Asynchronous Two-Level Logic of Reduced Cost

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

Petr Fišer
Dept. of Computer Science & Engineering
Czech Technical University in Prague
Prague, Czech Republic
e-mail: fiserp@fel.cvut.cz

Abstract— **We propose a novel synthesis method of a dual-rail asynchronous two-level logic of reduced cost. It is based on a model that operates under so called modified weak constraints. The logic is implemented as a minimized AND-OR structure, together with the completion detection logic. We formulated and proved the product term minimization constraint that ensures a correct logic behavior. We processed the MCNC benchmarks and generated asynchronous two-level logic. The implementation complexity was compared with the state-of-the-art approach. Using our approach, we achieved a significant improvement.**

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 the new output state is produced. There is no assumption about the 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 the response on 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 the time. In [2], the generalized fundamental mode was proposed where 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]. The methods of hazard-free technology mapping were proposed in [5, 6].

In case of *unbounded delays*, the circuit should be capable 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. To change an input state the environment should reset it first (change to so called space state). The output state resets too, as a result. 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, all outputs change their state when all inputs have changed their state. For the multi-level logic, the hazard-free design methodology of circuits implemented using simple gates and so called Null Convention Logic was proposed [9, 10]. The transformation includes converting each internal node into two nodes to produce the outputs for both positive and negative form. The drawback is that the circuit cost doubles.

The *two-level* logic (represented as a sum of product terms) is attractive due to its high performance (a signal from inputs to outputs propagates through two levels only), regularity, and a good starting point for a multi-level logic synthesis [11]. Moreover, two-level PLA structures are now gaining a newborn interest for their regularity, allowing their implementation using nano-scale sublithographic techniques [12, 13]. In [14, 15], the two-level (Delay-Insensitive Minterm System - DIMS) multi-output function logic was proposed, where the C-element logic [16] is used for the first level to implement minterms (terms of n literals, where n is the number of inputs) and OR gates for the second level. Each function is implemented in both positive and negative form. The behavior is based on Seitz's strong constraints. The DIMS cost is very high since the term minimization is not allowed. Therefore, 2^n minterms must be generated to implement each function's positive and negative forms and each minterm is implemented using an n-input C-element, which is complex itself. In this paper, we propose a method of reducing the two-level logic cost. It is done in two ways: 1) terms minimization is allowed; 2) the C-element logic is replaced by AND functions. The method is tested on MCNC [17] benchmarks and compared with DIMS [14, 15].

II. Preliminaries

A. Input/Output Dual-Rail Encoding

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

Let $F = \{f_1, f_2, \ldots, f_q\}$ be an asynchronous multi-output function of n inputs X: $X = \{x_1, x_2, \ldots, x_n\}$ and q outputs.

It is supposed that each input and output 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 x_i, $i = 1, 2, \ldots, n$, two signals $x_i^{(1)}$ and $x_i^{(0)}$ are introduced, where $x_i^{(1)} = 1$ and $x_i^{(0)} = 0$, if x_i is in state 1, $x_i^{(1)} = 0$ and $x_i^{(0)} = 1$ if x_i is in state 0, $x_i^{(1)} = x_i^{(0)} = 0$ if x_i is in the space state. The combination $x_i^{(1)} = x_i^{(0)} = 1$ is not allowed. Similarly, to implement a three-state output, the function f_c, $c = 1, 2, \ldots, q$ should be represented in both positive $f_c^{(1)}$ and negative $f_c^{(0)}$ forms. If $f_c^{(1)} = 1, f_c^{(0)} = 0$, then the function f_c is in a state 1, if $f_c^{(1)} = 0, f_c^{(0)} = 1$, then the function f_c is in state 0, if $f_c(1) = f_c(0) = 0$, then function f_c is in the space state. The combination $f_c(1) = f_c(0) = 1$ is not allowed. To change the input state, the environment should reset it first to the space state and after that set it to a 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.

