Polarized Observability Don't Cares

Harm Arts, Michel Berkelaar and C.A.J. van Eijk
Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract

A new method is presented to compute the exact observability don't cares (ODC) for multilevel 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 ODCs 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 1 circuit in at most a few seconds.

1. Introduction

Computation of observability don't cares (ODC) plays a central role in the synthesis of Boolean networks. Together with the external don't cares (EDC) and the satisfiability don't cares (SDC) they represent the freedom one has to optimize the network. Especially the computation of the ODC 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 ODCs 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 ODCs 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 ODCs. Although the method does not need to calculate the ODC separately for each primary output, 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 ODCs could be usually 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 ODCs 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 ODCs 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 PODCs without the explicit use of "xor" or "xnor" operations. Another contribution is the large results table. All above mentioned previously published papers are either completely theoretical or show very few results, which leaves no room for comparison of different methods. Our result 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 this method is that it makes the use of EDCs very simple. The PODCs calculated at the input of a network can be handed over as EDCs to a feeding network directly, representing the complete EDC. These PODCs also directly imply the Boolean relations for the equivalence classes [3, 6].

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

The method is implemented and tested on the entire set of MCNC combinational multilevel benchmarks.

2. Definitions and notation

ODCs are commonly calculated using a Boolean network, see [1], to model a combinational circuit. In a

ICCAD '96
1063-6757/96 $5.00

Boolean network each node is associated with a Boolean expression (eg. a SOC) in terms of its fanin nodes (or fanin edges). We will however 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 poses no limitation since any Boolean expression itself is 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 ODCs.

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

Definition 2.1:

$OP : N \setminus I \rightarrow \{"+", "*"\}$

$INV : C \rightarrow \{0, 1\}$

OP(i) returns the operation represented by node n_i.
INV(ij) returns 1 if connection c_{ij} has an inverter property.

Variable v_i denotes the value 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, we also introduce variables for all connections: v_{ij} denotes the value 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.

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

$$\mathsf{f}_j = \begin{cases} \sum_j v_{ij} & \text{if } OP(j) = "+" \\ \prod_j v_{ij} & \text{if } OP(j) = "*" \end{cases} \qquad \mathsf{f}_{ij} = \begin{cases} v_i & \text{if } INV(ij) = 0 \\ \bar{v}_i & \text{if } INV(ij) = 1 \end{cases}$$

Definition 2.2:

The fanin of a node: $FI(j) = \{ij \mid ij \in C\}$
The fanin of a connection: $FI(ij) = \{i\}$
The fanout of a node: $FO(i) = \{ij \mid ij \in C\}$
The fanout of a connection: $FO(ij) = \{j\}$
The transitive fanin: $TFI(p) = \bigcup_{q \in FI(p)} TFI(q) \cup FI(p)$
The transitive fanout: $TFO(p) = \bigcup_{q \in FO(p)} TFO(q) \cup FO(p)$

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

Definition 2.3: Cofactoring: $\mathsf{f}|_{v_p} = \mathsf{f}(v_p = 1)$
$\mathsf{f}|_{\bar{v}_p} = \mathsf{f}(v_p = 0)$

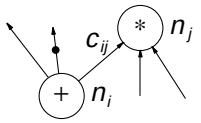

Figure 1. Nodes and connections

3. Observability Don't Care

The Observability Don't Care (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 3.1: The ODC of variable v_p at node n_k:
$ODC_p^k = \mathsf{f}_k|_{v_p} \odot \mathsf{f}_k|_{\bar{v}_p}$

Where $\odot$ is the exclusive-nor 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 3.2: The ODC of variable v_p at all primary outputs: $ODC_p = \prod_{k \in O} ODC_p^k$

Creating the ODC, using these definitions, as a network of factored forms is relatively simple, but the resulting network turns out to be very complex. As a result the calculation of the ODC in this way, by expressing it in sum-of-cubes or BDDs, in terms of primary inputs or local variables, is known to be very expensive [1].

