Synchronous Logic Synthesis: Circuit Specifications and Optimization Algorithms

Maurizio Damiani Giovanni De Micheli

Center for Integrated Systems
Stanford University

Abstract

Synchronous logic networks, that model interconnections of combinational logic gates and synchronous registers, provide a useful abstraction model for digital design. We characterize synchronous logic networks in terms of graphs, logic functions and synchronous *don't care* conditions induced by the external and internal interconnection of the network. We present algorithms to compute the *don't care* set for each local logic function of the network from the synchronous *don't care* conditions that characterize the entire network. Such local *don't care* conditions can be used to optimize locally each logic function to produce a smaller, faster and better testable network.

1 Introduction

Synthesis techniques for combinational multiple-level logic circuits have been the object of extensive investigation [1,2,3,4] and commercial implementations have shown to be practical for product-level design of digital circuits.

Most circuits of interest in digital design are **synchronous** circuits, *i.e.* interconnections of gates and registers with synchronous clocking. Methods for synthesizing and/or optimizing synchronous circuits have been lagging behind, due to the additional complexity of handling registers and feedback connections. Most logic synthesis systems deal with such circuits by partitioning them into an interconnection of a combinational logic component and registers and by optimizing the combinational portion of the circuit by means of combinational logic algorithms.

Techniques directed to sequential logic synthesis traditionally use **behavioral** descriptions of the circuits (in terms of state diagrams or equivalent representations) [5,6,7,8,9]. 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. In addition, such an approach is appropriate when structural circuit specifications are derived automatically from high-level descriptions via high-level synthesis techniques [10].

In this paper we analyze first the concept of **synchronous Boolean network**, and we show that it can be characterized by a set of *synchronous don't care* conditions derived from the structural model. We then show that these conditions can be used to simplify the internal Boolean functions describing the network, as well as identifying the redundancies to provide better testable synchronous networks.

2 Basic concepts and definitions

We consider synchronous circuits that are interconnections of combinational logic gates and synchronous registers with no direct combinational feedback. We assume a single clocking scheme for the sake of simplicity. We model synchronous circuits by **synchronous Boolean networks**. A synchronous Boolean network is described by its **weighted network multigraph** $G = (V, E, W)$. The elements of the vertex set $V = V^I \cup V^G \cup V^O = \{v\}$ are in one-to-one correspondence with primary inputs, logic gates, and primary outputs, respectively. There is an edge e from a vertex μ to a vertex ν with weight $w(e)$ if the output of the gate in μ is connected to an input of the gate in ν through a cascade of (possibly zero) $w(e)$ registers.

Example 1. A synchronous Boolean network and its graph are shown in Fig. 1 and 2, respectively. It is a portion of the phase decoder of the Digital Audio Input-Output Chip [11], 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 □.

In our network model, each Boolean variable corresponds to an edge, and is denoted by a string (e.g. x_1). The edge associated to a variable, say x_1, is indicated by e_{x_1}. 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.

We assume an integer discretization of time into time-points $1, 2, \cdots, n$. A **synchronous literal** is a logic variable or its complement evaluated at a given time-point. The value that a literal x takes at time n is denoted by $x(n)$. A **synchronous cube** or **synchronous product** or shortly a cube is a product of synchronous literals, *e.g.* $x_1(n)x_1(n-1)$.

A **synchronous Boolean function** specifies the value of a variable at a time point in terms of synchronous literals; in particular it can be cast as a sum of synchronous products. The value of a fanout variable $y(n)$ of a vertex ν is given by an expression f_y of the fanin variables, specifying the functionality realized by the logic gate in ν. For example, in the circuit of Fig. (1), $y_3(n) = x_4(n) \overline{+} x_5(n-1)$; hence $f_{y_3} = x_4(n) \overline{+} x_5(n-1)$.

The **retiming** of a synchronous literal $x(n)$ by τ (hereafter indicated by $R^{\tau}(x(n))$) is the literal $x(n+\tau)$. The retiming of an expression by τ is the retiming of all its synchronous literals by τ.

