MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Computation Structures Group Memo 122

Cellular Arrays for Asynchronous Control

by

Suhas S. Patil

(A paper published in the Proceedings of the ACM 7th Annual Workshop on Microprogramming 1974)

This research was supported in part by the National Science Foundation under grant GJ-34671.

April 1975

Proceedings of the ACM 7th Annual Workshop on Microprogramming, 1974

Cellular Arrays for Asynchronous Control

by

Suhas S. Patil
Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

The microprogrammed controls which have evolved following Wilkes' suggestion are for synchronous control structures. This paper presents a microprogrammable array for realizing an asynchronous control structure. The work presented here extends Jump's asynchronous control arrays to include many essential control capabilities which were lacking. The arrays realize structures which allow sequencing of operations, conditional branch, subroutine call, fork operation to create parallel processes, join of parallel processes, merging of control, and arbitration with semaphore primitives to achieve controlled access to limited resources. The control to be realized is specified as a Simple Petri net which is then translated into an array representation which represents the microprogram. The cellular array is then obtained by selecting cell configurations in accordance with the node of the array representation of the control. The circuits obtained are asynchronous and speed independent. The microprogrammed array is a cellular form of C, S, NOR, XOR synthesis of Petri Nets.

Introduction

With many design considerations in mind, Wilkes many years ago suggested a microprogrammed control for digital systems. The unique feature of the microprogrammed control was that it utilized a two-dimensional arrangement of cells that could be programmed to obtain the desired control structure. At that time, the high cost of electronic components precluded control structures for computers from being realized in this fashion, but advances in technology steadily reduced the cost of electronic components to a level that microprogramming has become both practical and desirable [8, 9, 11, 13].

The microprogrammed controls which evolved following Wilkes' suggestion are intended for synchronous (clocked) digital systems and are not well suited for asynchronous control such as needed in handling interacting parallel activities where flexibility in synchronization and coordination of parallel processes is required. The purpose of the present paper is to attempt to extend microprogramming to bring the benefits of microprogramming to asynchronous control structures. We feel that asynchronous structures are more suited for controlling interacting parallel activities [6] and that the continuing decrease in the cost of electronic components and the advancements in LSI technology will eventually make microprogramming of asynchronous control a practical reality.

The structures proposed here are two-dimensional cellular structures which can be regarded as extensions of the asynchronous control arrays of Jump [10]. The control structures proposed by Jump allow parallel processing but can realize only a very restricted class of controls, as they do not allow conditional branches or a call to an operator from more than one location in the control. The control structures proposed here can realize a much larger class of controls as they allow conditional operation, call to an operator from more than one location, and are able to perform the necessary arbitration for controlled access to scarce resources. Viewed from the point of programming languages, these control structures allow sequencing of operations, conditional branching, subroutine calls, splitting of a process into parallel processes, merging of parallel processes, synchronization of processes, and the use of Dijkstra's semaphore primitives P and V.

Each cell in the two-dimensional array can communicate with its four neighbors. Programming this array involves selecting appropriate configurations for the cells in the body of the array and at the edge of the array. The array communicates with the outside world by means of control links which are connected to the cells on one edge of the array. The array is completely asynchronous, and does not use any clock signals, and it is speed independent under the assumption that within any individual cell the wires have negligible delays in comparison with the gate delays [7]. Since the array is asynchronous, different parts of the array can act independently subject only to the constraints of the desired control. The array thus fosters parallel operation in a natural way.

In this paper we first describe the class of controls these arrays can realize; this class corresponds to the subclass of Petri Nets known as Simple Nets. We give examples of control and present a convenient notation in which one can express the desired control. The control is then

*This research was done at Project MAC, MIT, and was supported in part by the National Science Foundation under grant GJ-34671.

translated into a matrix notation in which the nodes correspond to transitions and the columns to the places and the nodes at their intersection are marked to indicate how the transitions and places are connected to each other. The desired control is then realized by selecting the cell configurations in accordance with the nodes of the matrix. To prepare the reader in understanding circuits of the cells and how they combine to form the desired control, we first describe a synthesis procedure for asynchronous circuits which underlies these cellular structures; the synthesis procedures are presented only as an aid to understanding and are described briefly. We present the cells of the desired array and show how the desired control can be realized with a two-dimensional arrangement of these cells.