Deriving the ODC of a node from the ODCs of its direct fanouts to reduce the 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: $ODC_{ij}^k = ODC_j^k + ODC_{ij}^i$ (3.1)
However, deriving the ODC of a variable v_i from the ODCs of its fanout variables v_{ij} is much more difficult if the degree of the fanout 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:
$ODC_i^k = ODC_{ij_0}^k \odot ODC_{ij_1}^k \odot ODC_{ij_1}^k|_{v_{ij_0}} \odot ODC_{ij_1}^k|_{\bar{v}_{ij_0}}$ (3.2)
In [6] a new method was presented which needs substantially less exclusive-or operations and no higher order derivatives.
Suppose: $FO(i) = \{ij_0, ij_1, \ldots, ij_n\}$ then:
$ODC_i^k = \bigodot_{m=0}^{n} ODC_{ij_m}^k|_{\bar{v}_{ij_0},\ldots,\bar{v}_{ij_{m-1}},v_{ij_{m+1}},\ldots,v_{ij_n}}$ (3.3)
This formula still results in such a complex ODC that in practice (less complex) approximations of the ODC must be used.
[8] introduced a method which does not need the "and" operation over all outputs (see definition 3.2).
Suppose: $FO(i) = \{ij_0, ij_1, \ldots, ij_n\}$
let: $ODC_i = \sum_{j \in O \setminus \{i\}} v_j \oplus \mathsf{g}_j$ for all $i \in O$, and let:

$\theta_i^m = (v_{j_m} \odot \theta_i^{m-1})\overline{ODC}_{j_m}|_{v_{j_{m+1}},\ldots,v_{j_n}} + \theta_i^{m-1} ODC_m|_{v_{j_{m+1}},\ldots,v_{j_n}}$
with: $\theta_i^0 = 1$ then: $ODC_i = \theta_i^n|_{v_i} \odot \theta_i^n|_{\bar{v}_i}$ (3.4)
Where g_i represents the global function of output v_i in terms of the primary inputs.
Although [8] does not need the "and" operator over all primary outputs, the operations needed per fanout are more complex.

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

4. Polarized Observability Don't Care

To calculate the ODC in a new and more efficient way we will introduce a new operator called: *polarization*.
First we clarify the difference between a variable and a literal. With every node and connection we associated a variable v. Literals v and $\bar{v}$ are the algebraic representations of the variable resp. the complemented variable.

Analogous to the complement operator, we now introduce the polarization operator, just as a notation, without yet specifying the semantical meaning.

Definition 4.1: The polarization operator applied to variable v, introduces the literal $\tilde{v}$ such that $\tilde{\tilde{v}} = v$.

So now variable v implies 4 literals: v, $\bar{v}$, $\tilde{v}$, and $\tilde{\bar{v}}$.

Definition 4.2: The polarized Boolean function $\tilde{f}_p$ is associated with literal $\tilde{v}_p$ and is defined as:

$$\tilde{f}_j = \begin{cases} \sum_j \tilde{v}_{ij} & \text{if } OP(j) = "+" \\ \prod_j \tilde{v}_{ij} & \text{if } OP(j) = "*" \end{cases} \qquad \tilde{f}_{ij} = \begin{cases} \tilde{v}_i & \text{if } INV(ij) = 0 \\ \tilde{\bar{v}}_i & \text{if } INV(ij) = 1 \end{cases}$$

Polarization in Boolean networks can be labeled as an edge property. So an edge can have complementing and/or polarizing properties. The polarization operator can be shifted trough a node in almost the same way as the complement operator: For complementing the DeMorgan rule applies, however for polarization the operation on the node does not toggle. If for example: $f = g + \tilde{h}$, then $\bar{f} = \bar{g} * \tilde{\bar{h}}$, but $\tilde{f} = \tilde{g} + h$.

We extend the definition of cofactoring to polarized Boolean functions.

