Logic Module Independent Mapping
for Table-Lookup FPGAs *

Ulrich Weinmann 1, Wolfgang Rosenstiel 2

1 Computer Science Research Center at the University of Karlsruhe (FZI)
Haid- u. Neustr. 10-14, 76131 Karlsruhe, Germany
2 FZI and University of Tübingen, Sand 13, 72076 Tübingen, Germany
e-mail: weinmann@fzi.de

Abstract

This paper presents an architecture independent method of technology mapping for table-lookup FPGAs. The internal architecture of logic blocks is a parameter for the mapping step. The first part of the paper describes a new mapping algorithm, which results were compared to existing mapping tools for the Xilinx 4000 series. In a second step the tool was used to examine different new architectures with respect to chip area and delay minimization. The mapping of those architectures showed an improved factor between gate equivalent and used chip area in comparison to commercially available architectures.

1 Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly popular for designers, who want a cheap and fast implementation of their circuits. Fast redesigns and easy chip programming techniques give the designer the possibility to choose an FPGA among different architectures and to find the optimal technology for the specific application [16]. Very important for those decisions are the technology mapping tools, which provide information about area consumption and maximal delays. The technology mapping for table-lookup FPGAs [2,6,7,8,10,11] has reached a point of optimization, which is difficult to improve in respect to area and delay minimization. Furthermore all mapping algorithms are restricted and optimized for only one architecture and can not be applied to other generations of FPGAs.

Therefore our approach combines optimized technology mapping for combinational and sequential circuits with the possibility to use the architecture of a chip as parameter for the mapping procedure. This technique shows several advantages: First the designers only need one tool to compare implementation results for different architectures. Second it is very easy to expand the tool for other generation of FPGAs. For the first time it is possible to examine new

* This research is supported by the Esprit LINK project 6855
and not yet existing architectures in respect to their behaviour for benchmark circuits. In this paper we investigated two new architectures for comparison with existing chips.

Section two describes the general approach to the technology mapping and the algorithms used. The chapters three and four show how the technology independence was reached. In order to proof the method in section five different MCNC-Benchmarks were tested for the Xilinx XC2000/4000 series [17] and compared with results of existing mapping tools. Furthermore new logic block architectures were implemented and evaluated for their efficiency.

2 The Technology Mapping

This chapter will discuss the possibilities of mapping any circuit into any kind of architecture based on table-lookup structures. The approach uses known algorithms together with new methods for the decomposition and mapping. Figure 1 depicts the general overview. Like with other technology mapping approaches first a technology independent logic minimization has to be performed [4,5]. The grey highlighted mapping part shows the new approach. First a decomposition has to split the circuits into its elementary nodes, which are the basis for the logic blocks of an architecture. In a second step the desired architecture has to be divided into basic fragments, which are stored in a fragment library and include all implementation possibilities. The actual mapping step assigns all elementary nodes to the fragments and to the final logic blocks. Afterwards the minimized and mapped circuit can be placed and routed for the implementation in a FPGA.

![Figure 1: Technology Independent Mapping](image-url)
2.1 Decomposition

The decomposition of a circuit has the goal to cut a minimized circuit into elementary nodes. Elementary nodes are defined as fragments of a logic block. As an example, the logic blocks of the Xilinx XC 4000 have the elementary nodes *Lookup Tables with up to four inputs and two D-Flipflops* [17]. To implement a logic structure into a FPGA all not elementary nodes have to be cut. In our tool we implemented five different decomposition methods, which will be described in the following. For each node it can be individually decided, which method is the best to be applied. Different experimental results showed, that some methods lead to a better implementation in terms of area consumption, others support the speed enhancement. A function including these variables will be introduced after the description of the methods.

Therefore three main parameters for the choice of a decomposition method can be defined: the architecture, which determines the structure of an elementary node, the optimization goal, like area or speed and the structure of the nodes to be partitioned into smaller parts. The size and the complexity of the function is important for the decomposition.

Figure 2 gives an overview of the decomposition. In the first step all non elementary nodes have to be found. For each of those nodes one or more methods for decomposition have to be selected and applied. This process often results in too small nodes, which can be collapsed in a final step.