Initially, each function f_c is represented as a triplet: $f_c = (f_c^{(1)}, f_c^{(0)}, f_c^{(2)})$, where $f_c^{(1)}, f_c^{(0)}, f_c^{(2)}$ describe ON-, OFF- and DON'T CARE (DC) sets. Let $f^{(-)}$ be a subset, that belongs to DC sets of all functions: $f^{(-)} \subseteq f_c^{(2)}$, $c = 1, 2, \ldots, q$. We call $f^{(-)}$ as a set of unreachable input states (input combinations, that can't appear on the circuit inputs). Then, each function f_c, $c=1, 2, \ldots, q$, can be represented as a quadruplet: $f_c = (f_c^{(1)}, f_c^{(0)}, f_c^{(\sim)}, f^{(-)})$, where $f_c^{(1)}, f_c^{(0)}, f_c^{(\sim)}$ describe ON-, OFF- and DC-sets of reachable input states, $f_c^{(\sim)} = f_c^{(2)} \setminus f^{(-)}$. For the cost effective implementation, each function f_c should be represented as a pair of minimized (under a constraint that will be formulated below) ON- OFF-set terms: $f_c = (F_c^{(1)}, F_c^{(0)})$, where: $f_c^{(1)} \subseteq F_c^{(1)}, f_c^{(0)} \subseteq F_c^{(0)}, f_c^{(\sim)} \subseteq \{F_c^{(1)} \cup F_c^{(0)}\}, F_c^{(1)} \cap F_c^{(0)} \subseteq f^{(-)}$. Note, that in contrast to the synchronous logic, we distinguish DC set of reachable states ($f_c^{(\sim)}$) and DC set of unreachable ones ($f_c^{(-)}$), since each reachable state must be covered either by $F_c^{(1)}$ or by $F_c^{(0)}$ but not by both sets, to recognize the moment when the circuit responds on the new input state.

Each function f_c can be described as a Sum–Of-Product terms (SOP) for both positive $f_c^{(1)}$ and negative $f_c^{(0)}$ forms: $f_c^{(1)} = t_1 + t_2 + \ldots + t_k$, $f_c^{(0)} = t_{k+1} + t_{k+2} + \ldots + t_p$, $c = 1, 2, \ldots, q$, $p \leq 2^n$, where t_i are product terms containing n or less literals, $i = 1, 2, \ldots, p$, $F_c^{(1)} = \{ t_1 \cup t_2 \cup \ldots \cup t_k\}$, $F_c^{(0)} = \{t_{k+1} \cup t_{k+2} \cup \ldots \cup t_p \}$. A sum of orthogonal products is called a Disjoint-Sum-Of-Products (DSOP). If each term contains exactly n literals (minterm) we call such a representation as a Sum-Of-Minterms (SOM).

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.1.

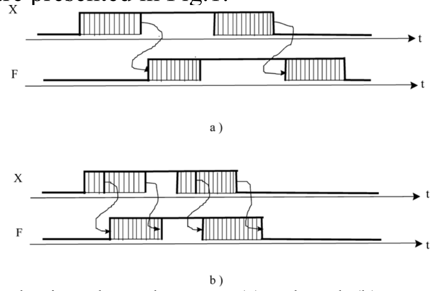


Fig. 1. Behavior rules under strong (a) and weak (b) constraints

The diagrams depict inputs X and outputs 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 a space to working state and vice versa.

Under the strong constraints (Fig. 1a), the behavior rule is as follows:

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

Under the weak constraints (Fig. 1b), 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 a space state then some outputs are going to a space state;
4. If all inputs are in a space state then all outputs are going to a space state;
5. Go to 1.

In both cases, it is supposed that the states of *all outputs* depend on the state of *all inputs*. As a result, the term minimization is not allowed. However, a careful observation shows that in some cases, the state of outputs can be determined based on the state of some (not all) inputs. For example, consider a function $f^{(1)} = x_1^{(1)} + x_1^{(0)}x_2^{(0)}$. If $x_1^{(1)} = 1$, then independently on the input x_2 state, the function $f = 1$. Based on this observation, one can modify the (weak) constraints as follows: if some inputs are in the working/space state then some *or even all outputs* are going to the working/space state. In Section IV it will be shown, that under such a modification, the term minimization is allowed.