Definition 4.3: $f|_{v_p} = f(v_p = 1, \tilde{v}_p = 0)$
$f|_{\tilde{v}_p} = f(v_p = 0, \tilde{v}_p = 1)$

So if we want to calculate $f|_{v_p}$, any variable v_p which is on a path from f to v_p with an even number of polarizations, has to be substituted with constant 1, and any variable v_p which is on a path with an odd number of polarizations with constant 0

The following property follows from this definition: $f|_{v_i} = \left(\tilde{f}|_{\tilde{v}_i}\right)\tilde{}$, while: $f|_{v_i} = \overline{\left(\tilde{f}|_{v_i}\right)}$.

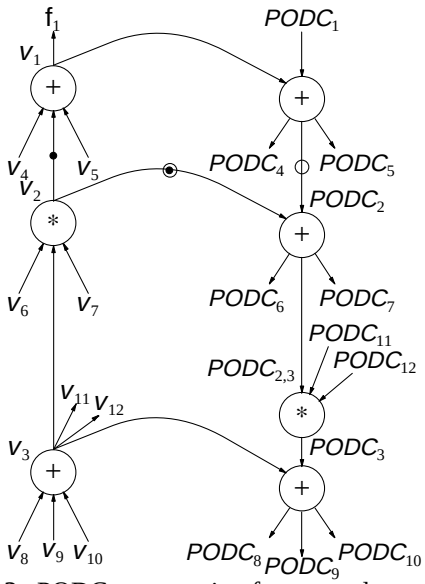

Figure 2. PODC construction for a sample network

Definition 4.4: $\#f_i$ removes all polarization from the *TFI(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: $f = g + \tilde{h}$, with g and h not polarized,
then: $f|_a = g|_a + \tilde{h}|_a = g(a{=}1) + \tilde{h}(\tilde{a}{=}0)$
and: $\#\left(f|_a\right) = \#\left(g(a = 1) + \tilde{h}(\tilde{a} = 0)\right)$
$= g(a = 1) + h(a = 0) = g|_a + h|_{\bar{a}}$

Now we will define the Polarized Observability Don't Care (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 proven that if the PODC is cofactored and the polarization is removed, 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 4.5: $PODC_i = 0$ 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 4.6:

$$PODC_{ij} = \begin{cases} PODC_j + f_j & \text{if } OP(j) = "+" \text{ and } INV(ij) = 0 \\ PODC_j + \tilde{\tilde{f}}_j & \text{if } OP(j) = "*" \text{ and } INV(ij) = 0 \\ \tilde{PODC}_j + \tilde{f}_j & \text{if } OP(j) = "+" \text{ and } INV(ij) = 1 \\ \tilde{PODC}_j + \bar{f}_j & \text{if } OP(j) = "*" \text{ and } INV(ij) = 1 \end{cases}$$

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

Definition 4.7: $PODC_i = \prod_{ij \in FO(i)} PODC_{ij}$

If we cofactor the PODC and remove polarization we get

the ODC.

Theorem 4.1: $ODC_p = \#\left(PODC_p|_{\bar{v}_p}\right)$

So using definitions 4.6 and 4.7 and theorem 4.1 we can create the ODC of any node or connection in the network. Figure 2 shows how the PODC is constructed for a sample network. Note that in the method described in [6], see equation 3.3, exclusive-nor operations are needed at multiple fanout nodes, here we need simple and-operations.

In order to prove theorem 4.1 we first define a property which holds for every cutset through the network. This cutset can contain node as well as connection variables.

Definition 4.8: A cutset C is defined as a set of indices 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.

Definition 4.9:

$$PODC_C = \prod_{q \in C}\left(\left(\mathsf{f}_q + \tilde{\bar{\mathsf{f}}}_q + PODC_q\right)\left(\tilde{\mathsf{f}}_q + \bar{\mathsf{f}}_q + \tilde{POD}C_q\right)\right)$$

Any cutset divides the network into two parts. We will refer to all indices and index-pairs which are in C or in their transitive fanin as $IP(C)$.

