LIBRARY OF THE
UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN
510.84
Il6r
no.111-130
cop.3

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of baaks are reasans far disciplinary action and may result in dismissal fram the University.

To renew call Telephane Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

BUILDING USE ONLY

SEP 21 1980

SEP 20 1980

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/furtherstudiesin130swar

Il 6r
no. 130
cop. 3

UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DIGITAL COMPUTER LABORATORY

REPORT NO. 130

FURTHER STUDIES IN SPEED-INDEPENDENT LOGIC FOR A CONTROL

by

Robert Earl Swartwout

December 13, 1962

(This work is being submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Electrical Engineering, December 1962, and was supported in part by the Office of Naval Research and the Atomic Energy Commission under contract No. AT(11-1)-415.)

UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DIGITAL COMPUTER LABORATORY

REPORT NO. 130

FURTHER STUDIES IN SPEED-INDEPENDENT LOGIC FOR A CONTROL

by

Robert Earl Swartwout

December 13, 1962

(This work is being submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Electrical Engineering, December 1962, and was supported in part by the Office of Naval Research and the Atomic Energy Commission under contract No. AT(11-1)-415.)

cop 3

ACKNOWLEDGMENT

The author wishes to express his appreciation to Professor James E. Robertson for his counsel and guidance in the preparation of this thesis and for the knowledge gained of the fundamentals and subtleties of the computer art as the result of countless discussions.

The author also wishes to thank Professors R. E. Meagher and A. H. Taub as Heads of the Digital Computer Laboratory for their encouragement and support in the form of research appointments. The author would also like to thank Professor E. C. Jordan of the Electrical Engineering Department for the opportunity to teach and to do graduate study at the University of Illinois. There are many others, too numerous to mention, in both the Digital Computer Laboratory and the Electrical Engineering Department to whom the author is indebted.

The author wishes to express his appreciation to Dean C. A. Arents and Professor E. C. Jones, Sr. of the College of Engineering of West Virginia University for the opportunity to complete this dissertation after joining the faculty of West Virginia University.

The author greatly appreciates the opportunity which was his in working with Professor David Muller and studying the theory and practical aspects of Speed Independent Circuitry. It was a privilege to be a member of the design team which created the first major computer logic utilizing Speed-Independent Circuitry.

Finally, the author would like to thank Dr. Edwin C. Jones, Jr. and Dr. John O. Penhollow for their friendship, for their comments and criticisms and for the mutual encouragement gained from innumerable hours of discussion and study.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

LIST OF TABLES

LIST OF ILLUSTRATIONS

1. INTRODUCTION

1.1 Computer Reliability and Maintenance

Anyone who has operated any form of digital computer, be it asynchronous or synchronous, large or small, fast or slow, will agree that such machines are useful only so long as they operate correctly and consistently. As the new high-speed digital computers have been designed to operate faster and faster, and as they have grown larger and larger in size, the requirements for malfunction-free [IR-1]* operation by each of the circuit elements within the machine have become extremely high. Present day computers are performing several millions of basic decisions per second and there is little doubt that future computers will operate many orders of magnitude faster. One can quite easily determine that if a reasonably large digital computer is to operate without a malfunction for only one day at a time, many of its circuit elements will be required to perform as many as 10^{15} operations without error. Although some degree of reliability can be attained by merely selecting reliable components, reliability of this magnitude can only be attained through a combination of techniques. That is, reliable components must be used; plus the utilization of proper fabrication techniques; plus the use of a design method that will produce a logical configuration proven to operate as free of malfunctions as possible.

Despite the seemingly unattainable reliability requirements caused by the increased size and higher speed of computing machines, the newer machines will be larger and they will operate faster than their predecessors. As a

---

* This symbol [IR-1] refers to a reference which will be found in the bibliography listed opposite this same symbol. This particular reference is to the IRE Dictionary of Electronic terms and symbols from which the designation "malfunction" was taken rather than the more common word "error." According to this dictionary, an error results from approximations used in numerical methods, a mistake results from incorrect programming, and a malfunction results from failures in the operation of machine components such as gates, flipflops, amplifiers, etc.

result, some "lost time" due to malfunctions will be unavoidable. It is relatively easy to visualize the problems associated with locating a defective element within a computer containing 10,000 or more circuits. For this reason the computer's circuitry must also be designed so that it is reasonably easy to service as well as to provide the reliability mentioned earlier.

Dependable operation is needed in all sections of the computer; however, certain portions of the machine must be more reliable and must be easier to maintain than the rest of the machine. When a circuit within one of the Arithmetic Units of a computer malfunctions, it is nearly certain that an incorrect result to a calculation will be given. Eventually this will be detected and some corrective maintenance action taken. An arithmetic malfunction can usually be localized through the use of diagnostic routines. On the other hand, if a circuit within control malfunctions, the chance of locating this trouble through the use of even the most elegant routines is much less than for arithmetic malfunctions. This is in part due to the fact that the incorrectly operating circuit may cause the diagnostic routines to be improperly executed, thus causing the operator distress if not outright frustration.

A possible second reason for requiring the design of control circuitry to be more dependable is that in many computers the Arithmetic Units can be serviced by replacement of large subsections of logic whereas the same is not true for control. If a replacement is not available, the control must be serviced in the computer, whereas the replaced subsection of Arithmetic Unit logic can be serviced "off line."

For these two reasons, among others, it is felt that the control for an Arithmetic Unit should operate with greater reliability and be easier to service than any other portion of the machine. This dissertation will be concerned with a design method for producing logical configurations that operate as free of malfunctions as possible and which are relatively easy to maintain.

1.2 Logic Design Methods

One of the early attempts to formulate a design method that would produce a reliable logic was Huffman's, who stated in September, 1955, that one could design "Hazard-Free" [HU-2] logic. Concerning a network designed using his method he stated:

> "The terminal behavior of the resulting hazard-free networks are independent of the timing of the contacts (in the case of relay networks) or of the delays along the signal paths (in the case of an electronic gate circuit)."

The design of a network using this method involved a Karnaugh Map type of method for designing logic free of first-order (static) and second-order (dynamic) hazards. His paper alluded to methods whereby higher-order dynamic hazards could be eliminated from logic; however, it did not develop them.

