MIT/LCS/TM-62

AN ASYNCHRONOUS LOGIC ARRAY

Suhas S. Patil

May 1975

An Asynchronous Logic Array

Suhas S. Patil

May 1975

This work was supported by the National Science Foundation under research grant DCR 74-21822.

This document was originally published as
Computation Structures Group Memo 111,
January 1975, and revised in February 1975.

An Asynchronous Logic Array

Suhas S. Patil
Project MAC
Massachusetts Institute of Technology
Cambridge, Massachusetts 01239

Abstract: A new asynchronous logic array for the general synthesis of asynchronous digital circuits is presented. The parallel and asynchronous nature of the array gives the realized systems the speed and characteristics of hardwired circuits even though they are implemented in a uniform diode array with appropriate terminating circuits. The logic array is particularly suited for implementing control structures and should help extend the field of micro-control to asynchronous and parallel computers.

Introduction

The purpose of this paper is to present a new asynchronous logic array, called Kolte array, which is capable of realizing the full class of asynchronous digital circuits without the loss of speed or other characteristics of hardwired circuits. The array is particularly suited for realizing the control structures of asynchronous and parallel computer systems.

The logic array is based on the Petri net model for the representation of asynchronous systems. The full class of Petri nets are realized in the array whose rows and columns implement the transitions and places of the Petri nets. The array is capable of performing the arbitration necessary to resolve conflicts. Control tasks such as mutual exclusion, synchronization of parallel processes and controlled access to shared resources can be easily performed.

The operation of the array is truly asynchronous (it employs no clocks), and it is realized from a diode array which is arranged in rows and columns which are terminated by suitable circuits at the edges. If the system that the array realizes does not require any arbitration, then the array automatically realizes a speed independent system; the operation of this system is unaffected by the propagation delays of the diodes and the circuits in the array.

The array performs its operation in parallel. Therefore, parts of the system that do not interact can carry out their activities in the array without any interference.

Representation of Asynchronous Systems

Petri nets is a parallel schema model which is well suited for the representation of asynchronous systems [6, 8, 16, 18]. In particular, the nets are very useful in the specification of asynchronous digital circuits and control structures.

Petri nets consist of places and transitions which are connected by directed arcs; a directed arc connects either a place to a transition or a transition to a place [8, 9, 18] (there is no arc that directly connects a transition to another transition or a place to another place). The places from which there are arcs incident on a transition are called the input places of that transition; the output places of a transition are similarly defined to be those places which are connected to the transition by arcs which originate at the transition and terminate at the places. A place may have a token or it may be empty. When there is a token in each input place of a transition, that transition is said to be enabled because it is ready to fire. The act of firing involves removing a token from each input place and putting a token in each output place. We will only deal with nets in which no firing sequence for the net can lead to more than one token at a place. Such nets are called safe nets. The safe nets can represent all the finite state systems, since an unsafe net with a bound on the number of tokens can always be translated into a safe net. A fact to remember about transition firing in a safe net is that when two transitions share an input place, even if both are enabled, only one of them can actually fire because the single token at the shared place is consumed in the firing of one transition and the other transition is disabled in want of a token. The number of tokens in a net are not necessarily conserved because a transition does not necessarily have an equal number of input and output places.

An asynchronous system communicates with the external world by means of input and output signals which are sent over input and output wires. In the Petri net representation, each input wire and output wire is represented by a place, and placing a token in a place is regarded as a signal. For example, a 2 out of 3 decoder, that has input wires a, b and c and output wires x, y and z, and which sends output on x if inputs are received on a and b, on y if inputs are received on a and c and output on z if inputs are received on b and c, is represented by a Petri net (Figure 1) which has input places a, b and c and output places x, y and z and which has three transitions t_{ab}, t_{ac}, t_{bc} to perform the logic of the system. For example, if signals are received at a and b then places a and b will get tokens and then transition t_{ab} will fire, placing a token at the output place x. In general, the movement of signals in an asynchronous system will appear as the movement of tokens in the Petri net representation of the system.

In the next illustration (Figure 2b) we show a Petri net representation of a control structure. 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 the passage of a token through the acknowledge place corresponds to an acknowledge signal. The decision link consists of three wires, ready, false and true and is thus represented by three places called ready, false and true, respectively. A ready signal sent to a predicate connected to the decision link results in the reply of a signal on the false wire or the true wire in response.

Drawing two places for a link (three for each decision link) and bringing arcs to a link from all the locations from where it is activated constitutes cumbersome details which can be avoided by the following notational simplification which enhances understanding of the system by improving transparency. In this notation (Figure 2c) an ordinary (ready acknowledge) link is represented by two parentheses, one representing the output place and one the input place, which are placed against each other to look like a place split into two halves, and the place is labelled with the name of the link. Such a pair of parentheses are drawn at every location from which a given link is activated; these parentheses represent only two places, the one corresponding to the ready wire and the other corresponding to the acknowledge wire of the link; the primary purpose of this convention is to vaoid having to draw many lengthy arcs; the fact that they all represent the same link is known from their common link name. If a link is an input link, an initial token is placed at one of the pairs of parentheses corresponding to the link, otherwise the space is left blank. Similarly, the decision links are also represented by a set of parentheses, one corresponding to the ready wire and the others corresponding to the various acknowledge wires such as the false and true wires. It should be again emphasized that the notation that we have introduced is only a linguistic means for expressing the desired control structure with greater clarity.

In Figure 2, the same control structure is expressed first in the notation of control modules [2, 5, 19, 20], then as a Petri net, and finally in the simplified notation discussed above. This control structure has one input link L1, three output control links L2, L4, L5, and a decision link L3. When a ready signal is received on link L1, transition t_1 fires and sends ready signals on links L2 and L5. At the same time a token is placed in place p_1. When an acknowledge signal is received on link L2, transition t_2 fires and sends a ready signal on the decision link. If the decision link acknowledges with a signal on the true input, transition t_3 activates link L2 and puts a token on p_2 to record the fact that L2 is being called from the second location, so that when L2 acknowledges, transition t_4 would fire. Transition t_4 activates (calls) link L4 and when L4 acknowledges, a ready signal is again sent on the decision link to determine if the process of iteration of L2 and L4 should be continued or terminated. When the decision link acknowledges on the false wire, t_6 fires after an acknowledge signal is received on L5. The firing of transition t_6 completes the action of the control structure, and an acknowledge signal is returned on link L1.

If it is desired that the operators controlled by links L4 and L5 not operate concurrently because of a conflict over some common resource, then this constraint can be represented by the use of a shared place with one initial token which plays the role of a semaphore as illustrated in Figure 3. Before a transition can activate L4 or L5 it must obtain the token at the shared place S, and because there is only one token in this place L4 and L5 will never be active at the same time.

Row Column Representation

A Petri net can be represented in a matrix notation in which the rows represent transitions and the columns the places.* If there is an arc from a place to a transition then a dot (•) is placed in the cell at the crossing of the row corresponding to the transition and the column corresponding to the place; if there is an arc from the transition to the place then a cross (x) is placed in the cell. In the Petri nets that we shall deal with, a place can be either an input place or an output place of a transition but not both. Thus a cell may have a dot, a cross or it may be empty.

A row and column representation of the system of Figure 3 is shown in Figure 4. Each control link consists of two columns, one corresponding to the place representing the ready wire and another corresponding to the acknowledge wire. A decision link is represented by three columns corresponding to the three wires of the link; an n-way link would have been represented by n+1 columns. Columns not connected to the outside represent the internal places. In the illustration L1r and L1a represent link L1 and columns p_1 and p_2 represent places p_1 and p_2. The internal places, such as $p_1, \ldots, p_5$, help realize the logic and

*The row column notation is due to Holt.

the control in the system. Row t_1 represents transition t_1 which has an input from link L1 and has outputs to link L2 and places p_1 and p_4. Therefore, this row has a dot in column L1a and crosses in columns L2r, p_1 and p_4.

A control structure specified as a Petri net can be realized either by a synthesis procedure which yields circuits which employ special circuit elements such as arbiters and C gates [22], or they can be implemented in the logic array which is discussed below.

