Area and Speed Oriented Implementations of Asynchronous Logic Operating Under Strong Constraints

Igor Lemberski
Baltic International Academy, Riga, Latvia
e-mail: Igor.Lemberski@bsa.edu.lv

Petr Fišer
Faculty of Information Technology
Czech Technical University in Prague, Czech Republic
e-mail: fiserp@fit.cvut.cz

Abstract— **Asynchronous circuit implementations operating under strong constraints (DIMS, Direct Logic, some of NCL gates, etc.) are attractive due to: 1) regularity; 2) combined implementation of the functional and completion detection logics, what simplifies the design process; 3) circuit output latency is based on the actual gate delays of the unbounded nature; 4) absence of additional synchronization chains (even of a local nature). However, the area and speed penalty is rather high. In contrast to the state-of-the-art approaches, where simple (NAND, NOR, etc.) 2-input gates are used, we propose a synthesis method based on complex nodes, i.e., nodes implementing any function of an arbitrary number of inputs. Synchronous synthesis procedures may be freely adopted for this purpose. Numerous experiments on standard benchmarks were performed and the efficiency of the proposed complex gate based method is clearly shown. DIMS and Direct Logic based asynchronous designs are considered in the paper.**

Keywords-asynchronous circuits, DIMS, direct logic, NCL

I. INTRODUCTION

The asynchronous logic is classified depending on the mode of interaction with the environment. In the *input-output mode*, the environment is allowed to change the input state once a new output state is produced. There is no assumption about internal signals and the environment is allowed to change the input state before the circuit is stabilized in response to the previous input state. In the *fundamental mode*, the logic operates based on the following discipline: the environment changes the input state once the output state has changed in response to the current input state and each gate inside the circuit is stable. The design methodology assumes either bounded or unbounded gate and wire delays.

In case of *bounded delays*, the moment when the environment may change the input state is estimated based on the worst-case propagation delay [1]. Within this model, only one input signal can be changed at a time. In [2], the generalized fundamental mode was proposed. Here multiple input changes are allowed during a narrow time interval. For such a mode, the method of hazard-free two-level implementation was proposed [3]. The multi-level (hazard not increasing) transformation is applied to optimize the implementation [1, 4]. Methods of hazard-free technology mapping were proposed in [5, 6].

In case of *unbounded delays*, the circuit should be able to recognize the moment when input and output states have changed. For this purpose, both inputs and outputs are implemented using a dual-rail encoding. A four-phase behavior discipline is supposed: to change an input state, the environment should reset it first (change to so called *space state*). As a result, the output state is reset too. After that the environment sets a new input state. It implies a new output state. The behavior rule is based on Seitz's strong or weak constraints [7, 8]. Under the strong constraints, each output changes its state only when all inputs have changed their state. Under the weak constraints, some outputs are permitted to change their state when some (not all) inputs have changed their state. In both cases, output signals also serve as the completion detection ones and indicate the moment when both internal and output signals become stable. The dual-rail implementation under the four-phase discipline is based on Delay-Insensitive Minterm Synthesis (DIMS) technique [9]. Within it, a function is implemented as a two-level structure with C-elements on the first level and an OR gate on the second one. A similar approach [10] is based on using threshold functions (Null Convention Logic-NCL). However, NCL circuits are designed based on 27 library gates that capable of implementing any function of four or less inputs [25]. Although in the single-rail logic most of gates operate under the weak constraints, in dual-rail logic some gates (for example, $ab + bc + ad$) can be implemented under the strong constraints (supposing c and d be complements of a and b, respectively). In case of a dual-rail logic, each literal is considered as a separate variable. Therefore, not *any* single-rail function of more than two inputs can be implemented in NCL; the feasibility of a function implementation depends on the number of literals in its dual-rail representation.

The DIMS cost is very high since the term minimization is not allowed. Therefore, 2^k minterms (where k is the number of C-element inputs) must be generated to implement each function's positive and negative forms and each minterm is implemented using a k-input C-element. Finally, a C-element is more complex than a simple gate. The implementation of the two-level C-OR logic as a single CMOS gate (Direct Logic) significantly reduces the area [11].

To reduce the area further, implementations were proposed based on a separation of the functional and completion detection logics [12, 13]. The drawback is that the signals from the node outputs should be brought up to a single point (a completion detection block), which may result in routing problems. In [12], the circuit is decomposed into a network of simple gates (NAND, NOR, etc.) and each such a gate is doubled to ensure monotonicity and proper

completion detection. Compared to the synchronous implementation, the circuit cost doubles.

In [14], a method was proposed where the initial function is decomposed into a single-rail network of two-level AND-OR nodes first. Then the network is transformed into a dual-rail one, to ensure monotonicity and hazard-free implementation. Although the approach slightly increases the complexity of the functional logic, the completion detection logic complexity reduces significantly, since the number of nodes that should be supplied with the completion detection signals is less than in [12]. As a result, considerable improvement in sense of the total complexity is obtained.

