1-1-1974

Towards a Theory of Universal Speed-Independent Modules

Robert M. Keller
Harvey Mudd College

Recommended Citation
Keller, R.M. "Towards a theory of universal speed-independent modules." IEEE Transactions on Computers, Vol. C-23, Issue 1 (January 1974): 21-33. DOI: 10.1109/T-C.1974.223773

Towards a Theory of Universal Speed-Independent Modules

ROBERT M. KELLER

Abstract—Of concern here are asynchronous modules, i.e., those whose activity is regulated by initiation and completion signals with no clocks being present. First a number of operating conditions are described that are deemed essential or useful in a system of asynchronous modules, while retaining an air of independence of particular hardware implementations as much as possible. Second, some results are presented concerning sets of modules that are universal with respect to these conditions. That is, from these sets any arbitrarily complex module may be constructed as a network. It is stipulated that such constructions be speed independent, i.e., independent of the delay time involved in any constituent modules. Furthermore it is required that the constructions be delay insensitive in the sense that an arbitrary number of delay elements may be inserted into or removed from connecting lines without effecting the external behavior of the network.

Index Terms—Asynchronous, module, networks, parallel, speed-independent, switching.

INTRODUCTION

IT has been suggested that computer design will, in the future, be dominated by organizations which employ large arrays of modules operating simultaneously. This is attributed both to the use of "large scale integration," in which the cost and size of each module is very small, and also the realization that "parallelism" is capable of providing a substantial increase in computing speed and hardware utilization. Most work done to date on arrays of modules has been concerned with synchronous modules, i.e., those which are controlled by a master clock. In this paper, we will be concerned with asynchronous modules, i.e., those whose activity is regulated by initiation and completion signals, with no clocks being present. Computer designers have long recognized that utilization of hardware could be improved by the use of asynchronism, since the length of time that an operation requires, as viewed by a system containing that operation, is equal to the actual time required, rather than the least upper bound, as in the synchronous case. Moreover, the problem of random-varying or unbounded delays may be avoided by using asynchronism, providing a kind of built-in error checking. The fact that a system may be subject to evolutionary changes, or modification while operating may also be provided for. The approach here is also useful in systematic "flowchart" design methods in which a system can be designed and implemented almost directly from a flowchart with little regard for considerations such as races, hazard, fan-out, etc. Such schemes have the effect of reducing the complexity of the design-automation process. A number of systems have been proposed or designed in this spirit. Representative discussions may be found in Muller [23], Clark [7], [8], and Dennis [11].

Manuscript received July 21, 1972; revised August 4, 1973. This work was sponsored by the National Science Foundation Grant GJ-30126 and by Bell Telephone Laboratories, Murray Hill, N.J. A portion of the material presented in this paper is based on work reported in Technical Report TR-102 of the Computer Science Laboratory, Department of Electrical Engineering.

The author is with the Department of Electrical Engineering, Princeton University, Princeton, N.J.

The purpose of this paper is twofold. First, we intend to make precise a number of operating conditions deemed essential or useful in a system of asynchronous modules, while retaining an air of independence of particular hardware implementations as much as possible. Second, we present some results concerning sets of modules that are universal with respect to these conditions. That is, from these sets we are able to construct any arbitrarily complex module as a network. It is stipulated that such constructions be speed-independent, i.e., independent of the delay time involved in any constituent modules. It is here that the techniques differ from more conventional investigations of asynchronous switching (cf., [28]). In the latter, dependence on timing to avoid races, etc., present such overwhelming difficulties that any proposed modifications to a system would likely require that it be redesigned. In the present work we are interested in constructing arbitrarily large systems without global considerations of such things. Furthermore, we are interested in a practical problem that does not presently seem to be amenable to treatment by such theories, namely, the problem of simultaneous input changes.

We are especially interested in sets of modules that have as few interconnecting lines as possible, since the latter would tend to be representable by arrays of less interconnection complexity. Although universal sets of modules are presented here, the author's intention should not be construed as a suggestion that all computers be constructed of such modules. In many cases, commonly used modules could be implemented much more efficiently than the methods presented here would suggest. In this case a decomposition technique, which constructs a network using cheaply implemented modules wherever possible to replace more complex constructions, would be very useful. Furthermore, more general models could be conceived and implemented that might make certain schemes used in the present model unnecessary. Such techniques are discussed briefly in Section V.

I. Preliminary Definitions

Definition 1.1: A *sequential machine* is a 6-tuple $N = (Q, q_0, \Sigma, \Delta, f, g)$, where 1) Q is a finite set of *states*, 2) $q_0 \in Q$ is the *initial state*, 3) Σ is the *input alphabet*, 4) Δ is the *output alphabet*, 5) f: $Q \times \Sigma \rightarrow Q$ is a partial function, the *state-transition function*, 6) g: $Q \times \Sigma \rightarrow \Delta$ is a partial function, the *output function*.

Definition 1.2: A *module* is a 4-tuple (I, O, N, A) where

1) I is a finite set of *input lines*,

2) O is a finite set of *output lines*,

3) $N = (Q, q_0, \Sigma, \Delta, f, g)$ is a sequential machine with Σ in one-to-one correspondence with I and Δ in one-to-one correspondence with $P[O]$.[1]

4) A: $Q \rightarrow P[P[I]]$ is a function which specifies for each state the combination of inputs which can occur.

An element of Σ represents the occurrence of a (one-valued) "signal" on the corresponding element of I. Similarly, an element of Δ represents the occurrence of a signal on the corresponding elements of O. Signals are placed on the input lines of a module m by other modules external to m. In turn, m "assimilates" these signals by possibly changing state and creating signals on its output lines, according to the specification of its machine N. When the state-transition function or output function is undefined for some particular state and input combinations, it is assumed that this combination will never occur in actual operation.

In many practical cases it is necessary to place constraints on the way a module's environment acts upon it, because of the internal construction of the module. A general way of doing this is to specify that in a certain state, certain inputs cannot occur. To make this precise, the function A is present in the definition of a module. For any internal state q of a module, $A(q) \subseteq P[I]$ represents the *allowable input sets*; i.e., if $S \in A(q)$ then any subset of the lines corresponding to elements of S are allowed to be signalled concurrently.

Unless otherwise stated, all modules will be required to satisfy the following.

Condition 1: I and O are disjoint.

Condition 2: A module, once having created a signal on a line, cannot "withdraw" the signal before it is assimilated by a module on the opposite end of the line.

Condition 3–(Arbitration Condition): If two signals appear on different input lines of a module simultaneously, or very close together in time, the action of the module should be as if one signal, then the other, occurred as specified by the sequential machine. If the action depends on the order of occurrence, then the action may be chosen arbitrarily by the module. Hence a module may assimilate signals in one order, even if the actual order of occurrence is just the opposite.

This condition implies the following restriction on the sequential machine: If $f(q, \sigma)$ and $f(q, \pi)$ are both defined, where $\sigma \neq \pi$, then so are $f(f(q, \sigma), \pi)$ and $f(f(q, \pi), \sigma)$.

If the net effect of two concurrent input signals is the same regardless of the order in which two signals are assimilated, we will say that the arbitration condition holds "trivially." The need for nontrivial arbitration arises, for example, in systems in which certain resources such as a memory must be shared among two "processes" in a manner in which either can use the resource but both cannot use it simultaneously. The modules described in Definition 3.5 are useful in this regard.

Condition 4: There may be an arbitrary delay between the assimilation of an input signal by a module and the production of a corresponding output signal. This delay is always finite, but is not necessarily bounded.

Conditions 1 and 2 have been introduced mainly for consistency with certain types of physical implementations. It is possible to remove them and obtain a different class of modules as far as mathematical modeling is concerned. Condition 3 is introduced because there is no way of determining the exact order of two signals occurring sufficiently close together, due to physical limitations on the speed of signal propagation inside the module itself. Condition 4 is fundamental to the assumptions of asynchronism.

