Computing observability don't cares efficiently through polarization

Computing Observability Don't Cares Efficiently Through Polarization

Harm Arts, Michel Berkelaar, *Member, IEEE*, and Koen van Eijk, *Member, IEEE*

***Abstract*—A new method is presented to compute the exact observability don't cares (ODC's) for multiple-level combinational circuits. A new mathematical concept, called polarization, is introduced. Polarization captures the essence of ODC calculation on the otherwise difficult points of reconvergence. It makes it possible to derive the ODC of a node from the ODC's of its fanouts with a very simple formula. Experimental results for the 39 largest MCNC benchmark examples show that the method is able to compute the ODC set (expressed as a Boolean network) for all but one circuit in at most a few seconds.**

***Index Terms*—Don't cares, logic synthesis.**

I. Introduction

OBSERVABILITY don't cares (ODC's) play a central role in the synthesis of Boolean networks. Together with the external don't cares (EDC's) and the satisfiability don't cares (SDC's) they represent the freedom one has to optimize the network. Especially the computation of the ODC's has been topic of research because of its complexity.

Several papers have been published on the subject of ODC calculation. In [1], Bartlett *et al.* propose to calculate the ODC's by flattening the network. This is, however, impractical for most circuits, because of the size needed for the representation. In [7], Muroga *et al.* propose exhaustive simulations, which is very time consuming. To reduce computational complexity it was proposed to calculate the ODC of a node from the ODC's of its direct fanouts in [4] by Brayton *et al.* However, computing the ODC in this way is not straightforward in the presence of reconvergent fanouts. To solve this problem, [4] proposes using the chain rule, originally introduced by Chiang *et al.* in [5]. As it turns out, the use of this rule results in very complex calculations very quickly, so [4] proposes using approximations for large circuits. In [6], Damiani *et al.* present a method which is computationally less complex, but still approximations are needed for the larger circuits. In [8], Savoj *et al.* use an observability relation to calculate the ODC's. Although the method does not need to calculate the ODC for each primary output separately, the operations per node are much more complex. The paper itself does not present any results, but the authors themselves comment [9]: "We implemented the algorithm of the ICCAD paper but the algorithm was not practical for large circuits. We concluded that ODC's could be usually [only] computed for circuits that were collapsible in two levels."

In this paper we present a method which also derives the ODC of a node from the ODC's of its direct fanouts and also does not need to calculate the ODC for each primary output separately, but the operations per node are very simple: only an AND over the ODC's of the fanouts, and a cofactor operation are needed. This is obtained by introducing the concept of *polarization*. For each node the *polarized observability don't care* (PODC) is calculated. The polarization exactly models the reconvergence in a network such that cofactoring the PODC will "expand" and/or "shrink" the PODC such that the resulting ODC will be correct.

We feel the main contribution of this paper is the simple mathematical formulation of the construction of the ODC network with the use of the PODC's without the explicit use of XOR or XNOR operations. Another contribution is the large results table. All previously published papers mentioned above are either completely theoretical or show very few results, which leaves no room for comparison of different methods. Our results section shows that the complete ODC network can be derived with our method even for large circuits, and allows future papers to compare their results to ours.

Another advantage of the method presented in this paper is that it makes the use of EDC's very simple. The PODC's calculated at the input of a network can be handed over as EDC's to a feeding network directly, representing the complete EDC. These PODC's also directly imply the Boolean relations for the equivalence classes [3], [6], as is shown in Section IV.

In this paper we express the ODC's as a Boolean network. This network can be used directly by the synthesis system [2]. Alternatively, the ODC's could be expressed in other representations suitable for Boolean reasoning, such as BDD's, but this approach is not tested in this paper.

The method is implemented and tested on the entire set of MCNC combinational multiple-level benchmarks.

II. Definitions and Notation

ODC's are commonly calculated using a Boolean network [1] to model a combinational circuit. In a Boolean network, each node is associated with a Boolean expression [e.g., a sum of cubes (SOC) expression] in terms of its fanin nodes (or fanin edges). In this paper, we will use a network of factored forms. In such a network each node is associated with a simple AND or OR expression, and inverters are modeled on the edges. This is no limitation since any Boolean expression itself is

Manuscript received August 14, 1997; revised January 23, 1998. This paper was recommended by Associate Editor A. Saldanha.

H. Arts is with Ambit Design Systems Inc., Santa Clara, CA 95054 USA (e-mail: harm@ambit.com).

M. Berkelaar and K. van Eijk are with Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands (e-mail: m.r.c.m.berkelaar@ele.tue.nl; c.a.j.v.eijk@ele.tue.nl).

Publisher Item Identifier S 0278-0070(98)05200-2.

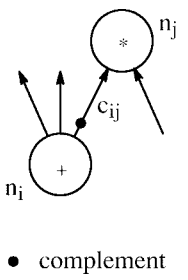

Fig. 1. Nodes and connections.

also a factored form. The advantage of using a network of factored forms is that there is no implicit reconvergence, which is clearly of great importance when calculating ODC's.

A multiple-output combinational circuit is modeled by a network of factored forms. The network can be specified by an acyclic graph $G = (N, C)$ (see Fig. 1). Each node $n_i \in N$ represents either a primary input or a basic Boolean operation, i.e., an AND or an OR operation. There is a directed edge $c_{ij} \in C$ for each connection from node n_i to node n_j. Each connection can have an inverter property. The primary input (output) nodes in N are identified by the set of indexes $I(O)$. A primary input (output) does not have any incoming (outgoing) edges.

We define two functions.

Definition 1: $OP : N \setminus I \rightarrow \{+, *\}$ returns the operation represented by node n.

$INV : C \rightarrow \{\text{FALSE}, \text{TRUE}\}$ returns TRUE if c has an inverter property.

Variable v_i denotes the variable at the output of node n_i. Since we want to be able to distinguish between the value at the output of a node and the value which is at the input of a connected node (see Fig. 1), we also introduce variables for all connections: v_{ij} denotes the variable at input c_{ij} of node n_j. The letters p and q will be used to denote either an index or an index pair, so variable v_p can denote either a node or a connection variable.