Desynchronization [15] is a modern paradigm that is based on adopting synchronicity to the asynchronous logic design. If bounded delays are supposed, matched delays are introduced for the synchronization purpose. In the case of unbounded delays, separate completion detection logic should be present to indicate the circuit stability [12]. As mentioned, the latter doubles the circuit complexity. A network of local controllers is designed to provide proper local synchronization signals. It results in an additional area penalty. Finally, the circuit is equipped with output latches and a new output state is available only once a latch signal enables.

The asynchronous logic operating under the strong constraints benefits from the regularity and combined implementation of the functional and completion detection logics, which simplifies the synthesis process. The circuit output latency is based on actual gate delays of the unbounded nature. Furthermore, additional synchronization chains (even of a local nature) are not required.

The regularity and simplicity make the approach attractive for fast prototyping into conventional reconfigurable logic. It is based on the developing Muller gate library [16]. However, the area and speed penalty is rather high. Although DIMS and Direct Logic implement *any function* of a given number of inputs and NCL library gates implement any function of four or less inputs, the state-of–the-art methods are based on the initial representation as a Boolean network of the *simple gate* logic (NAND, NOR, etc.) and further implementation of each gate as two-level (AND-OR) dual-rail logic. It results in not only high complexity of the circuit implementation (in sense of the area), but also in a low speed, since each simple gate (single level structure) is implemented as a two-level (C-OR) structure.

In the method we propose in this paper, the function is decomposed into a network of general nature nodes using a tool intended for the synchronous logic synthesis (ABC [17]). Then each node is implemented as DIMS or in Direct Logic [11]. Experiments show clearly, that our method results in reducing both the implementation area and delay.

The paper is organized as follows: after some preliminaries, different dual-rail asynchronous logic implementations are described in Section III. The motivation for our investigation is presented in Section IV. Section V contains the experimental results justifying the efficiency of the proposed method. These results are discussed in Section VI. Section VII concludes the paper.

II. PRELIMINARIES

A. Multi-Level Single- and Dual-Rail Representations

Let $F = \{f_1, f_2, ..., f_q\}$ be an asynchronous multi-output function of n inputs X: $X = \{x_1, x_2, ..., x_n\}$ and q outputs. Let $Y = \{y_1, y_2, ..., y_m\}$, $f_1, f_2, ..., f_q \in Y$, $m \geq q$, be a set of single-output Boolean nodes obtained as a result of a synthesis and technology mapping. Each node function y_c depends on given k or less inputs: $y_c = y_c(z_1, z_2, ..., z_k)$, $z_1, z_2, ..., z_k \in \{X \cup Y\}$ and it is represented as a two-level (AND-OR) logic (Fig. 1). We call it a single-rail multi-level representation.

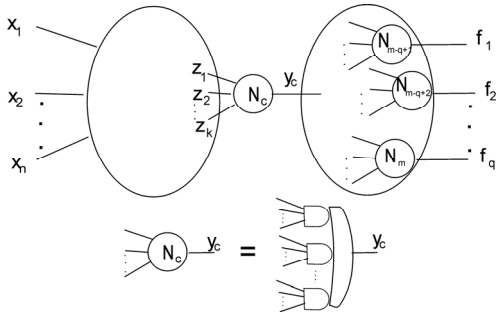

Figure 1. Single–rail multi-level Boolean network of complex nodes

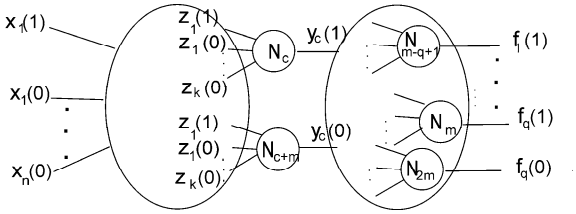

Figure 2. Dual–rail multi-level Boolean network of complex nodes

Generally, an asynchronous logic should be capable:

1) to recognize the moment when a new input state (generated by the environment) appears on the inputs and the moment when the circuit generates a new output state in response to the input one;

2) to notify the environment on new input and output states. After receiving the notification, the environment can generate the next input state. To solve this problem, inputs/outputs are implemented using a dual-rail representation.

In dual-rail logic (Fig. 2), it is supposed for every node y_c, that each of its inputs z_i, $i = 1, 2, ..., k$, may be in one of these three states: states 1, 0 (so called working states) or undefined (space state). To implement a three-state input z_i, two signals $z_i^{(1)}$ and $z_i^{(0)}$ are introduced, where $z_i^{(1)} = 1$ and $z_i^{(0)} = 0$, if z_i is in state 1, $z_i^{(1)} = 0$ and $z_i^{(0)} = 1$ if z_i is in state 0, $z_i^{(1)} = z_i^{(0)} = 0$ if z_i is in the space state. The combination $z_i^{(1)} = z_i^{(0)} = 1$ is not allowed. Similarly, to implement a three-state node function, the function y_c, $c = 1, 2, ..., m$, should be represented in both positive $y_c^{(1)}$ and negative $y_c^{(0)}$ forms. If $y_c^{(1)} = 1$, $y_c^{(0)} = 0$, then the function y_c is in state 1, if $y_c^{(1)} = 0$, $y_c^{(0)} = 1$, then the function y_c is in state 0,

