STAN-CS-70-160

SU-SEL-70-014

A Model For Parallel C'omputer Systems

by

T. H. Bredt
E. J. McCluskey

Technical Report No. 5

April 1970

This work was supported in part by
the Joint Services Electronic Programs
U. S. Army, U. S. Navy, and U. S. Air Force
under Contract N-00014-67-A-0112-0044 and
by the National Aeronautics and Space
Administration under Grant 05-020-337.

DIGITAL SYSTEMS LABORATORY

STANFORD ELECTRONICS LABORATORIES

STANFORD UNIVERSITY . STANFORD, CALIFORNIA

STAN-CS-70-160 SEL -70-014

A MODEL FOR PARALLEL COMPUTER SYSTEMS

T. H. Bredt

E. J. McCluskey

April 1970

Technical Report No. 5

DIGITAL SYSTEMS LABORATORY

Stanford Electronics Laboratories Computer Science Department
Stanford University

Stanford, California

This work was supported in part by the Joint Services Electronic Programs U. S. Army, U. S. Navy, and U. S. Air Force under Contract N-00014-67-A-0112-0044 and by the National Aeronautics and Space Administration under Grant 05-020-337.

Technical Report No. 5 April 1970

A MODEL FOR PARALLEL COMPUTER SYSTEMS

T. H. Bredt

E. J. McCluskey

Digital Systems Laboratory
Stanford Electronics Laboratories
Stanford University, Stanford, California

ABSTRACT

A flow table model is defined for parallel computer systems. In this model, fundamental-mode flow tables are used to describe the operation of system components, which may be programs or circuits. Components communicate by changing the values on interconnecting lines which carry binary level signals. It is assumed that there is no bound on the time for value changes to propagate over the interconnecting lines. Given this delay assumption, it is necessary to specify a mode of operation for system components such that input changes which arrive while a component is unstable do not affect the operation of the component. Such a mode of operation is specified. Using the flow table model, a new control algorithm for the two-process mutual exclusion problem is designed. This algorithm does not depend on the exclusive execution of any primitive operations used in its implementation. A circuit implementation of the control algorithm is described.

TABLE OF CONTENTS

List of Tables

List of Figures

INTRODUCTION

A computer system can be viewed as a collection of programs, which are sets of instructions to be executed by processors, and logic circuits, sets of logic gates and flip-flops interconnected by wires. Much effort has been expended on the development of formal design procedures for logic circuits resulting in the body of knowledge known as switching theory. This theory provides procedures for circuit *analysis*, the determination of-what a particular circuit does, and circuit *synthesis*, the design of a circuit to accomplish some task. Unfortunately, designers of computer systems do not have similar techniques available to them. These techniques would allow programs and circuits to be treated in common framework and would make it possible to analyze a system formally, without expensive testing and debugging, to determine what the system does. They would help a designer decide whether a circuit or program implementation is most appropriate. The ability to consider both hardware and software implementations is particularly important in the design of *operating systems* where there is often a choice between a program or circuit implementation. In this paper, a model is proposed for parallel computer systems which it is hoped will aid in the fulfillment of these objectives. The model depends on the use of *fundamental-mode flow tables*, used previously to design sequential circuits [23], to describe the operation of both program and circuit components.

To motivate the need for such a model and to illustrate the difficulties involved in describing the operation of a computer system, let us consider a well known problem which occurs in multi-processor computer systems. This problem, called the _mutual exclusion_ or _interlock problem_, occurs when two or more processes are active simultaneously. Such processes are called _concurrent processes_. The use of the term _process_ implies that some component in the system is active, performing a task. The activity of the component distinguishes a process from a processor. A _processor_ is an entity which has the capability of performing a task. A further discussion of the distinction between a process and a processor is given by Dennis and Van Horn [5], Dijkstra [6 , 7 , 8], Saltzer [26], and Lampson [19]. In the mutual exclusion problem, each process is assumed to contain certain special operations in a portion of the process known as a _critical section._ The processes usually represent the execution of programs containing infinite loops in which they enter, leave, and then re-enter their critical sections. The mutual exclusion problem requires the specification of a control mechanism to prevent two or more processes from entering their critical sections simultaneously. In addition, it must be guaranteed that, if one process wants to enter its critical section, the process cannot be blocked by other processes entering, leaving, and then re-entering their critical sections. Knuth [18] has shown that this latter possibility exists in one control algorithm proposed for the mutual exclusion problem. The exact nature or content of a critical section is not important in the development of a solution to the problem. Typically, critical sections modify common storage files or system tables. A precise statement of the mutual exclusion problem for two processes is given below.

Problem: (Mutual Exclusion)

Given two concurrent processes, each containing a critical section, control these processes so that the following two restrictions are always satisfied.

Restriction 1: At most one process is in a critical section, at any instant.

Restriction 2: If a process wants to enter its critical section, it is eventually allowed to do so.

This problem is slightly different from the one Dijkstra posed. He wanted to ensure that the decision as to which process enters its critical section cannot be postponed indefinitely. While a decision must always be made, a particular program may be blocked indefinitely.

Many solutions have been proposed to this problem [5 , 6 , 7, 8 , 18 , 19]. Most of these solutions depend on the existence of special instructions which are executed whenever a process wants to enter its critical section. Examples of these instructions are the Test-and-Set instruction which- is a machine instruction for the IBM 360 series computers [14], the LOCK and UNLOCK statements for high level languages like FORTRAN and ALGOL discussed by Dennis and Van Horn [5], and the P and V operations proposed by Dijkstra [7 , 8]. Two programs using Dijkstra's P and V operations to achieve exclusive access to their critical sections are shown in Table 1. The programs are specified in a version of the ALGOL programming

Table 1. Dijkstra's P, V Solution to the Mutual Exclusion Problem

```
BEGIN INTEGER S;        s :=1;
    PARBEGIN
        PROCESS 1:  BEGIN
                    L1:   P(S);
                          CRITICAL SECTION 1;
                          V(S);
                          REMAINDER OF PROCESS 1;
                          GO TO L1;
                    END;
        PROCESS 2:  BEGIN
                    L2:   P(S);
                          CRITICAL SECTION 2;
                          V(S);
                          REMAINDER OF PROCESS 2;
                          GO TO L2;
                    END
    PAREND
-END.
```

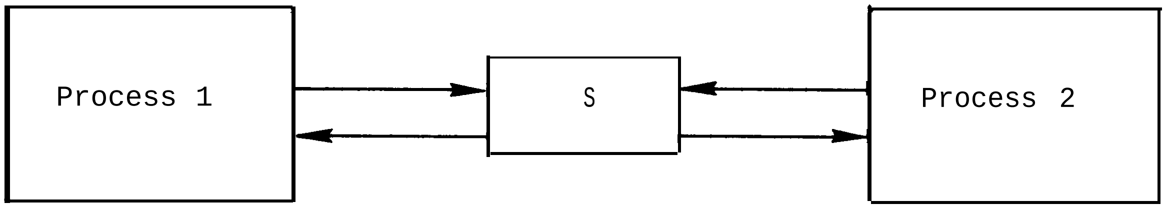
language. The integer variable S is called a semaphore. We will describe the solution for two processes although it can be generalized to handle an arbitrary number of processes. For two programs, the semaphore variable takes on only two values, 1 and 0. When S=1, neither process is in its critical section and when S=0, one of the processes is in its critical section. The identifiers PARBEGIN and PAREND were introduced by Dijkstra to denote that every statement appearing between these two identifiers can be executed concurrently. This is Dijkstra's version of the FORK and JOIN statements proposed by Conway [4] and others. The P operation or statement is performed on a semaphore variable and has the following effect. If the value of S is 1, S is set to 0 and the next statement is executed. If s is 0, the process must "wait" until S becomes 1 before it may proceed. The V operation is also performed on a semaphore variable and increases the value of the variable by 1. For two processes, V(S) is equivalent to setting the value of S to 1. There are two possible forms of activity while a process waits for a semaphore to become 1. The process may go into a tight loop repeatedly executing the P(S) operation until S becomes 1. This form of waiting is called "busy waiting" since a processor must be assigned to the process continuously. In the other form of waiting, the process is added to a queue associated with the semaphore where it resides until the semaphore becomes 1. In this case, when a V operation is performed, the queue for the appropriate semaphore variable must be examined and any process which is eligible to proceed restarted. This form of waiting allows the processor associated with the idle process to be freed to execute other processes.

Dijkstra makes the following two assumptions about the P and V operations.

1. The P and V operations are indivisible. That is, it is impossible for one P or V operation to be initiated and then for another P or V operation to be initiated before the first is complete.
2. P and V operations may not be executed simultaneously.

Given these two assumptions, Dijkstra proceeds to analyze the behavior of the system containing the two processes and concludes that the mutual exclusion problem has been correctly solved although 'he does not claim that the analysis presented is formal.

