Multilevel Logic Optimization of Very High Complexity Circuits

L. Burgun, N. Dictus, A. Greiner, E. Prado Lopes* and C. Sarwary

Laboratoire MASI/CAO-VLSI - Université Pierre et Marie Curie
4, place Jussieu, 75252 Paris Cedex 05 FRANCE

Abstract

The traditional approaches for multilevel logic optimization involve representing Boolean functions in Sum-of-Product forms that are minimized and then factorized in multilevel expressions. We have investigated an alternative approach called graphic synthesis that is based on a set of Reduced and Ordered BDDs, namely a multi-ROBDD for representing Boolean functions. This approach allows the optimization of very high complexity circuits that cannot be synthesized by classical algorithms without applying drastic restrictions. We propose an algorithm for generating a multilevel expression from a BDD and a procedure that optimizes a multi-ROBDD by exploiting Don’t Cares and by searching an input variable ordering for minimizing the final cost of a Boolean network in terms of number of literals. We present results over a range of very high complexity circuits (up to 5,000 gates).

1 Introduction

In the last years, several methods for multilevel logic optimization have been published and implemented in CAD tools [4]. Most of these tools start the synthesis process from Sum-of-Product forms and they use Algebraic or Boolean Division factoring techniques. Some authors have already shown the limitations of such techniques, in particular for symmetric [11] or large circuits [1]. Two methods based on intermediate representations corresponding to Decision Diagrams have been more recently proposed [9] [10]. Decision Diagrams provide built-in mechanisms for simplifying and factorizing multilevel expressions. The multilevel logic optimization then consists in constructing Decision Diagrams from the initial description and translating these graphs into a set of multilevel expressions corresponding to a Boolean network. Those methods that we call graphic synthesis in contrast to the algebraic synthesis methods, have not succeeded due to their complexity and their lack of efficiency.

In this paper, we present a ROBDD based approach for multilevel logic optimization that makes it possible to handle very high complexity circuits. Unlike Functional Decision Diagrams [10], the generated expressions use xor but also or and and operators allowing the technology mapping on any standard cells library. ROBDDs provide easier manipulations (for instance the re-ordering of the input variables) than ITEs [9]. In addition, complex decision functions can be rapidly calculated from ROBDDs while keeping the canonicity properties. Finally, ROBDDs allow the implementation of the procrastination algorithm proposed by Johannsen [8] by raising the delayed input variables within the multi-ROBDD so that these inputs come through as few logic operators as possible.

This paper is organized as follows. Section 2 of this work describes the terminology used in this paper. Section 3 shows the architecture of the multilevel logic optimizer. Section 4 presents an algorithm that produces a multilevel expression from a ROBDD. Section 5 shows how to minimize the multi-ROBDD representation in order to generate a well minimized Boolean network. Section 6 presents a comparison with optimization techniques implemented in Sis [14] and Asyl [1].

2 Terminology and definitions

A Completely Specified Boolean Function (CSBF) \( F \) is defined by \( F : B^n \rightarrow B \) with \( B = \{0, 1\} \). We represent the Boolean space \( B^n \) by a \( n \)-dimensional cube in which each point is called a minterm. A minterm is a vector \([v_1, \ldots, v_n]\) in \( B \) associated with the input variables \((x_1, \ldots, x_n)\). A minterm \( M_i = [v_1, \ldots, v_n] \in F \) if and only if \( F(M_i) = F(v_1, \ldots, v_n) = 1 \). A CSBF \( F \) is included \((\subseteq)\) in another CSBF \( G \) if and only if \( \forall M_i, F(M_i) = 1 \implies G(M_i) = 1 \). Then we have \( F \subseteq G \iff F \land G = G \).

