Project 7000

Maintenance Development Memo #11

SUBJECT: Circuit Techniques for Checking Control Circuitry

BY: Frank B. Hartman

DATE: October 23, 1957

CONCLUSIONS:

1. Good design technique consists of adding a minimum amount of controlled redundancy to achieve continuity of checking, consistent with the generation of error signals that allow adequate automatic fault location.

2. The most useful schemes appear to be those which utilize a parity bit generator and subsequent code check. Complete duplication of parts is sometimes necessary in the interests of continuity of checking. Circuit "timing" checks are necessary wherever ring-like structures are utilized, to check that the ring flip-flops have not failed and that the timed events do not take longer than expected.

3. The adequacy, convenience, and efficiency of the few schemes presented here cannot be further evaluated until details are obtained regarding the structure of the specific circuits and systems to be checked.

INTRODUCTION

The first report of the Checking Task Group established the desirability of the following design techniques:

1) a vigorous effort to reduce the likelihood of undetected error, leading to a high degree of continuity of checking;

2) implementation of automatic error diagnosis, leading to the rapid location of faults to a basic module; and

3) use of highly efficient checking schemes to keep added checking hardware to a minimum.

These techniques tend to reduce the mean repair time (T_r) of the system by rendering maintenance easier. The mean free error time (T_{mfe}) of the system is reduced by the added checking circuitry, but efficient checking schemes should keep this decrease to a minimum. If mean repair time is reduced by a greater amount than mean free error time, then the mean operate time (T_o), expressed as

$$T_o = T_{mfe} - T_r ,$$

will increase, indicating an increase in reliability of the system.

In this report we shall investigate how these techniques may be applied in the design of checking circuitry for control circuits. The material is arranged in the following order:

1) We first discuss the nature and structure of control systems, with particular emphasis upon the important types of control circuitry.

2) Asynchronous controls are discussed, and checking circuits proposed for a fairly general control system. The checking circuits are then evaluated in terms of how well they meet the design objectives.

3) Synchronous controls are discussed in a similar manner.

This report is one of a series that have been planned on the topics of checking techniques for data-flow, arithmetic operations, and controls. It is assumed that all circuits of a computing system fall into one of these three categories. Since the term "controls" seems to be less specific than either of the other two, it seems natural to consider as control any circuit that cannot be clearly classified as data-flow or arithmetic. If the reader will reflect that such things as Code Checkers, Error Correctors, Recognition Circuits, in addition to Operation Decoders, Command Generators, etc., also perform "control" functions, he will realize that control systems are inherently the most complex.

For obtaining proper perspective, it should be realized that circuit techniques are not the only ones available for checking controls. Powerful means of program checking could conceivably be developed which would locate faults once other circuits have performed the essential job of detection. Failures of control circuits can often safely be detected indirectly in terms of data-flow or arithmetic error, so that circuits for direct checking may not be needed in some cases.

Failure processing occurs in the natural order of (1) detection, (2) location, and (3) correction. The first two subprocesses have as object the facilitation of the third. Each one takes time and circuitry. By adding circuitry, the amount of time required to complete the first two stages may be reduced, on the average. Up to a point the cost of the added hardware may be less than the money saved due to the time saved. Past such a crossover point, the cost of added checking circuits is unjustified.

THE NATURE OF CONTROLS

Certain two-valued lines within a computing system are clearly recognized as control lines. Such lines are often labeled according to the corresponding control function, as, e.g., "Read Memory", "Char. Reg. 1 to Adder", "Step Mem. Addr. Counter Plus One", "Route Decimal Carry In", "Suppress Code Check", etc. Such lines are outputs from the so-called "command generator" and initiate the actions indicated by their labels, assisted perhaps by standard timing pulses.

Certain other two-valued lines are classified as recognition lines. They are often given names according to the condition recognized, as, e.g., "Char. Reg. 2 Equals Storage Mark", "Digit Adder Output Equals Zero", "Decimal Carry Out", "Char. Reg. 1 Code Chk Odd", "End of Memory", etc. The "on" condition of such a line can be said to recognize the condition corresponding to its name.

That most "control" lines exhibit the qualities of "recognition" lines and vice versa is often obscured by the labeling of the line as one or the other. To illustrate, consider the command generator, Figure 1. According to switching algebra, each output must be considered as a function of all the inputs. This statement could be translated two

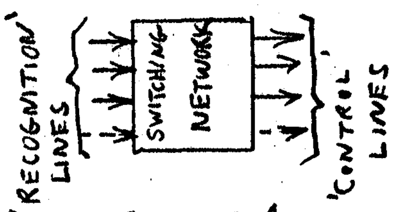


FIGURE 1
COMMAND GENERATOR

ways:

(1) the inputs "control" the outputs, and
(2) the outputs "recognize" certain input conditions.