The Class of Controls

The class of controls which we will be able to realize is defined by the class of control flow diagrams that can be constructed by primitives that permit sequencing of operations, conditional branch, merging of control, subroutine calls, creation of parallel activities by fork, synchronization of parallel activities into one activity by join, and arbitration to control access to scarce resources by Dijkstra's semaphore primitives P and V where each semaphore is a single non-negative integer variable. This class corresponds to the subclass of Petri Nets known as Simple Nets. For comparison, the class of controls realized by the conventional synchronous microprogrammed control is represented by the class of control flow diagrams which can be constructed from primitives that permit sequencing of operations where each operation can consist of simultaneous calls to several operators, conditional branch, and merging of flow of control. The proposed cellular array adds to this the ability to start parallel processes which can proceed at their own pace and the ability to coordinate activities of interactive processes by semaphores.

Petri Nets consist of places and transitions and have directed arcs connecting places to transitions and transitions to places [4, 15]. Places can have tokens in them; a token in our use of Petri Nets would indicate a point of control. We will only deal with nets in which each place can get at most one token. A transition fires when each input place of the transition has a token. The transition fires by first taking away a token from each input place and then it places a token on each of the output places, these tokens are then consumed by other transitions. We shall not permit a place to be both an input and an output place of the same transition.

The Simple Nets are a subclass of Petri Nets in which at most one input place of a transition may also be an input place to some other transition. Such a place is called a shared place. A shared place may have any number of inputs and it may be an input place to any number of transitions, but the rest of the input places of a transition which has a shared input place must be unshared. Since a token represents a point of control, a shared place reflects a situation in which the control follows one of alternate paths depending on which transition fires.

In Figure 1b we show a Petri Net representation

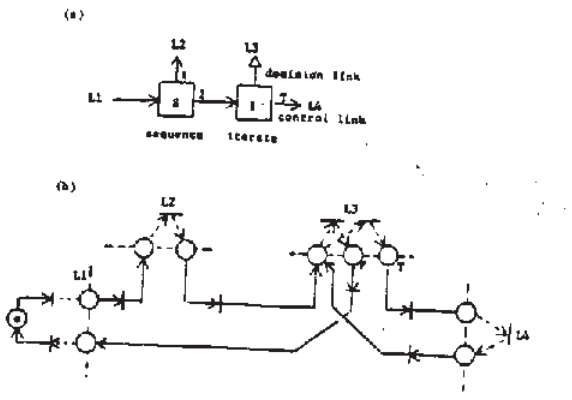


Figure 1. Petri Net representation of asynchronous control.

of the asynchronous control structure of Figure 1a [2, 5, 17]. The control links, which consist of a ready and an acknowledge wire, are represented by two places, one for each wire; the passage of a token through the place associated with the ready wire corresponds to a ready signal and similarly for the acknowledge signal. The decision link consists of three wires, ready, false and true, and is represented by three places corresponding to the three wires. A ready signal to the predicate to which the link is connected generates a response of a signal on the true or false wire.

The control structure of Figure 1 has one sequence module and one iterate module. The sequence module causes the operator under the control of link L2 to operate before the control is sent to the iterate module. The iterate module tests the predicate connected to the decision link L3 at the beginning and if the decision is true then it goes through one cycle of iteration which in this case consists of executing only one operator, the one under the control of link 4. After completing a cycle of iteration the iterate module tests the predicate once again; the iteration thus continues until the outcome of testing a predicate is false.

Control structures represented by Simple Nets can initiate parallel actions and carry out the control necessary to invoke (call) an operation from more than one place in the control and return the control to the appropriate point on the completion of the operation. This capability of Simple Nets is illustrated in Figure 2 which shows how the operator under the control of link L2 is once invoked from outside the iteration and later from inside the iteration loop.