III. DIMS

Two-level (DIMS) implementation was proposed in [14, 15], where both positive and negative functions should be represented as SOMs. It is supposed that the set of unreachable states is empty. Therefore, 2^n minterms should be generated to represent each function in the positive and negative form. On the first level, each minterm is implemented using the C-element. For each input $x_i \in X$, either signal $x_i^{(1)}$ or $x_i^{(0)}$ 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 its all input signals switch on/off. Therefore, C-element output indicates whether the circuit inputs are in a working or space state. Since minterms are not intersecting, only one C-element switches on/off at a time. The signal from the C-element output 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 see that DIMS operates under Seitz's strong constraints. Note, that under the strong constraints, output states are capable to indicate *all* new input states (i.e., no special indication logic is required), because *all* new input states imply *all* new output states.

DIMS implementation of the 2-variable AND function is depicted in Fig. 2. Suppose: $x_1^{(1)} = x_1^{(0)} = x_2^{(1)} = x_2^{(0)} = 0$ ("all inputs are in the space state"). One can check that in this case: $f^{(0)} = f^{(1)} = 0$ ("all outputs are going to the space state"). Suppose, the environment switches inputs x_1, x_2 to a working state, say 10: $x_1^{(1)} = x_2^{(0)} = 1$, $x_1^{(0)} = x_2^{(1)} = 0$. As a result, the C4-element output signal switches on and after propagating through the path (Fig. 2-bold) the circuit output switches to the working state 0: $f^{(1)} = 0$, $f^{(0)} = 1$ ("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 C4-element output switches off and therefore the output $f^{(0)}$ switches off ("if all inputs are in the space state then all outputs are going to the space state").

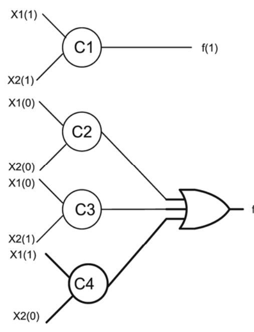


Fig. 2. DIMS implementation of AND function

IV. Model Based on Modified Weak Constraints

A. *Behaviour Rule*

The goal of our approach is to reduce the two-level logic cost. It is done in two ways: 1) term minimization is allowed; 2) the C-element logic is replaced by AND logic. Thus, the functions are represented as a two-level AND-OR (or, better, an equivalent NAND-NAND) logic. As shown above, in some cases the output state can be determined based on some (not all) inputs, see Section II.B. To design the minimized two-level AND-OR logic, we introduce the model with modified weak constraints under the following behavior rules (Fig. 3):

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

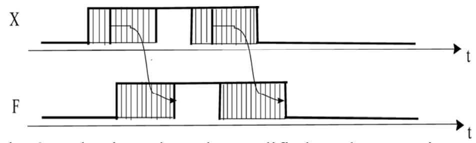


Fig. 3. Behavior rule under modified weak constraints

The structure consists of two blocks (Fig. 4): a two-level AND-OR structure (instead of C-OR) and the completion detection logic. Since minimization is allowed, functions contain less than 2^n product terms ($p \le 2^n$ - Fig. 4) and product terms may contain less than n literals: $|S(t_k)| \le n$, where $S(t_k)$ is a set of term t_k literals (input signals), $k = 1, 2, \ldots, p$. Note, that in contrast to DIMS, outputs are not capable to indicate the states of *all* inputs because output states may depend on *some* (not all) inputs. Furthermore, since multi-output functions are supposed, an additional signal is required to indicate the moment when *all* outputs are in a proper state (working or space). Therefore, the completion detection logic with an output signal D is introduced. The signal D switches on, when *both* inputs and outputs are in a working state.

