Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp590-595)

ALGORITHMS FOR PHYSICAL IMPLEMENTATION OF MULTIPLE-VALUED CIRCUITS

DORIN SIMA
Computer Science Department
University "L.Blaga"
Str. Emil Cioran, No.4, Sibiu, 550025
ROMANIA
http://csac.ulbsibiu.ro/

Abstract: The task of logic synthesis is to convert the logic description of set function into a netlist of gates that implements the functions. This paper describes the possibility of implementing some combinational and sequential circuits with multiple-valued PLAs (MVPLA), by multiple-valued multiplexers (MVLMUX) or multi-valued switches. The algorithms are based on multiple-valued decision diagrams (MDD) representation of the functions. The developed methodology offers some elegant algorithms that automatically map a MMD functions representation in to some certain multi-valued physic circuits. Also, these algorithms convert high logical functions representations into a lower one, very useful taking into account technological restrictions.

Key words: Multiple-valued Decision Diagrams, Logic Synthesis, Technology Mapping

1 Introduction

In CAD area, we often meet the situation that the logical functions are naturally described in certain logic (p-valued), but the available technology demands the using of another logic (n-valued). By exeample, in the case of combinational circuits implementation by MVPLAs (multivalued PLAs) there are often used binary structures, which accomplish the output functions. In the same way, for the sequential circuits, STG is naturally described in a p-valued logic, but the implementation imposes state, input and output encoding in order to get the available technological logic.

In this paper are presented some automatic methods for multiple-valued circuits synthesis. The methods are based on MDD representation of functions. In this sense, we present some concrete implementations using well-known multiple-valued circuits like muliplexers cells (MVL-Mux), multiple-valued PLA (MVL-PLA). Multi-valued switches based implementation sample can be found on extended version of this paper.

2 Preliminaries

We start with a short review of multivalued notations and functions reprezentation. For more details about MDDs see [3].

2.1 Multiple-valued logic functions

Let F be a multiple-valued input, multiple-valued output function of n variables : $x_1, x_2, \ldots, x_n$.

$$F : P_1 \times P_2 \times \ldots \times P_n \rightarrow Y$$

Each variable, x_i, may take any one of the pi values from a finite set $P_i = \{0, 1, \ldots p_i - 1\}$.

The output function F may take *m* values from the set $Y = \{0, 1, \ldots m - 1\}$.

Let T_i be a subset of P_i. The *Literal* of variable x_i is defined as the Boolean function:

$$x_i^{Ti} = \begin{cases} 0 \text{ if } xi \notin Ti \\ 1 \text{if } xi \in Ti \end{cases}$$

The *Cofactor* of F with respect to a variable x_i taking a constant value *j* is:

$$F_{x_i^j} = F(x_1, \ldots x_{i-1}, \boldsymbol{j}, x_{i+1}, \ldots x_n),$$

function depending on n -1 variables

Other notations for cofactor: $F_{xi=j}$, $F^j{}_{xi}$

Note: If F not depend on x_i, then $F_{xi=j} = F$.

The Shannon decomposition of a function F with respect to a variable x_i is:

$F = \sum_{j=0}^{pi-1} x_i^j \cdot F_{xi=j}$, where operations are max and min

2.2 Multi-valued Decision Diagrams

Definition. A multi-valued decision diagram (MDD) is a rooted, directed acyclic graph. Each nonterminal vertex **v** is labeled by a multi-valued variable var(**v**), which can take values from a range range(**v**). Vertex **v** has arcs directed towards | range(**v**) | children vertices, denoted by $child_k$(**v**) for each k ∈ range(**v**). Each terminal vertex **u** is labeled a value:

value(**u**) ∈ {0, 1 . . ., m -1}

For each nonterminal vertex **v** representing a function F, its child vertex $child_k$(**v**) represents the function $F_{v=k}$ for each k ∈ range(**v**).

Table I Diagram for function Max(x,y)

x \ y	**0**	**1**	**2**
0	0	1	2
1	1	1	2
2	2	2	2

For a given assignment to the variables, the value yielded by the function is determined by tracing the path from the root to a terminal vertex, following the branches indicated by the values assigned to the variables. The terminal vertex label then gives the function value.

