Area Reduction in Asynchronous Circuits by Signal Transition Graph Transformations

George S. Panagiotaras and Odysseas Koufopavlou

VLSI Design Lab. Electrical & Computer Engineering Dpt.
University of Patras, 26500 Patras, GREECE
e mail: panagiot@ee.upatras.gr

ABSTRACT

Many asynchronous synthesis techniques utilize Signal Transition Graphs (STGs) as the initial circuit specification. In this paper, graph transformations that are applied to deterministic STGs are presented. Modifications of the graph concurrency achieve reduction in the resulting circuit area. The proposed transformations reduce the concurrency without influencing the functional operation of the circuit. The transformations are performed to a falling non-input signal transition. Three different STG examples are presented in the paper which indicate, by using the proposed transformations, up to 33 % reduction in the number of gates.

I. INTRODUCTION

Among the other possible advantages of asynchronous circuits, as compared to synchronous circuits, is their lower power dissipation, mainly due to the fact that there is no need to drive long clock lines at every cycle. However, the design of asynchronous circuits is not easier than the design of synchronous ones. Thus, design methodologies have been developed by engineers in order to face the difficulties of designing asynchronous circuits. One solution is the automatic synthesis of asynchronous circuits from a high level design description.

Many asynchronous circuits synthesis methodologies, use the Signal Transition Graph (STG) as the initial circuit specification [1-8]. In most of these methodologies, the goal was the optimization of the resulting circuit in speed and area. In this paper we focus on the reduction of the required area.

The rest of the paper is organized as follows: In section II some definitions are given. The proposed graph transformations are explained in section III. The results of the transformations used in three different STGs are presented in section IV. Finally, we conclude in section V.

II. DEFINITIONS

In this section some basic definitions are presented.

The STGs can be considered as a subset of Petri nets. A *Petri Net* is a particular kind of a directed graph, together with an initial state called the initial marking, M_0 [9]. More formally, a Petri net is defined as triple <P,T,F>, where P is a set of places, T is a set of transitions, and F is the flow relation $F \subseteq (P \times T) \cup (T \times P)$ [1]. A *Token Marking* of a PN is a non-negative integer labeling of its places. If $(p, t) \in F$, a place $p \in P$ is called a predecessor of a transition $t \in T$, and t is called a successor of p. Conversely, if $(t, p) \in F$, a transition t is a predecessor of a place p, and p is a successor of t. A *Free-Choice Petri Net* (FCPN) is a PN where, if a place p has more than one transition as its successors, then p must be the only predecessor of its successor transitions [10]. A *marked graph* is defined as a PN in which every place has only one predecessor and one successor [4].

A *Signal Transition Graph (STG)* is an interpreted FCPN where transitions are interpreted as value changes on input, output or internal signals of the specified circuit [1]. *Positive* transitions represent the rising ($0 \rightarrow 1$) signal transitions and are labeled with "+". *Negative* transitions represent the falling ($1 \rightarrow 0$) signal transitions and are labeled with "-". The set of signal transitions T can be partitioned in input, internal, and output signal transitions.

Input signal transitions are those that occur on the input signals of the circuit, and in the STG representation are underlined. O*utput signal transitions* are those that occur on its outputs. *Internal signal transitions* are the transitions that can happen on internal signals. The fundamental difference between transitions of input and non-input (internal and output) signals is that the former are caused by the external environment while the latter by the system [1] ,[2].

An STG is conventionally graphically represented by a directed graph, where transitions are denoted by the corresponding labels and places are denoted by circles. Places with one predecessor and one successor are usually omitted [1]. An *arc* in an STG represents a causal relation: for example $x^+ \rightarrow y^+$ means that x must go high before y can go high. When is said that a transition is *enabled* it means that the corresponding event can happen in the circuit. A transition is enabled whenever all its predecessor places are marked with at least one token. An enabled transition may *fire*. This means that the corresponding signal changes value in the circuit. The firing of an enabled transition consists of removing one token from every

predecessor place and adding one token to every successor place. An arc is *redundant* if it never determines the firing of the signal transition to which it points. Two transitions are *concurrent* if a simple path containing both of them does not exist. Otherwise they are *ordered* [2], [11].

