

Cost Effective Implementation of Asynchronous Two-Level Logic

Igor Lemberski, *Member, IAENG*

Abstract - We proposed the cost effective (in sense of gate number) asynchronous two-level logic. It is based on AND-OR implementation of minimized logic functions. We formulated and proved the product term minimization constraint that ensures the logic correct behaviour. We pointed out the existing tool that yields the term minimization under the constraint formulated. We processed examples and generated asynchronous two-level logic by applying our approach and known ones. The implementation complexity was compared. Using our approach, we achieved significant improvement.

Index Terms - asynchronous logic, minimization, two - level logic.

1. 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 with 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 [16]. Within this model, only one input signal can be changed at the time. In [10], the generalized fundamental mode was proposed where multiple input change is allowed during the narrow time interval. For such a mode, the method of hazard – free two-level implementation was proposed [11]. The multi-level (hazard not increasing) transformation is applied to optimize the implementation. It is based on the rules formulated in [16] and extended in [7]. The methods of hazard-free technology mapping were proposed in [2,14].

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 dual-rail encoding. To change an input state the environment should reset it first (change to so called space state). As a result the output state resets too. After that the environment sets a new input state. It implies a new output state.

The behaviour rule is based on Seitz's strong or weak constraints [5,13]. 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 multi-level logic, the hazard-free design methodology of circuits implemented using simple gates and NCL was proposed [4,9]. It is based on circuit transformation into monotonic (and therefore, hazard-free) one. The transformation includes converting each internal node into two nodes to produce the outputs for both right and inverse form. The drawback is that the circuit cost increases twice.

The two - level logic is attractive due to its high performance (a signal from inputs to outputs propagates through two levels only), regularity and good starting point for multi - level logic. In [1,15], the two-level (Delay-Insensitive Minterm System-DIMS) multi- output logic was proposed where C-elements [12] are used on the first level to implement product terms and OR-gates are used on the second one to generate functions. The behaviour is based on Seitz's strong constraints. However, DIMS implementation cost is very expensive since term minimization is not allowed. Therefore, 2^n (all possible) terms of length n each (n -number of inputs) should be generated and each term is implemented using the n – input C-element, which is complex itself. In [8], the two-level logic behaviour is based on the modified weak constraints namely, *all outputs* are permitted to change their state if *some – not all - inputs* have changed their state. As a result, the term minimization is allowed but the minimization procedure should generate mutually orthogonal product terms to ensure correct behaviour (in terms of modified weak constraints [8]). However, the implementation cost still remains expensive since each term is implemented using C-element. In this paper, we propose the cost effective two-level implementation. It is based on the further modification of the weak constraints: we suppose that in the reset phase, some or even *all outputs* may change their state if a *single input* has changed its state. We

Manuscript received March 22, 2007.

I. Lemberski is with Baltic International Academy, Lomonosova 4, Riga, LV-1003, LATVIA (phone: +37122 33 29 61; fax: +371 7 241 272; e-mail: Igor.Lemberski@bsa.edu.lv).

show that such a behaviour constraint leads to minimized two-level logic where the first level can be implemented using AND (instead of C-elements) gates. We formulate and prove the product term minimization constraint to ensure correct behaviour. We point out the existing tool that yields the product term minimization under the constraint formulated. Finally, we compute examples based on our approach and known ones. We estimate and compare the implementation complexity.

II. PRELIMINARIES

A. Input/Output Dual-Rail Encoding

Generally, an asynchronous logic should be capable 1) to recognize the moment when the new input state (generated by the environment) appears on the inputs and the moment when the circuit generates the new output state in response on the input one; 2) to notify the environment on the new input and output states. After receiving notification the environment can generate next input state. To solve this problem, inputs/outputs are implemented using dual-rail encoding.