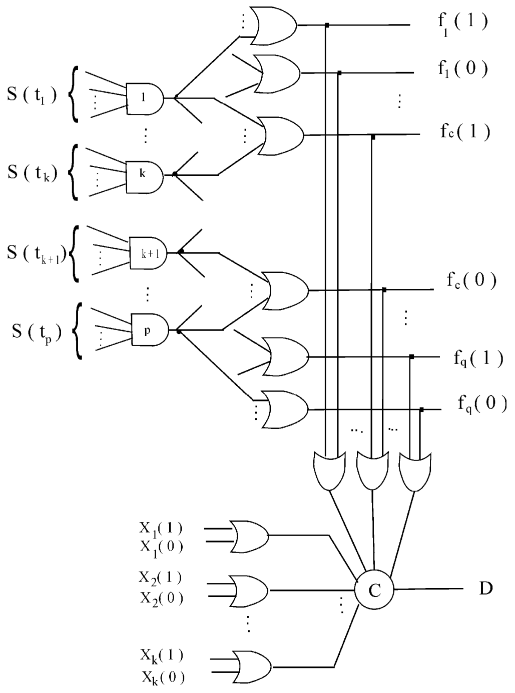


Fig. 4. Implementation under modified weak constraints

B. *Minimization Constraint*

It is supposed that within the cost effective implementation, the product term minimization is allowed. We should formulate the minimization constraint to ensure a proper (Section IV.A) behavior rule.

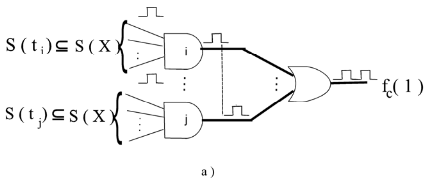
Given an implementation (Fig. 4) of a multi-output function F.

Theorem 1. The behaviour of any function f_c (Fig. 4), $f_c \in F$, does not violate the behavior rule (Section IV.A) iff: $\{t_i \cap t_j\}\backslash f^{(-)} = \emptyset$, for $\forall(t_i, t_j)$: $t_i, t_j \in F_c^{(1)}$, and $t_i, t_j \in F_c^{(0)}$.

Proof. Necessity. Suppose, that $\{t_i \cap t_j\}\backslash f^{(-)} \neq \emptyset$. Terms t_i, t_j can't belong to different sets $F_c^{(1)}$, $F_c^{(0)}$, since $\{F_c^{(1)} \cap F_c^{(0)}\}\backslash f^{(-)} = \emptyset$. Therefore, either $t_i, t_j \in F_c^{(1)}$ or $t_i, t_j \in F_c^{(0)}$. First, suppose $t_i, t_j \in F_c^{(1)}$. Consider the circuit fragment containing AND$_j$ gates connected to an OR gate implementing the function $f_c^{(1)}$ (Fig. 5a). Let $S(t_i)$ be a set of term t_i literals (input signals). Define a joint set $S(t_{ij})$: $S(t_{ij}) = S(t_i) \cup S(t_j)$. Let S(X) be a set of input signals that switch on once inputs switch to given reachable working state. For example, given input working state is: $x_1^{(1)} = x_2^{(0)} = 1$. Then $S(X) = \{x_1^{(1)}, x_2^{(0)}\}$. Suppose: $S(t_{ij}) \subseteq S(X)$ and: 1) a signal $x_l(u)$ has the longest switching delay among the signals of set

S(t_i); 2) a signal x_r has the longest switching delay among the signals of set S(t_j), $x_l(u) \in$ S(t_i) , $x_r(u) \in$ S(t_j) , $u \in \{0,1\}$. When the signal $x_l(u)$ switches on then the AND$_j$ gate output switches on. Similarly, when the signal $x_r(u)$ switches on then the AND$_j$ gate output switches on. As a result, signal 1 propagates through two paths: AND$_i$ - OR, AND$_j$ - OR. Suppose that the sum of the signal $x_l(u)$ switching delay and the path AND$_i$-OR delay is shorter than the sum of the signal $x_r(u)$ switching delay and the path AND$_j$ - OR delay. In this case, the signal 1 propagating through the path AND$_i$ - OR switches the output $f_c^{(1)}$ on (the output f_c goes to the working state).