An Incompletely Specified Boolean Function (ISBF) \( H \) is defined by \( H : B^n \rightarrow Y \) with \( Y = \{0, 1, *.\} \). It may be expressed as a triplet of CSBFs \((H_{on}, H_{de}, H_{off})\) where \( H_{on} = \{ M_i | H(M_i) = 1 \}, H_{de} = \{ M_i | H(M_i) = * \}, H_{off} \)

---

* Supported by CAPES-Brazil

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

(c) 1994 ACM 0-89791-687-5/94/0009 3.30
= \{ M_i | H(M_i) = 0 \}. A CSBF F is a cover of an ISBF H if and only if: \( H_{on} \subseteq F \) and \( F \subseteq H_{on} + H_{dc} \). Given two CSBFs \( F \) and \( G \) we construct a new ISBF \( H = F(G^*) \), such as: \( H_{on} = F \cdot G' \), \( H_{dc} = G \), \( H_{xor} = F' \cdot G' \). For any ISBF \( H \) we have: \( H = H_{on}(H_{dc} + g) \).

A Binary Decision Diagram (BDD) is a directed acyclic graph representing a CSBF decomposed by the Shannon formula. The graph has two special vertices called terminal vertices that represent the functions 0 and 1. All other vertices are called non-terminal vertices and are labelled with a Boolean variable. A non-terminal vertex is called an atomic vertex if its sons are terminal vertices. A non-terminal vertex is called a semi-atomic vertex if one of its sons is a terminal vertex. A BDD is a canonical form for a CSBF if the variable ordering is total (OBDD) and if each vertex represents a distinct CSBF (ROBDD). A Multi-ROBDD is a set of shared ROBDDs using the same ordering and it represents a multiple output CSBF. A Boolean network can be represented by a multi-ROBDD by linking a ROBDD with each CBSF of the nodes belonging to the Boolean network.

3 Overview of the multilevel logic optimizer

The starting point of our multilevel optimization system is a VHDL behavioral data-flow description from which we extract a combinational Boolean network as defined in [4].

3.1 Construction of the multi-ROBDD

The first phase consists in constructing the multi-ROBDD. One of the key points of this approach is to define the intermediate variables that we have to keep. Our system proposes two modes corresponding to either a local optimization or a global optimization. A local optimization keeps most of the intermediate variables so that we hold the initial architecture. In this case, the input variables of the multi-ROBDD are either primary input variables or intermediate variables. This mode is well-suited for optimizing arithmetic circuits such as ISCAS benchmarks. A global optimization collapses the circuits by removing most of the intermediate variables and consequently the outputs may be directly expressed in function of the primary inputs. This mode allows the optimization of FSMs or random circuits. A heuristic algorithm defines an initial input variable ordering as proposed in [12, 13]. A multi-ROBDD representing both the on-set and the dc-set of each node of the Boolean network is built using this initial ordering.

3.2 Optimization of the multi-ROBDD

The multi-ROBDD is then minimized by dynamically re-ordering input variables and by exploiting the Don’t Care conditions. The re-ordering phase is driven by a specific cost function which evaluates the number of literals of the Boolean network after optimization. Then we perform a Don’t Care minimization by applying an operator called graphMin that is an advanced version of the restrict operator proposed by Coudert et al. [6].

3.3 Optimization of the Boolean Network

Since the multi-ROBDD has been minimized, we reconstruct an optimized Boolean network from the initial Boolean network (that may have been collapsed) and the minimized multi-ROBDD. Two methods are possible. The former that we call Direct Graphic Synthesis consists in adding a node in the Boolean network for each vertex of the multi-ROBDD having at least two fathers. This method often yields poor results because it does not allow a good simplification of the expressions and consequently of the Boolean network. The latter called Indirect Graphic Synthesis consists of two distinct phases that are the synthesis of all the multilevel expressions and the reduction of these expressions into the Boolean network. Due to the canonical properties of ROBDDs, the generated expressions are normalized and the reduction can be easily done by identifying and reducing the equivalent sub-expressions. Our system implements this latter method.