We will use the $PODC_C$ to prove theorem 4.1. First we will prove that the $PODC_C$ is invariant for any cutset C. And from this property we will prove theorem 4.1.

Lemma 4.1: $PODC_C = \prod_{i \in O}\left(\mathsf{f}_i + \tilde{\bar{\mathsf{f}}}_i\right)\left(\tilde{\mathsf{f}}_i + \bar{\mathsf{f}}_i\right)$

Proof: By definition 4.5, if $C = O$, the lemma holds. Now we will use induction to prove that the $PODC_C$ does not change if the cutset is moved towards the primary inputs.

Step 1: Move cutset over a node (see figure 3).

Suppose lemma 4.1 holds for a given cutset . Now consider another cutset $C' = FI(j) \cup C \backslash \{j\}$. The cutset C' does not yet cross any possible inverters on connection c_{ij}.

Step 1a: $OP(j) = "+"$

According to definition 4.6: $PODC_{ij} = PODC_j + \mathsf{f}_j$.

So: $PODC_{C'} =$

$$\prod_{ij \in FI(j)}\left(\left(\mathsf{f}_{ij} + \tilde{\bar{\mathsf{f}}}_{ij} + PODC_{ij}\right)\left(\tilde{\mathsf{f}}_{ij} + \bar{\mathsf{f}}_{ij} + \tilde{POD}C_{ij}\right)\right)\prod_{q \in C' \setminus FI(j)}(\cdots) =$$

$$\prod_{ij \in FI(j)}\left(\left(\mathsf{f}_j + \tilde{\bar{\mathsf{f}}}_{ij} + PODC_j\right)\left(\tilde{\mathsf{f}}_j + \bar{\mathsf{f}}_{ij} + \tilde{POD}C_j\right)\right)\prod_{q \in C \setminus \{j\}}(\cdots) =$$

$$\left(\mathsf{f}_j + \tilde{\bar{\mathsf{f}}}_j + PODC_j\right)\left(\tilde{\mathsf{f}}_j + \bar{\mathsf{f}}_j + \tilde{POD}C_j\right)\prod_{q \in C \setminus \{j\}}(\cdots) = PODC_C$$

Step 1b: $OP(j) = "*"$

According to definition 4.6: $PODC_{ij} = PODC_j + \tilde{\bar{\mathsf{f}}}_j$.

So: $PODC_{C'} =$

$$\prod_{ij \in FI(j)}\left(\left(\mathsf{f}_{ij} + \tilde{\bar{\mathsf{f}}}_{ij} + PODC_{ij}\right)\left(\tilde{\mathsf{f}}_{ij} + \bar{\mathsf{f}}_{ij} + \tilde{POD}C_{ij}\right)\right)\prod_{q \in C' \setminus FI(j)}(\cdots) =$$

$$\prod_{ij \in FI(j)}\left(\left(\mathsf{f}_{ij} + \tilde{\bar{\mathsf{f}}}_j + PODC_j\right)\left(\tilde{\mathsf{f}}_{ij} + \bar{\mathsf{f}}_j + \tilde{POD}C_j\right)\right)\prod_{q \in C \setminus \{j\}}(\cdots) =$$

$$\left(\mathsf{f}_j + \tilde{\bar{\mathsf{f}}}_j + PODC_j\right)\left(\tilde{\mathsf{f}}_j + \bar{\mathsf{f}}_j + \tilde{POD}C_j\right)\prod_{q \in C \setminus \{j\}}(\cdots) = PODC_C$$

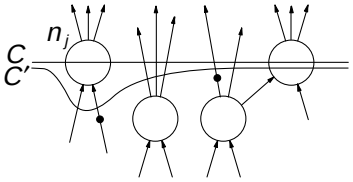

Figure 3. Move cutset over a node

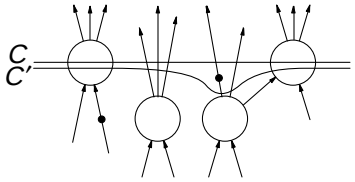

Figure 4. Move cutset over an inverter

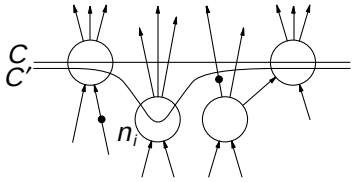

Figure 5. Move cutset over a fanout connection

Step 2: Move cutset over an inverter (see figure 4).