Let $F = \{f_1, f_2, \dots, f_q\}$ be the set of functions of n inputs X : $X = \{x_1, x_2, \dots, x_n\}$. We also call F as a multi-output function of multi-input X . The canonical representation of two-level logic is the sum of the 1-product terms: $f_c(1) = t_1 \vee t_2 \vee \dots \vee t_k$, $c = 1, 2, \dots, q$, $k \leq 2^n$, t_i – product term, $i = 1, 2, \dots, k$. Let T^1, T^0 be sets of function f_c 1- and 0- terms respectively. Two terms t_i, t_j are orthogonal ($t_i \text{ ort } t_j$) if one term contains a literal in the right form and the other one – the same literal in the inverse form. Otherwise, terms are not orthogonal ($t_i \text{ nort } t_j$).

It is supposed that each input and output (function) may be in three states: state 1, 0 (so called working states) or undefined (space state). To implement three-state input x_i , $i = 1, 2, \dots, n$, 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 (combination $x_i(1) = 1$ and $x_i(0) = 1$ is not allowed). Similarly, to implement three-state output, function f_c , $c = 1, 2, \dots, q$, should be represented in both right $f_c(1)$ and inverse $f_c(0)$ forms where functions $f_c(1), f_c(0)$ are represented as sums of 1- and 0-product terms respectively: $f_c(1) = t_1 \vee t_2 \vee \dots \vee t_k$, $f_c(0) = t_{k+1} \vee t_{k+2} \vee \dots \vee t_m$, $m \leq 2^n$. If $f_c(1) = 1$, $f_c(0) = 0$ then function f_c is in state 1, if $f_c(1) = 0$, $f_c(0) = 1$ – function f_c is in state 0 (working states), if $f_c(1) = f_c(0) = 0$ – function f_c is in the space state (combination $f_c(1) = f_c(0) = 1$ is not allowed). To change the input state the environment should reset it first (change to space state) and after that set it to the proper working state. Once the input is in the working state, the new input state is recognized. In the reset phase, the output state changes from a working one to space one and in the set phase the new output state is recognized.

B. Strong and weak constraints

In the fundamental mode, the asynchronous logic operates following so called strong or weak constraints [13] (fig.1). Under the strong constraints, the behaviour rule is as follows:

1. If all inputs are in a space state then all outputs are in a

space state;

2. If all inputs are in a working state then all outputs are in a working state;

3. Go to 1.

Under the weak constraints, some outputs are permitted to change their states when some (not all) inputs have changed their states:

1. If all inputs are in a space state then all outputs are in a space state;

2. If some inputs are in a working state then some outputs are in a working state;

3. If all inputs are in a working state then all outputs are in a working state;

4. If some inputs are in a space state then some outputs are in a space state;

5. Go to 1.

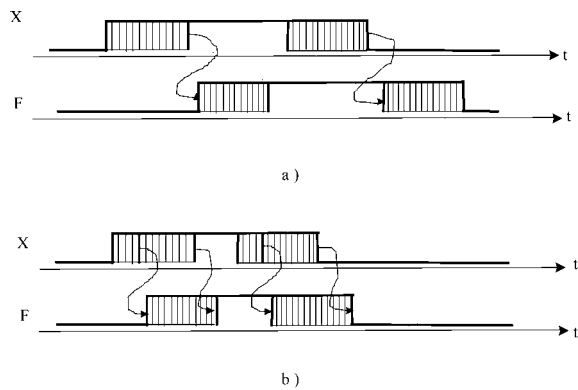


Fig.1. Behaviour rule under strong (a) and weak (b) constraints

In both cases, it is supposed that the state of *all outputs* depends on the state of *all inputs*. As a result, the term minimization is not allowed (DIMS). However, the 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 single output function $f = x_1(1) \vee x_1(0)x_2(0)$. If $x_1(1) = 1$ then independently on input x_2 state, function $f = 1$. Based on this observation, the weak constraints were modified as follows [8]: if some inputs are in the working/space state then some *or even all outputs* are in the working/space state. It was shown [8] that under the modified weak constraints, the term minimization is allowed. However, to ensure correct behaviour the resulting product terms should be mutually orthogonal.