if $y_c^{(1)} = y_c^{(0)} = 0$, then the function y_c is in the space state. The combination $y_c^{(1)} = y_c^{(0)} = 1$ is not allowed. As a result, each dual–rail function $y_c^{(1)}$, $y_c^{(0)}$ depends on $2k$ or less inputs. To change the input state, the environment should reset it first to the space state and after that set it to the proper working state. In the reset phase, the output state changes from the working state to the space one and in the set phase the new output state is recognized.

Each node positive and negative function can be represented in a Sum-Of-Products (SOP) form. If each term contains exactly k literals, we call such a representation as a Sum-Of-Minterms (SOM).

Example: Suppose a 2-input XOR node function, where SOP and SOM representations concise:

$y = z_1z_2' + z_1'z_2$, where z_1', z_2' denote complements of z_1, z_2 respectively. In dual-rail logic, the single-rail node is split into two nodes with the functions as follows:

$$y^{(1)} = z_1^{(1)} z_2^{(0)} + z_1^{(0)} z_2^{(1)}$$
$$y^{(0)} = z_1^{(1)} z_2^{(1)} + z_1^{(0)} z_2^{(0)}$$

Note, that the single-rail node function depends on two variables, however the function in dual-rail depends on four variables.

B. Strong and Weak Constraints

In the fundamental mode, the asynchronous logic operates under so called *strong* or *weak constraints* [8]. Timing diagrams depicting behavior rules under strong and weak constraints are presented in Fig. 3.

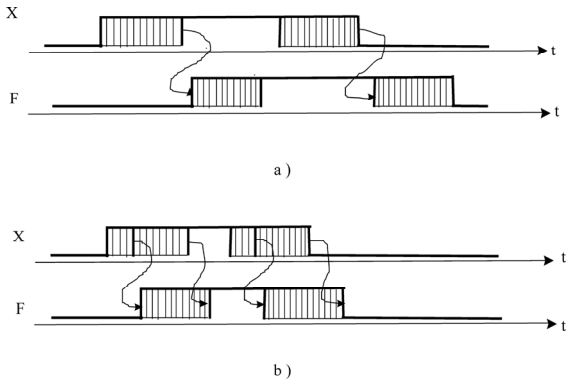

Figure 3. Asynchronous logic behavior under strong (a) and weak (b) constraints

The diagrams depict the input X and output F state changes in time t along with input/output states dependency. Each input and output may be either in a working (high level) or a space (low level) state. Vertical shading depicts time intervals, where input and output states are going from the space to a working state and vice versa.

Under the strong constraints (Fig. 3a), the behavior rule is as follows:

1. If all inputs are in the space state, then all outputs are going to the space state;
2. If all inputs are in a working state, then all outputs are going to a working state.

Therefore, under the strong constraints, the function block outputs are changed once all inputs are in a working or space state. Within the two-level representation, the term minimization is not allowed. Under the weak constraints (Fig. 3b), some outputs are permitted to change their states when some (not all) inputs have changed their states:

1. If some inputs are in a working state, then some outputs are going to a working state;
2. If all inputs are in a working state, then all outputs are going to a working state;
3. If some inputs are in the space state, then some outputs are going to the space state;
4. If all inputs are in the space state, then all outputs are going to the space state.

As a result, some outputs may not change their state until all inputs change their state. The method of reducing the implementation complexity based on the literal minimization was proposed in [18].

The method proposed in this paper is based on the logic behavior under the strong constraints.

III. IMPLEMENTATIONS

A. DIMS Logic

A two-level DIMS implementation of asynchronous logic, where both positive and negative functions should be represented as SOMs, was proposed in [9, 19]. 2^k minterms in total must be generated to represent each function in its positive and negative form. On the first level, each minterm is implemented using a C-element. For each input $z_i \in \{X \cup Y\}$, either the signal $z_i^{(1)}$ or $z_i^{(0)}$ is connected to an input of each C-element. The second level is implemented using OR gates. The C-element is a strongly indicating logic: its output signal switches on/off once all its input signals switch on/off. Therefore, the C-element output indicates whether the circuit inputs are in a working or space state. Since minterms do not intersect, only one C-element switches on/off at a time. The C-element output signal propagates through a single path to the OR gate (circuit) output. As a result, the circuit output notifies on the new input state correctly. One can easily check that DIMS operates under Seitz's strong constraints.

