Decomposition and Isomorphism of Logical Systems

Ján Bača[1]

[1]Department of Computers and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
Jan.Baca@tuke.sk

Abstract. The contribution deals with decomposition of logical systems for the purpose of solving analysis, synthesis and diagnostics tasks. The system can be specified by its structure or by algebraic expressions of its function. Particular attention is paid to propose algorithms for ordering of components of algebraic expression, decomposition of algebraic expression into substrings, and composition of modularly-organized logical circuit from those substrings. Also a way for the determination of identical and isomorphic modules of de/composed circuits is presented.

Keywords. logical system, decomposition of systems, isomorphic modules, algebraic expression.

Key Terms. Mathematical Model, Research, Development.

1 Introduction

The tasks complexity of the logical systems synthesis, analysis and diagnostics are often very high and it depends on the system dimension.

If complexity $c(n)$ of the solution of the task is not linear $c(n) \neq O(n)$, but polynomial $c(n) = O(n^k)$, or exponential $c(n) = O(g^n)$, then the reduction of the total solution complexity by the decomposition of the system $S(n)$ into p subsystems $S_1(n_1)$, $S_2(n_2)$, ..., $S_p(n_p)$, $n_i < n$ is used with great advantage (n – parameter which determines the system size). This is the reason for decomposition of systems into several smaller modules and solving the tasks for separate modules and composing module results back into result for whole system. Decomposition makes sense if decreasing of the solution time $t(S(n))$ in decomposed systems is bigger than increase of the solution time t_{ds} related with decomposition and back composition of the system

$$t(S(n)) > \sum_{i=1}^{p} t(S_i(n_i)) + t_{ds}$$

Time $\boldsymbol{t_{ds}}$ grows with numbers of modules and numbers of connections among them. The decomposition on modules with one (or minimal number of) output is more suitable. If some modules are isomorphic (or identical) it is enough solve tasks for one

of them and apply for others. For example if we generate tests for complex system, we can decompose it into several smaller modules, prepare tests for each group of isomorphic modules and compose the test for whole system on the base of module tests.

In separate tasks the systems can be described by different descriptions or formal specifications [1], [2], [3]. The definitions of the system can be split into two basic groups. In the first group the systems are defined by its function. For example the combinational logical circuit can be present by algebraic expressions of system output functions. In the second one the systems are defined by its structure. In these cases the system can be presented by netlist. The function of the system described by structure can be derived from functions of individual components and connections between them. To understand the dependence between these different system descriptions it is essential to mention, that the given function can be realized by different structures, however the given structure of the circuit realizes a unique determined function.

The logic circuit structure is given by the equation $S = (E, C)$, where E is the set of logic elements and C is the set of mutually connections, which are performed in input N_i, internal N_r or output N_o nodes.

Each element $E_j = (F_j, X_j, Y_j)$ is characterized by the logic function F_j, the inputs $X_j = (x_{j_1}, x_{j_2}, ...x_{j_{m_i}})$ and the outputs $Y_j = (y_{j_1}, y_{j_2}, ...y_{j_{m_i}})$ - $F_j : X_j \to Y_j$. The $x_{j_u} \in N_i \cup N_r$, $y_{j_v} \in N_r \cup N_o$ signify the node names, which are the elements of the set C.

Elements $E_p = (F_p, X_p, Y_p)$, $E_q = (F_q, X_q, Y_q)$ are mutual connected, if $\{x_{p_1}, x_{p_2}, ...x_{p_{k_i}}, y_{p_1}, y_{p_2}, ...y_{p_{m_i}}\} \cap \{x_{q_1}, x_{q_2}, ...x_{q_{kj}}, y_{q_1}, y_{q_2}, ...y_{q_{m_j}}\} \neq \varnothing$.

The module structure $S_m = (E_m, C_m)$ is connective, if for $\forall E_p, E_q \in E$, $\exists$ the string $E_p, E_{i_1}, E_{i_2}, ...E_{i_u}, E_q$ ($u \geq 0$), in which all adjacent elements are mutual connected.

Elements $E_p = (F_p, X_p, Y_p)$, $E_q = (F_q, X_q, Y_q)$ are identical $(E_p \equiv E_q)$, if $F_p \equiv F_q$, $X_p \equiv X_q$, $Y_p \equiv Y_q$.

