}(n))$.

Definition 3.3 *Given a pair of expressions f_ν, g_ν, we denote for $k \geq 0$*

$$\underline{\Delta}_{n-k}^n = \left(\underline{\mathcal{F}}_y^n|_{y(m)=f_\nu^m;m\leq n-k} \overline{+} \underline{\mathcal{F}}_y^n|_{y(m)=g_\nu^m;m\leq n-k}\right)|_{y(m)=f_\nu^m;m>n-k} . \tag{11}$$

Note that $\underline{\Delta}_n^n$ is precisely the first term in the l.h.s. of Eq. (9).
By adding twice $\underline{\mathcal{F}}_y^n|_{y(m)=g_\nu^m;m<n-k;y(m)=f_\nu^m;m\geq n-k}$ in Eq. (11) we have that

$$\underline{\Delta}_{n-k}^n = \underline{\Delta}_{n-k-1}^n \overline{+} \left(\underline{\mathcal{F}}_y^n|_{y(m)=g_\nu^m;m<n-k,y(m)=f_\nu^m;m=n-k} \overline{+} \underline{\mathcal{F}}_y^n|_{y(m)=g_\nu^m;m\leq n-k}\right)|_{y(m)=f_\nu^m;m>n-k} . \tag{12}$$

Given an arbitrary Boolean function $\mathcal{H}$ of arguments $x_0, \cdots, x_n, y$ and a Boolean variable z,

$$\mathcal{H}(x_0, \cdots, x_n, y) \dot{+} \mathcal{H}(x_0, \cdots, x_n, z) = (y \dot{+} z)\frac{\partial \mathcal{H}}{\partial y} . \tag{13}$$

By complementing both terms of this identity and applying it to the term in parentheses in Eq. (12), we obtain the following recursive expression:

$$\underline{\Delta}_{n-k}^n = \underline{\Delta}_{n-k-1}^n \overline{+} ((f_\nu^{n-k} \overline{+} g_\nu^{n-k})\underline{1} + \underline{ODC}'_{y(n-k)}(n)) \tag{14}$$

where $\underline{ODC}'_{y(n-k)}(n) = \underline{ODC}_{y(n-k)}(n)|_{y(m)=g_\nu^m;m<n-k;=f_\nu^m;m>n-k}$. It is now possible to prove the following Lemma:

Lemma 3.1 *Let y denote the fanout variable of a vertex ν. If*

$$(f_\nu^n \dot{+} g_\nu^n)\underline{1} \subseteq \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}'_{y(n)}(n) + \underline{ODC}_{ext}(n) \tag{15}$$

and

$$\underline{\Delta}_{n-1}^n + \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}_{ext}(n) \equiv \underline{1} \tag{16}$$

then $g_\nu \overset{n}{\approx} f_\nu$

Proof. By contradiction. First note that, from Eq. (9) and Definition 3.3, $g_\nu \overset{n}{\approx} f_\nu$ if and only if

$$\underline{\Delta}_n^n + \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}_{ext}(n) \equiv \underline{1} . \tag{17}$$

By using identity (14), Eq. (17) is transformed into

$$\underline{\Delta}_{n-1}^n \overline{+} ((f_\nu^n \overline{+} g_\nu^n)\underline{1} + \underline{ODC}'_{y(n)}(n)) + \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}_{ext}(n) \equiv \underline{1} . \tag{18}$$

The only condition for which Eq. (18) could not hold is when at least one entry, say i, of $\underline{SDC}_{ext} + \underline{SDC}_\nu + \underline{ODC}_{ext}(n)$ is 0 and $\underline{\Delta}_{n-1}^n$ differs from the expression in parentheses in its i^{th} entry. But in this case, by Eq. (15) and (16), the i^{th} entry of $\underline{\Delta}_{n-1}^n$ is 1 and, for entry i, $(f_\nu^n \dot{+} g_\nu^n)\underline{1} \subseteq \underline{ODC}'_{y(n)}(n)$ holds. Therefore, the i^{th} entry of the expression in parentheses in (18) must be 1, and equals the corresponding entry of $\underline{\Delta}_{n-1}^n$. Hence, the i^{th} entry of the l.h.s. of Eq. (18) cannot be 0 □.