Drawing two places for each link (three for each decision link) and bringing arcs to a link from all locations from where the link may be activated constitutes cumbersome details which can make the control net hard to follow. We have, therefore, devised a graphical notation that expresses the same control but with economy of symbols and which at the same time enhances the transparency of the control. In this notation an ordinary (ready acknowledge) link is represented by two parentheses, and the decision link is represented by one open parenthesis and a number of closed parentheses, one for each possible outcome of the decision. If a link

the node at the crossing of the row corresponding to that transition and the column corresponding to that place (or link) is marked by a circle; if the place (or link) is a shared place (or link) then the node is marked by a square box; and if the place is an output place (or link) of that transition then the node is marked by a cross. Since we do not allow a place to be both an input and output place of a transition, and since a place can be either shared or not shared, each node inside the matrix is either marked as a circle, a square, or a cross, or it may not be marked at all. Figure 4 shows the control of Figure 3 now in the array (matrix) form.

Nodes at the edges terminate the columns and rows. These nodes are marked by square brackets as shown in Figure 4. The net represented in Figure 4 does not contain an internal place which has an initial token; had there been such a place in the net, the initial token would have been represented by a dot next to the top square bracket terminating the place.

The desired physical array will consist of physical cells at the nodes. Each cell will be connected to each of its immediate neighbors by two wires, and each can be programmed to be in one of four configurations corresponding to the four different marks. The rows and columns will be terminated by another kind of cells which can also be placed in one of four configurations depending on whether they terminate the rows or the columns, whether or not the terminated column has an initial token and whether the column is connected to a link.

To prepare the reader in understanding the circuits for the cells and the operation of the cellular array, we first present synthesis procedures for realizing circuits for the class of control represented by Simple Nets.

C, S, NOR, XOR Synthesis

A control structure expressed as a Simple Net can be transformed into a circuit by a synthesis procedure called C, S, NOR, XOR synthesis [16, 17] which substitutes appropriate circuit elements for the components of the nets.

In the circuits realized by this synthesis procedure, transitions in levels (i.e. both 0 to 1 and 1 to 0 transitions) are regarded as signals, and the movement of signals in the circuit corresponds to the movement of tokens in the net. If the net has only the kind of places which have one input arc and one output arc (i.e. no shared places), then the circuit can be realized by substituting wires for places without tokens, NOR gates for places with tokens (note we do not allow any places to have more than one token), and Muller's C element for transitions. This is the C/NOR synthesis [16] (which is a special case of the C, S, NOR, XOR synthesis). One input of the NOR gate is treated as an initialize input, and the other input comes from the circuit representing the transition which feeds the place. The links which communicate with the outside are treated as places; if the place representing the link has no token, the wire representing the place is taken out to the outside to form an output link, and if the place has a token then the NOR gate is taken out to form an input link. The C element representing the transition takes care of fan-in of arcs at the transition; the fan-out just involves feeding the output of the C element to all the places which feed from the transition.

As it has been stated already, C/NOR synthesis works when there are no shared places and each place has only a single input arc. The Simple Nets, however, may have shared places, and a place may have more than one input arc. The C/NOR synthesis is extended by the C, S, NOR, XOR synthesis to take care of these cases [17]. This synthesis also obtains circuits for the nets by the process of direct substitution. The substitution rule is given in the table of Figure 5, and Figure 6 shows a circuit for the net of Figure 3 derived with this synthesis procedure. In this synthesis, the XOR gate takes care of fan-in of arcs at a place. An XOR gate achieves merging of signal paths because a change in the level at any input of an XOR gate causes a change in the level at the output. The case of a shared place is handled by the S element (switch element). Since the token in a shared place can go to only one of the transitions which share the place, the token must be switched (sent) to the appropriate transition. The S module realizes all the transitions

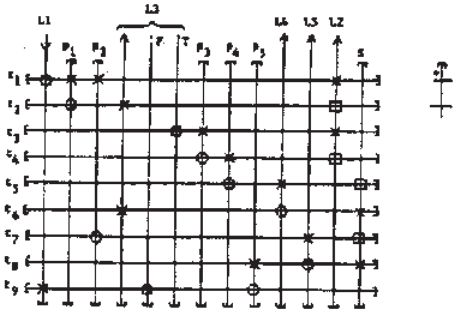


Figure 4. Control of Figure 3 expressed in the array (matrix) form.