Dijkstra's conclusions are hard to accept for several reasons. First, he has not said enough about the system environment to determine if the P and V operations will work. The situation is described in Fig. 1. Each process is able to read and change the value of the semaphore S. Dijkstra does not say whether he intends the system to operate in a synchronous manner under the control of a master clock or whether the components in some way operate independently. It is important to account for delays which may be present in the environment. These delays may be in the lines over which , processes access the semaphore variables and also in the processes themselves. It is possible in a physical system for operations to occur simultaneously and this possibility should not be dismissed



Figure'l. System configuration for Dijkstra's mutual exclusion problem solution.

by simply assuming simultaneous interactions do not occur. Any analysis procedure or model should consider all possible variations in timing of system operations. Another objection is that Dijkstra has solved the mutual exclusion problem for programs by presenting another mutual exclusion problem which must be solved in the logic circuits of the system. In order to guarantee that the assumptions about the indivisibility of the operations and the absence of simultaneous P and V executions hold, another mutual exclusion problem, nearly identical to the one presented earlier mutual exclusion must be solved. In fact, the statement of the problem given will suffice if we replace the words "critical section" by "P or V operation".

We do not intend to be overly critical of Dijkstra's work. Other published solutions to the mutual exclusion problem depend on the exclusive execution of some primitive operation. A possible exception is the work of Clark [3]; however, we are not aware of the details of their implementation. We feel that there remain unanswered questions and a need for more work in this area. In this and subsequent papers, we discuss a new approach to the study of parallel systems. Methods based on the use of <u>flow tables</u> are presented which allow circuits and programs to be described in a common framework. These methods permit the formal analysis of the operation of systems of the type we have just described and make it possible to consider the effects of delays. They are applicable in the synthesis of solutions to problems such as the mutual exclusion problem. A mode of operation is described for parallel systems which does not depend on synchronous operation or the exclusive execution of any primitive operations.

PARALLEL SYSTEMS

A diagram of a portion of a possible system configuration is , shown in Fig. 2. The square boxes represent system <u>components</u>. These components may be <u>programs</u> or <u>circuits</u>. The operation of a circuit or the execution of a program is referred to as a <u>process</u> in the sense used in the introduction.. Some of the components may act as <u>control mechanisms</u> which enable and disable other components. Each interconnecting <u>line</u> represents a physical wire which carries a binary level signal. Each line has associated with it a direction of propagation for transmission of signal value changes from the output of one component to the input of another. The direction of propagation is indicated by arrowheads in the system diagram.

The operation of the system can be described in a general way as follows. Whenever a process wants to perform an operation that could affect other processes, the process requests permission to perform the operation from a control mechanism. The permission has the form of an enabling signal sent from the control mechanism to the process. It is the responsibility of the control mechanism to ensure that no situation arises that violates restrictions placed on system operation. One control mechanism can seek authorization

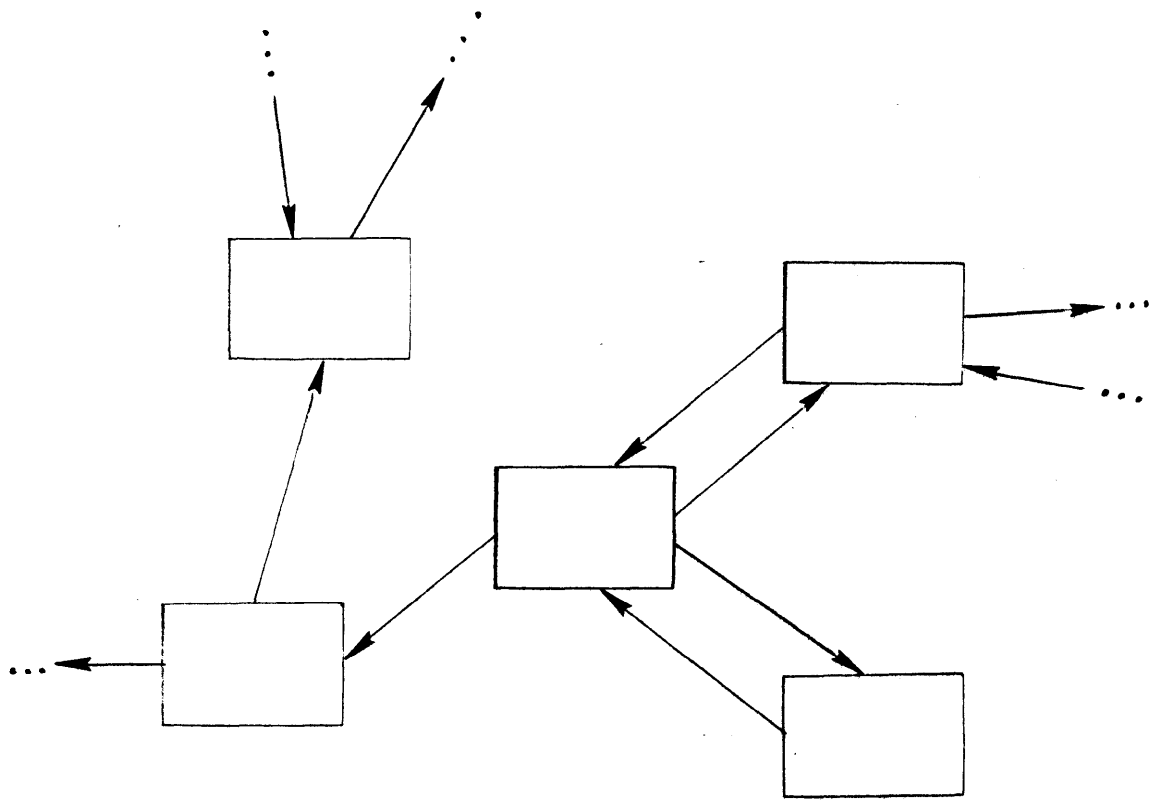


Figure 2. Typical parallel system configuration.

for an action from another control mechanism and there need be no central control mechanism responsible for the operation of the entire system.

The general form of a system component is shown in Fig. 3. The component has n input lines and m output lines. Each input line has an associated input variable x_i, $i = 1, \ldots, n$. The values of the input variables define the input state of the component. , Each input variable has two possible values, 0 and 1 . Each component produces outputs which are also binary signals. Each output line has an associated output variable or excitation variable z_i, $i = 1, \ldots, m$. The values of the output variables define the output state of the component. Each input and output line is connected to exactly one other component.

FLOW TABLES

In any model of computer systems, it is necessary to be able to describe precisely the operation of each system component. Many models of parallel computations and parallel computer systems have been proposed in which functions are used to describe component behavior [1, 2, 15, 16, 17, 20, 21, 22, 24, 25, 27]. These functions define mappings of component input states into output states. This approach has the advantage of complete generality in the types of component behavior that can be described. Any operation that can be described by a mathematical function can be represented.

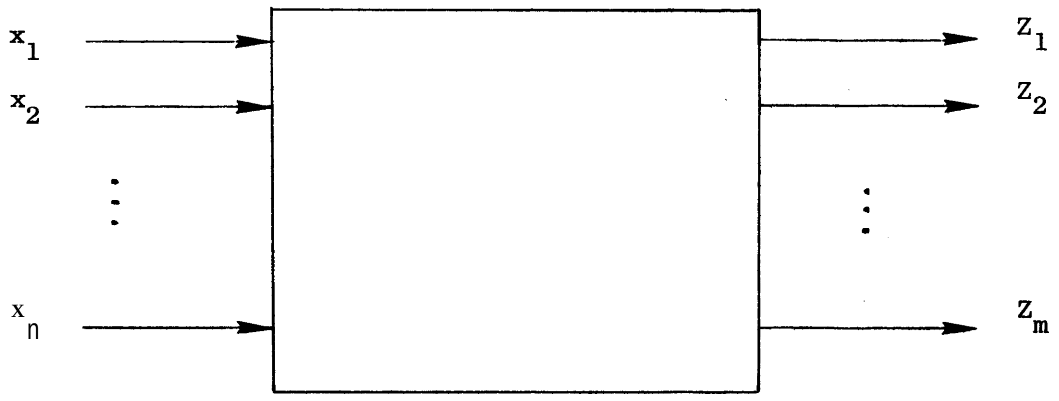


Figure 3. General form of a component.

There are several disadvantages associated with this approach however. First, most of the interesting properties of the system, such as whether or not the system ever halts or whether one system is equivalent to another, are undecidable. That is, there do not exist algorithms which determine for any arbitrary system if the system ever halts or for two arbitrary systems, if they do the same thing. A second disadvantage is that the function does not necessarily describe the program or circuit that implements the function.

The model we propose uses <u>flow tables</u> rather than functions to describe component operation. Flow tables were first introduced by Huffman [13] and are normally used in the design of sequential switching circuits [23]. There is a direct correspondence between a sequential circuit and a flow table. In another paper, we show that flow tables can be used to describe computer programs and give procedures for constructing the program that corresponds to a given flow table and the flow table that corresponds to a given program. Thus there is a direct correspondence between a program or circuit implementation and flow table used to describe the implementation. This correspondence is two-way. A program or circuit can be <u>analyzed</u> to determine what it does and a flow table solution to a problem can be <u>synthesized</u> or designed and then a program or circuit implementation produced.

A possible disadvantage of using flow tables is that only mathematical functions which require finite internal storage can be described. For example, given a component with two inputs x_1 and x_2 it is impossible to use a flow table to determine if an arbitrary number of 1-0-l transitions on the x_1 input is always followed by exactly the same number of 1-0-l input transitions on the x_2 input. We are interested in the study of interactions among components rather than the types of problems that can be solved using these systems. We feel these interactions are best studied in a model which requires finite storage and will show that problems which arise in interconnecting components can be solved using finite techniques. We view a system as a finite collection of components which have a finite number of interconnections and therefore mathematical properties such as termination and equivalence are decidable.

