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 MDD functions representation in to some certain multi-valued physical 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 example, 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 multiplexers 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 representation. 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 $p_i$ 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:

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

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

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

function depending on $n-1$ variables

Other notations for cofactor: $F_{x_i=j}$, $F^j_{x_i}$

Note: If $F$ not depend on $x_i$ then $F_{x_i=j} = F$.
The Shannon decomposition of a function $F$ with respect to a variable $x_i$ is:

$$F = \sum_{j=0}^{p_i-1} x_i^{j} F_{x_i}^j$$

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 $\text{var}(v)$, which can take values from a range $\text{range}(v)$. Vertex $v$ has arcs directed towards $|\text{range}(v)|$ children vertices, denoted by $\text{child}_k(v)$ for each $k \in \text{range}(v)$. Each terminal vertex $u$ is labeled a value:

$$\text{value}(u) \in \{0, 1, \ldots, m-1\}$$

For each nonterminal vertex $v$ representing a function $F$, its child vertex $\text{child}_k(v)$ represents the function $F_{v=k}$ for each $k \in \text{range}(v)$.

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 $v$, $\text{var}(v) < \text{var}(\text{child}_k(v))$ if $\text{child}_k(v)$ is also nonterminal.

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

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.

<table>
<thead>
<tr>
<th>$x$</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 1 Diagram for function $\max(x, y)$

A very important property of a 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 $F$:

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

The operator is defined only if $\text{range}(F) = \{0, 1, \ldots, m-1\}$. The function returned from the CASE operation has a range of $\text{range}(G_i)$. 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 the 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 (represented by two Mdds: $F$ and $G$), the 3-valued operator $\max$, can be expressed as:

$$\max(F, G) = \text{CASE}(F, \text{CASE}(G, 0, 1, 2), \text{CASE}(G, 1, 1, 2), \text{CASE}(G, 2, 2, 2))$$

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: $\neg(F) = \text{CASE}(F, 1, 0)$
- 3-valued: $\neg(F) = \text{CASE}(F, 2, 1, 0)$
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 a given operator, the maximum number of logical values over the range of all operands is $p_{\text{max}}$.

- The $p_{\text{max}}$-valued MDDs constructed for each n-ary operation (not, min, max, etc.), using the variables $0, 1, \text{ etc.}$, 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: $\text{id}(x) < \text{id}(y)$.

- Let be $'0', '1', '2', \text{ etc.}$, a reserved variable names and $\text{id}(0) = 0, \text{id}(1) = 1, \text{ etc.}$. If F is an MDD, $F.\text{id}$ denotes the id of root variable of F ($F.\text{id} = -1$ 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 children. We note that MDD by: $\langle v, G_0, G_1, \ldots, G_{p-1} \rangle$.