Definition 2: The fanin of a node: $FI(j) = \{ij \mid c_{ij} \in C\}$. The fanin of a connection: $FI(ij) = i$.

Definition 3: The fanout of a node: $FO(i) = \{ij \mid c_{ij} \in C\}$. The fanout of a connection: $FO(ij) = j$.

The Boolean function f of a node or a connection can be derived using the following rules:

$$f_j = \begin{cases} \sum_{ij \in FI(j)} v_{ij} & \text{if } OP(j) = + \\ \prod_{ij \in FI(j)} v_{ij} & \text{if } OP(j) = * \end{cases}$$

and

$$f_{ij} = \begin{cases} v_i & \text{if } INV(ij) = \text{FALSE} \\ \overline{v}_i & \text{if } INV(ij) = \text{TRUE}. \end{cases}$$

Definition 4: The transitive fanin is defined recursively

$$TFI(p) = FI(p) \cup \left(\bigcup_{q \in FI(p)} TFI(q) \right).$$

Definition 5: The transitive fanout is defined recursively

$$TFO(p) = FO(p) \cup \left(\bigcup_{q \in FO(p)} TFO(q) \right).$$

We say that function f_p depends on every variable which is in its transitive fanin.

Definition 6: Cofactoring a function to a variable or its complement:

$$f|_{v_p} = f(v_p = 1)$$
$$f|_{\overline{v}_p} = f(v_p = 0).$$

III. Observability Don't Cares

The ODC of variable v_p at node n_k is a Boolean function which gives the conditions for which the actual value of variable v_p can not be observed at node n_k.

Definition 7: The ODC of variable v_p at node n_k

$$ODC_p^k = f_k|_{v_p} \odot f_k|_{\overline{v}_p}$$

where $\odot$ is the XNOR operator.

The ODC of a variable at all primary outputs is a Boolean function which gives the conditions for which the actual value of the variable can not be observed at any primary output.

Definition 8: The ODC of variable v_p at all primary outputs

$$ODC_p = \prod_{k \in O} ODC_p^k.$$

Creating the ODC as a network of factored forms, using these definitions, is relatively simple, but the resulting network turns out to be very complex for many circuits. As a result the calculation of the ODC in this way, by expressing it in SOC or BDD's, be it in terms of primary inputs or local variables, is known to be computationally very expensive [1].

Deriving the ODC of a node from the ODC's of its direct fanouts, to reduce the computational complexity, has been topic of research before. The ODC of a variable v_{ij} can be derived easily from the ODC of variable v_j and the local ODC at node n_i

$$ODC_{ij}^k = ODC_j^k + ODC_{ij}^j. \tag{1}$$

However, deriving the ODC of a variable v_i from the ODC's of its fanout variables v_{ij} is much more difficult if the number of fanouts is more than one. Use of the chain rule [5] has been proposed by [1], but it becomes already very expensive for nodes with only two fanouts.

Suppose

$$FO(i) = \{ij_0, ij_1\}$$

then

$$ODC_i^k = ODC_{ij_0}^k \odot ODC_{ij_1}^k \odot ODC_{ij_1}^k|_{v_{ij_0}} \odot ODC_{ij_1}^k|_{\overline{v}_{ij_0}}. \tag{2}$$

In [6] a new method was presented which needs substantially fewer XOR operations and no higher order derivatives.

Suppose

$$FO(i) = \{ij_0, ij_1, \cdots, ij_n\}$$

then

$$ODC_i^k = \bigodot_{m=0}^{n} ODC_{ij_m}^k|_{\overline{v}_{ij_0},\cdots,\overline{v}_{ij_{m-1}},v_{ij_{m+1}},\cdots,v_{ij_n}}. \quad (3)$$

This formula still results in such a complex ODC that in practice (less complex) approximations of the ODC must be used.

A method which does not need the AND operation over all outputs (see Definition 8) is introduced in [8].

Let

$$ODC_i = \sum_{j \in O \setminus \{i\}} v_j \oplus g_j \quad \text{for all } i \in O$$

where g_j represents the global function of output n_j in terms of the primary inputs.

Suppose

$$FO(i) = \{ij_0, ij_1, \cdots, ij_n\}$$

and

$$\begin{aligned} \mathcal{O}_i^0 &= 1 \\ \mathcal{O}_i^m &= \left(v_{j_m} \odot \mathcal{O}_i^{m-1}\right)\overline{ODC}_{j_m}|_{v_{j_{m+1}},\cdots,v_{j_n}} \\ &\quad + \mathcal{O}_i^{m-1} ODC_{j_m}|_{v_{j_{m+1}},\cdots,v_{j_n}} \end{aligned}$$

then

$$ODC_i = \mathcal{O}_i^n\big|_{v_i} \odot \mathcal{O}_i^n\big|_{\overline{v}_i}. \quad (4)$$

Although [8] does not need the AND operator over all primary outputs, the operations needed per fanout are more complex. The paper does not report experimental results.

The method presented in this paper makes it possible to calculate the ODC without the use of any (explicit) X(N)OR operations and also without the AND operation over all outputs. The resulting network of factored forms is substantially less complex.

IV. Polarized Observability Don't Cares

To calculate the ODC in a new and more efficient way we first have to introduce a new operator: *polarization*.

Definition 9: The polarization operator applied to variable v introduces a new variable $\tilde{v}$ such that $\tilde{\tilde{v}} = v$.

Definition 10: The polarized Boolean function $\tilde{f}_p$ is associated with literal $\tilde{v}_p$ and is defined as

$$\tilde{f}_p(v_0, \cdots, v_n) = f_p(\tilde{v}_0, \cdots, \tilde{v}_n).$$

Polarization of a variable can be seen as the "twinning" of a variable. Note that the twin of a twin of a variable is the variable itself. Consistently, a polarized function (network) can be seen as a twin copy of the original function (network), using the twin copies of its variables. The only difference we will assume between the original and its twin is their behavior under the cofactor operator. Cofactoring a polarized variable will evaluate to the opposite constant value as its nonpolarized twin. So, we extend the definition of cofactoring to polarized Boolean functions.