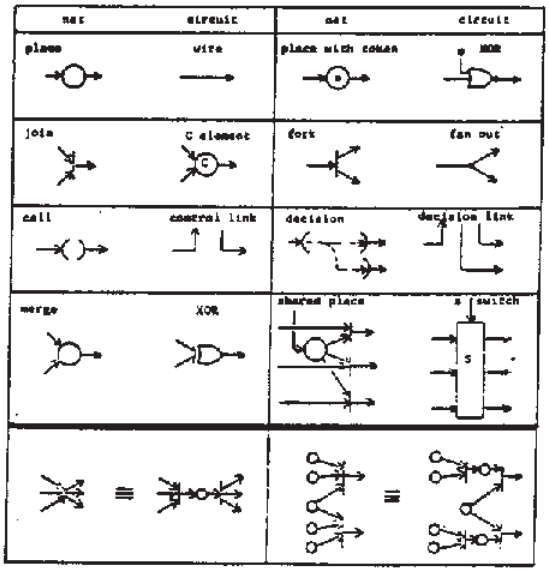
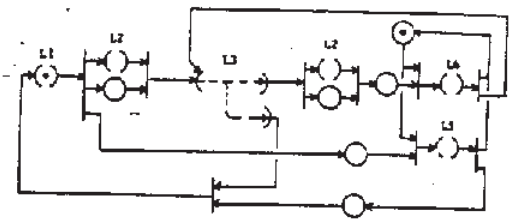


Figure 5. Substitution table for C, S, NOR, XOR synthesis.

(a) The control of Figure 1



(b) The circuit realized with C, S, NOR, XOR synthesis

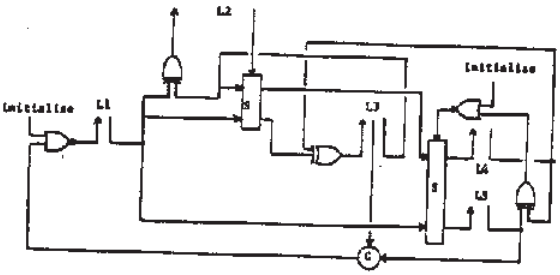


Figure 6. An example of C, S, NOR, XOR synthesis.

which have a particular common input place. We need to switch the signal of only one input place of these transitions because the nets which we are considering are Simple Nets. If more than one of the transitions which share the place are enabled then the switch must perform arbitration among them before performing its normal switching operation. Even though the number of shared input places to a transition is restricted to one, there is no restriction on the number of the unshared input places. Multiple unshared places are accommodated by collecting the wires from the unshared places into one wire by use of a C element placed before the S element. The switch element pairs the signal to be switched with a signal from one of the other inputs and then sends a signal on the corresponding output. The places that represent a decision link do not need a switch in the control circuit itself because the task of switching the signal is performed by the predicate to which the decision link is connected.

A, B, C, NOR, XOR Synthesis

In this section we derive a new synthesis procedure from the C, S, NOR, XOR synthesis to mold the synthesis for the two dimensional arrangements of cells. The new synthesis, called A, B, C, NOR, XOR synthesis derives its name from the elements used in the synthesis; letter A refers to an elementary arbiter, B refers to an element which has one input and two outputs and that sends the input signal to the two outputs alternatively, i.e. the input signals corresponding to 0 to 1 change cause signals to be sent to one output (this output is designated by a dot), and the signals corresponding to 1 to 0 change cause signals to be sent on the other output. An elementary arbiter has two inputs and two corresponding outputs. The function of the arbiter is to prevent the transitions in the level of the inputs from reaching the corresponding outputs as long as necessary to ensure that both outputs are neve level 1 at the same time [17, 18]. Thus if we gard the condition of the arbiter in which all puts and outputs are at level 0 as the initial dition and regard a 0 to 1 transition as a req for service, then the function of the arbiter stop some requests from reaching the output as as it is engaged in the service of the other i and in the event of simultaneous requests to a trate in the favor of one input.