An STG is *live* if: a) it is strongly connected, b) every single loop contains exactly one token, and c) transitions t^+ and t^- are ordered. An STG is *safe* if no arc in it is ever assigned more than one token and if no sequence of firings can bring more than one token to one arc.

The *State Graph (SG)* of an STG is a directed graph, where each node (called *state*) is in one-to-one correspondence with a marking of the STG, reachable from its initial marking [1]. Each state in the SG is assigned a binary code consisting of the signals present in the STG only. A SG has the *Complete State Coding* (CSC) property if and only if no two states have the same binary code assignment or the transitions of non-input signals, enabled in two states having the same binary code assignment, are the same. [3]. An STG has the *Unique State Coding (USC)* property if two distinct states in the SG do not have the same code.

The subsequent paragraphs are referred to deterministic STGs. A *deterministic* STG is defined as an interpreted marked graph, where transitions are interpreted as value changes on input, output or internal signals of the specified circuit [4].

III. PROPOSED TRANSFORMATIONS

In this section the proposed transformations are described. It is shown that the application of the transformations to the type of concurrency described below, results in such Karnaugh maps (K-maps) so simpler logic circuit implementations are achieved.

An STG must be live, safe and has the USC property so that a correct logic circuit implementation can be produced [11]. Then the State Graph (SG) is derived. In the SG, states are binary vectors that represent the values of the signals in the circuit, while transitions are the transitions of these signals.

To get a SG an initial state (for example all signals low) is picked and is translated to a marking in the STG. The next states are obtained by letting the marking tokens to flow free, one transition at a time. This must be repeated until the whole STG is covered. Next, the K-maps for each output and internal signal are constructed. The value of a K-map state is equal to the value of the (output or internal) signal in the next state in the SG. If there

are more than one next states and they assign both 0 and 1 to the value of the (output or internal) signal, then the value of the current K-map state is equal to the one that is different from the present output signal value. Finally, the Boolean expressions for the signals are derived.

In order to explain the transformations the general STG structure of Fig.1(a) is used. Assume a deterministic STG (that satisfies all the above mentioned properties) with a concurrency of the general form that is illustrated in Fig.1(a). Signal transition a^* is referred as the *start* transition of the concurrency, and b^* as the *end*. Signals c_i (i=1,2,...n), d_j (j=1,2,...m), e_r (r=1,2,…,k), f_s (s=1,2,…p), (n, m ≥ 1 but not both 1, and k, p ≥ 0) can be input, output or internal signals. The corresponding SG is illustrated in Fig.1(b).

If the falling signal transition c_n^- is a non-input signal transition then an arc from d_m^* to c_n^- is added. The arc from d_m^* to b^* becomes redundant and is removed (the arc appears in dashed line in Fig.1(c)). This reduces the concurrency since the signals transitions d_j^*, (j=1,2,...,m) are not concurrent with the signals transitions c_n^- and e_r^* (r=1,2,…k) anymore.

The reduced concurrency is also identified in the corresponding SG illustrated in Fig.1(d). It can be noticed, that the SG keeps the properties of the initial SG (CSC, USC), since it is essentially the same SG with some states deducted. The states that correspond to the removed concurrency are presented in dashed lines (“dashed states”). These states are deleted from the SG and become “don’t care” states in the K-maps of all non-input signals. The SG states that are presented in dotted lines (“dotted states”) correspond to K-map adjacent states that change value from 0 → 1, only in the K-map of the falling non-input signal transition to which the transformations are applied (signal transition c_n^-). These value changes are explained in more details below. Before the application of the transformations to c_n^- the values of its K-map states that correspond to the “dotted” SG states are zeros. This occurs because as it can be noted in the SG of Fig.1(b), there are two next states for each “dotted state”, and they assign both 0 (due to c_n^-) and 1 (due to the signal transitions d_j^*) to the value of the signal c_n . Since the value of c_n in the “dotted” states must be 1, the value of the corresponding states in the K-map is zero. After the application of the transformations, all the “dashed states” are removed (Fig.1(d)).