Associated with each component, as shown in Fig. 3, is a flow table of the form shown in Table 2. This table has 2^n columns, one column for each possible input state and r rows where each row represents an internal state of the component. Each internal state is designated by a unique integer number $(1,2,\ldots,r)$. The table entry designated by an internal-state input-state pair specifies the next internal state of the component. If the next-state entry is the same as the present internal state, the entry is called a stable entry and the component (flow table) is said to be in a stable state or stable. If the next-state entry is not the same as the present internal state, the entry is called an unstable entry and the component (flow table) is said to be in an unstable state or unstable. An output state is associated with each internal state. While it is possible for the output state to depend on the

Table 2. General Form of a Flow Table

Input State

$x_1x_2 \; . \; . \; . \; x_n$

Output State

Internal State			00...1		11...1			$Z_1Z_2\ldots Z_m$
i			$\textcircled{i}$		j			10 . . . 0
j					$\textcircled{j}$			1 1 . . . 0
r								

S (next state)

input state as well as the internal state [23], this will not be done in this paper. A flow table must satisfy the following restriction. Every unstable entry must specify a next internal state entry which is stable. Thus the table in Table 3a is a flow table but the table of Table 3b is not. To be precise, the following definition of a flow table is given.

-Definition 1:

A flow table is a table with 2^n columns, one for each input state, and r rows, one for each internal state. Associated with each row is an output state. Each unstable entry must specify a next internal state which is stable.

As a consequence of the fact that the output state is associated with an internal state and since each unstable entry leads directly to a stable entry, it follows that each output variable may change value at most once during any internal state transition. In order to describe situations such as in Table 3b, we define a state table.

Definition 2:

A state table is a table which is identical to a flow table except that it is not required that every unstable entry specify a next internal state that is stable.

Table 3. a) Flow Table Example b) Table Which is Not a Flow Table

	x		
	0	1	Z
1	①	2	0
2	3	②	1
3	③	4	1
4	1	④	0

a)

	x		
	0	1	Z
1	①	2	0
2	3	3	1
3	4	③	1
4	④	④	0

b)

Of course every flow table is also a state table. The distinction between flow tables and state tables has been made previously [23]. Others do not make this distinction; state tables are not introduced and the term normal flow table is used to describe the case where every transition leads directly to a stable state and each output variable changes at most once during each internal state transition [10 , 11 , 28].

DELAY ASSUMPTIONS

The following assumptions are made about physical delays present in a parallel system.

Assumption 1:

The time for a value change to propagate from a component output to a component input (the line delay) is finite and unbounded.

Assumption 2:

Within a component, the delays are finite and bounded.

The intent of Assumption 1 is that line delays cannot be controlled. If a "pulse" or short 1 value is produced at a component output, it is not assumed that this value necessarily must propagate to a component input. The consequences of these assumptions are explored in this and subsequent papers. It should be noted that if all delays are assumed to be bounded, a "synchronous" solution to the mutual exclusion problem can be obtained in which the maximum delay time is used to determine the basic cycle time for the system. Our line delay assumpis different from that made in other models where line delays are either assumed to be bounded or zero [1, 15, 16, 17, 20, 21, 22, 24, 25, 27].

A FLOW TABLE SOLUTION FOR THE TWO-PROCESS MUTUAL EXCLUSION PROBLEM

We now return to the two-process mutual exclusion problem discussed in the introduction and use flow table methods to design a solution or control algorithm for this problem. The system configuration is shown in Fig. 4. The variables shown are the input variables for each component. The interpretation of the variable values is given in Table 4. Suppose x_1 and z_1 both have the' value 0. When process 1 wants to enter its critical section (CS1), it sets the value of its output variable X_1 to 1. The 1 value eventually reaches the control mechanism input. The control mechanism sets the value of z_1 to 1. This value propagates to the input of process 1, enabling the process to enter its critical section. The sequence of actions on the part of process 1, just described in words, can be described by a flow table. Such a flow table is shown in Table 5a. The process is initially in internal state 1 with input state z_1=0. Internal-state input-state combination will be denoted by a pair of states separated by a dash (-), in this case, 1-0. The 1-0 entry in Table 5a is 2 indicating that eventually this process enters internal state 2. The output state for internal state 2 is X_1=1. The 2-0 entry is $\textcircled{2}$, a stable entry. The process remains in this stable state until the input transition z_1: $0 \rightarrow 1$ occurs. We assume that a process does

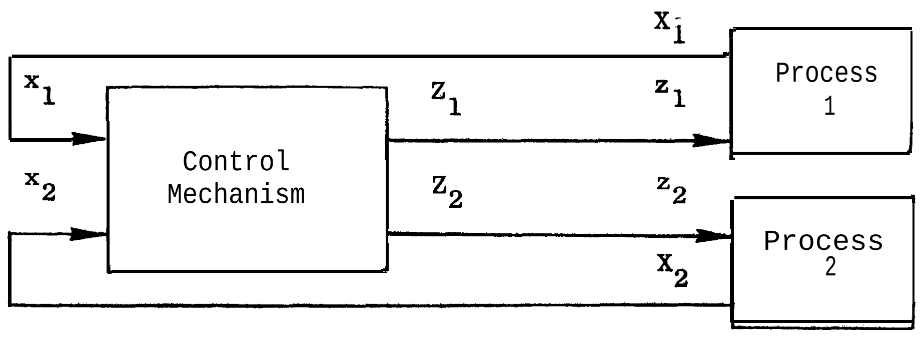


Figure 4. System configuration for the two-process mutual exclusion problem.

Table 4. Interpretation of Variable Values for Fig. 4 (i=1,2)

$x_i = 1$:	process i wants to enter critical section i (CSi) or process i is in CSi
$x_i = 0$:	process i does not want to enter CSi <u>and</u> process i is not in CSi
$z_i = 1$:	process i may enter CSi
$z_i = 0$:	process i may not enter CSi

Table 5. Design of a Flow Table for Process 1.

	z_1		
	0	1	x_1
1	2		0
2	$\textcircled{2}$		1

S

(a)

	z_1		
	0	1	x_1
1	2	$\textcircled{1}$	0
2	$\textcircled{2}$	1	1

S

(b)

Table 6. Flow Table for Process 2

	z_2		
	0	1	x_2
1	2	$\textcircled{1}$	0
2	$\textcircled{2}$	1	1

S

not remain in its critical section indefinitely. Therefore the 2-l entry must be unstable and should be 1 indicating that the process eventually return to internal state 1 where X_1 is set to 0. The unstable transition does not necessarily occur in a short time and, in general, a substantial portion of the operating time of a component may be spent in unstable transitions. The l-l entry in the flow table must be $\textcircled{1}$ This is necessary to ensure that the control mechanism recognizes that process 1 has left its critical section. The control does this by setting Z_1 to 0 which eventually enables process 1 to start its cycle again. The complete flow table for process 1 is given in Table 5b. The corresponding flow table for process 2 is given in Table 6.

If the l-l entry in Table 5b were 2, the table of Table 7 would be obtained. This table is a state table and not a flow table since the 2-l entry does not specify a stable state. If this table described the behavior of process 1, it would be possible for <u>both</u> restrictions on the mutual exclusion problem to be violated. First, the X_1: $1 \rightarrow 0 \rightarrow 1$ transition could be so short that the 0 value never is recognized by the control. Thus, if process 2 desires to enter its critical section, the control would never realize that process 1 was finished and process 2 would never be enabled violating Restriction 2. On the other hand, if process 2 desires to enter its critical section but is not enabled (x_2 = 1 and z_2 = 0) and the 0 value for xl appears momentarily and is recognized by the control, the control may disable process 1 ($Z_1 \rightarrow 0$) and enable process 2 ($Z_2 \rightarrow 1$).

Table 7. An Improper State Table

	z_1		
	0	1	x_1
1	2	2	0
2	②	1	1
	S		

But process 1 can re-enter its critical section before z_1 becomes 0. In this case, both processes would be in their critical sections simultaneously violating Restriction 1. This informal analysis is not intended to replace more formal analysis procedures to be presented in a later paper. However, it does point out the advantages of the flow table in making the designer consider all possible state transitions, allowing him to detect and eliminate many potential errors during the synthesis procedure.

Tables 5b and 6 describe the behavior of the two components of the system which contain critical sections. We are now ready to design the control mechanism. The control mechanism has two input variables x_1 and x_2 and four possible input states; therefore, there are four columns in the associated flow table. We require that if neither process is enabled to enter its critical section and if process requests arrive at the control simultaneously, access is given to the process that was not enabled last. Because of this requirement, the control mechanism must "remember" which process was enabled last. Two internal states are required for this purpose. Two more internal states are required, one to produce the enabling signal for process 1 and the other to produce the signal for process 2. Thus a minimum of four internal states are required for the control mechanism.

