MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Computation Structures Group Memo 71

The Description and Realization of Digital Systems

by

S. S. Patil and J. B. Dennis

A paper presented at the Sixth Annual IEEE Computer Society International Conference, San Francisco, California, Sept. 12-14, 1972. Published in the Digest of Papers 1972: Innovative Architecture, IEEE, New York, N. Y., pp 223-226.

October 1972

This work was supported by Project MAC, an MIT research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract N00014-70-A-0362-0001.

THE DESCRIPTION AND REALIZATION OF DIGITAL SYSTEMS

S. S. Patil and J. B. Dennis
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

With the growth in size and complexity of computer systems the problems of system description and correctness of system realization have become increasingly severe. Inadequacies of conventional means of system description make the unambiguous statement and communication of the functionally essential facts of a design very difficult, and make the correctness of a system built from current descriptions impossible to guarantee. Our research on the description and realization of digital systems concerns the development of methods of system description that have a sound formal foundation, are applicable to practical digital systems, and for which rules of implementation can be given with a guarantee that the resulting hardware faithfully realizes the behavior specified by the description.

DIGITAL SYSTEMS

A digital system often has the form of two interconnected parts: a data flow structure and a control structure. The data flow structure is made of registers, operators that take values from certain registers and place transformed values in other registers, and deciders that permit values held in registers to influence the future sequence of actions. The control structure generates signals that direct actions by the operators of the data flow structure according to the results of tests made by the deciders.

In an asynchronous realization of a digital system, signals are sent from the data flow structure to the control structure to indicate that action by an operator has been completed or that a decision by a decider has been made. These acknowledge signals are used directly in the control structure to generate ready signals that initiate actions by any operators and deciders which are now eligible for execution. In this way, actions in the data flow structure are not delayed beyond completion of those actions upon which correct functioning of the digital system depends.

If an asynchronous control structure correctly implements the description of a digital system regardless of the size of delays associated with its components, the control structure is called a speed independent interconnection of those components.

In this paper, we present and illustrate the methods we have developed for the description and realization of speed independent control structures for digital systems. We have also been exploring the feasibility of realizing complete digital systems (including the registers and operators of the data flow structure) as speed independent interconnections of simple circuits. For example, a study of the design of a speed independent counter-with-readout is included in [5].

DATA FLOW STRUCTURE

Figure 1 shows the notation used to represent the data flow structure of a digital system. The components are registers (a), operators (b) which receive values from certain registers and place results in one or more other registers, and deciders (c) that make decisions based on values held by certain registers. Registers, operators and deciders are interconnected by data pathways called data links (d).

A separate control link (e) from the control structure terminates on each operator. The control link consists of two wires: A signal on the ready wire tells the operator that values are present in its input registers ready for use by the operator. The operator sends a signal to the control structure by means of the acknowledge wire when the results of its action have been produced and placed in its output registers. The control structure communicates with the deciders of the data flow structure by decision links (f). A signal on the ready wire tells the decider that its input registers hold values to be tested. The acknowledge signal is sent to the control structure over one of several acknowledge wires according to the outcome of the test.

PETRI NETS

As illustrated in Figure 2, a Petri net [8, 2] is a directed graph having two kinds of nodes: places drawn as circles, and transitions drawn as bars. Each arc connects a place to a transition or a transition to a place. In the former case the place is an input place -- in the latter case an output place -- of the transition.

A marking of a Petri net is an assignment of some number of tokens to each place of the net. The net progresses from one marking to another by the firing of transitions. The firing rule is:

1. A transition is said to be enabled if each of its input places holds at least one token.
2. Any enabled transition may be selected to be fired.

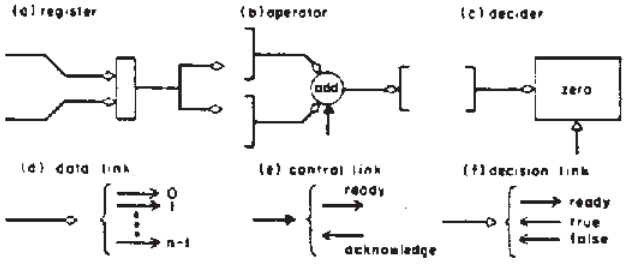

Figure 1. Components of a data flow structure.