Elements $E_p = (F_p, X_p, Y_p)$, $E_q = (F_q, X_q, Y_q)$ are isomorphic $(E_p \cong E_q)$, if there exists such one to one correspondence $\{x_{p_1}, x_{p_2}, ...x_{p_{k_i}}\} \Leftrightarrow \{x_{q_1}, x_{q_2}, ...x_{q_{kj}}\}$ $\{y_{p_1}, y_{p_2}, ...y_{p_{m_i}}\} \Leftrightarrow \{y_{q_1}, y_{q_2}, ...y_{q_{m_j}}\}$ and $F_p \equiv F_q$.

Modules $S_{mi} = (E_{m_i}, C_{m_i})$, $S_{mj} = (E_{m_j}, C_{m_j})$ are isomorphic, if there exists such one to one correspondence $E_{m_i} \Leftrightarrow E_{mj}, C_{m_i} \Leftrightarrow C_{m_j}$, that $(E_p \cong E_q)$, $k = 1,2,... p$, where p is the number of module elements.

Modules $S_{mi} = (E_{m_i}, C_{m_i})$, $S_{mj} = (E_{m_j}, C_{m_j})$ are identical, if they are isomorphic and $C_{m_i} \equiv C_{m_j}$.

The decomposition requirements:

a) connectivity of modules
b) minimal number of connections to other modules
c) minimal number of modules
d) minimal number of module types

e) minimal number of module elements

Contradictory requirement c) and e) can be solved by hierarchy of module decomposition.

Specification of logical system by structure is more suitable for manual decomposition because in their scheme we can see structure of modules and their connection to other ones. Specification of logical system by algebraic expression of function is more suitable for automatic decomposition because in the algebraic expression we can find identical or isomorphic parts which correspond with system modules.

By decomposition of logical systems it is necessary to differentiate between combinational part and sequential part of logical circuits. Combinational part which represents excitation and output function can be described by Boolean functions. Sequential parts can be described by the transition matrix of elementary memory elements. In sequential logical circuits are usually used one type - D, T, RS or JK of elementary memory elements which present isomorphic modules with different inputs and outputs. Excitation and output function in combinational part usually are different and it is useful decompose them into several smaller modules and look for their isomorphism.

2 Algebraic Expression of a Logical System Function

In case of combinational circuits the algebraic expressions – logical operations of conjunction (AND) $x * y$, disjunction (OR) $x + y$, negation (NOT) $\sim x$ Sheffer's operations (NAND) x | y, Pierce's operations (NOR) x $\downarrow$ y, non-equivalence (XOR) x $\oplus$ y - is fully sufficient for system specification. The brackets are used to specify priority of operations. Algebraic expressions are represented in the grammar-defined language [3]. Operation among primary inputs will be denoted as operation on level one. Operation is on level i+1 if maximal level of embedded operations is equal i. The maximal number of embedded operations, which must be realized by function calculation, will be denoted as depth of function embedding (the degree of the corresponding circuit).

3 De/composition of Logical Systems

3.1 Ordering of Algebraic Expression Elements

For decreasing of modules identity detection complexity, it is desirable to order the algebraic expression at first [4]. We will begin with the algebraic expression, where the individual variables are denoted as x_i, and may occur either in direct or in inverse instances in the expression. Position of the given variable in the sequence, according to which the expression will be ordered, is chosen in following manner:

$j = 2i$, for x_i,

$j = 2i - 1$, for $\sim x_i$, $i = 1,\ldots,v$

where v is the number of primary inputs of the circuit. Further sequence of primary variables is obtained this way is $\sim x_1, x_1, \sim x_2, x_2, \ldots, \sim x_n, x_n$,

The ordering itself is done in the bottom-up manner for every string, which represents one logical operation, separately. Composite strings, which represent the outputs of operations on the lower depth of function embedding, will be ordered by leading variables in the string, symbols of operators will be ordered by sequence: ~ (NOT), * (AND), + (OR), | (NAND), ↓ (NOR), ⊕ (XOR). This way of the strings ordering will be denoted lexicographic ordering. The algorithm of the ordering is following:

INPUT:

An algebraic expression representation in the grammar-defined language [3].

OUTPUT:

Ordered algebraic expression.

BODY:

1. Let $i = 1$, let n be the depth of function embedding.
2. For every logical operation on the level i order the strings according lexicographic ordering.
3. $i = i + 1$
4. Repeat steps 2 and 3, until $i \leq n$.

