Don't Cares and Multi-Valued Logic Network Minimization *

Yunjian Jiang Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
{wjiang,brayton}@eecs.berkeley.edu

Abstract

We address optimizing multi-valued (MV) logic functions in a multi-level combinational logic network. Each node in the network, called an MV-node, has multi-valued inputs and single multi-valued output. The notion of don't cares used in binary logic is generalized to multi-valued logic. It contains two types of flexibility: incomplete specification and non-determinism. We generalize the computation of observability don't cares for a multi-valued function node. Methods are given to compute (a) the maximum set of observability don't cares, and (b) the compatible set of observability don't cares for each MV-node. We give a recursive image computation to transform the don't cares into the space of local inputs of the node to be minimized. The methods are applied to some experimental multi-valued networks, and demonstrate reduction in the size of the tables that represent multi-valued logic functions.

1 Introduction

Multi-valued (MV) logic synthesis is becoming important in various applications. It can be used in hardware synthesis as a higher level representation before the circuit is encoded into binary. There are optimization opportunities at this stage that cannot be discovered in a binary domain. It can also be used in control dominated software compilation, where control variables are computed and tested just as variables in multi-valued logic. This explores logical relations between variables to restructure the control flow of a program. Such optimizations are usually not considered by traditional software compilers.

We address the optimization problem of multi-valued input, multi-valued output logic functions in a combinational logic network. The notion of don't cares used in binary logic [1] is generalized to multi-valued logic. These contain two types of flexibility, incomplete specification and non-determinism. We define multi-valued don't cares and partial cares to capture this flexibility. Multi-valued functions combined with multi-valued partial cares are similar to Boolean relations in binary logic, where a set of compatible functions are to be explored for the optimization. The algorithms developed for Boolean relation minimization, [2] [3] [4] [5], can be applied in partial care minimization for a multi-valued function.

We give algorithms for generating a compatible set of don't cares for MV-nodes. Observability don't cares (ODC) for an MV-node is the set of minterms, which, when applied in the network, blocks the output values of that node, i.e. the primary outputs do not depend on the values of that node. Compatible ODC's (CODCs) are don't cares which do not depend on how the don't cares at other MV-nodes in the network are used. The methods to compute ODC's and CODC's are extended from the binary case [6] [7]. Observability partial cares (OPC) are the set of the minterms that blocks a subset of the output values, i.e. the primary output can not distinguish any pair of values in that subset. OPC's provide additional flexibility for the implementation of an MV-node. This is a generalization of observability Boolean relations for binary networks. In order to use the OPC's, a multi-valued relation minimizer needs to be applied. We give one method for generating OPC's.

In Section 2, we give the definition of multi-valued partial cares. Section 3 gives the algorithms to compute compatible don't cares for a multi-valued node in a combinational network, and Section 4 discusses image computation for a MV-network. We give some experimental results in Section 5 and conclude in Section 6.

2 Multi-valued Functions

Consider a multi-valued function with multiple multi-valued inputs and single multi-valued output. A multi-valued relation is, like a binary relation, a one to many mapping. Let $P_1 = \{0,1,...,|P_1|-1\}, P_2 = \{0,1,...,|P_2|-1\}, P_n = \{0,1,...,|P_n|-1\}$ be the input space, and $Q = \{0,1,...,|Q|-1\}$ be the output space. The multi-valued relation $R : P_1 \times P_2 \times ... \times P_n \to 2^Q$ maps each minterm in the input space, $P_1 \times P_2 \times ... \times P_n$, to a set of values in Q, i.e. $m \in P_1 \times P_2 \times ... \times P_n, R(m) \subseteq Q$. We assume that R is complete, i.e. $R(m) \neq \emptyset$, for all m. Associated with R is a set of multi-valued functions $\{f_i\}$ compatible with R, $f_i \prec R$. The multi-valued minimization problem is to find an optimal implementation of f that is compatible with R.

Example 1 *Multi-valued relation R_1 is defined in the following sum-of-products table.*

[1]This work was supported by SRC under contract 684.004 and the California Micro program.