This work was supported by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract N00014-70-A-0362-0001.

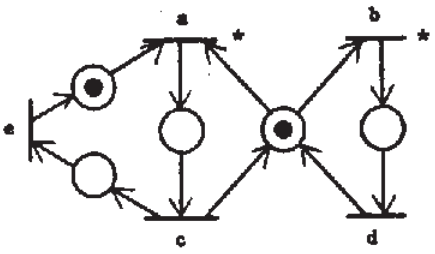

Figure 2. A Petri net.

3. A transition is fired by removing one token from each of its input places and then adding one token to each of its output places.

In Figure 2, transitions a and b (indicated by asterisks) are enabled for the marking shown.

A Petri net is said to be safe for some marking if applications of the firing rule can never lead to a marking in which some place has more than one token. A net is live for some marking if, no matter what marking has been reached, it is possible to fire any transition of the net by some firing sequence. The net in Figure 2 is both safe and live, as will be true for all Petri nets we consider.

If, for some marking, application of the firing rule to a Petri net never results in a transition ceasing to be enabled except by being fired, the net is persistent for that marking. The marked Petri net of Figure 2 is not persistent because firing transition a yields a marking in which transition b is no longer enabled.

EXAMPLES

Control structures for digital systems may be represented by Petri nets in which certain transitions are labelled to refer to operators of a data flow structure, and certain groups of transitions are labelled to refer to the outcomes of deciders. Figure 3 shows an interconnection of operators that act in pipeline fashion. The firing of a labelled transition involves four steps:

1. Remove tokens from input places.
2. Send a ready signal to the operator.
3. Wait for the acknowledge signal.
4. Put tokens in output places.

In Figure 3b the pair of places labelled A represents a control link used to coordinate the input of values to the pipeline system via data link A of the data flow structure. Arrival of a token in the upper place represents a ready signal indicating that a new value has been presented on data link A. Arrival of a token in the lower place represents an acknowledge signal by which the pipeline system indicates it has acted on the last value presented and a new value may be placed on data link A. The pairs of places labelled B and C coordinate transmission of data values from the pipeline system in a similar way.

A data flow structure for a simple conventional computer is shown in Figure 4a. Operators with the letter I written inside are identity operators that transfer values between registers. Register P is

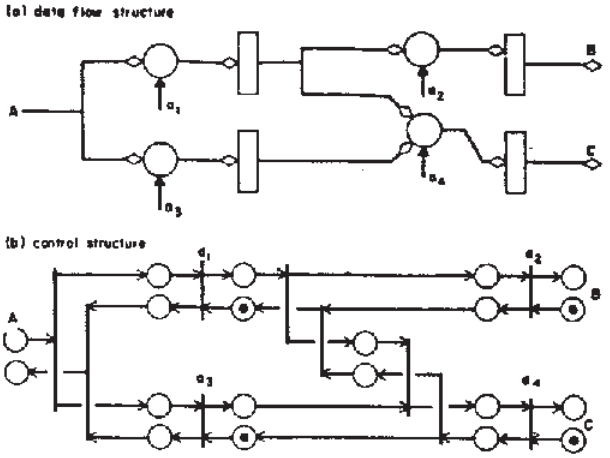

Figure 3. Petri net representation of a control structure for a pipeline system.

the program counter and I is the instruction register. The operators "fetch" and "read" retrieve words from a memory (not shown) according to the address in P or I, respectively, and put them in the instruction register I or the operand register M. The operator "write" stores the word in the accumulator A in the memory at the address in I. We suppose that the instructions of this machine fall into three classes: operate, store, and conditional. Instructions of class oper read one operand from memory and cause the word in A to be changed in some way. Instructions of class store cause the word in A to be written into memory. Instructions of class cond apply some test to the value in A and cause transfer of control according to the instruction address if the condition is met. The control structure of the machine is specified as a Petri net in Figure 4b. Decisions are modeled in the Petri net by the firing of one of a group of transitions that share an input place.