4 Translating a ROBDD into a multilevel Boolean expression

We now detail the algorithm for creating a multilevel expression with and, or, not and xor operators from a ROBDD. We show in particular the differences with the print-factored-form function proposed by Karplus [9]. Given a ROBDD representing an CSBF \( F \), the algorithm proceeds from the root of this ROBDD downwards as the apply algorithm [5]. If the root is a terminal vertex such as zero or one, the function is obviously expressed as \( F = 0 \) or \( F = 1 \) (1). We now calculate a multilevel expression corresponding to the vertex associated with an input variable \( a \). Let \( L(Low) \) and \( H(High) \) be the cofactors of \( F \) by \( a \) \((L = F(a = 0) \) and \( H = F(a = 1))\):

- Atomic vertices : two possible cases (2)

\[
F = \begin{cases} a' & \text{if } a = 0 \\ a & \text{if } a = 1 \end{cases}
\]


- Semi-atomic vertices : four possible cases (3)

\[
\begin{array}{cc}
1 & 0 \\
0 & 1
\end{array}
\]
If the vertex does not possess any of these ending properties the general form is: \( F = a'.L + a.H \)

Then our algorithm tries to optimize this form by using the following properties:

- If \( H = L' \) then \( F = a \oplus L \) (4)
- If \( H \subseteq L \) then \( F = a'.L + H \) (see [9]). In addition \( L \) can be simplified because the minterms in \( H \) may be eliminated from \( L : F = a'.L_{\text{min}\!\!\!H} + H \) (5) Where \( L_{\text{min}\!\!\!H} \) is a cover of \( L(H^*) \). In Section 5.1 we will describe how to calculate \( L_{\text{min}\!\!\!H} \).
- Similarly, if \( L \subseteq H \) then \( F = L + a.H_{\text{min}\!\!\!L} \) (6)

If none of these rules is applied, we may rewrite the function by extracting the consensus \( H.L : F = a'.L + a.H + H.L \). The minterms of \( H.L \) may be eliminated from \( H \) and \( L \) so that we have: \( F = a'.L_{\text{min}\!\!\!H} + a.H_{\text{min}\!\!\!L} + H.L \). Take \( L_1 = L_{\text{min}\!\!\!H}(H.L) \) and \( H_1 = H_{\text{min}\!\!\!L}(H.L) \), the minterms appearing simultaneously in \( L_1 \) and \( H_1 \) may be eliminated from \( H \) and \( L \) and we have finally:

- \( F = a'.L_1 + a.H_1 + (H.L)_{\text{min}\!\!\!L(H.L)} \) (7)

This solution is only kept when it enhances the general form in terms of number of literals. The pseudo-code of this algorithm is as follows:

```plaintext
function graph2Expr(F)
{
    /* Let \( a \) be the variable of the root of \( F \) */
    if F = 0 then return 0; /* 1 */
    if F = 1 then return 1; /* 1 */
    if H = 0 and L = 1 then return a'; /* 2 */
    if H = 1 and L = 0 then return a; /* 2 */
    if L = 0 then return a'.graph2Expr(H); /* 3 */
    if L = 1 then return a + graph2Expr(H); /* 3 */
    if H = 0 then return a'.graph2Expr(L); /* 3 */
    if H = 1 then return a + graph2Expr(L); /* 3 */
    if H.L = 0 and H+L = 1 then return
        a \( \oplus \) graph2Expr(L); /* 4 */
    if H \( \subseteq \) L then return
        a'.graph2Expr(graphMin(L,H',H)) + graph2Expr(H); /* 5 */
    if L \( \subseteq \) H then return
        graph2Expr(L) + a.graph2Expr(graphMin(H',L,L)); /* 6 */
    if H.L \( < \) 0 then {
        L_1 = graphMin(L,H',H.L); H_1 = graphMin(H,H',L,L);
        E_1 = a'.graph2Expr(L_1) + a.graph2Expr(H_1) +
            graph2Expr(graphMin(H,H(L_1),H_1,L,L)); /* 7 */
        E_2 = a'.graph2Expr(L) + a.graph2Expr(H);
    }
}
```

If \( \text{cost}(E_1) > \text{cost}(E_2) \) then return \( E_2 \) else return \( E_1 \).

Algorithm of the graph2Expr function

**Example**: Consider the function \( F = a'.b'.c'.d + a'b'.c'.d' + a'.b'.c.d + a'b'.c.d' + a.b'.c.d + a'.b.c.d' + a.b'.c.d' \) and its ROBDD with the ordering \([a, b, c, d] \).