Definition 1.3: A *network* is a collection of modules with some of their lines interconnected. If an input line of a module is unconnected, then it is an *input line* to the network. If an output line of a module is unconnected then it is an *output line* of the network.

Just as modules are required to satisfy certain conditions, networks of modules are required to satisfy the following additional conditions.

Condition 5: At most two modules in a network are ever connected by the same line, and this line must be an input to one module and an output from the other.

Condition 6: If a signal is produced by one module on an input line to another module, it must be assimilated before a second signal occurs on the same line.

Condition 7: A line interconnecting two modules has no intrinsic delay.

Whereas Conditions 1-4 were restrictions on module operation, Conditions 5-7 restrict the operation of an interconnection of modules. Conditions 5 and 6 have been introduced for the same reason as Conditions 1 and 2 and could be removed for more generality. Condition 7 is introduced primarily for ease in presenting the model. It will be seen later that no generality is lost as far as the results of this paper are concerned.

In some previous models [11], [23], Condition 6 is ensured by having each interconnecting line from a module m to another module m' accompanied by another line directed from m' to m. The function of this line is for m' to acknowledge to m that the previous signal has been assimilated. However the restriction to such paired signaling conventions causes an unnecessary loss of generality. That is, although it is necessary to have some form of feedback from m' to m, it is not necessary to have this feedback be direct. Instead, feedback may be present through a more indirect path.

From Conditions 2 and 6, it follows that the following must hold for any module m.

Condition 8: Two successive signals can be placed on an input line of m only if they are interspersed by at least one output signal from m, which occurs in response to the input

[1] For any set S, $P[S]$ denotes the set of all subsets of S.

signal. (Otherwise the modules external to m would have no way of knowing when a signal had been assimilated, and would tend to violate Condition 6.)

Condition 9: Two signals can be simultaneously placed on different input lines of m only if the outputs that occur in response to these signals individually are produced on disjoint sets of output lines. (This is because if the output signals were placed on the same line, Condition 6 would be violated for some module external to m.) Condition 9 may be restated as follows.

Condition 10: Two signals which could occur successively on (the same or different) input lines and produce signals on overlapping sets of lines S must be such the latter input occurs after the occurrence of all signals on S which are due to the former.

These conditions impose constraints on the specification of a module's behavior in certain cases.

Clearly a network of modules may be described in the same way as a module itself. The input and output lines of the network are those which are not otherwise connected. The internal state of the network is a set of combinations of the internal states of the constituent modules plus the states of the internal lines. (The state of a line is the presence or absence of a signal on it. Even though a line has zero delay, it is valid to say that it has such a state. A signal is "present" on a line between the time it is produced and the time it is "assimilated.")

In other words, there is a mapping from the set of states of the network into the product set of the set of states of the individual modules and the states of the individual lines. What we are defining then is a type of "realization" of a module by a network, although this word is reserved for use in the stronger sense of Definition 1.7. We are mainly concerned with synthesizing modules as networks of other more-basic modules. We will be interested in syntheses with certain special properties, as described in the following paragraphs.

Definition 1.4: A network is called *speed-independent* if its external behavior is independent of the delay of the constituent modules. (The possibility of arbitration, as in Condition 3, is permitted.)

Definition 1.5: A *delay element* is a module with one input line, one output line, and one state. It functions only to assimilate signals on its input and reproduce them on its output.

Definition 1.6: A network is called *delay-insensitive* if its external behavior remains unchanged, regardless of whether any number of delay elements are inserted into, or removed from any lines.

The delay-insensitive condition appears important for two reasons.

1) It permits a network of any size to be designed without regard to relative spatial distance between modules (which correspond physically to time-delays).

2) It permits a network to be modified by inserting or deleting other networks, so long as such networks have the external characteristics of a series of delay elements.

Although the speed-independent and delay-insensitive conditions appear similar, there is a technical difference due to the presentation of the model. Consider two modules m and m' with a line directed from m to m'. Suppose m must generate a signal on this line, then report to the rest of the network when m' has received the signal. If m generates a signal on this line then, since the line delay is assumed to be nonexistent, m "knows" that the signal has reached m' immediately. However if delay elements are inserted in the line this is not the case. Hence m must in general obtain some kind of feedback from m'. The point is that one reason for introducing the delay-insensitive requirement is so that the assumption about zero line delays cannot be used as a trick to eliminate feedback.

Definition 1.7: A *realization* of a module m is a speed-independent delay-insensitive network of modules which has the same external behavior as m. (A realization always implies some specific internal initial state.)

Finally we assume that every module satisfies, and every realization must be made to satisfy, the following condition.

Condition 11–(Finite-blocking condition): If a signal is present on an input line to a module, this signal must eventually be assimilated.

This condition is related to the "finite-delay" property in [18], [19], the "reliability condition" in [24], and absence of "blocking" and "individual blocking" in [20].

In most realizations presented in the sections to follow, it will be obvious that the conditions stated previously hold and therefore unnecessary to give explicit mention of them. They were, however, an important criterion in selecting the modules presented here and it is quite easy to find pitfalls in which sets of modules appear to be universal, but fail to satisfy these conditions. The easiest condition to overlook seems to be that of delay-insensitivity, although the finite-blocking condition is also subtle.

We now wish to investigate classes of modules which are universal in the sense that from modules of this class we can obtain realizations of any module. We begin by defining universality, and then showing that a certain set of modules is universal for a restricted class of networks. This is then used to obtain results about larger classes of networks.

Definition 1.8: Let $\mathscr{C}$ be a class of modules and $\mathscr{M}$ a set of modules types. $\mathscr{M}$ is called *universal* for $\mathscr{C}$ if every module in $\mathscr{C}$ may be realized (in the sense of Definition 1.7) by a network consisting only of module types in $\mathscr{M}$.

Definition 1.9: A module is called *serial* if it must operate under the condition that every input signal, regardless of upon which input line it occurs, must be followed by exactly one output signal on some line before another input can be applied. Note that this is a proper strengthening of Conditions 2 and 3.

II. Description and Universality for Serial Modules

The behavior of modules will be described either by the conventional state-transition/output table or directed labeled graphs, together with an indication of the allowable input set function A.

Definition 2.1–Merge Module (M): Referring to Fig. 1, we see that the M module produces an output signal in response

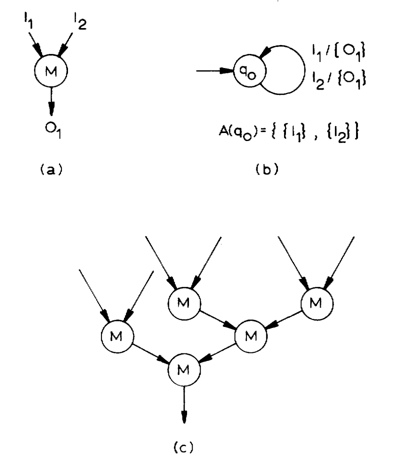


Fig. 1. Merge module (a) Representation. (b) Machine specification. (c) Binary-tree interconnection to form M_6.

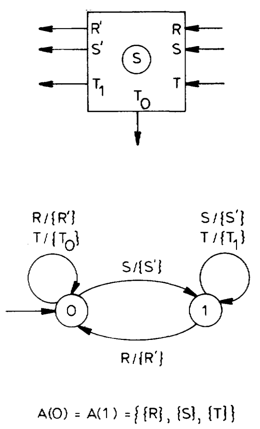


Fig. 2. Select module.