The Structure and Operation of the Array

An asynchronous digital system can be easily realized in the row and column form using the logic array which is described below. The rows of the array, which represent transitions, are each made up of three wires s, r, and $\bar{t}$, and columns, which represent the places, are made of wires c, p and $\bar{p}$. The cells at the intersection of the rows and columns are filled with appropriate diode configurations to form the arcs [Figure 5] of the Petri net representing the system.

The s and r row wires terminate at the set and reset inputs of a flip-flop and the complemented output of the flip-flop drives the $\bar{t}$ wire of the row. (The action of the arbiter shown in the figure will be explained later.) We will refer to this flip-flop as the transition flip-flop because for each firing of the transition it will turn ON at the initiation of the firing and turn OFF at the completion of the firing.

Each column terminates at a place circuit whose input terminal c is connected to the c wire and whose output terminals p and $\bar{p}$ drive the p and $\bar{p}$ wires of the column. Three kinds of place circuits are used to account for the three classes of places, namely the internal places, the input places and the output places of the system (Figure 6). For the place circuits which represent internal places and the input places of the system, a 1 at p denotes the presence of a token at the place and a 0 at p indicates that the place is empty. The level of the p terminal of the place circuit for the output place of the system does not represent a token but instead it represents the level of the output wire (this convention does not pose any problem because an output place is not an input place for any transition in that system).

Firing a transition generates a 1-to-0 change followed by a 0-to-1 change at the c input of all place circuits connected to it. These level changes alter the state of the place circuits. The particular choice of the level changes that turn the place circuits ON and that turn them OFF is so arranged that the tokens from the input places of a transition are removed before tokens are placed in the output places, which is a requirement for the proper firing of a transition. Circuit diagrams for the place circuits are shown in Figure 6. The JK flip-flops in the diagram are triggered by the positive edge, and because the J and K inputs are permanently set at level 1, the flip-flops flip on each 0-to-1 level change at the clock (c) input. The response of the place circuits to the signals on their input terminals is shown in the timing diagrams of the figure. It may be noted that in the case of the output place circuit, the output p changes in response to every 0-to-1 change in the c input. In the case of the input place, the output p changes in response to every change in input s and to every 1-to-0 change in input c. Under normal operation, the input place circuit turns ON as a result of a signal on input s and turns OFF as a result of a 1-to-0 change produced on input c by firing of a transition. The internal place circuit changes state either on a 0-to-1 or a 1-to-0 change in input c in accordance with the diagram shown. Each 1-to-0 change followed by a 0-to-1 change on input c corresponds to the firing of a transition connected to that place. If the place circuit is OFF, then as a result of the firing of a transition, the place circuit turns ON on the 0-to-1 change at input c, and if the place circuit is ON, then as a result of the firing of the transition it turns OFF on the 1-to-0 change at input c. Thus in the course of the firing of a transition, the input places of the transition are turned OFF (loose tokens) at a 1-to-0 change and the output places of the transition turn ON (gain tokens) at a 0-to-1 change.

The intersection of a row and a column is referred to as a cell. Each cell of the logic array that represents an arc is filled with diode configurations; if it represents an arc from a place to a transition (corresponds to a dot in the matrix representation) then the diodes connect the row wires s and r to the wires p and $\bar{p}$, respectively, and the column wire c to the wire $\bar{t}$; if the cell represents an arc from a transition to a place (the cell has a cross), then only the c wire is connected to the $\bar{t}$ wire. The cells which do not represent arcs have no diodes.

The diodes connected to the s wire form an AND gate whose inputs are the p wires of the circuits for all input places of the transition. Similarly the diodes connected to the r wire form an AND gate with $\bar{p}$ wires of those place circuits. The s wire is, therefore, at level 1 only when the place circuits of all input places of the transition are ON and the r wire is at level 1 only when the place circuits of all input places are OFF. The diodes connected to the c wire of a column also form an AND gate with inputs from the $\bar{t}$ wires of all the transitions connected to it. Since $\bar{t}$ wire is normally at level 1, the c wire is normally at level 1. When any one of the transitions connected to it fires, the level first changes to 0 and then returns to 1 at the completion of the firing.

We can now trace the steps in the firing of a transition starting from the condition when all input places of the transition are empty. In this condition the place circuits of all input places of the transition will be OFF, and therefore the s and r wires of the row corresponding to that transition will be 0 and 1, respectively. When all of these place circuits are turned ON by the actions of other transitions or the arrival of input signals, the s wire will change to a 1 (the r wire will have already changed to 0 when the first input place circuit was turned ON), and the transition flip-flop will be turned ON. The $\bar{t}$ wire will thus change to 0 and a 1-to-0 change will be produced on the c inputs of all place circuits connected to the transition. The place circuits of the input places of the transition will, therefore, be turned OFF (the place circuits which represent the output places of the transitions will be unaffected by this level change because the safety of the net ensures that they will be OFF at this time). When all input place circuits of the transition turn OFF, the s and r wires will return to levels 0 and 1, respectively, and the transition flip-flop will be turned OFF. The $\bar{t}$ wire will then return to level 1 and the c wires of the place circuits will experience a 0-to-1 change. This level change will turn ON the place circuits for the output places of the transition, and the action of firing the transition will thus be completed.

The mechanism for obtaining the initial tokens is as follows: When the array is reset at the start all but one place circuit is placed in the OFF state, and one particular place circuit, which will be the source of initial token, is placed in the ON state. A transition which has this place as the only input place and all the places which need the initial tokens as the output places then fires supplying the initial tokens to them. One should also note that the input places of the system get tokens only from the external input, and output places of the system are not input places to any internal transitions of the system. These conventions are not essential: they help simplify the place circuits without the loss of generality.

It may be noticed that all events in the array are conveyed by level changes, and if no conflicts (between transition) are involved, the array will be speed independent under the assumption that the propagation delays of the wire are negligible in relation to the gate delays [7, 17]. That is, the operation of the array will be unaffected by the variations in the propagation delays of the diodes and other circuits in the array.

Arbiters, placed between the wires of the rows and the transition flip-flops that terminate them, resolve conflicts among transitions over the tokens in the shared input places by permitting only one of those transitions to fire at a time. If a transition is enabled when none of the conflicting transitions are in the enabled condition, then the transition is allowed to proceed with firing without any delay; if several transitions are enabled at the same time then the transition whose row is closest to the top is chosen by the arbiter to fire first. When a transition completes firing, the transition closest to the top among those which remain enabled is chosen as the next one to fire. If a transition gets enabled first and a conflicting transition whose row is higher up gets enabled after a critical length of time, then there is a genuine conflict at the arbiter, and it is not possible to say which one of them will be selected by the arbiter, and further it is not possible to say just exactly how long the arbiter will take to make that choice [21]. In any case only one transition will be chosen and the other will be blocked.

The arbiter exercises its control over the firing of the transitions by controlling the propagation of 0-to-1 changes of the s wires to the set inputs of the transition flip-flops. Even if several s wires change from 0 to 1 at the same time, or nearly at the same time, only one of the changes propagates to the corresponding transition flip-flop, and the other changes are blocked at the arbiter. The transition flip-flop which gets the change proceeds with firing and removes tokens from all of its input places in the course of that firing. Transitions which share input places with this transition are therefore disabled. When the r wire gets a 1 at the completion of the firing of the transition, the arbiter is released so that it can attend to other transitions which might still be waiting to fire. The correct operation of the array under conflict requires that the propagation delay of the wires and the diodes of the cells be small in relation to the response time of the arbiter to the release signal. This condition can always be met by inserting additional delay in the circuitry of the arbiter if necessary.

The logic array can test the level of an input wire if two additional cell configurations called 0 and 1 are provided (Figure 5). In cell configuration 0 a diode connects the s wire to the $\bar{p}$ wire, and in cell configuration 1 a diode connects the s wire to the p wire. These are the only connections that are made in these cells. The input place circuits to which the input wire is connected is initially placed in the OFF state so that the level of the p wire follows the level of the input wire whose level is to be tested (see the diagram for the place circuit); the $\bar{p}$ wire will follow the complement of the level. Cell configuration 0 prevents the s wire from becoming 1 unless the input wire is at level 0, and cell configuration 1 prevents the s wire from becoming 1 unless the input wire is at level 1. If we take two conflicting transitions, one with a 0 cell and the other with a 1 cell in the column of the input wire, then the level of the wire will determine which of the two transitions will fire, and thus we have a means to test the level of the input wire.

