Modular Asynchronous Control Design

BERNARD J. NORDMANN, JR., MEMBER, IEEE, AND BRUCE H. McCORMICK

Abstract—**A modular design strategy for the implementation of control logic is discussed. After discussing several basic control modules, the overall design of control sequencing is described and a particular control sequence taken from the Illiac III computer is worked as an example of the technique. Finally, the diagnostic features incorporated as part of the technique are discussed and a comparison made with microprogramming.**

Index Terms—**Asynchronous control, control logic, control sequencing, microprogramming, modular design.**

I. INTRODUCTION

THE PURPOSE of this paper is to describe a modular design strategy for the implementation of control logic. Suitable general-purpose, modular, asynchronous control devices will be introduced and their use demonstrated in the control logic design process. The processes to be described were used in the design of the control logic of the Illiac III computer system, a large-scale pattern recognition computer designed at the University of Illinois.[1]

To accomplish this, we briefly describe the historical development of these modular control devices and their relationship to other similar work. Then we give a detailed description of their operation and their use in a fully integrated design process: from the specification of a control sequence, through the design of the control line drivers, to considerations of automated diagnosis. Finally, comparison with other control logic techniques, notably microprogramming, is made.

A. Historical Development

The control logic design technique to be described in this paper had its origin in an early theoretical formulation by Muller [1], [2], which in turn led to several ideas used in the design of the Illiac III computer at the Digital Computer Laboratory of the University of Illinois. The engineering result was called "speed-independent" asynchronous control [3]–[5]. This mode of control evoked reply signals from the data-flow logic being controlled to govern the sequencing from one step to the next. The "speed independence" requirement merely meant that the control logic would function regardless of how long it took the data-flow logic to complete its operation.

Manuscript received May 7, 1974; revised March 22, 1976.

B. J. Nordmann, Jr., is with the U.S. Naval Ordnance Laboratory, Silver Spring, MD.

B. H. McCormick is with the Department of Information Engineering, University of Illinois at Chicago Circle, Chicago, IL.

[1] Funded under AEC Contract AT(11-1)-1018.

Rigorous asynchronous design limits the possible control processes to basically an initiate-wait-continue type of operation. Because of the limited nature of asynchronous design, the Illiac III project staff decided to use what was basically a modification of the circuits used in the prior Illiac II computer control, updating these to current technology by using TTL. The result was a family of circuits which are called *control points*, with each processing unit in the system employing slightly different designs. Later a coherent design philosophy emerged [6]–[8], as well as a formalized diagnostic technique [9].

Most forms of asynchronous control operate in a fundamentally similar fashion. In particular, the Illiac III control points are almost identical in operation to Clark's macromodule concept of a *decision mode calling element*, which can initiate an operation and then wait for one of several returns [10]. The principal differences are in implementation and in the complexity of the operation to be controlled. Robinson [11] has also described a similar asynchronous control module which can be used with data-flow structures that are basically synchronous.

Still another group is Bell *et al.* who have described a set of register-transfer modules (RTM's) for high-level digital systems design [12]–[14]. Their system consists of various types of modules which allow digital systems to be expressed in a flow chart form with the complete construction information included. The K. evoke (Ke) module described in the RTM system is also very similar in operation to a control point.

In addition to the physically implemented asynchronous control modules, several other sets of modules have been suggested as the result of theoretical asynchronous control studies. In particular, Altman *et al.* [15], [16] and Patil [17] have proposed differing sets of such modules. In general, the theoretical approaches to asynchronous control have begun with the assumption of a separation of the data-flow structure from the control structure and then have proceeded with an analysis of the control structure in graph theoretical terms. This approach has also been taken by Luconi [18], Petri [19], and others.

B. Modular Design Strategy

The control organization of the Illiac III central processing unit (known as the Taxicrinic Processor) is an example of multilevel modular design. At the hardware level, the Taxicrinic Processor is divided into data-flow logic and control logic. Each of these areas is in turn subdivided: the

registers into various structural groupings, i.e., base registers, pointer registers, etc., and the control logic into functional instruction groups. A particular instruction within a functional instruction group is executed by activating appropriate control subsequences and thereby manipulating control lines to the data-flow logic. Thus the entire hierarchy of control logic for the processor consists of nested levels of control sequences, with each lower level performing more basic operations.

C. Control Sequence Description

A control sequence can be decomposed into a series of logical operations or *control steps* interspersed with *sequence steps* such as "branches" or "waits." The control steps include elementary logical/arithmetic operations on the data, transfers between registers, or the activation of other control subsequences. For each control step certain control lines must be activated and, upon completion of the operation, turned off. Then the next control step is activated, either directly, or by means of an intermediate sequence step.

Determining when an operation has been completed varies depending on the control philosophy used. In a *synchronous* system the time is determined by counting clock pulses for a time interval greater than the worst case time limit for the operation. In an *asynchronous* system the logic being activated produces a reply signal when it has completed its operation. In practice the production of this reply signal is often quite complex and for simple operations is not cost effective. As a result, a *pseudoasynchronous* philosophy is often used in which a "model" of the operation being performed (often a simple delay circuit) produces a "done" pulse at some adjustable time after the initiation of the operation. The control logic design strategy described here uses both asynchronous and pseudoasynchronous design philosophies.

II. Module Descriptions

Control steps are executed using a hardware device called a *control point* module. Sequence steps are implemented using *sequence* and *wait* modules. In the Illiac III system several different variants of these modules came into use. The versions to be described here incorporate those features found generally desirable and useful.

A. Control Point Implementation

The operation of a control point begins when it receives an *advance in* signal AI. It then performs its operation by activating a *task* signal T, which in turn activates the necessary control lines. Finally, when the operation has been completed, the *advance out* signal AO is activated, which in turn will activate the next control point.

The implementation of the control point module is shown in Fig. 1. The falling forward edge of the advance in pulse will set the input flip-flop so that on the rising edge of AI the task signal T will become active. A pseudoasynchronous delay model can be implemented by attaching one end of a capacitor to the charging pin CH and grounding the other end. The length of the T pulse (and the delay before the occurrence of the *delay out* signal DO) can then be controlled by the selection of the capacitor. If fully asynchronous operation is desired, pin CH can be grounded and T will remain active from the time the control point is activated until it is reset by some external signal.

Two reset signal options are provided: *normal return* NR and *interrupt return* IR. A logical '0' on either line will cause the control point to reset and will activate either the AO signal or the IO signal, respectively. Pseudoasynchronous operation is obtained by connecting DO to NR. Alternatively, NR can be connected to an external reply signal which will produce asynchronous operation. Note that the purpose of the IR and IO signals is to allow the logic being activated to indicate the occurrence of special conditions (such as an interrupt situation) which would call for a transfer of control to a control step other than the next one in the subsequence.