The A, B, C, NOR, XOR synthesis differs fr C, S, NOR, XOR synthesis mostly with respect t realization of the switch element S and the wa tiple fan-in at places and transitions is arra The switch element is now realized from the el mentary components (Figure 7), namely the elem A, B, C, NOR, and XOR. The advantage gained in realization of the switch element is that any n of inputs can now be accommodated using a linea rangement of cells. The operation of this swit element is briefly described below.

Consider the circuit from its initial cond in which all levels in the circuit are at 0, an suppose a signal is received on input s. This the signal which is to be switched to an appro output. The signal enters the circuit from gat and starts circulating in the vertical loop. F it travels as a 0 to 1 change and blocks the art as it passes through them, and after it is retu to gate W it starts out as 1 to 0 change releas the arbiters as it continues its journey. This cess of alternately blocking and releasing the ters continues until some other input, say input

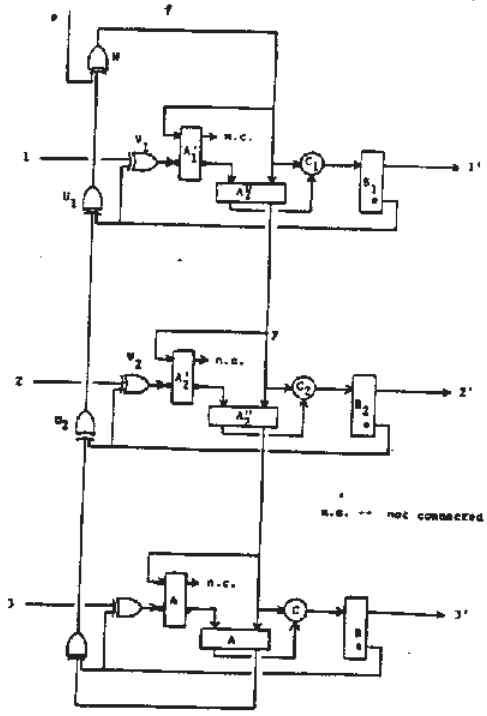


Figure 7. A realization of the switch (S) element.

is received and is able to block the arbiter A_2'' on its way to the associated C element. The circulating signal then halts at the arbiter (A_2''). At this time both inputs of the C element will be at level 1. The output of the C element therefore changes to level 1, and this completes the first phase of operation. Experiencing the 0 to 1 change the B element then sends out a signal to XOR U_2 to eliminate the circulating signal and also sends out a signal to V_2 to prepare the circuit at input 2 for future signals. Essentially, this phase of the operation brings the circuit back to the initial condition before the output is sent out. The signal sent to U_2 passes through the XOR gates U_1 and W and traveling through the arbiters reaches point y and brings the level at this point back to 0. The input to the C element and the input to arbiter A_2' from the vertical loop thus returns to 0. The signal that was sent to the XOR gate V_2 and was halted at A_2' until the above-mentioned action of initializing the vertical loop is completed now passes through arbiters A_2' and A_2'' and makes the other input of C_2 also 0. The output of C_2 then changes to 0, and the B element sends out a signal on output 2' which completes the action of the circuit. The function of arbiter A_2' is to prevent the arbiter A_2'' from being released before the vertical loop is initialized.

If two signals had been sent to the circuit, say on inputs 2 and 3, then the signal received on s would have been paired with one of them depending on which one of them was able to block the circulating input, and output signal would have been sent on the associated output. In the particular case in which the input that is to be switched is received after the inputs which select where the signal is to be switched, then the signal to be switched is paired with the input closest to the top.

The other differences between the A, B, C, NOR, XOR synthesis and the C, S, NOR, XOR synthesis are that instead of a multi-input C element we use a chain of C elements to realize a multi-input transition and instead of a multi-input XOR we use a chain of XORs so that the transitions and the places can also be realized with a linear arrangement of components just as the multi-input switch element was realized through a linear arrangement of cells. A transition which does not have any shared inputs now takes the form shown in Figure 8a, and a transition which shares an input takes the form shown in Figure 8b. The transition is realized as a loop because in the array realization each C element accounts for a distinct input place, and thus we are left with the need to take care of the unused input of the leftmost C element. The NOR gate in the loop provides the initial signal to this input and the loop being closed, the signal returns to the beginning of the loop for each successive firing of the transition.