Microprogramming With the Array

We have explained how the control of a digital system can be specified by Petri nets and how it can be physically realized with the help of the array. In this section we shall see how the array can be the basis of a high-performance micro-control for parallel and asynchronous computers. We shall focus our attention of how many of the concepts of conventional micro-control are also true of the new micro-control and how many difficult and interesting control structures can be easily realized in the new micro-control.

In conventional micro-control, one organizes the control world in fields so that a given control can be realized with words of smaller width [10]. This concept is equally valid for the asynchronous array.

The concept of a field is useful when among a number of control points, only one is activated at a time. For example, say, one of eight registers of a system is transferred to the data bus at a time. Then a three bit field could indicate which one of the transfers is to be performed. In the illustration (Figure 7) the field consists of places p_1, p_2, p_3 together with a ready-acknowledge type of a control link. If the transfer $X \rightarrow A$ is to be performed, transition t_1 puts tokens in places p_1 and p_3 (and leaves p_2 blank) so that the bit pattern 101, which corresponds to the choice of the transfer to be performed, is obtained. At the same time, the transition sends a ready signal on the associated control link to initiate the desired action. When the action is completed, transition t_2 empties places p_1 and p_3 and signals the other parts of the control about the completion of the action.

A micro-control must often decode a bit pattern to determine which one of alternate control functions should be performed. For example, one needs to decode the fields of an instruction to determine how the instruction is to be executed. The decoding task can be completely performed inside the array as illustrated in Figure 8, which shows the operation code of an instruction being decoded by the array. We make use of the 0 and 1 cell configurations. It may be recalled that in order for an enabled transition to fire the columns with 0's must have input of 0, and the columns with 1's must have input of 1. In the illustration of Figure 8, the 101 bit pattern leads to the selection of transition t_5 because this is the only transition with 101 in the appropriate columns. Transition t_5 removes the token from p_1 and puts tokens in appropriate places to begin the execution of the instruction designated by the operation code 101.

Next we shall examine how controlled access to shared resources can be achieved. Suppose there are six users who could place requests for a common resource. To perform resource allocation, we will arrange the control in such a way that the token in the place p_1 will represent the resource. There will be one input place for each user and the transition which allocates the resource to a user will have inputs from the place p_1 and the input place of that user. Firing of any allocating transition will remove the token from place p_1 making the resource unavailable to the other users. In the illustration (Figure 9) the fact that the allocation has been made to a particular user is conveyed to the outside by setting appropriate bits in a three-bit field and then sending a control signal on a control link. When the use of the resource is completed, an acknowledge signal on the control link leads to the firing of a transition which clears the field and puts a token in place p_1 to indicate that the resource is free. The conflicts over the resource (the token at place p_1) are resolved by an arbiter which arbitrates among all of the allocating transitions.

If more than one user is waiting for the resource, one which has the highest priority, the one whose allocating transition is closest to the top will be granted the resource by the arbiter. Needless to say, this simple priority structure does not achieve fair allocation because a low priority user may be indefinitely discriminated against if some higher priority users are always waiting. Again one must remember that the fairness of allocation entirely depends on what is considered fair. To obtain a fair allocator one would have to precisely specify the criteria of fairness and then devise an appropriate structure with places, transitions and arbiters to realize the characteristics of the fair allocator. For example, a fancy round robin allocator is shown in Figure 10. In this allocator the allocation proceeds from user 1 to user 4, but there is no latency time between allocation because the allocator operates associatively. Extending this allocator one can easily realize a round robin allocator in which the users are subdivided into classes; in each class there is a priority scheme and the classes are selected one at a time in round robin to make allocation.

As a last example of a coordination structure, we show a simple minded six-user two-server allocator in Figure 11. Interrupts to a computer can also be efficiently handled by the array in a similar way.

Relation to Other Asynchronous Arrays

The array discussed here is related to two previous arrays, one proposed by Jump [12] an another found earlier by the author [23], both of which attempt to realize asynchronous control structures based on the Petri net model. It was Jump's work on asynchronous arrays [12] which revived our interest in looking for asynchronous arrays for the synthesis of the full class of Petri nets. The array suggested by Jump realized only the subclass of Petri nets known as marked graphs [4]. Marked graphs represent only a very restricted class of control structure; with them one can create parallel processes but cannot perform conditional operation based on a predicate test. Thus the entire system operates in cyclic fashion, every operation being executed exactly once in each cycle. Jump's array is based on the C/NOR synthesis of Petri nets [19].

To permit other essential control features such as conditional operation, calls to an operator from more than one location, and the mutual exclusion of parallel processes, the author devised an asynchronous cellular array [23] capable of realizing the subclass of Petri nets known as Simple nets [24]. Simple nets can represent most of the control features desired in a control structure except that in a control based on Simple nets, the predicate connected to a decision link cannot be tested from more than one location in the control. Furthermore, an array which realizes only the Simple nets cannot be used for the general synthesis of asynchronous systems because many asynchronous systems such as the 2 out of 3 decoder discussed earlier cannot be represented by

Simple nets [24]. The second shortcoming of the cellular array was the complexity of the cells; each cell in the array required several gates.

The design of the cellular array was based on each cell having two wires to communicate with its immediate neighbors. If one insists on only two communicating wires between cells, the sophistication of the cellular array perhaps represents the minimum complexity with which one can achieve speed independence. The dramatic reduction in complexity of the array proposed in this paper is largely due to the fact that each row and column consists of three wires. This array cannot be regarded as a cellular array because the wires run from one edge of the array to the other edge and thus form a kind of bus structure instead of a local connection between adjacent cells. Yet the operation of the array is characterized by speed independence if one can regard the delays in the wires and the diodes to be small in relation to other circuits of the array.

The logic array presented here provides an alternative to the universal modules of Keller [14] for the general synthesis of speed independent circuits. Because of its asynchronous nature, this array falls in a different class in relation to most of the logic arrays which have been studied by researchers [11, 13, 15].

Remarks on Physical Realization

Programming the array involves selecting diode patterns in the diode matrix of the array. If the array is to be made field programmable, one could include nichrome resistors in series with the diodes which could be fused to disconnect the unwanted diodes the way it is done in the contemporary field programmable ROM and diode matrices. A simpler arrangement may be to use a mask programmable diode array which can be programmed in the same way as mask programmable ROM's are programmed.

A more novel technique of programming employs a transistor in place of the diode. The level of the base of the transistor can be raised or lowered to effectively include or exclude a diode at the intersection. Thus if we build one bit of memory for each transistor, then the array can be programmed by loading the memory with a bit pattern corresponding to the desired diode pattern. Such an array could be reprogrammed easily by loading the memory with a new bit pattern. Needless to say that the addition of memory to the array will increase its cost because fewer number of rows and columns could be realized in a given area of semiconductor chip, but the added flexibility may be desired in some special applications.

One possibility is to have an array in which a few of the rows and columns are reprogrammable and the rest are mask programmable. This may provide some freedom in making alterations without excessive costs. One application of this ability is in the realization of arrays for easy fault diagnosis. Many interesting diagnostic methods are possible. One of them is the program stop feature similar to one offered by program debugging aids. To achieve this, one of the columns of the array is arranged to be reprogrammable. The place circuit of this column is left in the OFF state. A program stop can then be placed on any transition by selecting the cell configuration 1 for the cell at the intersection of this column and the row for the transition. The transition will be inhibited from firing because the place circuit of the column is in the OFF state. Program stops may be placed on more than one transition using the same column. When desired, the control may be allowed to pass beyond a program stop by changing the cell configuration to 0 or to a no connection. Using multiple program stops thus one can trace the flow of control. A programmable row is also useful. For example, it can be used in testing transitions by filling appropriate places of the net to cause the transition under test to fire.

One could ask how the asynchronous array may favor against the conventional micro-control based on ROM in cost. The array has four diodes at each node as opposed to a single diode in the case of ROM realized with diode matrix. Therefore, one may expect the Array to be at least four times as expensive (when we compare the array to a ROM realized with diodes). Looking at the array in terms of the number of column and row wires, we see that it has three times the number of wires. Therefore, in the worst case we could expect the array to take nine times the area for a similar number of nodes in a ROM. But this fact can be compensated somewhat if we notice that the rows not involved in arbitration can be interchanged without any effect and columns can also be interchanged.