Thus, for each “dotted state” there is only one next state (due to the signal transitions d_j^*) and it assigns 1 to the value of c_n.

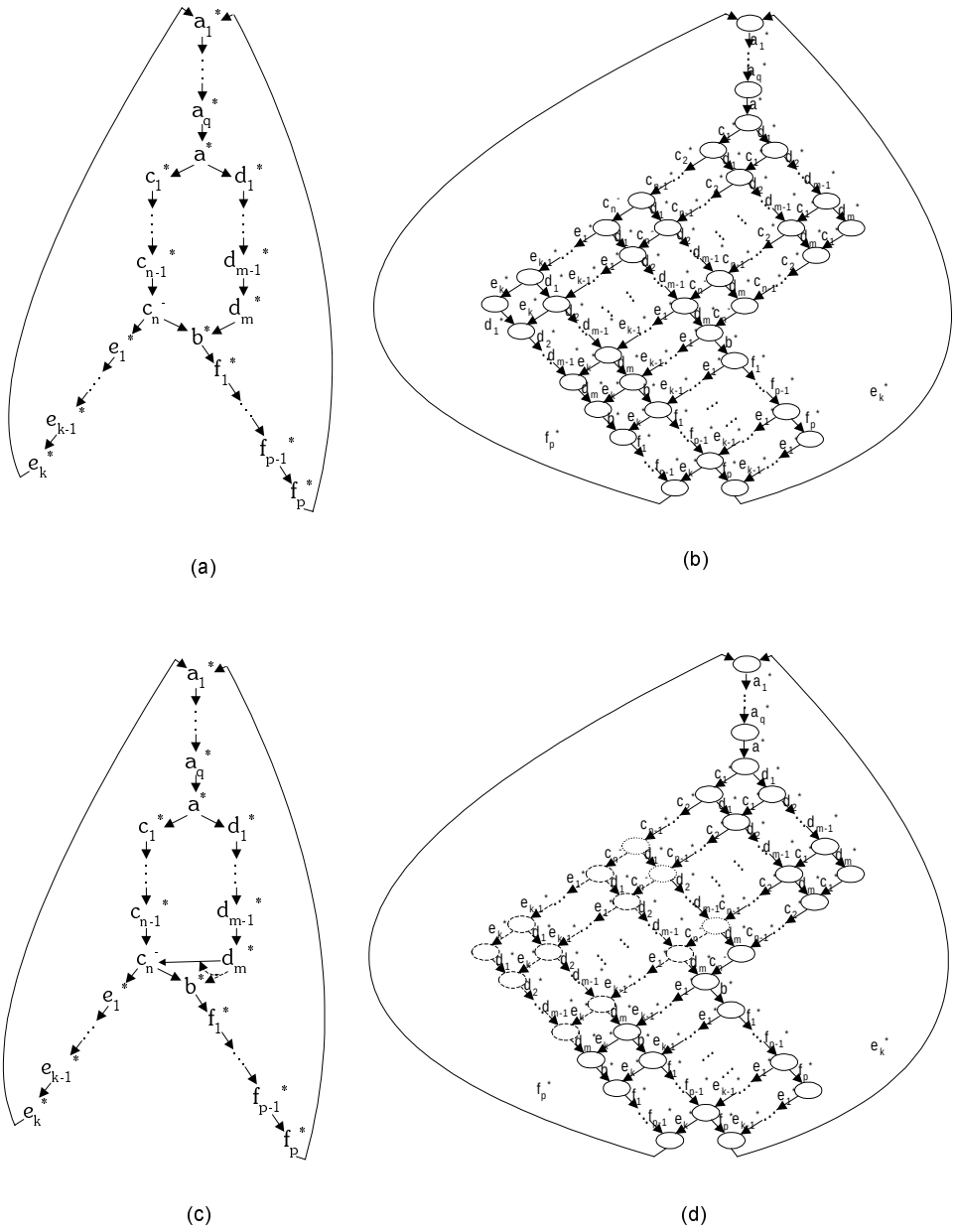


Fig.1:(a) General STG structure (b) Corresponding SG
(c) STG after transformations (d) SG after transformations

As it can be observed in the SG of the transformed STG, the resulting "don't care" states in the K-maps of all non-input signals due to the fact that the signals transitions d_j^* are not concurrent with the signal transitions e_r^* are: $m \times k$. Furthermore, the "don't care" states that result in all K-maps due to the elimination of the concurrency of c_n^- with the signal transitions d_j^* are *m*. So, the total "don't care" states that result in each K-map are: $m + (m \times k)$. In the K-map of the signal transition c_n^-, m more state value changes from $0 \rightarrow 1$ occur in the states that correspond to the SG "dotted states", as described above.

Thus, the resulting total state value changes in the K-map of the falling non-input signal transition to which the transformations are applied are: $m \times (k+2)$.

These changes result in more efficient K-maps from which simpler logic circuit implementations can be obtained, while there is no change in the functional operation of the circuit. If c_n is an input signal, the transformations cannot be applied, because the environment is not allowed to impose any restrictions on the inputs of the circuit.

IV. EXAMPLES AND RESULTS

The operation of the transformations and their impact in the area are illustrated by using an example STG, taken from [4], that is shown in Fig.2(a). The STG is live, safe and has the USC property. The input signals are R_i, A_{i1} and A_{i2}, and the output signals are R_0 and A_0. The start and the end transition of the concurrency are the signal transitions A_0^+ and A_{i2}^-, respectively. According to the general form in Fig.1(a) we have n = 4, m = 2 and k = 0.

The signal R_0 is an output signal, so the transformations can be applied to R_0^-, as it is illustrated in Fig.2(b). The SGs before and after the application of the method are illustrated in Fig.2(c) and Fig.2(d), respectively. The representation of the initial concurrency of signal transition R_0^- with R_i^- and A_0^- in Fig.2(c), includes the SG states with binary codes: 10011 and 00011. In the transformed SG (Fig.2(d)), these states appear in dashed lines ("dashed states"). They are removed due to fact that R_0^- is now ordered with R_i^- and A_0^-. Furthermore, they become "don't care" states (represented as *X*) in the K-maps of both R_0 and A_0. The dashed arrows labeled R_0^-, R_i^-, A_0^- are also removed. The states with binary codes 11011 and 01011 which appear in dotted lines ("dotted states") correspond to the adjacent states in the K - map of signal R_0 that change value from $0 \rightarrow 1$, since the next state value of R_0 in both of them is now 1.

All the afore-mentioned changes can be observed in the K-maps of signals R_0 and A_0 for the transformed STG (Fig.2(g),(h)) as compared to the corresponding K-maps for the original STG (Fig.2(e),(f)). It can be noted that the resulting "don't care" states in each K-map are: $m + (m \times k) = 2$, and the resulting total state value changes in the K-map of R_0 are: $m \times (k+2) = 4$.

The logic circuit implementations before and after the transformations are given in Fig.2(i) and Fig.2(j), respectively. In the derived circuits, C-elements (with the additional logic where needed) have been used instead of

SR latches, in order to avoid the SR latches limitations [5]. The circuit after the transformations is smaller in area (3 gates less) than the one before transformations.

In Table 1 measurements for three typical asynchronous circuits derived by applying the proposed method are shown. The area is measured in number of logic gates. circ1 is the STG which is already analyzed. 4ph is a typical 4 phase interface. ex2 is an STG with k=2. The examples of the 4ph and ex2 are illustrated in Fig.3 and Fig.4, respectively.