A CLEAR signal is used to initialized the control logic at turn-on time. The IN signal (an activation indicator) is used in diagnostics. The GOTASK signal will be discussed in conjunction with diagnostic techniques.

B. Sequence and Wait Implementations

The final action of any control point is the production of either an AO or an IO pulse signifying the completion of the control step. The routing of this pulse to the next control step to be executed is accomplished by a sequence step. In many cases, a sequence step is implemented with a single direct logic line. More complicated cases are handled by using combinational logic, which directs the advance out signal according to the status of various Boolean inputs. Alternatively, we can evoke a wait module which produces an output pulse after receiving input pulses on all of its input lines. Fig. 2 shows an example of a simple wait module.

Regardless of implementation, the effect of the sequence step is to deliver the advance out pulse of a preceding control point to the next control point in logical sequence. The falling edge of this negative pulse[2] sets the new control point and the subsequent rising edge will activate the new control step.

III. Control Logic Subsequences

A. Advantages of a Modular Approach

The advantages of a modular approach to the subsequence design are several. Each subsequence is designed

[2] For implementation reasons, most of the control point signals are negative-going pulses.

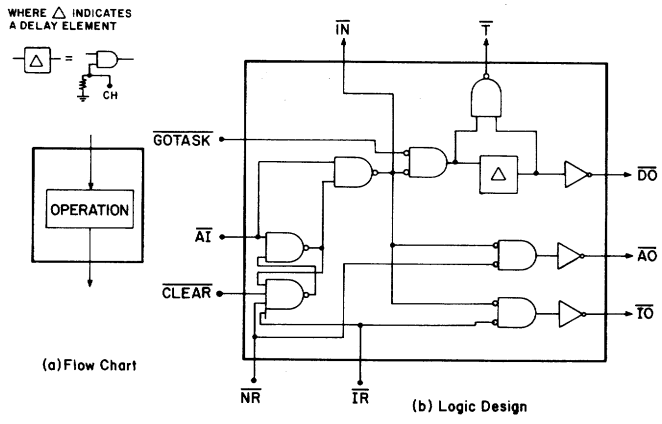


Fig. 1. Possible control point implementation.

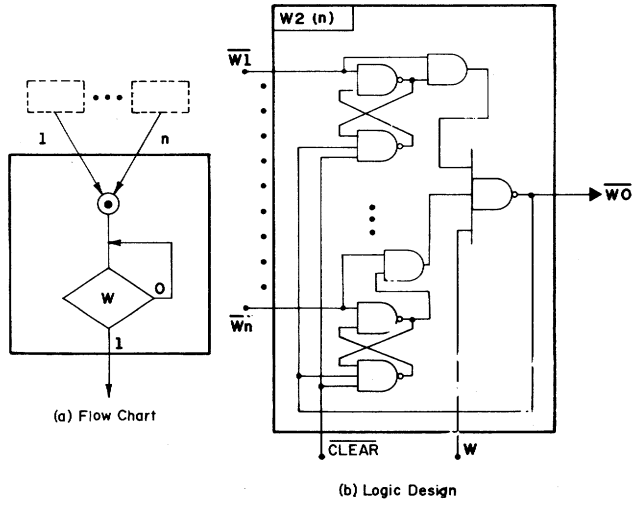


Fig. 2. Wait module.

as a logically complete entity and is usually fitted on an integral number of cards so that it can be checked out as a unit. The modular approach allows several designers to work on the control logic of a single processor with fewer problems of the interface definition. Furthermore, as will be seen in the next section, the design techniques used allow for a comparatively simple method of converting a flow chart (which is the initial level of control sequence design) into control logic which maintains the topological structure of the flow chart.

B. Calling of Subsequences

As mentioned previously, the control logic in the Illiac III central processing unit predominantly consists of subsequences of operations which are "called" by some other sequence. This structure simplifies the logic requirements by allowing common operations to be done using the same physical logic. The control logic is easier to visualize conceptually when it is placed in this modular format and is also easier to debug and check out.

Control logic subsequences resemble subroutines in software. Each sequence may have certain control flip-flop and/or control signals which must be preset to their desired states before the sequence is activated. Activation of the sequence is initiated by an external "calling" control point, which upon completion of the sequence is reset by the advance out signal of the final control step in the sequence. Activation is performed by a level signal which is "on" for the duration of the sequence execution. This use of a level signal has certain advantages since the task signal T can then be used to turn on control signals (i.e., "parameters") for the entire control sequence and the sequence activation signal is available for diagnostic purposes.[3]

C. Control Subsequence Preamble

Generally, the Illiac III central processor control logic subsequences make use of a fairly standardized "preamble." An illustrative example of this type of logic is shown in Fig. 3. The various control points which can activate the

[3] This scheme has a disadvantage in that the task signal (a level) cannot always be directly used as the advance in signal for the initial control point. In most situations involving a level signal, a control point can rest in the set position, and the level can be used as a rising input to set off the control step. However, if a decision must be made between two different control points with a level input, the method becomes unreliable, since any change in the parameters used in the choice may cause the "unchosen" CP to start. This problem is solved by having the level signal drive a one-shot or other impulse creating a circuit, the output of which is used to start the first control step. An alternative solution is to use a pulse-type task signal, but this then involves additional logic to obtain diagnostics and parameter selection.