<table>
<thead>
<tr>
<th>Search for non elementary nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>For all non elementary nodes</td>
</tr>
<tr>
<td>Selection of a decomposition method</td>
</tr>
<tr>
<td>Collapsing of elementary nodes</td>
</tr>
<tr>
<td>Until no elementary nodes exist</td>
</tr>
</tbody>
</table>

Figure 2: The decomposition step

The following chapters describe the implemented decomposition methods. The cube packing and co-factoring are already known and therefore just described in their functionality. Section 2.1.5 introduces a new method for decomposition based on the migration of product terms and therefore is described in detail.

2.1.1 Cube Packing

The cube-packing is a method for decomposition, which is used by various tools for technology mapping [13,9]. The goal is the cutting of table lookup blocks with the size $n$ into smaller parts with size $m$ ($m<n$). This requires, that no product term of the function does depend on more than $m$ variables. Therefore the algorithm can be formulated as followed:
1. Sort the product terms of the function by the number of variables they depend on.
2. Generate an empty table-lookup-list.
3. Do step 4 through 6 until the original table-lookup-list is empty.
   4. Take the largest productterm (PD) of the list.
   5. Look for a new lookup table (LUT) list, which allows the placement
      in terms of number of variables. If this is not possible, create a new LUT.
   6. Place PD in the chosen LUT.

Afterwards all resulting functions need to be connected by an OR function. Since very large
functions can result in many smaller functions, it can be disadvantageous to combine those
functions with an OR gate in terms of area consumption. Therefore we extended the algorithm
for large functions. Instead of a parallel implementation with an OR gate the functions were
combined in series.

2.1.2 Co-factoring

The co-factoring, also called Shannon decomposition, is often used for the technology map-
ing into multiplexer based architectures. Generally the following equation for the function $F$
is applied:

$$ F(x_1, \ldots, x_i, \ldots, x_n) = F(x_1, \ldots, 1, \ldots, x_i) \lor F(x_1, \ldots, 0, \ldots, x_i) \overline{x_i} \quad 1 \leq i \leq n $$

or

$$ F(x_1, \ldots, x_n) = F^1 x_i \lor F^0 \overline{x_i} \quad 1 \leq i \leq n $$

If the function $F$ has $n$ variables, the functions $F^1$ and $F^0$ depend on maximal $n-1$ variables.
Very important for a good decomposition is the selection of this variable $x_i$. The search for this
variable is performed by calculating the following a cost function $K$ for every parameter $x_i$
based on the support of input variables and finding $\min K(x_i)$.

2.1.3 Decomposition of Functions with one Product Term

Functions with only one product term can not be divided by cube-packing and the co-factoring
doesn't result in good decompositions. Therefore those functions will be cut with a simple
AND-Decomposition, which cuts the function into two equal parts and connects the results
with an AND gate.

Example: (lut=lookup table)

$$ F = \overline{abcde} \overline{fghi} \Rightarrow \begin{align*}
  \text{lut1} &= \overline{abcde} \\
  \text{lut2} &= \overline{fghi} \\
  \text{lut3} &= \text{lut1} \land \text{lut2}
\end{align*} $$

2.1.4 Extraction of Product Terms

The method is based on the search for equal productterms. Within lookup tables common pro-
ductterms or parts of them can be extracted and realized in a separate lookup table. This
reduces the number of input variables of the original function. Especially very symmetric
designs are suitable for this decomposition method. The procedure for finding the best terms to
be extracted is very expensive. All product terms $n$ of a function will be compared with each
other ($n*({n-1)/2}$ comparisons) and the ones with the most variables in common will be taken
for extraction. However the method is very useful for area optimizing implementations,
because no additional gate is necessary to connect the two partitions.
2.1.5 Decomposition based on Iterative Improvement

This approach is usually used for partitioning and placement of circuits [12,14]. Our technology mapping uses this method to find suitable partitions of larger functions and is based on the migration of product terms between functions. Generally with a function of \( n \) product terms \( 2^{(n-1)} \) possible partitions of any size can be achieved. Since an exhaustive calculation for larger functions is too time consuming, the product terms to be exchanged must be found by a cost function. This function calculates the absolute costs for exchanging two product terms from each LUT to the other. Important for the decomposition into logic blocks is the size of the resulting LUTs. Therefore the cost function respects the number of input variables of each function.