To be most realistic, no two-valued line can be regarded as having the function of pure control nor pure recognition, but must be regarded as having aspects of both (control-recognition). The sharp distinction between the control and recognition functions has arisen because of the practice of considering certain logical blocks such as adders, registers, counters, core arrays, etc., as those parts which are naturally controlled; and in considering certain other logical blocks, such as command generators, address registers, priority-determining lockout circuits, sequencing flip-flops, etc., as those parts which naturally do the controlling of the former class. That the former class of circuits not only is controlled by the latter but also controls the latter is often ignored.

The class of circuits which form the subject of this investigation then consists of circuits usually called controls, and, in addition, certain circuits usually referred to as recognition circuits. Circuits that recognize that a counter has reached a pre-set level, say zero; or that a certain word or byte of special significance has been set up in a register; the end of an operation, etc., form part of "controls" in the broader sense.

The reader should be aware that there are different levels of control. For example, suppose the line which gates the stepping of a counter is an output of a command generator which is in turn controlled by the output of some word register. Then we may say that the word register controls the command generator which controls the stepping of the counter. The word register exercises control of the control of the stepping, or second level control of the stepping. Such distinctions are often necessary, particularly in regard to those control signals which arise because of the error recognition circuits. A computer under failure conditions is different from that computer under normal conditions, the difference arising from the change in action due to error detection.

CONTROLLED EVENTS

A useful point of view towards controls may be developed by considering the basic controlled event, as shown in Figure 2. The box contains

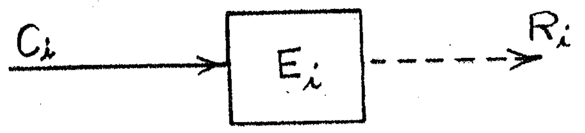


FIGURE 2

BASIC CONTROLLED EVENT

circuitry within which control line C_i may initiate event E_i. For example, C_i might be the line "Read Memory". The box may contain a core array together with its drivers. The corresponding controlled event is that of reading memory or cycling the memory cores. The line R_i may or may not be present, but if it is, it <u>recognizes</u> that the controlled event E_i has (correctly) taken place.

There are two basic techniques for checking that controlled events have occurred. These correspond to the two basic types of control, synchronous and asynchronous. Often in synchronous machines, a fixed time duration is allowed for the event to occur. If the recognition line is down at the end of such a period, its inverse allows an error flip-flop to be turned on, machine to be stopped, etc. In asynchronous machines, on the other hand, no fixed time duration is allotted. As soon as the recognition line comes up the machine is permitted to proceed to the next operation. The machine cannot proceed until the recognition line is up.

It is characteristic of the asynchronous type of control that for each controlled event E_i there must exist a corresponding recognition line R_i in order to signal <u>when</u> the event has been completed. Thus the basic purpose of recognition is to provide timing signals. It is an incidental (but valuable) function of recognition lines to act as checks upon correct completion of the event.

An asynchronous control system can be considered as

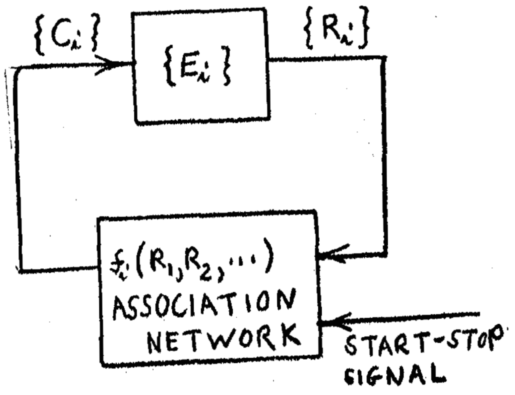
(1) an ensemble of control lines C_i controlling the ensemble of events E_i which generate in turn the set of recognition lines R_i , plus

(2) an association network which relates the control lines functionally to the recognition lines.

This arrangement is shown in Figure 3. We would expect that this

FIGURE 3

ASYNCHRONOUS CONTROL

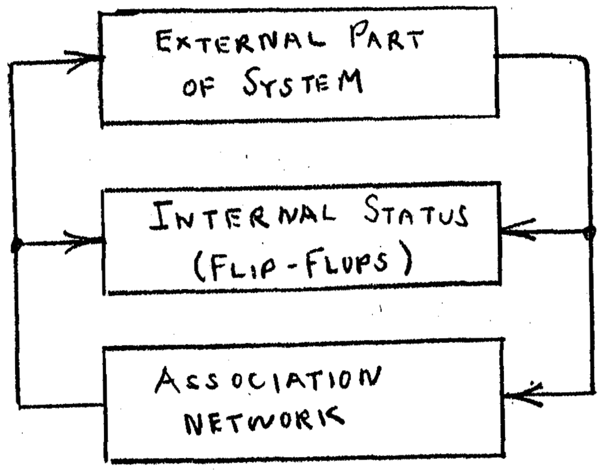


configuration, once started, should oscillate at its own natural rate. In general, these oscillations are started or stopped by external control. Once started the oscillations are continued, because the occurrence of one set of events is recognized and used to initiate the next set, etc. Thus all events proceed at their natural rates, with no fixed time durations.