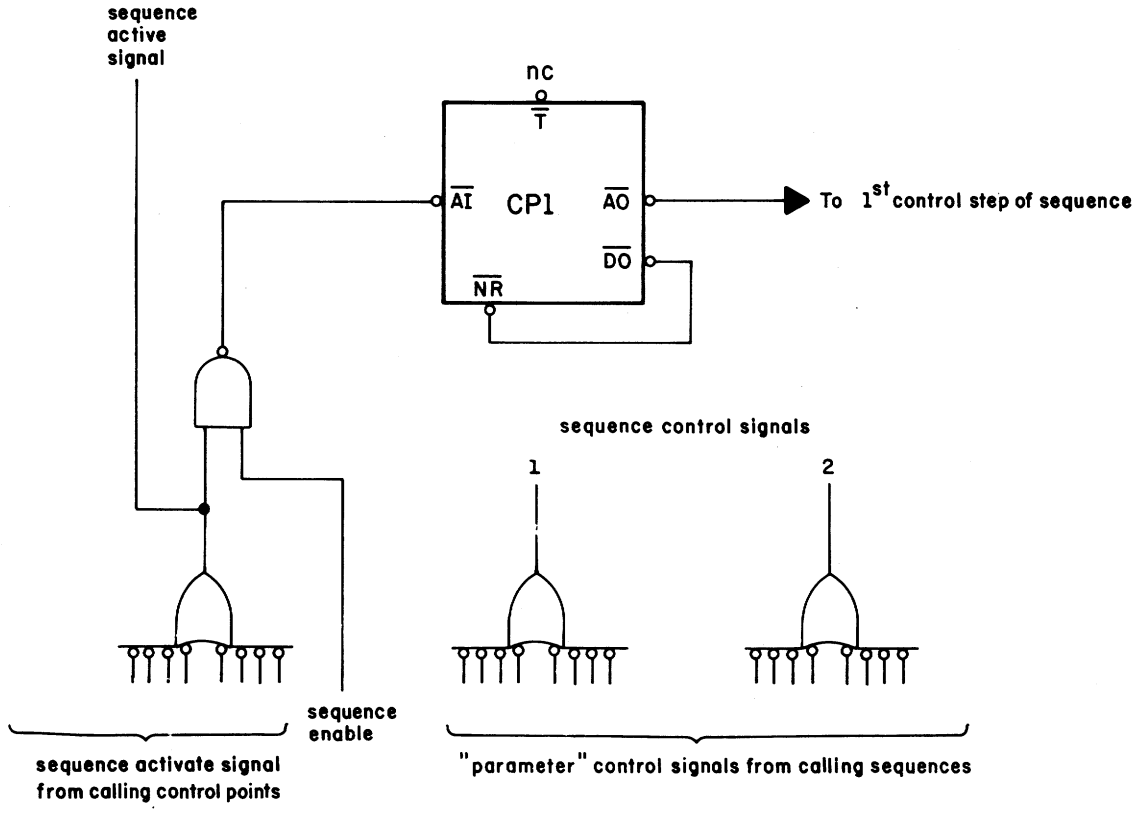


Fig. 3. Example of a control subsequence "preamble."

sequence have their task signals fanning into a single activate line (possible after several levels of fan-in logic spread throughout the physical mainframe). As part of the diagnostic hardware, a sequence halt signal is provided which, when activated, will prevent the control logic from proceeding. Since the activate line remains on, and since all of the subsequence activate lines are made accessible to the diagnostic hardware, a means is thereby available to trace the hardware call.

D. Sequence Returns

Once the sequence has been started, it will continue to run until it reaches a point where it must return. In general there can be either of two types of returns: a *normal return,* from which the calling sequence proceeds in a normal fashion by activating the AO signal, and the *interrupt return,* from which the calling sequence may have to perform some special repair sequence by activating the IO signal. The type of return which has been performed is differentiated by the return signal in the calling control point which the returning subsequence activated: NR for a normal return and IR for an interrupt return. Note that either of these signals will reset the control point flip-flop, but that each one only activates its own corresponding "out" signal.

At this point, a word might be said about interrupts. Throughout the Illiac III control logic a variety of so-called "interrupt conditions" can occur. A major reason for this is that a great deal of the supervisory bookkeeping is done by the machine hardware. As a result, the hardware control logic subsequences often encounter situations which must be "bucked" back to the supervisor program in the operating system. In addition, of course, the subsequences may also discover the more mundane interrupt situations involving memory faults, arithmetic overflow, etc.

The general interrupt philosophy used in the control logic subsequence design was that each subsequence must detect and identify any interrupt conditions which occur because of its own operations. If such a condition is found, the subsequence must attempt to return the machine to the state which existed when it was called and then make an interrupt return. If this can be done, then the sequence is "restartable" and there are no further hardware problems with the interrupt at least as that sequence is concerned.[4]

IV. Subsequence Design Technique

The fundamental problem in control logic design is to convert a description of a particular control process or

[4] Unfortunately, due to insufficient buffering and an occasional overambitious machine instruction, there are several cases in which the Illiac III central processor makes irrevocable changes in the data base before detecting an interrupt. Thus it became necessary to devise a logical structure which could stop for an interrupt in the middle of an instruction sequence and return later with the machine in the same state. Needless to say, this created several headaches. However, by more care in the instruction design, this problem can be eliminated entirely and only the traditional interrupt point between instructions need be provided.

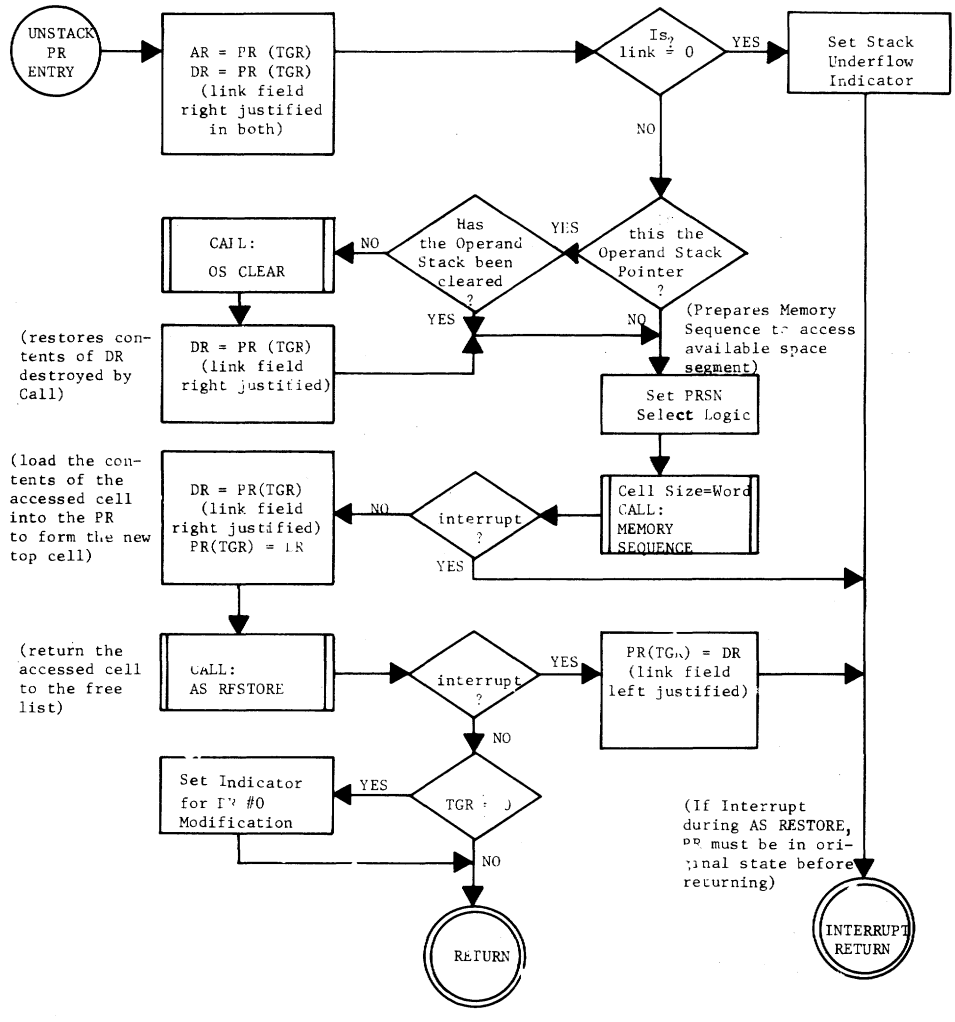


Fig. 4. UNSTACK PR sequence flow chart.

subsequence from a flow chart to a logical drawing from which a wiring list can be made. The primary advantage of using a modular control design technique is that this conversion process can be performed quickly, efficiently and with relative ease.