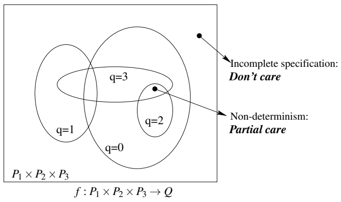

Figure 1: Multi-valued functions

$\{0,1\}$	$\times$	$\{0,1,2\}$	$\times$	$\{0,1,2\}$	$\rightarrow$	$\{0,1,2,3\}$
0		1		$-$	$\rightarrow$	$\{0,1\}$
1		$-$		1	$\rightarrow$	$\{0,2\}$
$-$		0		1	$\rightarrow$	$\{1,2\}$
0		$-$		2	$\rightarrow$	$\{1,3\}$

The mapping is shown graphically in Figure 1. As can be seen, two types of flexibility exist, namely incomplete specification and non-determinism. There are unspecified minterms in the table, e.g. $\{0\}\times\{0\}\times\{0\}$, which can take any value in Q for the mapping. This represents the traditional don't care minterms in the binary domain. There are also minterms that can take a subset of the values in Q, e.g. $\{0\}\times\{1\}\times\{2\}\rightarrow\{0,1,3\}$.

We call these *partial care* minterms. This situation is not present in binary logic, where each minterm can take either value 0 or value 1 if it is not a don't care.

Definition 1 (Don't Care) *A minterm $m \in P_1 \times P_2 \times \ldots \times P_n$ for multi-valued relation $R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q$ is a don't care, iff $R(m) = Q$.*

Definition 2 (Partial Care) *A minterm $m \in P_1 \times P_2 \times \ldots \times P_n$ for multi-valued relation $R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q$ is a partial care, iff $|R(m)| > 1$ and $R(m) \subset Q$.*

Partial cares can result from observability relations in a logic network, or special requirements given by the designer. Consider an MV-node n_i in an MV-network. There exists a set of minterms, which when applied at the primary input space, allow the output values of n_i to be within a subset of the values $v_s \in |x_i|$ for n_i, where x_i is the output variable for node n_i and $|x_i|$ is the number of values specified for x_i. This set of minterms, when mapped into the local input space of n_i, can be used to produce any value in the subset v_s for node n_i. This provides additional flexibility in the implementation of n_i, which does not affect the functionality of the network. In the hardware implementation, a function needs to be deterministic and produce one value for each input minterm. Therefore, the synthesis process needs to explore the flexibility given by partial cares and produce a deterministic function satisfying some optimality criteria. If the target application is software, however, the functionality of an MV-node need not be determinized for the purpose of output evaluation.

Definition 3 (Compatible) *A multi-valued function $f : P_1 \times P_2 \times \ldots \times P_n \rightarrow Q$ is compatible with a multi-valued relation $R : P_1 \times P_2 \times \ldots \times P_n \rightarrow 2^Q$ if $\forall m \in P_1 \times P_2 \times \ldots \times P_n$, $f(m) \in R(m)$.*

Given a multi-valued relation R with a set of don't cares and partial cares, the minimization problem for hardware implementation consists of two steps: (1) find a multi-valued function f compatible with R; (2) find an optimal implementation for f, in terms of the number of product terms and/or the number of multi-valued literals. For instance, the example given in Figure 1 can be optimized into:

$\{0,1\}$	$\times$	$\{0,1,2\}$	$\times$	$\{0,1,2\}$	$\rightarrow$	$\{0,1,2,3\}$
1		$-$		$-$	$\rightarrow$	0
0		$-$		$-$	$\rightarrow$	1

If the multi-valued output variable is encoded into binary variables, the multi-valued relation becomes a Boolean relation. Exact and heuristic algorithms for Boolean relation minimization [2] [3] [4] [5],etc., can thus be used to minimize multi-valued relations.

3 Observability in multi-valued logic networks

Observability Boolean relations have been studied for Boolean networks, e.g. [7][8][9], but computational intensity has prevented the methods from being practical. Multi-valued observability partial cares are a generalization of binary observability relations.