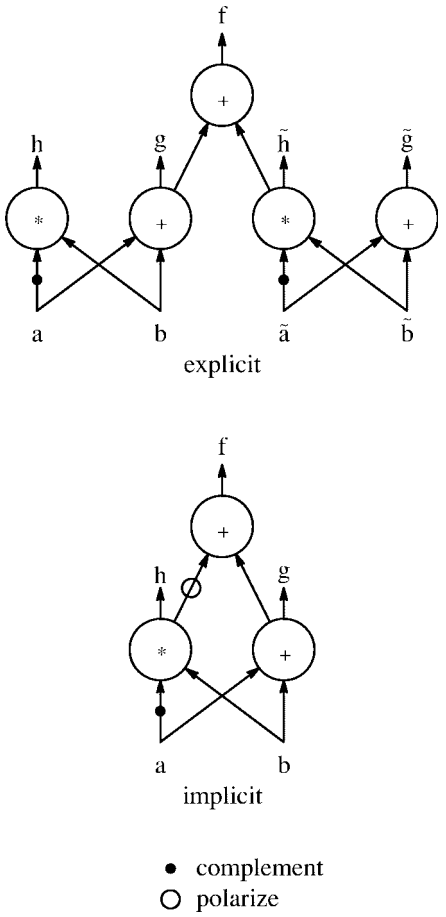

Fig. 2. Explicit and implicit twinning.

Definition 11:

$$\begin{aligned} f|_{v_p} &= f(v_p = 1, \tilde{v}_p = 0) \\ f|_{\overline{v}_p} &= f(v_p = 0, \tilde{v}_p = 1). \end{aligned}$$

Instead of explicitly copying a network to obtain its twin, we can also model polarization as an edge property (see Fig. 2). So, if we want to calculate $f|_{v_p}$ in a multiple-level network, any variable v_p which is on a path from f to v with an even number of polarizations has to be substituted with constant one. Any variable v_p on such a path with an odd number of polarizations has to be substituted with constant zero.

The following property follows from these definitions: $f|_{v_i} = \widetilde{(\tilde{f}|_{\overline{v}_i})}$, while $f|_{v_i} = \overline{(\overline{f}|_{v_i})}$.

Furthermore, we need a way to remove all polarizations from a network.

Definition 12: Remove polarity

$$\sharp f_i = f_i(\tilde{v}_p = v_p) \quad \text{for all } p \in TFI(f_i).$$

Polarization is used to mark factors in a network, such that they will cofactor to the opposite value as would be the case normally. For example, if we have $f = g + \tilde{h}$ (see Fig. 2) with g and h not polarized, then $f|_a = g|_a + \tilde{h}|_a = g(a = 1) + \tilde{h}(a = 0)$ and $\sharp(f|_a) = \sharp(g(a = 1) + \tilde{h}(a = 0)) = g(a = 1) + h(a = 0) = g|_a + h|_{\overline{a}}$.

Using these definitions we can rewrite the definition of the ODC (Definitions 7 and 8) as follows:

$$ODC_p = \sharp \prod_{k \in O} (f_k \odot \tilde{f}_k)|_{\overline{v}_p}. \tag{5}$$

Now we will define the PODC. It is defined recursively, so it can be constructed for all nodes by traversing the network from the outputs to the inputs in topological order. It will be proved that if the PODC is cofactored and the polarization is removed, then it will be equal to the ODC.

First we define the PODC of a primary output (in the case that there are no external don't cares specified).

Definition 13:

$$PODC_i = 0 \quad \text{for all } i \in O.$$

The PODC of a connection c_{ij} can be derived from the PODC of node n_j using the following definition.

Definition 14:

$$PODC_{ij} = \begin{cases} PODC_j + f_j & \text{if } OP(j) = + \\ & \text{and } INV(ij) = \text{FALSE} \\ PODC_j + \tilde{\overline{f}}_j & \text{if } OP(j) = * \\ & \text{and } INV(ij) = \text{FALSE} \\ \widetilde{PODC}_j + \tilde{f}_j & \text{if } OP(j) = + \\ & \text{and } INV(ij) = \text{TRUE} \\ \widetilde{PODC}_j + \overline{f}_j & \text{if } OP(j) = * \\ & \text{and } INV(ij) = \text{TRUE}. \end{cases}$$

The PODC of a node n_i can be derived from the PODC of all connections c_{ij} using the following definition.

Definition 15:

$$PODC_i = \prod_{ij \in FO(i)} PODC_{ij}.$$

If we cofactor the PODC and remove polarization we get the ODC.

Theorem 1:

$$ODC_p = \sharp(PODC_p|_{\overline{v}_p}).$$

So using Definitions 14 and 15 and Theorem 1 we can create the ODC of any node or connection in the network. Fig. 3 shows how the PODC network is constructed for a sample logic network. Note that in the method described in [6], see (3), XNOR operations are needed at multiple-fanout nodes, here we only need simple AND operations.

Before proving Theorem 1, we will first look what happens if we apply this theorem to Definition 14. For example, in the case of an AND gate, it is easy to prove that: $ODC_{ij} = \sharp(PODC_{ij}|_{\overline{v}_{ij}}) = \sharp(PODC_j|_{\overline{v}_{ij}} + \tilde{\overline{f}}_j|_{\overline{v}_{ij}}) = \sharp(PODC_j|_{\overline{v}_j}) + \sum_{k \neq i} \overline{v}_{kj} = ODC_j + \sum_{k \neq i} \overline{v}_{kj}$ [see (1)].

