Boolean function decomposition based on FPGA basic cell structure

By

Călin Mircea Monor* and Alexandru Valachi

“Gheorghe Asachi” Technical University of Iași, România, Faculty of Automatic Control and Computer Engineering

Received: January 8, 2015
Accepted for publication: February 25, 2015

Abstract. The authors present a method for the decomposition of Boolean functions starting from the structure of the basic cell of the FPGA technologies. Starting from a general method of implementing Boolean functions, the authors develop a new method of decomposition using sub-functions that can be synthesized using one single FPGA basic cell. The general method proves that there is a solution for implementing a Boolean function with a given number of variables. The method presented in the current paper provides, if it exists, an optimization of the general result. If the optimized solution exists, it will be working at higher frequency and, eventually, it will require a smaller number of basic cells for implementation.

Key words: Boolean function; decomposition; sub-function; basic cell; FPGA; LUT; frequency optimization; area optimization.

2010 Mathematics Subject Classification: 06E30, 49M27, 94C05.

*Corresponding author e-mail: cmonor@tuiasi.ro
1. Introduction

During the last decade the success of the research and development of the deep sub-micron technologies - 90 nm, 65 nm, 45 nm, 32 nm and 22 nm - solved the requirements for higher frequency and smaller area while implementing a digital design. For this reason, the research on new methods and algorithms of optimizing the implementation was left aside. In the same decade the trend of the electronic industry was to create devices with very large number of features. As long as the users will ask for devices more and more complicated and faster, the development of the deep sub-micron technologies will be pushed to the limit. Already, there are fewer silicon vendors able to provide chips with smaller size of the transistors with a reasonable yield.

The reasons mentioned before, the effort of optimizing the performance of digital circuits will be focused in the future in two directions. One direction is to develop new ideas and technologies which will change radically the approach of the infrastructure to be used. Another direction, easier to develop and that provided good results in the past, is to develop new methods for implementing a design, optimizing the critical aspects by using features of the technology that were not used until now.

The authors present a method of decomposition for Boolean functions while taking into consideration the particularities of the FPGA technologies. The existing methods and algorithms are based on classical decomposition using a set of simple sub-functions, with small number of variables or on various representation and manipulation of the Boolean functions (Scholl, 2001; Nowicka et al., 1999; Muthukumar et al., 2007; Yang, 1991; Bertacco & Damiani, 1997; Brayton & McMullen, 1982). Another research direction for optimizing the logic synthesis of Boolean functions is to represent the function as a BDD and to apply the specific operation (Vemuri et al., 2002; Pan et al., 1996; Yang et al., 2000).

2. FPGA Basic Cell Structure

A FPGA circuit (Field Programmable Gate Array) consists of a matrix of cells with identical structure, several groups of input-output cells, called banks, clocking resources, digital clock managers, embedded blocks of RAM, embedded arithmetical blocks and programmable interconnect resources (Xilinx, 2013).

Most parts of the design will be implemented using cells of the array and programmable interconnect resources. Most of the FPGA technologies developed a structure of the basic cell using the same few components: look-up tables (LUT), registers (DFF), multiplexers and buffers. A typical structure of a basic cell of a FPGA technology is shown in Fig. 1.
The basic cells of the various FPGA technologies include 3 or 4 input LUTs, which are RAM-based function generator. The LUTs can be configured to work as distributed RAM, 8 or 16 bit shift registers or to implement logic functions. During power-on sequence the LUTs can be initialized and the content written can be the truth table of a 3 or 4 input logic function.

The authors are using the special feature of a LUT, the possibility to implement a complex function with the number of variables equal to the number of LUT’s inputs. The existing design flows for FPGAs include, after the logic synthesis which has as target a virtual library including simple logic gates that does not exist in silicon, a stage called mapping. During this stage, using dedicated algorithms, a CAD software tries to map as many gates included into the output netlist as possible into the physical LUTs, (Farrahi & Sarrafzadeh, 2004; Wilton, 1998; Chen & Cong, 2004).

