TITLE: COMPARISON OF DIFFERENT DECOMPOSITION
TECHNIQUES OF A DIGITAL CIRCUIT - A STUDY CASE

AUTHORS: ♦ M. RAWSKI, ♦ P. TOMASZEWICZ, ♦ P. AMBLARD
♦ Warsaw University of Technology, Poland, ♦ TIMA–CMP, France

KEYWORDS: LOGIC SYNTHESIS, FUNCTIONAL
DECOMPOSITION, STRUCTURAL DECOMPOSITION

ABSTRACT: This paper presents a comparison of different synthesis methods in complex digital circuits. In the first method the decomposition of a complex problem is made by the designer; it is based on human understanding of the problem. In the second method, the decomposition is made by our program, Demain, dealing with logic minimization [3]. Both methods were used on a true example. The example was such that the two methods give very similar results in complexity. Conclusion will focus on decomposition in digital circuits.

INTRODUCTION

Designing digital circuits can be considered as a well established field. In a lot of manuals of digital circuits design, it is possible to learn different techniques and optimization tricks. Decomposition of a complex problems in simpler ones is always present. A lot of examples in the field of arithmetic are generally presented. The reader can verify that his/her prefered book contains the N bits adder in chapter 2! On another aspect, logic minimization techniques are presented on small examples (4 or 5 input variables). Nowadays CAD packages contain more sophisticated logic minimizations. These packages give good results according to criteria such as complexity or delay. They generally do not take into account the regularity (or the potential regularity) of a given design. We propose here an example of comparison between such techniques. We have tried on a true example
- decomposition based on arithmetic properties and
- decomposition based on a CAD package.

It is interesting to notice that, for THIS example, the results are very similar. This result, rather surprising, suggested us a lot of questions which will be discussed along the paper.

In a first part we present the example and the techniques used in decomposition, in a second part the comparison is made on the example and results are presented.

STRUCTURAL VS. FUNCTIONAL
DECOMPOSITION

Decomposition is probably the most popular and the oldest synthesis method. All designers use this approach of solving problems in their work. Classical synthesis, called hereafter structural decomposition, is the first step of design process, and is performed manually as it is based only on experience and knowledge of designer. However fast developing automatic synthesis can help or even replace synthesis based on structural decomposition.

In Fig. 1, there is shown an example of structural decomposition of a specific problem [1]. The presented block diagram describes algorithm of conversion: the day presented in form day-month-year is converted towards number of day in the year. For example from 18_06_1998 the result is 169 because June the 18th is the 169th day of the year 1998. Since this circuit is a combinational one, it is possible to describe it for example as a truth table. But its implementation (i.e. in FPGAs or in ASICs) would then be very inefficient, since automatic synthesis method do not manage very large functions very well. In this case problem has been structurally decomposed by designer into a net of subsystems, each of which is easier to synthesise and analyse and can be implemented in much more efficient way. Each of this subsystem realises some specific function and can, as a separate block, be used in many others designs. It means that the designer has decomposed this problem into standard subsystems such as multiplexers, arithmetic units, decoders, etc. This decomposition was possible only by use of knowledge and imagination of designer. It will probably be difficult for any automatic synthesis to split this system into such a logic structure. However some automatic tools can break problems into small
sub functions to make further synthesis easier and realisation more efficient in comparison with implementation of not decomposed problem. This difficulty depends mainly of the abstraction level of the input language. Obviously if the description contains the decomposition, the task is easy. We enter here in the world of the so called “synthesisable” descriptions.

There is a difficult trade-off in CAD tools:
- either they are able to synthesise a solution quite different from the structure suggested by the textual or schematics description. In this case the tool is highly sophisticated, but the results are sometimes surprising.
- or the tools give an actual implementation strongly related to the description organisation and we may consider them as poor.

Another synthesis approach based on decomposition is functional decomposition. It is especially well suited to FPGA based realisations. In this approach the functions used in the decomposition are not necessarily meaningful. The only aim is to find “good” functions with respect to criteria such that global complexity, global delay of the circuit, etc. Obviously the global behaviour of the circuit must be obtained as a composition of the elementary functions behaviours.

In the following sections, the comparison of the structural decomposition with the synthesis approach based on functional decomposition will be made.

A decoder from binary to BCD code from block diagram shown in Fig. 1 will be taken under consideration. This decoder converts numbers from the range 1 to 356 given on 9 input bits to 3 digits BCD (12 bits).