III. DIMS

Two-level (DIMS) implementation was proposed in [1,15]. After input/output dual-rail encoding, 2^n terms of length n are generated to represent each function in both right and inverse forms. On the first level, each term is implemented using the C-element. For every input $x_i \in X$, either signal $x_i(1)$ or $x_i(0)$ is connected to each C-element inputs. The second level is implemented using OR gates. 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 notifies on the moment when circuit inputs are in the working or space state. Due to term orthogonality, only one C-element switches on/off. The signal from C-element output propagates through the single path to gate OR (circuit) output. As a result, the circuit output notifies on the new input state. One can easily see that DIMS operates under Seitz's strong constraints. As an example, consider two-level implementation of function AND of 2 variables (fig.5). The function is given by its canonical representation: $f=x_1x_2$. After dual-rail encoding, function right and inverse form can be represented as follows: $f(1)=x_1(1)x_2(1)$, $f(0)=x_1(0)x_2(1) \vee x_1(0)x_2(0) \vee x_1(1)x_2(0)$.

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 in 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, C4-element output signal switches on and after propagating through the path (fig.2-bold) the circuit output switches to working state 0: $f(1)=0$, $f(0)=1$ ("if all inputs are in a working state then all outputs are in a working state"). When all inputs switch from working state 10 to the space state then C4 –element output switches off and therefore output $f(0)$ switches off ("if all inputs are in the space state then all outputs are in the space state").

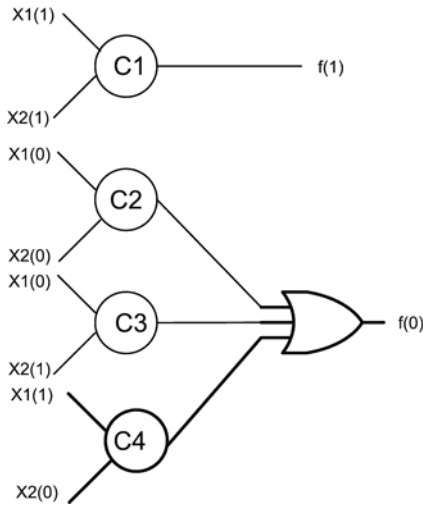


Fig.2. DIMS implementation

IV. IMPLEMENTATION BASED ON MODIFIED WEAK CONSTRAINTS

A. Behaviour Rule

For DIMS, one can see that 2^n terms should be implemented using C-elements. In other words, the term minimization procedure widely used in the synchronous logic to reduce complexity is not allowed. To avoid this restriction and therefore generate less complex (in sense of gate number) circuits, the modified weak constraints were proposed in [8]. It was noted that in some cases the output state can be determined

based on some (non-redundant) inputs (see example in paragraph II.B). The behaviour rule is as follows (fig.3):

1. If some inputs are in the working state then some or even all outputs are in the working state;
2. If all inputs are in the working state then all outputs are in the working state;
3. If some inputs are in the space state then some or even all outputs are in the space state;

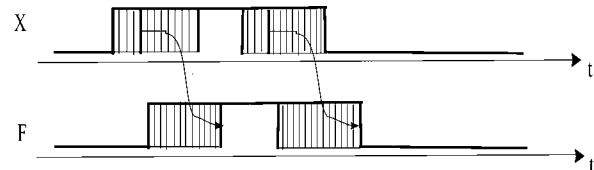


Fig. 3. Behaviour rule under modifies weak constraints

4. If all inputs are in the space state then all outputs are in the space state;
5. Go to 1.

The redundant inputs can be removed using the term minimization procedure. It was proven that asynchronous two-level logic [8] satisfies above formulated behaviour rule iff the minimization procedure generates mutually orthogonal terms.