Example: Given the 2-input XOR function (see Section II.A). Its DIMS implementation (as two nodes) is depicted in Fig. 4. Suppose: $z_1^{(1)} = z_1^{(0)} = z_2^{(1)} = z_2^{(0)} = 0$ ("all inputs are in the space state"). One can check that in this case: $y^{(0)} = y^{(1)} = 0$ ("all outputs are going to the space state"). Now suppose that the environment switches the inputs z_1, z_2 to a working state, say 10: $z_1^{(1)} = z_2^{(0)} = 1$, $z_1^{(0)} = z_2^{(1)} = 0$. As a result, the C1-element output signal switches on and after propagating through the path (Fig. 4 - bold) the circuit output switches to the working state 1: $y^{(1)} = 1$, $y^{(0)} = 0$ ("if all inputs are in a working state, then all outputs are going to a working state"). When all inputs switch from the working

state 10 to the space state, then the C1-element output switches off and therefore the output $y^{(1)}$ switches off ("if all inputs are in the space state, then all outputs are going to the space state"). Note, that under the strong constraints, the output state is capable of indicating *all* new input states (i.e., no special indication logic is required), because *all* new input states imply the output state. Different C–element transistor level implementations (both static and dynamic) can be found in [20].

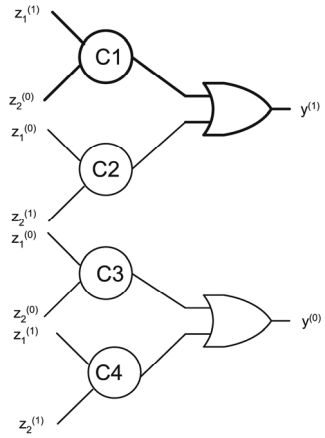

Figure 4. DIMS implementation of XOR function

The complexities (in terms of the number of transistors) and delays of single–output DIMS implementations of functions of different number of inputs can be found in Table I. The implementations are based on the dynamic C-element. A delay unit is the delay of the conventional inverter [11].

TABLE I. DIMS : NUMBER OF TRANSISTORS AND DELAY

Number of inputs	Number of transistors	Delay
2	24	8.2
3	64	15.1
4	160	21.3
5	384	--
6	896	--

B. *Direct Logic*

Within this approach, C-elements and output OR gates are merged and implemented as a single CMOS gate containing p- and n- transistors. The implementation of a XOR function is shown in Fig. 5 [11]. The p-transistors establish a path between the power supply and the outputs when all inputs are in the space state. It requires $2k$ p-transistors (where k is the number of input variables in the dual-rail logic). The n-transistors are arranged as a tree structure. It reduces the number of n-transistors required.

The complexity of the direct logic that is implemented based on the structured n-transistor tree is shown Table II [11]. One can see that the direct logic complexity is significantly less than the two-level DIMS one. In contrast to standard CMOS gates, in the direct logic gate there are different delays for each input. The respective maximum gate delays were copied from [11] and are shown in Table II, too. These are slightly less, than in the case of two-level DIMS.

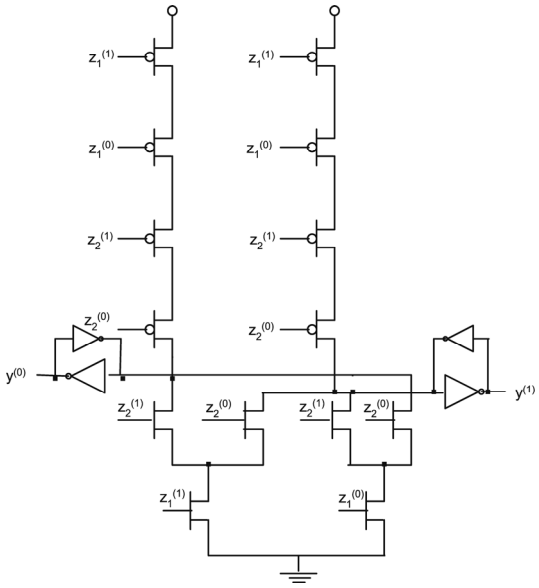

Figure 5. Direct Logic implementation of XOR function

TABLE II. DIRECT LOGIC : NUMBER OF TRANSISTORS AND DELAY

Number of inputs	Number of transistors	Delay
2	22	8.2
3	34	12.3
4	54	19.4
5	90	--
6	158	--

The number of stack transistors increases with the growing number of inputs. As a result, the logic operates slower. The problem may be avoided by the decomposition into multi-output nodes, where inputs indication is distributed between different outputs. Such a logic operates under the weak constraints. The implementation based on the multi-output nodes will be considered in a future.

IV. MOTIVATION

Suppose that the circuit (Fig. 6a) is represented as a network of 2-input NAND gates. Following the conventional methods, each NAND gate should be implemented using DIMS or direct logic. The complexity of 2-input NAND implemented in direct logic is 22 transistors (see Table II). Therefore, the circuit will be implemented using: 22*4 = 88 transistors. However, a group of NAND gates can be extracted and transformed into a three-input complex (AND-OR) node (Fig. 6b). Such a node can be implemented as one three-input gate. Its complexity is 34 transistors (see Table II). As a result, the circuit (Fig. 6b) total complexity is 56 transistors. Based on this observation, we propose a method that comprises of the following steps:

1) a Boolean network of complex nodes (instead of simple gates) with a given number of inputs is synthesized;
2) dual-rail network is constructed;
3) each complex node is implemented as DIMS or in direct logic.