Note at this point that Eq. (16) has the same structure as Eq. (17). Lemma 3.1 can thus be used iteratively, to obtain the following result.

Lemma 3.2 *If*

$$(f_\nu^{n-j} \dot{+} g_\nu^{n-j})\underline{1} \subseteq \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}'_{y(n-j)}(n) + \underline{ODC}_{ext}(n); \quad \text{for } j = 0 \quad k-1 \tag{19}$$

and

$$\underline{\Delta}_{n-k}^n + \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}_{ext}(n) \equiv \underline{1} \tag{20}$$

then $g_\nu \overset{n}{\approx} f_\nu$ □

$$(f_\nu^n \oplus g_\nu^n)\underline{1} \subseteq \underline{SDC}_\nu + \prod_{j=0}^{k-1}\left(\mathcal{R}^j(\underline{SDC}_{ext} + \underline{ODC}_{ext}(n)) + \underline{ODC}'_{y(n)}(n+j)\right) \quad (21$$

and

$$\underline{\Delta}^n_{n-k} + \underline{SDC}_\nu + \underline{SDC}_{ext} + \underline{ODC}_{ext}(n) \equiv \underline{1}$$

hold at time n, *then* $g_\nu \overset{n}{\approx} f_\nu$.

Proof. It is sufficient to observe that $f_\nu^{n-j} \oplus g_\nu^{n-j} \subseteq expr$ if and only if $f_\nu^n \oplus g_\nu^n \subseteq \mathcal{R}^j(expr)$ and to apply this property to the Eq. (19) □.

In the following section, we consider the application of Theorem 1 to the important case of definite networks. Note that any cyclic network can be decomposed in an acyclic network and feedback wires [3]. Hence, any optimizer that preserves the input/output functionality of the acyclic portion also insures the final correctness of the complete circuit.

3.2 Definite networks

Let P_ν denote the longest path from ν to the primary outputs in the network graph. Then, $\underline{\mathcal{E}}_\nu^n$ depends on at most P_ν past values of g_ν, and therefore $\underline{\Delta}^n_{n-k} \equiv \underline{1}$ for $k > P_\nu$. Thus, Eq. (22) certainly holds for $k > P_\nu$ and therefore

$$\underline{DC}_\nu(n) = \underline{SDC}_\nu + \prod_{j=0}^{P_\nu}\left(\mathcal{R}^j(\underline{SDC}_{ext} + \underline{ODC}_{ext}(n)) + \underline{ODC}'_{y(n)}(n+j)\right) \quad (23)$$

represents a ***don't care*** set for the vertex ν at time n. Any function g_ν such that

$$(g_\nu^n \oplus f_\nu^n)\underline{1} \subseteq \underline{DC}_\nu(n) \quad \forall n \geq 0 \quad (24)$$

is therefore equivalent to f_ν This condition is equivalent to

$$(g_\nu \oplus f_\nu)\underline{1} \subseteq \prod_{n=0}^{\infty}\mathcal{R}^{-n}\left(\underline{DC}_\nu(n)\right) \overset{def}{=} \underline{DC}_\nu \quad (25)$$

From Eq. (23), taking into account Eq. (3),

$$\underline{DC}_\nu = \underline{SDC}_\nu + \prod_{n=0}^{\infty}\prod_{j=0}^{P_\nu}\left(\underline{ODC}'_{y(0)}(j) + \mathcal{R}^{j-n}(\underline{SDC}_{ext} + \underline{ODC}_{ext}(n))\right) =$$

$$\underline{SDC}_\nu + \prod_{j=0}^{P_\nu}\left(\underline{ODC}'_{y(0)}(j) + \mathcal{R}^j\left(\prod_{n=0}^{\infty}\mathcal{R}^{-n}(\underline{SDC}_{ext} + \underline{ODC}_{ext}(n))\right)\right). \quad (26)$$