A multi-valued combinational logic network, or *MV-network*, is a network of nodes. Each node represents a multi-valued function with a single multi-valued output and multi-valued inputs. There is a directed edge from node i to node j, if the function at node j explicitly depends on the output variable at node i. We use $|x_i|$ to denote the number of values specified for the MV-variable x_i at node i. Algebraic methods like [10] [11] can be used to derive an appropriate structure for the MV-network. Once the structure has been decided, the multi-valued function at each node can be optimized according to the maximal permissible behavior allowed for this node. The flexibility is given by satisfiability don't cares (SDC), observability don't cares (ODC) and observability partial cares (OPC). For the definition and application of binary SDC and ODC refer to [7].

3.1 Maximal set of observability don't cares

Let y_i be the output variable, and $\{x_1,\ldots,x_r\}$ be the input variables of node i. Let $y_i \in \{0,\ldots,n\}$, and $x_j \in \{0,\ldots,t_j\}$. The MODC for the input edge x_j, is the set of minterms in the primary input space, such that the output MV-function y_i is insensitive to all values of x_j. This set of minterms can be used as don't cares for the minimization of the MV-function x_j, if y_i is the only fanout of x_j. We first compute the set of don't care minterms *MODC* in the local input space of y_i, under which the values of x_j are indistinguishable. This gives the maximal

ODC for edge $x_j \to y_i$. *MODC* can be defined as follows:

$$MODC(y_i,x_j) = \{m | f(m[x_j = 0]) = ... = f(m[x_j = t_j]), m \in P_1 \times ... \times P_r\} \quad (1)$$

We use $m[x_j = k], k \in \{0,\ldots,t_j\}$ to denote setting the value of x_j in minterm m to k. The value of y_i does not change, if we arbitrarily flip the value of x_j within the range $\{0,\ldots,t_j\}$ and keep the other parts of minterm m fixed. This gives the condition that the function produced by x_j is totally blocked by the other inputs and can not be observed at y_i. $x_j \to y_i$ becomes a redundant wire for this set of minterms. It can be shown that if f is specified as a function, i.e. deterministic, *MODC* can be computed using the following formula:

$$\begin{aligned} MODC(y_i,x_j) &= f^0_{x_j^0} \cdot f^0_{x_j^1} \cdots f^0_{x_j^{t_j}} + \cdots + f^n_{x_j^0} \cdot f^n_{x_j^1} \cdots f^n_{x_j^{t_j}} \\ &= \sum_{l=0}^{n} \prod_{k=0}^{t_j} f^l_{x_j^k} \quad (2) \end{aligned}$$

f^l is a binary function, $P_1 \times ... \times P_r \to B$, which defines the set of minterms in $P_1 \times ... \times P_r$ that produce output l for y_i. Function $f^l_{x_j^k}$ is the cofactor of binary function f^l with respect to literal x_j^k. It is independent of x_j and preserves the onset of f^l whenever $x_j = k$, i.e. $x_j \cdot f^l_{x_j^k} = x_j \cdot f^l$. Function $f^l_{x_j^0} \cdot f^l_{x_j^1} \cdots f^l_{x_j^{t_j}}$ defines the set of minterms in $P_1 \times ... \times P_r$ such that the output value for y_i is always l no matter what value x_j takes, i.e., the universal quantification over the values of x_j. Formula (2) represents the complement of the *multi-valued logic difference* $(\partial f / \partial x_j)$ when f is deterministically specified.

Theorem 1 (MODC) *The binary function (2) computes the set of MODC minterms for input edge x_j, in the local inputs space of y_i as defined in (1), if the functionality of y_i is deterministically specified in f.*

If f is specified as a relation, i.e. nondeterministic, more complicated methods, which use multi-valued logic difference, need to be applied.