This might make it possible to use chips which are not completely defect free. The ability to sustain five to ten percent defects in the array will substantially increase the yield and also make it possible to make larger array. If one can control the relative delays of the components, which should indeed be possible if the array is realized on a single chip, then only two wires are needed for each row and column and each cell needs only two diodes. This further cuts down the cost of the array. The cost of the array will in fact depend on many unknown parameters such as the quantity in which it is manufactured. Yet the array is likely to be useful because there are areas in which the conventional control is not adequate either because of the requirement of speed or other requirements such as the ability to perform interlocking and arbitration.

The array can be extended in either direction by adding other chips in wired OR fashion. It is not always essential to make a large array because the arrays being asynchronous parts of the control which are realized on separate arrays can be easily combined to form the large control.

Synchronous Versus Asynchronous

The traditional approach has been to design digital systems as synchronous systems based on a clock. Some of the important factors which influenced the methodology of digital system design to take this course were: Important components of computers, like the memory, were synchronous and it was necessary to operate the processor in synchrony with them. Among the methods that were known, the synchronous approach was the only effective method for preventing the circuit transients from affecting the proper operation of the system. The step-by-step nature of the synchronous systems made it easy to trace the sequence of actions in the systems, and when the gates constituted the major cost of the circuits, the synchronous systems were favored because they required fewer gates.

The advances in semiconductor technology (particularly LSI technology), and the increase in the size and sophistication of computers has gradually changed most of the factors that influence the choice of synchronous approach. The cost of gates has experienced a dramatic drop in relation to other parts. Now one can afford to use many additional gates if that helps improve the digital system. The problem of timing various parts, which was so well handled by synchronous approach when the systems were small, has become an important problem once again especially in the fast and multiprocessing computers. The speed of operation has increased to a point where the signal propagation delays in wires of the system have become significant in relation to the speed of the gates and therefore cannot be ignored. In fact, because of the propagation delay the same clock pulse can no longer represent the same time instance everywhere in the system. Because of the high speed, the problem of synchronization of outside signals to the internal clock of the system has also become difficult. The synchronization problem is a very fundamental problem; every synchronous computer has a nonzero probability of performing faulty computations because of synchronizer failure.

At the same time the theory and methodology of asynchronous systems has developed to a point where it can handle the timing problem, the problem of signal propagation delay and the problem of communicating with the outside and multiprocessing in a much better and clean way. If one begins with a correct paper design, the physically realized asynchronous hardware can be expected to have no timing problem and the costly debugging phase that most newly designed synchronous systems require could be avoided. The memories, input-output devices and other major components of computers now communicate with the computer asynchronously, and there is no need for the processor to be synchronous.

The asynchronous logic array presented in this paper should help the asynchronous approach in several ways. Firstly it will provide a simple, efficient and economical means for realizing the control of the asynchronous systems. Secondly, the MSI and LSI components which are manufactured for synchronous design can be converted for asynchronous operation with the aid of a small asynchronous array which can be accommodated on a single semiconductor chip. This should ease the problem of nonavailability of asynchronous components until the asynchronous approach becomes popular and the manufacturers start manufacturing components for asynchronous design.

The array can realize the control arrays of the micro-modular systems of Clark [3], register transfer modules of Bell [1] and modular systems of Dennis and Patil [5, 19, 20].

Concluding Remarks

The asynchronous logic array provides a general synthesis procedure for asynchronous digital systems. Control structures which previously required hardwired circuits can now be implemented in a uniform array in a manner very similar to a microcontrol but without the loss of either the speed or other characteristics of the hardwired control. The field of micro-programming can now be extended to parallel and high performance computers. The array is also likely to find application in industrial control where the ability to simultaneously handle independent units is essential.

We feel that the asynchronous logic array presented in this paper brings the asynchronous systems closer to where they can challenge the synchronous system in many applications.

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 30, 335-336.

4. Commoner, F., A. W. Holt, S. Even, and A. Pnueli. Marked directed graphs. J. of Computer and System Sciences 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. Holt, A. W. Introduction to occurrence systems. Associative Information Techniques (edited by E. L. Jacks), American Elsevier Publishing Co., 1971.

9. Holt, A. W., and F. Commoner. Events and conditions. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, 3-52.

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

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

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

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

14. Keller, R. M. Towards a theory of universal speed-independent modules. IEEE Trans. on Computers C-33 (January 1974), 21-33.

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

16. Noe, J. E., and G. J. Nutt. Macro E-nets for representation of parallel systems. IEEE Trans. on Computers C-22 (August 1973), 718-727.

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

18. Patil, S. S. Coordination of Asynchronous Events. Technical Report MAC-TR-72, Project MAC, M.I.T., Cambridge, Mass., June 1970.

19. 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.

20. 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.

21. Patil, S. S. Synchronizers and Arbiters. Submitted to the IEEE Trans. on Computers, April 1974.

22. Patil, S. S. Circuit Implementation of Petri Nets. Computation Structures Group Memo 73, Project MAC, M.I.T., Cambridge, Mass., October 1972.

23. Patil, S. S. Cellular arrays for asynchronous control. Proceedings of ACM Micro-7, September 1974.

24. Patil, S. S. Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes. Computation Structures Group Memo 57, Project MAC, M.I.T., Cambridge, Mass., February 1971.

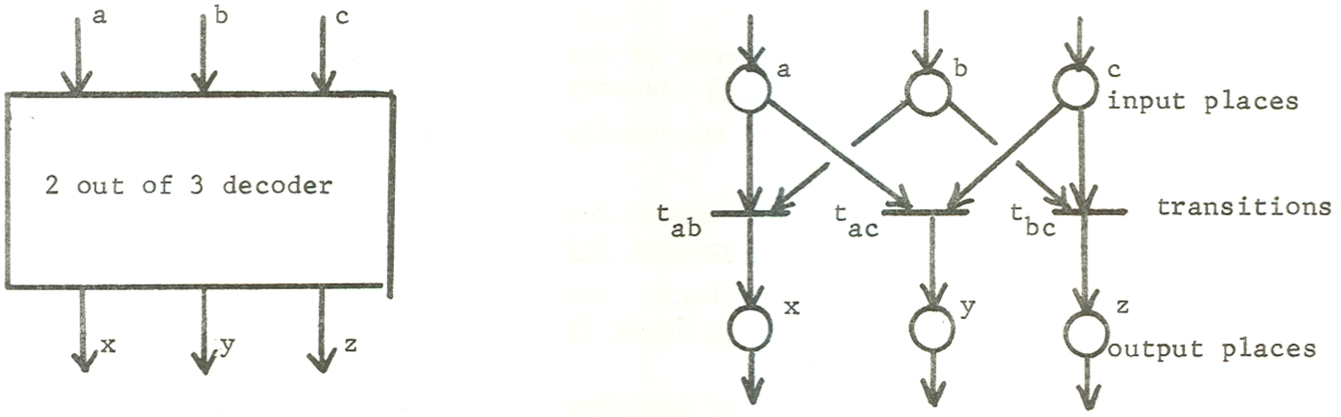
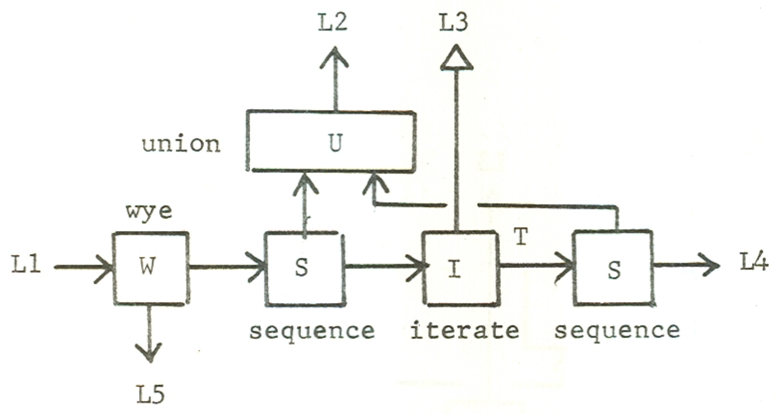
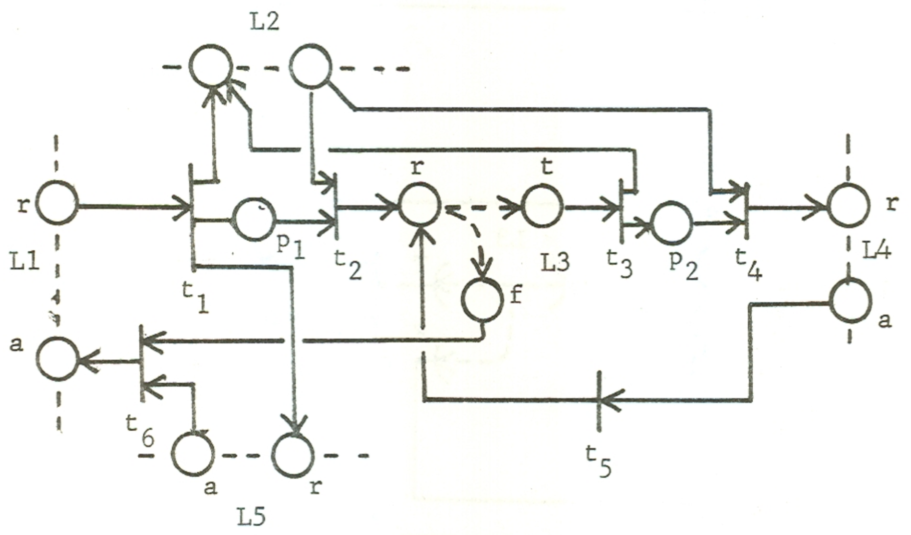