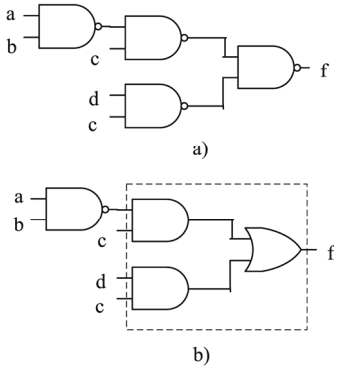

Figure 6. The circuit representation as simple gate logic (a). Transformation into two-level complex nodes (b)

Standard tools for synchronous synthesis may be used for the first step, since no limitations involved by the final asynchronous implementation are imposed here.

V. Experimental Results

A. Experimental Background

In order to justify our claims and to show the advantages of the proposed method, we have conducted numerous experiments. We have processed benchmarks from the MCNC [21], ISCAS'85 [22] and ISCAS'89 [23] sets, 228 circuits altogether. The synthesis was performed by ABC [17]. The benchmarks were synthesized by using an extended *choice* script (Fig. 7), followed by technology mapping. The technology mapping was performed both by the ABC *fpga* command mapping the circuit into look-up tables (LUTs) of a given size, and by the ABC *map* command. In the latter case, the default ABC standard cell technology library (MCNC) is used, as well as a mapping into NAND gates with a given number of inputs (*k*-NAND) is performed. The synthesis process was iterated 20-times, to reach better quality for an affordable cost of runtime.

As a result of the synthesis, a Boolean network described in Blif [24], i.e., a network of AND-OR nodes, is obtained.

The LUT mapping serves as an optimum way to perform the synthesis into complex gates, since the synthesis process treats LUTs as arbitrary *k*-input gates.

The number of NAND gates inputs, as well as complex node inputs, varied from 2 to 6. The limit of maximum 6 NAND inputs was chosen because more-input gates are being used in physical designs rarely. The limit of 6 inputs for complex nodes was determined by synthesis limitations of ABC.

The NAND implementation was chosen because of its simplicity and, more importantly, generality. The circuit is synthesized by ABC as a network of NAND gates and inverters. In the dual-rail logic, inverters are not needed, since complemented signals are provided directly. As a result, in dual-rail logic the NAND mapping fully represents and substitutes any simple gate (AND, OR, NAND, NOR) mapping. Mapping into more complex gates, like XORs, or-and-invert (OAI), and-or-invert (AOI), etc., is represented by the standard cell mapping.

```
fraig_store;
resyn; fraig_store;
resyn2; fraig_store;
resyn2rs; fraig_store;
share; fraig_store;
fraig_restore
```

Figure 7. Extended *choice* script

TABLE V. Summary Comparison Results for Different Implementations using the Direct logic

Method	*Number of transistors*	*Best in*
SC	2,492,238	1.8%
2-NAND	2,543,068	2.2%
3-NAND	2,561,846	3.9%
4-NAND	2,603,488	0.9%
5-NAND	2,738,028	0.9%
6-NAND	2,735,728	0.9%
2-CG	2,323,046	0.0%
3-CG	**2,009,328**	**50.4%**
4-CG	2,103,430	22.8%
5-CG	2,472,534	9.6%
6-CG	3,109,692	9.6%
NCL	2,233,104	8.3%

B. Area Comparison of Implementations

In this Section, we present a comparison of DIMS-based implementations obtained in different ways. Particularly, the influence of the node inputs limit on the resulting area will be discussed.

First, the *k*-input complex nodes (*k*-CN) DIMS implementation (Section III.A) is compared with the conventional one, where each simple gate is implemented as a two-level DIMS [10, 13].

The comparison results are shown in Table III. The benchmarks were synthesized, as described in the previous subsection, using the standard cell ABC technology library ("*SC*" row), *k*-NAND gates ("*k-NAND*" columns), and *k*-input complex nodes ("*k*-C*N*" rows). One can see that the 2-input complex nodes DIMS implementation is the most area-efficient one here.

Second, *k*-input nodes are implemented using *k*-input direct logic complex gates (*k*-CG) (Section III.B) and the result is compared with the conventional and NCL implementations [25] (Table IV). The NCL logic was synthesized by the same process as 2-CG, since the NCL library is restricted to 4-input gates at most. Thus, any 2-input gate in dual-rail logic can be implemented in NCL.

Total numbers of transistors for given implementations are shown in the respective Table IV rows/columns, together with their percentage comparisons, in terms of area reduction. We can see that the 3-CG implementation is the most efficient.

The complexity measurements are summarized in Table V, in a different way. Here the total sums of the numbers of transistors are shown. The percentage of benchmarks, for which the respective method gave the best result, is shown in the column "*Best in*". Note that the summary percentage value exceeds 100%. This is because two or more methods produced equal results in some cases.

C. Delay Considerations

In this section we study the delays of the circuits implemented based on complex nodes and NCL. In case of two-level DIMS, each node was assigned its respective delay (see Table I.) and then the circuit was topologically traversed from its inputs to outputs, to calculate its real maximum delay. The computed numbers of the circuit levels and maximum circuit delays, together with the percentage delay comparison, are shown in Table VI.

Since a direct logic gate delay depends on the number of its inputs (see Subsection III.B), the number of the circuit levels (its critical path length) is not a credible delay estimation, since there may appear gates having less inputs than is the given *k* threshold (*k*-NAND, *k*-CG). Therefore we had to analyze the circuits more thoroughly to calculate the delay upper bound. The delay estimation results are shown in Table VII.

D. Detailed Experimental Results

A detailed experimental evaluation of the direct logic implementations, for a set of the "biggest" 20 of the 228 benchmarks, is shown in Table VIII. The 3-CG direct logic implementation is compared with the state-of-the-art 2-NAND direct logic, DIMS 2-CN and NCL implementations. Numbers of transistors and maximum delays are shown for all these implementations. The delays are given in terms of units of the conventional inverter delay. The "Total" row shows total sums of all the 228 benchmarks.

VI. Discussion

As for the DIMS implementation, experiments have shown that the 2-input complex nodes implementation (Table III) is the most area-efficient one. The DIMS nodes complexity grows exponentially with the number of their inputs (see Table I.). Therefore, even though implementations using more-input gates require less nodes, their complexity (in terms of the number of transistors) is too high. The only difference between the 2-NAND and 2-CN implementations is that XOR gates are allowed in 2-CN. Since the implementation costs of all 2-input nodes in DIMS are equal, the 2-CN implementation naturally must have equal or less complexity than the 2-NAND one for any circuit. Therefore, the results obtained could have been expected.

Conversely, in the direct logic, the 3-CG implementation has been found the most area-efficient one. This can be explained by the construction of the direct logic gates; due to the tree-shaped n-transistor structures (see Fig. 5), the complexity of the gates does not grow exponentially with the number of inputs. The results shown in Table V verify, that the best way of an asynchronous implementation of most of the circuits is the mapping into 3-input, possibly 4-input complex gates. The conventional NAND-gate mapping has been found to be surprisingly inefficient.

The NCL implementation requires less transistors than the 2-CG direct logic one, however, 3- and 4-CG implementations require less transistors than NCL. Partially, the latter is because in the direct logic, two complex gates (for both rails) are implemented as a whole and the n-transistor tree structure is designed to reduce the number of n-transistors required for the implementation. On the other hand, the NCL complexity can be reduced if the dual-rail logic literal (instead of single-rail input) decomposition procedure is applied. Such an approach was sketched out in [25] and its effectiveness was demonstrated on a few simple examples. The approach should be formalized and automated.

The delay of DIMS implemented circuits increases with an increasing number of node inputs (see Table VI.). Even though the number of levels of the 3- and 4-input gate implementations have less levels, this delay benefit is surpassed by a rapid increase of the DIMS gates delays, when increasing the number of their inputs. Therefore, the 2-CN implementation becomes the most delay-efficient one as well.

In the direct logic case, there is an apparent delay minimum for the 3-CG implementation (see Table VII). This is because the 4- and more-input complex gate delays surpass the advantage of having less circuit levels. As a result, the 3-CG implementation becomes the most efficient implementation, both in the area and delay.

In an overall comparison of the three studied implementations (DIMS, direct logic, NCL), the 3-CG implementation using direct logic becomes the most efficient one, both in the area and delay.

VII. Conclusion

We have proposed several asynchronous circuits synthesis processes and evaluated the quality of their results on standard benchmark circuits. DIMS, Direct logic and NCL implementations were considered.

All the studied processes are based on an initial decomposition of original functions into a Boolean network of NAND gates or complex nodes (nodes implementing an arbitrary function) of a given number of inputs and further implementation of each such node as a two-level DIMS or a single CMOS direct logic gate.

We have shown experimentally, that the most efficient (both in area and speed) DIMS implementation, is by using 2-input complex gates. This is, however, no big surprise.

As for the direct logic implementation, 3-input complex gates implementation has been found as the most efficient way of synthesis (again, both for area and delay). This observation is the main contribution of the paper; up to the

knowledge of the authors, the state-of-the-art here is the implementation using standard 2-input gates, as it is in the DIMS-based logic. However, implementation using 3-input complex nodes reduces both the numbers of the synthesized circuit nodes and levels. In contrast to this, increasing the number of gate inputs involves a higher gate area and delay. Judging these two contradictory aspects, we have found a local minimum in the 3-input node implementations.

In comparison with the 2-input direct logic implementation, the 3-input complex gate implementation reaches 21% reduction in both area and 19% delay in average. When compared to the Null Convention Logic (NCL), the 3-input complex gate direct logic implementation reaches 10% area and 19% delay reduction.

