UDC 004.047
DOI: 10.20535/SRIT.2308-8893.2023.1.11

APPROACH TO POSITIONAL LOGIC ALGEBRA

M. KOVALOV

Abstract. The method of Boolean function representation in terms of positional logic algebra in compact operator form is offered. Compared with the known method, it uses position operators with a complexity of no more than two and only one type of equivalent transformations. The method is less labor intensive. It allows parallelizing logic calculations. The corresponding way of Boolean function implementation is developed. It competes with some known ways in terms of hardware complexity, resource intensity, and speed when implemented on an FPGA basis. Possibilities open up for creating effective automating means of representing Boolean functions from a large number of variables, synthesizing the corresponding LCs, and improving modern element bases.

Keywords: boolean functions, positional logic algebra, positional operators, equivalent transformations, logic circuits, FPGA.

INTRODUCTION

The development of information systems involves a significant expansion and complication of logic calculations. Therefore, the development of new directions in the study and implementation of Boolean functions (BF), one of which is positional logic algebra of (PLA), seems promising. It includes principles and methods that allow BF representing and calculating using equivalent transformations and positional operators with polynomial complexity [1]. The application of positionality principles for arithmetic and parallelization of logic calculations is substantiated in PLA [1, 2]. Therefore, effective application of its apparatus is possible for artificial intelligence, pattern recognition, cryptography and etc., where it is necessary to operate BF from a large number of variables.

However, the following problems might be identified within PLA framework:

- cumbersome apparatus of sequentially performed equivalent transformations α-, β-, γ-, μ, *i*-inversions; τ-permutations; multi-parametric Δ-, λ-, Θ-, ω-transformations) and complex positional operators). The known method of BF representing from *n* variables [1] actually includes cumbersome analysis of many *n* order operators. The operator, which generates the most similar BF and has identical level, is chosen from them. After that with no less difficulty a sequence of multi-parameter equivalent transformations is selected. Therefore, for large *n* value, the application of the method is significantly labor-intensive;
- researches within PLA framework were mainly theoretical (solving a complex systems of logic equations, determining ratios of NP- and P-complete problem classes, etc.). Its practical application has not been considered enough [2, 3].

In view of these problems, it is reasonable not to idealize PLA and to oppose it to Boolean algebra, but interaction. Therefore, a less labor-intensive method of BF representing was proposed. It does not involve enumeration of solutions, but the formation process of less complex operators, only one type of equivalent

transformations and some Boolean algebra apparatus. Also, prerequisites are created for logic calculations parallelization. In accordance with the method hardware implementation of the BF was developed, which competes with some known methods in terms of hardware complexity, resource intensity and speed.

BF REPRESENTATION

The essence of the offered method is in the following. An arbitrary BF is covered by its fragments given by the corresponding conjunctions. For each of them, a simple positional operator can be formed, which generates closed BF of the fragment, which must be corrected to obtain exact BF values:

$$f_i = \left(\bigwedge_{t=1}^{n-k} \widetilde{x}_{\alpha_t} \right) \wedge (f_{cori}^{1}(X_k) \vee (\boldsymbol{S}_{j_i}^{k}[X_k] \wedge f_{cori}^{0}(X_k))), \qquad (1)$$

here X_k — input argument vector with *k* digit capacity; $\left(\bigwedge_{t=1}^{n-k} \widetilde{x}_{\alpha_t} \right)$ — $(n-k)$-rank conjunction ($\alpha_t \in \overline{1,n}$), which defines f_i fragment; $\boldsymbol{S}_{j_i}^{k}$ — simple positional operator (j_i — binary vector of operator with dimension $(k+1)$, $0 \le j \le 2^{k+1}-1$; k — operator order, $k \le n$), which generates the BF-prototype (the closest to the f_i). BF-correction f_{corri}^{1} has "1" values on those X_k vectors, where $\boldsymbol{S}_{j_i}^{k}$ has "0" values but f_i has "1" values; BF-correction f_{corri}^{0} has "0" values on those X_k vectors, where $\boldsymbol{S}_{j_i}^{k}$ has "1" values but f_i has "0" values.

The covering process begins with shaping and analysis of great fragments. With a large number of differences between BF-prototypes and fragments its sizes should be reduced. For this reason, the BF might be covered by the operator's records (1) fast enough. If we disjunctively combine records (1) and transform the resulting expression to the operator form, it will be received a representation of the original BF in PLA terms. Obviously, it is necessary to minimize complexity of such representation. There is no general decision of this minimization task. Larger fragments may have many corrections, but smaller fragments may result in a large number of fragments and records (1) respectively. Therefore, the following rule is developed. If f_{corri}^{1} and f_{corri}^{0} don't require large number of operators (usually two), it is not further defragmentation. And according record (1) includes in BF representation. On the other hand, the method does not need more complex transformations of input arguments as inverting. For this reason, from large number of equivalent transformations it is expedient to use one-parameter λ_w transformation. It inverts binary digits of argument corresponding to "1" in binary *w* code. For example, $\bar{a}b\bar{c}d = S_{16}^{4}\lambda_{10}[abcd]$. It is obvious that the implementation of such transformation is trivial. So, the process of BF representing is as follows:

1. The set F of fragment representations in the form (1) that cover BF is assumed to be empty. In set G of conjunctions that define analyzed fragments include initial BF Z.
2. If set G is empty, then go to step 9.