Now suppose that any signal $x_a(u) \in$ S(t_i) switches off (the input x_a goes to the space state). It implies switching the AND$_i$ gate output off. As a result, the signal 0 propagates through the path AND$_i$-OR and the function $f_c^{(1)}$ switches off (the output f_c goes to the space state). Due to a longer propagation delay, the signal 1 propagating through the path AND$_j$ - OR may switch the output $f_c^{(1)}$ on again and later off due to switching any signal $x_b(u) \in$ S(t_j) off (erroneous pulse - Fig. 5b). It violates the behavior rule ("if some inputs are in the space state then some outputs are going to the space state" and remain in this state). The above conclusion is valid, if $t_i, t_j \in F_c^{(0)}$.



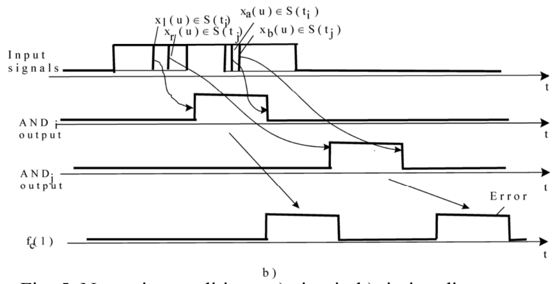


Fig. 5. Necessity conditions: a) circuit, b) timing diagram

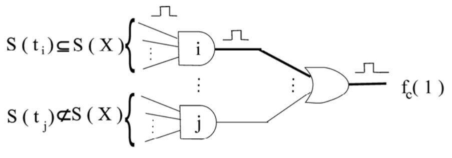
*Sufficiency***.** Again consider the same circuit fragment (Fig. 6a). Suppose: $\{t_i \cap t_j\} \setminus f^{(-)} = \emptyset$, $t_i, t_j \in F_c^{(1)}$ and S(t_i) ⊆ S(X). Then, S(t_j) ⊄ S(X) because of $\{t_i \cap t_j\} \setminus f^{(-)} = \emptyset$ and in contrast to the previous case, there is only one path (Fig. 6a, bold line) the signal propagates to the output through.

Therefore, if signals from the set S(t_i) switch on, then the AND$_i$ gate output switches on and, in turn, the output $f_c^{(1)}$ switches on (see Section IV.A, rule 1). Once any signal $x_a(u) \in$ S(t_i) switches off then the output $f_c^{(1)}$ switches off (see Section IV.A, rule 3) (Fig. 6b). As a result, the behavior of any function $f_c \in F$ doesn't violate the behaviour rule (Section V.A). This conclusion is valid for a function $f_c^{(0)}$, if $t_i, t_j \in F_c^{(0)}$.

■

C. Example

Consider the implementation (Fig. 7) of a minimized 2-variable AND function $f^{(1)} = x_1^{(1)}x_2^{(1)}, f^{(0)} = x_1^{(0)}x_2^{(1)} + x_2^{(0)}$.



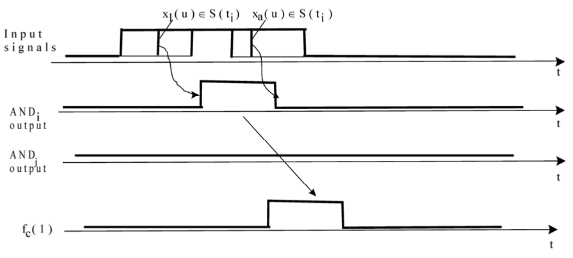


Fig. 6. Sufficiency conditions: a) circuit, b) timing diagram

The circuit is implemented using three two-input gates. The completion detection logic contains three OR gates that indicate the state of inputs x_1, x_2 and the output f. Let us show that the implementation satisfies the behavior rule formulated in the Section IV.A. Initially, all inputs and the output are in the space state. Suppose the input x_2 switches to the working state 0: $x_2^{(1)} = 0$, $x_2^{(0)} = 1$. In response, the output switches to the working state. The state of the input x_1 doesn't affect the output state. Therefore, when the input x_1 is in the working state, the output remains in the working state. The moment when both inputs together with the output are in the working state is indicated by the signal D = 1. Now suppose that $x_2^{(0)} = 0$. Then the output $f^{(0)} = 0$. The notification of the moment when the inputs and the output are in the space state is done by the signal D = 0.