A. Sequence Flow Chart

The starting point in the design technique is a *sequence flow chart*. The level of detail in a sequence flow chart is such that the operations consist of register transfers, simple arithmetic operations, or any logical operation which might be performed by simultaneous activation of a set of control signals such as gates, enables, inhibits, etc. The task operations themselves are usually represented by Boolean expressions.

An example of a sequence flow chart is given in Fig. 4. This is the sequence flow chart for an Illiac III control subsequence known as UNSTACK PR. Basically, this sequence removes the top entry from a linked list and returns the entry to an available space list. This linked list is a very special form, called a pointer stack in the Illiac III system. There are 15 of these stacks and the top entry of each resides in a hardware register within the central processor. Within these registers, the link field is contained in the left half of the word. The main points in the flow chart which are important to its understanding are as follows:

1) TGR is a 4-bit register which indicates which pointer register stack is to be manipulated.

2) Certain stacks (i.e., #13 and #0) require special operations in addition to the normal sequence.

3) AR, DR, and LR are the names of general purpose registers.

4) AS RESTORE is a control subsequence which will return the cell whose address is in the DR to the available space list.

5) MEMORY is a control sequence which accesses a cell in memory. The cell's address must be contained in the DR. If MSTOR = 0, the sequence performs a read operation and leaves the data in the LR.

With this in mind we can now begin the design process.

B. Control Step Flow Chart

The first step is to redraw the flow chart in a very explicit form (called the "control step" flow chart) so that every control signal which must be influenced is expressed. Thus

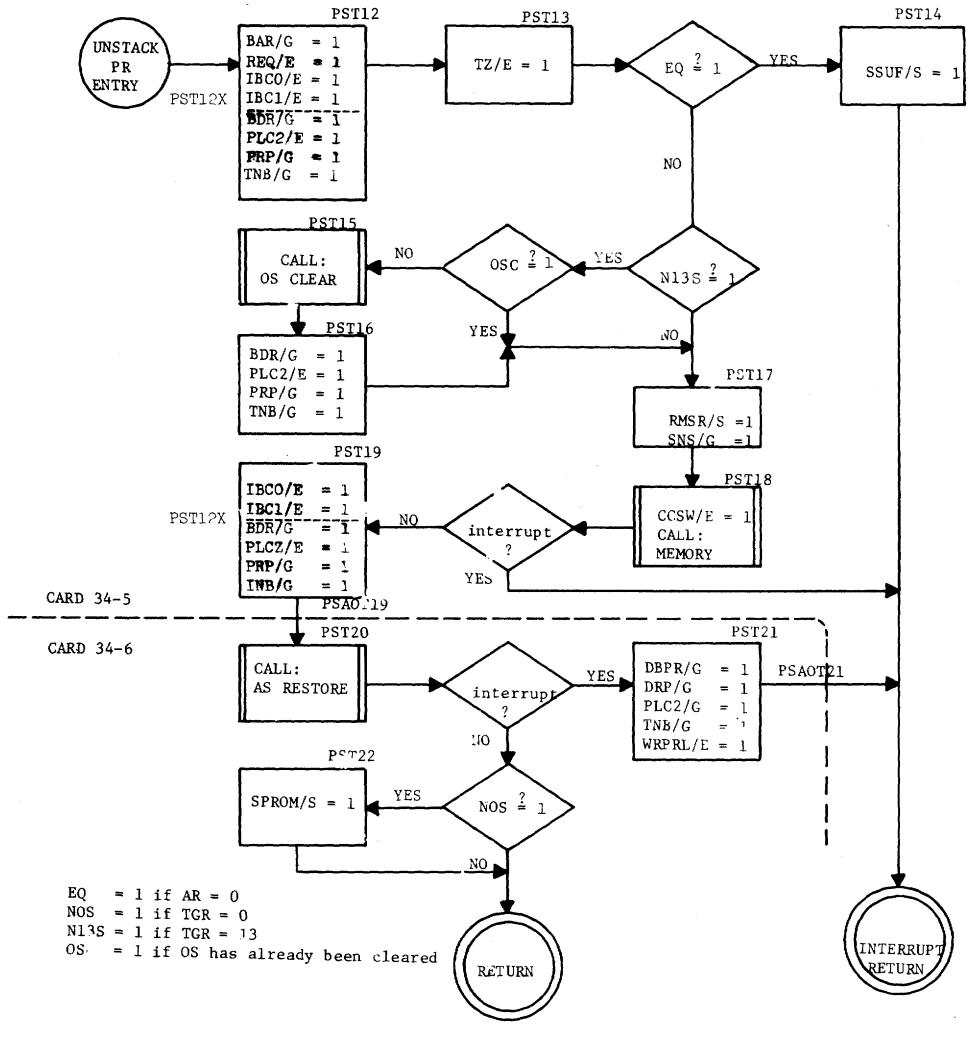


Fig. 5. UNSTACK PR control step flow chart.

if a particular operation consists of gating from one register to another while inhibiting certain bits, all the necessary gating signals which must be activated to accomplish the operation should be written down. *Each set of basic operations which can be performed in parallel should be written as the contents of one control step of the flow chart.* These control step blocks thus become in effect the list of control signals which must be turned on by the task signal of a given control point. The sequence logic between control points will be represented by the various decisions in the flow chart.

Fig. 5 shows the results of this conversion process for the UNSTACK PR sequence. Note that at this stage, each separate control step is clearly separated out and labeled (i.e., PST12, PST13, etc.).

Completion of the control step flow chart is the last design stage which must be performed by a designer intimately familiar with the total machine logic. From this point on, the design process becomes mechanical and can be performed entirely by support technicians. This modularity of the design process is especially important for implementing a large scale machine, such as the Illiac III, where only a very small number of the engineering staff can be expected to be familiar with the entire machine.