3. For each analyzed fragment in the set G generate:

– a binary vector $j=(j_0...j_k)$ of a simple positional operator $\boldsymbol{S}_{j_l}^{k}$ that generates the BF-prototype, the closest to $f_{pr\,l}(X_k)$. Here $j_m=1 \quad (0\le m\le k)$, if the case number when f_l=1 on vectors X_k with m 1's digits exceeds half of the total number of such vectors, else $j_m=0$;

– estimate the operator number for describing of BF corrections $f^1_{corr\,l}(X_k)$ and $f^0_{corr\,l}(X_k)$ based on mismatches between f_l and $f_{pr\,l}(X_k)$.

4. Select a fragment f_s with the simplest records of BF corrections.

5. If $f^1_{corr\,l}(X_k)$ and $f^0_{corr\,l}(X_k)$ records in PLA terms don't require large number of operators (typically two) then go to step 7.

6. Include in set G all conjunctions of the next $(n-k+1)$-rank, which define non-empty f_s fragments. Go to step 8.

7. The selected fragment f_s written in form (1) include in the set *F*.

8. Exclude from G conjunction defining f_s and alternative conjunctions obtained with it in step 7, except the conjunction with which chosen conjunction can be glued. Go to step 2.

9. Disjunctively merge all records in form (1) of fragments included in the set *F*, obtaining a combined original BF representation. It includes positional operators and logic operations of Boolean algebra:

$$Z=\bigvee_{i=1}^{r} f_i, \tag{2}$$

where *r* — number of resulting fragments $f_i \quad (1\le i\le r)$.

10. Open all brackets in (2) and simplify it using Boolean algebra relations.

11. The original BF representation obtained in previous step is finally reduced to an operator form using relations between PLA and Boolean algebra, for example:

$$\widetilde{a}_n\wedge...\wedge\widetilde{a}_1=S^n_{2^n}\lambda_w[a_n...a_1]; \tag{3}$$

$$a_n\vee...\vee a_1=S^n_{2^{n+1}-2}[a_n...a_1]; \tag{4}$$

$$\widetilde{a}_n\wedge...\wedge\widetilde{a}_{k+1}\wedge S^k_j[a_k...a_1]=S^{n-k}_{2^{n-k+1}}S^k_j\lambda_w[a_n...a_1]. \tag{5}$$

Example. Let some BF $F=f(X_5)$ has values "1" on vectors 1–3, 13–15, 19, 21–23 and 25–28. Following the known method [1], BF can be represented by a complex positional operator and a sequence of equivalent two-parameter transformations like this:

$$Z=S^1_4S^4_{14}\omega^{17}_{30}\omega^{19}_{29}\omega^{21}_{24}\omega^{23}_{20}\omega^{16}_{18}\omega^{16}_{17}[x5x4x3x2x1]. \tag{6}$$

The BF representation following the proposed method looks like:

s. 1: $F=\{\};G=\{Z\}$.

Cycle 1:

s. 2: G is non-empty, go to s. 3.

s. 3–5: for Z form: $f_{pr\,Z}(x5x4x3x2x1) = \boldsymbol{S}_{24}^{5}[x5x4x3x2x1]$. $f_{corr\,Z}^{1}(x5x4x3x2x1)$ has values "1" on vectors 1–3, $f_{corr\,Z}^{0}(x5x4x3x2x1)$ has values "0" on vectors 1–3, 7, 11, 29, 30. To record them more than 2 operators are required, hence go to s. 6.

s. 6, 8: $G = \{x1, x2, x3, x4, x5, \overline{x1}, \overline{x2}, \overline{x3}, \overline{x4}, \overline{x5}\}$.

Cycle 2:

s. 2: G is non-empty, go to s. 3.

s. 3–5: the fragment defined by the conjunction $\overline{x1}$ requires the simplest corrections. For it $f_{pr\,\overline{x1}}(x5x4x3x2) = \boldsymbol{S}_{8}^{4}[x5x4x3x2]$. There is no need in $f_{corr\,\overline{x1}}^{0}(x5x4x3x2)$, only one operator is enough for $f_{кор\,\overline{x1}}^{1}(x5x4x3x2)$ defining on vector 1 ($x5x4x3x2$=0001), hence go to s. 7.

s.7: write the selected fragment in the form (1) as $f_{\overline{x1}} = \overline{x1}(S_{8}^{4}[x5x4x3x2] \vee \overline{x5}\,\overline{x4}\,\overline{x3}x2)$. Hence $F = \{\overline{x1}(S_{8}^{4}[x5x4x3x2] \vee \overline{x5}\,\overline{x4}\,\overline{x3}x2)\}$.

s. 8: $G = \{x1\}$.