It is useful to classify controlled events into two categories: (1) those that occur to circuitry outside the control (external events); and (2) those that occur to circuitry within the control (internal events). The control system is assumed to consist only of electronic switching circuits, i.e., combinational switching circuits plus flip-flops. We consider all of the combinational switching circuits as part of the association network, and thus the only controlled events within the control occur to flip-flops. The basic structure of asynchronous controls is shown in Figure 4.

FIGURE 4

STRUCTURE OF ASYNCHRONOUS CONTROL



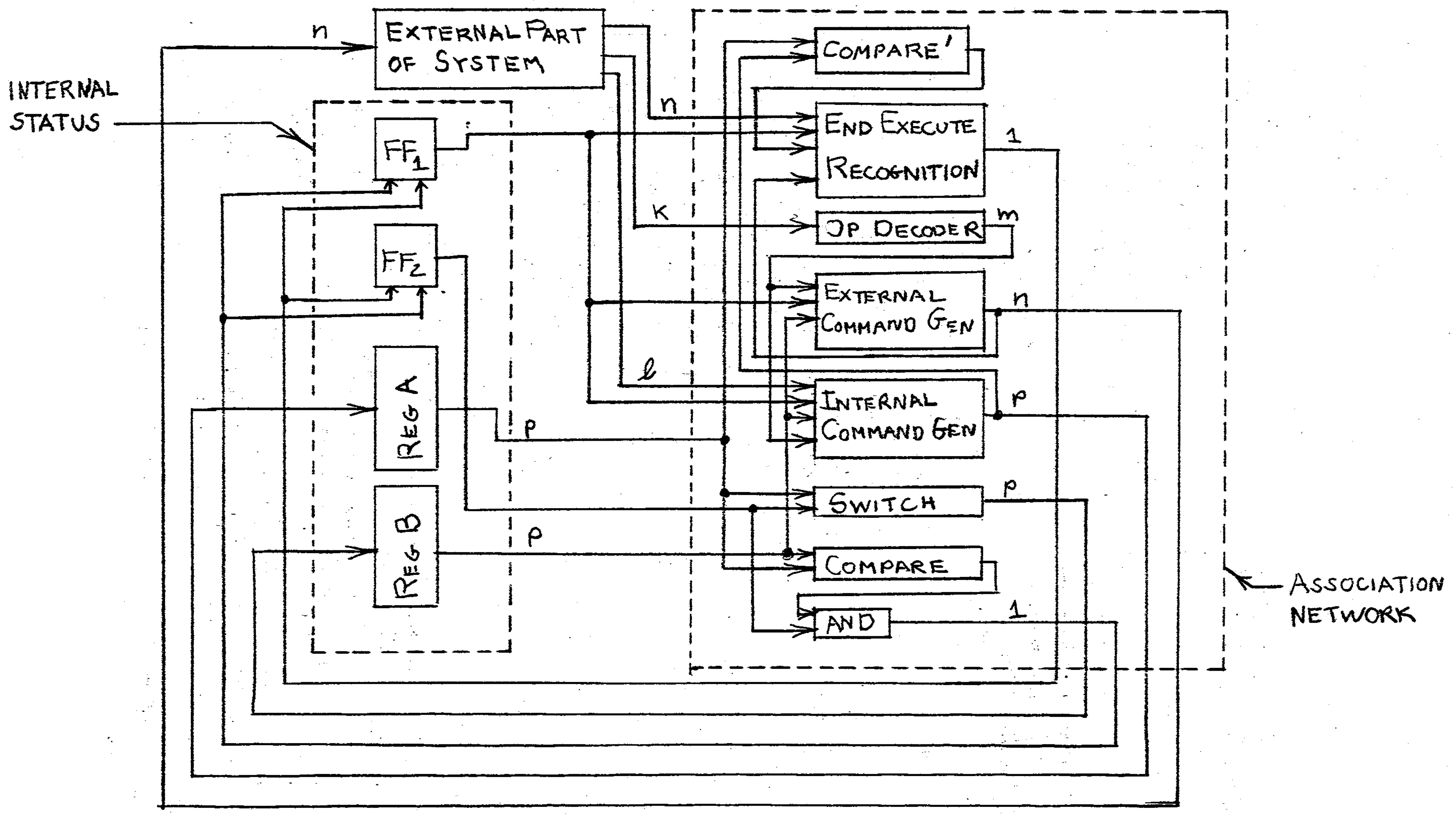


FIGURE 5 ASYNCHRONOUS CONTROL EXAMPLE

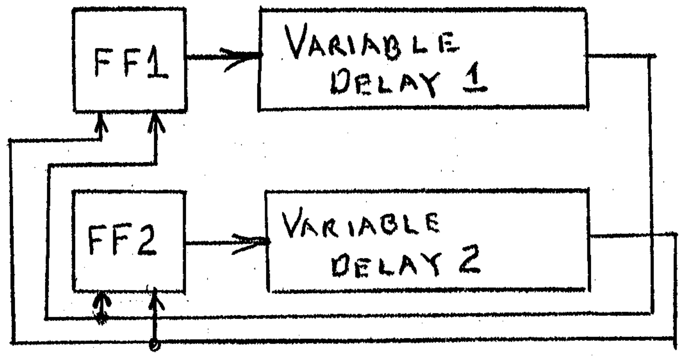
FBH. 11/1/57

ASYNCHRONOUS CONTROL

A detailed example of an asynchronous control is shown in Figure 5. The main function of this control is to generate control signals and send them to its environment in order to institute desired events in the desired sequence. The sequence of events depends upon the internal status and the conditions on the recognition lines being received by the control from its environment. Figure 5 shows n control lines going out of the control, and (n + k + 1) lines coming in. The first n lines coming in are in one-to-one correspondence with the control lines going out and indicate when the corresponding controlled event has been completed. The next k lines, excluding the parity bit, are received from an external operation register. This operation register is assumed to be at the end of a data-flow path and is not directly a part of the control. The last 1 lines are special recognition lines which are used to change the internal status of the system. Special recognition lines do not directly affect the environment, but must first affect the status of the control, which may then affect the environment. All higher level control lines from the environment must be special recognition lines, or operation register bits. A control can then be characterized by (1) number of essential bits of internal memory, (2) number of control lines generated, (3) number of operation register bits used, and (4) number of special recognition lines utilized.

The flip-flops 1 and 2 exhibit an oscillatory pattern of operation. These form a ring with structure shown in Figure 6.