The usual techniques of flow table synthesis produce first a primitive flow table which has one stable entry in each row. Simplification procedures are then used to eliminate unnecessary internal states [23]. To simplify this discussion a flow table with four internal states will be obtained directly. To see how the design proceeds, suppose that neither process is requesting access to its critical section, $x_1x_2 = 00$, and process 2 was in its critical section last. Let the corresponding table entry l-00 be as shown in Table 8a. The output state is $z_1z_2 = 00$, and the control is stable waiting for an input transition. If the transition x_1x_2: 00 ➤ 01 occurs, the control must enter a new internal state where process 2 is enabled, as shown in Table 8b. From l-00, the transition x_1x_2: 00 ➤ 10 may also occur and process 1 must be enabled, say by internal state 3. If the x_1x_2: 00 ➤ 11 transition occurs while in l-00, that is, simultaneous requests, state 3 is entered and process 1 is enabled since process 2 was enabled last. Internal state 4 is used to remember that process 1 was enabled last. From 4-00, transitions analogous to those from l-00 can occur'. The table as specified thus far is shown in Table 8c. Consider the 2-01 entry. In this state, process 2 is requesting access to its critical section and is enabled, $x_2 = 1$ and $z_2 = 1$. There are three possible input transitions, x_1x_2: 01 ➤ 11 or 01 ➤ 00 or 01 ➤ 10. The first indicates that process 1 also desires to enter its critical section and the 2-11 entry is 2 indicating that the control must wait until process 2 leaves its critical section before enabling process 1. Notice that, as defined by the flow table in Table 5b, once X_1 becomes 1, it does not change to 0 until after z_1, is set to 1. The x_1x_2: 01 ➤ 00 transition indicates that process 2 has left its critical section.

Table 8. Design of the Control Flow Table'for the Two-Process Mutual Exclusion Problem

		x_1x_2				
	S	00	01	11	10	Z_1Z_2
(2 last)	1	①				00

(a)

	x_1x_2				
S	00	01	11	10	Z_1Z_2
1	①	2			00
2		②			01

(b)

		x_1x_2				
	S	00	01	11	10	Z_1Z_2
(2 last)	1	①2		3	3	00
(2 gets)	2		②	②		01
(1 gets)	3			③	③	10
(1 last)	4	④	2	2	3	00

(c)

	x_1x_2				
S	00	01	11	10	Z_1Z_2
1	①2		3	3	00
2	1	②	②	3	01
3			③	③	10
4	④	2	2	3	00

(d)

In this case, the control return to internal state 1, remembering that process 2 was in its critical section last. The final possibility ,x_1x_2: $01 \rightarrow 10$ accounts for the simultaneous occurrence of both of the first two transitions. In this case, the control enters internal state 3 and enables process 1. These transitions are included in Table 8d. From the 2-11 entry, the only possible transition is x_1x_2: $11 \rightarrow 10$, which takes the control to internal state 3. The transitions from the 3-10 and 3-11 entries are analogous to those from 2-01 and 2-11. The complete control flow table and process flow tables are shown in Table 9. These flow tables describe all possible interactions of the components in this system. The next step in the design process is to produce the actual programs and circuits that implement this system. Before this can be done, it is necessary to discuss in more detail the mode of operation used for system components.

BASIC COMFONENT STRUCTURE

We assume that there is no bound on the time for value changes to propagate in lines (Assumption 1); therefore, no global timing constraints can be made on a system and master clocks, which are commonly used to synchronize the operation of digital systems, cannot be relied upon. One approach to the elimination of master clocks utilizes propagation-limited logic [12]. Extra connections are provided between components. Each component has special inputs which

Table 9. Flow Tables for the Two-Process Mutual Exclusion Problem

z_1

	0	1	x_1
1	2	①	0
2	②	1	1

(a) Process 1

z_2

	0	1	x_2
1	2	①	0
2	②	1	1

(b) Process 2

x_1x_2

	00	01	11	10	z_1z_2
(2 last) 1	①	2	3	3	00
(2 gets) 2	1	②	②	3	01
(1 gets) 3	4	2	③	③	10
(1 last) 4	④	2	2	3	00

(c) Control

determine when the component output values may be changed. Special component outputs are also provided so that a component can notify other components when it is ready to accept new input information.

We propose a structure for components which does not require extra connections such as are used in propagation-limited logic. Extra control connections complicate formal analysis of a system and in the case of the mutual exclusion problem are unnecessary. However, extra circuitry is required to isolate a component from input changes which may occur while the component is responding to an earlier input change. Consider the operation of a fundamental-mode sequential circuit in an environment where line delays are unbounded. Fundamental-mode operation is defined as follows [23]:

Definition 3:

A sequential circuit is said to be operating in <u>fundamental-mode</u> if and only if all input changes occur when the circuit is stable.

A general form for a sequential circuit with level inputs and level outputs for fundamental-mode operation-is shown in Fig. 5. Set-Reset flip-flops* are used to store the internal state of the circuit and the output values depend only on the internal state. The internal

* A Set-Reset (S-R) flip-flop has two inputs, S and R, and two outputs, y and y'. When S=0 and R=1, y=0 and y'=1. When S=1 and R=0, y=1 and y'=0. When S=0 and R=0, the output of the flip-flop is determined by the most recent 1 value for S or R. If S and R have 1 values simultaneously, the values of y and y' are arbitrary.

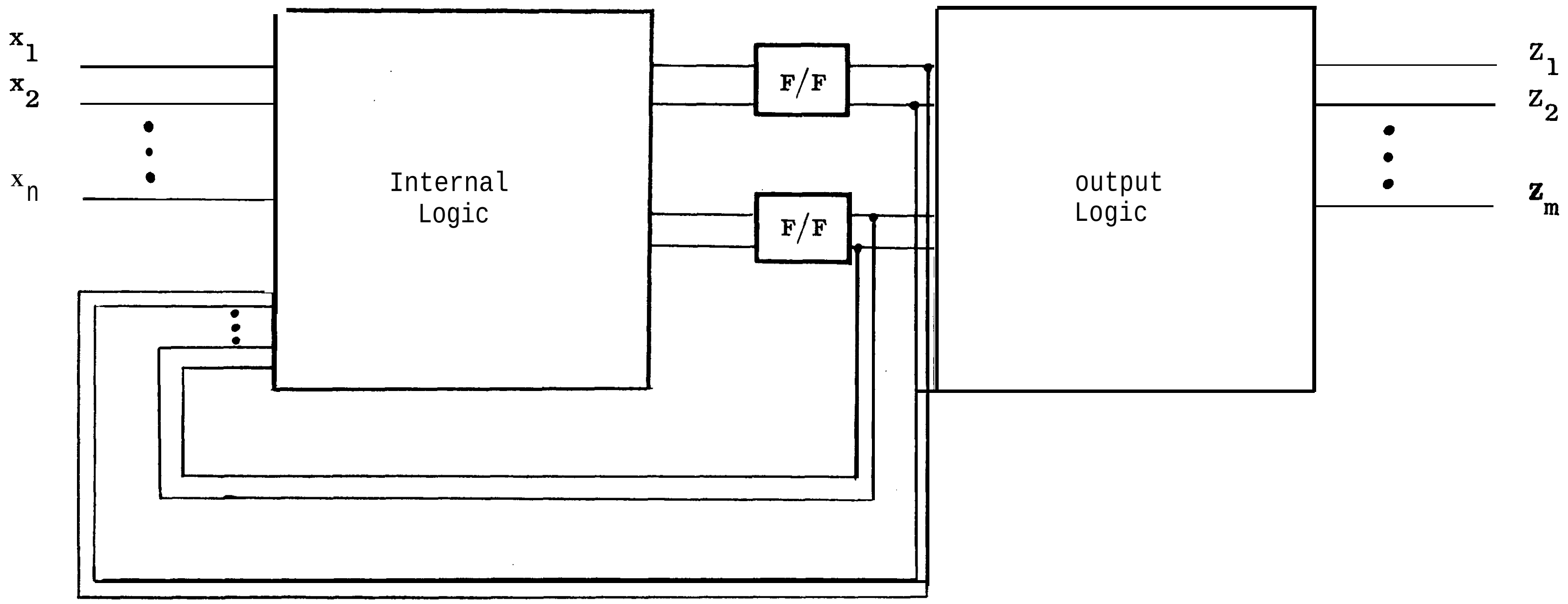


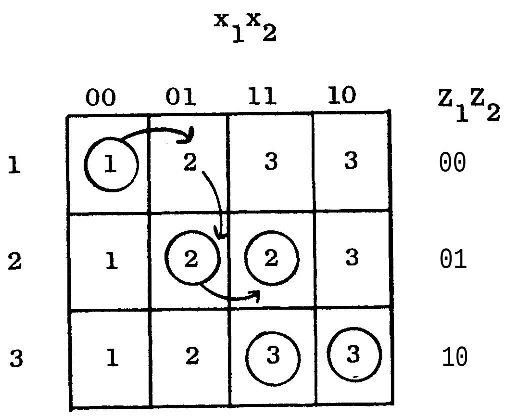
Figure 5. A general form for a sequential circuit using Set-Reset flip-flops.

logic and output logic boxes contain combinational circuits, circuits with no loops in the direction of signal propagation.