3.2 Compatible set of observability don't cares

The validity of MODC's for a particular input edge requires other input edges to produce certain values. Cyclic dependencies in this relationship cause incompatibility. Consider node i, with input edges: $x_1,\ldots,x_j,\ldots,x_l,\ldots,x_r$. Let $x_j \in \{0,\ldots,t_j\}$, and $x_l \in \{0,\ldots,t_l\}$. Let $MODC^j$ and $MODC^l$ be the maximal set of observability don't cares for the input edges x_j and x_l respectively. Let $m_q \in P_1 \times ... \times P_r$, be a minterm in the local input space of $\{y_i\}$, such that $m_q \in MODC^j$ and $m_q \in MODC^l$. The primary input minterms that produce m_q will be used as don't cares for both x_j and x_l. The optimization of x_j as a result of m_q may destroy the validity of $MODC^l$ and vice versa. The minterm in $MODC^j$, for input edge x_j, is said to be compatible with $x_l, l \neq j$, if it is not a minterm in $MODC^l$, or if $MODC^j$ does not depend on the value of x_l, i.e.

$$CODC^{j_l} = \{m | (m \notin MODC^l) \vee (\forall x_l(m) \in MODC^j), m \in MODC^j\}$$

$CODC^{j_l}$ is the subset of $MODC_j$ that is compatible with $MODC_l$. By "$\forall x_l$" we denote the computation of universal quantification over all values of x_l. $\forall x_l(m)$ is the multi-valued cube expanded from m and does not depend on x_l. The interpretation of this formulae is that: of all the minterms, $m \in MODC(f,x_j)$, where f is insensitive to x_j, m is said to be compatible with another input edge x_l, if (1) either m is not a don't care for x_l, or (2) no matter what value is chosen for x_l, f is still insensitive to x_j under m.

The input edges are implicitly ordered and the CODC for each input edge is computed by making the associated MODC compatible with all the preceding edges in the ordering. Given an ordering $x_1 \prec \ldots \prec x_j \prec \ldots \prec x_r$, the CODC for edge x_j can be defined as follows:

$$CODC(y_i,x_j) = \{m | \forall l < j, (m \notin CODC_l) \vee (\forall x_l(m) \in MODC_j), m \in MODC_j\}$$

This approach gives the first edge in the ordering the most flexibility. Successive edges are more restricted in order to be compatible with previous ones. The ODC set for each successive node is thus reduced for compatibility.

In practice, the set of minterms in MODC can be represented symbolically using MDD's [12] [13]. Also, the CODC set for each node can be inherited by all input edges. The formula thus can be constructed with MDD operations:

$$\begin{aligned} CODC(y_i,x_j) &= P_1(P_2(\cdots P_{j-1}(MODC(y_i,x_j)))) + CODC_{y_i} \\ P_k(F) &= \overline{CODC_{x_k}} \cdot F + \forall x_k(F) \\ CODC_{x_j} &= \prod_{i \in fanout(x_j)} CODC(y_i,x_j) \quad (3) \end{aligned}$$

$CODC(y_i,x_j)$ is the *edge*-CODC for edge $x_j \to y_i$. $CODC_{y_i}$ is the *node*-CODC for the fanout node y_i. $CODC(y_i,x_j)$ is computed for each fanout edge of node x_j and they are intersected to give the *node*-CODC for x_j. This is passed to all fanin nodes of x_j. $P_k(F)$ is the compatibility operation which is applied to each fanin node of y_i that precedes x_j in the pre-assigned order.

Theorem 2 (CODC) *The set of minterms computed by (3) are don't cares for node x_j and they remain to be don't cares if the functions of all other nodes change within their computed CODC's.*

3.3 Maximal set of observability partial cares

CODC's do not capture all the flexibility for MV-networks. For the input edge x_j of node i, there are minterms such that y_i is insensitive to only a subset of the values for x_j. In other words, minterms under which a subset of the values for x_j is

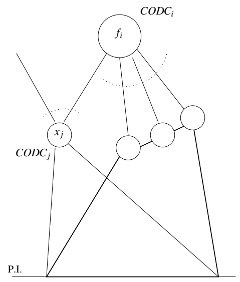

Figure 2: CODC computation

indistinguishable at y_i. These minterms are, by the definition given in Section 2, partial cares for the function implemented at node x_j, and can also be used in the minimization of x_j.