Suppose, for example, a binary operation that can be implemented by “applying” the corresponding formal-mdd over the two MDDs representing some $p_{\text{max}}$-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(fOP.\text{id},
Apply(OP0, f0, f1),
\ldots,
Apply(OP_{p_{\text{max}}-1}, f0, f1))
```

Here, OPi represent child(OP), i = 0,1, \ldots, \text{p_{max}-1}. For example, consider F0 and F1 with the MDDs from figure 3 (with $p_{\text{max}} = 3$) 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:

\[ \text{OP} = \langle S_0, \text{OP0, OP1, OP2} \rangle, \]
where: \[ \text{OP0} = \langle S_1, 0, 1, 2 \rangle, \quad \text{OP1} = \langle S_1, 1, 1, 2 \rangle, \quad \text{OP2} = 2 \]

From figure 3a:

\[ F_0 = \langle X, 0, y, 0, 1 \rangle, \quad F_1 = \langle Z, 2, 0 \rangle \]

We have also \( \text{OP}.id = 0 \), \( \text{OP0}.id = \text{OP1}.id = 1 \), so that:

\[
\text{Apply}(\text{OP}, F_0, F_1) = \text{CASE}(F_0, \text{CASE}(\text{OP0}, F_0, F_1), \text{CASE}(\text{OP1}, F_0, F_1)) = \text{CASE}(F_0, \text{CASE}(F_1), 2) \]

Finally, from the \text{CASE} definition we have the result:

\[ \langle X, Y, Z, 2, 0 \rangle \]

### 3 Mapping p-valued MDD into n-valued MDDs

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

\[ F_p : \mathbb{P}^m \rightarrow \mathbb{P}, \text{ where logic set } \mathbb{P} = \{0, 1, \ldots, p-1\} \]

Suppose the \( F_p \) 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 \( \mathbb{N} = \{0, 1, \ldots, n-1\} \), so that \( n < p \).

We want to encode the \( F_p \) by \( a \) functions, where \( a = \lceil \log_p n \rceil \), and these functions are n-valued. In fact we need to construct \( a \) n-valued MDDs of these functions.

Let be \( F_n^a : \mathbb{N}^a \rightarrow \mathbb{N} \), where \( \mathbb{N} = \{0, 1, \ldots, n-1\} \), and \( a = \lceil \log_p n \rceil \).

We shall use below the notations:

- \( \text{MDD}_F \) – The Mdd of function \( F \)
- \( \text{MDD}_F_n \) – Mdd of function having n-valued inputs(encoded) and p-valued output
- \( \text{MDD}_F_0, \text{MDD}_F_1, \ldots \) - the Mdds for functions that encode \( F \)
- \( \text{COD}_F \) - 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:

\[ x_0, x_1, \ldots, x \lceil \log_p n \rceil \text{as in figure 4}. \]

The resulting MDD is \( \text{MDD}_F_n \).

**Step 2.**

Choose the coding scheme for \( F \) and construct \( \text{COD}_F \). In step 2, we’ll choose the coding for \( F \) using \( F_0, F_1, \ldots, F \lceil \log_p n \rceil \), as the coding variables.

**Step 3.**

Based on \( \text{COD}_F \), built \( \text{COD}_F_0, \text{COD}_F_1, \ldots \), the “formal-mdds”(unary operators for \text{Apply})

used for \( \text{MDD}_F_i \) building:

\[ \text{MDD}_F_0 = \text{Apply}(\text{COD}_F_0, \text{MDD}_F_n) \]

\[ \text{MDD}_F_1 = \text{Apply}(\text{COD}_F_1, \text{MDD}_F_n) \]

### 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](image)

![Fig 5 b) MDD_Q3,4](image)
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 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 shown in fig 6c. Literal generator for 3-valued input is in fig 6d. We want to obtain the binary functions (\( h_0 \) and \( h_1 \) in fig 6a) As is stated in[4] between the binary literals \( Ax \), \( Bx \), \( Cx \) the next relations exists:

\[
\begin{align*}
Ax + Bx &= Ax + Cx + Bx + Cx = 2 \\
\text{not} Ax &= Bx - Cx ; \\
\text{not} Bx &= Ax - Cx ; \\
\text{not} Cx &= Ax - Bx
\end{align*}
\]

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 \( h_0 \) is obtained and from MDD\_F1 the expression for \( h_1 \):

<table>
<thead>
<tr>
<th>Table II Sample State Transition Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table III Karnaugh for next-state function</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q )</td>
</tr>
<tr>
<td>( v )</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

\( Ax + Bx = Ax + Cx = Bx + Cx = 2 \)
\( \text{not} Ax = Bx - Cx ; \quad \text{not} Bx = Ax - Cx ; \quad \text{not} Cx = Ax - Bx \)
Using relations between literals, the positive form for $h_1$ can be obtained:

$$h_1 = Ax \cdot Ay + Ax(By \cdot Cy) \cdot (Ax \cdot Cx) = Ax \cdot Ay + Ax \cdot Cx \cdot By \cdot Cy$$

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 into some certain multi-valued physical circuits. 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.

References:


