Nonlinear Compression Block Codes Search Strategy

Ondřej Novák

October 3, 2022

Nonlinear Compression Block Codes Search Strategy

Ondřej Novák
Institute of Information Technologies and Electronics
Technical University in Liberec
Liberec, Czech Republic
ondrej.novak@tul.cz

Abstract—This paper deals with extending linear compression codes by nonlinear check bits that improve the usability of decompressed patterns for testing circuits with more inputs. The earlier works used a purely random or partially random search of the nonlinear check-bits truth tables to construct the first nonlinear structures. Here, we derive deterministic rules that characterize the relationship among the nonlinear code check bits. The efficiency of the rules is demonstrated on different codes with the number of specified bits equal to three. The code parameters obtained after applying the rules overperform the parameters of the linear codes. Keeping the restrictions makes the search for the check bit truth tables faster and more efficient than can be got by a simple random search. The reached nonlinear block code (136,5,3) is the most efficient code among other loose compression codes.

Keywords — Test vector compression technique, Pseudo-exhaustive testing, Binary nonlinear codes, Graph theory, Minimum clique cover problem, Design for testability.

I. INTRODUCTION

This contribution deals with codes used for test pattern compression and expansion. The character of the circuits under test enables us to perform a loose compression and decompression. Only a few bits with given logical values must be delivered to specified input positions for testing a circuit under test [6]. Test patterns have considerable redundancy (a high number of don't care values in a test pattern), and thus they can be compressed. In testing a narrow stream of bits into n-bit test patterns applied to functional combinational circuit inputs or parallel scan chain inputs [1], [3], [9], [13], [16], [19], [20]. The compression scheme can be either application-dependent or universal [11]. Paper [12] abstracts the test vector redundancy and requires that the decompressor must be able to set any r-tuple of the generated test vector bits to arbitrary values. The decompressor is then specified by the number n of scan chains (decompressed test vectors width), the number of information bits (input width), and the number r (number of specified bits) only. This abstraction was used by other authors afterward. Using the linear block codes (LBC) theory, we can design either a Built-in Self-test equipment (BIST) or a decompressor that guarantees that arbitrary r-tuple within the n bits can be set to a random value. The best-known linear codes were tabulated [8]. We can find the minimum number of decompressor inputs that guarantee a given n and r and vice versa; we can see the maximum value of decompressor outputs n for given i and r. A code with n, i, and r may be denoted as an (n, i, r) code.

Codes with a high value of the minimum code distance of the dual code d_{min} are well suitable for combinational pattern compression [20], [5], [4]. The decompressor parameter r corresponds to the code's value d_{min}-1, dual to the applied codewords [20]. Unfortunately, LBCs provide a limited value of n for all values of r. Search for the best LBCs has a long history, and there is no hope that some new, substantially better codes will be discovered.

An extension to codes other than linear is obvious. They promise much more freedom to choose the decompressor function and cause a much more extensive search space. While using the NBC for error correction does not bring substantially better code parameters than it can be obtained for linear ones, the compression codes are considerably more effective when using nonlinear functions. This fact can be demonstrated by comparing the resulting linear and nonlinear code parameters given in Table. 1. The NBC codes are much more potent than the LBCs and are worthy of study. Comparing the LBC and NBC lengths, we can see that if the NBC decompressors are easily constructed by hardware means, they could represent a promising possibility of test pattern compression and decompression.

TABLE 1. THE MAXIMUM NONLINEAR [16][8] AND LINEAR [8] CODES

# of information bits i	# of specified bits r	Maximum LBC length n	Maximum NBC length n reached till now
4	3	8	12
5	3	16	128
6	3	32	536
	4	8	18
7	3	64	2500
	4	10	63
	5	9	13
8	4	13	531
	5	12	24
9	4	23	816
	5	14	111
	6	11	16
10	5	24	232
	6	17	54

In recent publications the nonlinear block codes (NBC) were investigated [14], [15], [16], [18]. These code structures are not systematically explored enough. The codes have complex and multiple dependencies among the code bits. It is impossible to perform an exhaustive search among all codes of a given size. For this reason, the examples of the larger codes are obtained with the help of a random generator. Thus, the prediction of the extremal code lengths cannot be simply made. Finding some creation rules that may reduce the search space without losing potentially efficient codes is challenging.

Paper [18] presents a systematic approach to suboptimal NBC construction. The authors have shown that the minimum NBC code search can be transformed into a Clique Cover Number (CCN) solution [10]. The CCN is a minimum number of graph partitions such that each graph subset induces a clique. The authors of [18] developed a methodology of finding *i* information bits (decompressor inputs) necessary for a given number *r* of specified bits within *n* outputs. The parameter *r* represents 2^r requirement cubes for each r-tuple of output bits within *n* outputs. Each requirement cube requires at least one output vector with *r* specific bits within the *n* outputs. All possible values of all potential r-bit output vectors correspond to a complete set of requirements. Requirement cubes may be identified with graph vertices and requirement cube's compatibility with the graph's edges. There exist compatible requirements, which means that these requirements can be fulfilled using one output vector only. These requirements form a clique. The compatible requirements have the same logical values on the corresponding output positions, or the requirements have a don't care bit on the places where some requirement has a defined value. Then, finding the CCN solves the problem of an NBC with a given *n* and minimal *i*. The CCN problem is NP-hard, in general. Therefore, no efficient algorithms were found for this kind of problem.

We may formulate the problem of finding a code with *r* specified bits as a problem of setting truth tables (TT) of code check bits. Within all r-tuples of check and information code bit TTs, there must be at least one row in which an arbitrary r-tuple of logical values can be found. NBC construction methodology was proposed in [14]. The check bit TTs were got by a random search within the space of all TTs with a balanced number of ones and zeros and checking whether the condition mentioned above holds. The check bit TTs were obtained by a step-by-step process. The algorithm was improved in [15] and [16]. It is advantageous to use an LBC with a given *i* and *r* and extend the code length by nonlinear code bits. Performing this, we can find particular rules that improve the probability of check bit TTs that complete the code words without any corruption of the number of specified bits. The search of the TTs is done randomly.

We do not know the Boolean expressions characterizing the TTs as there exists a substantial ambiguity of conditions that characterize the code structure. We can derive only examples of the Boolean terms for larger codes as there exist many parametric solutions to the problem. The search space is enormous. Let us consider having *p* check bits. The thorough examination among all possible check bit TTs has to exercise (2 exp (2 exp *i*)) exp *p* combinations of TTs whether they fulfill the condition of *r* specified bits among the whole codeword. A greedy algorithm was used to speed up the code search process for larger values of *i* and *p*. A step-by-step creation finds a check bit TT, fixes it, and does not use other possible TTs to find the complete code structure. There exist many code variations fulfilling the condition of *r* specified bits. For this reason, a random or semi-random search was performed in our previous code constructions [16]. This kind of search may hit an efficient code structure due to the high variability of successful TTs. It is possible to reduce the search space by preferring some TT shapes that were successful in the previous check bit TT searches and using variations for the subsequent check bit TT searches. We have found that using the rules that are advantageous to be followed to get a required code substantially helps to improve the code parameters.

This paper proposes a method of NBC creation. It uses an LBC and extends it by nonlinear check bits. This approach benefits from the LBC check bit structure as it balances logical values on all r-tuples and (r-1)-tuples. The regularity of logical values on (r-1)-tuples makes the following concatenation NBC check bits easier to keep the number of specified bits. Additional constraints are used to reduce the check bit TT search space. Following the rules substantially increases the obtained medium code length *n*. This fact is demonstrated by a randomly driven set of code creation experiments.

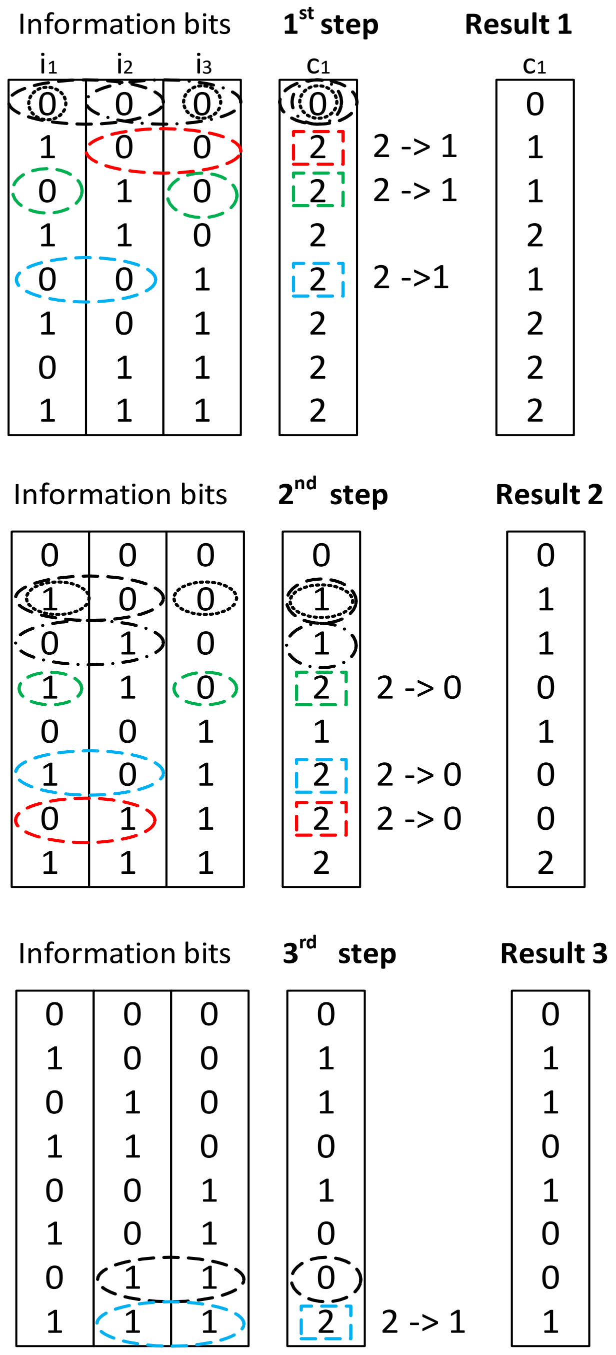
II. Code generation

We develop rules for codes with different numbers of information bits and the number of specified bits equal to three.

(4, 3, 3) code

Let us consider all information bit values; they are given in 8 rows of the columns i1. i2. and i3. Let us consider adding one check bit c1. In the beginning, the check bit TT column values are set to an unknown value denoted by symbol 2. At this point, there is not any restriction on choosing an arbitrary bit in a random row to an arbitrary logical value. If the bit is set, we have to check if it requires setting another bit to a specific value to keep the existence of all possible triplets within all code bits. This process is repeated until all bits are set.

The stages of code formation are illustrated in Fig. 1. Let us set the 1st row bit to log. 0. It is shown in the 1st step part of the figure. There exist three triplets of bits on the 1st row; they are denoted by the black circles (Differently dashed circles or ellipses mark bits of different triplets.) Setting the 1st bit in c1 to logical 0 implies the logical value 1 in row 2. The logical value is enforced because only two rows of the TT have the logical values in columns i2 and i3 equal to values in the already set row. This situation is denoted by red circles and a red square in the figure. The same problem is on rows 3 and 5, indicated by green and blue circles and squares. To guarantee the existence of the values (0, 0, 1) on the corresponding positions, we set the values in the check bit TT rows equal to 1. The resulting TTis of the 1st step is shown in column Result 1. The logical values are given in Result 1 part of the figure. The Result 1 logical values imply Result 2 values, and the Result 2 values condition Result 3. All the logical values given in Result 3 TT are necessary to be used in the code. All triplets containing the bit c1 reach all possible combinations of logical values. If there were a contradiction in the requirements, the code would not exist. Two different codes can be formed depending on the logical value chosen



Information bits			1st step		Result 1
i1	i2	i3	c1		c1
0	0	0	0		0
1	0	0	2	2 -> 1	1
0	1	0	2	2 -> 1	1
1	1	0	2		2
0	0	1	2	2 ->1	1
1	0	1	2		2
0	1	1	2		2
1	1	1	2		2

Information bits			2nd step		Result 2
0	0	0	0		0
1	0	0	1		1
0	1	0	1		1
1	1	0	2	2 -> 0	0
0	0	1	1		1
1	0	1	2	2 -> 0	0
0	1	1	2	2 -> 0	0
1	1	1	2		2

Information bits			3rd step		Result 3
0	0	0	0		0
1	0	0	1		1
0	1	0	1		1
1	1	0	0		0
0	0	1	1		1
1	0	1	0		0
0	1	1	0		0
1	1	1	2	2 -> 1	1

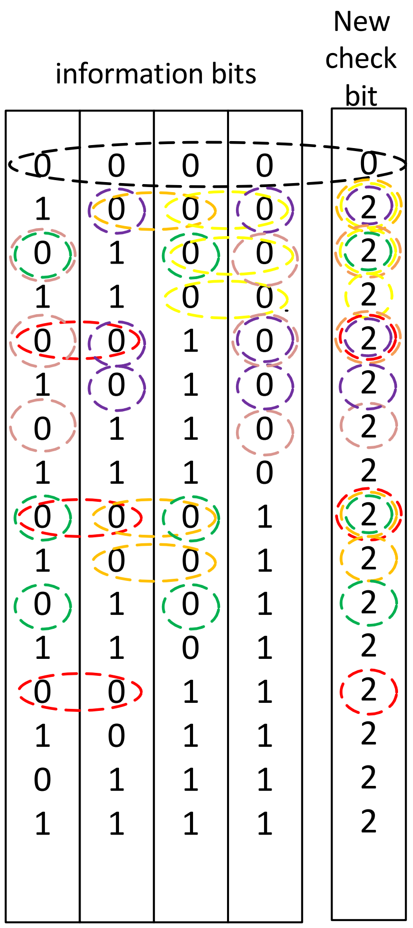
Figure 1. Step-by-step (4, 3, 3).code formation

in c1, 1st row. Setting the starting bit equal to 1 causes the check bit TT to have opposite values than we obtained in Result 3. It is impossible to add another check bit as the requirements contradict each other. If the starting bit was set to 0, the resulting check bit TT corresponds to the logical function $i_1 \oplus i_2 \oplus i_3$. It means that the formed code is linear. If the starting bit was set to 1, the code is nonlinear.

(12, 3, 3).code:

The ambiguity problem of restriction rules arises for this kind of code. It is illustrated in Figure 2. There exist six different triplets in each row of the TT. Let us choose to set the first row of the check bit TT to log. 0. An ellipse surrounds the bits forming the triplets influenced by the setting. Setting the bit of the check bit column causes a set of restrictions for other row settings. Colored circles denote the triplets in the influenced rows.

We have to set each check bit row marked by one or more used colors at least once. For this reason, it is not given which check bit TT row should be set to a logical value to guarantee the diversity of the triplet values in the considered columns. There is partial freedom in selection which restrictions should be followed and kept. A similar limitation holds for setting all rows in all check bit TTs. We need to find rules that guarantee to keep all necessary restrictions within a (12,4,3) code and simultaneously do not inhibit the extension of the number of check bits.



information bits				New check bit
0	0	0	0	0
1	0	0	0	2
0	1	0	0	2
1	1	0	0	2
0	0	1	0	2
1	0	1	0	2
0	1	1	0	2
1	1	1	0	2
0	0	0	1	2
1	0	0	1	2
0	1	0	1	2
1	1	0	1	2
0	0	1	1	2
1	0	1	1	2
0	1	1	1	2
1	1	1	1	2

Figure 2. The setting of the 1st bit of the check bit TT to log. value 0. Six triplets in the 1st row are denoted by one black ellipse. The influenced triplets are denoted by red, green, orange, blue, violet, and yellow.

LBCs have a very regular structure. It seems helpful to start with the LBC and complete it with the nonlinear check bits that do not reduce the number of specified codeword bits. For example, the LBC with *i*=4, *r*=3 has four check bits. The bit TTs of the codeword check bits can be constructed as truth tables of all possible three input XORs. There exist 4 XOR functions defining the check bit behavior, for example:

$i_1 \oplus i_2 \oplus i_3$, $i_1 \oplus i_2 \oplus i_4$ $i_1 \oplus i_3 \oplus i_4$ $i_2 \oplus i_3 \oplus i_4$

The truth tables are given in Figure 3, where the LBC check bit columns correspond to the TTs of the given functions. Looking at the TT, we can derive the following code properties:

1. The number of ones equals the number of zeros for each TT column.
2. The logical values on all pairs of bits are evenly distributed. This means that each pair of TT columns has the logical values (0,0), values (0,1), values (1,0) and values (1,1) just on 4 rows.
3. Let us consider quadruplets of rows arranged according to Figure 3 and pairs of all possible columns. Then the following cases may occur:

   Case a: All rows in the quadruplet column pair have constant values. It implies that four different quadruplets exist in the TT with different logical values on the pair of columns.

