Communication Based Multilevel Synthesis for Multi-output Boolean Functions

Paul Molitor
Department of Computer Science
Humboldt-Universität zu Berlin
D 10099 Berlin, FRG

Christoph Scholl
Department of Computer Science
Universität des Saarlandes
D 66041 Saarbrücken, FRG

Abstract

A multilevel logic synthesis technique for multi-output boolean functions is presented which is based on minimizing the communication complexity. Unlike the approaches known from literature [1, 5, 6, 8] which in the final analysis decompose each single-output function \( f_i \) of a multi-output function \( f = (f_1, \ldots, f_m) \) independently of the other single-output functions \( f_j \) \( (j \neq i) \) the approach presented in this paper gives special attention to the fact that there possibly exist some decomposition functions which can be used by different outputs during the decomposition of the single-output functions of \( f \). The benchmarking results (taken from 1991 MCNC multilevel logic benchmarks) which close the paper are promising.

1 Introduction

Most of the approaches attacking the multilevel logic synthesis problem use gate count as optimization criterion. This is based on the belief that gate count is a good estimator for layout area. However, in many cases, this criterion may not be the best estimator. Some recent papers [5, 6, 8] propose an approach different from the one addressed above. This approach to multilevel logic synthesis which originates from Ashenhurst [1], Curtis [3], Hotz [4], and Karp [7] is based on minimizing communication complexity. The methods used to reduce communication complexity employ functional decomposition. A decomposition of a boolean function \( f : \{0,1\}^n \rightarrow \{0,1\} \) with respect to the input partition \( \{X, Y\} \) \( X = \{x_1, \ldots, x_p\}, \ Y = \{y_1, \ldots, y_q\}, \ X \cap Y = \emptyset, p + q = n \) is a representation of the form

\[
f(x_1, \ldots, x_p, y_1, \ldots, y_q) = g(\alpha_1(x_1), \ldots, \alpha_l(x), \beta_1(y_1), \ldots, \beta_r(y))
\]

for all \( (x_1, \ldots, x_p, y_1, \ldots, y_q) \in \{0,1\}^n \). \( \alpha_i \) and \( \beta_j \) are called decomposition functions of \( f \). \( g \) is called composition function.

With respect to a given input partition \( \{X, Y\} \), a single-output function \( f \) can be represented as a \( 2^n \times 2^n \) matrix \( M(f) \), the decomposition matrix of \( f \) or the chart of \( f \) with respect to \( \{X, Y\} \). Each row and column of \( M(f) \) is associated with a distinct assignment of values to the inputs in \( X \) and \( Y \), respectively, such that \( f(x, y) = M(f)[x, y] \) where \( M(f)[x, y] \) represents the element of \( M(f) \) which lies in the row associated with \( x \) and the column associated with \( y \). Then \( r \geq \lceil \log p_1 \rceil \) \( (s \geq \lceil \log p_2 \rceil) \) where \( p_1 \) \( (p_2) \) is the number of distinct row patterns (column patterns) in \( M(f) \). In the following we always assume \( r = \lceil \log p_1 \rceil \) and \( s = \lceil \log p_2 \rceil \). Of course the required number of decomposition functions depends strongly on the choice of the input partition \( \{X, Y\} \).

In many cases the efficiency of good realizations of boolean functions is based on a clever reuse of subcircuits. Our approach takes this observation into account by a special processing of multi-output functions\(^1\). The decomposition of a multi-output function \( f = (f_1, \ldots, f_m) : \{0,1\}^n \rightarrow \{0,1\}^m \) has the following form:

\[
f_i(x_1, \ldots, x_p, y_1, \ldots, y_q) = g_i(\alpha_{i1}(x), \ldots, \alpha_{i\ell}(x), \beta_{i1}(y), \ldots, \beta_{is}(y))
\]

for all \( 1 \leq i \leq m \). In order to reuse subcircuits our algorithm tries to choose as many decomposition functions \( \alpha_{ij} \) as possible as identical functions. Unlike other approaches our synthesis method doesn’t decompose each single-output function \( f_i \) independently of the other single-output functions \( f_j \) \( (j \neq i) \) and only tests whether some \( \alpha_{ij} \) are identical by accident. On the contrary our approach tries to compute common decomposition functions for different \( f_i \).