The structure consists of two blocks (fig.4): a two-level circuit to implement functions and completion detection logic. In the circuit, each term is implemented using the C-element and several functions may share the same C-elements. Since minimization is allowed, the functions can be implemented using less than 2^n C-elements ($m \leq 2^n$ - fig.4). Also, some C-elements may contain less than n inputs: $|S(t_k)| \leq n$, where $S(t_k)$ – set of term t_k literals (input signals), $k=1,2,\dots,m$. One can see, that the circuit outputs may be used to notify on the working/space state of non-redundant inputs when C_k – element switches on/off. The notification on the state of any redundant (and therefore, removed) input x_i , $x_i \in \{x_{i1}, x_{i2}, \dots, x_{iv}\}$, $x_i(0) \notin S(t_k)$, $x_i(1) \notin S(t_k)$ should be done by the completion detection logic where the pair of ORed signals $x_i(0), x_i(1)$ is connected to C-element input. Signal D indicates (by switching to 1/0) the moment when redundant inputs are in the working /space state.

B. Example

Again, consider function AND of two variables: $f(1)=x_1(1)x_2(1)$, $f(0)=x_1(0)x_2(1) \vee x_1(0)x_2(0) \vee x_1(1)x_2(0)$. The minimization procedure that yields mutually orthogonal terms can be applied to the inverse form: $f(0) = x_1(0)x_2(1) \vee x_2(0)$. One can see that function $f(0)$ second term contains neither $x_1(1)$, nor $x_1(0)$. Therefore, if inputs switch from/to the space state to/from one of the working states: $x_1(0)=x_2(0)=1$ or $x_1(1)=x_2(0)=1$, input x_1 state can't be notified by output signal $f(0)$. The notification should be done by completion detection logic.

V. COST EFFECTIVE IMPLEMENTATION

A. Behaviour Rule

The goal of our approach is to propose further reducing the complexity of the two-level logic. Actually, we wish to implement the logic as a two-level circuit with AND gates (instead of C-elements) on the first level and OR gates on the second level. For this purpose, further modification of the behaviour rule is introduced (fig.6):

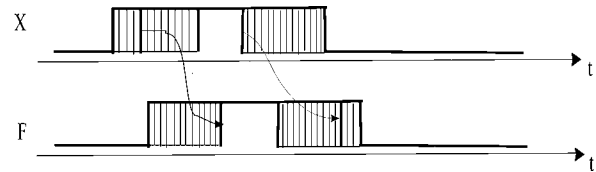


Fig.6. Behaviour rule for cost effective implementation

1. If some inputs are in the working state then all outputs are in the working state;
2. If all inputs are in the working state then all outputs remain in the working state;
3. If a single input is in the space state then some or all outputs are in the space state;
4. If all inputs are in the space state then all outputs are in the space state;
5. Go to 1.

Comparing with the rule (paragraph IV. A), the difference is in the reset phase 3 where switching a single input to the space state results in switching some or even all outputs to the space state. The cost effective logic (fig.7) consists of a two-level AND-OR (instead of C-OR) circuit and completion detection that indicates the state of all inputs $x_i \in X, i=1, 2, \dots, n$. It differs from the completion detection (fig.4) that indicates the state of redundant inputs only. The reason of such a modification is as follows: C- elements (first level logic) are replaced for AND gates. Therefore, in the reset phase, the circuit outputs are not capable to indicate the state of non-redundant inputs since changing the state of any single input implies changing the state of all outputs.