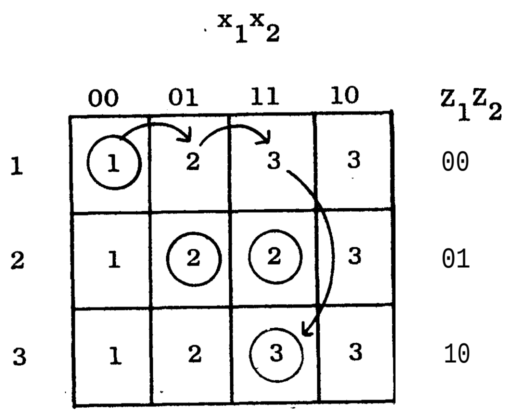
Fundamental-mode operation is usually attained by requiring that there be an interval between input value transitions of sufficient duration to allow the circuit to become stable. In our model, if a circuit or component has more than one input, it is possible for the circuitry to be responding to a change on one input line when a change on another input line occurs. Thus, it is not reasonable to assume fundamental-mode operation. To illustrate-the difficulties if a circuit in the form of Fig. 5 is not operated in fundamental mode, consider the flow table shown in Table 10. Suppose the present table entry is l-00 and the input value sequence x_1x_2: 00 ➤ 01 ➤ 11 occurs. One possible internal transition sequence is shown in Fig. 6a and another in Fig. 6b. In the first case, the duration of the 01 input state is sufficient to cause the 2-01 table entry to be entered. This is not the case in Fig. 6b. As a result, seemingly identical input sequences can produce different internal state transitions and as a consquence different output states. Such behavior is not desirable. Discussions of these difficulties are concerned with hazards in flow tables and circuits which undergo multiple-input transitions and are beyond the scope of this paper [9 , 10]. Instead, we define a mode of operation which guarantees that every input transition results in a unique internal-state transition and a unique output-state transition. Before presenting our mode of operation, another mode of operation commonly used with sequential circuits

Table 10. Flow Table Example

	x_1x_2				
	00	01	11	10	Z_1Z_2
1	①	2	3	3	00
2	1	②	②	3	01
3	1	2	③	③	10



(a)



(b)

Figure 6. Possible transitions for flow table of Table 9 given input sequence x_1x_2: 00 $\rightarrow$ 01 $\rightarrow$ 11 .

must be discussed. This mode of operation is called pulse mode and may be used with clocked sequential circuits [23].

Pulse Mode and Clocked Circuits

For our purposes, a clocked sequential circuit is a sequential circuit with level inputs and level outputs, such as shown in Fig. 5, but with one extra input called a clock input. If the clock input has the characteristic that it has a 1 value for a much shorter time that it has the 0 value, the clock-input is said to be a pulse type input or pulse input instead of a level input. For a pulse input, the 1 value for the clock input is called the clock pulse. Clocked sequential circuits can be designed to operate in fundamental mode as well as in pulse mode [23]. If a clocked sequential circuit is operated in pulse mode, the following assumptions are made about the width of the input pulse.

Assumption 3:

The pulse input is of sufficient duration to cause the appropriate flip-flops to change state.

Assumption 4:

The duration of the pulse-input is short enough that it is no longer present at the circuits which generate the flip-flop input signals when the change in flip-flop outputs has propagated to the input circuitry.

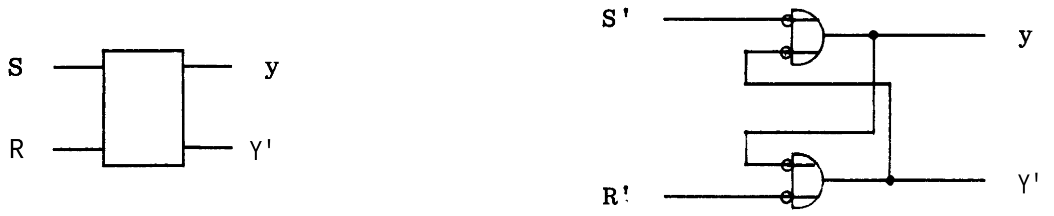
The actual definition of pulse mode operation for a sequential circuit is as follows.

Definition 4:

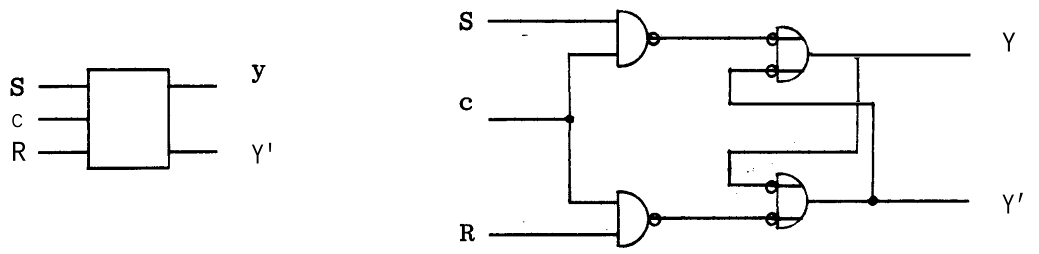
A sequential circuit is said to be operating in pulse mode if and only if the following conditions are satisfied:

1. At least one input is of pulse type.
2. Changes in internal state occur only in response to the occurrence of a pulse at one of the pulse inputs.
3. Each input pulse causes only one change in internal state.

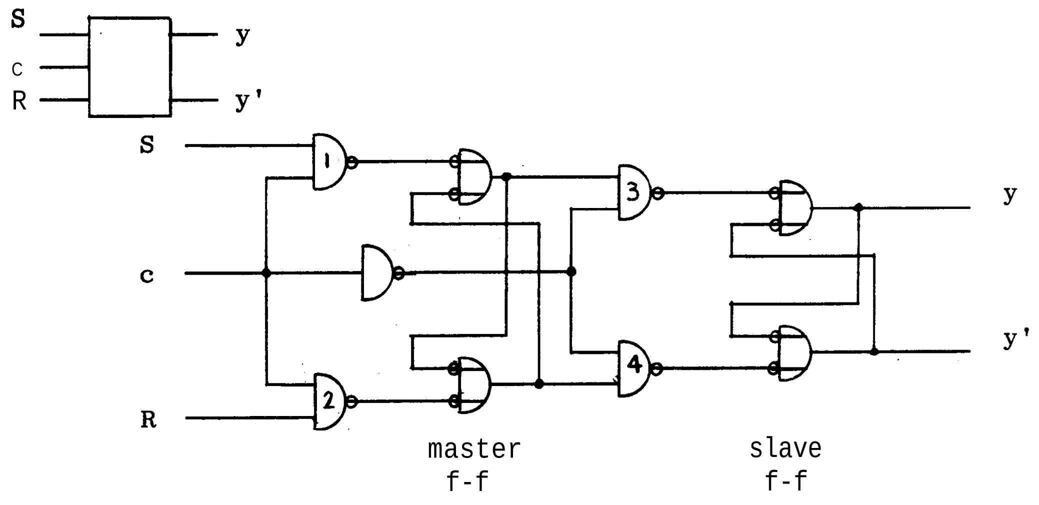
In clocked sequential circuits designed for pulse-mode operation, clocked Set-Reset flip-flops are often used. These flip-flops have an extra input for the clock signal and change state only when the clock pulse is present. In order to insure that Assumption 4 holds, double-rank or master-slave flip-flops can be used. These flip-flops change their outputs after the clock pulse has disappeared, preventing feedback signals from changing while the clock pulse is present. In Fig. 7, three basic flip-flop-designs are shown, the S-R flip-flop, the clocked S-R flip-flop, and the clocked, master-slave S-R flip-flop. The flip-flop of Fig. 7c operates in the following manner. When c is 0, changes in the S and R inputs are isolated from the master flip-flop and the outputs of the master flip-flop determine the outputs of the slave flip-flop. When c becomes 1, the clock input to gates 3 and 4 must become 0 isolating the slave flip-flop before any changes



(a) S-R flip-flop



(b) Clocked S-R flip-flop



(c) Clocked, master-slave S-R flip-flop

Figure 7. S-R **flip-flops.**

appear in the master flip-flop outputs. While c = 1, the master flip-flop records the new input conditions. When c becomes 0 again, the clock inputs to gates 1 and 2 must become 0 isolating the master flip-flop before any changes in the slave flip-flop outputs propagate to the S and R inputs of gates 1 and 2. As long as the gate delays in this flip-flop are greater than the line delay from the clock to the inputs of gates 1 and 2, this latter condition will be satisfied.

