DK 621.3.019:061.3

PROCEEDINGS OF THE 1988 INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

August 15-19, 1988

Vol. I Architecture

Fayé A. Briggs, Editor

Sponsored by

Department of Electrical Engineering
PENN STATE UNIVERSITY
University Park, Pennsylvania

THE PENNSYLVANIA STATE UNIVERSITY PRESS
UNIVERSITY PARK AND LONDON

TABLE OF CONTENTS

SIGNAL GRAPHS: A MODEL FOR DESIGNING CONCURRENT LOGIC

A.Yu.Kondratyev, L.Ya.Rosenblum, A.V.Yakovlev

Computing Science Department
Leningrad Electrical Engineering Institute
Leningrad 197022 USSR

Abstract -- Asynchronous digital circuits exhibit a high degree of concurrency. Self-timed implementation is the most appropriate design discipline for them. We examine the signal graphs that are subject to formal treatment and mechanical translation to delay-insensitive circuits. An example of designing a piece of logic for typical interface adapter effectively illustrates the approach and sheds light on future work.

1. Introduction

Modern technologies allow to build VLSI circuits whose internal behavior exhibits a high degree of parallelism. To operate correctly under the presence of such undesired phenomena as electronic metastability, signal skews due to higher values of wire vs gate delay ratios, parametric instabilities of gates etc. these circuits are designed using self-timed, or delay-insensitive fashion [1,2]. The most widely cited examples of concurrent hardware are regular structures like pipeline and wavefront arrays which are easily decomposed in sequential,parallel or recursive way. On the other hand such objects as asynchronous interface adapters which are a lot less regular but can be equally concurrent are far from being attempted at a formal treatment as they have been the privelege of engineers using normally timing diagrams or flow charts.

The ultimate goal of our research is to mechanize the design process to such a degree when it is comfortably fitted in a CAD environment for developing distributed systems, e.g. for translating a physical layer protocol specification into a collection of self-timed modules. This paper demonstrates the technique of using a formal model of concurrency for constructing basic units of interfacing logic. This technique accomodates a step-wise design procedure involving such steps like architectural decomposition, functional specification of components, their behavioral signalling expansion, and its validation with respect to correctness and completeness notions, and finally Boolean function derivation.

2. Modelling concurrency in logic

A self-timed system is often regarded as a collection of self-timed modules that communicate via asynchronous protocols [1]. It does not require a global clock. All system level events are ordered in time by the causal relations between the modules actions. The order as it has been established by the designer must further be preserved in a final circuit thereby guaranteeing the correct operation independently of element and wire delays.

The evolution of logic design methods shows that the Huffman state machine model is no longer an adequate model for asynchronous logic since it can not deal with "granulated" concurrency in VLSI. The existing formal models for self-timed VLSI systems can be split into four groups:
(i) graphical notations, state or event oriented, like Petri nets, transition diagrams, parallel flow charts etc.;
(ii) symbolic notations, like traces or path expressions;
(iii) models based on high level programming languages, e.g. Ada-like notation;
(iv) combined models.

The study of these formalisms shows that the usefulness of a model for the self-timed circuit design depends on a large number of various issues. For example, it is affected by the structure type (regular vs non-regular, or data-flow vs control-flow), the degree or granularity of parallelism and data dependence, the necessity of abstract data typing, the depth of delay-independence (with respect to transistor, gate or component level).

Our formalism, a signal graph based on a subclass of Petri nets, is an effective substitute for widely used timing diagrams because it can be analyzed in a mathematically sound manner and mechanically translated to Boolean functions implementation.

3. Signal graphs: properties and analysis

Signal graphs are very attractive formal model for analyzing behavioral specifications of both signalling protocols and corresponding interface logic. They represent a more narrow class of processes than that that can be generally defined by, say, Petri nets. This is concerned with their inability to define alternatives in processes. However, when we need to define a highly concurrent behavior they provide the succinct description and what is more important, the polynomially complex analysis.

We presume some knowledge of Petri nets and their subclasses, particularly marked graphs. Marked graph (MG) generates distributive marking diagram (MD) 3 . MD is an oriented graph whose vertices are reachable markings and arcs are labeled with firing transitions. The term "distributivity" is related to the lattice which can be defined on a set of vectors of transition firing numbers with respect to a given initial marking.