ACKNOWLEDGMENT

For the second author, this research has been supported by MSMT under research program MSM6840770014 and by the grant of the Czech Grant Agency GA102/09/1668.

REFERENCES

[1] S.H. Unger, "Asynchronous Sequential Switching Circuits", John Wiley & Sons, Inc., 1969

[2] S.M.Nowick, "Automatic Synthesis of Burst-Mode Asynchronous Controllers", Ph.D. thesis, Stanfort University, Mar March 1993

[3] S.M.Nowick, D.L.Dill, "Exact Two-Level Minimization of Hazard-Free Logic with Multiple-Input Changes", IEEE CAD, vol.14, August, 1995, pp. 986-997

[4] D. Kung, "Hazard-Non-Increasing Gate – Level Optimization Algorithm", IEEE Int. Conf. On Computer–Aided Design, Nov., 1992, pp. 631-634

[5] P. Beerel, K.Y.Yun, W.C. Chou, "Opimizing Average-Case Delay in Technology Mapping of Burst-Mode Circuits", IEEE Ont. Symp.on Advanced Research in Asynchronous Circuits and Systems, March, 1996 , pp.244-259

[6] P. Siegel, G.D. Micheli, D. Dill, "Automatic Technology Mapping for Generalized Fundamental Mode Asynchronous Designs", IEEE Design Automation Conference, June 1993, 61-67

[7] W.J.Dally, J.W.Poulton, "Digital Systems Engineering", Cambridge University Press, 1998, 663 p

[8] C.L Seitz, "System Timing", In: "Introduction to VLSI Systems", C. Mead, L. Conway, Addison—Welsey Publishing Company, 1980, pp. 218-262

[9] E.J. Sparsø, J. Staunstrup, and M. Dantzer-Sørensen, "Design of delay insensitive circuits using multi-ring structures", In Proc. of the Conference on European Design Automation, 1992, pp. 15-20.

[10] M.Ligthart, K.Fant, R.Smith, A. Taubin, A.Kondratyev, "Asynchronous Design Using Commercial HDL Synthesis Tools", 6-th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, pp. 114-125

[11] C.D.Nielsen, "Evaluation of Function Block Designs", Technical Report 1994-135, Department of Computer Science, Technical University of Denmark, Denmark, 1994, 43 pp.

[12] J.Cortadella, A.Kondratyev, L.Lavagno, C.Sotiriou, "Coping with the Variability of Combinational Logic Delays", IEEE Int. Conf. On Computer Design (ICCD), October 2004, pp.505-508

[13] A. Kondratyev, K.Lwin, "Design of Asynchronous Circuits Using Synchronous CAD Tools", IEEE Design and Test Computers, Vol. 19, No. 4, July-August ,2002, pp.107-117.

[14] I.Lemberski, P.Fišer, Multi-Level Implementation of Asynchronous Logic Using Two-Level Nodes, Proc. 4th Descrete-Event System Design (DESDes'09), Gandia Beach, Valencia (Spain), 6.-8.10.2009, pp. 213-218.

[15] J. Cortadella, A.Kondratyev, L.Lavagno, C.P. Sotiriou, "Desynchronization: Synthesis of Asynchronous Circuits from Synchronous Specifications", IEEE Trans. CAD, vol.25, No.10,October 2006, pp. 1904-1921

[16] Q.T.Ho, J.-B.Rigaud, L.Fesquet, M.Renaudin, R. Rolland, "Implementing Asynchronous Circuits on LUT Based FPGAs", Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL2002), Montpellier , France, 2002, pp. 36 - 46

[17] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", [online], http://www.eecs.berkeley.edu/~alanmi/abc/

[18] W.B. Toms. D.A. Edwards, Prime Indicants, "A Synthesis Method for Indicating Combinational Logic Blocks", 15th IEEE Symposium on Asynchronous Circuits and Systems, 2009, pp.139-150

[19] T.S. Anantharaman, "A Delay Insensitive Expression Recognizer," IEE VLSI Tech.Bull, Sept, 1986

[20] M.Shams, Jo.C.Ebergen, M.I.Elmasry, "Modeling and Comparing CMOS Implementations of the C-Element", IEEE Trans. VLSI Systems, vol.6,No.4, 1998,pp.563-567

[21] S. Yang, "Synthesis on Optimization Benchmarks User guide", Microelectronic Center, 1991.