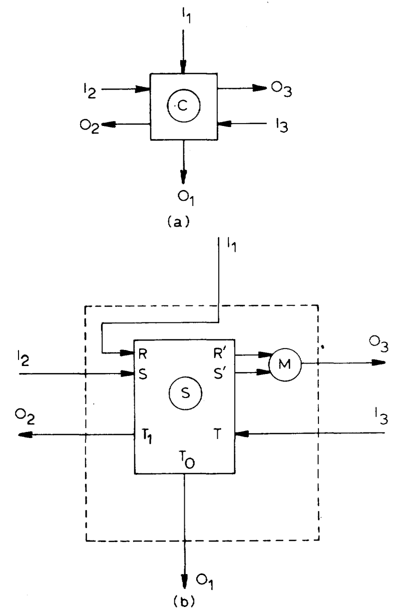


Fig. 3. Call module: (a) representation, (b) realization.

to a signal on either of its inputs. Simultaneous inputs are not allowed, hence this is an instance of a serial module. A generalization of this module is the n-way merge M_n, which may be realized by a binary-tree interconnection of M modules as shown in the figure.

Definition 2.2–Select Module (S): This module has two internal states, indicated by $X = 0$ or 1. The internal state is set by a signal on the S input (see Fig. 2), reset by a signal on the R input, and tested by a signal on the T input. The test signal produces one of two outputs, depending on whether the internal state is 0 or 1. It is important to note that the "signals" in the state diagram are not levels. They are abstract and may be implemented in a variety of ways. The reader having difficulty relating this to physical operation should consult Section IV.

Theorem 2.1: The set $\{M, S\}$ is universal for the class of serial modules.

In presenting the proof, some additional useful modules will be presented which are constructable from the set $\{M, S\}$. The delay-insensitive and speed-independent requirements follow from the fact that in each construction at most one line will have a signal on it at any one time.

It is frequently necessary to have a subsequence of operations initiated by two different sequences in non-overlapping time intervals. This is accomplished by the next module. The terminology is due to [8], [27]. This module is called a "union" module in [11].

Definition 2.3–Call Module (C): The C module has three input-output pairs of lines, $(I1, O1)$, $(I2, O2)$, and $(I3, O3)$, and an internal state X. A signal on either $I1$ or $I2$ causes a signal on $O3$. This is expected to be followed by a signal on $I3$ which causes a signal on $O1$ or $O2$, depending on whether the original signal was $I1$ or $I2$, respectively. Thus X "remembers" the original input signal. We refer to $(I1, O1)$ and $(I2, O2)$ as "calling" pairs and to $(I3, O3)$ as the "called" pair. A realization is shown in Fig. 3.

Several call modules may be interconnected to achieve any number of calling pairs. One way to do so is by the interconnection shown in Fig. 4. Here $(I(k+2), O(k+2))$ represents the "called" pair of lines. To prevent subsequent diagrams from being too cluttered, an abbreviation is adopted for this configuration as shown in Fig. 4. Here α is some symbol which acts as a link between all of the individual modules of an interconnection. A box containing "call α" represents a calling pair and the box containing "α call" represents the called pair. An alternate method of achieving this same effect is to use a binary-tree interconnection. This interconnection uses the same number of modules and is probably "faster" since only $\lceil \log_2(k) \rceil$ modules will be traversed, as compared to an average of $\lceil k + 1/2 \rceil$ for the linear interconnection, assuming that it is equally likely that the call originates from any unit.

Definition 2.4–D-call Module (DC): This is a module similar to the C, except that there are two called inputs and four calling outputs. The DC functions like a call which

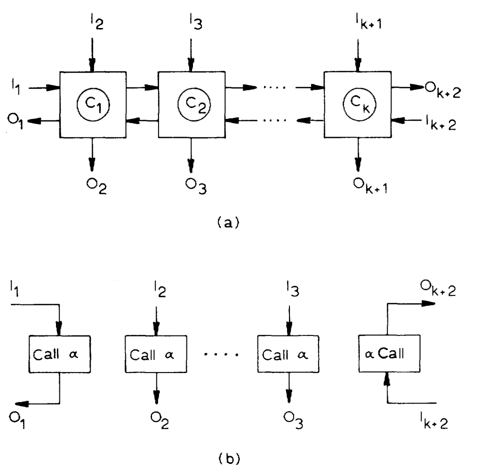


Fig. 4. (a) Cascade interconnection of call modules. (b) Abbreviated representation.

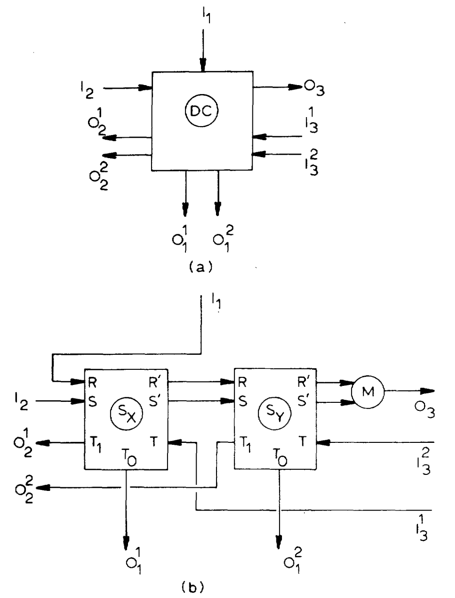


Fig. 5. *D*-call module: (a) representation, (b) realization.

preserves the outcome of some test. The operation is explained thoroughly by the diagram of the construction of this module as shown in Fig. 5.

Definition 2.5–m-way DC modules (DC_m): By using m S modules, a call may be constructed which preserves the outcome of an m-way test. The DC_m will be depicted as in Fig. 6. The construction is left for the reader. By cascading DC_m modules, as was done for C modules, any number of calling sequences may be achieved. This is abbreviated symbolically as in Fig. 7.

Using the modules presented previously, we are now able to show the construction of an arbitrary serial module $m = (I, O, N, A)$ where $N = (Q, q_0, \Sigma, \Delta, f, g)$. Assume that n has n states. We record the current state as a 0 state in one of a series of $(n - 1)$ S modules. The network of Fig. 8 shows the means of testing and setting the state using calls and D-calls. Assume that the current state is q. A signal on the jth input line initiates a control sequence as shown in Fig. 9 so as to first determine the state, given by $f(q, j)$, and then set the new state, given by $g(q, j)$. By "merge to $g(q, j)$" in the diagram, we mean that for each output line there is a tree of merge modules which has inputs from exactly those sequences which are to produce an output on the line $g(q, j)$. This completes the proof of Theorem 2.1. An alternate construction, which uses only $\lceil \log_2 |Q| \rceil$ S-modules for the state recording network is also possible. We leave this construction as an exercise for the reader.

Definition 2.6: A set of modules is said to have *modularity* n if n is the maximum number of lines on any one module in the set.

In the preceding discussion, it was shown that a set with modularity 7 and cardinality 2 is universal for the class of serial modules.

We now show that there are universal sets of modularity 6 and 5 for the class of serial modules. The modules G and H as

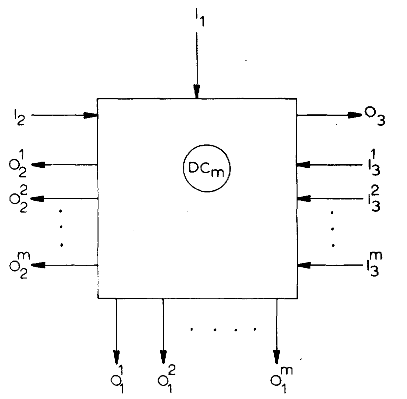


Fig. 6. DC_m modules.

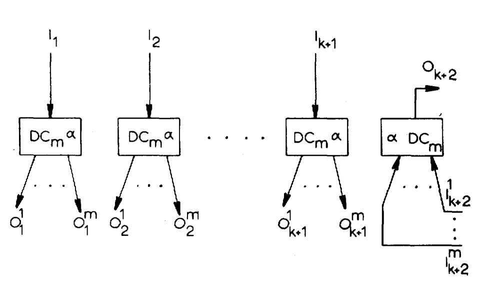


Fig. 7. Abbreviated representation of cascaded DC_m modules.

shown in Fig. 10 may be combined to realize an S module. Hence we have the following.

Theorem 2.2: The set $\{G, H, M\}$ is universal for the class of serial modules.