Let \( A = A_{N-1}, A_{N-2}, ..., A_1, A_0 \) be a Boolean vector of \( N \) bits. It represents an integer \( VA \) in binary. Let \( dA = dA_{P-1}, dA_{P-2}, ..., dA_1, dA_0 \) be the decimal writing of \( VA \) where \( dA \) are digits in the range \([0,9]\). Each decimal digit is coded in binary: \( dA_{P-1}_3, dA_{P-1}_2, dA_{P-1}_1, dA_{P-1}_0 \). This is called BCD (Binary Coded Decimal).

**Structural decomposition**

The translation from binary form to decimal form can be performed by following algorithm:

\[
\text{integer_in_binary: } 1010B = (\text{quotient}_1, \text{remainder}_1) \\
\text{quotient}_1 : 1010B = (\text{quotient}_2, \text{remainder}_2) \\
\text{quotient}_2 : 1010B = (\text{quotient}_3, \text{remainder}_3) \\
\vdots \\
\text{quotient}_{n-1} : 1010B = (\text{quotient}_n, \text{remainder}_n)
\]

The principle is to perform the calculation step by step. At each step the remainder and the quotient in a division by ten are computed. The remainder is given directly in binary form. It is smaller than 10. The steps are done by combinational circuit. Each step performs calculations on the quotient generated by the previous step. At each step the quotient is an input for the next step and the remainder is a final result. After processing \( \text{number_in_binary} \) can be presented as \( n+1 \) positional decimal digit, represented by remainders:

\[
\text{number_in_BCD} = \text{remainder}_n, \ldots, \text{remainder}_3, \text{remainder}_2, \text{remainder}_1
\]

Based on this algorithm a combinational circuit can be created for such a conversion. This circuit should consist of slices, each of which performs division by 10. Its block diagram is presented in Fig. 2.

![Figure 2. Block diagram of circuit translating number from binary to BCD.](image)

The process performed by each slice can be described as division by 10. Each slice is implemented by a simple set of cells. Each cell elaborates one bit of quotient and partial remainder for division by ten. Using very simple transformations it is needed to compute the quotient and remainder only by 5 instead of 10:

\[
(A\%10) = ((A/2)\%5)*2 + (A\%2) \\
(A/10) = ((A/2)/5)
\]

In connection to this, each slice can be implemented using combinational blocks performing division by 5. Implementation of the slice is presented in Fig. 2. Each cell works on 4 bits input number. These 4 bits code a number smaller than 10. The quotient by 5 is 1 or 0. The remainder is smaller than 5. This remainder is extended by a least significant bit taken from the input vector. In the input the first tree bits are extended by a zero most significant bit. This maintains the condition that the input is smaller than 10.
Each cell is a simple combinational circuit with 4 inputs and 4 outputs. It can easily be represented by the following truth table where \( q \) is the quotient bit and \( r_2, r_1, r_0 \) the remainder bits:

<table>
<thead>
<tr>
<th>( x_3 )</th>
<th>( x_2 )</th>
<th>( x_1 )</th>
<th>( x_0 )</th>
<th>( q )</th>
<th>( r_2 )</th>
<th>( r_1 )</th>
<th>( r_0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Truth table of a division by 5 cell

Final block diagram is presented in Fig. 2. This idea is similar to the one implemented by the 74185 circuit [4].

This implementation has a very important advantage. It can easily be expanded to the conversion of larger number of bits by adding new slices. However there is also disadvantage, namely this circuit cannot have been implemented without knowledge about rules of number conversion.

For someone who does not know how algorithm of conversion works or for someone who does not want to spend a time for implementing such an algorithm there is only one solution – automatic synthesis tools.

**Functional Decomposition**

Existing synthesis tools allow user to implement their systems manually or fully automatically. In this experiment the synthesis tools were used. Academic tool Demain [3] is used to perform functional decomposition and commercial tool Synopsys is used to implement structures obtained in structural and functional decompositions using CMOS and FPGA libraries.

Synthesis algorithm implemented in Demain is so called balanced decomposition. The balanced multilevel decomposition algorithm relies on successive decomposition of a given Boolean function with either parallel or serial decomposition applied at each phase of the synthesis process.

Let a function \( F \) represent functional dependence \( Y = F(X) \), where \( X \) is the set of input variables and \( Y \) is the set of output variables, as shown in Fig. 3a. It may be possible to decompose the function into one of the two basic forms, parallel or serial decomposition.

In parallel decomposition, the set of outputs \( Y \) is partitioned into two disjoint subsets, \( Y_h \) and \( Y_g \), so that the input support sets of the two resulting components, \( X_h \) and \( X_g \), are smaller than the set of primary inputs, \( X \), as shown in Fig. 3b.