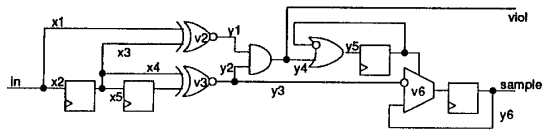

Figure 1: Synchronous Boolean Network.

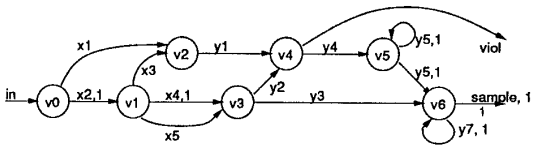

Figure 2: Network graph for the circuit in Fig. (1).

In general, a synchronous Boolean network may have cyclic dependencies, *i.e.* its corresponding graph be cyclic. We assume that each cycle has strictly positive weight, to model the restriction of breaking combinational logic cycles by at least one register. A network is called **unidirectional** or **definite** [6] when the graph is acyclic.

The **fanin** set (**fanout** set) of a vertex v is the subset of vertices that are tail (head) of an edge incident to v and it is denoted by $FI(v)$ ($FO(v)$).

3 Synchronous *don't care* conditions.

Don't care conditions may be derived from circuit **functional** or **structural** specifications. When the circuit functional specifications are known, (e.g. state diagram), functional don't care conditions can be derived by considering the **state** of the network. For example, the knowledge of an input that can be neglected in a given state falls in this category. This type of conditions has been considered, for example, in [5,6] and generalized in [12]. When structural specifications are given, the *don't care* conditions are derived from the network topology. We consider in the sequel the structural *don't care* conditions because we assume as input datum the network structural specifications only.

Typically, *don't care* conditions arise when a synchronous Boolean networks is connected to other ones, or when its inputs are some form of encoded signals. Such *don't care* conditions represent input sequences that never occur at the network input, or unobserved outputs at some time points. As a consequence, the network functionality is incompletely specified, and such *external don't care* conditions can be used to simplify the network structure. By the same token, the functionality of a portion of a Boolean network is incompletely specified, due to its embedding in the entire network. This fact gives rise to the *internal don't care* conditions.

A *don't care* condition can be represented by a synchronous cube. A set of *don't care* conditions can therefore be represented by a sum of synchronous cubes. In particular, a synchronous cube can represent an arbitrary sequence of values.

A synchronous cube is said to represent a **time-invariant don't care condition** if it represents a *don't care* condition $\forall n \geq 0$. Note that there are *don't care* conditions that depend on the initial values of the registers that are not time-invariant and that represent the transient part of the *don't cares*.

Example 2. Consider the circuit of Fig.(3), representing the cascade interconnection of two simple synchronous networks. 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 $M2$. However note that $z1(n) + z2(n) + \overline{z}3(n)$ may occur for $n > 0$. Therefore this condition represents a *transient* controllability *don't care* and not a time-invariant one.

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 the cube $z_1(1)\overline{z}_2(0)\overline{z}_3(0)$ 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 $M1$. Therefore $z_1(n+1)\overline{z}_2(n)\overline{z}_3(n)$ cannot be strictly considered as a time-invariant *don't care*. We prefer therefore to rewrite the condition as $z_1(n+2)\overline{z}_2(n+1)\overline{z}_3(n+1)$; $n \geq 0$ □.

Definition 3.1 *The* **external controllability don't care set** *(ECDC) of a synchronous Boolean network is the sum of all the cubes of input literals representing input conditions that cannot occur.*

With reference to Example 1, the cube $z_1(n+2)\overline{z}_2(n+1)\overline{z}_3(n+1)$ belongs to the external controllability *don't care* set of $M2$.

We focus now on the external *observability don't care* conditions that arise from the interconnection of two synchronous networks.