FIGURE 6
PRIMITIVE FORM
OF
ASYNCHRONOUS
OSCILLATOR



Under certain conditions of the special recognition lines the following sequence of actions is allowed to proceed (See Figure 5):

(1) First, with FF_1 on, FF_2 off, the current operation is to be executed under control of the status register B and the External Command Generator. At the same time, the next status within the current operation is to be generated by means of Register B in conjunction with the internal command generator. This next status is entered into Register A. After all these actions are recognized to have been completed (in the End Execution Recognition box), FF_1 is turned off while FF_2 is turned on. The latter action initiates step 2.

(2) Second, with FF_1 off and FF_2 on, the contents of Register A are transferred to Register B. After this action is completed (recognized by Compare circuit), FF_1 is turned on while FF_2 is turned off. This initiates step 1.

The cyclic pattern described above can be broken only by a higher level of control that stops the oscillation, or else by failure of some component.

A set of checking circuits for the control of Figure 5 is shown in Figure 7. The main techniques of this scheme are:

(1) Status Register A, the "next status" register, is provided with a parity bit so that single-error detection is provided. The event "Set Register A to Next Status" is checked by comparing the contents of A with what the Internal Command generator has set up as the next operation. This check is performed by the circuit labeled "COMPARE". If there is no comparison, the FF_2 is not permitted to turn on and the system halts. When this event is not taking place, the parity checker for register A checks that the proper bit count exists in A. If there is an improper code in Register A, an error is signaled and system is prevented from proceeding (by a higher level control).

(2) Status Register B, the "current status" register, is checked in a manner similar to the method used for Register A. The event "Set Register B to Register A" is checked by comparing the output of registers A and B. If they do not agree, then FF_1 is prevented from turning on and the system is halted. A parity checker checks that Register B does not change during its "standby" period.