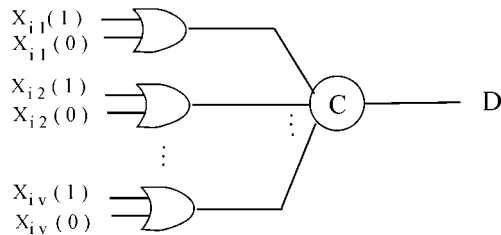
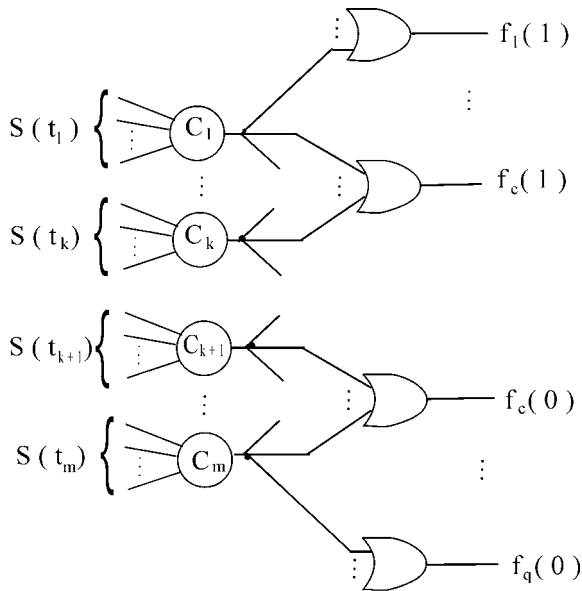


Fig.4. Structure

B. Minimization Constraint

It is supposed that within the cost effective implementation, the product term minimization is allowed. We should formulate minimization constraint to ensure proper (paragraph V.A) behaviour rule.

Given the cost effective implementation (fig.7) of function set F.

Theorem 1. The behaviour of any function f_c (fig.7), $f_c \in F$, doesn't violate the behaviour rule (paragraph V.A) iff: t_i ort t_j , for $\forall(t_i, t_j) : t_i, t_j \in T^1$, and $t_i, t_j \in T^0$.

Proof. Necessity. Consider the circuit fragment containing AND_i, AND_j gates connected to OR gate (fig.8a). Suppose: t_i nort $t_j : t_i, t_j \in T^1$. Suppose: $S(t_i)$ – set of term t_i literals (input signals). For non-orthogonal terms, define joint set $S(t_{ij}) : S(t_{ij}) = S(t_i) \cup S(t_j)$. Let $S(X)$ be the set of input signals that switch on once inputs switch to given working state. Suppose: $S(t_{ij}) \subseteq S(X)$. Suppose: signal $x_1(u) / x_r(u)$ has the longest switching delay among the signals of set $S(t_i)/S(t_j)$, $x_1(u) \in S(t_i)$, $x_r(u) \in S(t_j)$, $u \in \{0,1\}$. When signals $x_1(u), x_r(u)$ switch on then AND_j, AND_j gate outputs switch on. As a result, signal 1 propagates through two paths: AND_i-OR, AND_j-OR . Suppose that the sum of signal $x_1(u)$ switching delay and path AND_i-OR

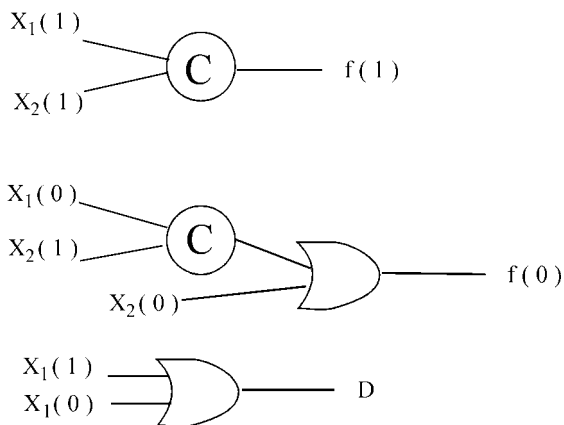


Fig. 5. Implementation example

delay is shorter than the sum of signal $x_r(u)$ switching delay and path AND_j-OR delay. In this case, the signal 1 propagating through path AND_i-OR switches output $f_c(1)$ on (output f_c goes to the working state).