The example of ordered algebraic expression for a circuit realizing the full adder circuit (Fig. 1) can be following. The circuit has two outputs – sum and carry into the higher level:

$s = \sim(\sim x_1 * x_2 + x_1 * \sim x_2) * x_3 + (\sim x_1 * x_2 + x_1 * \sim x_2) * \sim x_3$

$c = (\sim x_1 * x_2 + x_1 * \sim x_2) * x_3 + x_1 * x_2$

3.2 Decomposition of Algebraic Expression into Substrings

Decomposition of an algebraic expression comes out from the expression described in section 3.1, which is analyzed in bottom-up way. The result of decomposition is the decomposition of an algebraic expression into substrings that are realizable with only one elementary logical operation.

The algorithm of algebraic expression decomposition into substrings is following:

INPUT:

Ordered algebraic expression (section 3.1).

OUTPUT:

A system of algebraic expression substrings that contain exactly one logical operation.

BODY:

1. Search a substring that contains exactly one logical operation. Considered logical operations are ~ (NOT), * (AND), + (OR), | (NAND), ↓ (NOR), ⊕ (XOR).
2. Substitute identified substring by one variable marked by symbol x and two numeric values. The first value indicates the level of substring (the depth of related function embedding). The second value identifies selected substring within the given level.

3. Compare expressions x[i,j], for j=1, 2, … , u, where u is the number of expression with the level i. Identical expressions x[i,j] and x[i,j+v] replace by expression x[i,j].
4. Express initial algebraic expression using substitution variables.
5. Repeat steps 1, 2, 3 and 4 until initial expression is reduced to logical expression containing only one logical operation.

In the case of circuit with more outputs it is necessary to execute this algorithm as well as the next algorithm for algebraic expressions of all circuit output functions.

Example of an application of the presented algorithm for the circuit realizing full adder circuit is presented in the Table 1.

Table 1. Decomposition of full adder algebraic expression into substrings

Function	**Substitution**	**Identity**
s=~(~x1*x2+x1*~x2)*x3+(~x1*x2+x1*~x2)*~x3	x[1,1]=~x1	
s=~(x[1,1]*x2+x1*~x2)*x3+(~x1*x2+x1*~x2)*~x3	x[2,1]= x[1,1]*x2	
s=~(x[2,1]+x1*~x2)*x3+(~x1*x2+x1*~x2)*~x3	x[1,2]=~x2	
s=~(x[2,1]+x1* x[1,2])*x3+(~x1*x2+x1*~x2)*~x3	x[2,2]= x1*x[1,2]	
s=~(x[2,1]+ x[2,2])*x3+(~x1*x2+x1*~x2)*~x3	x[3,1]= x[2,1]+x[2,2]	
s=~ x[3,1]*x3+(~x1*x2+x1*~x2)*~x3	x[4,1] =~ x[3,1]	
s= x[4,1] *x3+(~x1*x2+x1*~x2)*~x3	x[5,1] = x[4,1]*x3	
s= x[5,1] +(~x1*x2+x1*~x2)*~x3	x[1,3]=~x1	x[1,1]
s= x[5,1] +(x[1,1]*x2+x1*~x2)*~x3	x[2,3]= x[1,1]*x2	x[2,1]
s= x[5,1] +(x[2,1]+x1*~x2)*~x3	x[1,3]= ~x2	x[1,2]
s= x[5,1] +(x[2,1]+x1*~x2)*~x3	x[2,3]= x1*x[1,2]	x[2,2]
s= x[5,1] +(x[2,1]+ x[2,2])*~x3	x[3,2]= x[2,1]+x[2,2]	x[3,1]
s= x[5,1] + x[3,1]*~x3	x[1,3]= ~x3	
s= x[5,1] + x[3,1]*x[1,3]	x[4,2]= x[3,1]*x[1,3]	
s= x[5,1] + x[4,2]	x[6,1]=x[5,1]+x[4,2]	
s= x[6,1]		
c= (~x1*x2+x1*~x2)*x3+x1*x2	x[1,4]=~x1	x[1,1]
c= (x[1,1]*x2+x1*~x2)*x3+x1*x2	x[2,3]= x[1,1]*x2	x[2,1]
c= (x[2,1]+x1*~x2)*x3+x1*x2	x[1,4]= ~x2	x[1,2]
c= (x[2,1]+x1* x[1,2])*x3+x1*x2	x[2,3]=x1* x[1,2]	x[2,2]
c= (x[2,1]+x[2,2])*x3+x1*x2	x[3,2]= x[2,1]+x[2,2]	x[3,1]
c= x[3,1]*x3+x1*x2	x[4,3]= x[3,1]*x3	
c= x[4,3]+x1*x2	x[1,4]=x1*x2	
c= x[4,3]+ x[1,4]	x[5,2]= x[4,3]+ x[1,4]	
c=x[5,2]		

3.3 Composition of Logical System from Substrings

Composition of a logical system comes out of a system of substrings obtained by a decomposition of an algebraic expression (section 3.2). The result of the composition is a system of algebraic expressions of all system modules output functions. Except the system of substrings, the degree of modules, which the composed circuit should consist of, is the input for the algorithm. Determination of module degrees depends upon the task which is the de/composition done for. The module levels can be equal

or different for composed modules. The modules composition must be without overlapping of modules.