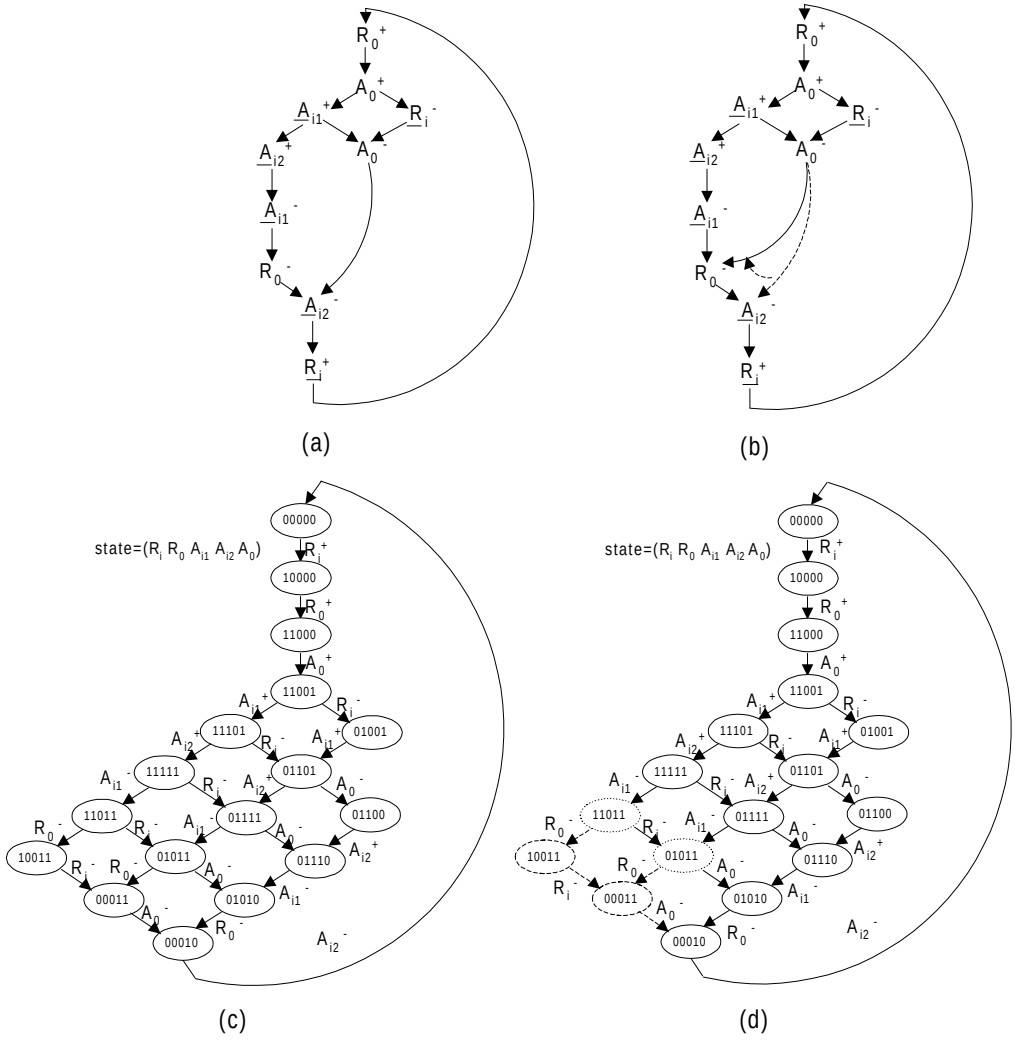


Fig.2: (a) STG circ1 example before transformations (b) STG after transformations
(c) SG before transformations (d) SG after transformations

R_iR_0 \ $A_{i1}A_{i2}A_0$	000	001	011	010	110	111	101	100
00	0	X	0	0	X	X	X	X
01	X	1	0	0	1	1	1	1
11	1	1	0	X	X	1	1	X
10	1	X	0	X	X	X	X	X

(e)

R_iR_0 \ $A_{i1}A_{i2}A_0$	000	001	011	010	110	111	101	100
00	0	X	0	0	X	X	X	X
01	X	1	0	0	0	0	0	0
11	1	1	1	X	X	1	1	X
10	0	X	1	X	X	X	X	X

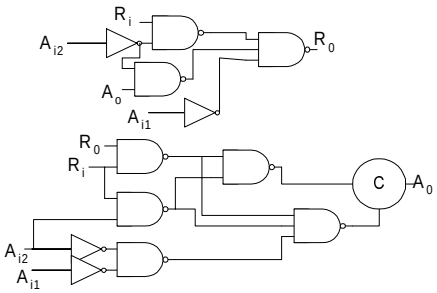
(f)