In A, B, C, NOR, XOR synthesis an unshared place is realized as a linear arrangement of XOR gates as shown in Figure 8c, which accommodates multiple inputs to the place. The transition to which this place is an input place provides a level of 0 to the unused input of the bottommost XOR gate.

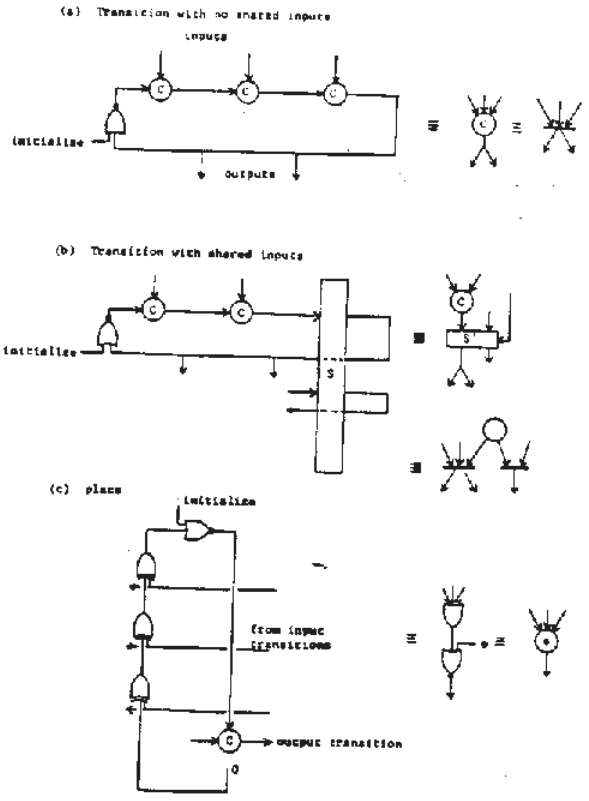


Figure 8. Realizations of multi-input transitions and places.

The Cells of the Array

The cells of the microprogrammable array can be placed in one of seven configurations; besides the configuration called identity, three configurations are needed in the main body of the control array, and three other configurations are needed at the boundary of the array. Figure 9 shows these seven configurations together with the symbols that represent them. The representation of asynchronous control in the array form has already been described. Selecting the configurations of the array cells in accordance with the specification of the nodes in the control expressed in the array form realizes the desired control structure. A row realizing a transition without any shared input places gives rise to a circuit in the array as shown in Figure 8a.

A column representing an unshared place realizes a fan-in of XOR gates as shown in Figure 8c. If the place has an initial token the NOR gate with initialize input is included in the circuits. The column representing a shared input place realizes a circuit for the switch element as shown in Figure 7. In the array such columns are always placed to the extreme right of all unshared places. Thus a transition with a shared place and some unshared input places is realized in the array, as shown in Figure 8b. The two-dimensional array thus realizes the A, B, C, NOR, XOR synthesis of the control in an organized way.

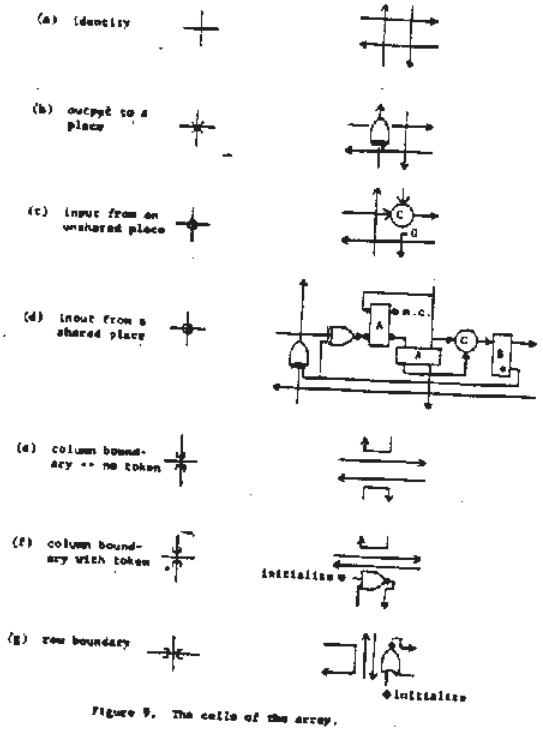


Figure 9. The cells of the array.