The computation of common decomposition functions for some single-output functions \( f_i \) requires a decomposition of all \( f_i \) with respect to the same input partition \( \{X, Y\} \). However we have to take into consideration that there possibly exist single-output functions \( f_i \) and \( f_j \) such that there does not exist an input partition good for both \( f_i \) and \( f_j \). That’s why we have to divide our algorithm into two steps: In the first step we partition \( \{f_1, \ldots, f_m\} \) into disjoint sets \( Y_1, \ldots, Y_m \) (see section 2). In the second step

\(^1\) Even if the original function \( f \) is a single-output function, applying functional decomposition recursively to the decomposition functions \( \alpha = (\alpha_1, \ldots, \alpha_l) \) and \( \beta = (\beta_1, \ldots, \beta_s) \) demands a generalization to multi-output boolean functions.
we decompose all single-output functions $f_i$ of the same set $Y_i$ with respect to the same input partition giving special attention to generate these functions in such a way that many can be used in the decomposition of different elements of $Y_i$ (see section 3).

2 Output/input partitioning

In this section we present a heuristic which performs the first step of our algorithm by partitioning $F = \{f_1, \ldots, f_m\}$ into disjoint sets $Y_1, \ldots, Y_k$. To each set $Y_i$, the algorithm assigns an input partition $\{X^{(k)}, Y^{(k)}\}$ which is 'near-optimal' for each $f_i \in Y_i$. The size $|X^{(k)}|$ of $X^{(k)}$ is given by a number $p^2$.

Let $IP_0$ be a subset of all possible input partitions $\{X, Y\}$ with $|X| = p$ computed in a preprocessing step. Furthermore, let $df_f(A)$ be the minimum number of decomposition functions required by a decomposition of $f_i$ with respect to the input partition $A \in IP_0$. We define $df_{\min} = \min \{df_f(A) ; A \in IP_0\}$ to be the minimum of these values $df_f(\cdot)$. An input partition $A \in IP_0$ is said to be near-optimal for $f_i$ if $df_f(A) - df_{\min} < \text{parameter} \cdot (n - df_{\min})$, where $0 < \text{parameter} \leq 1$ is given by the designer. Then, the following heuristic algorithm solves the output/input partitioning problem as defined above.

1. Let $u = 0$, $F = \{f_1, \ldots, f_m\}$.
2. For all $A \in IP_0$ and for all $f_i \in F$, compute $df_f(A)$, $df_{\min}$, and the difference $df_f(A) = df_f(A) - df_{\min}$ between the number of decomposition functions required by a decomposition of $f_i$ w.r.t. $A$ and the minimum number of decomposition functions required by a decomposition of $f_i$ w.r.t. $IP_0$.
3. If there is an $f_i \in F$ with $df_{\min} = n$, i.e., which cannot be decomposed with respect to any partition of $IP_0$ with less than $n$ decomposition functions, then let $Y_1 = \{f_i \in F ; df_{\min} = n\}$. $F = F \setminus Y_1$, $u = 1$. (These single-output functions will be decomposed by applying the well-known Shannon expansion which is a nontrivial decomposition\(^4\) for $n \geq 4$.)
4. Determine the input partition $A'$ such that $\sum_{f_i \in Y_i} df_f(A') L_i$ is minimal.
   (If $L = 1$ ($L = +\infty$), then $A'$ is the input partition
   \footnote{If $p$ is about $\Phi$, the partition scheme tends to lead to fast tree circuits. If $p$ is about 1 or $n - 1$, it tends to lead to smaller and slower circuits.}
\footnote{If parameter is chosen closer to 1, the probability for an input partition $A$ to be accepted as 'near-optimal' for a function $f_i$ increases, i.e., possibly $f_i$ is decomposed with respect to an input partition which doesn't lead to a minimal number of decomposition functions. On the other hand the size of the sets $Y_i$ then tends to be larger and we possibly have an increased potential for reusing subcircuits by choosing common decomposition functions for elements of $Y_i$.}
\footnote{A decomposition is said to be nontrivial if its composition function $g$ has less than $n$ inputs.}

for which the sum (maximum) of the deviations is minimal. $L$ is a parameter of the heuristic.)
5. Let $u = u + 1$, determine $Y_u$ which is the set of those single-output functions $f_i \in F$ for which $A'$ is near-optimal, i.e., $df_f(A') < \text{parameter} \cdot (n - df_{\min})$, and let $F = F \setminus Y_u$.
6. If $Y_u = \emptyset$, then let $Y_u = \{f_i \in F ; df_f(A')$ is minimal $\}$, $F = F \setminus Y_u$.
7. If $F \neq \emptyset$, then goto 4.