The above equation shows that in order to construct the *don't care* set associated to the vertex ν, the following vectors are needed:

1) $\underline{SDC}_\nu$; 2) $\underline{ODC}_{y(0)}(j)$; $j = 0, \cdots, P_\nu$; 3) the ***time-invariant*** component

$$\prod_{n=0}^{\infty}\mathcal{R}^{-n}(\underline{SDC}_{ext} + \underline{ODC}_{ext}(n))$$

of the external *don't care* conditions. The following section presents an algorithm for computing the internal observability *don't care* expressions.

3.3 Observability *don't cares* in definite networks

Given a network, it is possible in principle to compute $\underline{ODC}_{y(0)}(j)$ for any internal variable y by flattening the network Λ_y and applying Eq. (10). Alternatively, the ***chain rule*** [10] may be used, but it requires an exponential number of higher-order derivatives in presence of reconvergent fanout. We show here that it is possible to avoid the flattening operation on the network and compute exact and approximate versions of the observability functions with a single network traversal.

If $\underline{ODC}_{y(0)}(j)$ is known for the fanout variable y of a vertex v, then it is easy to obtain the corresponding expression for all the fanin variables y_i of v from

$$\underline{ODC}_{y_i(0)}(j) = \overline{\frac{\partial f_v^{w_{y_i}}}{\partial y_i(0)}}\underline{1} + \mathcal{R}^{w_{y_i}}\left(\underline{ODC}_{y(0)}(j - w_{y_i})\right) \tag{28}$$

A problem occurs in presence of a reconvergent fanout vertex, as the observability *don't care* function of its input does not necessarily coincide with any of its outputs. In this case, however, the observability *don't care* function can be computed as follows. We present here the method for two fanout variables only, the extension to larger fanout being straightforward. Let y and z_1, z_2 denote the input and output variables of a fanout vertex, respectively, and consider the network N_{12} obtained by cutting N in correspondence of e_{z_1}, e_{z_2}. It realizes a function $\mathcal{F}_{12}^n$. Then, from the definition,

$$\underline{ODC}_{y(0)}(j) = \mathcal{F}_{12}^j|_{z_1(w_y)=z_2(w_y)=0} \overline{+} \mathcal{F}_{12}^j|_{z_1(w_y)=z_2(w_y)=1} \tag{29}$$

By adding twice the term $\mathcal{F}_{12}^j|_{z_1(w_y)=1, z_2(w_y)=0}$ we obtain

$$\underline{ODC}_{y(0)}(j) = \left(\mathcal{F}_{12}^j|_{z_1(w_y)=0} \overline{+} \mathcal{F}_{12}^j|_{z_1(w_y)=1}\right)|_{z_2(w_y)=0} \overline{+} \left(\mathcal{F}_{12}^j|_{z_2(w_y)=0} \overline{+} \mathcal{F}_{12}^j|_{z_2(w_y)=1}\right)|_{z_1(w_y)=1} \tag{30}$$

The terms in parentheses correspond to $\underline{ODC}_{z_1(w_y)}(j)|_{z_2(w_y)=0} = \mathcal{R}^{w_y}(\underline{ODC}_{z_1(0)}(j - w_y)|_{z_2(0)=0})$ and $\underline{ODC}_{z_2(w_y)}(j)|_{z_1(w_y)=1} = \mathcal{R}^{w_y}(\underline{ODC}_{z_2(0)}(j - w_y)|_{z_1(0)=1})$, respectively. Thus,

$$\underline{ODC}_{y(0)}(j) = \mathcal{R}^{w_y}\left(\underline{ODC}_{z_1(0)}(j - w_y)|_{z_2(0)=0} \overline{+} \underline{ODC}_{z_2(0)}(j - w_y)|_{z_1(0)=1}\right) \tag{31}$$

Eq. (31) shows how to compute the observability in presence of reconvergent fanout. A more detailed analysis of exact and simplified expressions of observability *don't care* sets is given in [12] for the combinational case. The analysis can easily be extended, however, to the synchronous case. The following backward traversal algorithm can be used for determining the observability *don't care* sets of all network variables. In the algorithm G denotes the graph, and P its longest path.