Cycle 3:

s. 2: G is non-empty, go to s. 3.

s. 3–5: for the fragment defined by conjunction $x1$, form $f_{pr\,x1}(x5x4x3x2) = \boldsymbol{S}_{13}^{4}[x5x4x3x2]$. $f_{corr\,x1}^{1}(x5x4x3x2)$ has value "1" on vector 1, a $f_{corrx1}^{0}(x5x4x3x2)$ — has value "0" on vectors 3, 5, 14. More than 2 operators are required to write them, hence go to s. 6.

s. 6, 8: $G = \{x1x2, x1x3, x1x4, x1x5, x1\overline{x2}, x1\overline{x3}, x1\overline{x4}, x1\overline{x5}\}$.

Cycle 4:

s. 2: G is non-empty, go to s. 3.

s. 3–5: the fragment defined by the conjunction $x1\overline{x2}$ does not require correction. For it $f_{pr\,x1\overline{x2}}(x5x4x3) = \boldsymbol{S}_{5}^{3}[x5x4x3]$. Go to s. 7.

s. 7: write the selected fragment in the form (1) as $f_{x1\overline{x2}} = x1\,\overline{x2}S_{5}^{3}[x5x4x3]$. Hence $F = \{\overline{x1}(S_{8}^{4}[x5x4x3x2] \vee \overline{x5}\,\overline{x4}\,\overline{x3}x2), x1\,\overline{x2}S_{5}^{3}[x5x4x3]\}$.

s. 8: $G = \{x1x2\}$.

Cycle 5:

s. 2: G is non-empty, go to s. 3.

s. 3–5: for the fragment defined by conjunction $x1x2$, form $f_{prx1x2}(x5x4x3) = \boldsymbol{S}_{5}^{3}[x5x4x3]$. $f_{corr\,x1x2}^{1}(x5x4x3)$ has value "1" on vector 4, only one operator is enough for it, there is no need in $f_{кор\,x1x2}^{0}$. Go to s. 7.

s. 7: write the selected fragment in the form (1) as $f_{x1x2} = x1x2(S_5^3[x5\,x4\,x3] \vee x5\,\overline{x4}\,\overline{x3})$. Therefore $F = \{\overline{x1}(S_8^4[x5\,x4\,x3\,x2] \vee \overline{x5}\overline{x4}\overline{x3}x2),$ $x1\overline{x2}S_5^3[x5x4x3], x1x2(S_5^3[x5x4x3] \vee x5\overline{x4x3})\}$.

s. 8: $G = \{x1\}$.

The set G is empty. Entire original BF is covered with fragments from the set F, so go to s. 9.

s. 9: Get first Z representation:

$$Z = \overline{x1}(S_8^4[x5x4x3x2] \vee \overline{x5}\overline{x4}\overline{x3}x2) \vee x1\overline{x2}S_5^3[x5x4x3] \vee$$

$$\vee\, x1x2(S_5^3[x5x4x3] \vee x5\overline{x4x3}). \qquad (7)$$

s. 10: Open brackets in (7):

$$Z = \overline{x1}S_8^4[x5x4x3x2] \vee \overline{x5}\overline{x4}\overline{x3}x2\overline{x1} \vee x1\overline{x2}S_5^3[x5x4x3] \vee$$

$$\vee\, x1x2S_5^3[x5x4x3] \vee x5\overline{x4x3}x2x1.$$

Glue 3rd and 4th conjunctions and select common variables in the 2nd and 5th conjunctions:

$$Z = \overline{x1}S_8^4[x5x4x3x2] \vee \overline{x4x3}x2(\overline{x5x1} \vee x5x1) \vee x1S_5^3[x5x4x3]. \qquad (8)$$

s. 11: expression in brackets of the second term in (8) might be written as $S_5^2[x5x1]$, for each conjunction, use (5). It is necessary to use corresponding one-parameter transformations λ_w for variable inverting (3):

$$Z = S_4^1S_8^4\lambda_{16}[x1x5x4x3x2] \vee S_{16}^3S_5^2\lambda_{24}[x4x3x2x5x1] \vee S_4^1S_5^3[x1x5x4x3].$$

Now apply (4) and finally get original BF representing as:

$$Z = S_{14}^3[S_4^1S_8^4\lambda_{16}[x1x5x4x3x2], S_{16}^3S_5^2\lambda_{24}[x4x3x2x5x1], S_4^1S_5^3[x1x5x4x3]]. \qquad (9)$$

This example shows that the internal operators in complex positional operators serve to set the BF-prototypes of fragments, and the external ones set conjunctions that uniquely determine these fragments. Therefore, the complexity of positional operators in such representing in most does not exceed two, while maintaining their compactness.