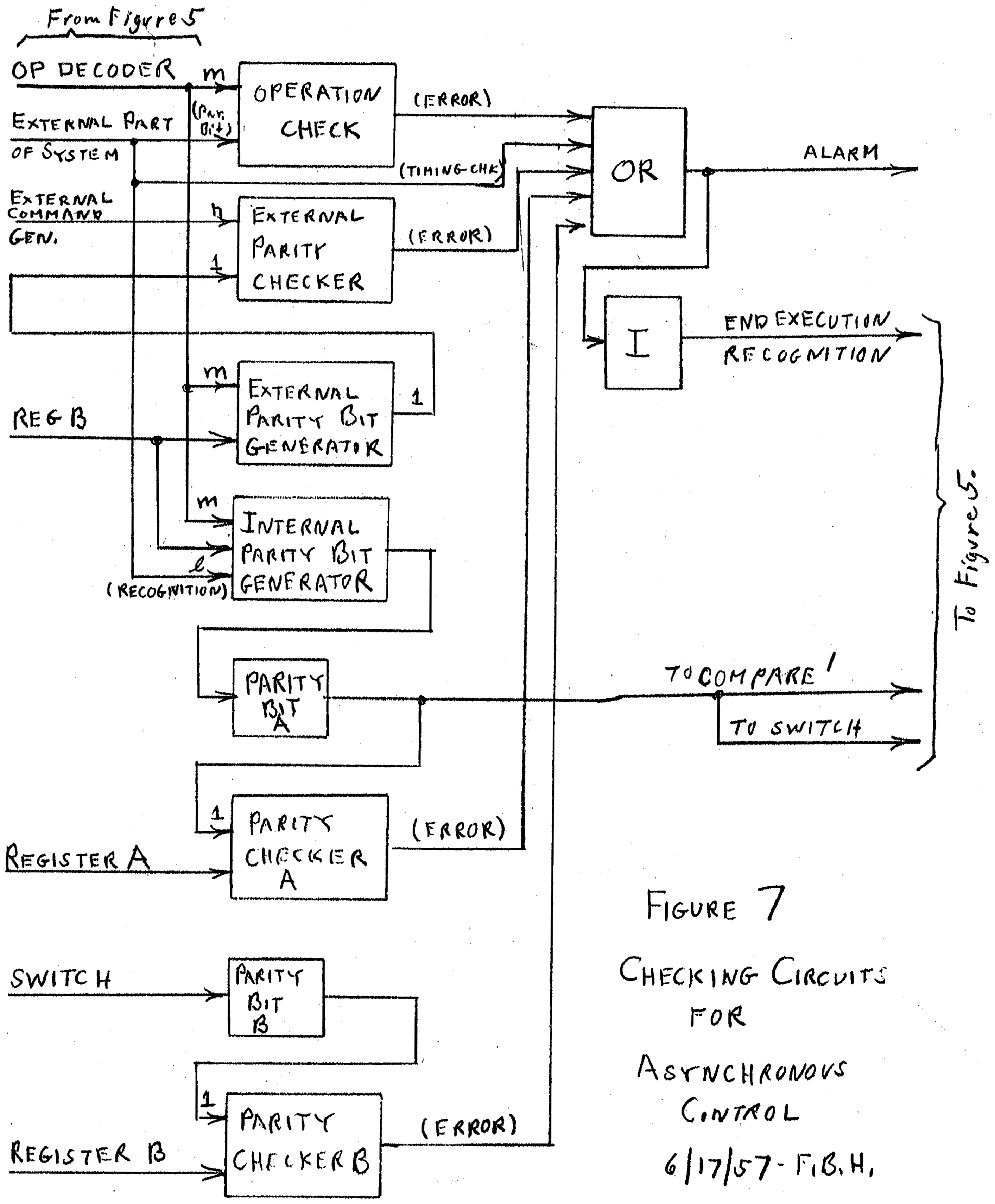


FIGURE 7

CHECKING CIRCUITS FOR ASYNCHRONOUS CONTROL

6/17/57 - F.B.H.

(3) The internal command generator is provided with a parity bit generator, but no parity checker is needed at this point. The action of the internal command generator, and of the internal parity bit generator, is checked at the same time the setting of register A is being checked. The check is performed by the same checking circuitry. The internal parity bit generator is apt to become quite complicated for reasonably complex systems, because any "recognition" lines generated within the internal command generator must be essentially duplicated in the internal parity bit generator in order to guarantee the proper parity bit for all recognized conditions. For example, part of the internal command generator may follow the rule:

> "If Register C (located in external part of system) contains all bits down, set all bits of register A down."

Then, within the internal command generator, there must exist an AND circuit which has the off side of each flip-flop of register C entering it, and whose output recognizes the condition in question. The point is, that if the condition does arise, the total number of internal command lines may change from odd to even or vice versa. Thus, it is seen that the AND circuit must be essentially duplicated within the internal parity bit generator in order to obtain the proper parity bit under all possible conditions.

(4) The external command generator is provided with a parity bit generator and a parity checker. The external command lines are functions of the operation decoder and current status, so that by properly correlating the status code with the external command generated, the external parity bit generator should be kept reasonable in size. This parity bit generator is checked by its parity bit checker.

(5) The operation decoder may be checked by a circuit that detects no output or more than one output line up. A combination parity check on the operation word and a check of the decoder may be performed by the same circuitry in a scheme presented later on.

(6) The circuits labeled "COMPARE' " and "COMPARE" are checked by parity checkers A and B, respectively. To make this clear, assume thar register B is being incorrectly set to register A and that the output of "COMPARE" is erroneously up. The system will still be unable to proceed because the output of Parity Chk B remains up. The latter condition would hold because the erroneous setting of B to A would most probably result in the wrong bit count for B.