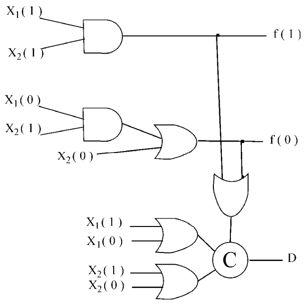


Fig. 7. Implementation example

V. Synthesis Procedure

The process of synthesis of the two-level AND-OR dual-rail logic is based on the tools ESPRESSO [18], BOOM [19] and DSOP [20]. These are conditions imposed on the final design: 1) both the minimized ON-set $F_c^{(1)}$ and OFF-set $F_c^{(0)}$ are produced for each output function f_c; 2) each ON-, OFF-set state of f_c must be covered by $F_c^{(1)}$ or $F_c^{(0)}$ accordingly; 3) each state of the sets $f^{(-)}$,$f_c^{(-)}$ must be covered either by $F_c^{(1)}$ or $F_c^{(0)}$, not both; 4) $F_c^{(1)} \cap F_c^{(0)} = \emptyset$; 5) ON-set terms must be mutually orthogonal (the same for OFF-set terms). Note, that due to tools [18, 19, 20] features, some of these conditions are stronger than required. Namely, as defined in the Section II.A, states of the set $f^{(-)}$ may be covered by both $F_c^{(1)}$ and $F_c^{(0)}$

and therefore, $F_c^{(1)}$ and $F_c^{(0)}$ may be intersecting sets. Finally, as stated in the Theorem 1 (Section IV.B), the terms may be non–orthogonal and the intersection of each pair of terms must be covered by the set $f^{(-)}$. Further, we expect to modify tools [18, 19, 20] to ensure implementation under the conditions as they are formulated in the Sections II.A and IV.B. As a consequence of the condition 3, each DC state must be assigned either to the ON- or to the OFF-set. Since no DC state should be covered by both the $F_c^{(1)}$ and $F_c^{(0)}$, the challenge comes to decide what DCs should be assigned to the ON- and what to the OFF- set. Unfortunately, available Boolean minimizers [18, 19] are not able to minimize the ON- and OFF-set simultaneously, and thus cannot exploit the DCs most efficiently. Therefore, the pre-minimization must be done to choose the assignment of each DC state either to ON- or OFF–set. Since the DC assignment determines the complexity of the final logic, each problem pre-minimization is done twice – once exploiting all the DCs by the ON-set minimization, once by the OFF-set. Both results are processed further and the best final result is selected.

Another issue emerging in the two-level dual-rail logic design is the possibility of sharing of terms between the ON- and OFF-sets. It is an analogy to a group minimization, where terms are being shared between different outputs. In the dual-rail logic design, terms may be efficiently shared between outputs in both their positive and negative forms. Of course, no term can belong to both the ON- and OFF-set of the same function. However, it may be an ON- and OFF-set term of different functions. Given function $f_c = (f_c^{(1)}, f_c^{(0)})$ obtained as a result of the pre-minimization. Note, that all DC states are already assigned either to $f_c^{(1)}$ or $f_c^{(0)}$ (completely specified function) . Let us extend the function F: $F = \{f_1, f_2, \ldots, f_q, f_{q+1}, f_{q+2}, \ldots, f_{2q}\}$, where $f_g = (f_g^{(1)}, f_g^{(0)}), f_g^{(1)} = f_c^{(0)}, f_g^{(0)} = f_c^{(1)}$, $g = q+1, q+2, \ldots, 2q$. After the minimization of the extended function F, we obtain both the ON- and OFF-sets of each function $f_c = (F_c^{(1)}, F_c^{(0)})$, where $F_c^{(0)} = F_{q+c}^{(1)}$, $c = 1, 2, \ldots, q$. Once the minimization is performed, we run DSOP [20] on both $F_c^{(1)}$ and $F_c^{(0)}$, to obtain the final result satisfying the above condition 5.