A general form for a clocked sequential circuit using clocked S-R flip-flops is shown in Fig. 8. In general, there may be multiple-input changes in the circuitry which determines the flip-flop inputs. Therefore, static and dynamic hazards may exist in the flip-flop inputs which cannot be eliminated by adding logic gates [9]. These hazards can result in spurious inputs to the flip-flops which could cause an internal-state flip-flop to be set or reset incorrectly if the inputs change while the clock pulse is present. One way to eliminate the effect of hazard pulses is to assume that the circuit inputs do not change while the clock pulse is present. However, if we made this assumption and went on to design a solution to the mutual exclusion problem, we would have solved one mutual exclusion problem by posing another one, just as Dijkstra did. This is so because we would have assumed the presence of the clock pulse and input changes are mutually exclusive. To eliminate the possibility of hazard pulses which can adversely affect component operation,

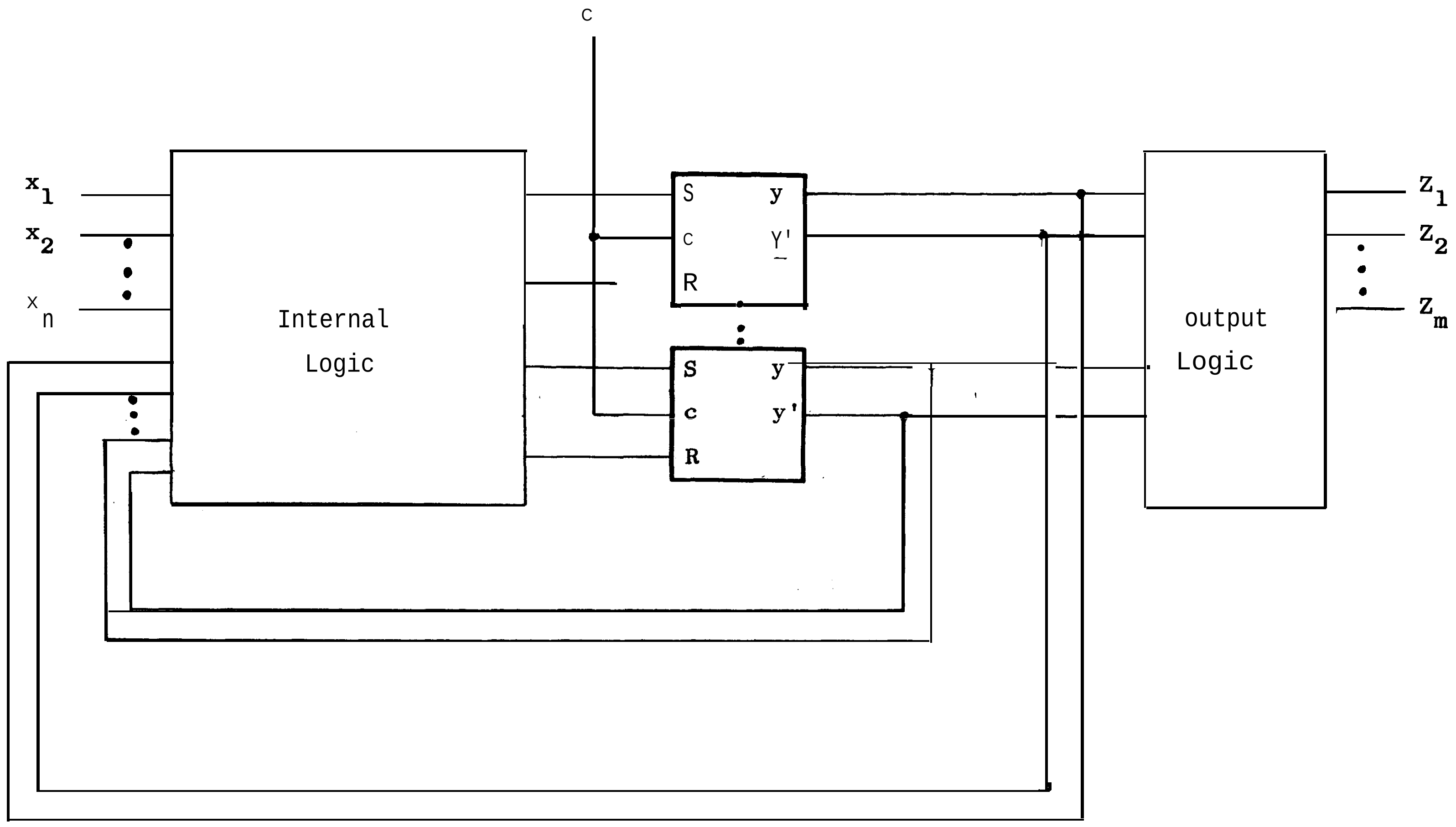


Figure 8. A general form for a clocked sequential circuit.

we define the following mode of operation for the components in parallel system.[*]

Component Operation

While most of this discussion is concerned with circuits, we intend that these techniques be used with components which are capable of executing programs as well. The basic form of a component circuit is shown in Fig. 9. This circuit has two ranks of clocked S-R flip-flops. These ranks are called the input rank and the output rank. The output rank corresponds to the flip-flops for clocked sequential circuits as shown in Fig. 8. Master-slave type flip-flops can be used in the output rank to isolate internal variable changes from the internal logic. The input rank of flip-flops serves the sole function of recording the values on the input lines. These flip-flops may be the simple clocked type shown in Fig. 7b. The component inputs $x_1,\ldots,x_n$ may change at any time. When the response to an input change is complete, a new input state is determined by applying a clock pulse on the c1 line. The cl pulse must be of sufficient duration to allow the input rank to become stable. When the cl pulse is removed, the effect of new input state must completely propagate to the inputs of the flip-flops in the output rank before the c_2 signal is set to 1. The 1 value for c_2 causes the effect of

[*] Friedman and Menon [9] have discussed the design of sequential circuits with multiple-input changes assuming the maximum time between successive input changes is bounded. This assumption cannot be made if line delays are unbounded.

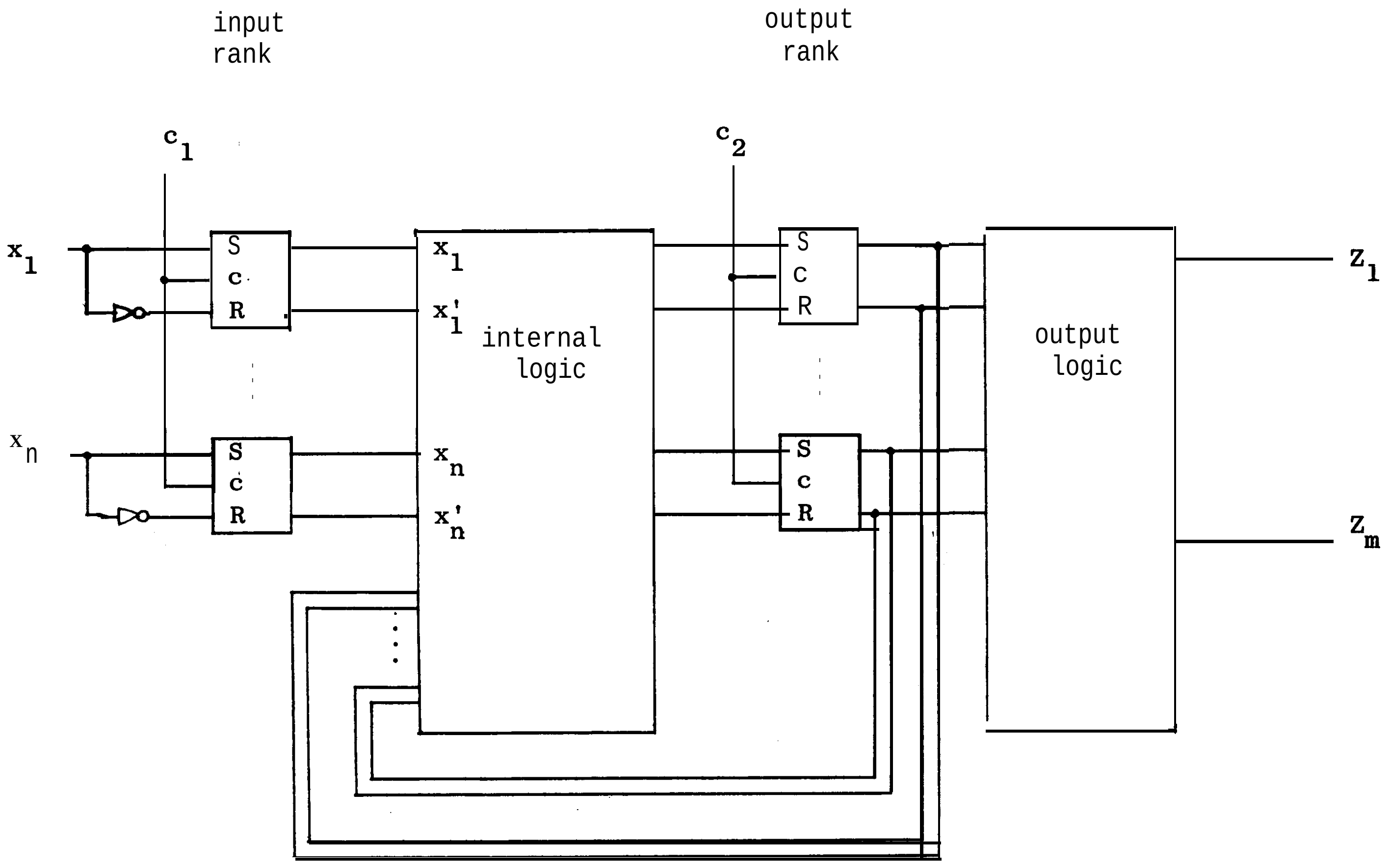


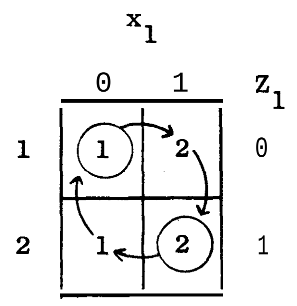
Figure 9. General form of a component circuit.

the input state to be recorded in the output rank. It is necessary to have bounds on the delays in the component circuitry to realize this mode of operation (Assumption 2).