Cost of the partitioning:

\[
K_{ij} = \left( \frac{n}{2} - n_i \right)^2 + \left( \frac{n}{2} - n_j \right)^2
\]

with

- \( n = |\text{sup}(F)| \) = Number of variables of the original function \( F \)
- \( n_i = |\text{sup}(\text{LUT}_i)| \) = Number of input variables of LUT\(_i\)
- \( n_j = |\text{sup}(\text{LUT}_j)| \) = Number of input variables of LUT\(_j\)

The cost function is symmetric and therefore supports the partitioning of a function into two functions with the equal amount of variables. The factor \( n/2 - n_x \) considers the desired LUT size of the architecture.

Using this function the following procedure for exchanging product terms can be performed.

Given:
(1) Lookup Table with \( n \) product terms
(2) An initial partition into two LUTs. (Random partitioning)

Minimizing the costs:

Do

\{
  \begin{align*}
    m &= 0; \\
    K_{i=0,j=0} &= \text{Costs of the initial partitioning;}
  \end{align*}
\}

While not all product terms locked

\{
  \begin{align*}
    \text{For each product term } i \text{ of LUT}_i \\
    &\; \text{For each product term } j \text{ of LUT}_j \\
    &\quad \text{calculate } K_{ij} \text{ for a possible exchange of the product term pair } ij;
  \end{align*}
\}

\[
m = m + 1;
\]

\[
\text{calculate } i \text{ and } j \text{ with the minimal cost } K_m \text{ for all } K_{ij};
\]

\[
\text{do exchange the corresponding product terms } i \text{ and } j
\]

\[
\text{and lock } i \text{ and } j
\]

Find the smallest \( K_p = \min(K_0,...,K_m) \);

\[
\text{Undo all exchanges with the cost } K_x \text{ with } 1 \leq x \leq m
\]

Until \( l > 0 \)
The algorithm calculates first the costs for all possible exchanges of two product terms. The one with the minimal cost will be executed. Both product terms are locked and will not be taken for further exchanges. This routine is performed until all product terms are locked. To find the best partition, the exchange of product terms with the minimal partitioning cost will be selected. This loop is executed until no improvement is possible.

In comparison to the Kernighan-Lin Algorithm this procedure doesn’t use a gain as cost function, but calculates the absolute cost for each migration. The advantage of avoiding local minima is still be given through the exchange of groups of product terms.

2.2 Collapsing

The decomposition often results in functions with too few input variables, which might not be optimal to be implemented in an architecture. Therefore small LUT have to be collapsed to achieve a better use of a given structure.

First the decomposed nodes will be tested, if they allow collapsing. Here single nodes and the resulting function of the collapsing step are not allowed to have more input variables, than the largest LUT of the given architecture. The heuristic searches for all nodes suitable nodes with the goal to minimize the number of input variables of the resulting node.

3 Selection of the Decomposition Algorithm

An important part of the technology mapping is the selection of the optimal decomposition method. Different parameters like design- and architecture structure and the optimization goal of the designer are important for this decision. The designer defined weights are divided into three categories: The optimization goals (area, speed) and the execution time for the mapping. The following table summarizes, which decomposition methods along with the design goal will be used for further investigations:

<table>
<thead>
<tr>
<th>No.</th>
<th>Decomposition Method</th>
<th>Fast Mapping</th>
<th>Speed Optimized</th>
<th>Area Optimized</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cube-Packing parallel</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Cube-Packing serial</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Co-factoring</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Decomp. of one product term</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>5</td>
<td>Extraction of product terms</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>6</td>
<td>Iterative Improvement</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

After this decision only some methods will be used for the mapping. The assignment of the optimization goal to a specific decomposition method was explained in the according section of the algorithm. If a function with one product term occurs, only method 4 is used.

In the next step the actually used decomposition processes have to be chosen. Here specific characteristics of the node to be partitioned have an influence on the decision. Therefore all remaining methods are executed and the best one will be chosen. This is a very simple process, but guarantees best result. The optimization goal is to find partitions, which fit into the logic
blocks of a specific architecture. Important is the number of input variables for each LUT, which must be conform with the given structure or list of fragments (see section 5) of the architecture. The table shows, that almost all decomposition methods are examined for area optimization. This is acceptable, because the execution time was even for circuits with more than 5000 gates under 10 minutes.