However, proving that Theorem 1 also holds for Definition 15 is not as easy: $ODC_i = \sharp(PODC_i|_{\overline{v}_i}) = \prod_{ij \in FO(i)} PODC_{ij} = \cdots$ [see (2)–(4)]. As a matter of fact, it is impossible to prove this by just assuming that $\sharp(PODC_{ij}|_{\overline{v}_{ij}}) = ODC_{ij}$, without taking into account the polarized information of $PODC_{ij}$.

In order to prove Theorem 1 we define a property which holds for every cut set through the network. This cut set can contain node variables as well as connection variables.

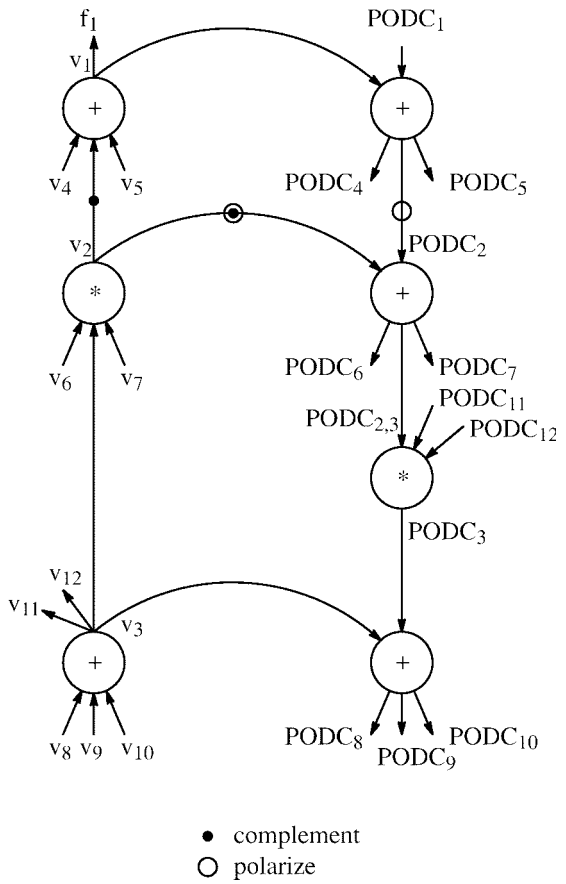

Fig. 3. PODC network construction.

Definition 16: A cut set C is defined as a set of indexes and index pairs such that on every path from any primary output to any primary input there is exactly one node or connection which appears in C.

To reason about cutsets, we will define $PEQV_C$, which can be understood best as a "polarized characteristic function."

Definition 17:

$$PEQV_C = \prod_{q \in C} \left[\left(f_q + \tilde{\overline{f}}_q + PODC_q \right) \left(\tilde{f}_q + \overline{f}_q + \widetilde{PODC}_q \right) \right].$$

Any cut set divides the network into two parts. We will use the $PEQV_C$ to prove Theorem 1. First we will prove that $PEQV_C$ is invariant for any cutset C. From this property we will prove Theorem 1.

Lemma 1:

$$PEQV_C = \prod_{i \in O} \left[\left(f_i + \tilde{\overline{f}}_i \right) \left(\tilde{f}_i + \overline{f}_i \right) \right].$$

Proof: By Definition 13, if $C = O$, the lemma holds. ■

So $PEQV_O$ is just the XNOR of the primary outputs of the original network and its twin.

Now we will use induction to prove that the $PEQV_C$ does not change if the cut set is moved toward the primary inputs.

Step 1: Move cut set over a node (see Fig. 4). Suppose Lemma 1 holds for a given cut set C. Now consider another cut set $C' = FI(j) \cup C \setminus \{j\}$.

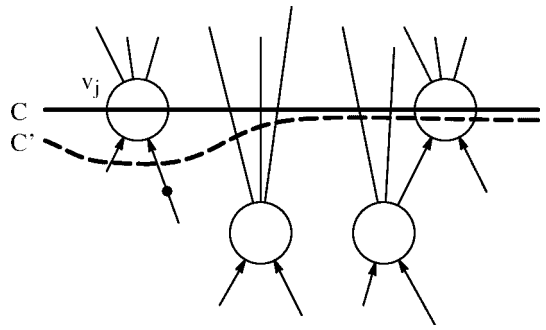

Fig. 4. Moving cut set over a node.

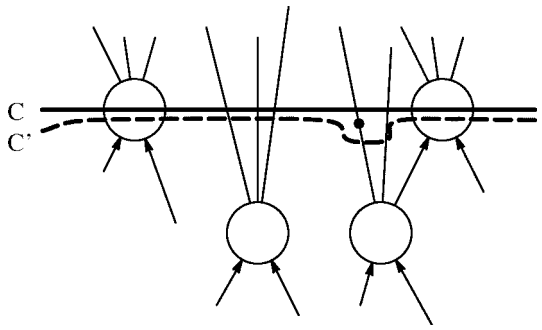

Fig. 5. Moving cut set over an inverter.

The cut set C' does not yet cross any possible inverters on connection c_{ij}.

Step 1a: Assume $OP(j) = +$.

According to Definition 14: $PODC_{ij} = PODC_j + f_j$. So:

$$\begin{aligned} &PEQV_{C'} \\ &= \prod_{ij \in FI(j)} \left[\left(f_{ij} + \tilde{\overline{f}}_{ij} + PODC_{ij} \right) \right. \\ &\quad \left. * \left(\tilde{f}_{ij} + \overline{f}_{ij} + \widetilde{PODC}_{ij} \right) \right] \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &\qquad \text{apply Definition 14} \\ &= \prod_{ij \in FI(j)} \left[\left(f_j + \tilde{\overline{f}}_{ij} + PODC_j \right) \right. \\ &\quad \left. * \left(\tilde{f}_j + \overline{f}_{ij} + \widetilde{PODC}_j \right) \right] \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &= \left(f_j + \prod_{ij \in FI(j)} \tilde{\overline{f}}_{ij} + PODC_j \right) \\ &\quad * \left(\tilde{f}_j + \prod_{ij \in FI(j)} \overline{f}_{ij} + \widetilde{PODC}_j \right) \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &\qquad \text{as } f_j = \sum_{ij} f_{ij} \text{ and therefore } \overline{f}_j = \prod_{ij} \overline{f}_{ij} \\ &= \left(f_j + \tilde{\overline{f}}_j + PODC_j \right) \left(\tilde{f}_j + \overline{f}_j + \widetilde{PODC}_j \right) \\ &\quad * \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &= PEQV_C. \end{aligned}$$

Step 1b: Assume $OP(j) = *$.