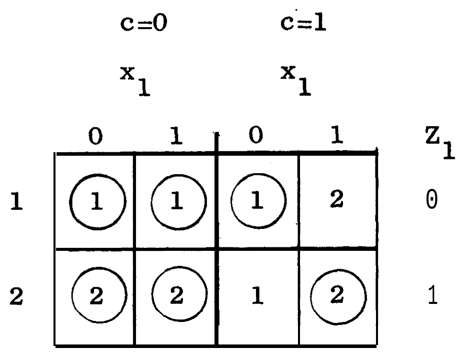
With this mode of operation, the only possible transient effect caused by the occurence of multiple-input changes is the following. When cl has the value 1, there may be 1 values on both the Set and Reset inputs of one or more input rank flip-flops. When cl becomes 0, these flip-flops will record the input value either just prior to the transient condition or just after. In either case, the next phase of component operation proceeds properly.

The design of component circuitry, exclusive of the input rank, is the same as the design of a clocked sequential circuit under the assumption, which is valid because of the presence of the input rank, that the circuit inputs $x_1,\ldots,x_n$ do not change when the clock (c_2) input is present. The clocked sequential circuit may be designed for pulse-mode or fundamental-mode operation.

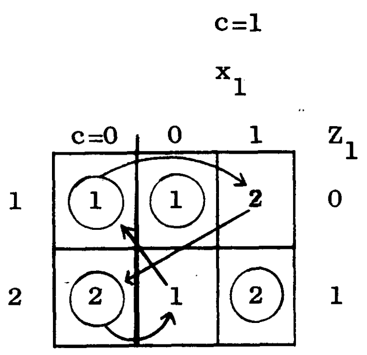
The flow tables used to describe the operation of a clocked sequential circuit have extra columns to account for the presence and absence of the clock (c_2) input. Fig. 10a shows a flow table designed without consideration of a clock input. The internal-state transitions are indicated by arrows. This same flow table with a clock input added is shown in Fig. 10b. When the clock signal is absent, the table is always stable in a particular internal state. For this reason, these flow tables are usually drawn in the form shown in Fig. 10c with only a single column to describe the operation when the



(a) Without Clock Input



(b) With Clock Input



(c) Single Column for Clock Input
(Pulse Mode)

Figure 10. Flow tables with and without clock input.

clock input is 0. For pulse-mode operation, the internal state transitions are diagonal transitions as shown in Fig. 10c. This is because of Assumption 4.

Since the clock connections are specified and the non-clock inputs and outputs are level signals, the design of the internal logic circuits in Fig. 9 can proceed, ignoring the clock input, using flow tables in the form shown in Table 2. It is understood that these flow tables specify the operation of the component when c_2 has the value 1 and that when c_2 is 0, the component is in the stable state specified during the previous interval when c_2 had the value 1.

If pulse-mode operation is used, critical races and essential hazards cannot affect the operation of the clocked sequential circuit. Critical races may occur in a fundamental-mode circuit if two or more internal variables are unstable simultaneously. If the final internal state depends on the order in which the internal variables change value, a critical race exists [23]. In pulse mode, Assumptions 3 and 4 guarantee that the next internal state is unique. Essential hazards may occur in a fundamental-mode circuit if an internal variable changes before the propagation of an input change is complete. As with a critical race, the effect of the essential hazard is such that the next internal state is not uniquely specified. For pulse mode, Assumption 4 eliminates the possibility of adverse effects from any essential hazards.

Using the basic component structure just described, if each component input changes value at intervals greater than the component cycle time (successive O-l-0 transitions for c_1 and c_2), every input transition will result in a unique internal state transition. It is possible for an input variable to change value more than once during the basic component cycle time. If this happens, input transitions will not be recognized by the component. General consideration of such input changes will be given in a later paper. In the case of the solution to the two-process mutual exclusion problem, all input transitions must be recognized. This is so because when each component changes one of its output values, it does not change the output value again until it recognizes an input transition which is produced in recognition of its own output value change. For example, when access to critical section 1 is requested, Xl is set to 1. This value is not changed until the xl: $0 \to 1$ input transition is recognized by the control and the control sets Z_1 to 1 allowing the critical section to be entered.

The mode of operation we have described can be used by a processor in a multi-processor computer system. When the processor is executing instructions which have no effect on the rest of the system, the internal clock signals c_1 and c_2 both have the value 0. Output values, which are inputs to other components, may be changed at any time. When a processor reaches a point where further action depends on the values of inputs from other parts of the system, the present input state is recorded by applying the c1 pulse. After the input state is determined, the 1 value for c_2 initiates the appropriate processor action.

The use of an input rank of flip-flops to record the input state should be used by all components for which it is possible to have moe than one input change at any instant. If a component has only a single input, a fundamental-mode implementation can be obtained without using clock signals. Belay elements may be necessary in feedback paths to eliminate essential hazards [23 , 28].

In a later paper, we show how a <u>sequential program</u> can be produced to implement a given flow table specification. In the next section, a control circuit for the two-process mutual exclusion problem is designed.

A CONTROL CIRCUIT FOR THE TWO-PROCESS MUTUAL EXCLUSION PROBLEM

The flow table specification of the control mechanism was given in Table 9. Let us assume the c_2 input is a pulse-type input and the circuit is operated in pulse mode. The steps in this design process are discussed in detail by McCluskey [23]. The design of the circuit will be sketched only briefly.

Since the control flow table has-four internal states, two internal variables, y_1 and y_2, are required. An internal state assignment, an assignment of internal variable values for each of the internal states in the flow table, is given in Table 11. Such a table is called a <u>transition table</u>. This table has exactly the same form as a flow table except that each table entry specifies the next

Table 11. Transition Table for the Control Circuit

		x_1x_2				
	y_1y_2	00	01	11	10	Z_1Z_2
(2 last)	00	$\textcircled{00}$	01	10	10	00
(2 gets)	01	00	$\textcircled{01}$	$\textcircled{01}$	10	01
(1 last)	11	$\textcircled{11}$	01	01	10	00
(1 gets)	10	11	01	$\textcircled{10}$	$\textcircled{10}$	10

Y_1Y_2

values or excitation values of the internal variables, Since the internal states are realized with Set-Reset flip-flops, the next step is to produce the excitation table for the Set and Reset input lines to the flip-flops corresponding to the internal variables yl and y_2, This table is shown in Table 12. The entries in this table can be determined from the corresponding entries in the transition table using standard techniques [23]. From the excitation table, the excitation functions for the two flip-flops are obtained using techniques for combinational network synthesis [23]. These excitation functions specify the combinational network used in the internal logic portion of the circuit. The excitation functions are given below.

$$S_1 = x_1(x_2' + y_2')$$
$$Rl = x_2(x_1' + y_2)$$
$$S_2 = x_1'(x_2 + y_1)$$
$$R_2 = x_2'(x_1 + y_1')$$

The output excitation functions determine the combinational network for the output logic circuit. These functions depend only on the internal variables. y_1 and y_2 and can be determined from Table 11. They are given below.

$$Z_1 = y_1y_2'$$
$$Z_2 = y_1'y_2$$

The complete control circuit diagram including the input rank and clock inputs is shown in Fig. 11. This completes the synthesis of the control circuit.

Table 12. Excitation Table for the Control Circuit

x_1x_2

y_1y_2	00	01	11	10
00	0d,0d	0d,10	10,0d	10,0d
01	0d,01	0d,d0	0d,d0	10,01
11	d0,d0	01,d0	01,d0	01,d0
10	d0,10	01,10	d0,0d	d0,0d