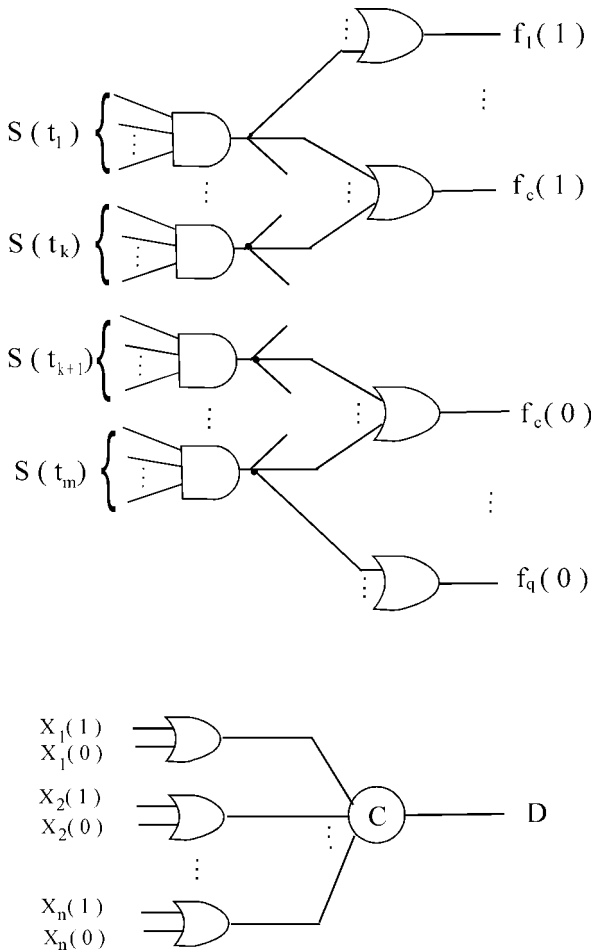


Fig.7. Cost effective implementation structure

Now suppose that any signal $x_a(u) \in S(t_i)$ switches off (input x_a goes to the space state). It implies switching AND_i gate output off. As a result, signal 0 propagates through path AND_i-OR and function $f_c(1)$ switches off (output f_c goes to the space state). Due to longer propagation delay, the signal 1 propagating through path AND_j-OR may switch 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.8,b). It violates the behaviour rule (“if a single input is in the space state then some or all the outputs are in the space state”). The above conclusion is valid for function $f_c(0)$, if $t_i, t_j \in T^0$.

Sufficiency. Again consider the same circuit fragment (fig.9a). Suppose: t_i ort t_j , $t_i, t_j \in T^1$ and $S(t_i) \subseteq S(X)$. In this case, $S(t_j) \not\subseteq S(X)$, $\forall t_j \in T^1 \setminus t_i$ due to the orthogonality of t_i, t_j and in contrast to the previous case, there is only one path (fig.9,a, bold line) the signal propagates to the output through. Therefore, if signals from set $S(t_i)$ switch on then AND_i gate

output switches on and in turn, output $f_c(1)$ switches on (“if some inputs are in the working state then some or all outputs are in the working state”). Once any signal $x_a(u) \in S(t_i)$ switches off then output $f_c(1)$ switches off (“if a single input is in the space state then some or all outputs are in the space state”) (fig.9b). As a result, the behaviour of any function $f_c \in F$, doesn't violate the behaviour rule (paragraph V.A).

The above conclusion is valid for function $f_c(0)$ if we suppose: $t_i, t_j \in T^0$. ■

Note that the product term constraint (mutually orthogonal terms) is the same as for the two-level logic [8]. Therefore, the tool [3] already mentioned in [8] can be used for the term minimization in the case of the cost effective implementation.