Figure 1. Representation of an asynchronous system.

(a) The Control Structure



(b) Petri net representation



(c) activated notation

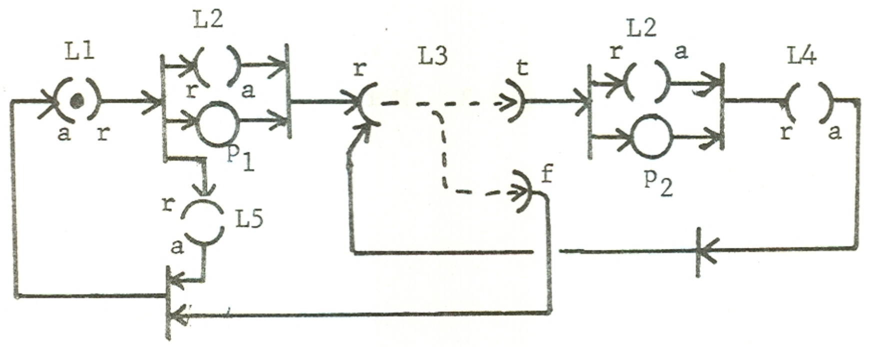
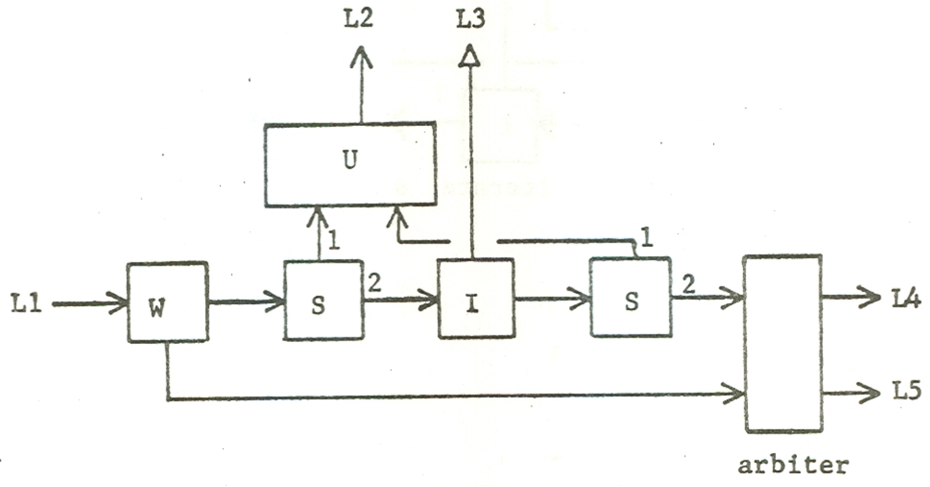


Figure 2. Representation of a Control Structure



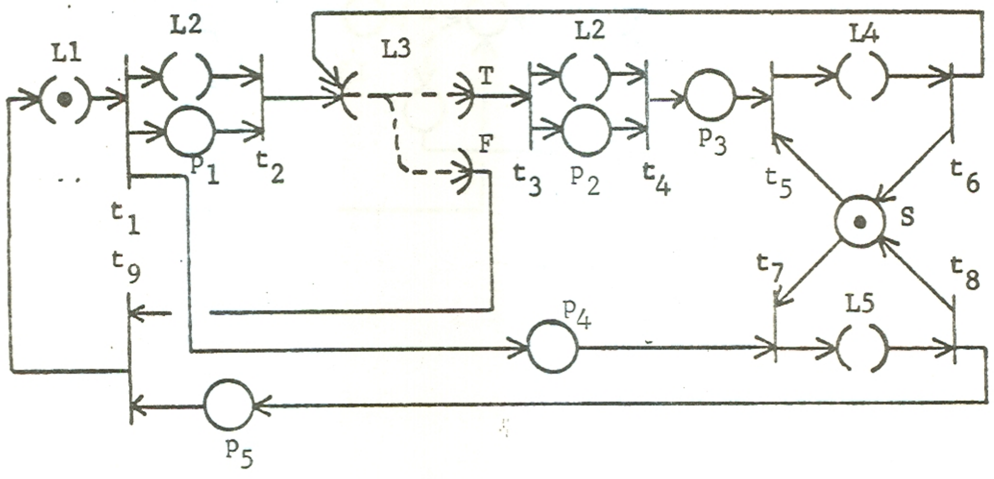


Figure 3. Mutual exclusion.