A serial decomposition of the Boolean function \( F(X) = Y \) is shown in Fig. 3a and is defined as follows. Let \( X = A \cup B \) be the set of input variables, \( Y \) the set of output variables and \( C \subseteq A \). There exists a serial decomposition of \( F \) if \( F = H(A,G(B,C)) = H(A,Z) \) where \( G \) and \( H \) denote functional dependencies \( G(B,C) = Z \) and \( H(A,Z) = Y \), and \( Z \) is the set of output variables of \( G \).

Intertwining of serial and parallel decomposition strategies opens up several interesting possibilities in multilevel decomposition. Experimental results show that the right balance between the two strategies and the selection of a set of bound variables in serial decomposition severely influence the area and depth of the resultant network.

Applying algorithm implemented in Demain, the circuit of conversion from binary form to BCD can be implemented using the net of blocks presented in Fig. 4. The decomposition was made for size of cells with 4 inputs. Numbers in block’s representation denotes number of CLBs of FPGA necessary for implementation of this block.

**COMPARISON AND RESULTS**

In this section a comparison between these two methods is made. For the experiment the nets of blocks obtained using structural decomposition and functional decomposition were implemented by Synopsys using standard cell library and FPGA library. Also not decomposed circuit described by truth table was implemented using Synopsis. Results are presented in Table 2.

**Method Used In Comparison**

The comparison was made in two steps. In the first step the functional synthesis of the circuit was driven by our tool Demain. The input was the truth table of the transformation from binary to Binary Coded Decimal form. The input contains 365 lines corresponding to number in interval \([1–366]\). The output of Demain is a net of blocks, each of which represents particular subfunction obtained in the process of decomposition.
In the second step these results were taken from Demain, both with results of structural decompositions, and were introduced into the Synopsys CAD packages. For FPGA implementation, Look-Up Tables (LUT) were generated. We obtained four solutions:
- LUT from structural decomposition,
- Gates implementation from structural decomposition,
- LUT from functional decomposition,
- Gates implementation from functional decomposition.
As we already noticed, the structural decomposition gave some 4-input blocks, so the comparison was easy in the LUT field.

**Results**

The results presented hereafter give the number of cells and number of levels. Obviously, for gate synthesis, the number of logical levels between the inputs and outputs of the circuit is higher than for FPGA synthesis. The global delay of the circuit is related directly to this number of levels.

<table>
<thead>
<tr>
<th>Project BIN2BSD</th>
<th>cells</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non decomposed</td>
<td>664 (169)</td>
<td>–</td>
</tr>
<tr>
<td>Structural dec.</td>
<td>171</td>
<td>17</td>
</tr>
<tr>
<td>Functional dec.</td>
<td>172</td>
<td>18</td>
</tr>
</tbody>
</table>

Table 2a. Results of implementation (CMOS library)

<table>
<thead>
<tr>
<th>Project BIN2BSD</th>
<th>cells</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non decomposed</td>
<td>479 (100)</td>
<td>–</td>
</tr>
<tr>
<td>Structural dec.</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>Functional dec.</td>
<td>39</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 2b. Results of implementation (FPGA library)

For non-decomposed project, optimisation implemented in Synopsys system was performed. The second number presents number of cell obtained after several optimisations cycles. This optimisation was aimed on area reduction that reduced also the number of cells.

**Comments**

It was an interesting surprise for us to find such an example where the results were so similar. In this case study, structural and functional decomposition give similar results.

The reader should keep in mind that the structural decomposition of this example can be extended to any number of bits. In a full custom implementation we could also expect the structural decomposition to be topologically more regular.

**CONCLUSIONS AND PEDAGOGICAL COMMENTS**

Up to now a lot of books about digital design present several techniques:
- small size minimisation of logic functions. Karnaugh maps, Quine-MacCluskey algorithm are commonly used by human designers for small size functions,
- structural decomposition is introduced, very often through the N bits adder, as a synthesis paradigm used in big circuits.

These two techniques are used in human made synthesis. In CAD at the logic level, the current state-of-art. is based on Binary Decision Diagrams. They are used in minimisation, but cannot be consider in structural decomposition.

Intermediate size circuits are in a complexity such that CAD techniques and structural decomposition and fine-grain optimisations give the same results. This is the case for our example of 9 bits binary to BCD converter.

Teachers of digital techniques will take care of this new perspective. Intermediate size circuits can be synthesised by different techniques.

**Acknowledgements**

This work has been performed as a part of TEMPUS project S_JEP_09159_95

**References:**

[1] Amblard P. and 9co-authors, Design of a Complex Combinational A.S.I.C. with Educational Aims,