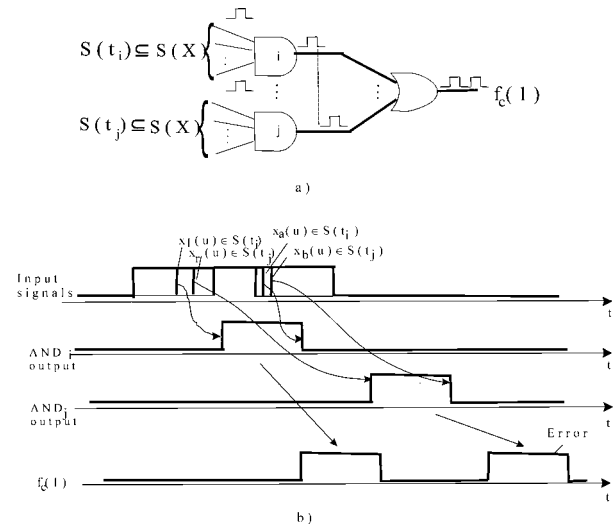


Fig.8. Necessity conditions, a) circuit, b) timing diagram

C. Example

Again, consider the implementation (fig.10) of minimized function AND of 2 variables: $f(1)=x_1(1)x_2(1)$, $f(0)=x_1(0)x_2(1) \vee x_2(0)$. The circuit is implemented using three two-input gates. Completion detection contains two OR gates that indicate the state of inputs x_1, x_2 . Let us show that the implementation satisfies the behaviour rule formulated in paragraph 5.1. Initially, all inputs and the output are in the space state. Suppose input x_2 switches to working state 0: $x_2(1)=0$, $x_2(0)=1$ (“some inputs are in the working state”). In response, the output switches to the working state (“all outputs are in the working state”). Since input x_1 is the redundant one its state doesn't affect the output state. Therefore, when input x_1 is in the working state, the output remains in the working state. The moment when all inputs (both redundant and non-redundant) are in the working state is indicated by signal $D=1$. Now suppose that signal $x_2(0)=0$ (“if a single input is in the space state...”). Then output $f(0)=0$ (“... then all outputs are in the space state”). The notification on the moment when all inputs are in the space state is done by signal $D=0$.

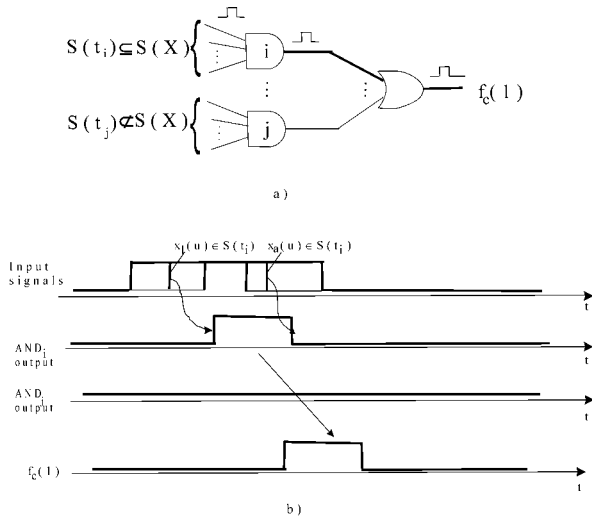


Fig.9. Sufficiency conditions, a) circuit, b) timing diagram

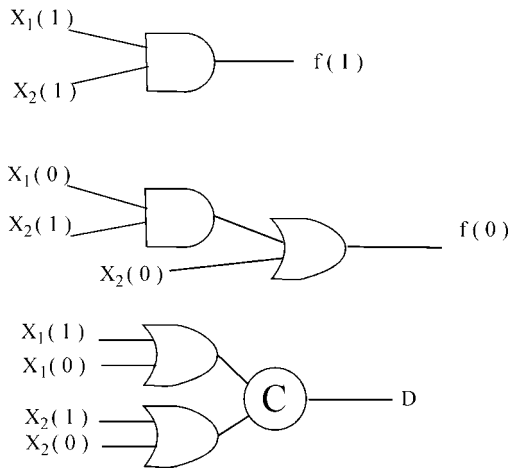


Fig.10. Implementation example

V. EXPERIMENTAL RESULTS

We processed 3 examples: function AND of 2, 3 and 4 variables (further: AND(2), AND(3), AND(4)). For minimization, espresso tool [3] with the option -dlmerge (espresso -dlmerge) is used. It merges terms that differ in one variable and therefore produces mutually orthogonal terms.