An algorithm of a logical module composition from substrings representing one-level circuits can start from primary outputs, primary inputs of circuit or points which present inputs for more than one modules. One algorithm of a logical module composition, which starts from primary outputs, is following (example of an application of the presented algorithm for a full adder circuit is presented in the Table 2):

Table 2. Composition of full adder algebraic expression from substrings

Function	**Substitution**
s= *x*[6,1] s= *x*[5,1] + *x*[4,2] s= *x*[4,1]*x3+ *x*[4,2] s= =~ *x*[3,1]*x3+ *x*[4,2] s= =~ *x*[3,1]*x3+ *x*[3,1]**x*[1,3] s= =~ *x*[3,1]*x3+ *x*[3,1]* ~x3	*x*[6,1]=*x*[5,1]+*x*[4,2] *x*[5,1] = *x*[4,1]*x3 *x*[4,1] =~ *x*[3,1] *x*[4,2]= *x*[3,1]**x*[1,3] *x*[1,3]= ~x3
x[3,1]= *x*[2,1]+*x*[2,2] *x*[3,1]= *x*[1,1]*x2+*x*[2,2] *x*[3,1]= ~*x*1*x2+*x*[2,2] *x*[3,1]= ~*x*1*x2+ x1*x[1,2] *x*[3,1]= ~*x*1*x2+ x1*~*x2*	*x*[2,1]= *x*[1,1]*x2 *x*[1,1]=~*x*1 *x*[2,2]= x1*x[1,2] *x*[1,2]=~*x2*
c=*x*[5,2] c=*x*[4,3]+ *x*[1,4] c=*x*[3,1]*x3+ *x*[1,4] c=*x*[3,1]*x3+ x1*x2	*x*[5,2]= *x*[4,3]+ *x*[1,4] *x*[4,3]= *x*[3,1]*x3 *x*[1,4]=x1*x2

INPUT:

A system of algebraic expression substrings that contain exactly one logical operation (section 3.2).

Degree of modules, which the resulting circuit should consist of.

OUTPUT:

System of algebraic expressions that represent circuit output functions and system modules output functions.

BODY:

9. Let *n* be the depth of function embedding.
   Let *m* be the depth of modules function embedding (degree of modules) that resulting circuit should consist of.
   If (*n* mod *m*) = 0, let *m_temp* = *n* - *m*, *k* = *m*, else *m_temp* = *n* - (*n* mod *m*), *k* = (*n* mod *m*).
10. Beginning from a variable of the highest level, continuously substitute all variables of higher level than *m_temp* by strings that contain variables of lower level.
11. In this expression, replace variables with level lower than *m_temp* by substitution of particular substrings. This substitution should be performed until the number of substitutions in substring in which the corresponding variable belongs to is not higher than *k*. The obtained expression presents the resulting function of module.

12. In this expression, if exists no variable $x[p,q]$, $p > 0$ then stop else:

4.1. For every variable $x[p,q]$, $p > 0$ do the following:

if $p \geq m$, let $m\_temp = p - m$, else $m\_temp = 0$.

4.2. Let $k = m$, proceed to step 2.

Functions of outputs $s = x[6,1]$, $c = x[5,2]$ gained by substitutions presented in section 3.2 are inputs for the algorithm. We chose $m = 3$, because the circuit has to be realized by logical elements NOT, AND, OR, and gain a circuit consisting of three-level modules this way.

We can find out that the module M1 with output *x[3,1]* is isomorphic to the module M2 with output *s*, what becomes interesting from the diagnostic point of view [5]. The way of determination of identical and isomorphic modules is described in section 4. Structural scheme of a de/composed full adder circuit is shown in Fig. 1.