C. Control Logic Layout

The next design step is to draw the actual control logic. This basically involves converting every control step represented in the flow chart to a control point, producing the necessary sequence logic between control points, and designing any additional combinational logic which might be necessary for the operation of the sequence. The logic drawing for the first part of UNSTACK PR is shown in Fig. 6. The notational conventions used are described in the Appendix. Generally, we have tried to keep each sequence on one card or some whole multiple of cards. This allows easier debugging.

Although there are no examples in UNSTACK PR, it often happens that a control step will contain signals which are only activated if certain conditions hold. This type of control signal is called a *conditional task signal* and in the control step flow chart would be represented by a separate box which is entered if the condition is satisfied. The

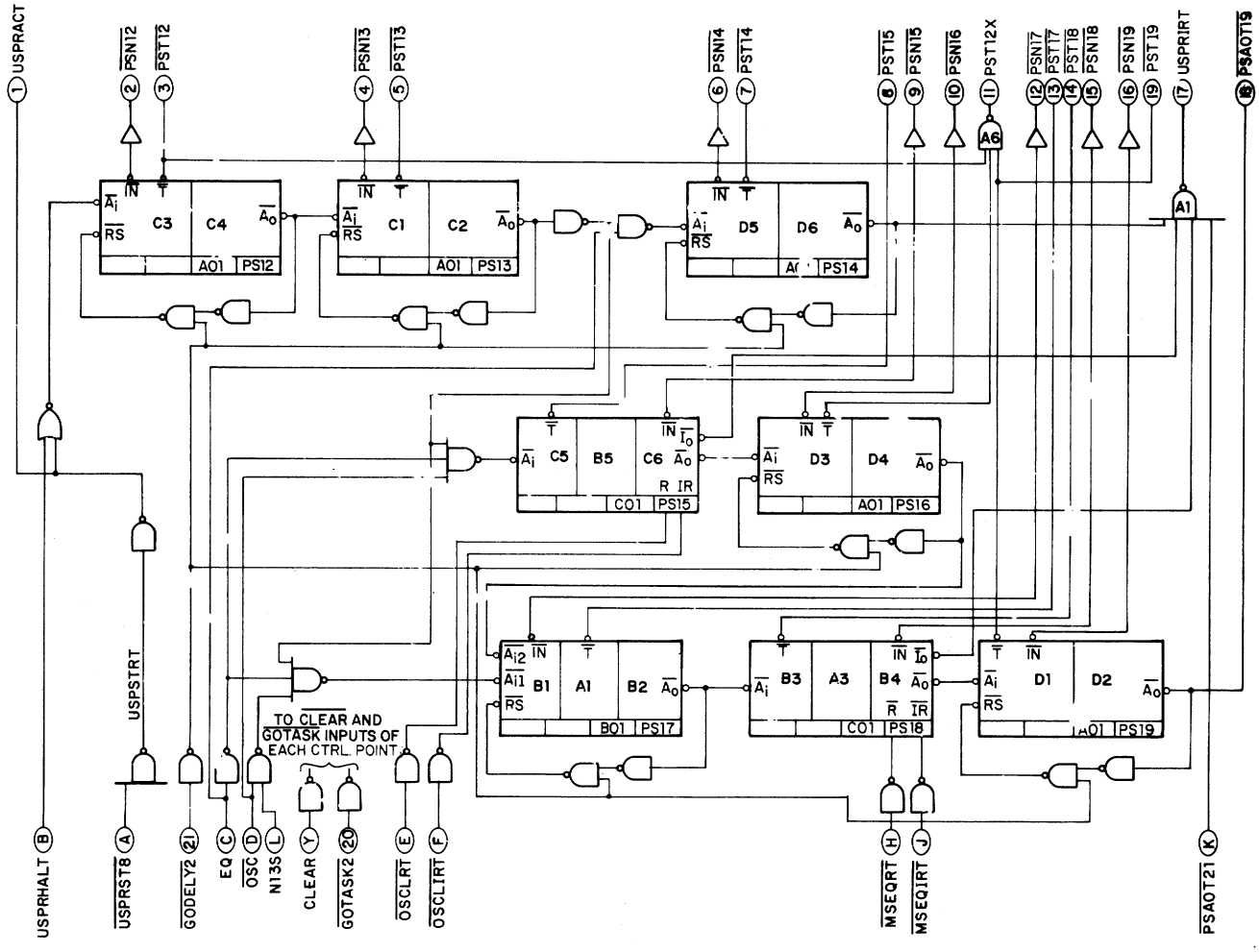


Fig. 6. UNSTACK PR logic drawing.

conditions are included on the sequence card and a separate pin is used for the conditional task output.

At this point, the logical design for the single sequence is finished. In the Illiac III central processing unit, the various subsequences are grouped according to their functional operations. Thus, for example, in the case being discussed, UNSTACK PR is grouped with those other subsequences which are concerned with the pointer registers and their stacks. The rationale for these functional groupings is that it tends to group those sequences which use the same control lines.

D. Control Signal Driver Design

Once all of the sequences in a functional group have been completed, the design of the control signal drivers can be started. The task and conditional tasks from all of the cards in the group are collected on a small number of driver cards. These cards collect the task signals from the group which turn on each particular control line and produce one output for each control line activated by the group. These outputs are then connected with the outputs of other functional groups (usually by means of common collector connections) to produce a single control line going to the logic to be activated. Thus we have a modular means of handling the very large fan-in of control task signals to the control lines themselves. (See, however, Section VI for an alternative strategy.)

The final step in the control sequence design process is to update the original control step flow chart so that it reflects the final logical design. In its general layout, the control step flow chart should now follow the logical design, i.e., the parts of the flow chart corresponding to each logic card should be delimited by dotted lines across the entire flow chart. Lines crossing these delimited areas as well as those coming from entry and exit circles represent specific control lines on the logic cards and if labelled, allow an easy comparison between flow chart and logic. Fig. 5 exhibits the layout for the UNSTACK PR sequence.

V. Control Point Diagnostics

The problem of diagnosing faults in control logic can be separated into two levels: the initial "local" checkout of a particular subsequence (intracard) and the "global"

checkout of the interactions between subsequences (intercard).

A. Local Checkout

In previously published work [9], [20], Rey has described an automated test procedure which can be used to analyze and then test each control logic card. The essential analysis technique involves:

1) partitioning the control design into "independent subnetworks";
2) finding the tests for these subnetworks; and
3) concatenating these tests according to a set of rules which take into account the connectivity properties of the control network.

The tests are physically applied using a PDP-8/e minicomputer connected to the card through a special interface called the sequence tester. The tester submits input patterns to the cards and preprocesses the card response before transmitting it to the PDP 8/e. A more detailed explanation of the test procedure along with the techniques used in the initial circuit analysis can be found in the cited references.