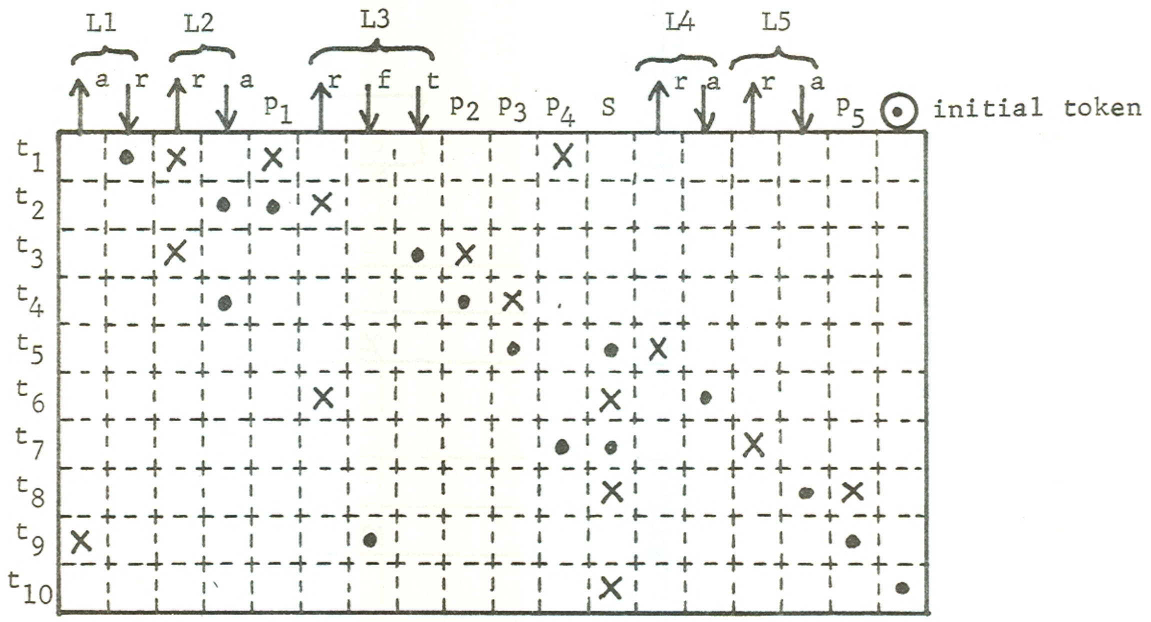
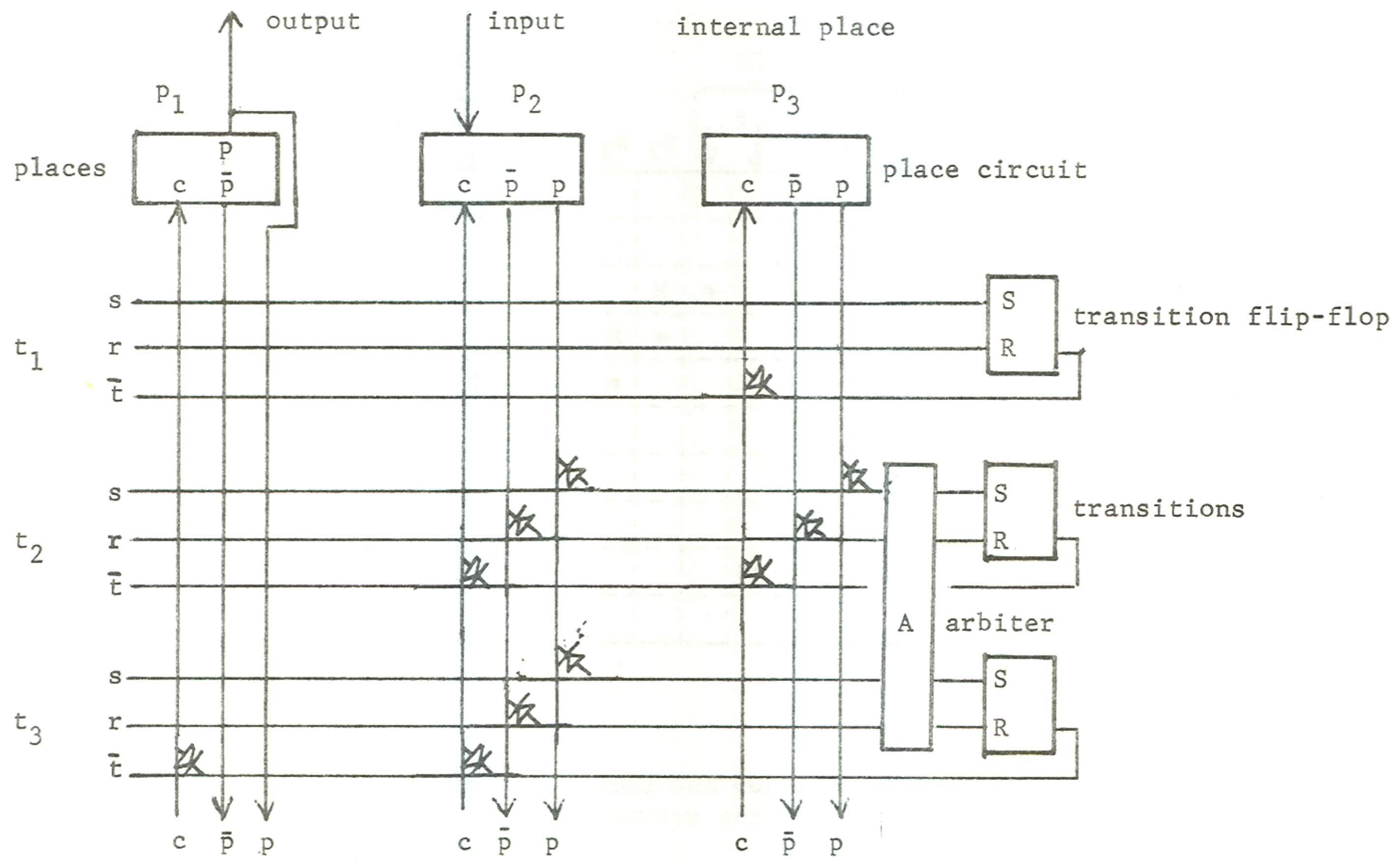


Figure 4. The Row and Column representation
(for the control structure for Figure 3).

(a) The array



(b) The cell configurations

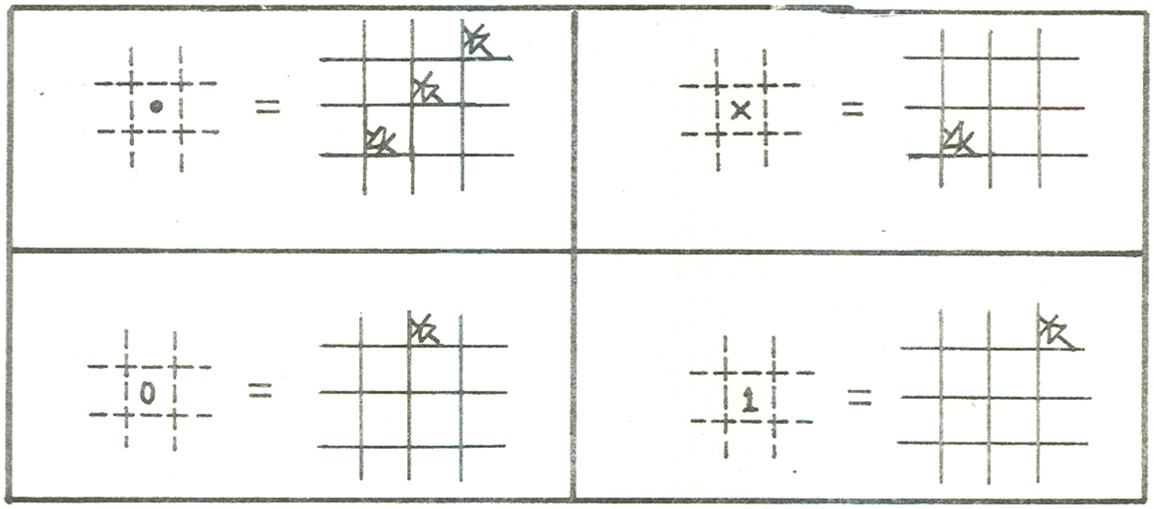
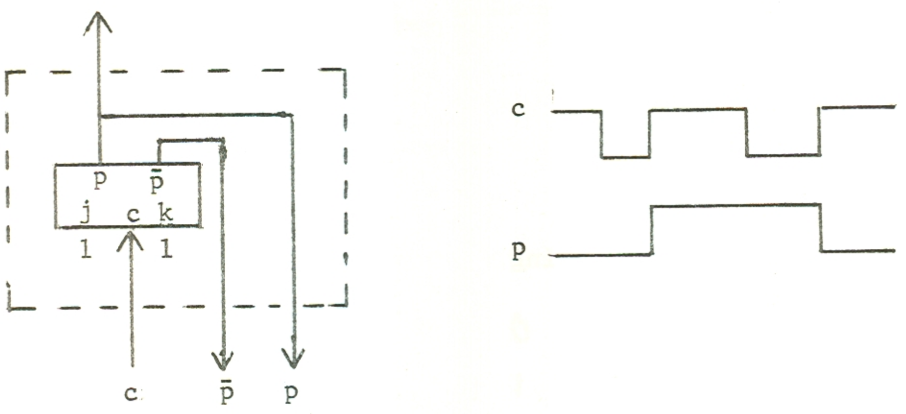
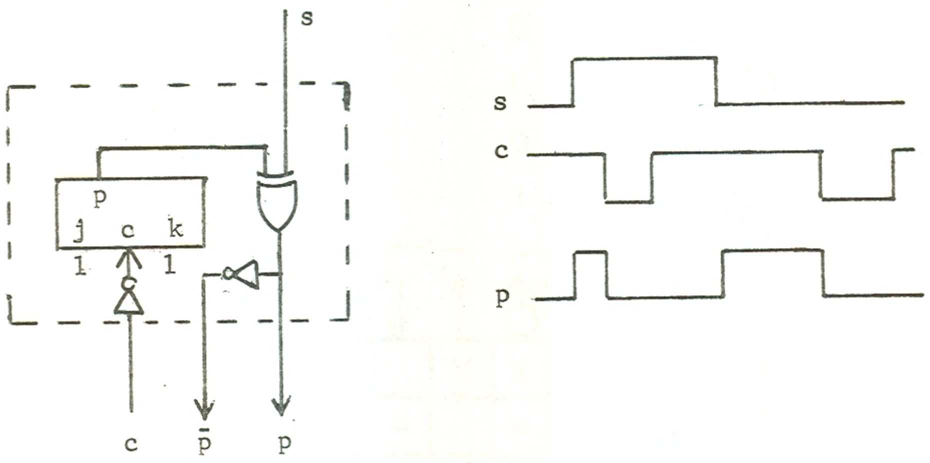


Figure 5. The Kolte array.