By using a trick of "sharing" two input lines of the K module of Fig. 11, we may realize an H using K and M to obtain the next result.

Theorem 2.3: There exists a set (namely, $\{K, G, M\}$) of modularity 5 which is universal for the class of serial modules.

Open Problem: Is there a set of modules with

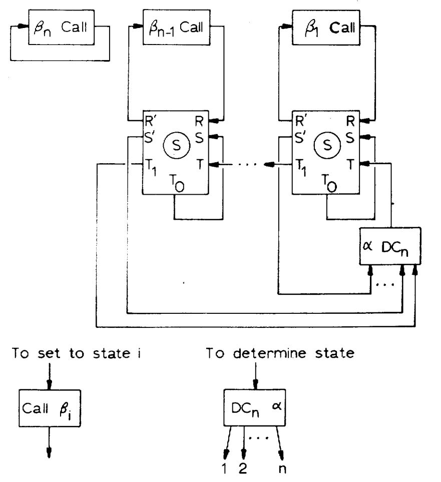


Fig. 8. State testing and setting network for realization of arbitrary serial module.

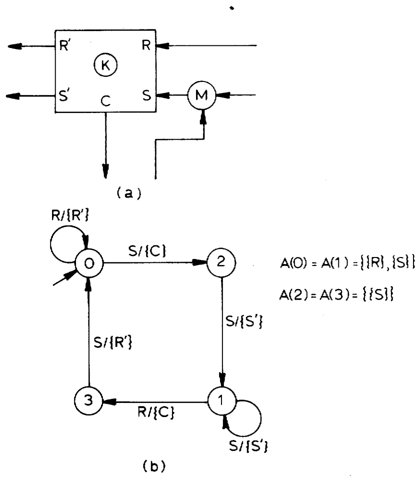


Fig. 11. (a) Realizing H with K and M. (b) Machine specification for K.

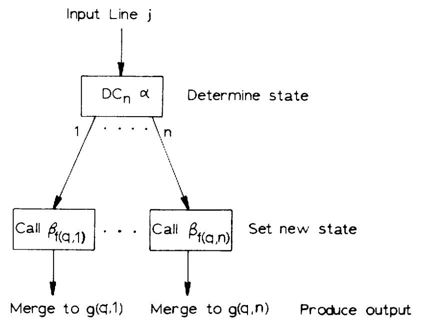


Fig. 9. Scheme for realization of an arbitrary serial module.

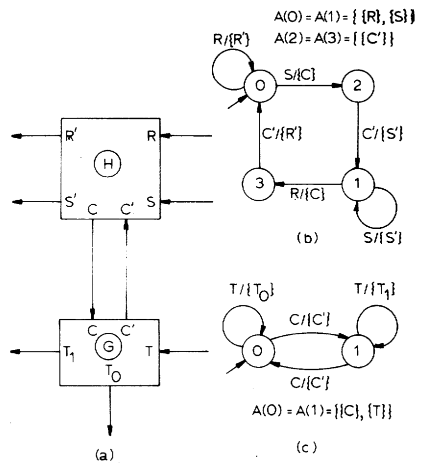


Fig. 10. (a) Realizing S with G and H. (b) Machine specification for H. (c) Machine specification for G.

modularity 4 or less which is universal for the class of serial modules?

III. Further Results on Universality

Definition 3.1: A module is called *parallel* if it allows more than one signal on different inputs which are not necessarily separated by an output signal, or if it may produce more than one output signal on different lines due to a single input. (Conditions 2 and 6 are still to be observed however. Hence it is always assumed, in the case of parallel modules, that the sequential machine is specified in a manner consistent with Conditions 8-10.)

Examples of parallel modules are given next.

Definition 3.2–Join Module (J): The J produces an output signal only after both inputs have been signaled, as shown in Fig. 12.

Definition 3.3–Fork Module (F): The F produces one output on each of two lines in response to each input, as shown in Fig. 13.

Definition 3.4–Arbitrating Test-and-Set Module (ATS): This module operates in a manner similar to the S module except that there are fewer lines, the T always sets the internal state to 1, and simultaneous signals are permitted on both input lines. By the conditions stated previously, if the inputs are signaled simultaneously, the module acts as if one input, then the other, occurred (see Fig. 14). Note that according to the allowed input function A, any number of T inputs may occur in any state, but no R inputs can occur in state 0. Thus an occurrence of T_0 or T_1 indicates that the previous T has been assimilated and the occurrence of T_0 indicates that an R input has been assimilated. However T_0 never occurs in response to R alone, but only R in conjunction with T.

Observation 3.1: It is not difficult to show that, given an AS module ("arbitrating S" module, defined to have the same specification as the S module, except simultaneous inputs are allowed), the ATS can be realized. Our purpose in introducing the ATS is that we wish, for reasons which will become apparent, to minimize the complexity of atomic parallel modules as much as possible.

Definition 3.5–Lock-Unlock Modules (LUn): This is a class of modules, each with a maximum of $2n$ input and $2n$ output lines. Each module in this class is realizable from the set $\{F, M, \text{ATS}\}$. There are n pairs IUi, OUi, $i = 1, 2, \cdots, n$ of *unlock lines* and $m \leq n$ pairs ILj, OLj of *lock lines.* Whenever a signal occurs on a lock input ILi, a signal is produced on the

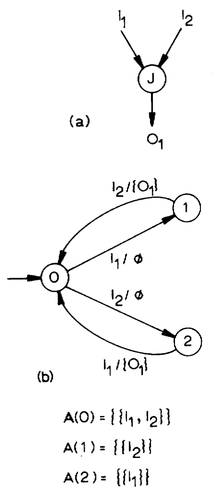


Fig. 12. Join module: (a) representation, (b) machine specification.

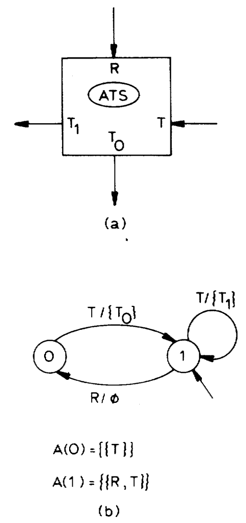


Fig. 14. Arbitrating test-and-set module: (a) representation, (b) machine specification.

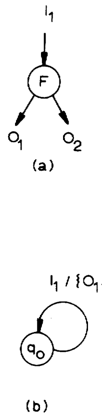


Fig. 13. Fork module: (a) representation, (b) machine specification.

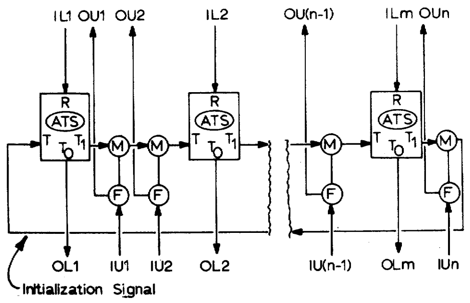


Fig. 15. Realization of a lock-unlock module.

corresponding lock output line *OLi*, and all other lock inputs are "locked out". This means that signals on other inputs are not assimilated. This condition holds until one of the unlock inputs between *OLi* and *OL*$(i + 1)$ (additional modulo *m*) is signaled. Then the next input signal occurring cyclically to the right, if any, is assimilated and locks the module. The realization is shown in Fig. 15. The module's operation depends on the fact that there is an initial signal present in the network. This signal circulates in a "busy-waiting" loop until an input signal occurs.

It is not difficult to see that sets such as $\{M, S\}$ which were universal for the class of serial modules are not generally universal. This is summarized next.

Theorem 3.1: No set of serial modules can be universal for the set of all modules.

Proof: Consider the realization of a parallel module from only serial modules. The requirement that only one input be signalled at a time means that the realization is separable into disjoint serial parts. However in general this is not possible; e.g., the output of the *J* module depends on both inputs, which could be signalled concurrently.

We next investigate sets which are generally universal.