In order to define a signal graph a set of binary variables (signals) $Z = \{z_1, z_2, \ldots, z_n\}$ is introduced. We denote transitions of signal z_i: from 0 to 1 by $+z_i$ and from 1 to 0 by $-z_i$.

Signal graph (SG) is defined as an MG in which vertices are labeled with signal transitions (changes) of the form dz_i where $d \in \{+,-\}$.

We call a labeling function conflict-free if for each reachable marking and variable z_i there is at most one enabled vertex labeled with dz_i . SG with a conflict-free labeling is called coherent. The coherence is not sufficient for the specification to be correct because despite all the changes for each z_i are linear-ordered they may be unmatched with respect to their signs.

We call a labeling function sign-balanced if for each sequence of signal transitions with respect to initial marking between any two transitions of the same sign there exists at least one transition of the other sign. SG with a sign-balanced labeling is called consistent. The consistency implies the necessary level of correctness of a specification given by SG that is expressed in the following statement.

Statement 1. A consistent SG generates a state transition diagram.

A state transition diagram (STD) is an oriented graph whose vertices are labeled with full states of a specified circuit, i.e. they are binary n-tuples of values of z_i , and arcs are labeled with corresponding changes dz_i. The values that can change between a given state and another one connected to each other by an arc are marked with *-token. A variable whose value in n-tuple is marked with * is called excited in a given state. In this paper we omit the description of algorithms of converting a consistent SG to STD and vice versa. We only hint that such a conversion may use the ordinary procedure of building an MD by the depth-first search where each marking in MD relates to a corresponding state in STD.

A consistent SG may however generate a STD with multiple states, i.e. the states which are labeled with equal n-tuples of signal values. Such an STD is called contradictory. Informally, the contradiction of this kind means that the system is under-specified and some components are still hidden from the designer's eye. For example, when SG defines an interface protocol these components may be interpreted as an internal memory of controller.

We further incorporate a higher level of correctness into the hierarchy of SG classes by the notion of a normal SG which guarantees the completeness of a specification. An SG is called normal if it is consistent and for each allowed sequence of markings it has no proper subset of variables $Z' \subset Z$ which can proceed through the full cycle of their values while other variables (from $Z \setminus Z'$) stay unchanged. An STD of a normal SG is non-contradictory and distributive [3].

It is suitable to check the consistency and normalicy using the relations of precedence and concurrency built on the set of signal transitions. The formalization of these relations requires the introduction of a concept of a history, or so-called unfolding, which is an infinite and acyclic object generated by an SG. Each occurrence of a transition in an SG yields a unique vertex in the unfolding. This technique due to the lack of space can not be fully described here though we mention that the unfolding can be floored to its first two periods and the above relations can thus be computed on a finite object. The algorithm of checking consistency has the complexity of $O(n^3)$ where n is the number of vertices in the original SG.

In order to establish whether a consistent SG is normal we use a special formal concept - operational coupledness. We define a coupled relation on a set of variables Z. This relation has the following hierarchy: directly strongly coupled, strongly coupled, weakly coupled of rank r , $r \geq 0$, and coupled. The coupled relation partitions the set Z into the disjoint classes. Omiting here formal definitions and proofs which can be found elsewhere [4] we only state the following.

Statement 2. A consistent SG is normal iff all its variables belong to single coupledness class.

The complexity of an algorithm for normalicy check is of $O(n^4)$.

The main advantage of our checking techniques stems from the fact that they do not require to convert an SG to

MD or STD - a step having exponential complexity with respect to the power of Z.

4. An example of self-timed logic design

In the above section we have sketched how we can check the normalicy of an SG which is a sufficient condition for the existence of a distributive STD and hence of a delay-insensitive circuit [2] . The circuit can be derived from the normal SG by means of obtaining the Boolean functions (BFs) for variables z_i of set Z using a truth table (TT) which can be built from the STD corresponding to the SG. However the chain SG-STD-TT-BFs involves exponentially complex steps. Therefore we look for an alternative technique for the direct (but semantics preserving) conversion of the SG to the system of BFs. Such a bridling of the design complexity is concerned, first of all, with laying out some restrictions upon the complexity of the coupledness hierarchy.