(a) output place circuit



(b) input place circuit



(c) internal place circuit

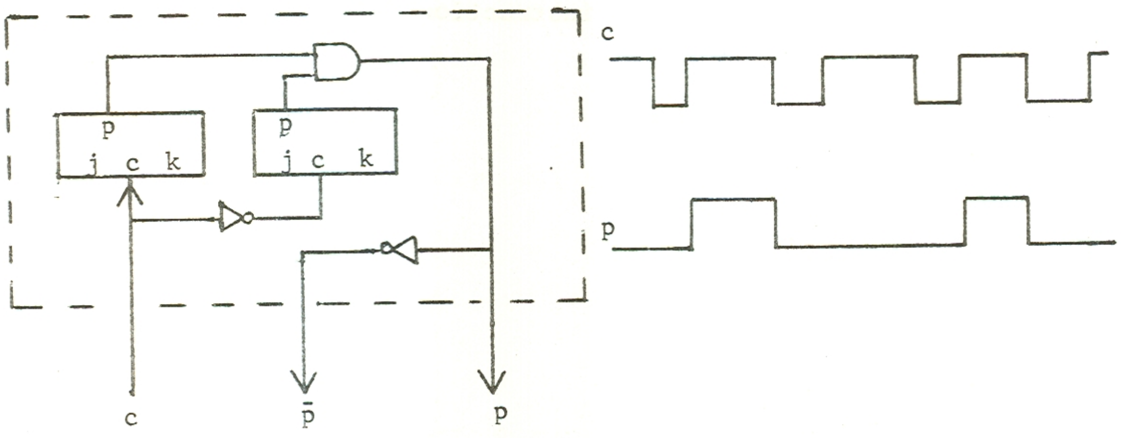


Figure 6. The place circuits.

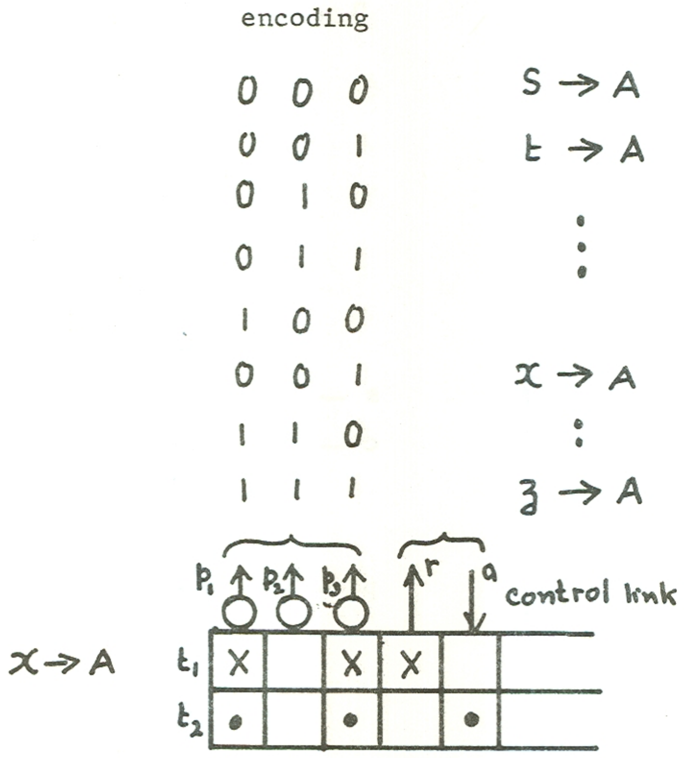


Figure 7. A micro-instruction field.

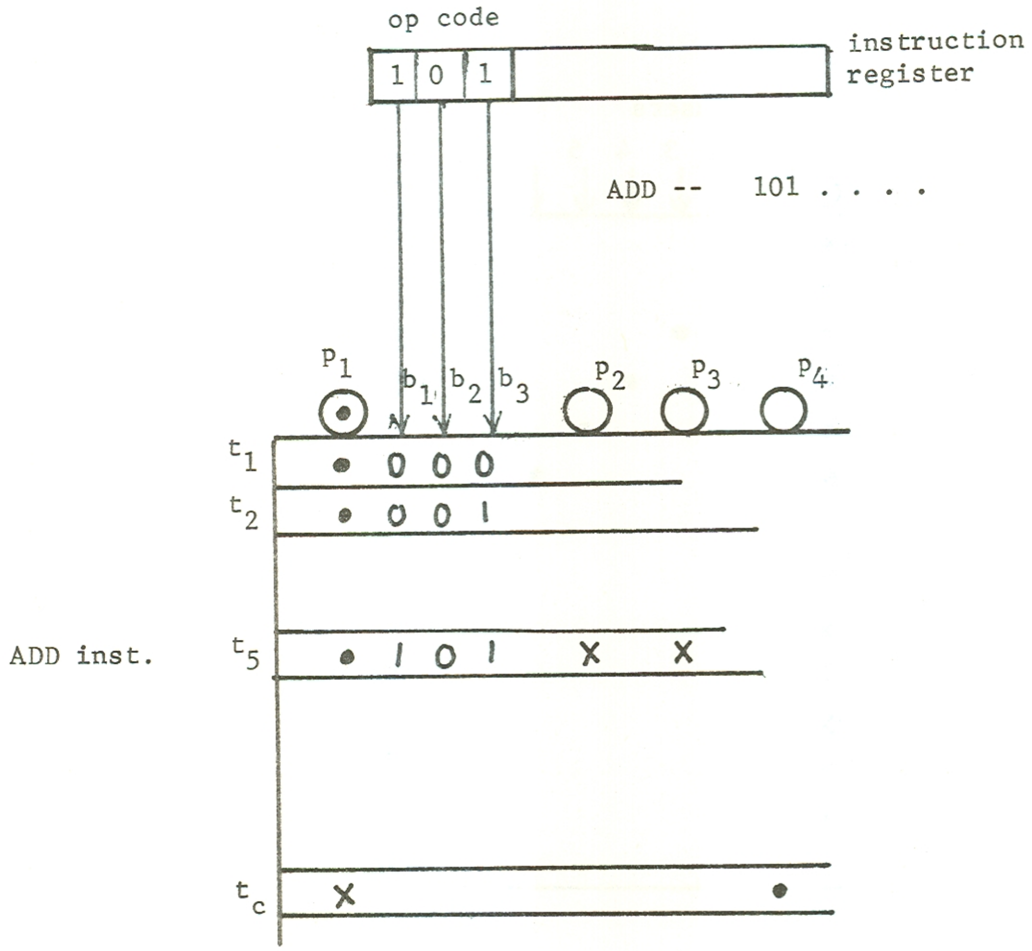


Figure 8. Decoding an instruction.