Example: The MDD in Figure 1 represents the discrete function F = max(x ,y) in 3- valued logic.(see Table I)

An MDD is *ordered* if there is a total order '<' over the set of variables such that for every nonterminal vertex v, var(v) < var($child_k$(v)) if $child_k$(v) is also nonterminal.

An MDD is *reduced* if:

1. it contains no vertex v such that all outgoing arcs from v point to a same vertex, and
2. it does not contain two distinct vertex **v** and **v1** such that the subgraphs rooted at **v** and **v1** are isomorphic.

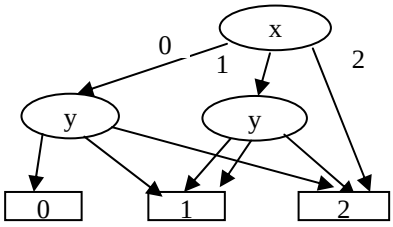
A reduced ordered multi-valued decision diagram (ROMDD) is an MDD, which is both reduced and ordered.

Variable ordering must be decided before the construction of any MDD. We assume that this has been decided and that the naming of input variables have been permuted so that xi < x i+1 .

MDDs are guaranteed to be reduced at any time during the constructions and operations on two MDDs. Each operation returns a resultant MDD in a reduced ordered form.

Example: The MDD in Figure 1 is ROMDD . The variable ordering is x < y. Note that one redundant nonterminal vertex and six terminal vertices have been eliminated.

Fig. 1 MDD for Max(x,y)



A very important property of an ROMDD is that it is a **canonical** representation.

It is efficient to use the strong canonical forms: expressions in different locations represent different function. As in binary case, using the symbol table and building the MDDs in bottom-up manner, all MDDs are in strong canonical form.

Definition The CASE operator selects and returns a function G_i according to the value of the function F:

$$CASE(F, G_0, G_1, \ldots, G_{m-1}) = G_i \text{ if } (F=i)$$

The operator is defined only if range(F) = {0, 1, …, m – 1}. The function returned from the CASE operation has a range of range(G_i). In particular, if the G_i are binary-valued, the resultant function will also be a binary-valued output function.

The pseudo-code for recursive CASE algorithm is given in fig 2 [3]. The *symbol-table* stores nodes of MDDs and *computed-table* maintain the sub-instances already computed with CASE.

Notably is the fact that any multi-valued operator can be implemented using CASE. For example, if f and g are 3-valued functions (reprezented by two Mdds: F and G), the 3-valued operator Max, can be expressed as:

Max(F, G) = CASE (F,
CASE (G,**0, 1, 2**),
CASE(G, **1**, **1**, **2**),
CASE(G, **2**, **2**, **2**))

where bolded numbers **0 , 1 , 2** denotes the *terminal nodes* representing logical values 0, 1 and 2.

2.3 The Apply definition

In practical implementations of MDD packages, problems can arise from the supposition that the logic is known and/or the operands work in the same logic. The negation, for example, in different logical system is expressed as below:

- 2-valued: not(F) = CASE(F, **1, 0**)
- 3-valued: not (F) = CASE(F, **2, 1, 0**)

Fig 2. Algorithm for CASE

```
CASE(F, G0,…,Gm-1){
    if terminal case retun result
    if CASE(F, G0,…,Gm-1) in computed-table return result
    let x = top-variable of  F, G0,…,Gm-1
    let p logic-set of x
    for j = 0 to p-1 do
        H_{x=j}=CASE(F_{x=j}, G0_{x=j},…,Gm-1 _{x=j});
    //F _{x=j} is cofactor of F with respect to x=j
    if H_{x=0} =H_{x=1} =…=H_{x=p-1}
        return H_{x=0};
    result = addORfind <x, H_{x=0},…,H_{x=p-1} > in symbol-table
    insert result in computed-table for CASE(F, G0,…,Gm-1)
    return result
```

In this paper we define a new operator called Apply. The main idea is to construct dynamically the MDD representing a multi-valued operator and then, 'apply' this MDD to the MDD of operand(s).

Notations.