Example 3. With reference to Fig. (3), let us consider the situation at the circuit outputs of $M1$. The interconnection of the two networks limits the observability of the primary outputs of $M1$. In particular, the variable $z_1(n)$ is not observable at the output of $M2$ when $z_3(n+2) + \overline{z}_2(n+1) = 1$, $\forall n \geq 0$. The expression $z_3(n+2) + \overline{z}_2(n+1)$ represents the *don't care* conditions associated to $z_1(n)$, meaning that this output of $M1$ is not observed (sampled) at a given time n □.

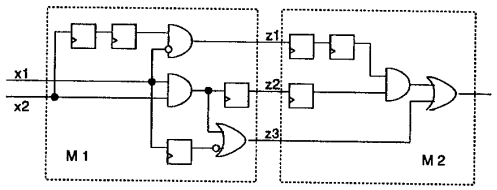

Figure 3: Cascade of two synchronous networks. The network M2 limits the observability of M1, while M1 limits the controllability of M2

Definition 3.2 *The* **external observability don't care set** $(EODC_z(n))$ *of a primary output variable* $z(n)$ *of a synchronous*

Boolean network is the sum of all the cubes of output variables representing output conditions that are not observed.

From Example 2, $z_3(n+2)+z_2(n+1)$ then belongs to the external observability don't care set of $M1$. In general, an external don't care set condition involves a sequence of output variables.

Let us now consider the don't care conditions induced by the internal network topology.

Definition 3.3 *The* **internal controllability don't care set** *($ICDC$) of a synchronous Boolean network is the sum of all the conditions of type: $y(n) \oplus f_y$; $n \geq 0$ for each internal variable y of the network.*

This definition is an extension of the corresponding notion for the combinational networks [2]. Notice that each vertex imposes a time-invariant constraint. Hence, the internal controllability *don't care set* can be represented by the sum of a finite number of time-invariant *don't cares.*

Example 4. The condition $z_1(n) \oplus \overline{(}x_1(n)x_2(n-2))$; $n \geq 2$ belongs to the internal controllability *don't care* set of $M1$ □.

The notion of internal observability in a synchronous network can be described by saying that a value of an internal variable y at time n is not observable if it can be perturbed (*i.e.* by changing $y(n)$ into $\overline{y}(n)$) without affecting any primary output at time n or later.

Example 5. Consider the circuit in Fig. (4). It is easy to verify that the variable y_4 is not observable at time n (*i.e.* that the value $y_4(n)$ is not observable) if $y_6(n-2) = 0$. The condition for the non observability of $y_4(n)$ can be then expressed by the function $\overline{y_6}(n-2)$ □.

Definition 3.4 *The* **internal observability don't care set** *$IODC_y(n)$ is the set of conditions for which $y(n)$ is not observed at any output at time $n+\tau$, $\forall \tau \geq 0$.*

Notice that the concept of observability in a synchronous network is related to the value of a variable at a specific time-point rather than to the variable itself. In general, it is not possible to derive a time-invariant expression of any $IODC_y(n)$ in terms of arbitrary network variables. This is shown by the following example.

Example 6. In Example 4, we derived $IODC_{y_4}(n) = \overline{y_6}(n-2)$; $n \geq 0$, which is actually a time-invariant expression. On the other hand, $y_6(n-2)$ depends on the initial conditions in the network for $n < 4$. In particular, if the register are all initially set to zero, we have that $IODC_{y_4}(0) = IODC_{y_4}(1) = 1$; $IODC_{y_4}(2) = x_4(0)$, $IODC_{y_4}(3) = x_4(1)$, and finally $IODC_{y_4}(n) = x_4(n-2)(\overline{x_3}(n-4) + \overline{x_2}(n-4))$ for $n \geq 4$. The above expression, in terms of the network primary inputs, is obviously not time-invariant □.

Algorithms for the computation of expressions for the observability *don't cares* will be considered in the next section.