4 Building Fragments out of Elementary Nodes

Chapter two described different methods for partitioning a circuit into smaller parts, which fit into a given structure of a FPGA architecture. These parts have to be placed into the logic structure, which in the following will be called fragments. As an example the logic block of an Xilinx XC4000 architecture [17] can be divided into 12 fragments displayed in figure 3. These fragments can be extended to 25 different fragments considering all possible input combinations. Starting from the inputs to the outputs all nodes will be assigned to a special fragment. This is performed by a recursive pattern-matching algorithm. An exact description can be found in [15]. In a second step all fragments are assigned to the final logic structure of the architecture. Here a set of rules define the possible combinations of the fragments.

The advantage of this method is the total freedom in defining the logic structure or the fragments the structure is based on. The example of the XC 4000 architecture showed, that the whole structure can be defined by 25 fragments. Using this method the XC2000/3000 series have even less than 10 fragments being built out of. Furthermore every combination of lookup tables and flipflops, even multiplexers, allow research on new architectures.

![Figure 3: Possible fragments for the XILINX XC4000 architecture](image_url)

5 Results

The technology mapping showed, that it is possible to implement every kind of table lookup based architecture into the algorithm. Therefore the results will be divided into two parts. First the advantage of the technology independence will be used for the implementation of new (not existing) architectures, which were compared to the Xilinx XC2000/4000 series. Second it is
necessary to show the equivalence to existing mapping tools, which were developed for the optimization of one logic structure. All benchmark circuits were first minimized with MIS standard script [4].

**Figure 4: Different logic block architectures**

**Table 2: Architecture Comparisons**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9symml</td>
<td>82,(10)</td>
<td>54,(10)</td>
<td>38,(9)</td>
<td>36,(8)</td>
</tr>
<tr>
<td>alu2</td>
<td>167,(23)</td>
<td>112,(24)</td>
<td>77,(21)</td>
<td>73,(16)</td>
</tr>
<tr>
<td>alu4</td>
<td>314,(23)</td>
<td>198,(25)</td>
<td>132,(20)</td>
<td>138,(21)</td>
</tr>
<tr>
<td>count</td>
<td>50,(11)</td>
<td>25,(7)</td>
<td>21,(6)</td>
<td>18,(7)</td>
</tr>
<tr>
<td>sao2</td>
<td>52,(15)</td>
<td>35,(10)</td>
<td>24,(9)</td>
<td>23,(9)</td>
</tr>
<tr>
<td>vg2</td>
<td>31,(7)</td>
<td>18,(7)</td>
<td>14,(5)</td>
<td>13,(7)</td>
</tr>
<tr>
<td>vda</td>
<td>289,(13)</td>
<td>174,(11)</td>
<td>138,(12)</td>
<td>101,(9)</td>
</tr>
<tr>
<td>z4ml</td>
<td>18,(7)</td>
<td>10,(8)</td>
<td>7,(6)</td>
<td>4,(6)</td>
</tr>
<tr>
<td>5xp1</td>
<td>40,(13)</td>
<td>26,(12)</td>
<td>17,(9)</td>
<td>16,(10)</td>
</tr>
<tr>
<td>9sym</td>
<td>90,(11)</td>
<td>52,(10)</td>
<td>35,(11)</td>
<td>21,(8)</td>
</tr>
<tr>
<td>duke2</td>
<td>174,(14)</td>
<td>98,(12)</td>
<td>75,(13)</td>
<td>71,(10)</td>
</tr>
<tr>
<td>misex1</td>
<td>14,(6)</td>
<td>9,(6)</td>
<td>7,(5)</td>
<td>6,(5)</td>
</tr>
<tr>
<td>misex2</td>
<td>43,(6)</td>
<td>25,(6)</td>
<td>20,(5)</td>
<td>16,(5)</td>
</tr>
<tr>
<td>rd73</td>
<td>22,(11)</td>
<td>12,(10)</td>
<td>9,(7)</td>
<td>10,(10)</td>
</tr>
<tr>
<td>rd84</td>
<td>42,(9)</td>
<td>23,(7)</td>
<td>17,(7)</td>
<td>17,(8)</td>
</tr>
<tr>
<td>Σ (Logic Blocks,Stages)</td>
<td>1428,(179)</td>
<td>871,(165)</td>
<td>631,(145)</td>
<td>563,(139)</td>
</tr>
<tr>
<td>Interconnections</td>
<td>5173</td>
<td>4745</td>
<td>4537</td>
<td>4646</td>
</tr>
<tr>
<td>Σ LUT bits (bits * #Log.Bl.)</td>
<td>16*1428</td>
<td>24*871</td>
<td>40*631</td>
<td>40*563</td>
</tr>
<tr>
<td></td>
<td>22848</td>
<td>20904</td>
<td>25240</td>
<td>22520</td>
</tr>
</tbody>
</table>