- Suppose that, for given operator, the maximum number of logical values over the range of all operands is $\mathbf{p_{max}}$.
- The $\mathbf{p_{max}}$**–valued** MDDs constructed for each n-ary operation (not, min, max, etc.), using the variables \$0, \$1, etc., are named here *formal mdds*
- As we have seen, there is a total order over the set of all variables. If each variable is identified by a unique id (natural number), then: x<y *if and only if*: id(x) < id(y).
- Let be '\$0','\$1','\$2', etc, a *reserved variable names* and id(\$0)=0, id (\$1)=1,etc. If F is an MDD, **F.id** denotes the id of root variable of F (F.id=-1 if F is a terminal node).
- In a MDD, if a root contains a p-valued variable v, then the root node will have the MDDs $G_0, G_1, \ldots, G_{p-1}$ as a children. We note that MDD by: <**v**, $G_0, G_1,\ldots,G_{p-1}$>

Suppose, for example, a binary operation that can be implemented by "applying" the corresponding formal-mdd over the two MDDs representing some p_{max}-valued operands. The next two equations describes the recursive algorithm for Apply:

Apply(OP, f0,f1) = OP **if** OP is terminal node

Apply(OP,f0,f1) = CASE($f_{OP.\,id}$,
Apply(OP0, f0,f1),
…,
Apply(OP p_{max-1}, f0,f1))

Fig 3. a) MDD forF0 and F1; b) Result=Apply(OP, F0, F1)

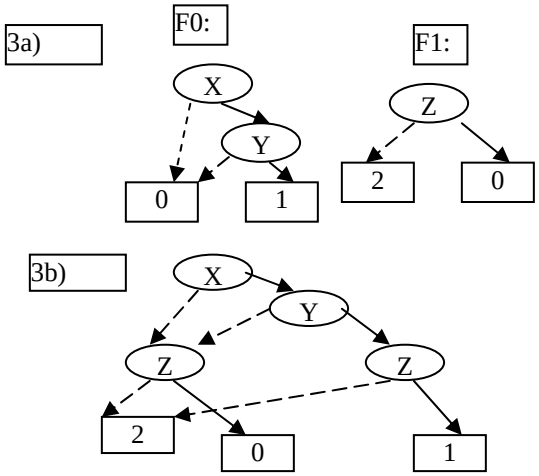
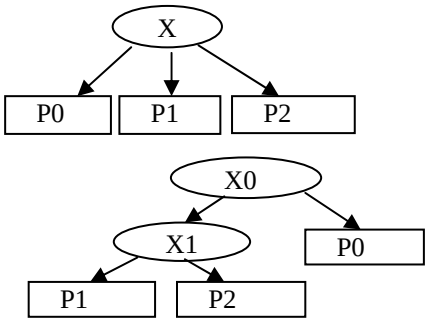


Fig. 4 p-valued node replacing



Here, OPi represent $child_i(OP)$, i = 0,1,…,p_{max-1}.
For example, consider F0 and F1with the MDDs from figure 3 (with p_{max} =3) and OP is the MDD from figure 1, corresponding to *Max* operator for 3-valued. In the MDD for OP, variables are '\$0' in root of OP and '\$1' in the nodes on level 1.

Form figure 1 we can see that:
OP=<$0,OP0, OP1, OP2>,
where: OP0=<$1, **0, 1, 2**>, OP1=<$1,**1, 1, 2**>, OP2=**2**

From figure 3a:
F0 = <X, **0**, <y, **0, 1**> >, F1 = <Z, **2, 0**>

We have also OP.id=0 , OP0.id=OP1.id=1 , so that:
Apply(OP, F0, F1) = *CASE*(F0,
Apply(OP0,F0,F1),
Apply(OP1,F0,F1),
Apply(OP2,F0,F1)) =
CASE(F0,
CASE (F1 ,**0, 1, 2**),
CASE(F1 ,**1, 1, 2**), **2**))
Finally, from the CASE definition we have the result:
<X, < Z, **2, 0**>,
<Y, < Z, **2, 0**>, <Z, **2, 1**>>
>

3 Mapping p-valued MDD into n-valued MDDs

Let be an m-variable p-valued input function with p-valued output:

$F_p : P^m : \rightarrow P$, where logic set P is P={0,1,…,p-1}

Suppose the Fp is represented by p-valued MDD (node's variables are p-valued and there are p terminal nodes).
Let be *n* the number of logical values of n-valued logic set N= {0, 1…, n-1}, so that n < p.
We want to encode the Fp by *a* functions, where $a=\lceil \log_n p \rceil$, and these functions are n-valued. In fact we need to construct *a* n-valued MDDs of these functions.