[22] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan", Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985

[23] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989

[24] "Berkeley Logic Interchange Format", University of California, Brekeley, Tech. report, 2005

[25] Scott C. Smith, Jia Di, "Designing Asynchronous Circuits using NULL Convention Logic (NCL)", Morgan & Claypool, 96 p., 2009

TABLE III. AREA IMPROVEMENT: TWO-LEVEL DIMS

		2-NAND	*3-NAND*	*4-NAND*	*5-NAND*	*6-NAND*
	Number of transistors	2,774,256	3,659,976	4,546,384	5,563,944	5,550,736
SC	4,578,216	-39.40%	-20.06%	-0.70%	17.72%	17.52%
2-CN	**2,534,232**	8.65%	30.76%	44.26%	54.45%	54.34%
3-CN	3,489,120	-20.49%	4.67%	23.26%	37.29%	37.14%
4-CN	5,483,896	-49.41%	-33.26%	-17.10%	1.44%	1.20%
5-CN	9,074,128	-69.43%	-59.67%	-49.90%	-38.68%	-38.83%
6-CN	15,301,752	-81.87%	-76.08%	-70.29%	-63.64%	-63.72%

TABLE IV. AREA IMPROVEMENT: DIRECT LOGIC

		2-NAND	*3-NAND*	*4-NAND*	*5-NAND*	*6-NAND*	*NCL*
	Number of transistors	2,543,068	2,561,846	2,603,488	2,738,028	2,735,728	2,233,104
SC	2,492,238	2.00%	2.72%	4.27%	8.98%	8.90%	-10.40%
2-CG	2,323,046	8.65%	9.32%	10.77%	15.16%	15.08%	-3.87%
3-CG	**2,009,328**	20.99%	21.57%	22.82%	26.61%	26.55%	10.02%
4-CG	2,103,430	17.29%	17.89%	19.21%	23.18%	23.11%	5.81%
5-CG	2,472,534	2.77%	3.49%	5.03%	9.70%	9.62%	-9.68%
6-CG	3,109,692	-22.28%	-21.38%	-19.44%	-13.57%	-13.67%	-28.19%
NCL	2,233,104	12.19%	12.83%	14.23%	18.44%	18.37%	0.00%

TABLE VI. DELAY IMPROVEMENT: TWO LEVEL DIMS

		2- NAND	*3-NAND*	*4-NAND*
	Number of levels / Delay	2382 / 20,155.6	1877 / 21,865.3	1715 / 23,323.0
SC	1507 / 20,371.3	-1.06%	6.83%	12.66%
2-CN	2248 / **18,433.6**	8.54%	15.69%	20.96%
3-CN	1308 / 19,398.9	3.75%	11.28%	16.83%
4-CN	970 / 20,128.4	0.13%	7.94%	13.70%

TABLE VII. DELAY IMPROVEMENT: DIRECT LOGIC

		2- NAND	*3-NAND*	*4-NAND*	*NCL*
	Number of levels / Delay	2382 / 20,155.6	1877 / 19,007.6	1715 / 21,033.5	2248 / 19,505.2
SC	1507 / 18,005.7	10.67%	5.27%	14.40%	7.69%
2-CG	2248 / 18,433.6	8.54%	3.02%	12.36%	5.49%
3-CG	1308 / **15,879.3**	21.22%	16.46%	24.50%	18.59%
4-CG	970 / 18,238.5	9.51%	4.05%	13.29%	6.49%
NCL	2248 / 19,505.2	3.23%	-2.55%	7.27%	0.00%

TABLE VIII. DETAILED COMPARISON OF IMPLEMENTATIONS

	Direct Logic 3-CG		*Direct Logic 2-NAND*		*DIMS 2-CN*		*NCL*	
	Transistors	*Delay*	*Transistors*	*Delay*	*Transistors*	*Delay*	*Transistors*	*Delay*
alu4	38436	110.7	42350	123.0	49632	106.6	43434	113.8
apex2	56522	110.7	41096	131.2	50184	114.8	43911	122.8
apex3	29632	86.1	25256	98.4	28080	98.4	24573	104.8
apex4	43044	86.1	38588	106.6	42024	98.4	36771	104.8
c6288	22334	360.8	63910	705.2	35640	541.2	32496	581.8
des	52872	106.6	65494	131.2	6600	73.8	58206	115.8
i10	29478	49.2	35882	57.4	33384	237.8	29553	257.8
mainpla	52176	110.7	60016	164	63384	155.8	55503	167.8
misex3	51518	98.4	49060	98.4	63000	98.4	55134	104.8
prom1	103908	98.4	120648	123.0	126624	98.4	110979	107.8
prom2	45234	86.1	52184	123.0	55776	98.4	48936	107.8
s13207.1	31952	123.0	45034	172.2	42120	147.6	37368	155.8
s15850.1	41308	184.5	58696	246.0	56112	205	49638	217.8
s35932	97540	45.1	181236	73.8	116400	49.2	105915	51.8
s38417	118058	143.5	175802	188.6	172896	155.8	152652	166.8
s38584.1	148140	123.0	215160	180.4	218088	172.2	191820	185.8
s9234.1	20496	131.2	29062	196.8	27288	180.4	24192	193.8
seq	33274	98.4	36102	106.6	41088	98.4	35970	104.8
too_large	57426	123.0	61424	114.8	70104	114.8	61341	122.8
xparc	37656	123.0	44330	180.4	46320	155.8	40569	167.8
Sum:	**2,009,328**	**15,879.3**	2,543,068	20,155.6	2,534,232	18,433.6	2,233,104	19,505.2