   Case b: The rows in the quadruplet column pair have two different values. It implies that there exist just two quadruplets with the same logical values.

   Case c: The rows in the quadruplet column pairs have all possible logical values. It implies that all four quadruplets have the same logical values on the pair of columns.

4. The logical values on all triplets of bits are evenly distributed. Each triplet of TT columns has logical values (0,0,0). (0,0,1), (0,1,0), (0,1,1), (1,0,0),(1,0,1), (1,1,0) and (1,1,1) just on 2 rows

Example:

Let us consider the quadruplets of rows in the first information bit TT and the first LBC check bit. The rows of the quadruplet column pairs reach the logical values (0,0), (1,0), (0,1), and (1,1). The same holds for the resting quadruplets of this column pair. It corresponds to case c of the 3rd property. Let us consider the quadruplets of the 1st and 3rd information bit. The rows of the 1st quadruplet column pairs reach the logical values (0,0), (1,0), (0,0), and (1,0). The rows of the 3rd quadruplet reach the same logical values. Other quadruplets reach other values. It corresponds to case b of the 3rd property. Let us consider the quadruplets of the 3rd and 4th information bit. The 1st quadruplet reaches only the bit pair values (0,0), the 2nd values (1,0), the 3rd values (0,1), and the 4th values (1,1). It corresponds to case a of the 3rd property.

The TT properties of the LBC mentioned above are directly derived from the Boolean expressions defining the check bits. Having the LBC code part, we can add the NBC check bit TTs that guarantee that the NBC TT column, together with every possible pair of LBC columns, reaches the state where all possible combinations of the triplet rows are contained within the NBC TT.

Code construction

Let us consider constructing an NBC with a balanced number of ones and zeros and with logical values on all pairs of bits evenly distributed. The assembled TT has to reflect LBC properties given in LBC property 3. It can be done by completing the check bit TT from a limited number of quadruplets of bits.

row quadruplet		inf. bits				LBC check bits				NBC check bits			
	1	0	0	0	0	0	0	0	0	0	1	0	1
1st	2	1	0	0	0	1	1	1	0	0	0	1	1
	3	0	1	0	0	1	1	0	1	1	1	0	0
	4	1	1	0	0	0	0	1	1	1	0	1	0
	5	0	0	1	0	1	0	1	1	1	1	1	1
2nd	6	1	0	1	0	0	1	0	1	0	1	1	0
	7	0	1	1	0	0	1	1	0	1	0	0	1
	8	1	1	1	0	1	0	0	0	0	0	0	0
	9	0	0	0	1	0	1	1	1	0	0	0	0
3rd	10	1	0	0	1	1	0	0	1	1	0	0	1
	11	0	1	0	1	1	0	1	0	0	1	1	0
	12	1	1	0	1	0	1	0	0	1	1	1	1
	13	0	0	1	1	1	1	0	0	1	0	1	0
4th	14	1	0	1	1	0	0	1	0	1	1	0	0
	15	0	1	1	1	0	0	0	1	0	0	1	1
	16	1	1	1	1	1	1	1	1	0	1	0	1
		LBC											
		NBC											

Figure 3. Resulting(12,4,3) NBC obtained as an extension of the LBC by the NBC check bit TTs. The first column shows TT row numbers and the considered quadruplets of them. The yellow part corresponds to the TTs of the information bits, the green and yellow region of the Table corresponds to the LBC, and the yellow, green, and blue area to the NBC.