In December of 1955 Muller published a report [MU-1] in which he described Speed Independent Asynchronous Circuits. To emphasize the importance of such circuits he claimed:

> "One might think that in order to design asynchronous circuits one would have to keep close account of the time taken by the various elements in acting so that the entire circuit would behave in the way intended. This is true in some designs, but it is possible to design circuits in such a way that the relative speeds of the elements do not affect the over-all behavior of the circuit."

Immediately following this, Muller and Bartky released three more papers [MU-2, MU-3, MU-4] in which they developed a rigorous mathematical description for each of the subclasses within the class of speed-independent circuits.* In the Huffman terminology, speed-independent logic is free of all hazards of all orders. The preliminary design report for the new Illinois computer [DC-1] stated that,

---

* Within the class of speed-independent circuits there are four subclasses. Throughout this dissertation the term speed-independent circuits will be used to mean the entire class. This includes all circuits which are semi-modular, distributive, or totally sequential, as well as those which are just speed-independent.

> "Speed Independent circuits have properties which are of particular importance in the design of reliable asynchronous circuits."

As a result, it was decided that the Arithmetic Control for the new Illinois computer (abbreviated DC)* should be designed to operate in a speed-independent manner. DC has been designed and built and is now controlling the arithmetic operations of the computer.

To the writer's knowledge, the design of DC was the first application of the theory of speed-independent circuits to an operating portion of a large size digital computer. As will be described completely in Sec. 3, a small percentage of the circuitry for DC only meets the requirements of the mathematical model of a speed-independent circuit when it is examined in the light of a non-physical assumption. The fact that this design effort was very close to being successful has served as the motivation for these studies into methods of designing speed-independent logic. The characteristics of DC and the problems encountered in the design of DC will be discussed to the extent necessary to establish the motivation for the new design methods developed in later sections.

1.3 Statement of the Problem

The persons responsible for establishing the design specifications of the new Illinois computer felt that the characteristics of speed-independent circuits as described by Muller were desirable for use in the logic used to control the Arithmetic Units. Therefore, the design objective established for the logic design group was to create a speed-independent arithmetic control. Despite sincere efforts of the designers to achieve this, there are a few sections of the DC circuitry which require that an idealized assumption be made

---

* The abbreviation DC will be used for the Arithmetic Control and AC will be used for the Advanced Control of the new Illinois computer.

if the logic is to meet all of the requirements of the mathematical model of a speed-independent circuit. Using this fact as the initial point, the general purpose of these studies is to investigate the possibility of completing a control design in which all portions of the logic are speed-independent. The design of DC will be studied first to determine the primary problem areas and then this information will be used as a guide in the formulation of several new approaches to speed-independent logic design. These studies are not directed toward the creation of a new theory of asynchronous logic but rather toward an engineering evaluation of several methods of designing speed-independent logic.

Logic design, like many other synthesis procedures, seldom produces a unique configuration. Thus one could redesign a given logic using a different design method and obtain a completely new logical topology which would perform the intended function. Since the writer was privileged to be a member of the design group which created the logic for DC, many ideas, alternative design methods and new techniques have come to mind since it became apparent that all of DC would not quite operate in the desired manner. Two of these conjectures seemed worthy of careful consideration and form the basis of the new design methods developed in this dissertation.

The specific problem is this: given a new logic design problem in the form of a set of operations to be sequenced, can a new design method produce a truly speed-independent control? It was assumed, and subsequently verified, that a logic designed for this new problem, after the fashion of DC, would require the same assumptions to be considered speed-independent. In addition, each of these new design methods appears to require fewer transistors than the DC design method; thus it will also be worthwhile to study the transistor requirements of the various methods.

The logic of DC operates quite well and performs its intended function of sequencing operations in the Arithmetic Units. Therefore, one might be tempted to say that the mode of operation obtained is satisfactory and that the requirements of the mathematical model of a speed-independent circuit are unrealistic. It is the author's opinion that this is not true and the advantages of reliability and ease of maintenance mentioned earlier are desirable and are not fully realized unless the circuit is speed-independent. The basis for this opinion will be shown in the following short discussion of the mathematical model of a speed-independent circuit.

A mathematical model which takes account of every physical factor might be very true to nature; however, it would quite possibly be difficult to use due to its complicated structure. The model of a speed-independent circuit is based on three non-physical assumptions which deserve attention. These are best understood in the light of the following two definitions from Muller's first paper [MU-1]:

> Definition 1. Decision Element. A decision element has one output line f and a specified number k of input lines x_1, x_2, x_3, ..., x_k. It may be represented by a circular symbol.... At any given time a signal having a value of either 0 or 1 must appear on each of the (k + 1) lines.
>
> Definition 2. Asynchronous Circuit. An asynchronous circuit is an interconnection of decision elements. This interconnection is done by attaching lines together at points called nodes. Each node has at least one line connected to it and no more than one decision element output connected to it. No delay is assumed to take place along lines in the circuit.

There are two non-physical assumptions involved in Definition 1; first, it assumes that all variables are discrete whereas the voltages and currents used in electronics equipment to define a 0 or a 1 are continuous variables. This assumption, though non-physical, is reasonable since the characteristics of the circuitry utilized can be studied and specifications established for the current or voltage range for a 0 or a 1 signal. If then,

the level shift and/or level restoration of each circuit is carefully controlled, the continuous variables can be correctly interpreted as discrete variables. The second non-physical assumption included in Definition 1 is the statement that signals are either 0 or 1. This implies that a signal value is unique; that all elements of circuitry attached to a given node will interpret the signal value of that node in exactly the same manner. In most cases this is true; however, there are a few in which this happens only when special precautions are taken. Both Muller [MU-6] and Robertson [RO-2] discuss this problem and suggest means of overcoming these difficulties. Therefore, the second non-physical assumption is also a reasonable one.

The third non-physical assumption is the one stated in Definition 2; no delay is assumed to take place along lines of the circuit. In speed-independent theory an element may take any amount of time to react to an input combination which causes it to be excited. However, the time delay is assumed to be in the decision element and not along the output line. If the signal delay along a line is appreciable, in comparison to the operating times of the decision elements, then the circuit can be accurately represented by introducing into the logical expression for the circuit a delay element equivalent to the line delay. If then one includes the possibility of adding the delay elements in the expression for the circuit behavior, this third non-physical assumption is also a reasonable one to make. This places the responsibility upon the designer to be realistic in his evaluation of the circuit and also requires that he make his evaluation with some knowledge of the actual physical configuration of the logic. One can visualize many circuits in which the introduction of signal delay elements would cause the circuit to fail speed-independence. There are also as many, or more, situations where the introduction of these delay elements would not cause a circuit to fail speed-independence.