OBSERVABILITY(G);
$S := \{primary\ output\ vertices\}$;
while ($S \neq V$) {
  select $v \in V - S$ **such that** $FO(v) \subseteq S$;
  foreach fanout variable z_i **of** v {
    $\mu = head - end - vertex(e_{z_i})$
    for $(j = 0, j < P, j++)$ $\underline{ODC}_{z_i(0)}(j) = \overline{\partial f_\mu^{w_{z_i}}/\partial z_i(0)}\underline{1} + \mathcal{R}^{w_{z_i}}(\underline{ODC}_{y_\mu(0)}(j - w_{z_i}))$

  if (v **is a fanout vertex**)
  $y = fanin - variable(v)$
  for $(j = 0, j \leq P, j++)$
    $\underline{ODC}_{y(0)}(j) = \mathcal{R}^{w_y}(\overline{+}_{k=1}^{|FO(v)|}\underline{ODC}_{z_k(0)}(j - w_y)|_{z_1(0)=\cdots=z_{k-1}(0)=0; z_{k+1}(0)=\cdots=z_{|FO(v)|}(0)=1})$
  $S := S \cup \{v\}$;

The correctness of the above algorithm stems directly from the correctness of Eq. (28) and (31) and from the order in which the vertices are visited. We illustrate here the algorithm on the circuit of Fig. 4.

Example 5. The algorithm begins by computing $\underline{ODC}_{y_9(0)}(0)$ and $\underline{ODC}_{y_8(0)}(1)$ to obtain, from Eq. (31),

$$\underline{ODC}_{y_7(0)}(0) = \begin{pmatrix} 1 \\ y_5(-1) \end{pmatrix};\quad \underline{ODC}_{y_7(0)}(1) = \begin{pmatrix} y_6(1) \\ 1 \end{pmatrix};\quad \underline{ODC}_{y_5(0)}(1) = \begin{pmatrix} 1 \\ y_9(1) \end{pmatrix}$$

The algorithm then computes $\underline{ODC}_{y_4(0)}(1)$ and $\underline{ODC}_{y_6(0)}(0)$ to obtain

$$\underline{ODC}_{y_3(0)}(1) = \begin{pmatrix} \overline{y_8}(0) \end{pmatrix};\ \underline{ODC}_{y_3(0)}(2) = \begin{pmatrix} 1 \\ x_4(1) + y_9(2) \end{pmatrix}$$

y_1 is the fanin variable of a reconvergent fanout vertex. First,

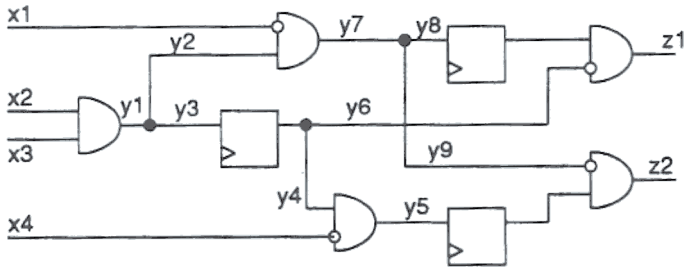

Figure 4: Example circuit for the observability.