According to Definition 14: $PODC_{ij} = PODC_j + \tilde{\overline{f}}_j$. So

$$\begin{aligned} &PEQV_{C'} \\ &= \prod_{ij \in FI(j)} \left[\left(f_{ij} + \tilde{\overline{f}}_{ij} + PODC_{ij} \right) \right. \\ &\quad \left. * \left(\tilde{f}_{ij} + \overline{f}_{ij} + \widetilde{PODC}_{ij} \right) \right] \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &\qquad \text{apply Definition 14} \\ &= \prod_{ij \in FI(j)} \left[\left(f_{ij} + \tilde{\overline{f}}_j + PODC_j \right) \right. \\ &\quad \left. * \left(\tilde{f}_{ij} + \overline{f}_j + \widetilde{PODC}_j \right) \right] \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &= \left(\prod_{ij \in FI(j)} f_{ij} + \tilde{\overline{f}}_j + PODC_j \right) \\ &\quad * \left(\prod_{ij \in FI(j)} \tilde{f}_{ij} + \overline{f}_j + \widetilde{PODC}_j \right) \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &\qquad \text{as } f_j = \prod_{ij} f_{ij} \\ &= \left(f_j + \tilde{\overline{f}}_j + PODC_j \right) \left(\tilde{f}_j + \overline{f}_j + \widetilde{PODC}_j \right) \\ &\quad * \prod_{q \in C' \setminus FI(j)} (\cdots) \\ &= PEQV_C. \end{aligned}$$

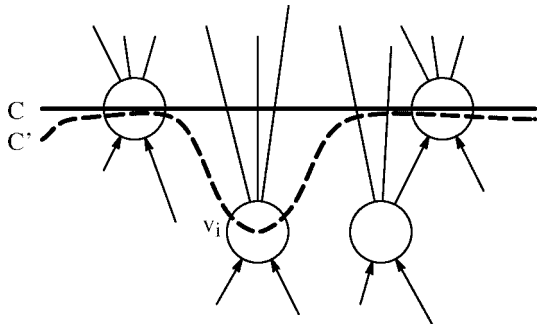

Fig. 6. Moving cut set over a fanout connection.

Step 2: Move cut set over an inverter (see Fig. 5).

Let ij_0 be in C, and ij_1 in C'. According to Definition 14: $PODC_{ij_1} = \widetilde{PODC}_{ij_0}$. Since

$$\begin{aligned} &\left(f_{ij_1} + \tilde{\overline{f}}_{ij_1} + PODC_{ij_1} \right) \left(\tilde{f}_{ij_1} + \overline{f}_{ij_1} + \widetilde{PODC}_{ij_1} \right) \\ &= \left(\tilde{f}_{ij_0} + \overline{f}_{ij_0} + \widetilde{PODC}_{ij_0} \right) \left(f_{ij_0} + \tilde{\overline{f}}_{ij_0} + PODC_{ij_0} \right) \end{aligned}$$

again we see that $PEQV_{C'} = PEQV_C$.

Step 3: Move cut set over a fanout connection (see Fig. 6).

Suppose Lemma 1 holds for a given cut set C. Now consider another cut set: $C' = \{i\} \cup C \setminus FO(i)$. According to Definition 15: $PODC_i =$

$\prod_{ij\in FO(i)} PODC_{ij}$. The following is now true:

$$
\begin{aligned}
&PEQV_{C'}\\
&= \left(f_i + \tilde{\overline{f}}_i + \prod_{ij\in FO(i)} PODC_{ij}\right)\\
&\quad * \left(\tilde{f}_i + \overline{f}_i + \prod_{ij\in FO(i)} \widetilde{PODC}_{ij}\right) \prod_{q\in C'\setminus\{i\}} (\cdots)\\
&\qquad \text{bring first two terms into product}\\
&= \prod_{ij\in FO(i)} \left[\left(f_i + \tilde{\overline{f}}_i + PODC_{ij}\right)\right.\\
&\quad \left.*\left(\tilde{f}_i + \overline{f}_i + \widetilde{PODC}_{ij}\right)\right] \prod_{q\in C'\setminus\{i\}} (\cdots)\\
&\qquad \text{as we do not cross inverters, } f_i = f_{ij}\\
&= \prod_{ij\in FO(i)} \left[\left(f_{ij} + \tilde{\overline{f}}_{ij} + PODC_{ij}\right)\right.\\
&\quad \left.* \left(\tilde{f}_{ij} + \overline{f}_{ij} + \widetilde{PODC}_{ij}\right)\right] \prod_{q\in C'\setminus\{i\}} (\cdots)\\
&= PEQV_C.
\end{aligned}
$$

Using these steps we can obtain any cut set C through the network. ■

Proof of Theorem 1: According to Lemma 1 we know that the $PEQV_C$ remains constant for any cut set. For the initial cut set (through all primary outputs) we have

$$
\begin{aligned}
\sharp(PEQV_C|_{\overline{v}_p}) &= \sharp\left[\prod_{i\in O}(f_i + \tilde{\overline{f}}_i)(\tilde{f}_i + \overline{f}_i)\right]\Bigg|_{\overline{v}_p}\\
&= \prod_{i\in O}(f_i|_{v_p} \odot f_i|_{\overline{v}_p})\\
&= ODC_p.
\end{aligned}
$$