At the first level \( H \subseteq L \) and the rule (5) gives: \( F = a'.L_{\text{min}\!\!\!H} + H \). The algorithm is then recursively applied to the subgraphs representing \( L_{\text{min}\!\!\!H} \) and \( H \). Rule (2) is applied to \( L_{\text{min}\!\!\!H} \) and gives the expression \( L_{\text{min}\!\!\!H} = \beta' \), and rule (4) is applied to \( H \) and gives \( H = c \oplus d \). Finally we obtain: \( F = a'.\beta' + c \oplus d \)

In fact the implemented algorithm is more complex because it searches complex decision functions (as the if-part of the ITE [9]) to replace the single variable \( a \). A complex decision function is incrementally calculated by exploring the sons of the initial vertex.

**Proposition**: Given a ROBDD representing the CSBF \( F \) and \( H \) the high son of its root, if there exists a path from the root to a vertex denoting the CSBF \( L_n \), such that these vertices are labelled by the variables \((a_0,a_1,\ldots,a_n)\) and they have one of their son equal to \( H \), then there exists an OR-decision expression \( D_n \) such that \( F \) can be expressed as follows:

\( F = D_n'.L_n + D_n.H \) where \( D_n = a_0 + V_1 + \ldots + V_n \), \( V_i = a_i \) or \( a_i' \), \( i \in [1,n] \)

The following figure shows the ROBDD corresponding to the general case \((x_i,x'_i)\) denoting the polarity of \( V_i \).
A dual proposition can be done about the low son \( L \) and a \textit{AND}-decision expression may be found. This pattern matching procedure is performed iteratively and a multi-level decision expression may be calculated.

\textbf{Example} : Take \( F = a.b.d + c.d + a'.e + b'.c'.e \). A decision expression \( D1 \) is firstly extracted : \( D1 = a.b \) and \( F = D1'.(c.d + e'.c + d) \).
The variable \( c \) is added and we obtain : \( D2 = a.b + c \) and \( F = D2'.e + D2.d \) as shown in the following figure.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig}
\caption{Example of re-ordering in multi-ROBDD}
\end{figure}

\textbf{Remarks} : The rewriting rules in Graph2Expr are applied regardless of the decision expression (the single variable \( a \) may be replaced by a complex expression).

\section{5 \ Multi-ROBDD minimization}

\subsection{5.1 \ Re-ordering the Multi-ROBDD}

Several methods have been proposed for re-ordering a multi-ROBDD [7]. In our approach, we need to find a good ordering and therefore we have used a gradual improvement method as described in [2].

We now investigate the impact of the cost function during the re-ordering phase. The number of vertices could be a good cost function. However, a good solution in terms of the number of ROBDD vertices could be a good ordering and therefore we have used a gradual improvement method as described in [2].

Table 1 shows results obtained for some MCNC benchmarks using a re-ordering procedure minimizing either the number of vertices (\# Vertex based) or our specific cost function (Synthesis based). For some circuits such as \textit{misex1} or \textit{ttt2}, the number of vertices based minimization yields poor results in terms of number of literals.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{circuits} & \textbf{\# Vertex based} & \textbf{Synthesis based} \\
& \textbf{\# literals} & \textbf{\# vertices} & \textbf{\# literals} & \textbf{\# vertices} \\
\hline
\textit{misex1} & 77 & 39 & 67 & 48 \\
\textit{vg2} & 106 & 87 & 100 & 89 \\
\textit{alu2} & 323 & 186 & 292 & 197 \\
\textit{misex2} & 135 & 96 & 129 & 97 \\
\textit{cht} & 172 & 122 & 171 & 131 \\
\textit{ttt2} & 214 & 130 & 165 & 145 \\
\textit{x4} & 468 & 437 & 395 & 467 \\
\textit{apex3} & 1689 & 965 & 1597 & 1351 \\
\hline
\end{tabular}
\caption{Re-ordering Experiment on MCNC Circuits}
\end{table}