Thus the design problem is to select a logic configuration which is speed-independent with equivalent delay elements, if they are needed, and without them, if they are not needed.

In the remainder of this dissertation it will be assumed that the mathematical model of a speed-independent circuit is a reasonable one despite the fact that it is based upon several non-physical assumptions. All portions of this analysis will utilize the existing mathematical model to evaluate whether a circuit is speed-independent or not.

1.4 Method of Attack

As indicated by the title of this dissertation, it describes several separate, and yet interrelated, studies into the problems of designing speed-independent logic for a control. Four different methods of designing speed-independent logic are developed, discussed and evaluated. In order that these design methods might be realistically compared, four separate controls were designed, each one using a different design method. Each of the controls was designed using the same design objective, namely to sequence the operations of a fictitious arithmetic unit. Section 2 discusses the details of this arithmetic unit and the general rules adopted for use in the designs. The next four sections (3, 4, 5 and 6) are devoted to the descriptions of the several design methods and the associated control design. Section 7 contains the conclusions derived from these studies.

The purpose of these studies is to develop new methods of designing speed-independent logic for a control. It is therefore imperative that any new idea be carefully and completely tested for speed-independence prior to its presentation. Speed-independent operation is sometimes a difficult thing to prove or disprove. It is easily possible that a non-speed-independent

realization might be missed after hours of checking. This has been avoided by testing each new section of logic with the Illiac computer by using a Circuit Analysis program called Q 5 [DC-3]. This program allows each section of a logic to be coded and tested. In almost all cases, the new section of logic was placed in a loop closed upon itself and the circuit tested until the computer ascertained that the logic was cycling through a fixed sequence of states.

2. SPINDAC, THE BASIS FOR COMPARISON

Later sections of this dissertation will develop four separate methods for designing speed-independent logic for a control. The primary evaluation or comparison of these methods is whether or not it is possible to design a speed-independent control using them. However, in an effort to form a reasonably realistic basis for a comparison of these design methods, it was decided that four controls would be designed--one for each of the methods. For reference purposes these controls will be called "Spindac" for speed-independent arithmetic control. Each Spindac will be further designated with a code letter which will identify the design method, e.g., Spindac P, Spindac R, etc.

2.1 Flow Chart

Each of the Spindacs was designed to perform exactly the same functions. The basis for these designs was a flow chart which described the time ordering of the various operations required by the fictitious computer. Except for the special symbols described in the last paragraph of this section, the flow chart was of the type described by Gillies [GI-1]. This flow chart is shown in Fig. 1 and the order code associated with it is given in Table I. This relatively simple floating-point machine can perform thirteen arithmetic instructions including addition, multiplication, exponent arithmetic, and four types of store orders. Although this flow chart is small in comparison to the flow charts or flow diagrams of contemporary computers, it is believed that it contains almost all of the difficult logic design problems.

Most of the designations on the flow chart are self-evident or are explained in Gillies' paper; however, the letter-number designations, such as A 3, are reference numbers used to correlate the flow chart operations and the logic. These designations, called control points, will be more fully explained