To reach full coverage of triplets containing LBC column pairs described in case a, we have to guarantee that the NBC check bit TT has at least one row in each quadruplet equal to logical one and one row equal to logical zero. This condition is valid for TTs with at least two different quadruplets on the concerned positions. To reach full coverage of triplets containing column pairs of the LBC described in cases b and c, we have to guarantee that the check bit TT quadruplets are different at least once in those positions where the quadruplet pairs of bits of the LBC are equal. This condition is valid for

TTs where no quadruplet is used twice. Let us consider four different quadruplets of bits that have a balanced number of ones and zeros. These quadruplets will be placed on the corresponding check bit TT rows. An example of the quadruplets may be (0,1,0,1), (0,0,1,1), (1,0,1,0), and (1,1,0,0). Appropriate placing them into the NBC TT can guarantee to keep the number of specified bits

Example of NBC assembling

Let us consider the code building blocks (0,1,0,1), (0,0,1,1), (1,0,1,0) and(1,1,0,0). Let us choose the 1st NBC TT quadruplet equal to (0,0,1,1). Then the second quadruplet has to be either (0,1,0,1) or (1,0,1,0). Other quadruplets are not suitable as using them would cause a violation of property 2, and the number of logical values on code pairs of bits would be unbalanced. We choose (0,1,0,1). The next quadruplet should be (1.0.1.0) as other quadruplets violate restrictions caused by case b of the LBC. The last quadruplet should be (1,1,0,0) as we have to avoid repeating any quadruplet already used in the TT. The first quadruplet of the next NBC TT can be chosen arbitrarily. We choose (1,01,0). The next quadruplet has to be selected either (1,1,0,0,) or (0,0,1,1). We choose (1,1,0,0) Then the resting quadruplets are (0,0,1,1) and (0,1,0,1). The next two columns are set similarly. The resulting NBC is given in Figure. 3. No more check bits can be found as placing the quadruplets violates the conditions.

(136,5,3) code

The LBC with five information bits is regular: each pair of bits has each logical value just eight times within the TT, and each triplet has every logical value just on four rows. The LBC code length is equal to 16. The high number of rows with identical logical values on the LBC columns causes higher freedom in placing the quadruplets in the check bit TTs. The NBC check bit TTs can be generated using the identical quadruplets as in the previous case; each may be used repeatedly. They are applied to guarantee maximally uniform distribution of logical values within all pairs of truth tables. It is not practical to enumerate all placement possibilities. Instead, we may select several successful TT patterns that contain a unique ratio of the quadruplets and then verify the cases of their variations.

Code construction example

At first, we find ten check bit TT composed of two or four different considered quadruplets. We take into account only those TTs with each logical value present within their rows at least six times for each pair of the code bit. We exclude those TTs with simple quadruplet order variations of other TTs. In the second step, we permute the order of quadruplets in the successfully found TTs from the previous step. After performing all possible 40320 permutations of the first step TTs, we obtain a subset of new TTs for each of them. The number of NBC TTs is the resulting NBC length. Obtained code lengths within an experiment with 30 random settings of the ten first successful TTs are plotted in Figure 4. The maximum code length obtained by the proposed approach equals 136. Figure 4 compares resulting code lengths with an entirely random method where the code lengths were chosen as maximum code length values after 100 000 attempts.

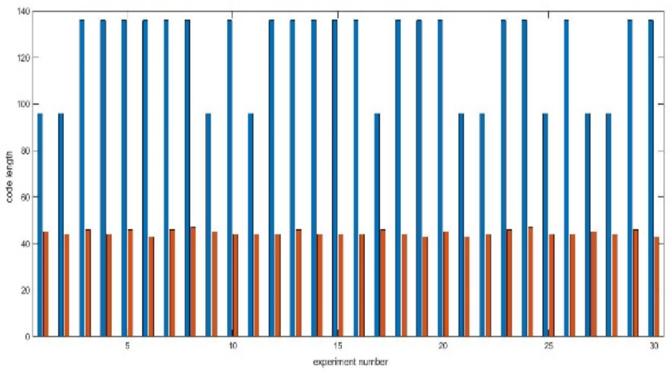


Figure 4: The obtained proposed method code lengths (blue columns) and the randomly obtained maximum code lengths (red columns).

Using the proposed constraints enables us to find code lengths that the random and or exhaustive search cannot provide. The search process is substantially more efficient than an exhaustive search in the space of all possible NBC check bit TTs. An example of the 136-bit NBC structure is given in Figure 5. The code word TTs corresponds with the columns of the figure. There are 16 bits of the LBC and 120 NBC check bits. Comparing the code lengths, we can claim that the NBC is more than eight times more effective in pattern decompression than the LBC. Compared with [18], we can see that the clique coverage strategy provides codes with less efficient parameters. A code with a length equal to 32 requires at least six information bits for three specified bits.