The maximal set of partial cares for edge $x_j \to y_i$ can be defined on the power set, 2^{P_j}, of the values for x_j. Let $x_j \in P_j = \{0, \ldots, m\}$ and $v = \{r_1, \ldots, r_v\} \in 2^{P_j}$. For each subset of the values $v_t \in 2^{P_j}$ for x_j, there exists a set of minterms S_t in the local input space, such that the values in the subset v_t can not be distinguished at the output y_i. We compute these sets of minterms S_t for each such subset of values v_t. When mapped into the primary input space, S_t represents the set of partial care minterms for node x_i, where x_i can take any value in v_t.

$$\begin{aligned} OPC(f,v,x_j) &= \{m | f(m[x_j = r_1]) = \cdots = f(m[x_j = r_v]), \\ &\quad m \in P_1 \times \cdots \times P_r\} \\ MOPC(y_i,x_j) &= \{(m,v) | m \in OPC(y_i,v,x_j), v \in 2^{P_j}\} \end{aligned} \tag{4}$$

$OPC(f,v,x_j)$ gives the set of minterms in the local input space of f, such that a subset v of the values for x_j are indistinguishable at y_i, i.e. the output function of y_i does not change, if we arbitrarily change the value of x_j within the subset of values $v = \{r_1, \ldots, r_v\}$, while keeping the other parts of m fixed. Therefore, MOPC by definition is a set of pairs, (m,v), where $m \in P_1 \times \cdots \times P_r$, is a minterm in the local input space of y_i, and $v = \{r_1, \ldots, r_v\}$ is a subset of the values for x_j. Similar to (1) MOPC's can be computed by the following formula if f is deterministically specified.

$$\begin{aligned} & MOPC(y_i,x_j) \\ = & \sum_{v \in 2^{P_j}} \left(v, \left(f^0_{x_j^{r_1}} \cdot f^0_{x_j^{r_2}} \cdots f^0_{x_j^{r_v}} + \cdots + f^n_{x_j^{r_1}} \cdot f^n_{x_j^{r_2}} \cdots f^n_{x_j^{r_v}} \right) \right) \\ = & \sum_{v \in 2^{P_j}} \left(v, \sum_{l=0}^{n} \prod_{k \in v} f^l_{x_j^k} \right) \end{aligned} \tag{5}$$

This computation can be expensive due to the power set summation. OPC needs to be computed for every subset of $|P_j|$, which is exponential. In practice multi-valued variables usually have a small set of values to choose from, thus suggesting the feasibility of (5). However algorithmic trade-offs need to be explored to assess the practicality of MOPC.

Algorithm [CODC-based MV-network minimization]:
input: MV-network ntk
input: external don't care XDC_j at each primary output j
local $CODC_i$: CODC set for node i
local DC_i: complete don't care set for node i
Traverse each node j in ntk in reverse DFS order
    If j is primary output
        $CODC_j =$ external don't cares (XDC_j)
        Continue;
    For each fanout node k
        D = $MODC(f_k, y_j)$
        For each fanin node i of k that is already visited
            $D = [\overline{CODC_i} + \forall y_i] \cdot D$
        $D = D + CODC_k$;
        $CODC_j = CODC_j \cap D$;

    Collapse $CODC_j$ into primary input space
    Quantify out the variables not in the TFI cone of y_j
    $DC_j = \neg image(\neg CODC_j)$
    $MINIMIZE(ONSET_j, DC_j)$
End

Figure 3: CODC-based MV-network minimization

Like observability don't cares, observability partial cares may also become invalid if the functions of related nodes are changed. Similar to the approach used in CODC, we can order the input edges for each node, and make the MOPC set compatible with each preceding edge. The advantages of COPC for MV-network minimization needs to be evaluated further and are not explored here.

4 Implementation

We give an algorithm to compute a set of CODC's for each node in a multi-level MV-network. The algorithm is an extension of the binary CODC computation from [7]. The computation of MODC and CODC is implemented in a multi-valued logic synthesis infrastructure called MVSIS. MOPC and COPC computations are expensive and require a multi-valued relation minimizer for the optimization of each node; it is not implemented in the current version.

4.1 MV-network optimization

All computation is carried out using MDD operations. A heuristic depth-first search MDD variable ordering is implemented. The logic function of each node in the network is represented by a multi-valued *table* structure, as defined in VIS [14]. A *table* is a sum-of-products representation of a multi-valued function. Each row is partitioned into an input part and an output part, and represents a multi-valued cube.