$$\underline{ODC}_{y_2(0)}(0) = \begin{pmatrix} 1 \\ \overline{y}_5(-1) + x_1(0) \end{pmatrix} \underline{ODC}_{y_2(0)}(1) = \begin{pmatrix} y_6(1) + x_1(0) \\ 1 \end{pmatrix}$$

are computed, and then from Eq. (31)

$$\underline{ODC}_{y_1(0)}(0) = \begin{pmatrix} 1 \\ \overline{y}_5(-1) + x_1(0) \end{pmatrix}; \underline{ODC}_{y_1(0)}(1) = \begin{pmatrix} (y_6(1) + x_1(0))|_{y_3(0)=0} \overline{+} \overline{y}_8(1)|_{y_2(0)=1} \\ 1 \end{pmatrix} =$$
$$\begin{pmatrix} x_1(0) \overline{+} y_8(1) \\ 1 \end{pmatrix}; \underline{ODC}_{y_1(0)}(2) = \begin{pmatrix} 1 \\ x_4(1) + y_9(2) \end{pmatrix}$$

The algorithm terminates after the computation of the observability *don't care* functions of the primary inputs. These are the external observability *don't care* conditions for the circuit driving the inputs x_i □.

4 Summary

We have presented an extension of the conventional *don't care* theory of combinational networks to the synchronous case. We have shown that, differently from the combinational case, the functions that can replace a combinational gate in a network cannot be expressed by a simple *don't care* set. Expressions for *don't care* sets, however, are derived, that can advantageously be used in a logic optimization framework. These expressions show the key role played by *don't care* conditions and in particular by the internal observability *don't care* sets.

References

[1] S.Devadas, T.Ma, A. Newton, and A.Sangiovanni-Vincentelli, "A Synthesis and Optimization Procedure for Fully and Easily Testable Sequential Machines" *IEEE Transactions on CAD/ICAS*, Vol. CAD-8 No. 10, pp. 1100-1109 October 1989.

[2] G. Saucier , M. Crastes de Paulet and P. Sicard, "ASYL: A Rule-Based System for Controller Synthesis", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6 No. 6, pp. 1088-1097 November 1987.

[3] M. Damiani and G. De Micheli, "Synchronous Logic Synthesis: Circuit Specifications and Optimization Algorithms", *Proc. ISCAS 1990*, pp. 1566-1570.

[4] 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 Transactions on CAD/ICAS*, vol. CAD-7, No. 6, pp. 723-739, June 1988.

[5] G. D. Hachtel and M. R. Lightner, " Top-Down Synthesis of Multiple vel Logic Networks" *Proc. ICCAD 1987*, pp. 316-319, S. Clara, Nov. 1987.

[6] S. Muroga, Y.Kambayashi, H.Lai and J.Culliney, "The Transduction Method - Design of Logic Networks Based on Permissible Functions ", *IEEE Trans. Comp.*, vol. 38, No. 10, pp. 1404-1424, 1989.

[7] M. Ligthart, A. Bechtolsheim, G. De Micheli and A. El Gamal, "Design of a Digital Audio Input Output Chip", *Proceedings of the Custom Integrated Circuit Conference*, San Diego, pp. 15.1.1-15.1.6, May 1989.

[8] 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", *Proc. ICCAD 1987*, pp. 62-65, S. Clara, Nov. 1987.

[9] H. Fujiwara, *Logic Design and Design for Testability*, MIT Press, Cambridge, 1985.

[10] A. C. L. Chang, I. S. Reed, A. V. Banes, "Path Sensitization, Partial Boolean Difference and Automated Fault Diagnosis", *IEEE Transactions on Computers*, C-21, pp. 189 - 194, Feb. 1972.

[11] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, A. Wang , "MIS: A Multiple-Level Logic Optimization System", *IEEE Transactions on CAD/ICAS*, Vol. CAD-6, No. 6, pp. 1062-1081, November 1987.

[12] M. Damiani and G. De Micheli, " Observability *don't care* sets and Boolean Relations ", *Proc. ICCAD 1990*, S. Clara, Nov. 1990.