B. Global Checkout

In the "global" checkout the PDP 8/e again plays a part, this time as a "console processor" working through a special interface to the central processing unit of the Illiac III. The interface allows the console processor to read and/or change the data in any register in the machine and also to issue a very simple set of debugging commands.

There are two goals in the second level of diagnosis:

1) to check the control sequence operation with respect to calls, parameter passing, an other subsequence interactions; and

2) to check the fan-in logic connections to the data flow logic to ensure that each task signal does indeed cause the desired action to occur in the bowels of the machine.

C. Control Point Hardware Aids

The design of the control logic includes several features to facilitate the debugging process. These include:

1) a method of halting a normally running sequence whenever it "calls" an arbitrary picked subsequence or set of subsequences;

2) a method of determining the current location of control in a stopped sequence; and

3) a method to provide step-by-step execution of a control sequence.

The sequence halt feature utilizes a special multistate sequence halt selector which can be set by commands from the console. If the selector bit for a particular sequence is set, the sequence halt line in that sequence's "preamble" will be activated and the execution will halt when the sequence is reached. The sequence can be made to continue by using a momentary override of the sequence halt selector, thereby disabling the halt condition.

The current location of control is, by definition, the control point which is currently activated to perform some operation. In any particular sequence, there can be only one such control point. However when one subsequence is "called" by another there can be several active control points, since a calling control point remains active during a call, and the called subsequence will also have an active control point. In the Illiac III diagnostic logic, the control point indicator signals IN are encoded on a bus so that each control point produces a number corresponding to its name on the control step flow charts. In order to avoid the problem of several active control points during subsequence calls, the control points which perform calls are not encoded. Instead each subsequence has a status line indicating if it is active. By establishing a hierarchy of sequences based on their calling patterns, the subsequences can be broken down into levels such that no two subsequences within a given level can be on simultaneously. Thus an encoding scheme by levels can be used to specify the flow of control through the various subsequences. In order to specify any control point in the machine one need only concatenate the individual control point encoding with the active subsequence information.

D. Use of GOTASK and GODELY

The implementation of the step-by-step control mode of operation involves the use of two control signals, GOTASK and GODELY, which are sent to every control point. These control signals are generated using the logic shown in Fig. 7. Note that when the machine is running normally, both signals will be active. In the step-by-step mode, only one of these two signals will be active, with the normal rest state being GOTASK = '1' and GODELY = '0'.

Stepping through a sequence is illustrated by the two control point sequence shown in Fig. 8. When the GOTASK signal first goes to 1 (assuming an immediately prior control point has activated CP#1), the task signal, CPT1, will turn on. After a short delay caused by CP#1's delay element, CPT1 will turn off again. However, since GODELY = '0', the delayed signal will not be able to set the next control point or reset itself. Thus CP#2 will be unable to begin and the results of CP#1's operation can be inspected. At this time, CP#1's indicator, CPIN1, will be active, but CP#2's will not.

When the engineer decides to continue, the console processor will cause the single step line to be activated, which activates the GODELY/GOTASK one-shot. When GOTASK becomes '0' and GODELY becomes '1', the delay signal will propagate through and set CP#2 and reset CP#1. After a short delay, advance out of CP#1, CPAO1, will return to '0'. This would normally cause the next control step to start except that GOTASK is now '0'. However, since the timing of the GODELY/GOTASK one-shot is set to be only moderately larger than the longest possible delay in a control point's resetting loop, the rest state of

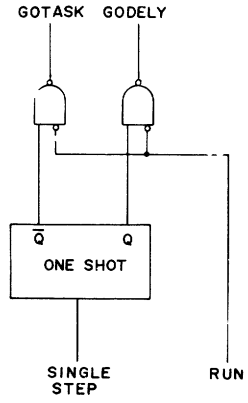


Fig. 7. Logic for production of GOTASK and GODELY signals.

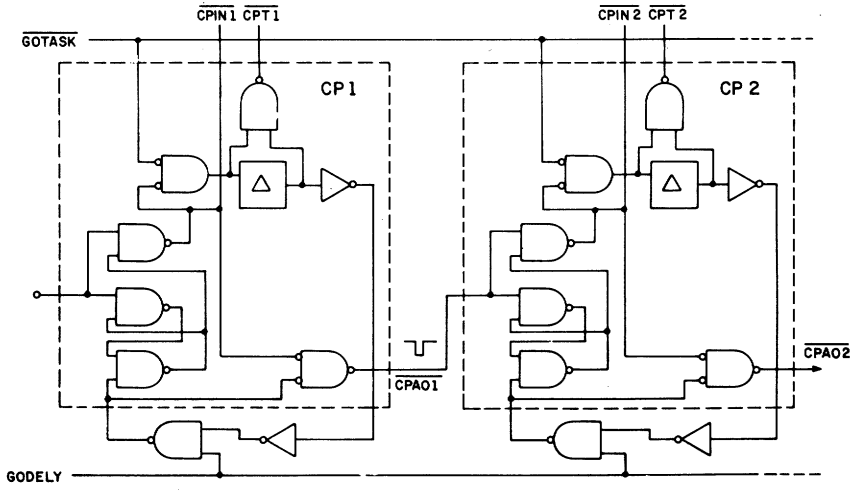


Fig. 8. Control point sequence showing the use of GOTASK and GODELY.

GOTASK = 1 and GODELY = 0 is rapidly reached and CP#2 becomes fully activated and left to perform its task.

The operation just described used pseudoasynchronous control points as examples. The technique also works for asynchronous tasks provided the tasks are not forced to halt by the operation of GOTASK and GODELY. If they are not, their return signals (levels, not pulses) will eventually come back and be halted by the '0' state of GODELY; otherwise, the technique remains the same.

The major points in favor of this stepping scheme are its simplicity, its requirement for only two control signals throughout the entire machine, and its ability to let each control step be performed at the speed which it would normally operate in a nonstepping mode (i.e., task signals remain on for the same length of time in either mode). The "artificial" delays occur after the task is completed and after the step sequencing logic has made up its mind and committed itself. The major disadvantage is that it is necessary to single step through every control step in a sequence to stop at a single intermediate point.

Together the single step and position location facilities allow the checkout procedure to proceed on a step-by-step basis throughout the machine. By using a table lookup scheme in the console processor, the sequence activation information can be converted to the original alphanumeric coding used in the documentation, with the result that the unit can be checked out from the console using the control point flow charts.

It can be argued that the diagnostic scheme just outlined is a rather expensive and complicated technique just to check out the operation of sequence interactions and the proper wiring of the task signals. For a production model machine, this may be true. However, in a highly experimental machine with many novel instruction sequences such as the Illiac III, a large number of control sequence errors must be anticipated, despite the fact that considerable simulation was done prior to actual construction. In addition, although the concepts in the diagnostic techniques are somewhat complicated, the additional logic is simple and can be added in a more or less mechanical fashion.