The running time of the algorithm strongly depends on the choice of subset $IP_0$ of all possible input partitions $\{X, Y\}$ with $|X| = p$. For the computation of $IP_0$ we use heuristics like iterative improvement (or simulated annealing) in the following way: start with any input partition $A_0$, form a new input partition $A_{k+1}$ by exchanging a pair of variables in $A_k$ as long as there is an $f_i \in F$ such that $df_f(A_k)$ can be decreased by such an exchange. Then, the set $IP_0$ of the input partitions which are processed by the algorithm above consists of the input partitions $A_0, A_1, \ldots, A_k, \ldots$ computed.

3 Common decomposition functions

In this section we describe the second step of our algorithm. Suppose that the first step gives us a set $Y_i$ of single-output functions which have to be decomposed with respect to the same input partition $A = \{X, Y\} \in IP_0$, which will be fixed in the following. In order to choose common decomposition functions in the decomposition of the elements of $Y_i$ we have to solve the following subproblem, which we denote by CDF (common decomposition functions problem).

**Given:** A set $Z = \{f_1, \ldots, f_l\}$ of single-output boolean functions\(^5\), $\{X, Y\}$ with $X = \{x_1, \ldots, x_p\}$, $Y = \{y_1, \ldots, y_q\}$, and a natural number $k \leq r$, $(\forall i \in [k])$, where $p_i$ is the number of distinct row patterns in the chart $M(f_i)$.

**Find:** $k$ single-output boolean functions $a_1, \ldots, a_k$, which can be used as decomposition functions of every single-output function $f_i$, $i = 1, \ldots, l$, such that there is a decomposition of the form

\[
  f_i(x, y) = g_i(a_1(x), \ldots, a_k(x), a_{1+1}(x), \ldots, a_{k+1}(x), \ldots, a_l(x), \ldots, a_{l+1}(x), \ldots)
\]

(\forall i \in [1, \ldots, l]).

CDF can be posed for $Y$ in an analogous manner.

Unfortunately, there is no great hope of finding an efficient algorithm solving CDF.

**Lemma 1 CDF is NP-complete.**

**Proof:** Reduction from the 3-partition problem \([9]\).

- We start by a theoretical result working towards a solution to CDF. It gives a condition necessary and sufficient that $k$ single-output functions $a_1, \ldots, a_k$ are common decomposition functions of $f_1, \ldots, f_l$. It is a generalization of

\footnote{Regard $Z$ as a subset of $Y_i$.}
of a lemma shown by Karp [7]. For this, we need the following notations. The rows of $M(f_k)$ induce a partition of $\{0,1\}^p$ into equivalence classes $K_{1}^{(k)}, \ldots, K_{p_{1}^{(k)}}^{(k)}$ such that every row pattern $v, v' \in \{0,1\}^p$ belong to the same class $K_{i}^{(k)}$ if and only if the two corresponding row patterns of $M(f_k)$ are equal.

Let $\theta^{(k)} : \{0,1\}^p \rightarrow \{1, \ldots, p_{1}^{(k)}\}$ be the function where $\theta^{(k)}(v)$ is the index $j$ of the class $K_{j}^{(k)}$ to which $v$ belongs. Furthermore, for all $a \in [0,1]^h$, let $S_{a}^{(k)}$ be the set $\{\theta^{(k)}(v) : \alpha_{1}, \ldots, \alpha_{h}(v) = a\}$ of those classes which contain a row mapped to $a$ by $\alpha_{1}, \ldots, \alpha_{h}$. Note that, for given $\alpha_{1}, \ldots, \alpha_{h}$, $S_{a}^{(k)}$ and $S_{a'}^{(k)}$ need not to be disjoint for $a \neq a'$, and that $|S_{a}^{(k)}|$ equals the number of distinct row patterns of $M(f_k)$ mapped to $a$ by $\alpha_{1}, \ldots, \alpha_{h}$.

Let $\Delta(\alpha_{1}, \ldots, \alpha_{h})$ be defined as $\max\{|S_{a}^{(k)}| : a \in [0,1]^h\}$.

Lemma 2 $\alpha_{1}, \ldots, \alpha_{h}$ are decomposition functions of $f_{1}, \ldots, f_{j}$ with respect to $A$. i.e., there is a representation of $f$ of the form:

$$f(x,y) = g_{k}(\alpha_{1}(x), \ldots, \alpha_{h}(x), \alpha_{h+1}(x), \ldots, \alpha_{h}(x), \beta_{1}(y), \ldots, \beta_{h}(y))$$

If $k \in \{1, \ldots, l\}$ if and only if $\Delta(A, f_{k}, \{\alpha_{1}, \ldots, \alpha_{h}\}) \leq 2^{2^{h}-1}$. (V.4)

Proof: A decomposition of the above form exists if and only if $(\alpha_{1}, \ldots, \alpha_{h+1}, \ldots, \alpha_{h+2})$ assigns different values to rows of chart $M(f_k)$ with different row patterns ($\forall k$).