In this paper we are far from being ambitious to show how the problem of obtaining the general way of deriving functions directly from an SG can be solved. We rather illustrate our design approach with an instructive example of designing a piece of interface logic.

FIFO buffers are typically incorporated in interfacing adapters as they help to keep the performance of the whole distributed system at its highest communication rate. The original specification of a one-value FIFO cell was inspired by [5] .

Let the FIFO cell consist of two subcells: the data cell (DC) and the control cell (CC) as shown in Fig.1.

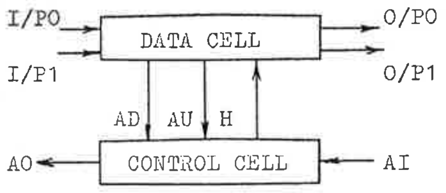

Figure 1. The structure of FIFO cell

The meaning of the signals is as follows. I/P0 and I/P1 are data inputs, and O/P0 and O/P1 are data outputs. Both use the two-rail coding discipline [1] where for "zero" and "one" values the combinations 10 and 01 are respectively used on the above pairs, and the all-zero spacer (00) is used for representing the "data undefined" value. AO and AI are the acknowledgement signals: AO is generated by the cell and AI is produced by the environment. AD is "All defined" indication signal, AU is "All undefined" indication signal, and H is "Hold" command signal. AD and AU are both used to detect the state of the inputs (if I/P0 = I/P1 = 0 then AD = 0, AU = 1, and if I/P0 ≠ I/P1 then AD = 1, AU = 0). H directs the DC to latch the incoming value. All AD, AU, and H wires run the width of the buffer.

Fig.2 shows the SG specification of the CC operation. Analyzing this SG we can establish that it is consistent: each variable has all its transitions ordered within one synchrocycle (a cycle containing exactly one token). However the SG is not normal. The coupled relation partitions the set Z = {AI,AO,AD,H,AU} into two disjoint classes: K1 = {AI,H} and K2 = {AD,AU, AO} . It can be shown that adding only one extra variable to the specification while preserving the established order of signal changes for variables in Z will not suffice for making all variables coupled. After adding two variables d1 and d2 we obtain the resulting SG shown in Fig.3 which is normal.

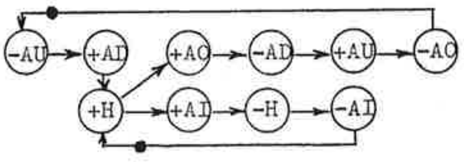

Figure 2. An original SG specification of the control cell operation

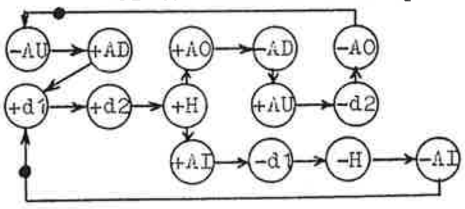

Figure 3. A normal SG obtained after adding extra variables

From this SG we derive BFs in the following form:

$$z = Sz + \overline{Rz} \cdot z,$$

where Sz is the set function and Rz is the reset function. Both Sz and Rz are independent of z. We also demand that the invariant $Sz \cdot Rz = 0$ holds in order to avoid conflicts between transitions which may lead to undesired races in a circuit.

In order to derive Sz and Rz we search through the SG for immediate predecessors of the transition +z for including them into the essential Sz-term, and those of transition -z for Rz-term. If these predecessors correspond to the variables that are strongly coupled with z we proceed further to the orthogonalization step. If some of the variables whose transition is a predecessor for a given transition dz is weakly coupled (of any rank $r \geqslant 0$) with z then there is a so-called overtaking of the essential term by some other term which must be added to corresponding set or reset function. For example, when deriving function S_{AO} the essential term is $H \cdot d2$ but before AO changes from 0 to 1 H may begin to change from 1 to 0 (in parallel with AO changing), and hence we must cure the overtaking by an additional term which will involve a variable that is strongly coupled with H, i.e. the term $\overline{d1} \cdot d2$. Using d2 in both terms for S_{AO} helps us also to eliminate the inclusion of the term for R_{AO} which is simply $\overline{d2}$ because d2 is immediately strongly coupled with AO. Thus we obtain a BF for AO which is non-selfdependent, i.e. free of feedback