Let ij_0 be in C, and ij_1 in C'.

According to definition 4.6: $PODC_{ij_1} = \tilde{POD}C_{ij_0}$.

Since: $\left(\mathsf{f}_{ij_1} + \tilde{\bar{\mathsf{f}}}_{ij_1} + PODC_{ij_1}\right)\left(\tilde{\mathsf{f}}_{ij_1} + \bar{\mathsf{f}}_{ij_1} + \tilde{POD}C_{ij_1}\right)$

$= \left(\tilde{\mathsf{f}}_{ij_0} + \bar{\mathsf{f}}_{ij_0} + \tilde{POD}C_{ij_0}\right)\left(\mathsf{f}_{ij_0} + \tilde{\bar{\mathsf{f}}}_{ij_0} + PODC_{ij_0}\right)$,

$PODC_{C'} = PODC_C$.

Step 3: Move cutset over a fanout connection (see figure 5). Suppose lemma 4.1 holds for a given cutset C.

Now consider another cutset: $C' = \{i\} \cup C \backslash FO(i)$.

According to definition 4.7: $PODC_i = \prod_{ij \in FO(i)} PODC_{ij}$,

the following is true: $PODC_{C'} =$

$$\left(\mathsf{f}_i + \tilde{\bar{\mathsf{f}}}_i + \prod_{ij \in FO(i)} PODC_{ij}\right)\left(\tilde{\mathsf{f}}_i + \bar{\mathsf{f}}_i + \prod_{ij \in FO(i)} \tilde{POD}C_{ij}\right)\prod_{q \in C' \setminus \{i\}}(\cdots) =$$

$$\prod_{ij \in FO(i)}\left(\left(\mathsf{f}_i + \tilde{\bar{\mathsf{f}}}_i + PODC_{ij}\right)\left(\tilde{\mathsf{f}}_i + \bar{\mathsf{f}}_i + \tilde{POD}C_{ij}\right)\right)\prod_{q \in C' \setminus \{i\}}(\cdots) =$$

$$\prod_{ij \in FO(i)}\left(\left(\mathsf{f}_{ij} + \tilde{\bar{\mathsf{f}}}_{ij} + PODC_{ij}\right)\left(\tilde{\mathsf{f}}_{ij} + \bar{\mathsf{f}}_{ij} + \tilde{POD}C_{ij}\right)\right)\prod_{q \in C' \setminus \{i\}}(\cdots) =$$

$PODC_C$

Using these steps we can obtain any cutset through the network. □

Proof of theorem 4.1: According to lemma 4.1 we know that the $PODC_C$ remains constant for any cutset.

For the initial cutset (through all primary outputs) we have: $\#\left(PODC_C|_{\bar{v}_p}\right) = \#\left(\prod_{i \in O}\left(\mathsf{f}_i + \tilde{\bar{\mathsf{f}}}_i\right)\left(\tilde{\mathsf{f}}_i + \bar{\mathsf{f}}_i\right)|_{\bar{v}_p}\right)$

$= \prod_{i \in O} \mathsf{f}_i|_{v_p} \odot \mathsf{f}_i|_{\bar{v}_p} = ODC_p$

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

$\#\left(PODC_C|_{\bar{v}_p}\right) =$

$\#\left(\prod_{q \in C}\left(\left(\mathsf{f}_q + \tilde{\bar{\mathsf{f}}}_q + PODC_q\right)\left(\tilde{\mathsf{f}}_q + \bar{\mathsf{f}}_q + \tilde{POD}C_q\right)\right)|_{\bar{v}_p}\right) \quad =$