We compared the complexity (expressed as the number of 2 input gates $G(2)$) of our implementation (Minimized AND-OR structure - AND-OR/M) with DIMS [1] where minimization is not allowed and C-OR Minimized (C/OR/M) implementation [8]. To measure complexity, the approach [6] has been used. Within this approach, the complexity of the logic is estimated as follows: $C(n) = (n-1)C(2)$, $C(2) = 4G(2)$, $G(n) = (n-1)G(2)$, where $C(n)$ - C-element of n inputs, $G(n)$ - n -input One can see (Table1) that we achieved significant improvement and believe

that the improvement will be even higher for the functions of more variables.

Table 1. Function AND implementation complexity

Examples	DIMS[1]	C-OR/M[8]	AND-OR/M
AND(2)	18G(2) (fig.2)	10G(2) (fig.5)	9G(2) (fig.10)
AND (3)	102G(2)	28G(2)	18 G(2)
AND (4)	206G(2)	50G(2)	28 G(2)

VI. CONCLUSION

We proposed the cost effective (in sense of gate number) dual-rail implementation of asynchronous two-level logic. It is based on the modified weak constraints. Using the existing tool (espresso [3]) we processed examples and produced dual-rail two-level logic. We compared our implementation complexity with ones obtained using known approaches [1,8]. We achieved significant improvement.

REFERENCES

- [1] T.S. Anantharaman, A Delay Insensitive Expression Recognizer, IEE VLSI Tech.Bull, Sept, 1986
- [2] P. Beerel, K.Y.Yun, W.C. Chou, Optimizing 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
- [3] R.K.Brayton, et al, Logic Minimization Algorithm for VLSI Synthesis, Norwell, MA: Kluwer Academic, 1984
- [4] 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
- [5] W.J.Dally, J.W.Poulton, Digital Systems Engineering, Cambridge University Press, 1998, 663 p.
- [6] I.David, R.Ginosar, M.Yoeli, An Efficient Implementation of Boolean Functions as Self-timed Circuits, IEEE Trans. Computers, vol. 41, No 1, 1992, pp. 2- 11
- [7] D. Kung, Hazard-Non-Increasing Gate - Level Optimization Algorithm, IEEE Int. Conf. On Computer -Aided Design, November, 1992, pp. 631-634
- [8] I.Lemberski, M.B.Josephs, Optimal Two- Level Delay-Insensitive Implementation of Logic Functions, PATMOS2002, Spain, pp.109-119
- [9] 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 (ASYNC'00), pp. 114-125
- [10] S.M.Nowick, Automatic Synthesis of Burst-Mode Asynchronous Controllers, Ph.D. thesis, Stanford University, Mar March 1993
- [11] 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
- [12] Principles of Asynchronous Circuit Design, Ed. J.Sparsø, S.Furber, Kluwer Academic Publishers, 2001, 337 p.
- [13] C.L Seitz, System Timing, In: Introduction to VLSI Systems, C. Mead, L. Conway, Addison-Welsey Publishing Company, 1980, pp. 218-262
- [14] P. Siegel, G.D. Micheli, D. Dill, Automatic Technology Mapping for Generalized Fundamental Mode Asynchronous Designs, IEEE Design Automation Conference, June 1993, 61-67
- [15] J.Sparsø, J.Staunstrup, M. Dantzer-Sørensen, Design of Delay Insensitive Circuits Using Multi-Ring Structures, pp. 15-20
- [16] S.H. Unger, Asynchronous Sequential Switching Circuits, John Wilcy & Sons, Inc., 1969