According to representation (9), it is possible to build a flow graph of logic calculations (Fig. 1). If it is assumed that transformation and operator are performed during the same time, then the calculation of the BF will take 4 steps. The sequential process of calculating the same BF according to expression (6) lasts 8 steps. It might also be seen that, compared with the known method, the proposed method also allows parallelize execution of positional operators and equivalent transformations, reducing the time for BF calculating.

It is difficult to compare the complexities of the proposed and known [1] methods directly. However, it might be done by evaluating the number of analyzed fragments in the maximum case and positional operators of order n. In accordance with the proposed method, first, the entire original BF is considered,

then — $(2n)$ fragments depend on (n–1) variables, then — $(2^2(n-1))$ fragments on (n–2) variables, etc. Therefor the maximum number of fragments is evaluated as $N_f \approx 2^{n+1}$. Estimating the number of operators analyzed in accordance with the known method reduces to the combinatorial task of partitioning the number n into terms. Using the Hardy-Ramanujan formula [4] and considering that for any complexity and order n it might be build (2^{n+1}) operators at least, get $N_{op} \approx \frac{2^{n+1} e^{\pi\sqrt{\frac{2n}{3}}}}{4n\sqrt{3}}$ all types positional operators. Obviously, the developed method is significantly less labor intensive.

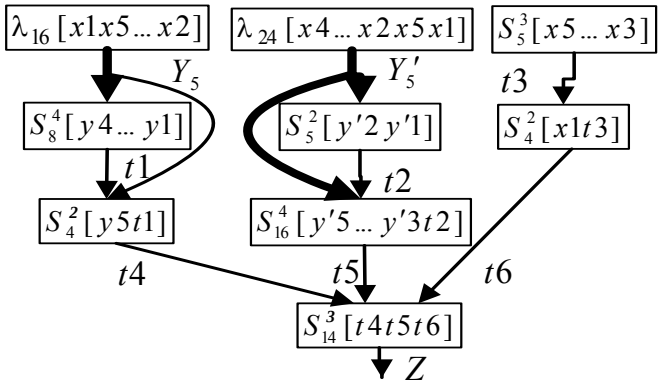

Fig. 1. Flow graph of logic calculations $Z = f(X_5)$

HARDWARE IMPLEMENTATION

The BF implementation in accordance with the proposed method is a combinational logic circuit (LC). It should be based on the form (2). For example, hardware implementation of the conjunctions and disjunctions in (1) and (2) is more efficient directly using logic elements. And BF corrections are implemented in accordance with their disjunctive normal form. The maximum number of fragments in form (2) is a power of two. Hence the same number of identical functional blocks (FBs) is applicable for its implementation, as well as the corresponding number of other logic blocks and elements. Thus, structure of such LC (Fig. 2) includes:

(2^{n-k-1}) blocks for determining the number of "1" in the input argument vector X_k;

(2^{n-k}) FBs for fragments f_i implementing;

OR logic element 4 for disjunction implementing in accordance with the form (2).

Each FB contains:

group of logic elements 1, which with the corresponding block for determining the number of "1" implements a simple positional operator $\boldsymbol{S}_{j_i}^k[X_k]$ for current fragment;