VI. Control Point Design Versus Microprogramming

The control point design technique outlined above has several interesting similarities and contrasts with microprogramming control techniques. In the latter, a microprogram controller operates with a read-only memory (ROM) to execute a control sequence and turn-on control signals.

Both techniques utilize *task signal* producing logic and *control signal* producing logic (Fig. 9). The control point design produces task signals as a direct result of the activation of control points. These task signals are transmitted to the control line driver logic to activate the appropriate control signals. By comparison microprogram design uses a programmed selection of a particular ROM word to activate a specific task and the control line fields within the selected ROM word then activate the appropriate control lines.

The economic tradeoffs between control point and microprogramming design appear to hinge on the scale and applications of the machine to be designed. For small designs the control point technique has a low "initial cost," since the control point modules are self-contained. However, as the designs become bigger, the number of control points increases and the fan-in problems for the control line drivers become worse. Microprogramming, on the other hand, has a higher "initial cost" because the microcode controller must be designed before any control functions can be implemented. As the number of control sequences become larger, the incremental microprogramming cost is basically the cost of increased memory. Thus for a constant number of control lines, as the number of control sequences is increased, a crossover point occurs where for larger machines the microprogram approach is cheaper and less complicated.

Hybrid modes of control implementation would appear appropriate for systems of medium complexity. For example, it would appear advantageous to generate task signals using the basic control point idea but then create control signals, as in microprogrammed control, by the use of a ROM. The resulting control system has the advantage of low overhead because of the absence of a microcode controller while retaining the advantage of low-cost, regular control line driver logic as implemented in a microprogrammed control.

To implement a large scale computer system, it would appear reasonable to retain a mixed control point microcode controller design. The microcode controller would activate subsequences while delegating individual register-to-register transfers to the subsequence control point logic. Both the number of words needed to execute a particular macro-instruction and the length of the microinstructions are considerably reduced by this strategy.[5] The problem of large-scale computer control logic design thus reduces to choosing a set of basic control functions which are activated as "subroutines" by microcode sequences. If the data flow design is also on a functional modular basis, the basic control functions can be tied in a one-to-one relationship and thus minimize the control line fan-in problem still further. The computer architecture thus takes the form of a group of functional blocks, each with local control and buffering, interconnected by buses and controlled centrally by a microcode controller. In many respects this is similar to the structure proposed in RCA's LIMAC concept [21], [22], although as presently conceived, the functional units could probably not be manufactured as single LSI chips if only because of their experimental status. However, once we know what is appropriate, the LSI fabrication of the functional units would be the next logical step.

[5] It should be noted in reference to footnote 4, that such a scheme would greatly ease the Illiac III interrupt problem since interrupts could then be bucked back to the beginning of the last microstep which would then be the only control information to be stored.

VII. Summary and Conclusions

This paper has discussed the techniques used in the design of the modular asynchronous control logic of the Illiac III computer. After discussing several basic control modules, the overall design of control sequencing was described and a particular control sequence was worked as an example of the technique. Finally, the diagnostic features incorporated as part of the technique were discussed and a comparison with microprogramming was made.

We have tried to show how the modular design of a control logic system, at both the functional and component levels, can lead to a conceptually simple design process. The use of standardized control point modules allows the control logic design process to proceed in a relatively mechanized manner once the control sequence algorithm has been specified.

The main advantages of modular control point design would appear to be in small units or in larger units when the data flow logic is highly variable with time as in some experimental machines. A possible merge of the modular control point and microprogramming techniques might occur in larger machines whose main microcode controller operates various numbers of specialized functional units implemented with control points.

Appendix

Control Point Notation Conventions

In actual use control points are represented on logic drawings by one of the symbols shown in Fig. 10. The CLEAR and GOTASK signals are omitted from the input signals since their use is always the same and their presence would have cluttered up the symbol. They are always connected to standardized input pins on the card. The upper part of the symbol is divided into either two or three boxes (depending on the control point variant being used), which represent the chips used to implement the control point. The location of these chips on the circuit board is indicated by the letter and number at the center of each box.

The four boxes in the lower part of the symbol are used to describe parameters for the control point. The leftmost box contains the control point delay in nanoseconds. The second box contains the capacitor value needed to generate this size time delay. The third and fourth boxes contain the control point variant name and the unique name for that particular control point, respectively.

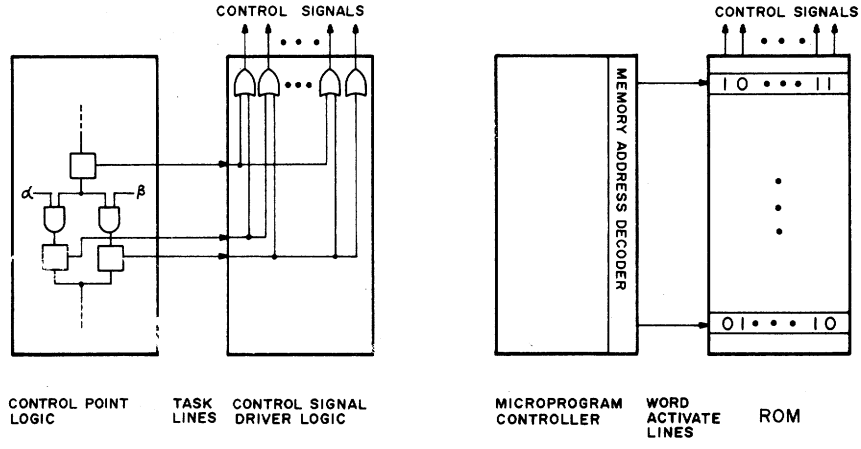


Fig. 9. Structural comparison of control point and microprogrammed control systems.

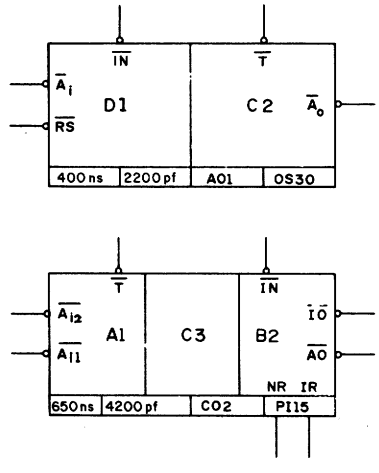


Fig. 10. Control point symbols.