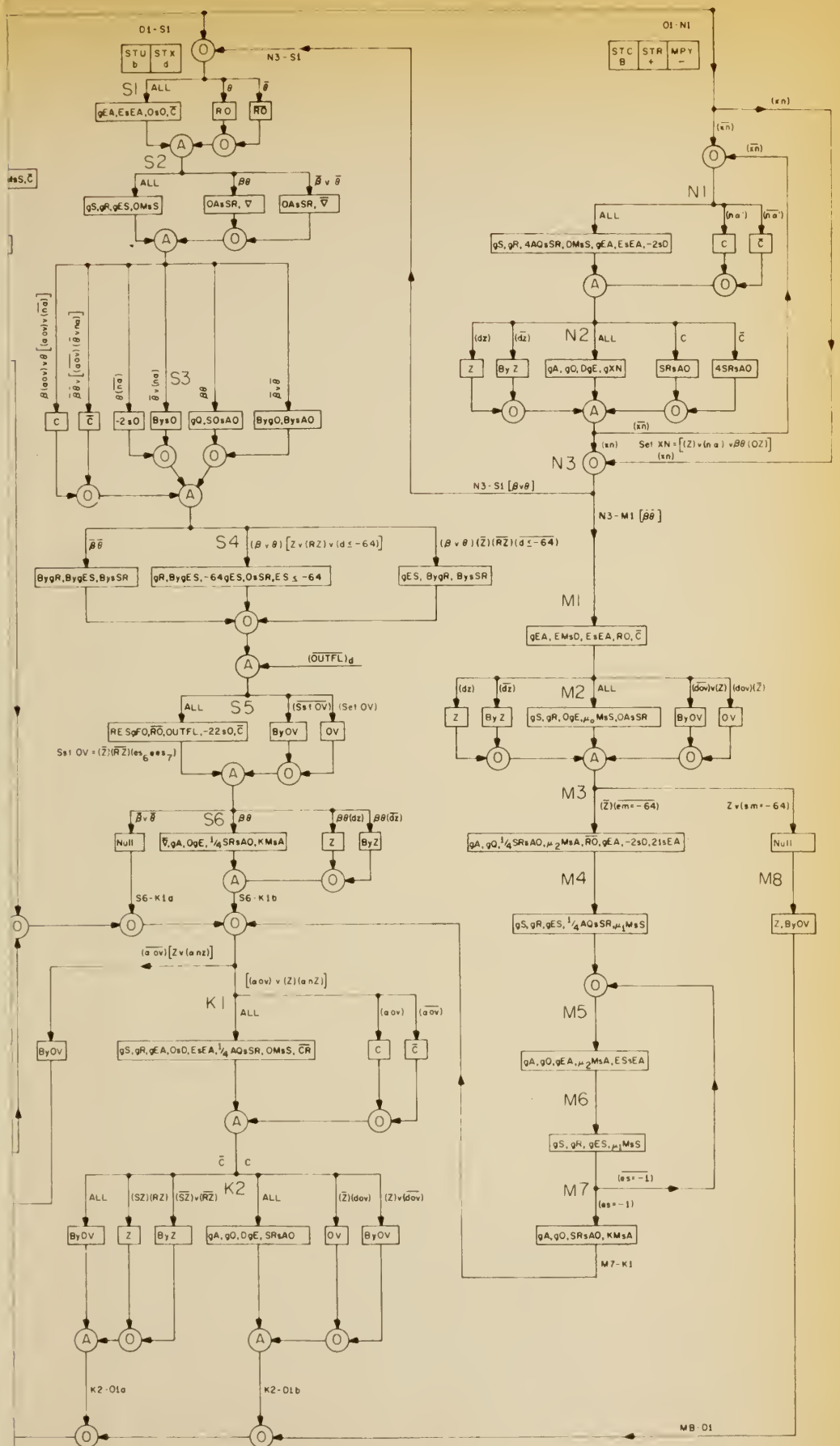


FIGURE 1 - COMPLETE FLOW CHART for SPINDAC

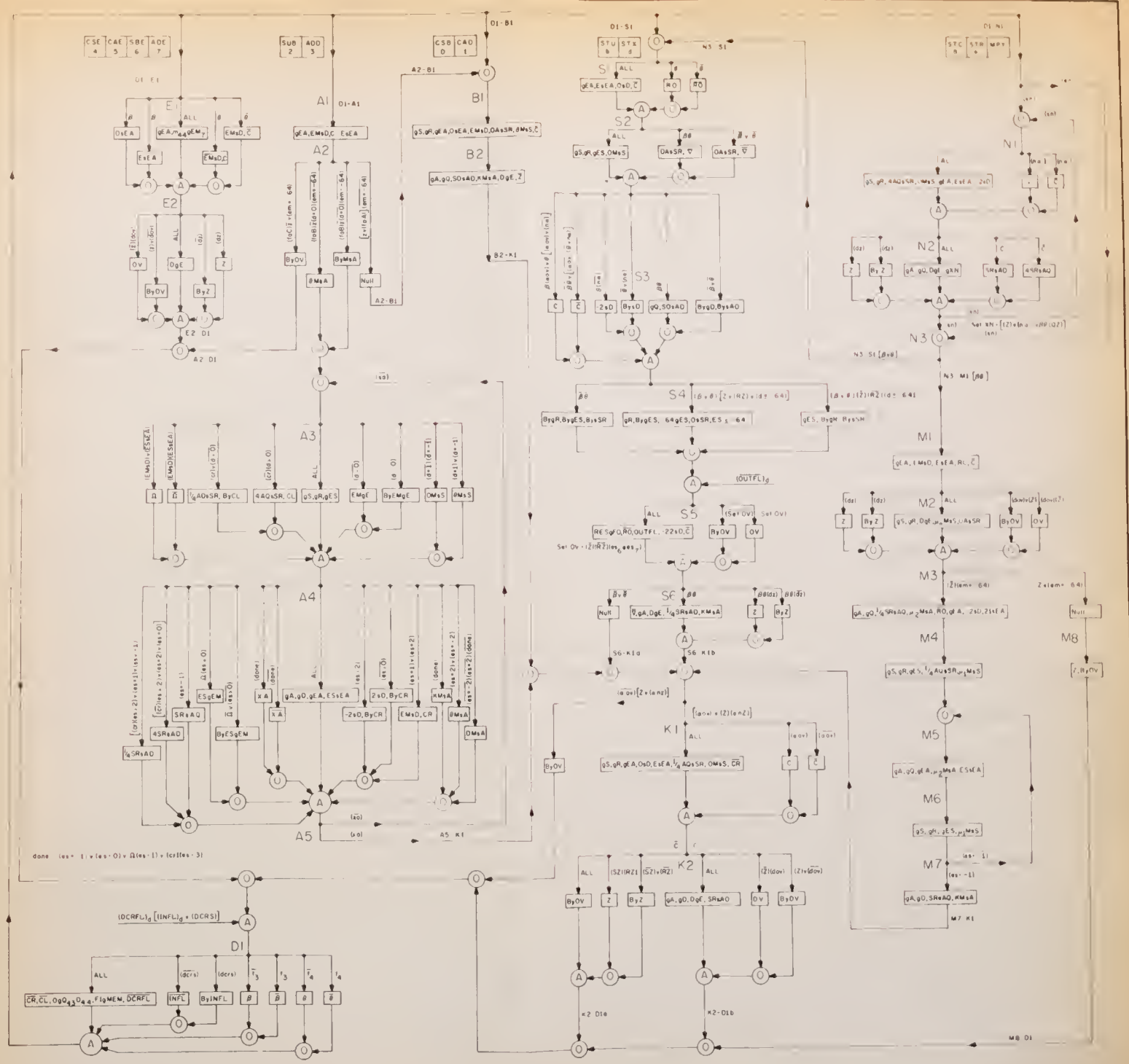


FIGURE 1 - COMPLETE FLOW CHART for SPINDAC

TABLE I Order Code for the Spindacs

Mnemonic	Order	Initial Sequencing Control Point	Order Register Bits $f_1 f_2 f_3 f_4$	Sexadecimal Value
CSB	Clear Subtract	B1	0 0 0 0	0
CAD	Clear Add	B1	0 0 0 1	1
SUB	Hold Subtract	A1	0 0 1 0	2
ADD	Hold Add	A1	0 0 1 1	3
CSE	Clear Subtract Exponent	E1	0 1 0 0	4
CAE	Clear Add Exponent	E1	0 1 0 1	5
SBE	Hold Subtract Exponent	E1	0 1 1 0	6
ADE	Hold Add Exponent	E1	0 1 1 1	7
STC	Store Clear	N1	1 0 0 0	8
-	-	--	1 0 0 1	
STR	Store	N1	1 0 1 0	+
MPY	Multiply	N1	1 0 1 1	-
-	-	--	1 1 0 0	
STU	Store Unnormalized	S1	1 1 0 1	b
-	-	--	1 1 1 0	
STX	Store Exponent	S1	1 1 1 1	d

in Sec. 3.1.2. All of the operations associated with A's are part of the floating-point addition subsequence, the B's are for clear addition, D 1 is the instruction decoder, the E's are for the base four exponent arithmetic sequence, the K's form a subsequence necessary to correct overflow and detect zero, the M's are for multiplication, the N's form the normalization subsequence and the S's are used to store results.