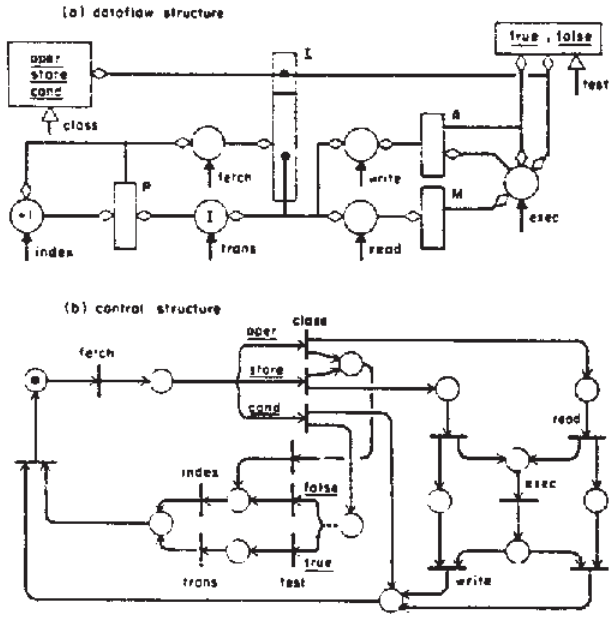

Figure 4. Data flow and control structures for a simple computer.

SPEED INDEPENDENT CIRCUITS

Speed independent switching circuits have been studied extensively by Muller and his colleagues [9, 10, 12]. Here we are concerned with circuits formed by interconnecting copies of the switching elements defined in Figure 5. The behavior of each element type is specified by a form of Karnaugh map called a transition diagram. Each cell of a transition diagram represents a total state of the element -- a particular combination of values for its input nodes and output node. For certain total states an element is active, and the corresponding cells in the map are marked with asterisks. An element that is active may fire, changing the value of its output node from 0 to 1 or from 1 to 0.

A circuit constructed from components which may be either the basic switching elements of Figure 5, or certain modules constructed by interconnecting basic elements, may be a type 1 or a type 2 speed independent implementation of a system. If the firing sequences of the circuit carry out exactly the behavior specified by a system description, the circuit is a type 2 implementation of the system. If, moreover, the behavior of the circuit remains correct when identity elements are inserted in every connection between components of the circuit, the interconnection of components (modules or switching elements) is a type 1 implementation of the system.

A type 1 implementation ensures correct operation of a hardware system realization in spite of arbitrary delays in its components or in the interconnecting wires. A type 2 implementation ensures correct operation in spite of delays in switching element operation but may have faulty behavior if arbitrary delays occur in the wires. It is attractive to implement systems as type 1 interconnections of standard modules which are type 2 interconnections of elementary circuits.

IMPLEMENTATION OF CONTROL STRUCTURES

In this section we present rules for translating Petri net descriptions into speed independent circuits constructed as type 1 interconnections of a fixed set of module types. The rules presented here are for a subclass of Petri nets known as simple Petri nets. This subclass has proven adequate for representing all control structures we have studied in our research. This class and two successively more restrictive subclasses of Petri nets are defined below.

1. A place is a shared input place of a transition if it is an input place of that transition and also an input place of some other transition.
2. In a simple Petri net each transition has at most one shared input place.
3. In a free-choice net, a transition with a shared input place has only one input place.
4. In a marked graph, no transition has any shared input places.

We also require that the Petri nets be live and safe. Liveness ensures that no part of a control structure described can reach a condition such that further activity is impossible. Safety ensures that the number of distinct markings of a net is finite.

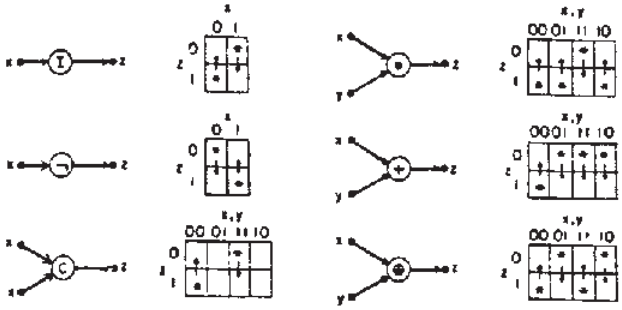

Figure 5. Transition diagrams for elementary circuits.