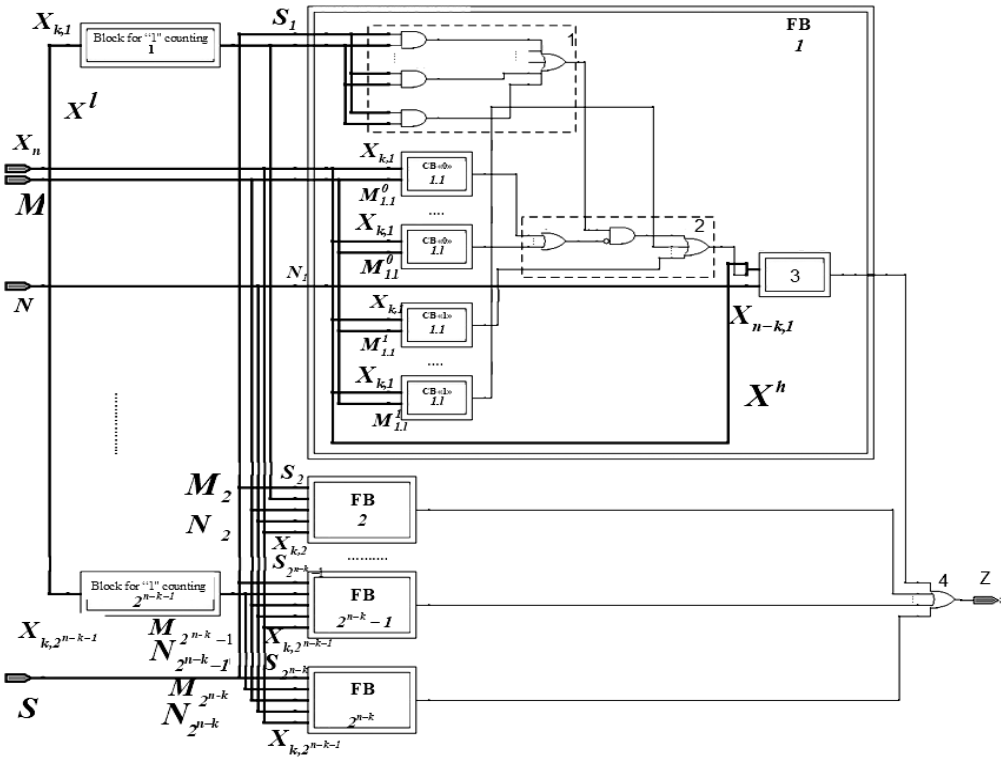

Fig. 2. LC for the BF implementation by the proposed method

l correction blocks (CBs "0" or "1"), that implement respectively BF-corrections $f^0_{corr\,i}(X_k)$ and $f^1_{corr\,i}(X_k)$ with a group of logic elements 2. Each of them is a *k*-input logic element AND with controlled input inverters;

block 3 of logic element AND, one input of which is the f_i value. Remaining (*n-k*) inputs have controlled inverters for the implementation of the conjunction $\left(\bigwedge_{t=1}^{n-k} \tilde{x}_{\alpha_t}\right)$ accordance with (1).

Input signals of the LC:

input vector of arguments X_n (input *X*);

2^{n-k} vectors of positional operators (input *S*);

2^{n-k} control signals for CB inverters (input *M*);

2^{n-k} control signals for inverters of block 3 (input *N*).

The number of BFs, calculable by the LC, is directly determined by thes number and bit capacity *k* of FBs, CBs:

$$N_{BF\,PAL} \approx (N_{BF\,fr}(k,l))^{2^{n-k}} \prod_{i=0}^{n-k-1} (n-i)^{2^i}, \qquad (10)$$

where $N_{BF\,fr}(k,l)$ – the number of BFs depending on *k* variables, which, taking into account *l* CBs "0" and "1", are calculated by one FB.

Being a complex combinatorial dependence $N_{BF\,PAL} >> 2^{n+1}$ (number of all simple positional operators). Therefore, this LC is capable to calculate a large number of practically used regular BFs depend on *n* variables. In addition, many partially defined BFs might be extended to a form convenient for calculating on similar LCs.

The following hardware characteristics of the proposed LC were defined:

– complexity as a number of two-input logic elements:

$$L(n,2)_{PAL} = 2^{n-k-2}(3k^2 + 4n + k(8l+5)) - 1; \quad (11)$$

– LC depth defined as a number of cascades on signal way from input to output through the block for determining the number of "1", groups 1, 2 of logic elements and block 3:

$$T_{PAL} = n + k +]\log_2 k[+]\log_2 (n-k)[+ 1; \quad (12)$$

– input numbers:

$$N_{in\,PAL} = n + 2^{n-k}(2n + k(2l-1) + 1). \quad (13)$$

The developed LC allows to simply change the number and bit capacity of the main logic blocks. It allows flexible change the ratio between its functionality and hardware requirements, that follows from (10)–(13).

Let's implement BF considered in the example by this LC. According the LC structure, it is necessary transform (7) so that all operators have same order 3. Simple positional operator $\boldsymbol{S}_j^k[X_k]$ might be represented with simple operators of smaller order in the next way:

$$\boldsymbol{S}_j^k[x_k \ldots x_i \ldots x_1] = x_i \boldsymbol{S}_{j1}^{k-1}[x_k \ldots x_{i+1} x_{i-1} \ldots x_1] \vee \overline{x_i} \boldsymbol{S}_{j2}^{k-1}[x_k \ldots x_{i+1} x_{i-1} \ldots x_1]. \quad (14)$$