(7) Flip-flops 1 and 2, which make up a two-stage timing ring, may be checked for proper operation by comparing the actual time delay encountered while one of the flip-flops is on to the maximum (and minimum) allowable time delay for such a period. (Circuits for performing this check are not shown.) If the actual time delay exceeds the maximum (or is less than the minimum) allowable, an error flip-flop may be turned on and an error signal generated. Note: it may be undesirable to have the system halt immediately upon detection of such an error, since no incorrect results are generated because of it, but it has only slowed down the generation of the results.

(8) The switch which transfers data from A to B is checked when setting of register B is checked. Thus, if the switch is defective, some bit will be transferred incorrectly giving the wrong parity count, resulting in an error indication from the parity checker attached to register B.

(9) The two AND circuits, the inverter, and the Compare Response to Control circuits are checked by the time delay checks described under (7). These are all either timing circuits or feed directly to timing circuits. It should be noted that the checking for minimum delay is probably more difficult than checking that maximum delay is not exceeded, yet is necessary to check all the circuits of this section.

If all of the checking techniques listed above are utilized, the only remaining unchecked circuitry consists of checking circuitry. Checking of checking is second-level checking and as such, out of the scope of this paper, even if sometimes desirable. We assume that other techniques will be developed to process failures within such circuits. It seems a fairly reasonable assertion that regardless of the complexity of the checking circuitry attached to a computer, there will always exist some circuitry which may not be checked by hardware. Such circuits may lead to undetected error unless a vigorous effort at periodic inspection is instituted that will test such circuits for proper operation.

The efficiency of this checking scheme cannot be measured until it is applied to a specific case. This efficiency should be measured as the ratio total number of checking components (transistors, say) to total number of components checked.

In order to locate faults to a basic module, it might be advisable to provide several external command parity bit generators with corresponding checkers, so that within one basic module one obtains one completely checked (partial) command generator. There would be more but smaller command generators, each of which may be checked separately. In general, fault location can be improved by <u>detecting</u> an error in any of a group of smaller logical blocks. This principle can be applied to checking of any large logical block, such as operation decoder, status register, etc.

SYNCHRONOUS CONTROLS

In synchronous controls there is usually a "clock" (ring circuit driven by oscillator) which serves as a generator of standard timing pulses. Each action within the system is initiated by one of these pulses. The sequence of operations then takes place within the framework of the sequence of cycles. The designer must provide enough flip-flops to distinguish each different type of cycle. For different type cycles unique command lines must be generated. These flip-flops may be said to represent the internal status of the computer. Along with internal flip-flops, the control will contain such things as Operation Decoder, External and Internal Command Generators, etc., as shown on Figure 8.

A checking scheme that could be applied to this case is included in Figure 8. As in the case of asynchronous control, the Internal Parity Bit Generator is apt to become quite bulky because of the presence of recognition circuitry. Again, in a manner similar to the asynchronous case, the size of the External Parity Bit Generator may be kept reasonable by correlating the status code with the external commands generated.

An interesting means of checking the operation decoder and the operation register contents (on the assumption that a parity bit is used) has been described by J. V. Batley.[2] This scheme is illustrated for a 2-bit operation register in Figure 9. In this scheme, the outputs of the operation decoder feed a group of OR circuits in such a way that the inputs to all AND circuits feeding a particular OR circuit are all different lines. The outputs of the OR's are fed to EXCLUSIVE OR's so that eventually two lines are generated corresponding to "Operation