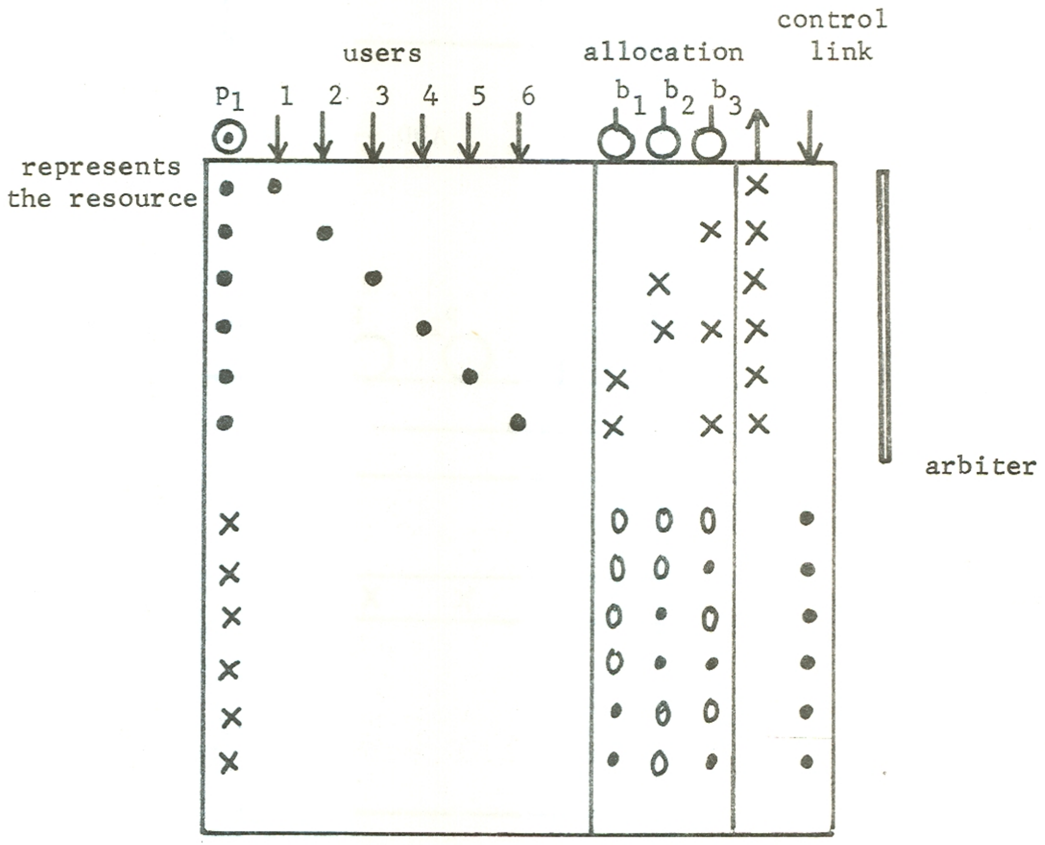


Figure 9. Controlled access to shared resource.

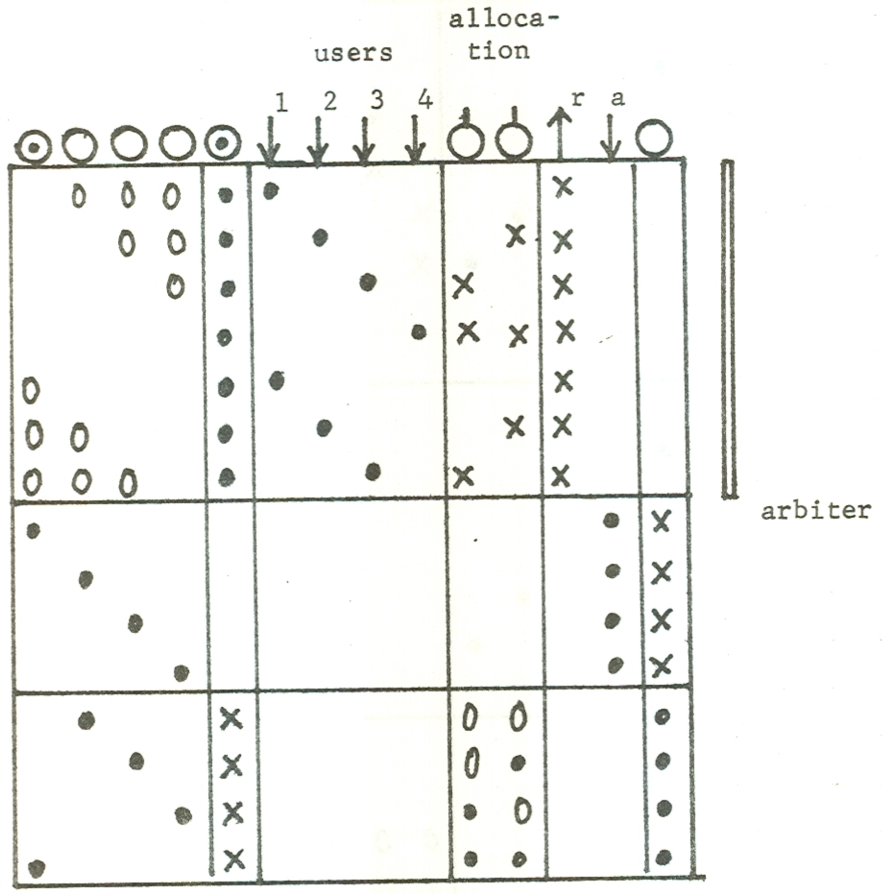


Figure 10. A round robin allocator without ripple.

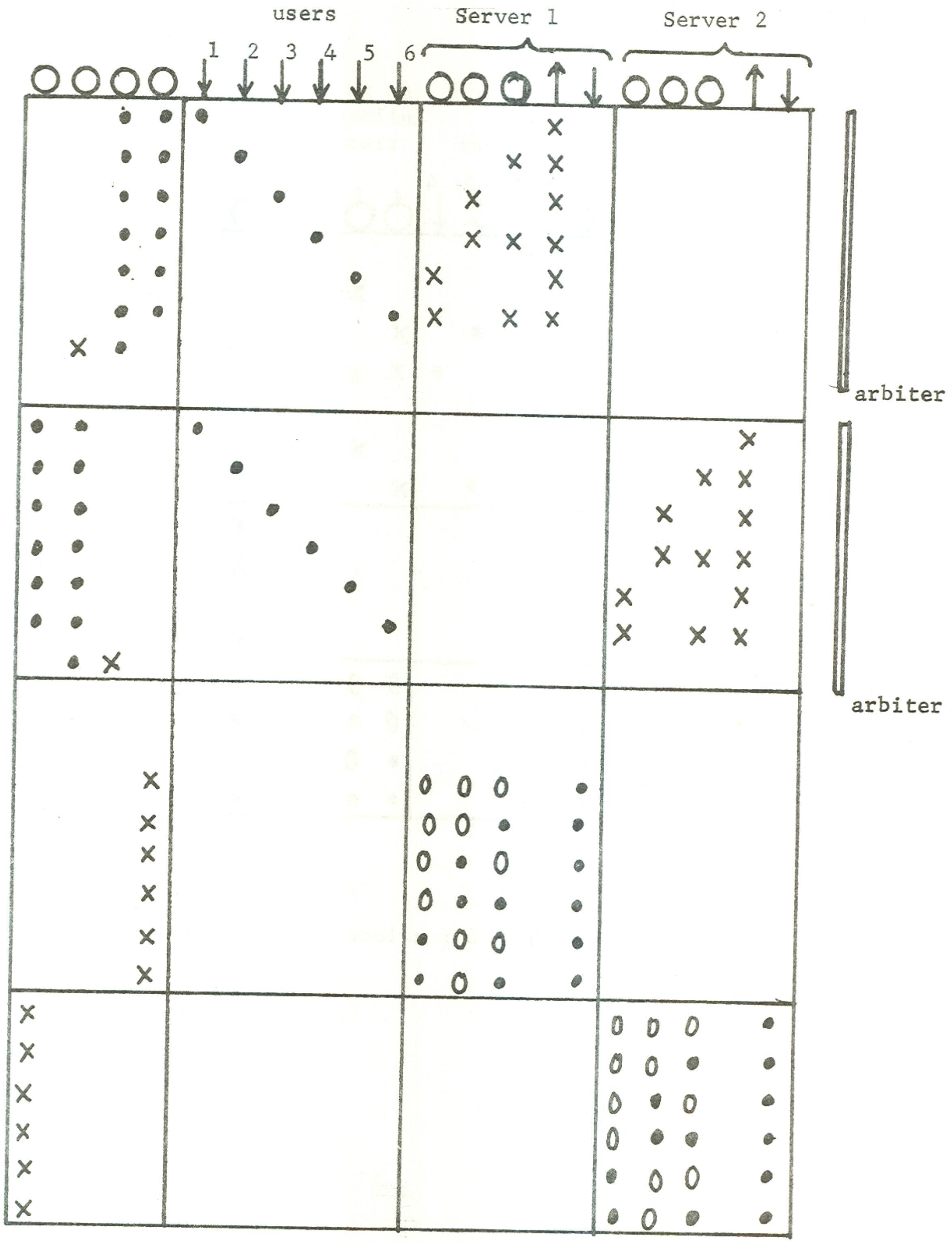


Figure 11. A simple 6 user 2 server allocator.