S_1R_1, S_2R_2

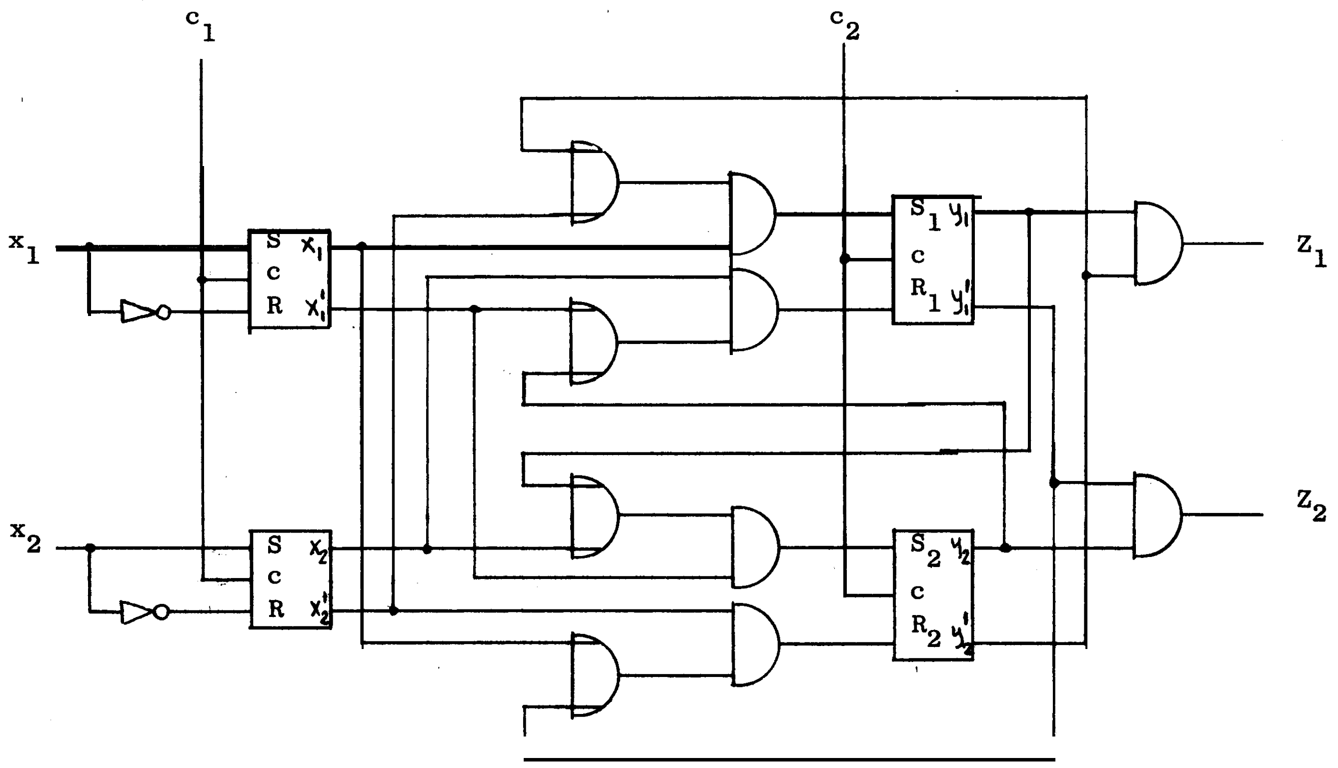


Figure 11. Control circuit for the two-process mutual exclusion problem.

It is interesting to observe that the clocked sequential circuit portion of the control circuit in Fig. 11 also realizes the control flow table of Table 9 when c_2 is a level signal and the circuit is operated in <u>fundamental mode</u> <u>if</u> the time for all flip-flops in the output rank to become stable is less than the minimum delay in propagation of flip-flop outputs to flip-flop inputs. The latter condition is sufficient to eliminate the effects of essential hazards. The control circuit contains no critical races because, in the 00 and 11 columns of Table 11, only one internal variable changes at a time. The reason both pulse mode and fundamental mode result in proper operation of the circuit is because of the form of the control flow table, in which every unstable entry specifies a stable entry [23a]. The next internal state is unique even if c_2 retains the value 1 until the circuit becomes stable.

As we have defined a parallel system, every flow table must have the form specified by Definition 2. Therefore, any circuit obtained in the manner described in this section can be operated in pulse mode or fundamental mode with respect to the c_2 clock input. The question which remains to be answered is the following-. Must it always be possible to describe component operation by a flow table? We answer this question affirmatively with the following argument.

Suppose the component (program or circuit) that produces an input value transition for a variable intends that the variable return to its original value before the component which recognizes the input transition enters its next internal state. This type of operation should not be used because the line delays between components cannot be controlled (Assumption 1). Next suppose the input value remains constant. Either the component which recognizes the input value eventually becomes stable or it does not. A state table which never becomes stable for some fixed input state is said to contain a _cycle_. Consider such a table. During a cycle either some output changes value or no output changes value. If an output value changes and it changes more than once, the intermediate values may or may not be recognized because of the line delay assumption. Therefore cycles with multiple transitions for a single output should not be used. On the other hand, a cycle with at most one change for each output variable can always be replaced by a transition to single stable state without affecting the external behavior of the component. The output state for the final stable state is specified by the final value for each output variable. We will assume that this is always done and that any state table used to describe the operation of a component is _cycle-free_. If a state table is cycle-free and the input value remains constant, the table (component) must eventually enter a stable entry. The stable entry may be entered after a sequence of unstable transitions. During this sequence, consider what can happen to the value of each output variable. Either the value does not change or it changes exactly once or it changes more than once. Multiple value changes must not be used for the same reason that ruled out multiple transitions during a cycle, the intermediate output values may not be recognized. In the cases where the output value does not change or changes exactly once, the entire sequence of unstable transitions can be replaced by a single transition directly to the final stable entry without affecting the external behavior of the component (the line delay assumption (Assumption 1) makes it impossible to control the order in which output values actually propagate to component inputs). We conclude that the tables used to describe meaningful component operation can always be put in a form which is cycle-free and which is such that every unstable entry specifies a stable entry. That is, they can be represented as flow tables.

CONCLUSIONS

The flow table model has been shown to be valuable in the design of a control algorithm for the two-process mutual exclusion problem. From a flow table, sequential circuit implementations can be designed. In a later paper, it will be shown that flow tables can be used for the analysis and synthesis of sequential programs. As a result, our model provides a common basis for the treatment of program and circuit implementations of control algorithms.

REFERENCES

[1] Adams, D. A. A computation model with data flow sequencing. CS-117 (Thesis), Computer Science Department, Stanford University, Stanford, California (Dec. 1968).

[2] Ashcroft, E. and Manna, Z. Formalization of properties of parallel programs. Memo No. AIM-110, Stanford Artificial Intelligence Project, Stanford University, Stanford, California (Feb 1970).

[3] Clark, W. A. Macromodular computer systems. Proc. SJCC (1967), 335-336.

[4] Conway, M. E. A multi-processor system design. Proc. FJCC (1963).

[5] Dennis, J. B. and Van Horn, E. C. Programming semantics for multi-programmed computations. Comm. ACM, 9 (March 1966), 143-155.

[6] Dijkstra, E. W. Solution of a problem in concurrent programming control. Comm. ACM, 8 (Sept 1965), 569.

[7] Dijkstra, E. W. Co-operating sequential processes. in Programming Languages, Genuys, F. (Ed.), Academic Press, New York (1968).

[8] Dijkstra, E. W. The structure of the "THE" multiprogramming system. Comm. ACM, 11 (May 1968), 341-346.

[9] Eichelberger, E. B. Hazard detection in combinational and sequential switching circuits. IBM Research Journal (March 1965), 90-99.

[10] Eichelberger, E. B. Sequential circuit synthesis using hazards and delays. Ph.D. Thesis. Department of Electrical Engineering, Princeton University, Princeton, New Jersey (1963).

[11] Friedman, A. D. and Menon, P. R. Synthesis of asynchronous sequential circuits with multiple-input changes. IEEE Trans. on Computers, C-17 (June 1968), 559-566.

[12] Goldberg, J. and Stone, H. S. Asynchronous propogation-limited logic. IEEE Conference Record of the 7th Annual Symposium on Switching and Automata Theory (1966), 215-226.

[13] Huffman, D. A. The synthesis of sequential switching circuits. in Sequential Machines: Selected Papers, E. F. Moore (ed.), Addison-Wesley Publishing Co., Inc. (1964), 3-62.

[14] IBM System 360 Principles of Operation. File No. S360-01, Form A22-6821-4, IBM corporation (1966).

[15] Karp, R. M. and Miller, R. E. Properties of a model for parallel computations: determinacy, termination, queueing. SIAM J. Appl. Math., 14 (Nov 1966), 1390-1411.

[16] Karp, R. M. and Miller, R. E. Parallel program schemata: a mathematical model for parallel computation. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 55-61.

[17] Karp, R. M. and Miller, R. E. Parallel program schemata. J. of Computer and System Sciences 3, 2 (May 1969), 147-195.

[18] Knuth, D. E. Additional comments on a problem in concurrent programming control. Comm. ACM, 9 (May 1966), 321-322.

[19] Lampson, B. W. A scheduling philosophy for multiprocessing systems. Comm. ACM, 11 (May 1968), 347-360.

[20] Luconi, F. L. Completely functional asynchronous computational structures. IEEE Conference Record of the 8th Annual Symposium on Switching and Automata Theory (Oct 1967), 62-70.

[21] Luconi, F. L. Asynchronous computational structures. MAC-TR-49 (Thesis), Massachusetts Institute of Technology, Cambridge, Massachusetts (Feb 1968).

[22] Luconi, F. L. Output functional computational structures. IEEE Conference Record of the 9th Annual Symposium on Switching and Automata Theory (Oct 1968), 76-84.

[23] McCluskey, E. J. Introduction to the Theory of Switching Circuits. McGraw-Hill Book Co., New York, N. Y. (1965).

[23a] McCluskey, E.J. Fundamental mode and pulse mode sequential circuits. Proc. IFIP Congress, Munich, Germany (1962), 725-730.

[24] Muller, D.E. and Bartky, W.S. A theory of asynchronous circuits. Proc. of an International Symposium on the Theory of Switching, The Annals of the Computation Laboratory of Harvara university, Vol. 29, Part 1, Harvard University Press (1959), 204-243.

[25] Rodriguez, J. E. A graph model for parallel computations. Ph.D. Thesis, MIT, Department of Electrical Engineering, Cambridge, Massachusetts (Sept 1967).

[26] Saltzer, J. H. Traffic control in a multiplexed computer system. MAC-TR-30 (Thesis), MIT, Cambridge, Massachusetts (July 1966).

[27] Slutz, D. R. The flow graph schemata model of parallel computation. MAC-TR-51 (Thesis), MIT, Cambridge, Massachusetts (September 1968).

[28] Unger, S. H. Hazards and delays in asynchronous sequential switching circuits. IRE Transactions on Circuit Theory, CT-6 (March 1959), 12-25.