$$AO = (H + \overline{d1}) \cdot d2 + d2 \cdot AO = (H + \overline{d1})d2.$$

One of the important issues in deriving Sz and Rz is their mutual orthogonalization, i.e. providing that $Sz \cdot Rz = 0$ is satisfied. This can be done by strengthening their terms with common variables. For example, when we obtained S_{AO} we had d2 as such a variable. Another example is the function for H whose $S_H = d1 \cdot d2$ is strengthened by d1 because $R_H = \overline{d1}$.

Finally, the above technique yields the following system of BFs:

$$\begin{aligned} d1 &= AD \cdot \overline{AI} \cdot \overline{d2} + \overline{AI} \cdot d1 \\ d2 &= \overline{AU} \cdot \overline{H} \cdot d1 + \overline{AU} \cdot d2 \\ H &= d1 \cdot d2 + d1 \cdot H \\ AO &= \overline{d1} \cdot d2 + d2 \cdot H \end{aligned}$$

This system is easily implemented with four AND-OR-NOT gates and six inverters (four of them produce d1,d2,H,AO, and the other two complement AI and AU, however the latter can obviously be eliminated at the transistor level by using the inhibit inputs of the first two gates).

The circuit is delay-insensitive with respect to delays in gates and inverters as well as in those wires which are not the feedback connections. The feedback delays are presumed negligible as the corresponding elements are accommodated within equichronic regions.

5. Conclusion

The main characteristic of the above approach comparing it with those given elsewhere[6,7,8]is that it provides the technique for effective managing with concurrency at the logic level using the formal model which is quite simple for comprehension for a wide audience of hardware designers used to timing diagrams, and at the same time powerful enough to be formally analyzed with respect to correctness and completeness by means of such key concepts as normalicy and coupledness. This facilitates some constructive ways to the correction of specifications while preserving the original semantics of signal change ordering. The method has been tested on a large number of difficult examples including designing asynchronous control logic for interfaces (Unibus, Futurebus, token ring etc.) and FIFO buffers of various architectures.

The proposed technique obviously needs further research efforts both in theory as, for example, in establishing restrictions on coupledness classes to find out how they affect the BF derivation rules outlined above, and in practical aspects through developing the software for such a mechanized translation to be a versatile interactive design environment. Some pieces of such an environment are in progress now.

References

[1] C.L. Seitz, System Timing, Chapter 7 in: Introduction to VLSI Systems, C.Mead and L.Conway, Addison-Wesley, (1980), 400 pp.

[2] V.I.Varshavsky, Hardware Support of Parallel Asynchronous Processes, Digital Syst. Lab., Helsinki University of Technology, Series A, No.2, (Sept. 1987), 236 pp.

[3] L.Ya. Rosenblum, A.V. Yakovlev, Signal Graphs: from Self-Timed to Timed Ones, Intern. Workshop on Timed Petri Nets, Torino, Italy,(1985), pp.199-207.

[4] A.V. Yakovlev, Design and Implementation of Asynchronous Interface Protocols, PhD Thesis, (1982).

[5] E.E. Barton, Non-metric Design Methodology for VLSI, in: VLSI-81, Academic Press, London, (1981), pp. 25-34.

[6] A.J. Martin, Compiling Communicating Processes into Delay-Insensitive VLSI Circuits, Distributed Computing, Vol. 1, No. 4 (1986), pp. 205-225.

[7] T.-A. Chu, On the Models for Designing VLSI Asynchronous Digital Systems, Integration, the VLSI journal, Vol.4 (1986), pp. 99-113.

[8] P.F. Lister, A.M. Alhelvani, Design Methodology for Self-Timed Systems, Proc. IEE, Pt. E, Vol. 132, No.1 (1985), pp 25-32.