During the mapping into different architectures three variables were observed: the number of logic blocks used, the longest path and the amount of connections between the logic blocks. For several combinational MCNC-Benchmarks [19] table 2 indicates the results of used logic blocks, which in the summation were used as a relative number to determine the actual use of area on a chip. This number was multiplied by the number of used bits per lookup table. Here a
lookup table with four inputs (e.g. XC2000 [17,18]) uses $2^4=16$ bits.

In figure 4 the architectures A1 and A4 are known as Xilinx 2000 and 4000 structures. Interesting are the new created architectures A2 and A3. A2 is based on two function generators, which are connected in series. The multiplexer in between allows the use of the TLUs independent from each other and in series. It was not considered for the area calculation, because a multiplexer uses very few space in comparison to lookup tables. The architecture A3 with three TLUs in series was implemented for longest path optimization and uses $2^4+2^4+2^3=40$ bits.

Important are also the number of interconnections between the logic blocks, because this number strongly influences the possibilities for the routing. The results show the lowest use of interconnections for the architecture A3. This could be expected, because the fewer logic blocks are used, the smaller is the amount of connections between the logic blocks. Very interesting are the results for the chip area for the XC4000 series. This logic structure uses a high number of lookup table bits in comparison to the investigated architectures, whereas the structure A2 showed the best relative area consumption.

Table 3 gives a comparison to existing mapping tools. The results here were achieved for combinational and sequential circuits [3]. The results for ASYL were taken from the publication in [2], which did not include results for sequential circuits. In order to use Xilinx PPR [18] the blif format was converted into xnf. The new mapping approach does not show the best result for all of the circuits, however the sum of the used logic blocks is better than ASYL and PPR.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>#Logic Blocks with PPR</th>
<th>#Logic Blocks with ASYL</th>
<th>#Logic Blocks with MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu2</td>
<td>77</td>
<td>51</td>
<td>73</td>
</tr>
<tr>
<td>alu4</td>
<td>142</td>
<td>212</td>
<td>139</td>
</tr>
<tr>
<td>count</td>
<td>19</td>
<td>22</td>
<td>18</td>
</tr>
<tr>
<td>z4mi</td>
<td>7</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5xpl</td>
<td>18</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>9sym</td>
<td>43</td>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>duke2</td>
<td>76</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>misex1</td>
<td>6</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>misex2</td>
<td>18</td>
<td>20</td>
<td>16</td>
</tr>
<tr>
<td>rd73</td>
<td>12</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>rd84</td>
<td>18</td>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>vg2</td>
<td>13</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>sao2</td>
<td>23</td>
<td>26</td>
<td>23</td>
</tr>
<tr>
<td>Σ (logic Blocks)</td>
<td>472</td>
<td>475</td>
<td>426</td>
</tr>
<tr>
<td>s820</td>
<td>60</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>s832</td>
<td>63</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>s953</td>
<td>74</td>
<td></td>
<td>74</td>
</tr>
</tbody>
</table>
5 Conclusion and Future Work

It was shown, that it is possible to develop an architecture independent mapping algorithm, which is comparable to existing mapping tools. It is especially possible to model and evaluate new architectures with different benchmark circuits in order to estimate the advantages and disadvantages. The technology mapping into different architectures indicated, that the logic structure inside a FPGA is an important criteria to decide about speed and area optimization. Furthermore new architectures were introduced, which could result in a better use of chip area, than the Xilinx XC 4000 series.

It is planned to expand the tool for architectures without Table Lookup, RAM based structures, like the multiplexer based anti-fuse technology [1]. This will lead to a technology mapping system, which easily allows the evaluation and mapping of designs into different logic structures.

Acknowledgment

I would like to thank Mr. Andreas Scheuermann very much, who contributed a lot of new ideas and the final implementation during his diploma thesis [15].

References

[1] ACTEL, "ACT Family Field Programmable Gate Array Data Book", 1992