R_iR_0 \ $A_{i1}A_{i2}A_0$	000	001	011	010	110	111	101	100
00	0	X	X	0	X	X	X	X
01	X	1	1	0	1	1	1	1
11	1	1	1	X	X	1	1	X
10	1	X	X	X	X	X	X	X

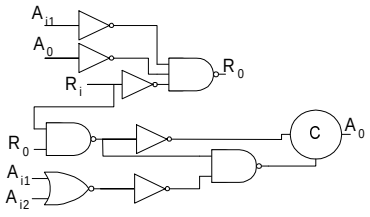
(g)

R_iR_0 \ $A_{i1}A_{i2}A_0$	000	001	011	010	110	111	101	100
00	0	X	X	0	X	X	X	X
01	X	1	0	0	0	0	0	0
11	1	1	1	X	X	1	1	X
10	0	X	X	X	X	X	X	X

(h)



(i)



(j)

Fig.2: (e) K-map of R_0 before (f) K-map of A_0 before
(g) K-map of R_0 after (h) K-map of A_0 after
(i) Logic diagram for STG circ1 before (j) Logic diagram after

Circuit Name	Area (in No. of Gates)	
	before transformations	after transformations
circ1	13	10
4ph	9	6
ex2	13	11

Table 1: Area measurements for circuits which produced by applying the proposed transformations in three typical STGs

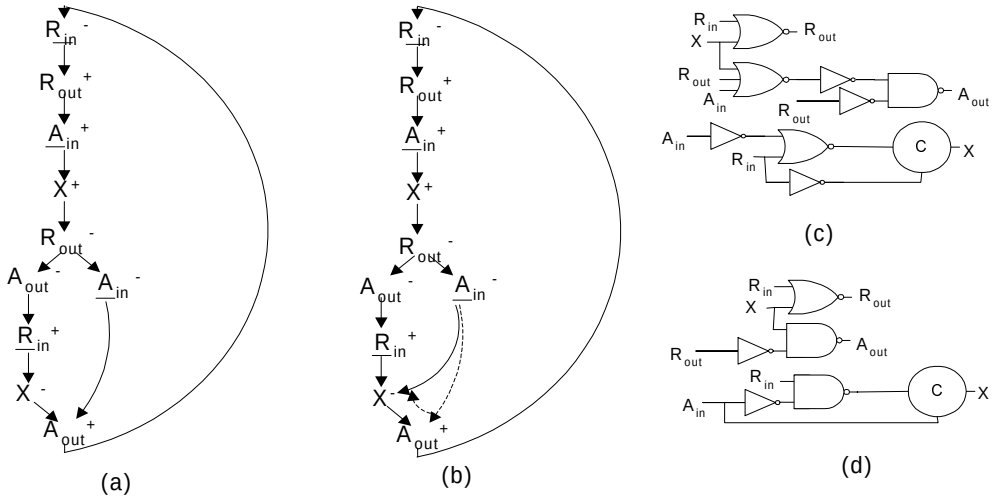


Fig.3: (a) STG 4ph example before transformations (b) STG after transformations
(c) Logic diagram before (d) Logic Diagram after