Theorem 3.2: The set $\{M, S, F, \text{ATS}\}$ (modularity 7) is universal for the class of all modules.

Proof: Let $m = (I, O, N, A)$ be a parallel module. The behavior of m is given by $N = (Q, q_0, \Sigma, \Delta, f, g)$ with Σ corresponding to I and Δ to $P[O]$. We first realize a serial module m' with inputs I and outputs $O' = P[O]$, by assuming that only one input to m changes at a time.

To complete the construction, the inputs of m are fed through the lock inputs of a lock-unlock module, and into m', while the outputs of m' are fed into the unlock inputs of the lock-unlock module. This ensures that only 1 input to m' occurs at a time. The unlock outputs are then fed into a "distributor" which converts the output signals of m' to possibly parallel output signals by using a set of fork and merge modules, in the manner shown in Fig. 16. The fork modules produce the simultaneous output signals while the merge modules merge the signals from different control sequences.

The only subtle point to be discussed lies in the possibility that once the LU module has been unlocked, another input may come in, causing m' to produce an output which would cause simultaneous inputs to a merge in the distributor.

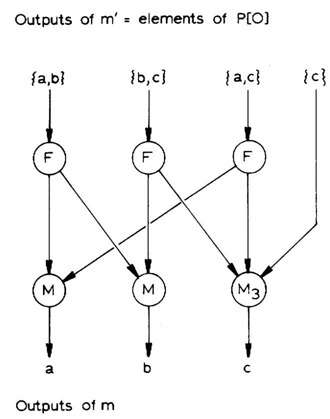


Fig. 16. The final output stage of a parallel module.

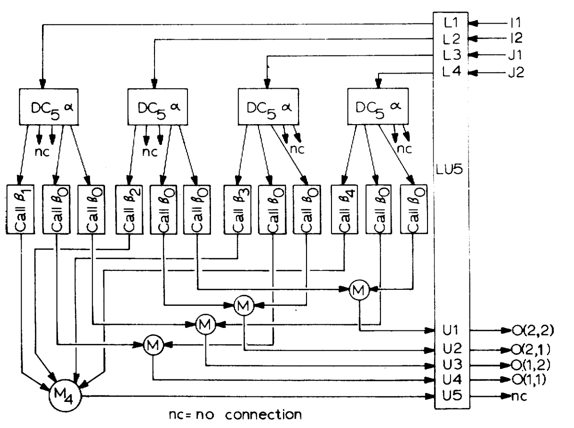


Fig. 17. Realization of a $J(2, 2)$ module, αDC and each β_i-call are not shown, being an instance of Fig. 8.

However this will not happen, since by Condition 10, two concurrent input signals must not produce signals on any of the same output lines.

Example: We demonstrate the construction above by presenting a realization of an $m \times n$ Join module ($J(m, n)$). This module is a generalization of the Join presented earlier. There are $m + n$ input lines, $I1, I2, \cdots, Im$ and $J1, J2, \cdots, Jn$, and $m \times n$ output lines $O(1, 1)$, $O(1, 2)$, $\cdots$, $O(1, n)$, $\cdots$, $O(m, n)$. The module waits for an input on one of $\{I1, I2, \cdots, Im\}$ and one of $\{J1, J2, \cdots, Jm\}$ and, supposing that these signals occur on Ii and Jj, respectively, the module produces a signal on $O(i, j)$. Construction according to the proof of Theorem 3.2 is shown in Fig. 17. For ease in presentation, we show only the case $m = n = 2$. In the figure, certain outputs of the DC modules are not used, and are marked "*nc*" to avoid cluttering. Also, the state changing network, an instance of Fig. 8 with $n = 5$, is not shown.

There are 5 internal states:

0 quiescent;
1 input on $I1$ received;
2 input on $I2$ received;
3 input on $J1$ received;
4 input on $J2$ received;

f, g, and A are specified in Table I.

Since only one output occurs at a time, no distributor is necessary in this case.

Corollary: The set $\{M, G, H, F, \text{ATS}\}$ (modularity 6) is universal for the class of all modules. The set $\{M, G, K, F, \text{ATS}\}$ (modularity 5) is universal for the class of all modules.

Proof: Theorems 2.1, 2.2, and 3.2.

Corollary: The set $\{M, \text{AS}, F\}$ (modularity 7) is universal for the class of all modules.

Proof: Theorem 2.1 and Observation 3.1.

It is natural to ask whether a realization is possible which does not use the busy-waiting construction described in Definition 3.5. Such a question arises, for example, if we are using a system of asynchronous modules to model interacting processes in an operating system. In this case, long-term busy-waiting implies the waste of the resource of one processor and is to be avoided whenever possible. In contrast, if we intend each module to be a piece of hardware, we do not really care about using busy waiting, except possibly from a timing point of view. The question posed above is now made precise and answered affirmatively by the following result.

TABLE I

f	I1	I2	J1	J2
0	1	2	3	4
1	-	-	0	0
2	-	-	0	0
3	0	0	-	-
4	0	0	-	-

g	I1	I2	J1	J2
0	ϕ	ϕ	ϕ	ϕ
1	-	-	{O(1,1)}	{O(1,2)}
2	-	-	{O(2,1)}	{O(2,2)}
3	{O(1,1)}	{O(2,1)}	-	-
4	{O(1,2)}	{O(2,2)}	-	-

A is given by

q	A(q)
0	{{I1, J1}, {I1, J2}, {I2, J1}, {I2, J2}}
1	{{J1}, {J2}}
2	{{J1}, {J2}}
3	{{I1}, {I2}}
4	{{I1}, {I2}}

Definition 3.6: We say that a realization is *without busy-waiting* if, whenever the realization's input lines are without signals for a sufficiently long period of time, the modules and lines internal to the realization eventually stabilize. If a set $\mathcal{M}$ of modules is such that any module in class $\mathcal{C}$ can be constructed without busy waiting, we say that $\mathcal{M}$ is *wbw-universal* for class $\mathcal{C}$.

We next look for sets which are *wbw*-universal.

Definition 3.7: An *arbitrating call* (AC) module has the properties of a call module, except that simultaneous signals on all inputs are permitted.

Theorem 3.3: The set $\{M, S, F, \text{AC}\}$ (modularity 7) is *wbw*-universal for the class of all modules.

Proof: The idea is to realize, without busy-waiting, an

LUn, for arbitrary n, using only elements of the set $\{M, S, \mathrm{AC}\}$. Each locking input line to the LUn module is connected so that the corresponding signal sets a unique S module. After doing so, the signal enters a binary-tree of AC modules which are connected so that only one signal gets through to the next stage. At this stage a circular test of the S modules is performed to determine an input which has been signalled. In order to insure finite-blocking (Condition 11), there must be a way of starting the circular test at any given module. This may be done by a second set of S modules. The circular test terminates with the production of a locking output. Finally, the unlocking input signal sets an S module, then routes a signal back through the tree of AC modules, and finally produces an unlocking output by testing the modules set by the unlocking input. Further details of the construction are left as an exercise for the reader.

Corollary: The set $\{M, G, H, F, \mathrm{AC}\}$ (modularity 6) is *wbw*-universal for the class of all modules.

IV. Implementations

The method of signal flow described in the preceding sections may appear unusual to some readers. It is the purpose of this section to discuss some possible signalling conventions and module implementations which could be used in practice. The manner in which implementations relate to the choice of "atomic" modules will also be discussed.

It should be mentioned that in implementing atomic modules, we *are* permitted to make assumptions about delays. This seems essential. In fact, it can be shown that certain modules, specifically those with "essential hazards" [28], require delays for their implementation. The utility of describing a network in terms of modules, however, is that the modules provide a kind of "sphere of protection" around those parts of the network in which delays are critical. In fact, we may introduce the concept of a *quasi-realization* as being a realization in which certain lines are granted immunity from the delay-insensitivity condition, and then relate this concept to the discussion in previous sections, but we will not do so.