\subsection{5.2 \ Don’t Care exploitation}

In this section, we propose an operator to rapidly generate a minimized ROBDD from two ROBDDs representing an ISBF. Let us suppose that \( F1 \) and \( F2 \) are two CSBFs such that \( F2 \subseteq F1' \). \( F1 \) and \( F2 \) are respectively the on-set and dc-set of the ISBF \( F1(F2*) \). We search for a CSBF \( F1_{minF2} \) covering \( F1(F2*) \) with a minimal cost (in terms of number of ROBDD vertices). Several situations may occur depending on the ROBDDs of \( F1 \) and \( F2 \):

\begin{itemize}
\item \( F2 = 0 \Rightarrow F1_{minF2} = F1 (1) \)
\item \( F1 = 0 \) or \( F1 = 1 \Rightarrow F1_{minF2} = F1 (2) \)
\end{itemize}

Let \( a \) be the variable of the root of \( F1 \) and \( b \) the variable of the root of \( F2 : F1 = a'.L1 + a.H1 \) and \( F2 = b'.L2 + b.H2 \)

\begin{itemize}
\item If \( a = b \), we have five possible cases :
\begin{enumerate}
\item \( H2 = 1 \Rightarrow H1 = 0 \) (because \( F2 \subseteq F1' \)), we have : \( F2 = a + L2 \) and \( F1 = a'.L1 \) so \( F1_{minF2} = L1_{minL2} \) \( (3) \)
\item Similarly if \( L2 = 1 \), we get \( F1_{minF2} = H1_{minH2} \) \( (4) \)
\item if \( L1 \subseteq H2 \) and \( H1 = 0 \), the variable \( a \) can be eliminated from \( F1_{minF2} \) by making its sons equal to \( L1 \) and we have : \( F1_{minF2} = L1 \) \( (5) \)
\item Similarly if \( H1 \subseteq L2 \) and \( L1 = 0 \), we get : \( F1_{minF2} = H1 \) \( (6) \)
\item if \( H1 \subseteq L1 + L2 \) and \( L1 \subseteq H1 + H2 \), we generalize the two last rules and we obtain : \( F1_{minF2} = (H1 + L1)_{minH2,L2} \) \( (7) \)
\end{enumerate}
\end{itemize}

Finally, if none of the above conditions are applied, we have : \( F1_{minF2} = a'.L1_{minL2} + a.H1_{minH2} \) \( (8) \)
The approach described above have been implemented in C language in the LaX [3] synthesis system as part of the ALLIANCE CAD package. In this section, we compare our results with two other methods. The first one is the Boolean factorization implemented in Sis [14] (boolean.script) and the second one is the lexicographic factorization implemented in Asyl [1]. In all cases, the goal is the minimization of the circuit area. We have synthesized high complexity circuits coming from Phillips (Imec) and Esprit Project (Hyeti). These combinational circuits have been generated by FSM synthesizers and correspond to the transition and the output functions of Moore controllers (200-400 states and 500-1300 transitions). The number of literals of the initial Boolean network varies from 6246 (hyeti2) to 41088 (imec2). Three tables show the results obtained for those circuits after Boolean optimization, technology mapping and placing-routing respectively. Table 2 shows the results obtained for the first phase. The results are expressed in terms of number of literals in the optimized Boolean networks (#lit.) and the computing time (cpu) is given in minutes. The results have been obtained on a Sun Workstation (Sparc 10 - 85 MIPS - RAM 128 Mb).

### Table 2: Number of Literal Comparison

<table>
<thead>
<tr>
<th>circuits</th>
<th>Sis</th>
<th>Asyl</th>
<th>LaX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#lit.</td>
<td>cpu</td>
<td>#lit.</td>
</tr>
<tr>
<td>hyeti2</td>
<td>1784</td>
<td>12'</td>
<td>2251</td>
</tr>
<tr>
<td>imec8</td>
<td>2027</td>
<td>14'</td>
<td>2474</td>
</tr>
<tr>
<td>imec5</td>
<td>2596</td>
<td>50'</td>
<td>3169</td>
</tr>
<tr>
<td>imec7</td>
<td>3847</td>
<td>512'</td>
<td>5226</td>
</tr>
<tr>
<td>imec4</td>
<td>-</td>
<td>-</td>
<td>6433</td>
</tr>
<tr>
<td>imec3</td>
<td>-</td>
<td>-</td>
<td>7160</td>
</tr>
<tr>
<td>imec2</td>
<td>-</td>
<td>-</td>
<td>10441</td>
</tr>
</tbody>
</table>