Using (14), represent the first operator in (7) as:

$$S_8^4[x5x4x3x2] = x2S_8^3[x5x4x3] \vee \overline{x2} S_4^3[x5x4x3].$$

Transform (7) to the form:

$$Z = \overline{x1}x2(S_8^3[x5x4x3] \vee \overline{x5}\,\overline{x4}\,\overline{x3}) \vee \overline{x1}\,\overline{x2} S_4^3[x5x4x3] \vee$$

$$\vee x1\overline{x2} S_5^3[x5x4x3] \vee x1x2(S_5^3[x5x4x3] \vee x5\overline{x4}\,\overline{x3}), \quad (15)$$

getting 4 fragments described by operators $S_8^3, S_4^3, S_5^3, S_5^3$. The first and last of them are corrected by $f_{corr1}^1 = \overline{x5}\,\overline{x4}\,\overline{x3}$ and $f_{corr4}^1 = x5\overline{x4}\,\overline{x3}$ respectively. Therefore, for BF calculating in the form (15) it is suitable LC that includes 3-bit 2 blocks determining the number of "1" and 4 FBs, including one CB per block. The input variables are inverted according to the input control signals M and N. Zero values might be entered to the information inputs of unused CBs of FBs 2 and 3. Thus, LC inputs (Fig. 2) are:

blocks determining the number of "1": $\boldsymbol{X_1^h} = \boldsymbol{X_2^h} \equiv x1x2$; $\boldsymbol{X_1^l} = \boldsymbol{X_2^l} \equiv$ $= x5x4x3$;

FB 1: $\boldsymbol{S_1} \equiv 1000$, $\boldsymbol{M_1} \equiv 111$, $\boldsymbol{N_1} \equiv 10$;
FB 2: $\boldsymbol{S_1} \equiv 0100$, $\boldsymbol{M_2} \equiv 000$, $\boldsymbol{N_2} \equiv 11$;
FB 3: $\boldsymbol{S_3} \equiv 0101$, $\boldsymbol{M_3} \equiv 000$, $\boldsymbol{N_3} \equiv 01$;
FB 4: $\boldsymbol{S_4} \equiv 0101$, $\boldsymbol{M_4} \equiv 011$, $\boldsymbol{N_4} \equiv 00$.

PRACTICAL RESEARCH

In practical research, in addition to the developed method some known ways of BF implementation [5] were considered. They implement any BF depend on n variables with optimal hardware complexity:

– multiplexer with $(n-1)$ selector inputs based on a rectangular decoder [6]. The values of its hardware complexity, depth and number of inputs are:

$$L(n,2)_{MUX} = 3\left(2^{n-1} + 2^{\frac{n}{2}} - 3\right), \tag{16}$$

$$T_{MUX} = \frac{3n}{2}, \tag{17}$$

$$N_{in\ MUX} = 2^{n-1} + n - 1; \tag{18}$$

– LC based on the cascade method [7]:

$$L(n,2)_{casc} = 3(2^{n-1} - 1), \tag{19}$$

$$T_{casc} = 2(n-1), \tag{20}$$

$$N_{in\,casc} = 2^{n-1} + n - 1. \tag{21}$$

Dependences of hardware complexity, depth, number of LC inputs are based on (10)–(13), (16)–(21). Also, for the developed LC structure with a different number of FBs and CBs (l CB "0" and "1" in each FB) the number of countable BFs depend on the number of variables n is obtained. The parameters of resource intensity and speed of the FPGA-based (Field Programmable Gate Array) implementation of the proposed LC were gotten. Schemes were described on VHDL, synthesis and modeling were carried out using Intel Quartus Prime and Siemens Modelsim. Comparative studies are made.

Dependencies on Fig. 3 and 4 (given on a logarithmic scale) show that FB number increasing (a k decreasing at invariable n) leads to the significant increase of the calculable BF number and hardware complexity of the proposed LC (dependencies 2, 5 and 7). However, it makes sense to increase, the number of CBs within the FB. This can increase the functionality of the proposed LC to the level provided by a large number of FBs, but with lower hardware costs (dependencies 4–6). When $n > 7$ developed LC structure provides an overwhelming advantage sod.

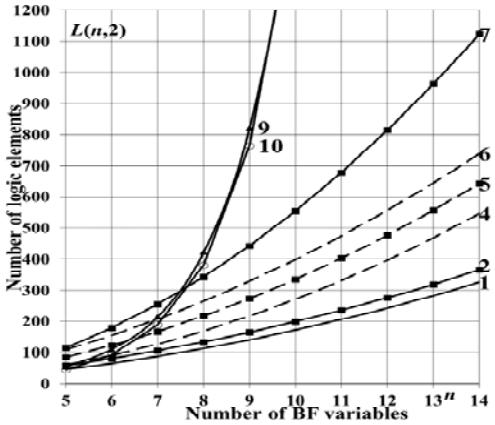

Fig. 3. Hardware complexity of *LCs*

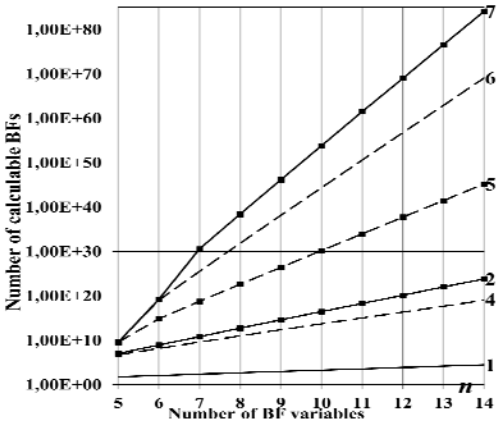

Fig. 4. Number of calculable *BFs*

The resource intensity of the FPGA-based LC implementations (Fig. 5) at most corresponds its hardware complexity (Fig. 3). When $n > 10$ the proposed LC provides a significant advantage over the known ways also in the LC input numbers (Fig. 6). From these dependences it is shown that for functionality expanding and equipment minimizing more effectively increase the CB number.