As $\alpha_{h+1}, \ldots, \alpha_{h}$ can produce at most $2^{2^{h}-1}$ different values, the statement of the lemma follows.

Then CDF can be solved by computing $\alpha_{1}, \ldots, \alpha_{h}$ by a branch and bound algorithm. The sets $S_{a}^{(k)}$ are constructed step by step. In the initialization phase, $\alpha_{1}, \ldots, \alpha_{h}(x)$ is set to undefined for all $x \in \{0,1\}^p$, and $S_{a}^{(k)}$ is set to the empty set for all $a$ and $k$. Each time we enter the main loop there is an $x \in \{0,1\}^p$ and a vector value $\in \{0,1\}^h$ such that $\alpha_{1}, \ldots, \alpha_{h}(x)$ is defined for all $v$ with $inf(v) < inf(x)$, and there is no extension of the present function table with $inf(\alpha_{1}, \ldots, \alpha_{h}(x)) < inf(value f)$ which does not violate the condition of lemma 2. In this step, we test whether the condition of lemma 2 is violated if $\alpha_{1}, \ldots, \alpha_{h}(x)$ is set to value $f$. If the condition is violated, we backtrack if $inf(value f) = 2^h - 1$, i.e., $value f = (1, \ldots, 1)$. If $inf(value f) < 2^h - 1$, enter the loop once again with value $f$ incremented by 1. The sets $S_{a}^{(k)}$ are updated in each step.

Integration of CDF in the synthesis tool We apply the branch and bound algorithm solving CDF in an heuristic manner in order to solve the multilevel synthesis problem for multi-output boolean functions.

Let $f = (f_{1}, \ldots, f_{m})$ be a multi-output function where each of the single-output functions $f_{j}$ has been decomposed with respect to the same input partition $A \in IP_{p}$.

Table 1: Comparison between our tool mulop and factorII with respect to the number of gates used.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of inputs</th>
<th>Number of outputs</th>
<th>No. of gates</th>
<th>factorII</th>
<th>mulop</th>
<th>ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>9symm1</td>
<td>9</td>
<td>1</td>
<td>75</td>
<td>38</td>
<td>1.97</td>
<td></td>
</tr>
<tr>
<td>cm138a</td>
<td>6</td>
<td>8</td>
<td>21</td>
<td>21</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>cm151a</td>
<td>12</td>
<td>2</td>
<td>37</td>
<td>43</td>
<td>0.86</td>
<td></td>
</tr>
<tr>
<td>cm162a</td>
<td>14</td>
<td>5</td>
<td>80</td>
<td>44</td>
<td>1.82</td>
<td></td>
</tr>
<tr>
<td>cm163a</td>
<td>16</td>
<td>5</td>
<td>47</td>
<td>37</td>
<td>1.27</td>
<td></td>
</tr>
<tr>
<td>cm82a</td>
<td>5</td>
<td>3</td>
<td>18</td>
<td>16</td>
<td>1.13</td>
<td></td>
</tr>
<tr>
<td>cm16</td>
<td>16</td>
<td>4</td>
<td>33</td>
<td>29</td>
<td>1.14</td>
<td></td>
</tr>
<tr>
<td>decode</td>
<td>5</td>
<td>16</td>
<td>31</td>
<td>32</td>
<td>0.97</td>
<td></td>
</tr>
<tr>
<td>f15m</td>
<td>8</td>
<td>10</td>
<td>107</td>
<td>63</td>
<td>1.70</td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td>10</td>
<td>7</td>
<td>65</td>
<td>47</td>
<td>1.38</td>
<td></td>
</tr>
<tr>
<td>z4m1</td>
<td>7</td>
<td>4</td>
<td>25</td>
<td>25</td>
<td>1.00</td>
<td></td>
</tr>
</tbody>
</table>

First, we apply CDF to compute (by binary search) a maximum number $k$ of common decomposition functions of $f_{1}, \ldots, f_{m}$. Then, CDF is applied to subsets of $(f_{1}, \ldots, f_{m})$ beginning with subsets of size $m - 1$, $m - 2$ down to size 2. During this iteration, decomposition functions already found for $f_k$ are used in the decomposition of $f_k$ in any case in order to reduce the running time of the multilevel synthesis tool. Note that the branch and bound algorithm can be generalized in a canonical manner for the case that some of the decomposition functions $\alpha_{i}^{(k)} (i > k)$ are already predetermined.