For any cut set through variable v_p we know that all other f_q in the cut set do not depend on variable v_p

$$
\begin{aligned}
&\sharp(PEQV_C|_{\overline{v}_p})\\
&= \sharp\left[\prod_{q\in C}\left(f_q + \tilde{\overline{f}}_q + PODC_q\right)\left(\tilde{f}_q + \overline{f}_q + \widetilde{PODC}_q\right)\right]\Bigg|_{\overline{v}_p}\\
&= \sharp\left[\prod_{q\in C\setminus\{p\}}((\cdots)(\cdots))\right]\Bigg|_{\overline{v}_p}\\
&\quad * \sharp\left[\left(f_p + \tilde{\overline{f}}_p + PODC_p\right)\left(\tilde{f}_p + \overline{f}_p + \widetilde{PODC}_p\right)\right]\Big|_{\overline{v}_p}\\
&\qquad \text{first term does not depend on } v_p, \text{ bring in } \sharp\\
&= \prod_{q\in C\setminus\{p\}}\left((f_q + \overline{f}_q + PODC_q)(f_q + \overline{f}_q + PODC_q)\right)\\
&\quad * \sharp\left[\left(f_p + \tilde{\overline{f}}_p + PODC_p\right)\left(\tilde{f}_p + \overline{f}_p + \widetilde{PODC}_p\right)\right]\Big|_{\overline{v}_p}\\
&= 1 * \sharp\left[\left(f_p + \tilde{\overline{f}}_p + PODC_p\right)\left(\tilde{f}_p + \overline{f}_p + \widetilde{PODC}_p\right)\right]\Big|_{\overline{v}_p}\\
&\qquad \text{take cofactor of second term}\\
&= \sharp[(0 + 0 + PODC_p|_{\overline{v}_p})(1 + 1 + \cdots)]\\
&= \sharp(PODC_p|_{\overline{v}_p}).
\end{aligned}
$$

So, $ODC_p = \sharp(PODC_p|_{\overline{v}_p})$. ■

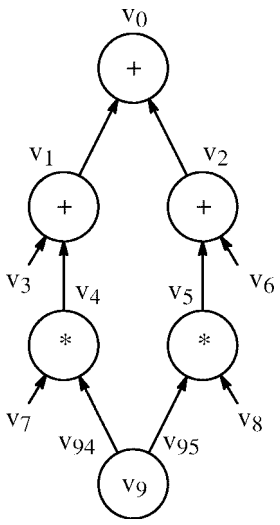

Fig. 7. Network for Example 1.

From this proof it can be derived that it is also possible to perform the cofactoring operations (to variable $\overline{v}_p$) already in Definitions 14 and 15, and change Theorem 1 into: $ODC_p = \sharp PODC_p$.

Since the PODC's on a cut set contain all the information needed to derive the ODC of any node in the input part of the cutset, it is obvious that the PODC's of all primary inputs of a given circuit can be handed over as (polarized) EDC to a feeding network. This then represents the complete ODC of the external circuit, and from it the Boolean relation for the equivalence classes [6] can be derived directly $EQV_{v_p^r,\ldots,v_q^r} = \sharp PEQV_C(\tilde{v}_p = v_p^r, \cdots, \tilde{v}_q = v_q^r)$.

V. Examples

In the following examples we will show how the different methods (traditional, Damiani, and polarized) compute the ODC. With "traditional" we refer to the method based on the definition of the ODC (Definitions 7 and 8). In all methods only constant propagation is used to obtain the final results. Example 3 also shows Savoj's method.

A. Example 1

See Fig. 7.

Traditional:

$$
\begin{aligned}
ODC_9 &= f_0(v_9 = 0) \odot f_0(v_9 = 1)\\
&= (v_3 + v_6) \odot (v_3 + v_7 + v_8 + v_6).
\end{aligned}
$$

Damiani:

$$
\begin{aligned}
ODC_9 &= ODC_{94}|_{v_{95}} \odot ODC_{95}|_{\overline{v}_{94}}\\
&= (v_2 + v_3 + \overline{v}_7)|_{v_{95}} \odot (v_1 + v_6 + \overline{v}_8)|_{\overline{v}_{94}}\\
&= (v_6 + v_8 + v_3 + \overline{v}_7) \odot (v_3 + v_6 + \overline{v}_8).
\end{aligned}
$$

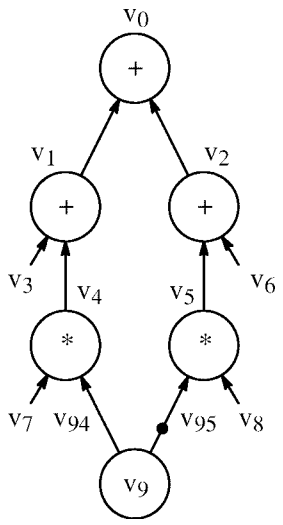

Fig. 8. Network for Example 2.

Polarized:

$$\begin{aligned} ODC_9 &= \sharp[(PODC_{94} * PODC_{95})|_{\overline{v}_9}] \\ &= \sharp\Big[\big((v_0 + v_1 + \tilde{\overline{v}}_4)(v_0 + v_2 + \tilde{\overline{v}}_5)\big)\Big|_{\overline{v}_9}\Big] \\ &= (v_3 + v_6 + \overline{v}_7)(v_3 + v_6 + \overline{v}_8). \end{aligned}$$

B. Example 2

See Fig. 8.