4 Optimization of synchronous Boolean networks using DC conditions

This section deals with the problem of determining the *don't care* conditions that are actually useful in simplifying the combinational Boolean function at the vertices of a synchronous Boolean network, given the circuit specifications. For this reason we want to compute the local *don't care* set of each combinational Boolean function. In particular, since the external *don't care* conditions are a problem datum for a given network and $ICDC$ can be easily computed, we will focus first on the problem of determining the **observability** *don't cares* for a vertex.

Let $DC_v(n)$ denote all the *don't care* conditions at time n for a vertex v:

$$DC_v(n) = ICDC(n) \cup IODC_v(n) \cup ECDC(n) \cup EODC(n)$$

It is obvious that a *don't care* can be useful for simplifying the vertex v only if it represents a *don't care* condition for every n. In other words, it has to be a time-invariant *don't care* condition.

The problem of extracting the time-invariant component of $DC_v(n)$ for a vertex will be considered in particular in Section 4.3. Once extracted, this component can be used by any two-level logic minimizer (*e.g.* ESPRESSO) for simplifying the Boolean function.

4.1 Computation of the observability *don't cares* in unidirectional networks

We present here an algorithm for the construction of an expression of $IODC_v(n)$ for the internal variables of a synchronous network. For multiple-output circuits, the algorithm maintains separate expressions for the observability of a variable at each output. For a variable y in a circuit with k output variables z_i ; $i = 1, \cdots, k$ these expressions are maintained in an array $\underline{IODC}_y(n) = (IODC_y(n, z_1), IODC_y(n, z_2), \cdots, IODC_y(n, z_k))$. Since a variable is said to be unobservable if it is unobservable at any output,

$$IODC_y(n) = \prod_{i=1}^{k} IODC_y(n, z_i) \tag{1}$$

The algorithm is based on the following results. Given $\underline{IODC}_v(n)$ of a vertex v, it is possible to compute $\underline{IODC}_y(n)$ for all its **fanin** variables y from the equation:

$$\underline{IODC}_y(n) = \underline{IODC}_v(n + w(e_y)) + \overline{\frac{\partial f_v(n + w(e_y))}{\partial y(n)}}. \tag{2}$$

We use $f_y(n + w(e_y))$ because $y(n)$ has to traverse $w(e_y)$ registers before affecting the gate inputs, and therefore could be observed only at $n + w(e_y)$ at the gate output.

The major problem of determining the observability of a vertex, however, is that of dealing with the problem of reconvergent fanout. We want to solve the problem (well known in the combinational case [13]) of determining the observability of a vertex given that of its fanout variables. We have shown in [14] that this is indeed possible in the combinational case. We use here an extension of that result for the synchronous case. Namely, given the *don't cares* of two **fanout** variables y_1 and y_2 of a vertex v, it is possible to obtain $\underline{IODC}_v(n)$ from

$$\underline{IODC}_v(n) = \underline{IODC}_{y_1}(n)|_{\overline{y}_2(n)} \overline{\oplus} \underline{IODC}_{y_2(n)}|_{y_1(n)} \tag{3}$$

The above expression generalizes to

$$\underline{IODC}_v(n) = \overline{\bigoplus}_{i=1}^{m} \underline{IODC}_{y_i}(n)|_{y_1(n), \cdots, y_{i-1}(n), \overline{y}_{i+1}(n), \cdots, \overline{y}_m(n)} \tag{4}$$

for a vertex with outdegree m [14].

Based on this formula, it is possible to write an algorithm for the computation of $\underline{IODC}_y(n)$ as follows:

OBSERVABILITY(G);
$S := \{primary\ output\ vertices\ with\ empty\ fanout\ set\}$;
while ($S \neq V$) {
  select $\nu \ni S$ such that $FO(\nu) \subseteq S$;
  foreach fanout variable y of ν {
    /* μ denotes the head-end vertex of the edge e_y */
    $\underline{IODC}_y(n) = \underline{IODC}_\mu(n + w(e_y)) + \overline{(\partial(f_\mu(n + w(e_y))/\partial y(n))}$
      /* compute $\underline{IODC}_y$ by Eq. (2) */
  }
  $\underline{IODC}_\nu(n) = \overline{\Phi_{i=1}}^n\ \underline{IODC}_{y_i}|_{y_1(n),\cdots y_{i-1}(n),\overline{y}_{i+1}(n)\cdots\overline{y}_n(n)}$
      /* compute $\underline{IODC}_\nu$ by Eq. (4) */
  $S := S \cup \{\nu\}$;
}

The algorithm is linear in the number of edges in the graph.