The algorithm traverses the MV-network in a reverse topological order from primary outputs to primary inputs. Each node in the network is traversed once. At each node, the *CODC* set is computed for each fanout edge, and then intersected to give the approximated *CODC* set for this node. Once calculated, the *CODC* set for each node is inherited by each of its own fanin nodes. The *CODC* set is mapped into the primary input space by variable substitution, and then mapped into the local input space by image computation. Given the DC set, the logic minimization of a multi-valued node is carried out using ESPRESSO-MV [15].

4.2 Multi-valued image computation

Two methods exist for image computation, transition relation and recursive range computation. We extend the recursive range computation from the binary domain to the multi-valued domain. For binary image computation, refer to [7]. Multi-valued cofactoring is used to reduce the computation in a recursive fashion.

In the local input space of node y_i, each input variable is cofactored by the complement of the don't care set $A(x)$, which is an MDD in terms of primary input variables. This array of cofactored functions give the transition functions that map the entire primary input space *PI* into the local *care set* of node y_i.

$$\begin{aligned} F_{A(x)} &= [(f_1)_{A(x)}, (f_2)_{A(x)}, \ldots, (f_r)_{A(x)}] \\ A(x) &= \overline{CODC^{y_i}_{PI}}(x) \end{aligned}$$

Each f_k is the multi-valued function for one of the fanins, x_k, of y_i. Each f_k is represented by an array of MDD's; each MDD represents the onset for one of the values of f_k. The cofactor $(f_k)_{A(x)}$ is obtained by *constraining* the MDD function for each value of f_k against $A(x)$. This is called the *constrain* computation. Once we have the array of range functions, we apply output cofactoring to carry out the recursive image computation:

$$\begin{aligned} \overline{CODC^{y_i}_{local}} &= IMAGE(\overline{CODC^{y_i}_{PI}}) = RANGE(F_{A(x)}) \\ &= \sum_{k=0}^{|P_1|-1} x_1^k \cdot RANGE([f_2, \ldots, f_r]_{f_1^k}) \end{aligned}$$

In the above formula, x_i^k denotes the literal for the intermediate variable x_i that takes the k^{th} value. $|P_i|$ is the total number of values specified for x_i. The range computation is recursively applied to the list of functions, until every one has been cofactored. The final result is a set of cubes in the local input space of y_i, which can then be used in the minimization of node y_i.

5 Experiments

Some experiments were performed on a number of MV-networks. We require that the input MV-network is completely specified at each node and deterministic. Local non-determinism is allowed at intermediate nodes. The first set of examples, *Sort*, *matmul* and *max* are made-up examples. The second set of examples come from the multi-valued benchmark set distributed with VIS [14]. We extract individual combinational modules and remove the latches to obtain the MV-network. For example, *bakery.proc* is the module *process* in benchmark *bakery*. The third set of examples come from the multi-valued logic benchmark suite from Portland State University. They are two-level multi-valued PLAs generated from machine learning applications.

	#cubes		*#literals*	
examples	simp	fullsimp	simp	fullsimp
max	91	91	168	168
matmul	72	48	240	136
sort	594	336	2296	1339
bakery-proc	79	73	208	188
coherence-cch	99	69	337	229
eisenberg-proc	57	57	140	140
slider-nsf	321	321	1210	835
car	23	23	92	88
iris	25	25	107	124
mm3	10	10	37	37
mm4	27	23	109	100
mm5	55	48	231	232
balance	61	55	244	267
nursery	53	40	280	208

Table 1: Results for fullsimp

The CODC-based MV-network minimization is implemented in MVSIS as command *fullsimp*. As a comparison, we implemented command *simp*, which calls ESPRESSO-MV for each node directly without computing CODC's. Table 1 shows the comparison in the number of multi-valued cubes and literals. There is about 20% gain by computing CODC's. We combine *fullsimp* with algebraic decomposition as in [11] and form heuristic network optimization scripts like *script.rugged* in SIS. The script repetitively applies decomposition, full simplification and node elimination until no improvement. Table 2 shows the optimization results for the same set of examples. Note that only deterministic examples can be minimized and produce meaningful results. The results are verified by the combinational verification package in VIS. Also note that only $|P|-1$ values are represented in the sum-of-products form if a node has $|P|$ values. The value with the most cubes is treated as default and is not counted.