Traditional:

$$\begin{aligned} ODC_9 &= f_0(v_9 = 0) \odot f_0(v_9 = 1) \\ &= (v_3 + v_6 + v_8) \odot (v_3 + v_7 + v_6). \end{aligned}$$

Damiani:

$$\begin{aligned} ODC_9 &= ODC_{94}|_{v_{95}} \odot ODC_{95}|_{\overline{v}_{94}} \\ &= (v_2 + v_3 + \overline{v}_7)|_{v_{95}} \odot (v_1 + v_6 + \overline{v}_8)|_{\overline{v}_{94}} \\ &= (v_3 + v_6 + \overline{v}_7) \odot (v_3 + v_6 + \overline{v}_8). \end{aligned}$$

Polarized:

$$\begin{aligned} ODC_9 &= \sharp[(PODC_{94} * PODC_{95})|_{\overline{v}_9}] \\ &= \sharp\Big[\big((v_0 + v_1 + \tilde{\overline{v}}_4)(v_0 + v_2 + \tilde{\overline{v}}_5)\big)\Big|_{\overline{v}_9}\Big] \\ &= (v_3 + v_6 + v_8 + \overline{v}_7)(v_3 + v_6 + v_7 + \overline{v}_8). \end{aligned}$$

C. Example 3

Example taken from [6]; see Fig. 9.

Traditional:

$$\begin{aligned} ODC_6 &= \prod_{i=1}^{2} f_i|_{v_6} \odot f_i|_{\overline{v}_6} \\ &= ((v_5 v_7) \odot 1)((v_5 + v_7) \odot 1) \\ &= (v_5 v_7)(v_5 + v_7). \end{aligned}$$

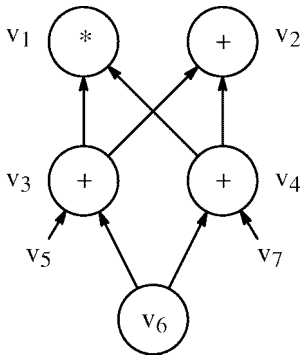

Fig. 9. Network for Example 3.

Damiani:

$$\begin{aligned} ODC_6 &= \prod_{i=1}^{2} ODC^i_{63}|_{v_{64}} \odot ODC^i_{64}|_{\overline{v}_{63}} \\ &= ((\overline{v}_4 + v_5)|_{v_{64}} \odot (\overline{v}_3 + v_7)|_{\overline{v}_{63}}) \\ &\quad * ((v_4 + v_5)|_{v_{64}} \odot (v_3 + v_7)|_{\overline{v}_{63}}) \\ &= (v_5 \odot (\overline{v}_5 + v_7))(1 \odot (v_5 + v_7)) \\ &= (v_5 \odot (\overline{v}_5 + v_7))(v_5 + v_7). \end{aligned}$$

Savoj:

given

$$\begin{aligned} ODC_3 &= v_4 \overline{g}_2 + \overline{v}_4 g_1 \\ ODC_4 &= v_3 \overline{g}_2 + \overline{v}_3 g_1 \end{aligned}$$

then

$$\begin{aligned} \mathcal{O}^1_6 &= v_3 + ODC_3|_{v_4} = v_3 + \overline{g}_2 \\ \mathcal{O}^2_6 &= \big(v_4 \odot \mathcal{O}^1_6\big)\overline{ODC}_4 + \mathcal{O}^1_6 ODC_4 \\ ODC_6 &= \mathcal{O}^2_6\big|_{v_6} \odot \mathcal{O}^2_6\big|_{\overline{v}_6}. \end{aligned}$$

Polarized:

$$\begin{aligned} ODC_6 &= \sharp[(PODC_{63} PODC_{64})|_{\overline{v}_6}] \\ &= \sharp\Big[\big((\tilde{\overline{v}}_1 v_2 + v_3)(\tilde{\overline{v}}_1 v_2 + v_4)\big)\Big|_{\overline{v}_6}\Big] \\ &= v_5 v_7. \end{aligned}$$

VI. Results and Conclusions

The described method has been implemented to generate the ODC's of all multiple-fanout nodes in a network. The resulting ODC's are created as a network of factored forms, with no optimizations except for constant propagation during cofactoring. The algorithm was tested on the complete MCNC benchmark set for multiple-level combinational networks. The results in Table I are from the circuits which contain initially more than 200 edges in the network of factored forms and are obtained on a HP 9000/755/99 (∼120 MIPS) with 256 MB of memory. Table I also gives the result for creation of the ODC's using Definitions 7 and 8. Note that we have only created the network for ODC's for multiple-fanout points in the network, as all others can be obtained trivially.

The number of nodes in Tables I and II refer to circuits composed of AND's and OR's only. Inverters are not counted, as they are annotated as edge properties.

TABLE I
CPU TIME, NUMBER OF NODES, AND NUMBER OF EDGES FOR THE ODC REPRESENTATION OF FACTORED FORMS

			traditional			PODC		
circuit	#nodes	#edges	time (s)	#nodes	#edges	time (s)	#nodes	#edges
9symml	162	387	0.0	1301	**3288**	0.1	2555	6126
C1355	515	992	7.8	303013	712480	4.7	169629	**386632**
C1908	474	1058	4.4	149109	418547	3.6	91497	**312625**
C2670	1008	1651	2.7	67731	158063	0.7	23751	**55691**
C3540	1001	2216	11.9	353483	840320	9.2	207267	**496674**
C432	192	368	0.9	43553	105815	0.2	8345	**20700**
C499	411	784	3.9	167341	405576	1.6	75149	**166952**
C5315	1617	3516	6.3	196252	483553	2.5	67767	**163393**
C6288	2400	4720	Out of memory			Out of Memory		
C7552	2355	4776	10.3	206709	531813	9.1	200483	**515761**
C880	354	640	0.5	22776	50629	0.4	19603	**42528**
alu2	302	694	0.4	19801	**47991**	0.6	26094	64601
alu4	607	1333	1.7	83168	**192220**	1.4	80098	196858
apex6	685	1216	0.6	11367	26014	0.2	6412	**13878**
apex7	230	414	0.1	6437	14651	0.1	2198	**4631**
b9	136	214	0.0	1219	2706	0.0	426	**977**
c8	170	326	0.0	1004	2234	0.0	731	**1645**
cht	234	391	0.0	1001	2087	0.0	773	**1334**
comp	152	250	0.1	4071	8815	0.1	2713	**5830**
count	146	238	0.1	4047	8892	0.0	2565	**4457**
des	3129	8291	8.2	130385	**582322**	10.3	122636	1129350
example2	305	496	0.3	6742	15912	0.1	1716	**3591**
f51m	153	324	0.0	567	**1237**	0.0	1216	2455
frg1	120	224	0.0	951	2444	0.0	968	**2410**
frg2	1151	2494	1.4	21728	53786	0.3	13004	**32928**
k2	326	2983	2.5	16303	**241694**	5.5	15641	498969
lal	130	257	0.0	1110	2652	0.0	721	**1654**
my_adder	241	416	0.2	12766	25892	0.1	4800	**9806**
pair	1538	2943	3.4	99135	209481	1.8	67462	**132303**
rot	610	1095	1.1	34775	85252	0.3	10549	**25113**
sct	102	210	0.0	653	1663	0.0	461	**1158**
term1	388	830	0.1	4125	**9792**	0.1	5170	11957
too_large	442	1564	0.2	5501	**31062**	0.4	7950	43111
ttt2	212	465	0.1	2035	4777	0.1	1900	**4374**
unreg	132	208	0.0	770	1598	0.0	322	**686**
vda	143	1426	0.5	8589	**77468**	0.9	7600	112252
x1	281	647	0.1	2843	8528	0.1	1422	**3676**
x3	845	1723	0.5	8848	20346	0.1	6620	**14860**
x4	451	892	0.3	7506	17870	0.1	4391	**10200**