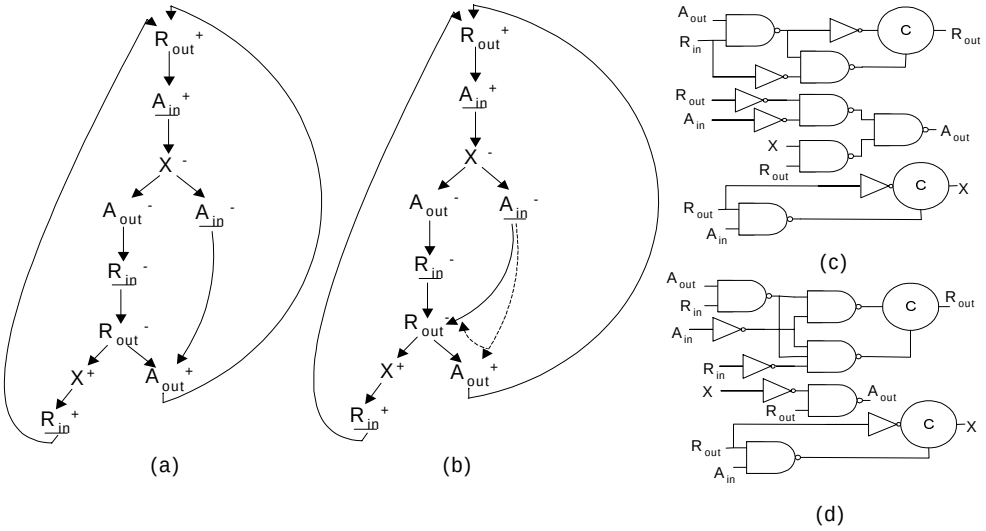


Fig.4: (a) STG ex2 example before transformations (b) STG after transformations
(c) Logic diagram before (d) Logic diagram after

V. CONCLUSIONS

In this paper transformations for Signal Transition Graphs are described which achieve reduction in the required area. The transformations can be easily integrated in existent synthesis tools. The changes of the proposed STGs transformations in the resulting State Graphs and Karnaugh Maps are explained. Results for three typical examples are presented and prove the significant area reduction that is achieved.

REFERENCES

[1] L. Lavagno, K. Keutzer and L. Sangiovanni-Vincentelli, “Synthesis of Hazard -Free Asynchronous Circuits with Bounded Wire Delays”, IEEE Trans. CAD of Integrated Circuits and Systems, vol. 14, no. 1, pp. 61-86, January 1995.

[2] T. Chu, “Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications”, in Proc. of the International Conference on Computer Design (ICCD), pp. 220-223, 1987.

[3] J. Gu, R. Puri, “Asynchronous Circuit Synthesis with Boolean Satisfiability”, IEEE Trans. on CAD Integrated Circuits and Systems, vol. 14, no. 8, pp. 961-973, August 1995.

[4] S.T. Jung, and C. S. Jhon, “Direct Synthesis of Efficient Speed-Independent Circuits from Deterministic Signal Transition Graphs”, in Proc. of the International Symposium on Circuits And Systems (ISCAS), pp. 307-310, 1994.

[5] C.W. Moon, P. R. Stephan and R.K. Brayton, “Synthesis of Hazard-free Asynchronous Circuits from Graphical Specifications”, in Proc. of the International Conference on Computer Aided Design (ICCAD), pp. 322-325, 1991.

[6] T.H.-Y. Meng, R.W. Brodersen and D.G. Messerschmitt, “Automatic Synthesis of Asynchronous Circuits from High-Level Specifications”, IEEE Trans. on CAD, vol. 8, no. 11, pp. 1185-1205, November 1989.

[7] L. Lavagno, K. Keutzer and A. Sangiovanni-Vincentelli, “Algorithms for synthesis of hazard-free asynchronous circuits”, in Proc. of the Design Automation Conference, pp. 302-308, 1991.

[8] P. Vanbekberken, G. Goossens, F. Catthoor and H.J. De Man, “Optimized Synthesis of Control Circuits from Graph-Theoretic Specifications”, IEEE Trans. on CAD, vol. 11, no. 11, pp. 1426-1438, November 1992

[9] T. Murata, “Petri Nets: Properties, Analysis and Applications”, in Proc. of the IEEE, vol.77, no. 4, pp. 541-580, April 1989.

[10] P.A. Beerel, C. T. Hsieh and S. Wadekar, “Estimation of Energy Consumption in Speed-Independent Control Circuits”, IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 15, no. 6, pp. 672-680, June 1996.

[11] M.-L. Yu and P.A. Subrahmanyam, “A New Approach for Checking the Unique State Coding Property of Signal Transition Graphs”, in Proc. of the European Design Automation Conference (EURO-DAC), pp. 312-321, 1992.