In the following sections it will be used the structure of a slice, the basic cell in several families of one of the most important FPGA vendor – Xilinx. The slice includes 2 LUTs with 4 inputs and a 1:2 MUX. The LUTs can implement 2 functions of 4 variables: \( f_x(x_3, x_2, x_1, x_0) \) and \( f_y(x_3, x_2, x_1, x_0) \). Using this feature and connecting the selection of the MUX to another variable, \( x_4 \), the slice can implement a 5 variable function:

\[
 f(x_4, x_3, x_2, x_1, x_0) = x_4 f_x(x_3, x_2, x_1, x_0) + x_4 f_y(x_3, x_2, x_1, x_0)
\]

(1)
3. Representation of Boolean Functions
Using Sub-Functions

Let be a function depending on 2 variables, \( x_4, x_3 \), and 2 sub-functions, \( f_1, f_2 \):

\[
G(x_4, x_3, f_1, f_0) = f_0 x_3 x_4 + f_1 x_3 x_4 + f_0 f_1 x_4
\]

(2)

where \( f_1 = f_1(x_2, x_1, x_0) \) and \( f_0 = f_0(x_2, x_1, x_0) \).

It can be written:

\[
\div G = 0000 0101 1100 0100
\]

(3)

Let be:

\[
\div f_1 = 0111 0100 \quad \text{and} \quad \div f_0 = 0101 0011
\]

(4)

It can be written:

\[
f_1 = x_1 x_0 + x_2 x_1 \quad \text{and} \quad f_0 = x_2 x_0 + x_2 x_1
\]

(5)

Now, the 5 variables function can be written:

\[
F(x_4, x_3, x_2, x_1, x_0) = x_2 x_1(x_4 + x_3) + x_4 x_3 x_2 x_0 + x_4 x_3 x_2 x_0 + x_4 x_3 x_1 x_0
\]

(6)

and \( \div F = 0000 0000 0101 0011 1000 1011 0000 0011 \)

(7)

The function F can be represented using the matrix:

\[
E_{\{x_4, x_3\} \setminus \{x_2, x_1, x_0\}} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}
\]

(8)

and the function G can be represented using the following matrix:
After observing that the columns of $E$ are also columns of $E'$, the following matrix can be built:

$$
E'_{[x_4x_5]}: \{f_1f_0\} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0
\end{bmatrix}
$$

(9)

After observing that the columns of $E$ are also columns of $E'$, the following matrix can be built:

$$
R_{JI} = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0
\end{bmatrix}
$$

(10)

using the following rules:

- each column of $R_{JI}$ has at most one value of logic 1;
- if a column of $R_{JI}$ has a value of logic 1, this will be located in the line with the index equal to the rank of the corresponding column in the matrix $E'$ identical to the column in the matrix $E$.

4. Decomposition of Boolean Functions Using Complex Sub-Functions

Let be $F$, a Boolean function of $n$ variables:

$$
F = F(x_{n-1}, x_{n-2}, \ldots, x_{n/2}, x_{n/2-1}, \ldots, x_1, x_0)
$$

When applying the Veitch-Karnaugh minimization method, the input variables are split into two partitions: $\{x_{n-1}, x_{n-2}, \ldots, x_{n/2}\}$ and $\{x_{n/2-1}, \ldots, x_1, x_0\}$, if the number of variables is even. If the number of the variables is odd, one partition will have one more variable than the other.

As shown previously, a FPGA basic cell can implement any 5 variable Boolean function. For this reason, we are trying to find a set of partitions from which one has 5 variables. After building the E matrix for that partition, the number of different columns will give the number of 5 variable sub-functions needed to decompose the function $F$.

To prove the method, let’s make a case study on a 9 variable Boolean function that we know that can be decomposed using a 5 variable function.

First, let’s consider a random 5 variable Boolean function described by:

$$
f_1 = R_1(0,3,5,10,12,16,23,25,28)
$$

(11)
The Karnaugh map associated is:

<table>
<thead>
<tr>
<th>(x_1x_0)</th>
<th>(x_4x_3x_2)</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>010</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The function is described by the expression:

\[
f_1 = x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0 +
     + x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0 + x_4x_3x_2x_1x_0
\]  

\( (12) \)

The general implementation method requires a single basic cell for implementation, as shown in the Fig. 2.