$\#\left(PODC_p\big|_{\bar{v}_p}\right)$
So: $ODC_p = \#\left(PODC_p\big|_{\bar{v}_p}\right)$ □

From this proof it can be derived that it is also possible to perform the cofactoring operations (to variable $\bar{v}_i$) already in definitions 4.6 and 4.7, and change theorem 4.1 into: $ODC_p = \#PODC_p$

Since the PODCs on a cutset contain all the information needed to derive the ODC of any node in the input part of the cutset, it is obvious that the PODCs 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:
$EQV_{v_p^r,\cdots,v_q^r} = \#PODC_C\left(\tilde{v}_p = v_p^r, \cdots, \tilde{v}_q = v_q^r\right)$.

5. 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 3.1 and 3.2). In all methods only constant propagation is used to obtain the final results. Example 3 also shows Savojs method.

Example 1 (see figure 6).
Traditional: $ODC_9 = \mathrm{f}_0(v_9 = 0) \odot \mathrm{f}_0(v_9 = 1) =$
$(v_3 + v_6) \odot (v_3 + v_7 + v_8 + v_6)$.
Damiani: $ODC_9 = ODC_{94}\big|_{v_{95}} \odot ODC_{95}\big|_{\bar{v}_{94}} =$
$(v_2 + v_3 + \bar{v}_7)\big|_{v_{95}} \odot (v_1 + v_6 + \bar{v}_8)\big|_{\bar{v}_{94}} =$
$(v_6 + v_8 + v_3 + \bar{v}_7) \odot (v_3 + v_6 + \bar{v}_8)$.
Polarized: $ODC_9 = \#\left((PODC_{94}.\, PODC_{95})\big|_{\bar{v}_9}\right) =$
$\left(((v_0 + v_1 + \tilde{\bar{v}}_4)(v_0 + v_2 + \tilde{\bar{v}}_5))\big|_{\bar{v}_9}\right) =$
$(v_3 + v_6 + \bar{v}_7)(v_3 + v_6 + \bar{v}_8)$.

Example 2 (see figure 7).
Traditional: $ODC_9 = \mathrm{f}_0(v_9 = 0) \odot \mathrm{f}_0(v_9 = 1) =$
$(v_3 + v_6 + v_8) \odot (v_3 + v_7 + v_6)$.
Damiani: $ODC_9 = ODC_{94}\big|_{v_{95}} \odot ODC_{95}\big|_{\bar{v}_{94}} =$
$(v_2 + v_3 + \bar{v}_7)\big|_{v_{95}} \odot (v_1 + v_6 + \bar{v}_8)\big|_{\bar{v}_{94}} =$
$(v_6 + v_3 + \bar{v}_7) \odot (v_3 + v_6 + \bar{v}_8)$.
Polarized: $ODC_9 = \#\left((PODC_{94}.\, PODC_{95})\big|_{\bar{v}_9}\right) =$
$\#\left(((v_0 + v_1 + \tilde{\bar{v}}_4)(\tilde{\bar{v}}_0 + \tilde{\bar{v}}_2 + \bar{v}_5))\big|_{\bar{v}_9}\right) =$
$(v_3 + v_6 + v_8 + \bar{v}_7)(v_3 + v_6 + v_7 + \bar{v}_8)$.