Code (536,6,3)

Considering six information bits of the LBC, we have 16 rows for each pair of the LBC columns where the logical values are identical. This fact provides such great freedom in choosing the NBC check bit TTs that it is more advantageous for the extension of the LBC to find the NBC check bits with the help of an entirely random search within a balanced number of ones and zeros TTs. The experiments we have done have the same results as those described in previous research [16]. The maximum code length obtained was equal to 536.

III. Conclusion

We have presented a nonlinear block code creation strategy based on detailed knowledge of the linear block code properties and completing bits that fit the linear part of the code to keep the number of specified bits. Based on understanding the most promising code bit structures, we developed a (136,5,3) code creation method that no exhaustive or random search strategy can find within a limited time. To reduce the search space of the added check-bit truth tables, we select only those that have rows guaranteeing a maximally uniform distribution of logical values within all pairs of truth tables. Simultaneously, all pairs of the original truth tables together with each new truth table exhaustively cover all logical values. The code is more than eight times more efficient in test pattern decompression than the linear code with the same number of information bits.

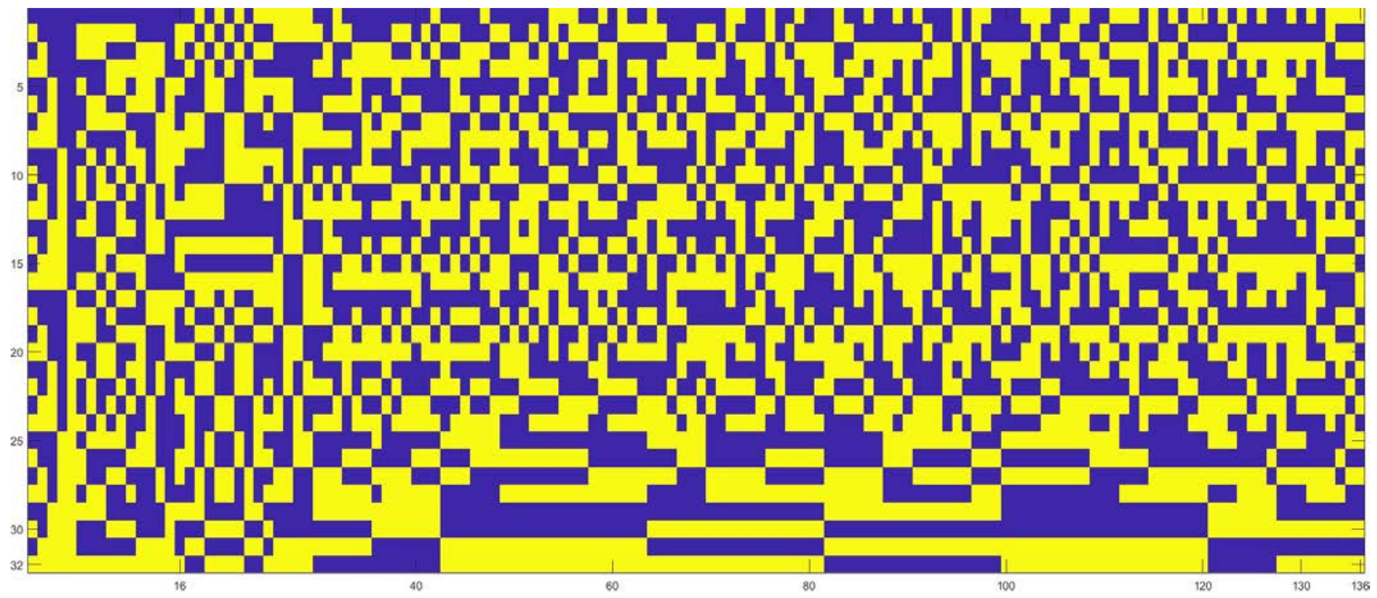


Figure 5:(136,5,3) NBC code structure. Each code bit TT corresponds with one column; logical values in the TT rows are zeros (blue color) and ones (yellow color).