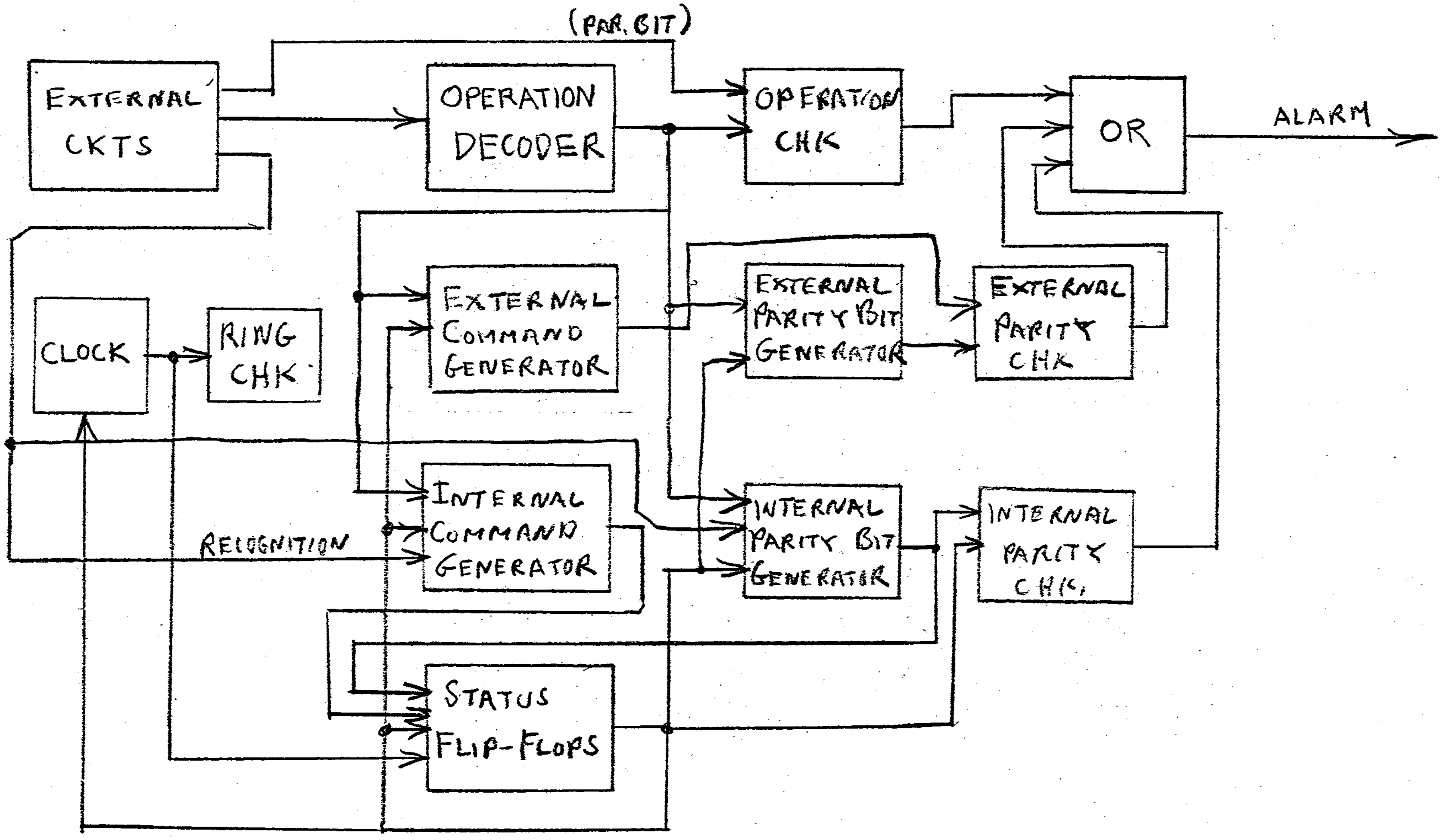


FIGURE 8
SYNCHRONOUS CONTROL EXAMPLE

F. B. H. 11/8/57

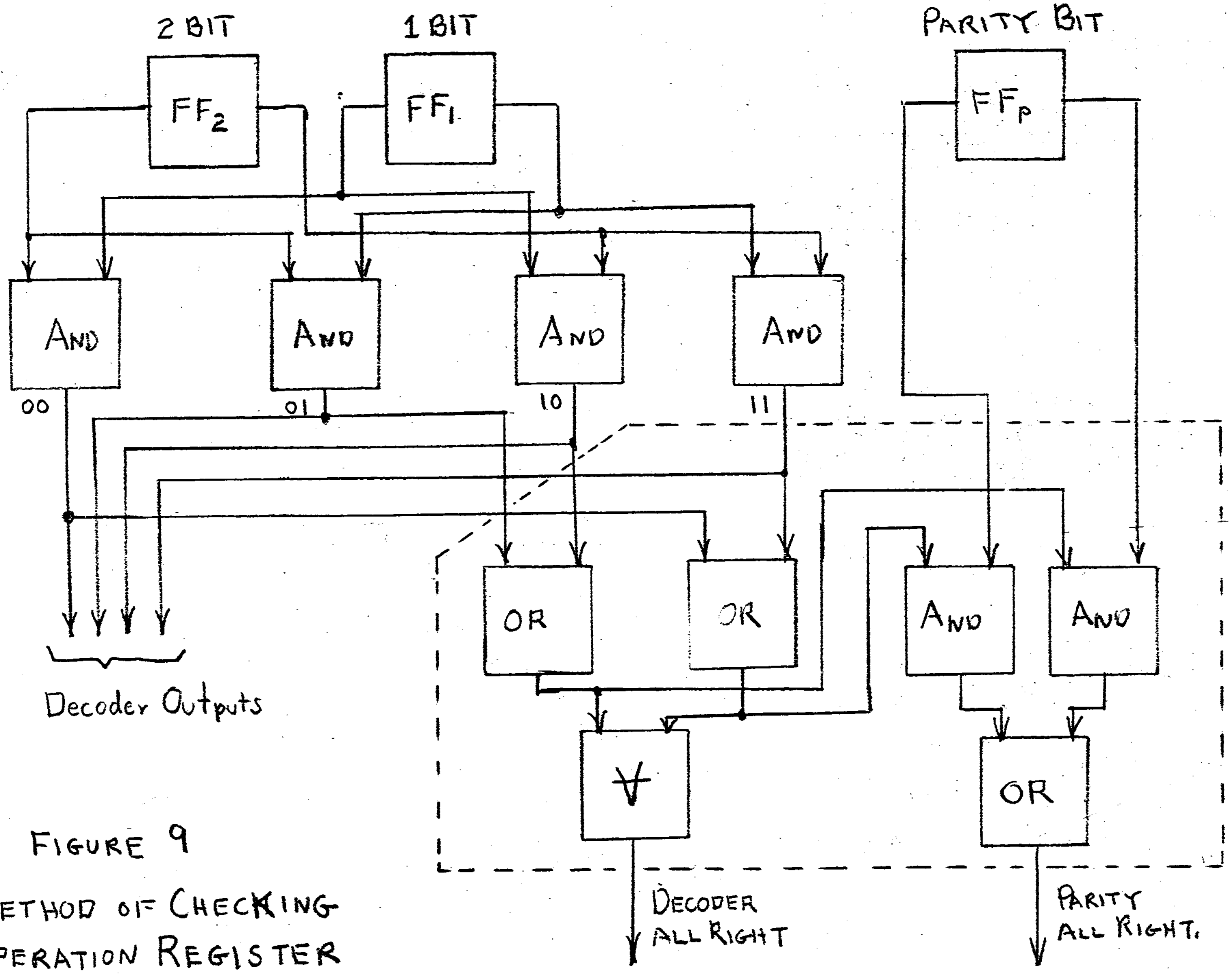


FIGURE 9

METHOD OF CHECKING OPERATION REGISTER AND DECODER
(Note: Even Parity Assumed)

F.B.H 11/8/57

Bits Odd". In the simple case illustrated, the OR's already determine the two required lines.

The check signals for the decoder and the operation-word parity are shown within the dotted lines. The main advantage of this method appears to lie in the fact that all failures in the decoder itself are checked except those which arise because of "output-with-no-input" failure in an AND circuit. The case not checked appears to require a double failure (at least for diode AND circuits). This scheme is probably more efficient than a detector of say "one-and-only-one line up", which utilizes a set of EXCLUSIVE OR's only. It is recommended wherever both a decoder check and a parity check on the operation register are necessary. It utilizes common equipment to perform the two checks. This scheme is also applicable to asynchronous systems.

For synchronous checking circuitry, the error detection may be said to be on a cycle basis. By this it is meant that the error lines are sampled each cycle, and in case of error, the machine may be halted.

A checking scheme for a command generator that works on an instruction basis has been described by J. V. Batley.[2] If the execution time for an instruction is a fixed number of cycles (as in fixed word-length machines), then for each cycle the total number of generated command lines is known (ahead of time) to be odd or even. Suppose the greatest number of commands occurring during a single cycle is N, and N flip-flops are provided. If the flip-flops begin with an even bit count (say all off), and if for each command generated within a particular cycle, one of the flip-flops is flipped, and if a dummy command is provided (where necessary) somewhere during execution time for each flip-flop, so that at the end of execution the flip-flops have again even parity, then errors in the command generator may be detected at the end of execution by using a parity check on the flip-flops.