The iterative shifting operations necessary to floating-point addition are done in what is called the add loop; A 3, A 4 and A 5, whereas the multiplication loop is formed by M 5, M 6 and M 7.

2.2 General Design Philosophy

The logic design for each of the Spindacs will contain only the essential elements necessary to perform the sequencing operations required by the flow chart. These will not be final logic designs in which all of the cable drivers, level restoring circuits, etc., necessary for a specific physical layout are included. The use of this philsophy has several disadvantages; however, it was used because it has the very important advantage that the results of these studies will be generally applicable rather than being limited to one type of physical configuration.

There are two disadvantages to this design philosophy. The first is that the modifications of the logic required to refine an essential logic into a final design will not be included in the evaluations of this study. Although this is unfortunate it is felt to be justifiable since all four of the designs will be subject to the same types of revisions when a physical configuration is chosen. In the event that such refinements might seriously affect the ability of a design to be considered speed-independent, these will be discussed in connection with each design method. The second disadvantage

of this design philosophy is that these refinements will affect the transistor count of each control design. Again this is considered justifiable due to the fact that very similar adjustments in the transistor count will be made for each control design. The objective is not to determine the precise number of transistors required by each of the design methods but to compare the numbers required by different methods. If then the additional logic required to convert an essential logic into a final design is directly proportional to the size of the essential logic, the comparison of sizes will be the same whether the essential logics or the final logics are used. Although this is not precisely true, it is near enough to the truth to provide the accuracy of comparison desired.

3. TYPE P LOGIC, SPINDAC P AND DC

This section of the dissertation will be concerned with the first of the four methods for designing speed independent logic for a control. This design method is the one used for the design of DC and has been given the designation of type P logic (P is for the Present control). Similarly, Spindac P is the control designed for this dissertation which utilizes type P logic. It seems appropriate to begin this section with a short description of the interrelationship of type P logic, Spindac P and DC. The abbreviation DC is used to designate the Arithmetic Control for the new Illinois computer. It is an operating piece of electronics equipment which was designed and built in the Digital Computer Laboratory of the University of Illinois and is presently controlling the Arithmetic units of the new Illinois computer. Spindac P, on the other hand, is a paper control design for the fictitious machine described by the flow chart of Fig. 1. Spindac P is a miniature equivalent of DC and was designed for the sole purpose of making a comparison with the other Spindacs. Spindac P and DC are similar in that their respective logics were created using the same design method, and thus for the purposes of these discussions they both contain type P logic. Therefore, the following discussion and evaluation of the logic design method for DC is an evaluation of type P logic which is also applicable to Spindac P.

The task of creating the logic design for DC was accomplished through the combined efforts of some dozen people. Although the writer was a member of this group and made original contributions to this effort, he does not claim that all of the design of DC was his own work. Whenever possible, credit has been given for original contributions by other members of the group. It is hoped that none have been inadvertantly overlooked. Despite the fact that all of the work involved in creating type P logic is not the author's original

work, it is important that this form of logic design be discussed in some detail. The primary reason for this statement is that the evaluations of DC and type P logic form the motivation for the three other design methods. A second, and nearly as important, reason is that many of the techniques for sequencing operations in type P logic are used by the other design methods as well. The evaluations of the design of DC in Sec. 3.2 are the author's own work.

3.1 The Design Method

3.1.1 Historical Background

As has been previously mentioned, Muller and Bartky published several papers in which they introduced and developed the concept of speed-independent operation of asynchronous logic. These papers presented the mathematical development necessary to define accurately the various subclasses with the class of speed-independent circuits. They also outlined a synthesis method which was subsequently formalized by Shelly [SH-2, SH-3].

When the logic design for DC began, the following three methods were available for designing speed-independent circuits:

a. The method of combining blocks

b. The method of simulating synchronous circuits

c. The method of design from change charts

These three methods were described in Report 80 [DC-1]. The change chart method is the one developed by Shelly and is the only direct synthesis method. It therefore seemed most advantageous. Although Shelly's method was attractive, it was based upon a memory element known as the "C" element. This memory element had the property that whenever the two inputs were in the same state the output duplicated that state but otherwise the output retained its previous

value. Logically this was a useful device, but its circuitry was not at all advantageous; and by the time the detailed logic design for DC was beginning, the "C" element had been replaced by memory elements of the Eccles-Jordan type [DC-2]. A new direct synthesis procedure was not developed for these memory elements but rather the control was designed using the method of combining blocks of speed-independent logic. This type of a design method was shown to be feasible in Muller's initial paper [MU-1]. In this paper he showed that two speed-independent circuits could be interconnected so that the resulting circuit was also speed-independent. Therefore, in lieu of a direct synthesis procedure, a set of speed-independent Basic Logic Diagrams (abbreviated BLD) was developed, and the control formed by interconnecting these in a speed-independent manner.

The type P logic design method and simplified versions of some of the commonly used BLD's were discussed in an earlier paper by the writer [SW-1]. This paper was tutorial in nature and was an introduction to speed-independent logic circuits. It also sought to describe the movement of signals through the logic. In this paper several of the BLD's were not mentioned at all and others considerably simplified. To make the discussion and evaluation of type P logic complete, all of the BLD's will be presented and discussed despite the partial duplication.

A second reason for discussing the BLD's is the fact that many of the techniques for sequencing operations are used by the other design methods as well as type P logic. The most important similarity betweenthe BLD's of type P logic and the other three design methods is the use of the NAND circuit to initiate and terminate the action of devices which are to be energized at one time. Type R logic, for example, will use these same BLD's with only slight modifications to the method of handling replies. In type D logic and

type C logic the control signals to the NAND circuit are different from those shown in the BLD's; however, the output signal from the NAND circuit is always used to initiate and terminate operations.