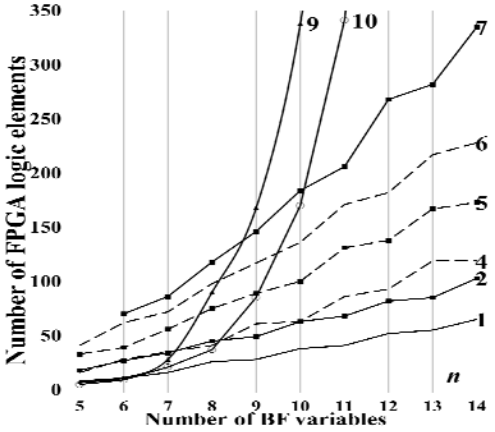

Fig. 5. Resource intensity of the LC integral implementations

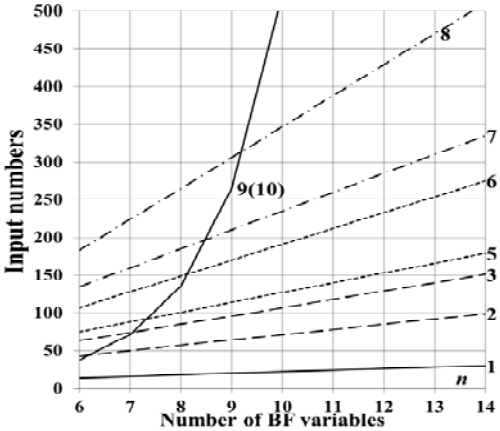

Fig. 6. LC input numbers

The depth of the developed LC (Fig. 7) depends mainly on n, so the dependences 1, 2, 5 and 7 are close to each other. A similar trend is observed for the performance of LC integral implementations (dependencies 1, 2, 4–7 on Fig. 8). Although the LCs based on multiplexer or cascade method, when implemented on FPGA basis, have less depth, but the difference in performance is practically leveled. The proposed LC structure may provide higher performance when a large number of variables (n>11).

CONCLUSIONS

In comparison with the known method the developed method of boolean function representation in terms of PLA is characterized by the following:

– uses positional operators with complexity no more than two and only one type of equivalent transformations;

– less labor intensive and compactness of BF representation;

– allows to parallelize execution of equivalent transformations and operators, reducing BF calculation time.

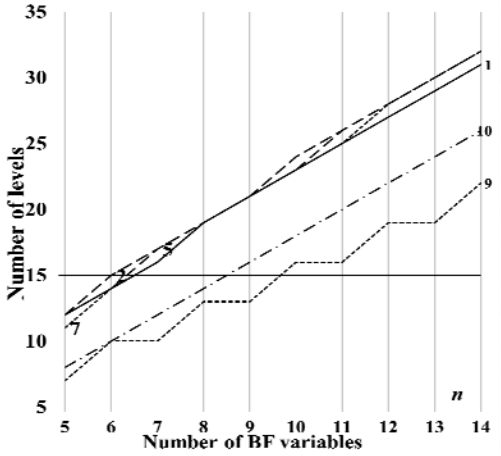

Fig. 7. Depth of the LCs

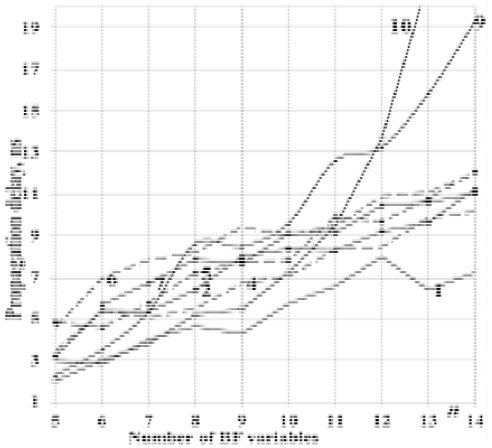

Fig. 8. Performance of the LC integral implementations

1 — proposed LC ($n-k=0, l=0$); 2 — proposed LC ($n-k=1,\ l=1$);
3 — proposed LC ($n-k=1, l=2$); 4 — proposed LC ($n-k=2, l=0$);
5 — proposed LC ($n-k=2, l=1$); 6 — proposed LC ($n-k=2, l=2$);
7 — proposed LC ($n-k=3, l=1$); 8 — proposed LC ($n-k=3, l=2$;
9 — multiplexer; 10 — LC based on the cascade method