The Boolean factorization (Sis) has been stopped for the largest circuits after 10 hours CPU. This confirms the limitation of the classical approach for optimizing high complexity controllers. As soon as the circuits are large, the CPU time and the memory usage increase drastically for the two factorization methods. In our approach, 90 percent of the computing time is dedicated to calculate a good input ordering of the multi-ROBDD. It is important to notice that the computing time and the memory space have a linear dependence on the circuit complexity, and this is not the case for the factorization approach.

The technology mapping has been done by the LaX mapper and we have used the IDPS CMOS standard cell library (about 30 combinational cells). Table 3 shows the number of gates after the standard cell mapping phase (#gates) and
the maximal number of gates between the primary inputs and the primary outputs (lev.).

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Sis</th>
<th>Asyl</th>
<th>LaX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#gates</td>
<td>lev.</td>
<td>#gates</td>
</tr>
<tr>
<td>hyeti2</td>
<td>960</td>
<td>14</td>
<td>1212</td>
</tr>
<tr>
<td>imec8</td>
<td>1094</td>
<td>14</td>
<td>1368</td>
</tr>
<tr>
<td>imec5</td>
<td>1422</td>
<td>15</td>
<td>1728</td>
</tr>
<tr>
<td>imec7</td>
<td>2069</td>
<td>14</td>
<td>2720</td>
</tr>
<tr>
<td>imec4</td>
<td>-</td>
<td>-</td>
<td>3415</td>
</tr>
<tr>
<td>imec3</td>
<td>-</td>
<td>-</td>
<td>3683</td>
</tr>
<tr>
<td>imec2</td>
<td>-</td>
<td>-</td>
<td>5605</td>
</tr>
</tbody>
</table>

Table 3: Number of gates Comparison

The placing-routing phase has been done by a commercial tool (COMPASS). Area is given in \( \text{mm}^2 \) for a 1.2 micron technology and the routing factor is computed by the following formula: \( RF = \frac{\text{Area}_{\text{Wiring}}}{\text{Area}_{\text{Gates}}} \).

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Sis</th>
<th>Asyl</th>
<th>LaX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>area</td>
<td>RF</td>
<td>area</td>
</tr>
<tr>
<td>hyeti2</td>
<td>4.54</td>
<td>2.9</td>
<td>5.33</td>
</tr>
<tr>
<td>imec8</td>
<td>6.38</td>
<td>3.6</td>
<td>7.19</td>
</tr>
<tr>
<td>imec5</td>
<td>8.54</td>
<td>3.7</td>
<td>10.73</td>
</tr>
<tr>
<td>imec7</td>
<td>14.27</td>
<td>4.1</td>
<td>21.62</td>
</tr>
<tr>
<td>imec4</td>
<td>-</td>
<td>-</td>
<td>28.87</td>
</tr>
<tr>
<td>imec3</td>
<td>-</td>
<td>-</td>
<td>32.65</td>
</tr>
<tr>
<td>imec2</td>
<td>-</td>
<td>-</td>
<td>73.38</td>
</tr>
</tbody>
</table>

Table 4: Area Comparison

Table 4 shows that our approach offers the best results in area for the largest circuits. We note that the routing factor is smaller than achieved with the lexicographic method. For these two methods, it is clear that the variable ordering enforces a scheme for placing-routing. However, an analysis of the resulting gate networks shows that the gates have smallest fanouts in our approach (because of a weak reduction of the Boolean network) so that makes placing-routing easier.

7 Conclusions

We have investigated an approach taking advantages both the form and the properties of the ROBDDs for optimizing Boolean networks. The main advantage of this approach is the fact that we can rapidly optimize very high complexity circuits. Thus, we have shown that this approach is well suited for large FSM circuits. In contrast to the algebraic synthesis, it is possible to contemplate optimizing several circuits independently of the initial architecture of the Boolean network. Future works will investigate improved algorithms for automatically keeping the intermediate variables.

References