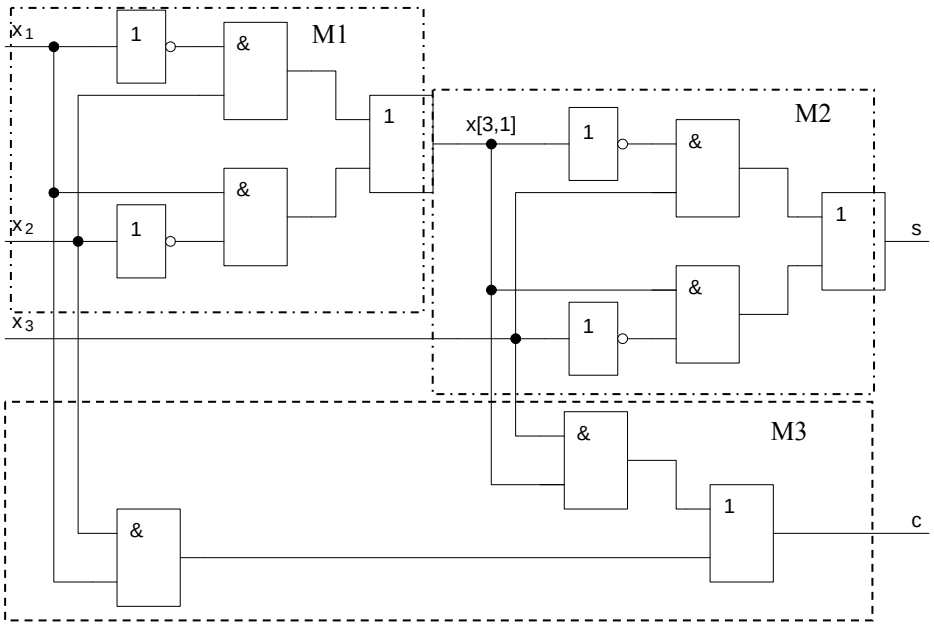

Fig. 22. Structural scheme of a de/composed full adder circuit.

4 Determination of Identical and Isomorphic Circuits

Determination of identical circuits is executed in the process of corresponding strings decomposition to one-level substrings (section 3.2). By comparison of expressions obtained this way, we can tell if the given circuits are identical.

Determination of isomorphic circuits is executed after system modules composition (section 3.3). For the determination of isomorphism of the strings it is necessary to find out, if the conditions of identity of corresponding operation (type of the operator and number of operand) and also the condition of variables substitution

are fulfilled. Detection of the last mentioned condition is very demanding, because the number of different substitutions is equal to the number of variables permutations. That's why the condition of substitution is detected only when all other conditions are fulfilled.

The number of substitutions can be lowered, if we consider only permutations that present only those variables that are inputs of the logical operations with the same operator and same number of variables.

One algorithm for decomposition of circuit into modules and for modules isomorphism determination was implemented in [6], [7].

5 Conclusion

Decompositions of logical systems specified by algebraic expressions of its function are proposed in the contribution. The degree of modules is input of algorithm of modules composition. We can repeat modules composition with different degree of modules and look for suitable decomposition.

Acknowledgments

This work is the result of the project implementation: Development of the Center of Information and Communication Technologies for Knowledge Systems (ITMS project code: 26220120030) supported by the Research & Development Operational Program funded by the ERDF.

References

1. Korečko, Š., Hudák, Š., Šimoňák, S.: Formal Methods Integration for Design and Analysis of Time-critical Systems. Informatika, Bratislava, pp. 211--216 (2003)
2. Chladný, V., Havlice, Z., Szaniszló, P.: Modeling Tools Description Language. In Proceedings of the International scientific conference MOSIS in Rožnov pod Radhoštěm, Czech Republic, pp. 107--112 (2000)
3. Bača J., Chladný V., Giertl J.: Formal Specifications and Decomposition of Logic Systems for Purposes of Analysis, Synthesis and Diagnostics, Problemy programmirovanija 16, 2-3, pp. 102--107 (2004)
4. Bača, J.: Decomposition of Logic Circuits. In Proceedings of International Conference Electronic Computers and Informatics '98, FEI TU Košice-Herľany, pp.100--103 (1998)
5. Hlavička, J., Kottek. E., Zelený, J.: Diagnostika elektronických číslicových obvodů. SNTL, Praha (1982)
6. Hlinka, V.: Dekompozícia logických obvodov. Bakalárska práca, Košice, Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky (2011)
7. Dzubajová, B.: Dekompozícia logických obvodov. Bakalárska práca, Košice, Technická univerzita v Košiciach, Fakulta elektrotechniky a informatiky (2011)