3.1.2 Basic Logic Diagrams

Each of the BLD's has its own special purpose for existence and as a result it has characteristics peculiar to itself. There are also many features which are common to all of the diagrams. In many cases the comments made about individual BLD's will have more meaning if the following general characteristics are discussed first.

a. For convenience sake, the sequencing portions of control have been divided into areas called control points (abbreviated CP). A CP includes the NAND and such other logic as is necessary to perform the conditional and sequencing operations. The CP numbers appearing on the BLD logic drawings and flow charts are there for reference purposes and the dashed lines are used to separate the various CP's.

b. The logical symbolism used is as follows: A is for an AND, $\overline{A}$ is for a NAND, O is for an OR, N is a NOT circuit, Δ is for a delay, CD is a noninverting cable driver, $\overline{\overline{CD}}$ is an inverting cable driver and the symbol for a memory element is described in connection with BLD 6. A circular symbol with only one circle implies a nonrestoring circuit whereas a symbol with two concentric circles implies a restoring circuit. Some of the restoring circuits also have an asterisk after

the logical description. These circuits have shifted thresholds which are necessary to maintain speed-independent operation at a point where a signal drives two restoring inputs. The need for such modifications to the circuits was discussed in Sec. 1.3 and the method of solution used in DC is discussed by Robertson [RO-1]. The two symbols  and  are used to represent the logical equivalent of the gate-driver mechanism and the selector-driver mechanism. Each of these symbols represents a great deal of logic, but from the standpoint of the sequencing control they appear as a delay with or without inversion.

c. As a signal moves from one CP to the next, the action of sequencing has two substeps. In the first substep the gate is turned on and a reply signifying this fact is detected in the sequencing control; in the second substep there are control operations culminating in turning the gate off. When sequencing control detects that the gate is off, it permits the first substep at the following CP to begin. The first substep ends when the reply is received signifying that the gate is on and the second substep ends when a reply is received indicating that the gate is off. These same two substeps are associated with every control step regardless of whether the operations performed are those of gating, changing selectors, or setting memory elements. The reply signals from complete devices are all the same;

the reply signal will eventually duplicate the request signal. Because of this fact, all of the BLD's, while given in terms of gates, are examples of sequencing action using any type of device.

d. In all cases where a conditional action is to be taken, it is assumed that some previous control step has verified that the conditional signals are indeed reliable. The cases where this assumption is not completely valid will be discussed in Sec. 3.2.

e. All of the BLD's which involve conditional operations perform these conditional operations using complementary conditional signals. In almost all cases it would be possible to perform these same conditional operations in a speed-independent manner without having complementary conditional signals.

f. A change chart is a method developed by Muller [MU-5] for describing the number of times a given node in a circuit has changed state. The flow charts used to describe control operations are a simplication of change charts [MI-1, GI-1, GI-2]. In using these flow charts it has been assumed that the two substep operations mentioned above are cross-coupled in the change chart sense, for each operation requested. That is, when two gates are requested by the same CP, the control action will not proceed until both gate replies have signified that the gates are open and similarly with turn off. This type of operation is

inherent in all of the BLD's given; however, this does not imply that a realization performing the operations in a different way would not be speed-independent.

BLD 1. Two Forms of Parallel Operation

BLD 1 (Fig. 2)* shows two different speed-independent methods for performing two gating operations at the same time. Historically the method shown for CP's 11 and 12 is the original, but the logic shown for CP's 12 and 13 is preferred since it operates faster. In the latter method the time required for the memory element at CP 13 to turn over is paralleled with the time required to turn the gate on and for the reply to arrive at CP 13. The time advantage of this method was first mentioned by Richard R. Shively of the University of Illinois' Digital Computer Laboratory.

BLD 2. Conditional Operation and Bypassing

One of the most commonly used means of simplifying control logic is to create conditional request signals. The sequencing NAND output opens the conditional OR's and the conditional signal determines the specific operation to take place. A reference to the flow chart, Fig. 1, will reveal that seventy conditional requests are used in each Spindac. One form of conditional request is the Bypass signal as shown on the BLD 2 drawing. This signal causes no action within the Arithmetic Units but creates a fictitious reply signal which tricks the sequencing control into reacting as though a machine operation had taken place. Any gate, selector, or memory element can be bypassed through the use of an AND circuit as shown on BLD's 2, 6 and 7.

---

* Each drawing of a BLD has both a Figure number and a BLD number assigned to it. For simplicity sake only the BLD number will be used in referencing.

An appreciation of the usefulness of the conditional request can be obtained through a study of the ADD loop, CP's A 3, A 4 and A 5 on the flow chart. All of the needed pre-add shifts, the actual addition, and the post-add shifts are combined into these CP's through the use of conditional requests and bypass requests. The use of conditional requests also permits the designer to combine operations when the reduction of sequencing logic is greater than the increase in conditional logic.

Regardless of the type of device bypassed, the logic may have only one bypassing AND circuit if the control is to operate in a speed-independent manner. When a device receives all of its requests and bypass signals from a single CP, the bypassing AND can be placed wherever convenient. However, if requests for the device's operation or bypass requests come from more than one CP, the bypassing AND must be located with the device logic. The output of this AND then becomes the reply signal for the device.

The logic of this BLD can be considered speed-independent only if the conditional signals are stable during the time that the NAND output is a "0". The problems associated with this requirement are discussed in Sec. 3.2.1.2.

BLD 3a. Multiple Use of a Device

BLD 3b. Request Fan-in and Reply Fan-out

Even a casual glance at the flow chart will reveal that each of the gates within the Arithmetic Units and most of the control devices will be actuated from many different CP's. In practice, a collapsing tree of AND circuits as shown in BLD 3b is used to collect all of the requests. Fortunately this collapsing tree of request logic is speed-independent. Any or all of the request lines from the CP's to the initial AND circuits can pass through

conditional request logic of the type shown in BLD 2 or conditional branching logic of the type to be discussed in connection with BLD 5. The bypass requests also form a small collapsing tree of speed-independent logic.

The expanding tree of reply logic is not speed-independent. The problems associated with this tree of reply logic and some of the reasons for this situation arising are discussed in Sec. 3.2.1.1. The delay element shown in the reply line of BLD 3a represents the expanding tree of reply logic shown in BLD 3b, and will be discussed in Sec. 3.2.1.1.