TABLE II
NUMBER OF NODES AND EDGES FOR THE ORIGINAL AND THE PODC NETWORK

	original		PODC	
circuit	#nodes	#edges	#nodes	#edges
des	3129	8291	11353	69667
C7552	2355	4776	7369	22562
k2	326	2983	2249	7766

From Table I we can see that the PODC method results in a ODC circuit with fewer edges (= literals) in 29 out of 39 examples. The traditional method wins nine times, and both methods fail (run out of memory) for the multiplier circuit of C6288. Run times are within seconds for all examples.

The failure of C6288 is probably the result of the very high degree of reconvergence of the multiplier structure. The reason that the PODC method in some examples results in a larger ODC circuit lies in the fact that these examples contain nodes with very large fanout and with reconvergent paths which contain almost all local nodes.

We feel confident that the results of the PODC method could be further improved with the addition of some Boolean simplification during the building phase of the network. Some initial experiments with optimization after the building phase show a gain of at least a factor of two. The traditional method cannot be improved easily in this way, as it expresses the ODC basically in copies of the original network, cofactored once, with an XOR at the primary output. The original network should be considered optimized already.

We do not present comparisons with other methods, because most papers do not present results on ODC size at all, except for [6] which presents an average number of literals needed to represent the ODC sets, but it is not clear which ODC's were computed (all nodes, only multiple-fanout nodes or inputs nodes). It should, however, be clear that the presented method is computationally easier than [6] since the algorithm traverses the network in the same way, but operations at each step are simpler.

Table II shows the size of the PODC network itself of some of the largest results in Table I. It can be shown that the size of this network is linear in the size of the original network. This is a useful property since the PODC network can be used to provide the don't care information for a feeding network as individual ODC's or as a single Boolean relation.

REFERENCES

[1] K. A. Bartlett, R. K. Brayton, G. D. Hachtel, R. M. Jacoby, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Multi-level logic minimization using implicit don't cares," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, no. 6, pp. 723–739, June 1988.

[2] R. A. Bergamaschi, D. Brand, L. Stok, M. Berkelaar, and S. Prakash, "Efficient use of large don't cares in high-level and logic synthesis," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1995, pp. 272–278.

[3] R. K. Brayton and F. Somenzi, "An exact minimizer for Boolean relations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1989, pp. 316–319.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, no. 2, pp. 264–300, Feb. 1990.

[5] A. C. L. Chiang, I. S. Reed, and A. V. Banes, "Path sensitization, partial difference, and automated fault diagnosis," *IEEE Trans. Comput.*, pp. 189–195, Feb. 1972.

[6] M. Damiani and G. De Micheli, "Observability don't care sets and Boolean relations," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990, pp. 502–505.

[7] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney, "The transduction method—Design of logic networks based on permissible functions," *IEEE Trans. Comput.*, vol. 38, no. 10, pp. 1404–1424, Oct. 1989.

[8] H. Savoj and R. K. Brayton, "Observability relations and observability don't cares," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1991, pp. 518–521.

[9] H. Savoj, private communication, Apr. 1996.

Harm Arts was born on September 27, 1963 in Venraij, The Netherlands. He received the M.S. degree in electrical engineering in 1989 from the Eindhoven University of Technology.

In 1989, he joined the Design Automation Section of the Department of Electrical Engineering of the Eindhoven University of Technology as a Researcher. In 1997, he joined Ambit Design Systems, Santa Clara, CA.

Michel Berkelaar (M'97) was born on September 24, 1959, in Noordwijkerhout, The Netherlands. He received the M.S. degree in electrical engineering "cum laude" in 1987 from the Eindhoven University of Technology. In 1992, he received the Ph.D. degree for his work on logic synthesis.

In 1987, he joined the Design Automation Section of the Department of Electrical Engineering of the Eindhoven University of Technology as a Researcher. In 1992, he joined the permanent staff of the Design Automation Section. In 1994 and 1995, he spent a year as a Visiting Scientist at the IBM T. J. Watson Research Center.

Koen van Eijk (M'97) was born on September 19, 1970, in Hilvarenbeek, The Netherlands. He studied Information Engineering at the Eindhoven University of Technology, from which he graduated with honors on August 27, 1992. In 1997, he received the Ph.D. degree for his work on formal verification.

In 1992, he joined the Design Automation Section of the Department of Electrical Engineering of the Eindhoven University of Technology as a Researcher. In 1997, he joined the permanent staff of the Design Automation Section. His research interests include formal verification and synthesis of digital circuits.