Example 7. We illustrate here the operation of the algorithm on the circuit of Fig. 4.

Initially, $S = \{v_4, v_5\}$. The algorithm begins by selecting the vertices v_2 and v_3, whose observability don't cares, calculated by Eqs. (2) and (3), are

$$\underline{IODC}_{v_2}(n) = \begin{pmatrix} y_3(n-1) \\ y_4(n+2) \end{pmatrix};\ \underline{IODC}_{v_3}(n) = \begin{pmatrix} y_5(n+1) \\ y_6(n-2) \end{pmatrix}$$

Then, $S = \{v_2, v_3, v_4, v_5\}$. The algorithm selects v_1, whose fanout variables have *don't cares* described by

$$\underline{IODC}_{y_2}(n) = \begin{pmatrix} \overline{x}_4(n+2) + y_3(n+1) \\ \overline{x}_4(n+2) + y_4(n+4) \end{pmatrix};$$

$$\underline{IODC}_{y_1}(n) = \begin{pmatrix} x_1(n+1) + y_5(n+2) \\ x_1(n+1) + y_6(n-1) \end{pmatrix}$$

From Eq. (3), it follows

$$\underline{IODC}_{v_1}(n) =$$
$$\begin{pmatrix} (\overline{x}_4(n+2) + y_3(n+1))|_{\overline{y}_1(n)} \overline{\Phi} (x_1(n+1) + y_5(n+2))|_{y_2(n)} \\ (\overline{x}_4(n+2) + y_4(n+4))|_{\overline{y}_1(n)} \overline{\Phi} (x_1(n+1) + y_6(n-1))|_{y_2(n)} \end{pmatrix}.$$

By noticing that $y_5(n+2)|_{y_2(n)} = 0$, and that $y_3(n+1)|_{\overline{y}_2(n)} = 0$, it follows that

$$\underline{IODC}_{v_1}(n) =$$
$$\begin{pmatrix} \overline{x}_4(n+2) \overline{\Phi} x_1(n+1) \\ (\overline{x}_4(n+2) + y_4(n+4))|_{\overline{y}_1(n)} \overline{\Phi} (x_1(n+1) + y_6(n-1))|_{y_2(n)} \end{pmatrix}.$$

In particular, note that the algorithm handled correctly the fanout reconverge at vertex v_1 and the dependence of $IODC_{v_1}(n)$ on the initial conditions $y_6(-2)$, $y_6(-1)$ for $n = 0, 1$.

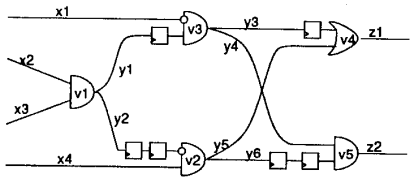

Figure 4: Example unidirectional network

In the end, the algorithm computes the observability *don't care* conditions for the primary input vertices, by means of Eqs. (2-3) again. Such *don't cares* represent *external* observability *don't cares* for the synchronous Boolean network driving the circuit under consideration, and may be used for its optimization.

4.2 Cyclic networks.

The computation of the full *don't care* conditions for cyclic networks requires a more elaborate treatment. For the sake of conciseness, we report here on an extension of the technique so far developed for acyclic networks, even though the computed *don't care* conditions may be incomplete.