There are three seemingly natural signalling conventions which may be used for communication from a module m to another module m' by a line L.

1) Pulse-m sends a pulse on L. m' must either transmit the pulse immediately, or contain a flip-flop which is set by the pulse, indicating the signal's arrival. In this case, m' has essentially assimilated the pulse when it resets the flip-flop.

2) Symmetric Transition: Line L is considered to have a value of one of two possible levels, 0 and 1. A signal is indicated by a transition from logic value 0 to 1, and from logic value 1 to 0. m' assimilates the signal by acting on the level change.

3) Asymmetric Transition: A signal is indicated by a transition from logic value 0 to 1. The value must be reset to 0 before m produces another signal.

These conventions are summarized in Fig. 18. Each of these types of signalling appears useful in finding simple

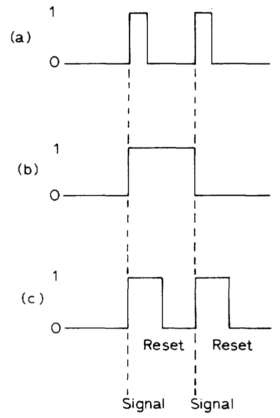


Fig. 18. Signalling conventions: (a) pulse, (b) symmetric transition, (c) asymmetric transition.

implementations of certain modules, but none seems to yield universally simple implementation. Hence it is instructive to discuss conversion from one type of signal to the other. To convert a pulse to a symmetric transition, the scheme shown in Fig. 19 is used. T represents a standard "trigger" flip-flop. To convert a symmetric transition to a pulse, the scheme of Fig. 20 is used. The $\oplus$ symbol indicates the standard "EXCLUSIVE-OR" and Δ indicates a delay whose length is roughly equal to the intended duration of the pulse.

To complete the picture, we must discuss conversion between symmetric and asymmetric transition signalling. We first note that a module employing asymmetric signalling can be viewed as one with symmetric signalling, with every other transition indicating a reset. Hence it suffices to consider the symmetric case only in specifying implementations. Furthermore, due to the necessity of resetting in the asymmetric case, this type of signalling appears to be usable only when every line is paired with an oppositely-directly line which indicates assimilation. In this paired case, a conversion from one convention to the other is shown in Fig. 21, where the component modules are S and M modules using symmetric transition signalling. It is interesting that the same network performs the conversion in either direction.

Under the assumption that a module is serial, several techniques are available for implementation. One can use the Huffman synthesis approach [16] for any of the signalling conventions, the "pulse-mode" technique [22] for pulse signalling, and the "transition logic" [6] analog of pulse mode for transition signalling. Some modules which seem to be naturally implemented using transition logic are shown in Fig. 22. MC denotes the "Muller C" element, an element with memory. An implementation of MC using NOR gates is given in Fig. 23. If both inputs of the MC are the same, then the output is equal to the input. If they are different then the output retains its former value. Hence in Fig. 22(c), assuming that both inputs are initially the same value, no output transition occurs until transitions have occurred on both inputs, which is effectively the behavior of a Join. In Fig. 22(d), an (I, J) pair of transitions cause a transition on the

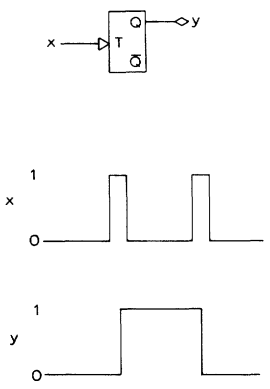


Fig. 19. Converting pulses to transitions using a trigger flip-flop.

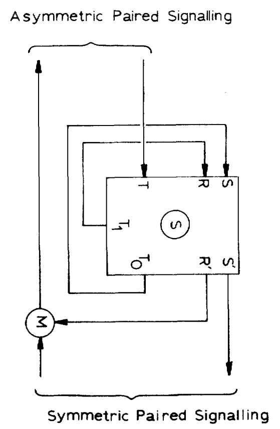


Fig. 21. Conversion between symmetric and asymmetric transition signalling.

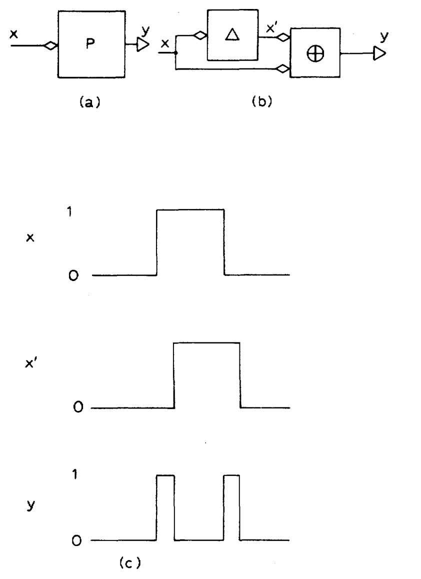


Fig. 20. Converting transitions to pulses: (a) "pulser" representation, (b) implementation, (c) diagram of operation.

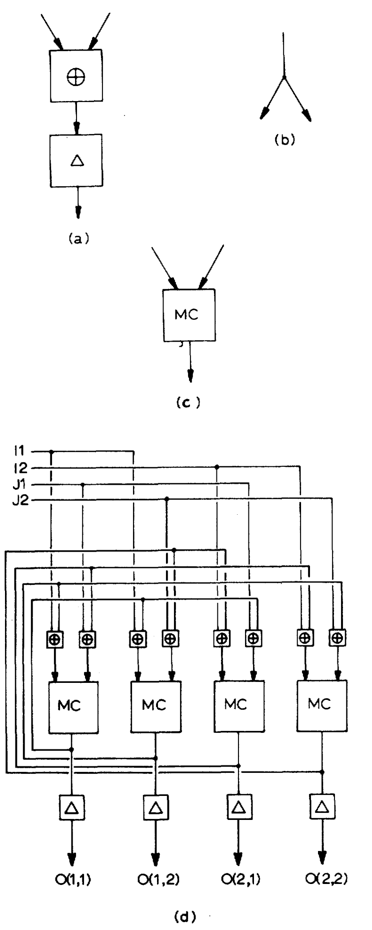


Fig. 22. Modules implemented by transition signalling: (a) merge, (b) fork, (c) join, (d) $J(2, 2)$.

output of exactly one of the MC elements. However a transition has also occurred on one input of two of the other MC elements. The feedback loops have the effect of cancelling these signals, through the use of EXCLUSIVE-OR's. The Δ denotes a delay which is introduced for the purpose of allowing the circuit to stabilize before signalling its environment. The S and ATS modules seem to be naturally suited to implementation using pulse signalling, as shown in Figs. 24 and 25. The reader may wish to observe this by attempting the design using transition-mode signalling. The ATS module implementation shown is still rather complicated and deserves explanation. Referring to Fig. 25, a pulse on the T input causes the value of FF_1 to be gated into FF_2. Then the trigger flip-flop value T_r is gated into FF_1. If R has been pulsed since the last T pulse, the values of FF_1 and FF_2 will then differ and T_0 will be pulsed. Otherwise T_1 will be pulsed. Delay Δ_1 is adjusted so that it is longer than the settling time of FF_2. It is desirable to adjust Δ_2 so that it is longer than the settling time of FF_1.

Unfortunately, there is a problem with this implementation. Since the relative timing of R and T is not constrained in any way, the output of T_r may be changing when gating into FF_1 occurs. This may cause a nonstandard pulse to be applied to FF_1, which in turn may enter a metastable state or which may oscillate, and in either case not stabilize in the normal settling time. Experience with these

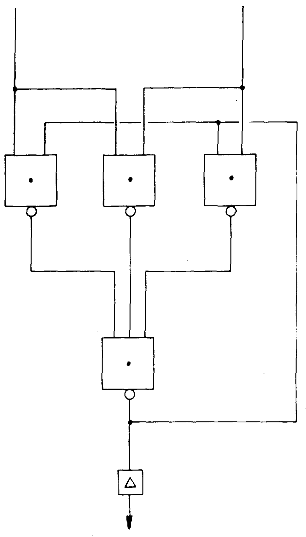


Fig. 23. Implementation of MC using NOR gates.

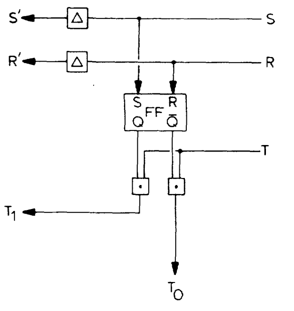


Fig. 24. Implementation of S module using pulse signalling.

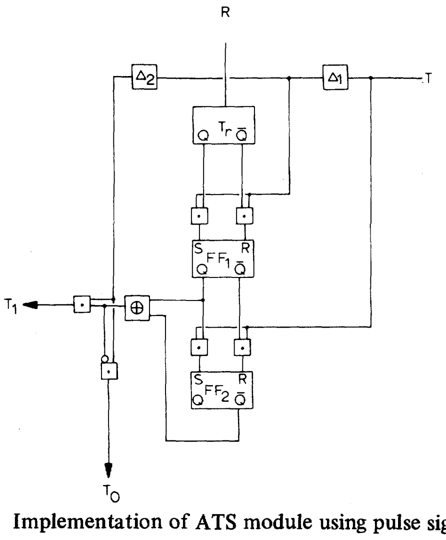


Fig. 25. Implementation of ATS module using pulse signalling.

phenomena has been reported [5], [9], [21], [31] but apparently they are not widely recognized. As time increases after the occurrence of this phenomenon, it can be observed empirically that the probability that the flip-flop has not stabilized grows smaller. Hence making Δ_2 large increases the reliability, but by no means to 100 percent. Note that by the arbitration condition, it does not matter for this module in which state FF_1 finally settles because if it does not reflect the true state of T_r, it certainly will on the next T pulse, as R cannot occur again until the T_0 output occurs, by the specification of the module's operation.

To the author's knowledge, no totally "reliable" solution to this problem exists. A recent workshop was directed to the problem without definite conclusions [32]. The interested reader may also wish to compare the above cited work to [33], [34]. Such problems appear to occur whenever the arbitration condition (Condition 3) does not hold in a trivial way; that is, the output produced depends on the order of assimilation of simultaneous inputs. For example, the implementation of the Join module as shown in Fig. 23 does not appear to have this flaw. We reemphasize at this point that it is for such reasons that we would like to do without nontrivial arbitration whenever possible.

V. Generalizations and Relation to Other Models

Discussed in this section are some generalizations of the model presented in previous sections and the relation of these generalizations to other asynchronous computation models. We first consider a generalization which allows multivalued signals on lines. This is motivated by the problem of transmitting "data" from a module m to a module m' which can take on one of $n > 1$ values. In the present scheme, this value could be encoded as a "unary-encoded" signal on one out of n lines. A more efficient binary encoding could be used which first encodes the data in binary, then represents each bit as a signal on one out of two lines, requiring at most $2 \lceil \log_2 n \rceil$ lines. This method is discussed in [3], [14], [23]. A still more efficient method is described in [27], however it is not amenable to description in the present model because assumptions about delay times are involved. The idea is that the data are encoded in binary using $\lceil \log_2 n \rceil$ lines, but an extra line accompanies these lines. A signal on this extra line indicates when data are being transmitted. This signal lags the data slightly, so that when it arrives at m', it is certain that the data lines have their proper values, assuming the delay in all lines in the bundle are the same. Other methods for implementing multiple-valued asynchronous signals are discussed in [3]. If the model presented here were extended so that lines could hold multiple-valued signals, then this implementation could be represented conveniently. Here then is a case in which it may be beneficial to extend the present model, since it does not succinctly represent the idea of data transmission. Related to this are practical considerations for bus structures, which are discussed in [35].

Another generalization is possible in which concurrent input signals are given relative *priorities* which would govern their order of assimilation by a module. It may or may not be desirable to remove the finite blocking condition in this case.

Another generalization of the present model would allow a line to be accessible by more than two modules. This differs from our intuitively-accepted idea of a line representing a signal path, but nevertheless could be analyzed mathematically. Such a generalized line could be realized by making each line correspond to a module. However we then have to have "lines" to connect these modules, and seem to have travelled in a complete circle. Such generalizations are discussed in [15], [26], in which the lines are referred to as "places." Related to this, and the previous generalization, are the "links" of [29] and, in a loose sense, the "cells" or "locations" of [18], [19].

It may be noted that for each of the modules described herein, there is a representation of the state-transition table in terms of a "Petri net," as described in [15], provided that we impose on the latter a condition which is the equivalent of the finite-blocking property. This means that Petri nets are, in a sense, universal for the representation of such structures. Other investigations have shown that Petri-nets, properly restricted, can be implemented using modules such as the ones described here [25], [30].

Another variation allows a line to be connected to only two modules, but allows it to be both input and output to each [4]. Condition 1 is therefore removed. This temporarily defies intuition, until we view it in the light of paired signalling conventions using transitions, such as discussed in [11], [23]. The idea is that each line is really two oppositely directed lines. To avoid confusion, we capitalize the first kind of line. A "Line" is then said to be in one of two states: *active* if the lines have different values, and *idle* if they have the same values. Thus either module in an interconnection can change the state of the Line by changing the value on the line directed from it. Still more complex Lines are discussed in [25], [30].

One final generalization to be mentioned is one in which each line acts as a queue. That is, Condition 6 is removed. A restricted case in which the signals are 1-valued corresponds to the "marked graph" model [15]. In the case in which signals can be multivalued, we have the models presented in [1], [17].

It has not been our purpose to generate an all-encompassing model in this paper, but rather to generate a model which can represent a wide class of systems and still be reasonably close to a hardware implementation. How well this goal has been achieved is summarized in the next section.

Conclusions and Discussion

A set of modules has been presented from which may be synthesized a wide class of modules. Furthermore, an effective way to perform this construction is indicated. Unfortunately, such constructions may produce somewhat unwieldy end results, as evidenced by Fig. 17, for example. Part of the problem appears to be due to the general need for arbitration. It is this property that adds some of the complexity to the general construction presented here, namely the requirement of the lock-unlock module. Another undesirable property is that the present implementation may be unnecessarily slow, since only one input can be affecting the state-changing network at a time. Additional considerations are due to the fact that nontrivial arbitration seems inherently more difficult to implement, as discussed in Section IV. Hence it seems desirable to be able to determine whether nontrivial arbitration is required or not, and if not, then to find a realization which does not use any atomic elements with nontrivial arbitration. For example, Fig. 21 and the corresponding explanation in Section IV depicts a much simpler implementation of the $J(2, 2)$ than shown in Section III. The MC element forms the only inherently-parallel basis of this construction. The author conjectures that the set $\{M, S, F, J\}$ is universal for the class of modules not requiring arbitration, however the conjecture is imprecise, since we do not yet have a formal statement of what it means to "require arbitration." The present exposition is not proposed as an end, but rather a point of departure for future investigation. In particular, the following questions remain to be answered.

1) Are there other universal sets which are simpler than those presented?

2) Are there other universal sets which provide for less complex constructions, although the modules in the set may be more complex than those presented?

3) Are there restricted classes which yield simpler realization methods?

4) What further can be said of the generalizations presented in Section V in relation to the present model? Are further generalizations, other than those mentioned, of significance?

The author is convinced that the answer to questions concerning universality will not be found in traditional investigations of universal switching elements, as such investigations have not strived for the properties deemed necessary here, e.g., speed-independence, delay-insensitivity, etc. The work by Muller [24] perhaps comes closest to these desiderata, but since it was concerned with "autonomous" and, in a sense, "serial" networks, it is not clear that the results are relevant to the problem presented. Another discussion, more similar to the present one, is found in Petri [26]. A preliminary negative result is cited in [10].