Example 3, from [6] (see figure 8).
Traditional: $ODC_6 = \prod_{i=1,2} \mathrm{f}_i\big|_{v_6} \odot \mathrm{f}_i\big|_{\bar{v}_6} =$
$((v_5.\, v_7) \odot 1)((v_5 + v_7) \odot 1) = (v_5.\, v_7)(v_5 + v_7)$.
Damiani: $ODC_6 = \prod_{i=1,2} ODC_{63}^i\big|_{v_{64}} \odot ODC_{64}^i\big|_{\bar{v}_{63}} =$
$((\bar{v}_4 + v_5)\big|_{v_{64}} \odot (\bar{v}_3 + v_7)\big|_{\bar{v}_{63}})((v_4 + v_5)\big|_{v_{64}} \odot (v_3 + v_7)\big|_{\bar{v}_{63}})$
$= (v_5 \odot (\bar{v}_5 + v_7))(1 \odot (v_5 + v_7)) =$
$(v_5 \odot (\bar{v}_5 + v_7))(v_5 + v_7)$.
Savoj (see equation 3.4): Given: $ODC_3 = v_4\bar{g}_2 + \bar{v}_4 g_1$
and $ODC_4 = v_3\bar{g}_2 + \bar{v}_3 g_1$

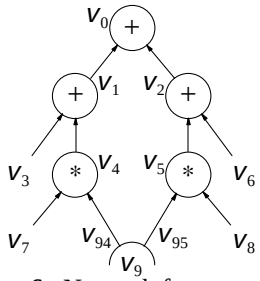

Figure 6. Network for example 1.

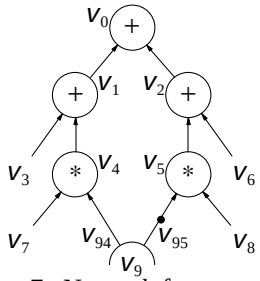

Figure 7. Network for example 2.

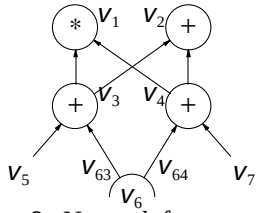

Figure 8. Network for example 3.

$\theta_6^1 = v_3 + ODC_3\big|_{v_4} = v_3 + \bar{g}_2$
$\theta_6^2 = (v_4 \odot \theta_6^1)\overline{ODC}_4 + \theta_6^1 ODC_4$
$ODC_6 = \theta_6^2\big|_{v_6} \odot \theta_6^2\big|_{\bar{v}_6}$.
Polarized: $ODC_6 = \#\left((PODC_{63}.\, PODC_{64})\big|_{\bar{v}_6}\right) =$
$\#\left(((\tilde{\bar{v}}_1.\, v_2 + v_3)(\tilde{\bar{v}}_1.\, v_2 + v_4))\big|_{\bar{v}_6}\right) = v_5.\, v_7$.

6. Results and conclusions

The described method has been implemented to generate the ODCs of all multiple fanout nodes in a network. The resulting ODCs 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 and ISCAS benchmark set for multilevel combinational networks. The results in table 1 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 (appr. 120 MIPS) with 256 MB of memory. The table also gives the result for creation of the ODCs using definitions 3.1 and 3.2.

From the table we can see that the PODC method results in a ODC circuit with fewer edges (= literals) in 26 out of 39 examples. The traditional method wins 12 times, and both methods fail (run out of memory) for the multiplier circuit 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 2. The traditional method cannot be improved that 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 ODCs were computed (all nodes, only multiple fanout nodes or inputs nodes). It should however be clear that the presented method is computationally simpler than [6] since the algorithm traverses the network in the same way, but operations at each step are simpler.

Table 2 shows the size of the PODC network itself for some of the largest results in table 1. 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 ODCs or as a single boolean relation.