Any cyclic network can be reduced to an acyclic one by *cutting* the feedback loops and adding the variables corresponding to the cut edges to the primary inputs and outputs. Alternatively, a cyclic network may be regarded as the feedback interconnection of two acyclic networks, the latter containing only the feedback interconnections.

Fig. (5) shows the circuit of Fig. (1) redrawn as a feedback connection of two acyclic networks $M1$ and $M2$, the second one constituted purely of wires.

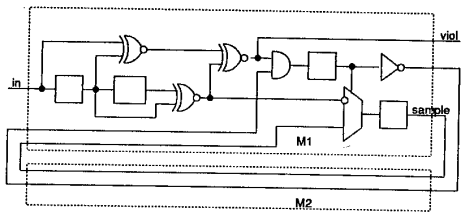

Figure 5: The DAIO circuit of Fig. (1) as a feedback connection of two acyclic networks $M1$ and $M2$

Consider in particular the problem of determining the observability *don't care set* of a vertex in a cyclic network. The OBSERVABILITY algorithm can be used on the acyclic portion of the network, assuming initially that the added primary outputs are completely observable. The algorithm then computes an estimate of the *don't cares* of the network. Such an estimate represents actually a *subset* of the true observability *don't care*. In fact, it assumes that a variable is not observable if it is not observable at *any* output in the acyclic network. But a variable can be not observable even if it is observable at some of the *added* outputs, provided that these latter are then not observable at any time in the future through the feedback interconnections.

4.3 Computing local time-invariant DC conditions

The internal controllability and observability *don't care* sets can be computed for each vertex of the network as shown above. We denote by $DC_\nu(n)$ such local DC set. Note that $DC_\nu(n)$ may be not time-invariant, and therefore not be useful as such for logic minimization.

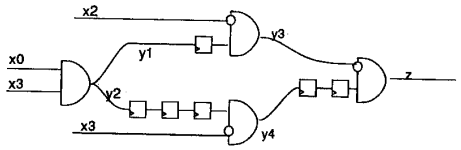

Figure 6: A redundant circuit.

We therefore need to extract the time-invariant component of $DC_\nu(n)$, hereafter denoted by $TIDC_\nu(n)$.

In the case of acyclic networks (or in the case of cyclic ones, where the observability *don't care* are computed as described in Section 4.2), then $DC_\nu(n)$ has a finite transient component plus a time-invariant

one. The instance of $TIDC_\nu(n)$ for $n = 0$ can be then computed as a finite product:

$$TIDC_\nu = \prod_{k=0}^{N} R^{-k}(DC_\nu(k)) \quad (5)$$

where N is an upper bound on the duration fo the transient. Then $TIDC_\nu(n) = R^n(TIDC_\nu(0))$. In particular for acyclic networks with no fanout, N is the longest path in the network graph. When external DC conditions are considered, the duration of their transient component also must be taken into account.

Example 8. We illustrate here on the circuit of Fig. (6) how the sequential *don't care* conditions can be used to simplify a synchronous Boolean network. Assume that:

- the inputs x_i; $i = 0, \cdots, 3$ represent the outputs of a 4-bit counter, so that in particular $EXDC(n) \supseteq x_0(n) \oplus x_0(n+4) + x_1(n) \oplus x_1(n+4)$; $n \geq 0$
- the internal registers are initially in the reset state.

The observability *don't care* set for $y_3(n)$ is given by $IODC_{y_3}(n) = \overline{y_4}(n-2)$. From the assumptions on the initial register conditions, $IODC_{y_3}(n) = 1$ for $n = 0, \cdots, 4$, and $IODC_{y_3}(n) = \overline{y_2}(n-5) + x_2(n-2)$ for $n \geq 5$.

From the input *don't care* conditions it can be easily verified that $y_2(n-5) = y_2(n-1)$; $n \geq 5$, while the internal structure implies $y_2(n) = y_1(n)$; $n \geq 1$. Hence, the *don't care* set of $y_3(n)$ can be rewritten as $DC_{y_3}(n) = 1$ for $n \leq 4$ and $DC_{y_3}(n) = x_2(n-2) + \overline{y_1}(n-1)$ for $n \geq 5$.