VI. Experimental Results

We have processed the MCNC set of benchmarks [17]. For each benchmark, the positive and negative form of each function is generated as a SOM and implemented as a DIMS [14]. Then the proposed minimization procedure is applied to obtain the minimized DSOP.

We have compared the complexity (expressed as the gate equivalents (GEs) number [21]) of our implementation with DIMS [14]. We suppose a technology independent synthesis and therefore, logic with unbounded inputs. The complexity of unbounded logic is estimated as follows: an n-input NAND or NOR gate requires $0.5n$ GEs, an n-input AND or OR gate requires $0.5(n+1)$ GEs and an n-input C-element requires $n+1$ GEs for implementation [16]. Let's suppose a multi-output function with n inputs and q outputs. In DIMS, the first level is implemented using 2^n n-input C-elements. Therefore, the complexity of this level is $2^n(n+1)$ GEs. Since both the on-set and off-set are to be produced for each output, the second level is implemented by $2q$ OR gates, while each OR gate pair has 2^n inputs in total. Therefore, the complexity of DIMS second level is $q(2^n/2+1)$ GEs. As a result, the DIMS total complexity is: $2^n(n+1) + q(2^n/2+1)$.

For the structure proposed, we estimate the complexity of the functional block and completion detection logic separately. Then, the total complexity is calculated. To avoid additional inverters and therefore decrease the implementation complexity, we use negative (NAND-NAND) gates instead of AND-OR ones in the functional block and NOR gates instead of OR ones in the completion detection logic. As a result, the signal D = 1 (D = 0), when all inputs and outputs are in the space (working) state. To implement an $(n+q)$-input C-element, $(n+q+1)$ GEs are required. To implement $n+q$ two-input NOR gates, $0.5(n+q)$ GEs are required. Totally, $(1.5n+1.5q+1)$ GEs are required to implement the completion detection logic for an n-input, q-output circuit.

The experimental results are summarized in Tables I and II, for selected representative MCNC [17] benchmarks. Table I presents a comparison of the proposed method with the synchronous logic and DIMS. Here only benchmarks with a relatively small number of inputs (up to 11) were considered, to make the comparison meaningful. For "bigger" benchmarks, the DIMS implementation would be of a prohibitive large complexity. The table columns are marked as follows: first, the circuit name is given, with the numbers of inputs (n) and outputs (q). Then the complexity of a conventional two-level synchronous implementation is shown, in sense of the number of product terms and GEs. The complexity of the DIMS implementation is indicated in the next two columns. The proposed method results follow. The synthesis was run twice, once minimizing the ON-set first, and then minimizing the OFF–set first. Numbers of product terms and GEs are shown in the next two pairs of columns. The number of the function logic terms is shown in the "*Terms*" column. The complexity of the completion detection logic is indicated in the "*CL*" column. The total complexity of the best design is shown in the "*Total*" column. The overall cost reduction, with respect to DIMS, is shown in the last column. The average improvement with respect to DIMS is 71.36% for these benchmarks.

The area increase, with respect to the synchronous implementation, is given in the "*Overhead*" column. In particular, it is the ratio of the GEs number increase in the case of the asynchronous implementation and the overall asynchronous logic complexity ("*Total GEs"*).

Note, that the complexity of the sequential logic memory (flip-flops, latches) is not included in the results.

In Table II, there are shown representatives of benchmarks having 30 and more inputs, just for illustration. No comparison with the state-of-the-art is possible here.

VII. Conclusion

We proposed a cost effective (in sense of gate equivalents) dual-rail implementation of asynchronous two-level logic. It is based on a model with the modified weak constraints. The proposed implementation consists of a functional AND-OR block and completion detection logic. A newly introduced minimization constraint allows us to reduce the AND-OR logic complexity, yielding a significant complexity reduction.

A universal method for a dual-rail logic synthesis was then proposed. Here simultaneous minimization of both positive and negative forms (ON- and OFF-set) of the function is involved, as a generalization of a group minimization.