The developed LC structure, in comparison with some known ways of BF implementation, has a significantly fewer hardware complexity when $n>7$. As a result, corresponding FPGA-based implementation also requires fewer logic resources and input number without losing in performance. To expand its functionality, it is more efficient to increase the number of CBs in FBs. The regularity and scalability of the LC structure provide effective control the involvement in operation process its parts by using, for example, "operand isolation" technology [8, 9]. This creates the prerequisites to change flexible the ratio between LC functionality, technical and economic parameters of the integral implementation, "bypassing" possible failures, increasing its reliability. In addition, possibilities for creating effective automating means of representing BF from a large number of variables, synthesizing the corresponding LCs and modern element bases improvement open up.

REFERENCES

1. M. Telpiz, *Algebra of positional operators and equivalent transformations*. M.: Scientific advice on a complex problem, 1988.
2. C.M. Hamann and L. Chtcherbanski, "Positional Logic Algebra — PLA – A Fascinating Alternative Approach,"*ICSI Technical Report TR-97-039*, Sep. 1997.
3. M.I. Telpiz, "Sigma-notation and the equivalence of p and np classes,"*J. Inf. Organ. Sci*., vol. 29, no. 2, Mar. 2012.
4. T. Jiang and K. Wang, "A generalized Hardy-Ramanujan formula for the number of restricted integer partitions,"*Journal of Number Theory*, vol. 201, pp. 322–353, Aug.

2019. Accessed on: Feb. 6, 2023. [Online]. Available: https://doi.org/10.1016/j.jnt.2019.02.006

5. D. Harris and S. Harris, *Digital Design and Computer Architecture: ARM Edition*. Morgan Kaufmann, 2015.
6. A.N. Borodzhieva, I.I. Stoev, and V.A. Mutkov, "FPGA Implementation of Boolean Functions Using Multiplexers," in *2019 IEEE XXVIII International Scientific Conference Electronics (ET), Sozopol, Bulgaria, Sep. 12–14, 2019*. Accessed on: Feb. 6, 2023. [Online]. Available: https://doi.org/10.1109/et.2019.8878504
7. A.N. Borodzhieva, I.I. Stoev, I.D. Tsvetkova, S.L. Zaharieva, and V.A. Mutkov, "FPGA Design of Boolean Functions Using a Cascade of Decoders and Logic Gates," in *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO), Opatija, Croatia, Sep. 28–Oct. 2, 2020, IEEE, 2020.* Accessed on: Feb. 6, 2023. [Online]. Available: https://doi.org/10.23919/ mipro48935.2020.9245448
8. A.A. M. Bsoul, S.J.E. Wilton, K.H. Tsoi, and W. Luk, "An FPGA Architecture and CAD Flow Supporting Dynamically Controlled Power Gating,"*IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 178–191, Jan. 2016. Accessed on: Feb. 6, 2023. [Online]. Available: https://doi.org/10.1109/tvlsi.2015.2393914
9. C. Ashok Kumar, B.K. Madhavi, and K.L. Kishore, "Enhanced Clock Gating Technique for Power Optimization in SRAM and Sequential Circuit,"*Journal of Automation, Mobile Robotics and Intelligent Systems*, pp. 32–38, Jan. 2022. Accessed on: Feb. 6, 2023. [Online]. Available: https://doi.org/10.14313/jamris/2-2021/11

Received 09.02.2023

INFORMATION ON THE ARTICLE

Mykola O. Kovalov, ORCID: 0000-0002-2590-4052, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Ukraine, e-mail: kovua@yahoo.com

ПІДХІД ДО ПОЗИЦІЙНОЇ АЛГЕБРИ ЛОГІКИ / М.О. Ковальов

Анотація. Запропоновано метод подання булевих функцій у термінах позиційної алгебри логіки в компактній операторній формі. Порівняно з відомим методом у ньому застосовуються позиційні оператори зі складністю не більше двох і лише одного виду еквівалентних перетворень. Метод відрізняється меншою трудомісткістю і розкриває паралелізм логічних обчислень. Запропоновано відповідний спосіб реалізації булевих функцій. Він становить конкуренцію деяким відомим способами за апаратною складністю, ресурсомісткістю та швидкістю із застосуванням базису FPGA. Відкриваються можливості для створення ефективних засобів автоматизації подання булевих функцій від великої кількості змінних, синтезу відповідних комбінаційних схем та вдосконалення сучасних елементних баз.

Ключові слова: булеві функції, позиційна алгебра логіки, позиційні оператори, еквівалентні перетворення, комбінаційні схеми, FPGA.