BLD 4. Conditional Bypassing of a Step

In many respects BLD 2 and BLD 4 perform the same operation. In both cases a function, or functions, are performed or not performed as a condition is met or not met. When there is a combination of unconditional operations and conditional operations at a CP, BLD 2 must be used. If all the operations at a CP are conditional on the same signal, then either BLD 2 or 4 could be used, but in almost all cases BLD 4 has been used since it operates faster. It can be seen from the BLD 4 drawing that if the step is to be bypassed, the functions of the next step are immediately activated, whereas if BLD 2 were used, all the devices would have to be bypassed and the control would wait for reply signals before proceeding. The similarity which exists between BLD 4 and 5 will be discussed in the next section.

BLD 5. Conditional Branching Out and Recombination

BLD 5 shows a speed-independent method to make a conditional branch and then subsequently recombine. It is not necessary to recombine after only one step as shown, but the method of recombination is the same. The two conditional OR's at the output of the NAND of CP 51 and the AND driving the

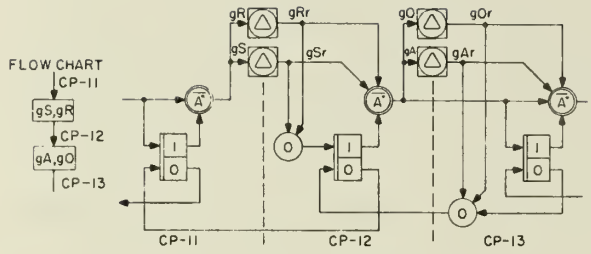


Figure 2 - BLD 1 - Two Forms Of Parallel Operation

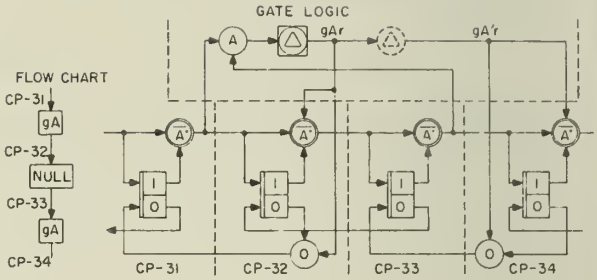


Figure 4 - BLD 3a - Multiple Use Of A Device

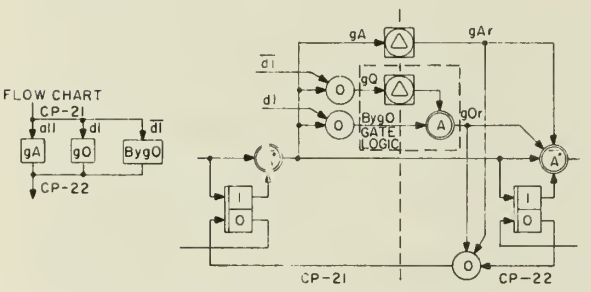


Figure 3 - BLD 2 - Conditional Operation And Bypassing

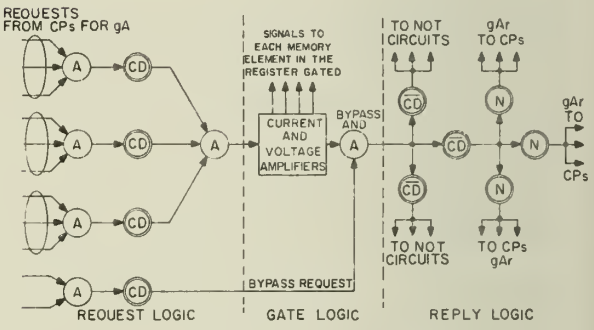


Figure 5 - BLD 3b - Request Fanin And Reply Fanout

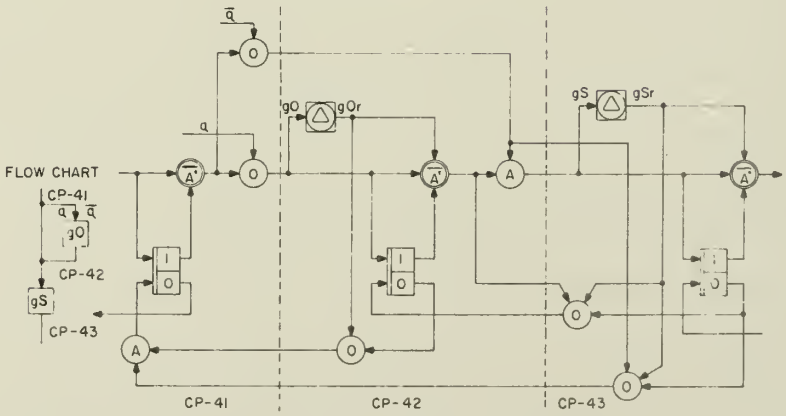


Figure 6 - BLD 4 - Conditional Bypassing Of A Step

"0" input to the memory element are the logic elements necessary to branch out. To recombine one needs the AND in CP 53 to combine inputs and then separate OR's are used to turn off the appropriate previous memory element.

In BLD 5 the control performed an operation as it was branching to the next CP. That is, when α is true the gate gE is actuated prior to the actions of CP 53. If it should happen that after a branch there are no operations to be performed until the next CP is reached, then logic of the form of BLD 4 can be used. On the flow chart control point N 3 is an example of a branch without operations to be performed and CP K 2 is an example of a branch with functions to be done.

In all of these discussions and those pertaining to BLD 4, it has been assumed that every branching operation will be done conditionally. This is just the way it happened to turn out as far as realizing the specified sequencing for DC and Spindac P. Speed-independent logical realizations of an unconditional branch and recombination are possible and have been drawn; however, they are not included here since neither DC nor Spindac P uses such logic.

BLD 6. Setting of Memory Elements

BLD 6 presents both the methods used to set memory elements and a logical equivalent of the memory element with reply signal. The logic of the memory element will be discussed first. The quiescent state for the element is with both inputs "1" and the outputs (1-0) or (0-1). Action will be initiated whenever a request (in input "0") is received. If the element is in the state requested by the input "0" the reply will come quickly. If the memory element must change state, a step-by-step check of the signal flow through the circuit will reveal that the reply is the last signal to be

generated. Thus when a "0" reply signal is received, the memory element is in the state requested. During the second substep of a control step, the request signal is returned to "1" which causes the reply to change to "1".