TABLE 1. CPU time (s), number of nodes and edges for the ODC as a network of factored forms.

	traditional ODC			polarized ODC		
example	time	#nodes	#edges	time	#nodes	#edges
symml	0.0	1310	**3288**	0.0	2400	5597
C1355	8.2	303054	712480	4.2	162762	**351448**
C1908	4.7	149142	418547	2.3	84521	**215576**
C2670	2.7	67885	158063	0.5	23053	**51230**
C3540	12.1	353533	840320	6.9	174626	**397345**
C432	0.9	43589	105815	0.2	8723	**21113**
C499	3.8	167382	405576	1.5	78250	**173608**
C5315	6.3	196425	483553	2.5	78093	**179940**
C6288	Out of memory			Out of Memory		
C7552	10.2	206913	531813	7.9	189782	**445038**
C880	0.6	22836	50629	0.4	20451	**44806**
alu2	0.4	20238	**48875**	0.5	23528	55595
alu4	1.7	83370	195707	1.4	68726	**160628**
apex6	0.6	11502	26014	0.2	10747	**23779**
apex7	0.1	6516	14651	0.1	3892	**8805**
b9	0.0	1260	2706	0.0	853	**1737**
c8	0.0	1025	**2217**	0.0	1356	2831
cht	0.0	1033	**2035**	0.0	1334	2476
comp	0.1	4079	8791	0.1	3069	**6533**
count	0.1	4082	**8892**	0.1	4696	9828
des	8.3	130271	581992	3.9	127812	**377508**
example2	0.2	6827	15912	0.1	3510	**8069**
f51m	0.0	568	**1259**	0.0	1191	2582
frg1	0.0	969	**2402**	0.0	1450	3327
frg2	1.4	22044	54217	0.5	19555	**47208**
k2	2.6	16373	241702	1.8	8387	**166013**
lal	0.0	1136	**2652**	0.0	1197	2668
my_adder	0.3	21079	42332	0.1	8745	**18094**
pair	3.4	100285	211518	2.2	90887	**189074**
rot	1.0	34785	84759	0.4	15153	**35339**
sct	0.0	663	1647	0.0	688	**1638**
term1	0.1	4161	**9756**	0.1	6177	13681
too_large	0.2	5808	**32091**	0.2	8883	33885
ttt2	0.0	2008	**4661**	0.1	2581	5737
unreg	0.0	806	1598	0.0	790	**1534**
vda	0.5	8606	77468	0.4	3434	**47587**
x1	0.1	2952	8472	0.1	3028	**7950**
x3	0.5	8996	**20361**	0.2	9903	21750
x4	0.2	7657	17977	0.1	6990	**16057**

TABLE 2. Number of nodes and edges for the original and the PODC network.

	original		PODC	
example	#nodes	#edges	#nodes	#edges
C6288	2400	4720	6175	13183
C7552	2355	4776	5039	11968
des	3127	8287	6436	19627
k2	327	2989	790	6361

References

[1] K.A. BARTLETT, R. BRAYTON, G. HACHTEL, R. JACOBY, C. MORRISON, R. RUDELL, A. SANGIOVANNI-VINCENTELLI, AND A. WANG, ''Multilevel logic minimization using implicit don't cares'', *IEEE Transactions on Computer-Aided Design*, vol. 7, no. 6, pp. 723-740 (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 *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 272-278. (Nov. 1995).

[3] R.K. BRAYTON AND F. SOMENZI, ''An Exact Minimizer for Boolean Relations'' in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 316-319 (Nov. 1989).

[4] R.K. BRAYTON, G.D. HACHTEL, AND A.L. SANGIOVANNI-VINCENTELLI, ''Multilevel Logic Synthesis'' in *Proceedings of the IEEE*, vol. 78, 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 Transactions on Computers*, pp. 189-195 (Feb. 1972).

[6] M. DAMIANI AND G. DE MICHELI, ''Observability Don't Care Sets and Boolean Relations'' in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 502-505 (1990).

[7] S. MUROGA, Y. KAMBAYASHI, H.C. LAI, AND J.N. CULLINEY, ''The Transduction Method - Design of Logic Networks Based on Permissible Functions'', *IEEE Transactions on Computers*, vol. 38, no. 10, pp. 1404-1424 (October 1989).

[8] H. SAVOJ AND R.K. BRAYTON, ''Observability Relations and Observability Don't Cares'' in *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 518-521. (Nov. 1991).

[9] H. SAVOJ, Private communication (April 1996).