$F_n^i : N^{a \cdot m} \rightarrow N$, where N={0,1,…,n-1}, $a=\lceil \log_n p \rceil$.
and i={0,1,…,a-1}

We shall use below the notations:

MDD_F – The Mdd of function F
MDD_Fn,p-Mdd of function having
n-valued inputs(encoded) and p-valued output
MDD_F0, MDD_F1,…- the Mdds for functions that encode F
COD_F- The Mdd for coding F

The algorithm for MDDs mapping is:

Step 1. Find the number of variables by which the p-valued variables are encoded. Replace each node from MDD_F with one sub-graph resulted from chosen coding scheme. If the replaced node contain variable X then the sub-graph will contain variables:
x0, x1,…, x $\lceil \log_n p \rceil$.-1,as in figure 4. The resulting MDD is MDD_Fn,p

Step 2.
Choose the coding scheme for F and construct COD_F. In step 2, we'll choose the coding for F using F0, F1,…F $\lceil \log_n p \rceil$, as the coding variables.

Step 3
.Based on COD_F, built COD_F0, COD_F1,…, the "formal-mdds"(unary operators for Apply), used for MDD_Fi building:

MDD_F0 = Apply(COD_F0, MDD_Fn,p)
MDD_F1 = Apply(COD_F1, MDD_Fn,p)

4 Technology Mapping

We'll show the application of the above algorithms using two very simple examples.

4.1 MVL-Mux based Implementation

Suppose that we have a p-valued function, represented by the corresponding MDD. If the p-valued multiplexer cells can be used then the physical implementation is directly: replace each MDD node by one multiplexer controlled by node variable and having as input data the

Fig 5 a) Reduced MDD_Q

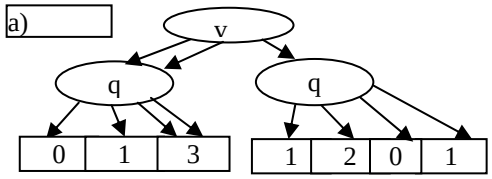
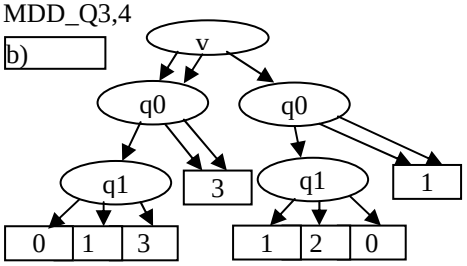