In general, the individual control point names are classified into functional groups corresponding to the functional grouping of the sequences. Each group is assigned one or two characteristic letters (i.e., M for memory sequence, MU for main utility sequences). These letters are prefixed to all of the signals related to each individual control point. For example, the main utility sequences use control points whose names are designated MUn where n is a 1 or 2 digit number, optionally followed by a single letter a, b, c, or d. Thus if it is necessary to assign a name to any of the signals directly attached to one of those control points, they would conform to the following format:

advance in	AI	MUAIn
advance out	AO	MUAOTn
indicator	IN	MUNn
interrupt out	IO	MUIOTn
task signal	T	MUTn.

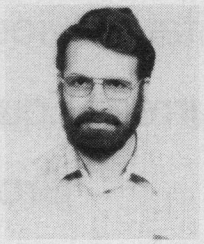
ACKNOWLEDGMENT

The evolving design of the Illiac III control point technique has been due to the contributions of many people. K. C. Smith, R. T. Borovec, R. M. Lansford, D. E. Atkins, and L. N. Goyal contributed early ideas on the modification of the original Illiac II design. R. G. Martin and C. A. Rey helped in the later development of the control point design philosophy. C. A. Rey was principally responsible for the automated diagnostic technique for evaluation of individual sequences.

REFERENCES

[1] D. E. Muller, "Asynchronous logic and application to information processing," *Switching Theory in Space Technology*, Aiken and Main, Eds. Stanford, CA: Stanford Univ. Press, 1963, pp. 289–297.
[2] D. E. Muller and W. S. Bartky, "A theory of asynchronous circuits," *Ann. Computation Lab. of Harvard Univ.*, vol. 29, pp. 204–243, 1959.
[3] D. B. Gilles, "A flowchart notation for the description of a speed independent control," Dept. of Comput. Sci., Univ. of Ill., Urbana, File 386, Aug. 1961.
[4] J. E. Robertson, "Problems in the physical realization of speed-independent control," Dept. of Comput. Sci., Univ. of Ill., Urbana, File 387, Aug. 1961.
[5] R. E. Swartout, "One method for designing speed-independent logic for a control," Dept. of Comput. Sci., Univ. of Ill., Urbana, File 388, Aug. 1961.
[6] R. G. Martin, "Standardization of control point realization," Dept. of Comput. Sci., Univ. of Ill., Tech. Rep. 400, May 1970.
[7] C. A. Rey, "Control point design using modular logic," Dept. of Comput. Sci., Univ. of Ill., Tech Rep. 463, June 1971.
[8] B. J. Nordmann, "Illiac III computer system manual: taxicrinic

processor vol. 2," Dept. of Comput. Sci., Univ. of Ill., Tech. Rep. 475, Aug. 1971.
[9] C. A. Rey, "Control point strategy and its automated diagnosis," Dept. of Comput. Sci., Univ. of Ill., Tech. Rep. 450, June 1971.
[10] S. M. Ornstein, M. J. Stucki, and W. A. Clark, "A functional description of macromodules," *1967 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 30, Washington D.C.: Thompson, 1967, pp. 337–355.
[11] D. M. Robinson, "Digital system design with control modules," in *Proc. 7th Ann. IEEE Computer Society Int. Conf.*, pp. 207–210, Feb. 1973.
[12] C. G. Bell and J. Grason, "Register transfer modules (RTM) and their design," *Computer Design*, May 1971.
[13] C. G. Bell, J. L. Eggert, J. Grason, and P. Williams, "The description and use of register-transfer modules (TRM's)," *IEEE Trans. Comput.*, vol. C-21, pp. 495–500, May 1972.
[14] C. G. Bell, J. Grason, and A. Newell, *Designing Computers and Digital Systems*. Maynard, Mass.: Digital Press, 1972.
[15] S. M. Altman and A. W. Lo, "Systematic design for modular realization of control functions," *1969 Spring Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 34, Washington, D.C.: Thompson, 1969, pp. 587–595.
[16] J. Bruno and S. M. Altman, "A theory of asynchronous control networks," *IEEE Trans. Comput.*, vol. C-20, pp. 629–638, June 1971.
[17] S. S. Patil, "Coordination of asynchronous events," M.I.T. Project MAC, Tech. Rep. MAC-TR-72, Ad 711-763, May 1970.
[18] F. L. Luconi, "Asynchronous computational structures," M.I.T. Project MAC, Tech. Rep. MAC-TR-49, Feb. 1968.
[19] C. A. Petri, "Communication with automata," Griffis Air Force Base, Supplement 1 to Tech. Rep. RADC-TR-65-377, Vol. 1, 1966.
[20] C. A. Rey, paper presented at the 6th Ann. Princeton Conf. on Information Sciences and Systems, Mar. 1972.
[21] H. R. Beelitz, S. Y. Levy, R. J. Lindhardt, and H. S. Miiller, "System architecture for large-scale integration," *1967 Fall Joint Comput. Conf., AFIPS Conf. Proc.*, vol. 31. Montvale, N.J.: AFIPS Press, 1967, pp. 185–200.
[22] S. Y. Levy, R. J. Lindhardt, H. S. Miller, and R. D. Sidnam, "System utilization of large-scale integration," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 562–566, Oct. 1967.



Bernard J. Nordmann, Jr. (S'64–M'66) was born in Lawton, OK in 1943. He received the S.B. and S.M. degrees in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1966 and the Ph.D. degree in Computer Science from the University of Illinois, Urbana, in 1972.

From 1966 until 1971 he was employed as a Research Assistant on the Illiac III Project at the Digital Computer Laboratory of the University of Illinois. Since then he has worked at the U.S. Naval Surface Weapons Center in Silver Spring, MD, where he is involved in the design of small-scale special purpose digital systems.

Dr. Nordmann is a member of the Association for Computing Machinery and was an IEEE delegate to the 1973 Popov Society Conference in Moscow.

Bruce H. McCormick was born in Oak Park, IL in 1928. He received the B.S. degree in physics from Massachusetts Institute of Technology, Cambridge, in 1950 and the Ph.D. degree in physics from Harvard University, Cambridge, MA, in 1955. He had a Fulbright Scholarship at Cambridge University, England, from 1950–52.

After working as a postdoctoral fellow at Brookhaven National Laboratory (1955–57) and as Staff Physicist at the Lawrence Radiation Laboratory (1957–60), he joined the staff of the University of Illinois at Urbana. While there he was Principal Investigator on the Illiac III Computer Project (1961–72). He is currently Head of the Department of Information Engineering at the University of Illinois at Chicago Circle in Chicago. His current interests are design and organization of computer systems, image processing, and biomedical applications.

Dr. McCormick is a member of the ACM and was a Visiting Lecturer for the ACM and also the IEEE Computer Society's Distinguished Visitors Program. He has served as Chairman of the NIH Computer & Biomathematical Sciences Study Section.