The simplest case of this scheme arises when N equals one (1). This means that either no command pulse or just one command pulse is generated during each cycle. Suppose the flip-flop is off to start. Then, if there are an odd number of commands, there must be one cycle during which no commands are generated, so that time is free to flip the flip-flop by means of the dummy command. At the end of execution, an error is signaled by the flip-flop being ON, since for correct operation it would have been flipped an even number of times and so would be off. Note that detection of error does not occur until end of execution, so that detection is on an instruction basis rather than on a cycle basis.

This system appears to satisfy the requirement of continuous checking within the command generator for fixed instructions, but ignores the possibility that, for a particular instruction, the total number of occurrences of command lines may be altered by external conditions, such as overflows, signs of numeric fields, etc. Taking care of these exceptions and, in particular, checking the operation of the recognition circuitry involved, requires more dummy commands (analogous to the internal parity bit generator of Figure 8). This system offers possibilities of fault location by analysis of the contents of the flip-flops.

APPLICATIONS TO PROJECT 7000

In order to determine the applicability of any of the techniques described in this paper to the checking requirements of the Project 7000 machines, it would be necessary to become intimately acquainted with the structural details of these machines, and in particular with the logic of their execution controls. For a number of reasons this has not been possible up to now. Hence, we have had to consider the problem from a more general point of view, leaving the specific applications to the future. It is hoped that if certain areas of the system have forms similar to those presented above, the results of this investigation may be found useful.

Frank B. Hartman

Frank B. Hartman

FBH:vjh

References:

1. "The Effect and Use of Checking Circuitry", Maintenance Development Memo #9, July 29, 1957.

2. J. V. Batley, "Checking Instruction Control", Machine Organization Memo #55.