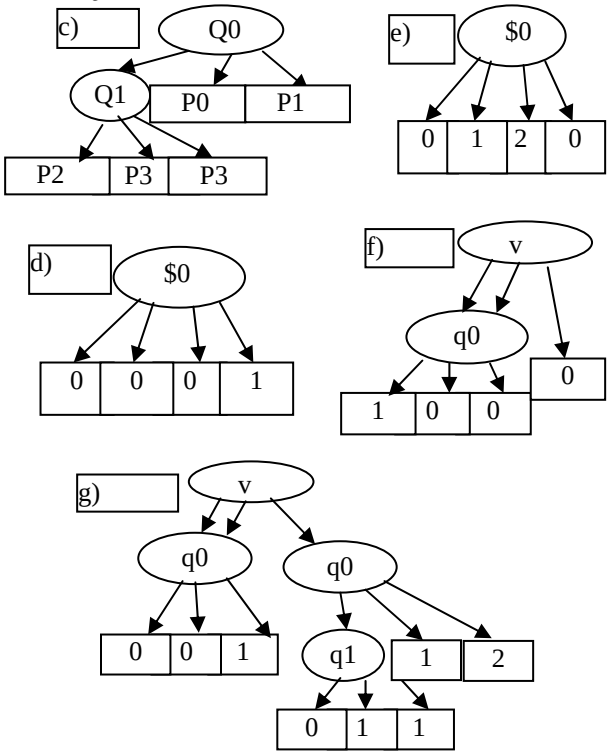
Fig 5 b) MDD_Q3,4



output of multiplexers corresponding to the children of node. If the function to be implemented is p-valued and the available circuits are n-valued, it is necessary to find the corresponding n-valued MDDs. Because each node in a MDD is implemented in this case by a multiplexer, the resulted MDDs can be optimized by using minimization techniques (see [2]).

Consider the synchronous sequential circuit having State Transition Table showed in Table II. The input signal represented by variable *v* is 3-valued and the state variable *q* is 4-valued. The MDD for next state function Q can be constructed from table III. The reduced MDD_Q is shown in fig 5a. Suppose that we want to implement this function with the 3-valued multiplexer cells. Because the q variable and the output

Fig. 5 c)COD_Q;d)COD_Q0;e)COD_Q1;f)MDD_Q0; g)MDD_Q1



of Q are 4-valued the mapping algorithm discussed in chapter III is used. Finally the MDD_Q0 and MDD_Q1, by which MDD_Q is represented, are obtained(fig. 5b…5g).

4.2 MVL-PLA based Implementation

For PLA implementation we have used the structure proposed in [4]. In fig .6a is given the MVLPLA structure. Function F to be implemented is given in the diagram fig 6b and the MDD_F for this function is

Table II Sample State Transiton Table

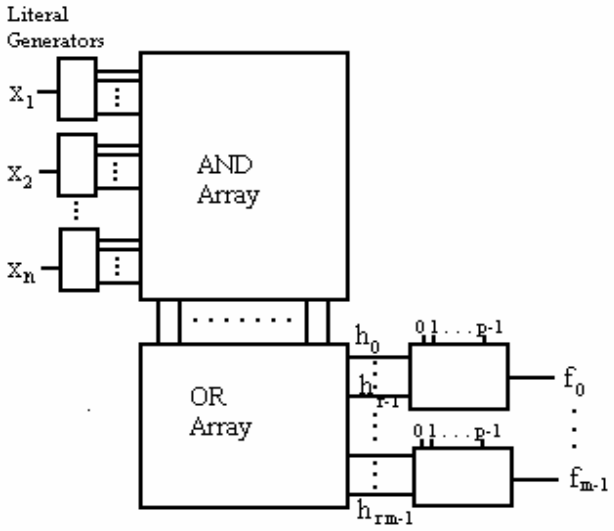
q	v	Q
0	{0,1}	**0**
0	2	**1**
1	{0,1}	**1**
1	2	**2**
2	{0,1}	**3**
2	2	**0**
3	{0,1}	**3**
3	2	**1**

Table IIII Karnaugh for nex-state function

q / v	**0**	**1**	**2**	**3**
0	0	1	3	3
1	0	1	3	3
2	1	2	0	1

shown in fig 6c. Literal generator for 3-valued input is in fig 6d. We want to obtain the binary functions (h0 and h1 in fig 6a) As is stated in[4] between the binary literals Ax, Bx, Cx the next relations exists:

Fig.6a MVL-PLA



Ax+Bx=Ax+Cx=Bx+Cx=2

notAx=Bx·Cx ; **not**Bx=Ax·Cx ; **not**Cx=Ax·Bx

So, for internal nodes the coding tree from fig 6e can be used and the resulting MDD_F2,3 is shown in fig 6f. As above the MDD_F0 and MDD_F1 are obtained in fig 6i and 6j Now, tracing all paths to the terminal node **1** [1] in MDD_F0 the expression for h0 is obtained and from MDD_F1 the expression for h1:

Figure 6b Function to be implemented

y / x	**0**	**1**	**2**
0	0	0	0
1	1	1	2
2	0	1	2

Fig 6c The MDD_F of the function in fig 6b

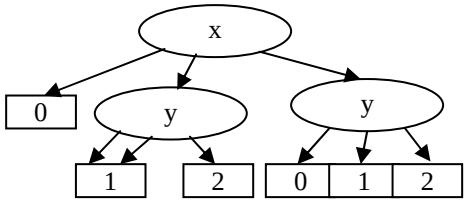
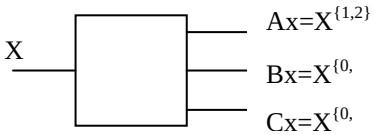


Fig. 6d Literal generator structure



h0 = Ax·Ay·By

h1=Ax·Ay+Ax·(**not**Ay)·(**not**Bx)

Using relations between literals, the positive form for h1 can be obtained:

h1=Ax·Ay+Ax(By·Cy)·(Ax·Cx) =

Ax·Ay+Ax·Cx·By·Cy

5. Conclusions

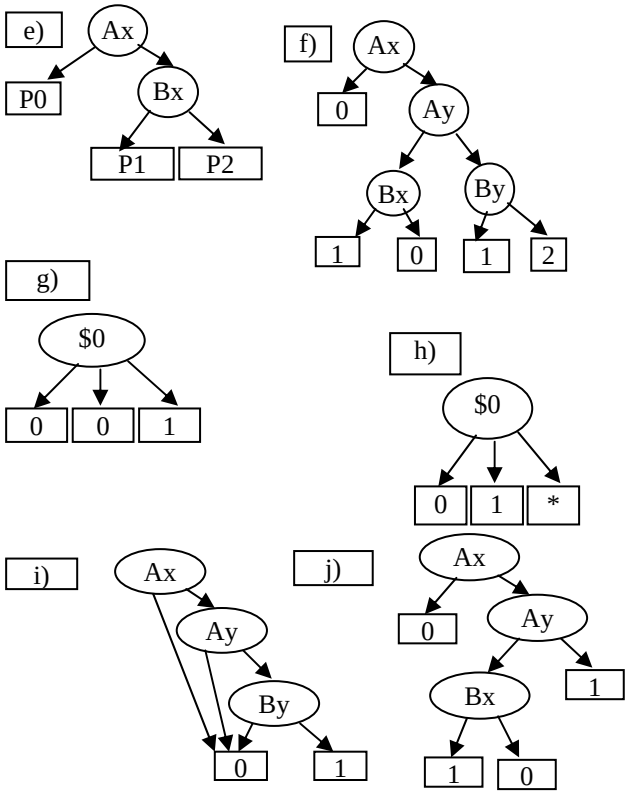
MDDs allow an efficient representation and handling of multi-valued logic functions.

In this paper I tried to emphasize an extra advantage: the possibility of designing efficient algorithms that allow circuit implementation in several technologies.

The developed methodology offers some elegant algorithms that automatically map a MMD functions representation in to some certain multi-valued physic circuits. Also, these algorithms convert high logical functions representations into a lower one, very useful taking into account technological restrictions.

As a further development we intent to quantify the obtained circuit complexity in order to decide the optimal implementation.

Fig 6:e)- internal nodes coding tree, f)- MDD_F2,3 , g,)CD_F0, h)COD_F1,i) Result of Apply(COD_F0, MDD_F23), j) Result of Apply(COD_F1, MDD_F23)



References:

[1] Bryant R - Graph-Based Algorithms for Boolean Function Manipulation, IEEE Trans. Comp. No.8 ,Aug, 1986

[2] Denis V. Popel, Rolf Drechsler - Efficient Minimization of Multiple-valued Decision Diagrams for Incompletely Specified Functions, 33-rd *ismvl*, p. 241, 2003

[3] Kam T.- State Minimization of Finite State Machines using Implicit Techniques, - PhD dissertation, ic.EECS.Berkeley.EDU, 1995

[4] Sasao T.- On the Optimal Design of Multiple-Valued PLA's, IEEE Trans.Comp No4 1989

[5] Hassan M. et al - A Framework for Design a Multivalued Logic Functions and Its Application Using CMOS ternary Switches – IEEE Transactions on Circuits and Systems, Vol. 43 No.4, Apr, 1996