Obviously, we have not to consider every subset of $(f_{1}, \ldots, f_{m})$. First of all, if $g_{k}$ decomposition functions are already found for function $f_{k}$, we can omit the remaining subsets containing $f_{k}$. Furthermore, it often happens that there is no common decomposition function for two single-output functions $f_{i}$ and $f_{j}$. Consequently, in this case, we have not to consider subsets containing $f_{i}$ and $f_{j}$. Therefore, in a first step, for all $f_{i}, f_{j}$, we test (by dynamic programming) whether $f_{i}$ and $f_{j}$ have a common decomposition function at all.

4 Benchmark results

Several examples of the 1991 MCNC multilevel logic benchmark set were synthesized to compare factor, factorII [5, 6] and misII [2] to our tool, which we will call mulop in the following.

First, we compared our tool mulop to factor and factorII. We ran the experiments with the technology mapping used in [6]. Since the quality of the layouts synthesized by factorII approximately equals the quality of the layouts synthesized by factor (see [6]), table 1 only reports the results of the comparison of mulop and factorII. For every circuit, the entries of the table correspond to the minimal number of gates obtained by the synthesis tools. Compared to factorII, our approach generates realizations with a smaller (or equal) number of gates for almost all circuits considered.

Table 2 shows the comparison between mulop and misII with respect to the number of gates, the cell area, and...
### Table 2: Comparison between `mulop` and `misII` with respect to gate count, cell area, layout size, and signal delay.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No. of gates</th>
<th>Sum of the cell areas</th>
<th>Layout size</th>
<th>Signal delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>misII</code></td>
<td><code>mulop</code></td>
<td>ratio</td>
<td><code>misII</code></td>
</tr>
<tr>
<td>9symm1</td>
<td>175</td>
<td>40</td>
<td>4.38</td>
<td>42636</td>
</tr>
<tr>
<td>C17</td>
<td>7</td>
<td>6</td>
<td>1.17</td>
<td>16568</td>
</tr>
<tr>
<td>cm138a</td>
<td>22</td>
<td>20</td>
<td>1.10</td>
<td>59280</td>
</tr>
<tr>
<td>cm151a</td>
<td>26</td>
<td>48</td>
<td>0.54</td>
<td>57545</td>
</tr>
<tr>
<td>cm152a</td>
<td>24</td>
<td>34</td>
<td>0.70</td>
<td>52896</td>
</tr>
<tr>
<td>cm162a</td>
<td>33</td>
<td>46</td>
<td>0.72</td>
<td>93480</td>
</tr>
<tr>
<td>cm163a</td>
<td>32</td>
<td>38</td>
<td>0.84</td>
<td>82800</td>
</tr>
<tr>
<td>cm82a</td>
<td>13</td>
<td>13</td>
<td>1.00</td>
<td>41304</td>
</tr>
<tr>
<td>cm85a</td>
<td>40</td>
<td>42</td>
<td>0.95</td>
<td>98496</td>
</tr>
<tr>
<td>cmh</td>
<td>48</td>
<td>24</td>
<td>2.00</td>
<td>112480</td>
</tr>
<tr>
<td>decode</td>
<td>32</td>
<td>31</td>
<td>1.03</td>
<td>72352</td>
</tr>
<tr>
<td>f51m</td>
<td>123</td>
<td>64</td>
<td>1.92</td>
<td>289560</td>
</tr>
<tr>
<td>majority</td>
<td>9</td>
<td>9</td>
<td>1.00</td>
<td>23560</td>
</tr>
<tr>
<td>parity</td>
<td>15</td>
<td>15</td>
<td>1.00</td>
<td>61560</td>
</tr>
<tr>
<td>x2</td>
<td>45</td>
<td>38</td>
<td>1.18</td>
<td>163208</td>
</tr>
<tr>
<td>z4m1</td>
<td>44</td>
<td>20</td>
<td>2.20</td>
<td>111264</td>
</tr>
</tbody>
</table>

5 Conclusion

We have presented a multilevel logic synthesis tool based on communication complexity which eliminates the drawbacks of similar approaches known from literature with respect to multi-output functions. Our method consists in two steps: output/input partitioning and constructions of decomposition functions while paying special attention that many of them can be used in the realization of different outputs.

The benchmarking results comparing our tool to `misII`, `factor` and `factorII` are promising both in respect to gate count and circuit delay.

The running time of the tool itself (measured on a SPARC2) was much better than that of `factor` without however coming up to the excellent running time of `factorII`. Here, we are investing further work, especially, using BDDs in the implementation of our synthesis tool.

---

8The layouts were generated by `wolfe` which is integrated in `actools`.

9The 2-input gates are taken from `stdcell2_2.genlib` available in `actools`.

### References