![Fig. 2](image)

Fig. 2 – Implementation of a 5 variable Boolean function.

The area utilized is 1 basic cell, the longest path from any input to the output is, obviously, one basic cell long.

The results of the synthesis of the same Boolean function using Precision Synthesis RTL Plus 2014, from Mentor Graphics, with the library of Spartan 3E family from Xilinx:
Let’s consider now a Boolean function depending on 4 variables and a sub-function \( f_2 = f_2(x_8, x_7, x_6, x_5, f_1) \) described by the following Karnaugh map:

<table>
<thead>
<tr>
<th></th>
<th>000</th>
<th>001</th>
<th>011</th>
<th>010</th>
<th>110</th>
<th>111</th>
<th>101</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

The function is described by the expression:

\[
f_2 = x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1' + x_8x_7x_0x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1 + x_8x_7x_6x_5f_1
\]

\[
(13)
\]

Using the decomposition of the Boolean functions using 5 variable sub-functions, a possible implementation for \( f_2 \) is shown in Fig. 3.

The results of the decomposition method are:
- area: 2 basic cells;
- longest path from any input to the output: 2 basic cells.
The results of the general implementation method using FPGA basic cell structure:
− area: 21 basic cells;
− longest path from any input to the output: 3 basic cells.
The results of Precision Synthesis RTL Plus:
− area: 6 slices/11 LUTs;
− longest path from any input to the output: 3 slices.
These statistics show the good results generated by the method presented here and the need of applying it, if possible to the general implementation method using FPGA basic cell structure.

The Karnaugh map for \( f_2 = f_2(x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \) can be deducted (Table 1). As stated before, the number of different columns in the E matrix will give the number of 5 variable sub-functions needed to decompose the function.

<table>
<thead>
<tr>
<th>( x_8 )</th>
<th>( x_7 )</th>
<th>( x_6 )</th>
<th>( x_5 )</th>
<th>( x_4 )</th>
<th>( x_3 )</th>
<th>( x_2 )</th>
<th>( x_1 )</th>
<th>( x_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>0011</td>
<td>0110</td>
<td>0111</td>
<td>0101</td>
<td>0100</td>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>0000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0011</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0110</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0111</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1100</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1101</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1110</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1010</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1011</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Inspecting the Karnaugh map, we can see that for the partitions \{x_8, x_7, x_6, x_5\} and \{x_4, x_3, x_2, x_1, x_0\} the columns have just 2 values:

\[
C_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\end{bmatrix}^T \quad \text{and} \quad C_1 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\end{bmatrix}^T
\]  \quad (14)

and this shows that there is enough one sub-function for the variables \(x_4, x_3, x_2, x_1, x_0\) to decompose \(f_2\).

5. Conclusions

The authors present a method dedicated to logic synthesis of Boolean functions while the target is a FPGA technology. The method combines a general implementation method using FPGA basic cell structure with a
mathematical method of decomposing, where possible, of the Boolean functions using sub-functions. The result of the method presented here by the authors has as outputs an implementation with two advantages:

− smaller number of basic cells to be used;
− shorter critical path between the inputs and the output.

The research will be completed with the analysis of the cases when choosing various partitions of input variables and with sub-functions depending on disjunctive variable partitions.

REFERENCES


DECOMPUNEREA FUNCŢIILOR LOGICE UTILIZÂND STRUCTURA CELULEI DE BAZĂ A TEHNOLOGIILOR FPGA

(Rezumat)

Autorii prezintă o metodă de descompunere a funcţiilor logice complexe, cu număr mare de variabile. Se foloseşte structura generală a celulei de bază întâlnită la majoritatea tehnologiilor FPGA şi caracteristicile Look-Up Table-urilor.

Se caută partiţii ale variabilelor în aşa fel încât să se determine un număr de subfuncţii care să aibă una din partiţiile de variabile ca parametru şi să poată fi folosite la descompunerea funcţiei originale.

Se va continua cercetarea şi către dezvoltarea unei metode de determinare a unor seturi de subfuncţii care să fie dependent de partiţii disjuncte de variabile de intrare şi care să poată fi folosite la descompunerea funcţiei iniţiale.