In a labelled Petri net the group of transitions associated with a decider is known as a free choice since some agent outside the net is free to make an independent choice of which transition takes the token from the shared input place. The transitions in a free choice are always nonpersistent, since they are always enabled and disabled together. We assume that the free choices associated with deciders include all nonpersistent transitions of the nets under consideration. This excludes situations in which happenstance can affect the order in which events are forced to occur. Such control structures require use of arbiters [11].

The behavior of a circuit is related to the behavior of the Petri net it implements by making a correspondence between events in the Petri net and events in the circuit. There are two correspondences that may be made:

1. simple signalling: Each transition ($0 \rightarrow 1$ or $1 \rightarrow 0$) on a wire of the circuit corresponds to an event in the net.
2. reset signalling: Each transition $0 \rightarrow 1$ on a wire corresponds to an event in the net; transitions $1 \rightarrow 0$ are called resets.

Reset signalling was ued in the control circuits of the Illiac II computer [12], and was used by Muller in his speed independent bit-processing modules [10]. Simple signalling has been studied by Patil [4].

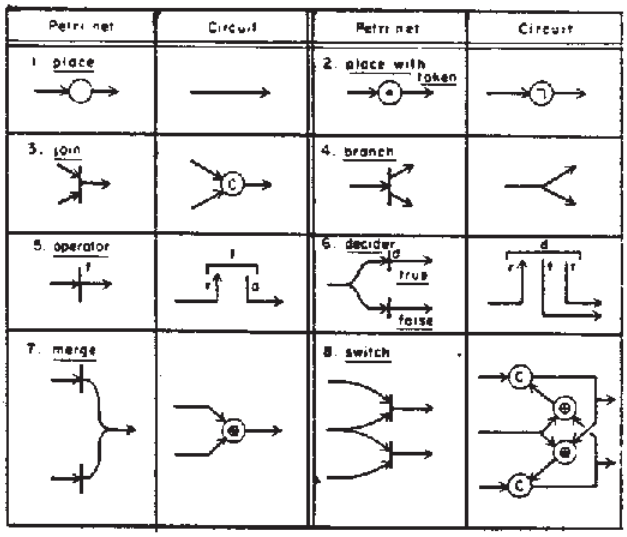

Figure 6. Implementation of simple Petri nets using simple signalling.

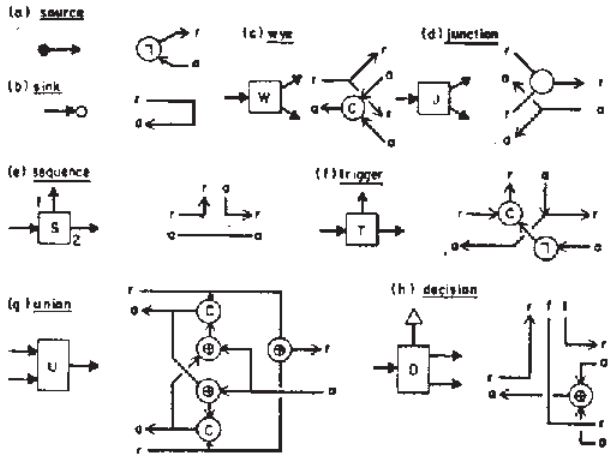

Figure 7. Control modules using ready/ acknowledge signalling.

Figure 6 gives the rules for implementing labelled simple Petri nets that are live, safe, and persistent using simple signalling. The second column specifies the circuit module to be substituted for the Petri net fragment in the first column. The connections between modules are always type 1 connections. Connections inside the modules are also type 1 connections except for one situation: The switch module is not a type 2 circuit because arbitrary delay in the operation of the XOR-elements can produce faulty behavior.