Finally we mention again that problems of this nature also relate to problems encountered in "concurrent programming" of computers. Since the flow of control in asynchronous modular networks is similar to that in concurrent programming, answers to questions presented here can yield answers to the questions of the adequacy of programming language constructs in effecting parallel computations. The questions we ask here also seem to ask something about the fundamental nature of asynchronous concurrent processes, apart from the physical realm of one implementation. For example, although the mutual exclusion problem [12], [13], [20], a form of arbitration, appears to be solved for software processes, what has in fact happened is that the problem has been pushed to a lower level. That is, if the processors are truly asynchronous, there must be an arbitrating device between them and the memory. Many readers will undoubtedly be aware of other such similarities.

Acknowledgment

The author wishes to thank the anonymous referee and J. Banning and D. Carroll for their comments and suggestions.

References

[1] D.A. Adams, "A model for parallel computations," in *Parallel Processor Systems, Technologies, and Applications*. Washington, D.C.: Spartan, pp. 311-33, 1970.

[2] S.M. Altman and A.W. Lo, "Systematic design for modular realization of control functions," in *1969 Spring Joint Comput. Conf. Proc., AFIPS Conf. Proc.*, vol. 34. Montvale, N.J.: AFIPS Press, pp. 587-595, 1969.

[3] D.B. Armstrong, A.D. Friedman, and P.R. Menon, "Design of asynchronous circuits assuming unbounded gate delays," *IEEE Trans. Comput.*, vol. C-18, pp. 1110-1120, Dec. 1969.

[4] J. Bruno and S.M. Altman, "Asynchronous control networks," *IEEE Trans. Comput.*, vol. C-20, pp. 629-638, June 1971.

[5] I. Catt, "Time loss through gating of asynchronous logic signal pulses," *IEEE Trans. Electron. Comput.* vol. EC-15, pp. 108-111, Feb. 1966.

[6] Y.H. Chuang, "Transition logic circuits and a synthesis method," *IEEE Trans. Comput.*, vol. C-18, pp. 154-168, Feb. 1969.

[7] W.A. Clark, "Macromodular computer systems," in *1967 Spring Comput. Conf., AFIPS Conf. Proc.*, vol. 30. Washington, D.C.: Thompson, pp. 335-336, 1967.

[8] W.A. Clark, M.J. Stucki, and S.M. Ornstein, "A macromodular approach to computer design," Washington Univ. Comput. Res. Lab., St. Louis, Mo., Tech. Rep. 1, Feb. 1966.

[9] R.G. Couranz, "An analysis of binary circuits under marginal triggering conditions," D.Sc. dissertation, Dept. Electrical Engineering, Washington University, St. Louis, Mo., January 1970.

[10] J.B. Dennis and S.S. Patil, "Speed independent asynchronous circuits," presented at the University of Hawaii Conf., 1971.

[11] J.B. Dennis, "Modular, asynchronous control structures for a high performance processor," in *Rec. Proj. Mac Conf. Concurrent Systems and Parallel Computation*, ACM, N.Y., pp. 55-80, 1970.

[12] E.W. Dijkatra, "Solution of a problem in concurrent programming control," *Commun. Assoc. Comput. Mach.*, vol. 8, p. 569, Sept. 1965.

[13] —, "Co-operating sequential processes," in *Programming Languages*, F. Genuys, Ed. New York: Academic Press, 1968.

[14] J. Goldberg and R.A. Short, "Antiparallel control logic," in *5th IEEE Annu. Symp. Switching Circuit Theory and Logical Design*, Oct. 1964, pp. 30-39.

[15] A.W. Holt and F. Commoner, "Events and conditions," in *Rec. Proj. MAC Conf. on Concurrent Systems and Parallel Computation*, ACM, N.Y., 1970, pp. 3-52.

[16] D.A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Institute*, vol. 257, no. 3, pp. 161-190, Mar. 1954; *ibid.*, no. 4, pp. 275-303, Apr. 1954.

[17] R.M. Karp and R.E. Miller, "Properties of a model for parallel computations: determinacy, termination, queueing," *SIAM J. App. Math.*, vol. 14, pp. 1390-1411, Nov. 1966.

[18] —, "Parallel program schemata," *J. Comp. Syst. Sci.*, vol. 3, pp. 147-195, May 1969.

[19] R.M. Keller, "On maximally parallel schemata," in *IEEE Conf. Rec., 11th Annu. Symp. Switching and Automata Theory*, pp. 32-50, 1970.

[20] D.E. Knuth, "Additional comments on a problem in concurrent programming control," *Commun. Assoc. Comput. Mach.*, vol. 9, pp. 321-322, May 1966.

[21] W. Littlefield and T. Chaney, "The glitch phenomenon," Washington Univ. Comput. Res. Lab., St. Louis, Mo., Tech. Memo 10, Dec. 1966.

[22] E.J. McCluskey, *Introduction to the Theory of Switching Circuits*. New York: McGraw-Hill, 1965.

[23] D.E. Muller, "Asynchronous logics and application to information processing," in *Switching Theory in Space Technology*. Stanford, Calif.: Stanford University Press, 1963.

[24] —, "The general synthesis problem for asychronous digital networks," in *IEEE Conf. Rec.*; also in *8th Annu. Symp. Switching and Automata Theory*, 1967, pp. 70-82.

[25] S.S. Patil, "Coordination of asynchronous events," Mass. Inst. Tech., Cambridge, Mass., Rep. MAC-TR-72, Proj. MAC, June 1970.

[26] C.A. Petri, "Communication with automata," Suppl. 1 to Tech. Rep. RADC-TR-65-377, vol. I, Griffiss Air Force Base, N.Y., 1966.

[27] M.J. Stucki, S.M. Ornstein, and W.A. Clark, "Logical design of macromodules," in *1967 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 30. Washington, D.C.: Thompson, pp. 357-364, 1967.

[28] S.H. Unger, *Asynchronous Sequential Switching Circuits*. New York: Wiley, 1970.

[29] F.L. Luconi, "Asynchronous computational structures," Mass. Inst. Tech., Cambridge, Mass., Rep. MAC-TR-99 (thesis), Project MAC, 1968.

[30] F.C. Furtek, "Modular implementation of Petri nets," M.S. thesis, Dep. Elec. Eng., Mass. Inst. Tech., Cambridge, Mass., Sept. 1971.

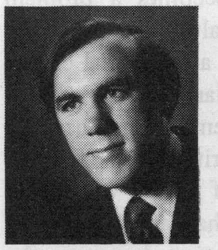
[31] T.J. Chaney and C.E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.* (Corresp.), vol. C-22, pp. 421-422, Apr. 1973.

[32] "Workshop on synchronizer failures," Washington Univ., St. Louis, Mo., Apr. 27-28, 1972.

[33] S.H. Unger, "Asynchronous sequential switching circuits with unrestricted input changes," *IEEE Trans. Comput.*, vol. C-20, pp. 1437-1444, Dec. 1971.

[34] W.W. Plummer, "Asynchronous arbiters," *IEEE Trans. Comput.*, vol. C-21, pp. 37-42, Jan. 1972.

[35] J.P. Banning, "Asynchronous modular systems," Comput. Sci. Lab., Dep. Elec. Eng., Princeton Univ., Princeton, N.J., TR-116, Jan. 1973.



Robert M. Keller was born in St. Louis, Mo., June 12, 1944. He received the B.S. degree in engineering science in 1966 and the M.S. degree in electrical engineering in 1968, both from Washington University, St. Louis. From 1967 to 1970 he was an NSF Graduate Fellow at the University of California, Berkeley, and received the Ph.D. degree in 1970.

Since 1970 he has been an Assistant Professor in the Department of Electrical Engineering at Princeton University, Princeton, N.J., where he has taught in the areas of computer organization, operating systems, switching, automata, and language theories, and parallel computation. His research interests include the theory of parallel and asynchronous computation and its application.