The dual-rail asynchronous logic synthesis method was tested on MCNC benchmarks. In comparison to the state-of-the-art method (DIMS [14]), we achieved a significant improvement in sense of the final design complexity. Benchmarks, for which the synthesis as DIMS was inconceivable, were easily synthesized by our method.

Table II.
"Big" benchmarks

			ON-set first		*OFF-set first*			
Bench	*n*	*q*	*Terms*	*GEs*	*Terms*	*GEs*	*CL*	*Total*
apex1	45	45	1180	8280.5	1103	8280.5	136	8416.5
apex3	54	50	313	5724.5	335	5820.5	157	5881.5
e64	65	64	417	3114.5	415	3099	194.5	3293.5
exep	30	63	976	4701.5	1214	6441	140.5	4842
ibm	48	17	896	4024	896	4020.5	98.5	4119
misg	56	23	158	512.5	158	512.5	119.5	632
seq	41	35	514	8461.5	463	8433.5	115	8548.5
soar	83	94	749	4621	749	4616.5	266.5	4883

ACKNOWLEDGMENT

For the second author, this research has been supported by MSMT under research program MSM6840770014.

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] 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

[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] 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

[12] A. DeHon, Nanowire-Based Programmable Architectures, ACM Journal on Emerging Technologies in Computing Systems, vol. 1, no. 2, pp. 109–162, 2005.

[13] A. DeHon and M. J. Wilson, Nanowire-Based Sublithographic Programmable Logic Arrays, in Proceedings of the International Symposium on Field-Programmable Gate Arrays, February 2004, pp. 123–132T.S. Anantharaman, A Delay Insensitive Expression Recognizer, IEE VLSI Tech.Bull, Sept, 1986

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

[15] 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.

[16] E.J. Sparsø, S. Furber, Principles of Asynchronous Circuit Design, Kluwer Academic Publishers, 2001, 337 p.

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

[18] R.K.Brayton, et al, Logic Minimization Algorithm for VLSI Synthesis, Norwell, MA: Kluwer Academic, 1984

[19] J. Hlavička and P. Fišer, BOOM - a Heuristic Boolean Minimizer, Proc. ICCAD-2001, San Jose, Cal. (USA), 4.-8.11.2001, pp. 439-442

[20] A. Bernasconi, V. Ciriani, F. Luccio, L. Pagli, A New Heuristic for DSOP Minimization, Proc. 8th Int. Workshop on Boolean Problems (IWSBP'08), Freiberg, Germany, 18.-19.9.2008, pp. 169-174

[21] G. De Micheli, Synthesis and Optimization of Digital Circuits. McGraw-Hill, 1994

Table I.
Comparison with synchronous logic and DIMS

			Synchronous		*DIMS*		*ON-set first*		*OFF-set first*					
Bench	*n*	*q*	*Terms*	*GEs*	*Terms*	*GEs*	*Terms*	*GEs*	*Terms*	*GEs*	*CL GEs*	*Total GEs*	*Overhead*	*Cost red.*
5xp1	7	10	65	172.5	128	1674	84	367	91	376.5	26.5	393.5	56%	76.49%
9sym	9	1	86	301	512	5377	206	948.5	206	949.5	16	964.5	69%	82.06%
apex4	9	19	436	2709.5	512	10003	489	5953	478	5977	43	5996	55%	40.06%
b11	8	31	27	83	256	6303	40	334	40	316	59.5	375.5	78%	94.04%
bw	5	28	22	173.5	32	668	24	387.5	24	378.5	50.5	429	60%	35.78%
clpl	11	5	20	35	2048	29701	40	115	40	115	25	140	75%	99.53%
dc1	4	7	9	27	16	143	12	64.5	12	64.5	17.5	82	67%	42.66%
m1	6	12	19	108	64	844	25	165	23	159	28	187	42%	77.84%
max1024	10	6	274	1133	1024	14342	460	2437	464	2411.5	25	2436.5	53%	83.01%
z9sym	9	1	86	301	512	5377	205	945	206	949.5	16	961	69%	82.13%
Average													62%	71.36%