IV. References

[1] S. Bahl et al., "Unifying scan compression," 2014iEEEinternational Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Amsterdam, 2014, pp. 191-196.

[2] Chalupa, D., Pospichal, J.. Analysis of iterated Greedy Heuristic for Vertex Clique Covering, Computing And informatics**,** Slovakia, 37, Jul. 2018

[3] Chandra, R. Kapur, and Y. Kanzawa, "Scalable Adaptive Scan (SAS)," Design, Automation & Test in Europe Conference & Exhibition, Nice, 2009, pp. 1476-1481.

[4] S. Chattopadhyay, "Efficient circuit-specific pseudoexhaustive testing with cellular automata," Test Symposium 2002. (ATS' 02). Proceedings of the 11th Asian, pp. 188-193, 2002, ISSN 1081-7735.

[5] Golan, P.: Pseudoexhaustive Test Pattern Generation for Structured Digital Circuits. Proc.IX International Conference on Fault-Tolerant Systems and Diagnostics FTSD9, Brno, Czechoslovakia, 1986, pp. 214-220

[6] Hamzaoglu and J.H. Patel, "Reducing Test Application Time for Full Scan Embedded Cores," Proc. Int'l Symp. Fault-Tolerant Computing, pp. 260-267, 1999.

[7] M. Keila, L. Stewart: "Approximating the minimum clique cover and other hard problems in subtree filament graphs," Discrete Appl. Math., 154 (14) (2006), pp. 1983-1995

[8] http://mint.sbg.ac.at/index.php, January 10, 2022

[9] Krishna, C.V., and NA. Touba, "Adjustable Width Linear Combinational Scan Vector Decompression," Proc. International Conf. Computer-Aided Design iCCAD 03), IEEE CS Press, pp. 863-866.

[10] Luce, R. Duncan; Perry, Albert D. "A method of matrix analysis of group structure," Psychometrika, **14** (2)95–116, doi:10.1007/BF02289146 (1949)

[11] K.J. Lee, J.J. Chen, and C.H. Huang, "Using a Singleinput to Support Multiple Scan Chains," Proc. Int'l Conf. Computer-Aided Design, pp. 74-78, 1998.

[12] S. Mitra, K.S. Kim, XPAND: An Efficient Test Stimulus Compression Technique, IEEE Trans. Computers 55, 2006, pp. 163-173

[13] S. S. Muthyala and N. A. Touba, "Improving test compression with scan feedforward techniques," 2014international Test Conference, Seattle, WA, 2014, pp. 1-10.

[14] O. Novák, "Extended binary nonlinear codes and their application in testing and compression," 2017 22ndiEEE European Test Symposium (ETS), Limassol, 2017, pp. 1-2.

[15] O. Novák, M. Rozkovec, J. Plíva, "Decompressors using nonlinear codes, "Microprocessors and Microsystems, Volume 76, 2020, 103076, ISSN 0141-9331

[16] O. Novák, "Search Strategy of Large Nonlinear Block Codes," *2021 24th Euromicro Conference on Digital System Design (DSD)*, 2021, pp. 527-534

[17] S. Samaranayake, E. Gizdarski, N. Sitchinava, F. Neuveux, R. Kapur, and T.W. Williams, "A Reconfigurable Shared Scan-In Architecture," Proc. IEEE VLSI Test Symp., pp. 9-14, 2003.

[18] Jan Schmidt and Petr Fišer: "Nonlinear codes for test patterns compression: the old school way," 14thinternational Workshop on Boolean Problems, Bremen, DE, 2020

[19] M.A. Shah and J.H. Patel, "Enhancement of the Illinois Scan Architecture for Use with Multiple Scaninputs," Proc. Int'l Symp. VLSI, pp. 167-172, 2004.

[20] R. Srinivasan, S. K. Gupta, and M. A. Breuer, "Novel test pattern generators for pseudoexhaustive testing," in IEEE Transactions on Computers, vol. 49, no. 11, pp. 1228-1240, Nov 2000.

[21] V. Tenentes, X. Kavousianos, and E. Kalligeros, "State Skip LFSRs: Bridging the Gap between Test Data Compression and Test Set Embedding foriP Cores," 2008 Design, Automation, and Test in Europe, Munich, 2008, pp. 474-479.