Sharing a Predicate

The cellular structures described so far are lacking in that for every point in the control structure where a predicate is required to influence the flow of control, we need a separate copy of the predicate. Thus sharing of predicate is not possible. It may be recalled that this difficulty arose because the restrictions of Simple Nets do not permit structures which can arrange for proper sharing of the control link particularly with regard to returning the control to the appropriate point. It will also be recalled that the operation of the predicate was to evaluate the predicate upon receiving the ready signal and to return a signal on either the true or the false wire depending on the outcome of that evaluation. Now in order to achieve sharing of a predicate we will change the predicate so that it is like any other operator; instead of being invoked by a control link it will evaluate the predicate under the command of an ordinary link and make the outcome of the evaluation available as a 0 or 1 level on a wire, and the control link will generate the true and false signals internally at each location where they are needed. The task of generating these signals in accordance with the level of the wire from the predicate is performed by a predicate cell (Figure 10). When the transition on which this cell is located fires, the cell produces a signal on the right hand wire of the column (which points downwards) if the wire from the predicate (the left hand wire of the column) is at level 1, otherwise no signal is produced. Employing two such cells and providing the level from the predicate and its complement one can simulate a control link as shown in Figure 11. When the transition fires, one of the two columns will

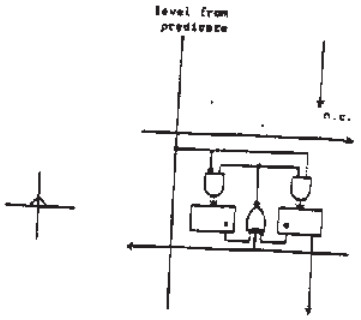


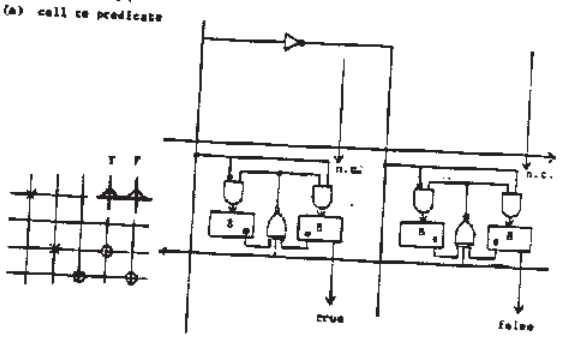
Figure 10. Predicate cell.

get a signal depending on the level of the predicate wire. Sharing a predicate is now possible because the wire that carries the level from the predicate can be common to all such arrangements of predicate cells in one column (Figure 11b).

The cellular arrays employing predicate cells are not completely speed independent as the proper operation requires that the delay of the internal loop of the predicate cell that does not produce any signal must be smaller than the delay of the loop that produces the output in the other predicate cell. Fortunately this can always be arranged by inserting sufficient delay in the path of the signal before it is sent out from the cell. If this is done, the array as a whole will again be speed independent.

We observe that the total number of distinct configurations for the cells of the array is now eight. However, if we preassign some columns of the array for use as a predicate, no individual cell of the array need to have more than four different configurations.

(a) call to predicate



(b) predicate sharing

Figure 11. Call to predicate and predicate sharing.