The experiments are performed on an Intel 500MHz machine with 128MB memory. The run time ranges from 1-10 minutes depending on the size of the example. The reduction in cube count and literal count is significant compared with the original specification. This translates into implementation cost savings whether in software or in hardware.

	#*cubes*		#*literals*	
examples	orig	script	orig	script
max	196	91	392	168
matmul	120	48	440	136
sort	594	158	2296	468
bakery-proc	788	69	1604	184
coherence-cch	120	66	433	230
eisenberg-proc	1341	57	2767	140
slider-nsf	1359	293	2574	891
car	518	21	3108	83
iris	100	25	400	124
mm3	111	10	555	37
mm4	598	23	2990	100
mm5	2055	48	10275	232
balance	337	55	1348	267
nursery	8640	40	69120	208

Table 2: Results for optimization scripts

6 Conclusion

We generalized the notion of don't cares used in binary logic to multi-valued logic, and defined partial cares for multi-valued relations. We gave a method to construct multi-valued observability don't cares in a combinational network. We showed how observability partial cares can be generated, and discussed their potential complexity. Experimental results were given to show the effectiveness of using CODC's for node minimizatoin.

Some future research directions are: (a) Devise heuristic algorithms to generate observability partial cares efficiently; (b) Apply a multi-valued relation minimizer in the optimization process, which uses partial cares; (c) Apply multi-valued logic minimization to other CAD areas, e.g. state encoding, software compilation and asynchronous hardware synthesis.

Acknowledgement

We are grateful for the support of the SRC under contract 684.004 and the California Micro program and industrial sponsors, Fujitsu, Cadence, Motorola and Synopsys.

References

[1] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Laboratory, Univ. of California, Berkeley, CA 94720, May 1992.

[2] R. K. Brayton and F. Somenzi, "Minimization of Boolean Relations," in *Proc. of the Intl. Symposium on Circuits and Systems*, pp. 738–743, May 1989.

[3] Y. Watanabe and R. K. Brayton, "Heuristic minimization of multiple-valued relations," *IEEE Transaction on CAD*, 1993.

[4] B. Lin and F. Somenzi, "Minimization of Symbolic Relations," in *Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 88–91, Nov. 1990.

[5] A. Ghosh, S. Devadas, and A. R. Newton, "Heuristic minimization of boolean relations using testing techniques," *IEEE Transaction on CAD*, 1992.

[6] H. Savoj, R. K. Brayton, and H. Touati, "Extracting Local Don't Cares for Network Optimization," in *Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 514–517, Nov. 1991.

[7] H. Savoj, *Don't Cares in Multi-Level Network Optimization*. PhD thesis, University of California Berkeley, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[8] M. Damiani and G. D. Micheli, "Observability don't care sets and boolean relations," in *Proc. of the Intl. Conf. on Computer-Aided Design*, 1990.

[9] Y. Watanabe, L. M. Guerra, and R. K. Brayton, "Permissible functions for multioutput components in combinational logic optimization.," *IEEE Transaction on CAD*, 1996.

[10] R. K. Brayton, "Algebraic methods for multi-valued logic," Tech. Rep. UCB/ERL M99/62, Electronics Research Laboratory, University of California, Berkeley, Dec. 1999.

[11] M. Gao and R. K. Brayton, "Semi-algebraic methods for multi-valued logic," in *Proc. of the Intl. Workshop on Logic Synthesis*, May. 2000.

[12] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton, "Algorithms for Discrete Function Manipulation," in *Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 92–95, Nov. 1990.

[13] T. Kam and R. K. Brayton, "Multi-valued deisoin diagrams," Tech. Rep. UCB/ERL M90/125, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, Dec. 1990.

[14] VISgroup, "VIS: A system for verification and synthesis," in *IEEE International Conference on Computer-Aided Verification*, 1996.

[15] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for pla optimization," *IEEE Transaction on CAD*, 1988.