It can be then verified that the time-invariant component is $TIDC_{y_3}(n) = x_2(n-2) + \overline{y_1}(n-1)$ so that the function expressing y_3 can be simplified to $y_3(n) = \overline{x_3}(n)$ □.

It is worth remarking the importance of taking into account correctly the initial conditions: in the same example, if it is assumed that all registers are preset to 1, no simplification is possible without changing the network behavior.

5 Conclusions.

We proposed here an approach to the synchronous logic optimization that starts from a *structural* description of a network. Each synchronous circuit is modeled by a synchronous Boolean network, and the interconnections are described by the external *synchronous don't care* conditions that circuits impose to each other. The network structure induces *internal don't care* conditions on the logic gates. These *don't care* conditions, together with the external ones, can be used to simplify the network itself, and to remove *redundancies*. In particular, an algorithm has been proposed for determining the local observability *don't care* sets, which typically represents the most difficult problem in the exploitation of such techniques.

6 Acknowledgments

This research was supported in part by a fellowship of the Rotary Foundation and by AT & T and DEC, jointly with NSF, under a PYI award program. We also acknowledge support from NSF under contract # MIP 8719546.

The helpful discussions with Frederic Mailhot and Michiel Ligthart are here gratefully acknowledged.

References

[1] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. " MIS: A Multiple-level Logic Optimization System". *IEEE Transaction on CAD*, CAD-6, No. 6:pp. 1062–1081, 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 Transaction on CAD*, CAD-7, No. 6:pp. 723–739, 1988.

[3] S.Muroga, Y.Kambayashi, H.Lai, and J.Culliney. " The Transduction method - Design of Logic Networks Based on Permissible Functions ". *IEEE Transaction on Computers*, 38, N. 10: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 *Proceedings of ICCAD*, pages 62–65, Nov 1987.

[5] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines*. Prentice - Hall, Englewood Cliffs, N. J., 1966.

[6] T. L. Booth. *Sequential Machines and Automata Theory*. J. Wiley & Sons, New York, 1967.

[7] K.-T. Cheng and W. D. Agrawal. " Design of Sequential Machines for Efficient Test Generation ". In *Proceedings of ICCAD*, pages pp. 358 – 361, Nov. 1989.

[8] S.Devadas, A. R. Newton, and A. Sangiovanni-Vincentelli. " Irredundant Sequential Machines via Optimal Logic Synthesis ". *IEEE Transaction on CAD*, 9, N. 1:pp. 8–18, Jan. 1990.

[9] S.Devadas and A. R. Newton. "Decomposition and Factorization of Sequential Finite-State Machines ". *IEEE Transaction on CAD*, 8, N. 11:pp. 1206–1217, Nov. 1989.

[10] G. De Micheli and D. Ku. " HERCULES - A system for High-Level Synthesis. In *Proc. 25th Design Automation Conference*, pages 483–488, 1988.

[11] M. Ligthart, A. Bechtolsheim, G. De Micheli, and A. El Gamal. " design of a Digital Audio Input-Output chip". In *Custom IC Conference*, pages pp. 15.1.1 – 15.1.6, May 1989.

[12] S. Devadas. " Redundancies and Don't Cares in Sequential Logic Synthesis". In *Proc. of IEEE Int. Test Conf.*, pages 491–500, Aug 1989.

[13] G. D. Hachtel, R. M. Jacoby, and P. H. Moceyunas. " On Computing and Approximating the Observability Don't Care Set ". In *Proceedings on the International Workshop on Logic Synthesis*, Research Triangle Park, May 1989.

[14] M. Damiani and G. De Micheli. " Efficient Computation of Exact and Simplified Observability Don't Care Sets for Multiple-level Logic Synthesis". In *Internal Report CSL-TR 90, Stanford University*, page , 1990.