Conclusion

In comparing the proposed asynchronous microprogrammable array with the conventional microprogrammed control, we notice that we need four different configurations for each cell of the proposed array as opposed to only two configurations for the conventional control, and each cell of the array has a greater number of components. On the other hand the asynchronous microprogrammable array offers parallel processing and the ability to coordinate activities of interacting units. The asynchronous array is intended to be used with a semi-speed independent data structure whose operators are invoked by ready-acknowledge links such as the micro-modular systems developed at Washington University [3], the RTM modules developed at Carnegie-Mellon University [1] and the modular structure developed at MIT [5, 17, 18]. A microprogrammed digital system realized in this fashion has the quality that it can be microprogrammed without the need to keep track of the timing of the operators in detail. The operation of the system can be completely viewed in a logical cause and effect relationship, and this relationship will be directly and faithfully realized by the asynchronous control. The convenience of not having to be bothered with timing details should substantially simplify the task of microprogramming and help bring microprogramming within the reach of many who do not wish to look at the cumbersome details and who would like to work at a higher level.

The developments in LSI technology are decisively changing our outlook towards the nature of the use of components, and it may become profitable to buy the conveniences and other desired qualities of asynchronous microprogrammable control in exchange for higher number of components.† The asynchronous control, being cellular, is well suited to reap all the advantages of LSI.

References

1. Bell, G. C., J. Grason and A. Newell. Designing Computers and Digital Systems. Digital Press, Maynard, Mass., 1972.

2. Bruno, J., and S. M. Altman. Asynchronous control networks. IEEE Conference Record, Tenth Annual Symposium on Switching and Automata Theory, 1969, 61-73.

3. Clark, W. A. Macromodular computer systems. AFIPS Conference Proceedings, Vol. 30, 335-336.

4. Commoner, F., A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. J. of Computer and System Sciences, Vol. 5 (May 1971), 511-523.

5. Dennis, J. B., and S. S. Patil. Computation Structures. Notes for Subject 6.232, Department of Electrical Engineering, M.I.T., Cambridge, Mass., 1971. Most of the relevant concepts were included in the edition of the notes used for a Summer Conference at Princeton University, 1968.

6. Dennis, J. B. Modular, asynchronous control structures for a high performance processor. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, 55-80.

7. Dennis, J. B., and S. S. Patil, Speed independent asynchronous circuits. Proceedings of the Fourth Hawaii International Conference on System Sciences, 1971, 55-58.

8. Husson, S. S. Microprogramming Principles and Practices. Prentice-Hall, Inc., Englewood Cliffs, N. J., 1970.

9. Jump, J. R., and D. R. Fritsche. Microprogrammed arrays. IEEE Trans. on Computers, Vol. C-21 (September 1972), 974-984.

10. Jump, J. R. Asynchronous control arrays. To be published in IEEE Trans. on Computers.

11. Kautz, W. H. Cellular-logic-in-memory arrays. IEEE Trans. on Computers, Vol. C-18 (August 1969), 719-727.

12. Keller, R. M. Towards a theory of universal speed-independent modules. IEEE Trans. on Computers, Vol. C-33, No. 1 (January 1974), 21-33.

13. Minnick, R. C. A survey of microcellular research. J. of the ACM, Vol. 14 (April 1967), 203-241.

14. Muller, D. E. Asynchronous logics and application to information processing. Switching Theory in Space Technology, Stanford University Press, Stanford, Calif., 1963.

15. Patil, S. S. Coordination of Asynchronous Events. Technical Report MAC-TR-72, Project MAC, MIT, Cambridge, Mass., June 1970.

16. Patil, S. S., and J. B. Dennis. Description and realization of digital systems. Proceedings of the Sixth Annual IEEE Computer Society International Conference, San Francisco, Calif., September 1972.

17. Patil, S. S., and J. B. Dennis. The description and realization of digital systems. Revue Francaise d'Automatique, Informatique et de Recherche Operationnelle, February 1973, 55-59.

18. Patil, S. S. Synchronizers and Arbiters. Computation Structures Group Memo 91, Project MAC, MIT, Cambridge, Mass., October 1973.

19. Patil, S. S. Circuit Implementation of Petri Nets. Computation Structures Group Memo 73, Project MAC, MIT, Cambridge, Mass., October 1972.

† Since the writing of this paper we have found a much more economical way of realizing the control.