The Petri net in Figure 3b is in fact a marked graph and requires just the first five rules in the table. When the rules are applied to this net the resulting circuit divides naturally into fragments that are members of the set of basic control modules defined in Figure 7. Instead of giving the circuit directly, we specify the circuit in Figure 8 as an interconnection of control modules. This set of control modules was introduced by Dennis and Patil [4] and has also been studied by Bruno and Altman [1]. The eight control modules are sufficient to implement any labelled Petri net of the class to which the rules of Figure 6 apply.

Applying the rules of Figure 6 to the control structure of the simple computer shown in Figure 4 gives the circuit shown in Figure 9. Although this computer is trivially simple, its control structure includes examples of most of the control problems (other than those requiring arbitration) encountered even in very powerful processing units. In particular, we have published [3] Petri net descriptions of the control structures of a hypothetical machine that illustrates the principles of operation of the CDC 6600 central processor, and realizations of these control structures using only the control modules of Figure 7 and several arbiter modules.

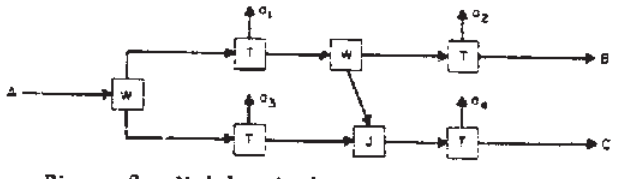

Figure 8. Modular implementation of a pipeline control structure.

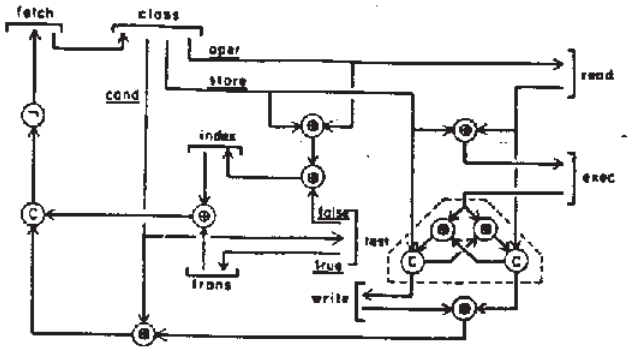

Figure 9. Implementation of the control structure for a simple computer.

REFERENCES

1. Bruno, J. and S. Altman. Asynchronous control networks. IEEE Conference Record of 1969 Tenth Annual Symposium on Switching and Automata Theory, October 1969, pp 61-73.
2. Commoner, F., A. W. Holt, S. Even and A. Pnueli. Marked directed graphs. J. of Computer and System Sciences, Vol. 5 (1971), pp 511-523.
3. Dennis, J. B. Modular asynchronous control structures for a high performance computer. Conference Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, pp 55-80.
4. Dennis, J. B. and S. S. Patil. Computation Structures. Course notes published by the Department of Electrical Engineering, M.I.T., Cambridge, Mass. 1971.
5. Dennis, J. B. and S. S. Patil. Speed independent asynchronous circuits. Fourth Hawaii International Conference on System Sciences, January 1971.
6. Holt, A. W. and F. Commoner, Events and Conditions. (In three parts), Applied Data Research, Wakefield, Mass. 1970. [Chapters I, II, IV and VI appear in Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970.
7. Muller, D. E. and W. S. Bartky. A theory of asynchronous circuits. Proceedings of an International Symposium on the Theory of Switching. The Annals of the Computation Laboratory of Harvard University, Vol. 29, Part I, Harvard University Press, Cambridge 1959, pp 204-243.
8. Muller, D. E. Asynchronous logics and application to information processing. Switching Theory in Space Technology, Stanford University Press 1963, pp 289-297.
9. Plummer, W. W. Asynchronous arbiters. IEEE Transactions on Computers, Vol. C-21, No. 1, January 1972, pp 37-42.
10. Swartwout, R. E. One method for designing speed independent logic for a control. Proceedings of the Second Symposium on Switching Circuit Theory and Logical Design, AIEE, October 1961, pp 94-105.