The primary advantage of this memory element is that it has no disallowed input states. The normal inputs are (1-1) when the element is not being changed and (1-0) or (0-1) when a change is occurring. The control is so designed that the entire tree of request logic for both inputs to the memory element and for the bypass input will be in a "1" state unless a CP is in communication with the element. Therefore, the "hang-up" state of a (0-0) input will occur only when there is a malfunction that causes an unwanted request and some CP requests the other input. Since this input state creates a "1" reply, the control will "hang-up" waiting for a reply "0" as mentioned above. Although there are other possible ways for the control to "hang-up" due to a malfunction on a request line, many faults have been located due to this operation of either a selector or a memory element.

Note that in the case of the CB memory element the state of the memory element is determined by the decoder. The gating OR's can be placed at either the input to or the output from the decoder but the speed-independent logic includes only the OR's and subsequent logic. Type P logic utilizes both forms of gating. This division between speed-independent and non-speed-independent logic is due to the fact that the control was designed to be speed-independent, whereas the Arithmetic Units controlled were not. The designer is faced with the decision: where does control end and the Arithmetic Unit begin? The logic to decode the next multiplier digit has the gating OR's ahead of the decoder, but the decoders used to set the exit memory elements used in the Add and Normalize loops (see BLD 8) are ahead of the gating OR's.

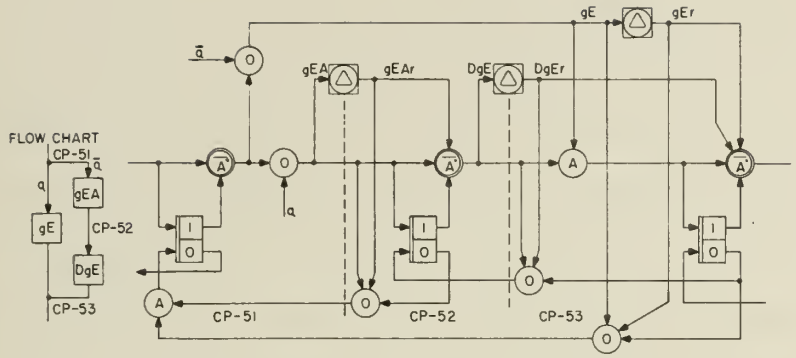
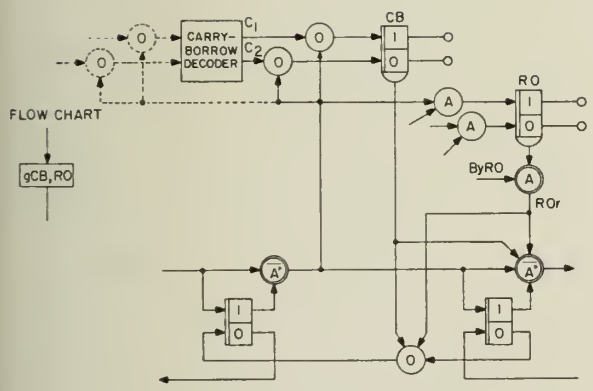
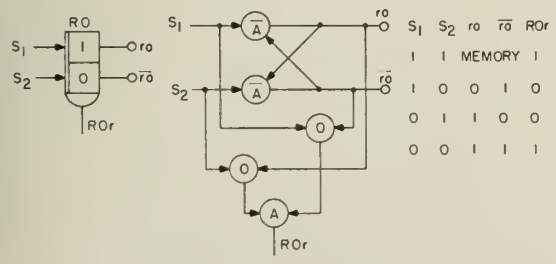


Figure 7-BLD 5-Conditionol Bronching Out And Recombinotion



LOGICAL DIAGRAM



LOGICAL EQUIVALENT OF MEMORY ELEMENT WITH REPLY

Figure 8-BLD 6-Setting Of Memory Elements

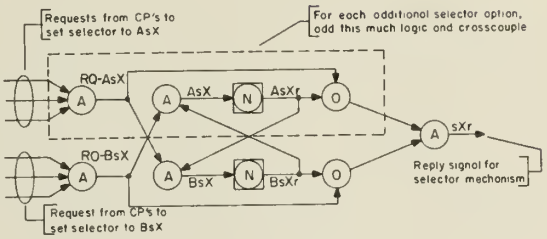


Figure 9-BLD 7a-Simplified Selector Control

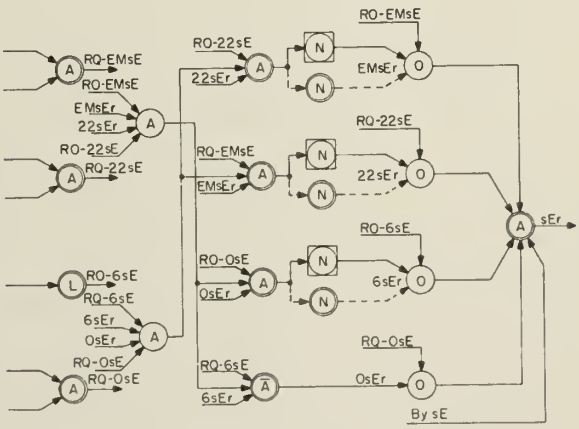


Figure 10-BLD 7b-Typical Selector Control (sE)

BLD 7a. Simplified Selector Control

BLD 7b. Typical Selector Control (sE)

If one compares the logical equivalent of a memory element shown on BLD 6 with BLD 7a it is obvious that the selector control is nothing more than the cross-coupled NAND type of memory element in which the NOT function has been accomplished in the individual selector-driver mechanism. If one also checks the step-by-step change in states as one of the memory elements goes from one output state to the other, it will reveal that at no time are both outputs in the "0" state. The selector control is so designed that when an output is "0" that particular selector is "on"; therefore, only one selector option is ever energized. Since this selector control is basically a memory element, whichever selector option is requested is turned on and left on until the next request is received. A subsequent request turns off the previous setting prior to turning the new one on.

The configuration of the registers in the Arithmetic Units is such that the selector mechanism and the gate mechanism leading into a register are in cascade. Thus it is imperative that the selector output be stable during the time that the gate is on. In particular, the selector output must remain unchanged until the gate has indeed turned off. It is for this reason that the selector control was designed with memory. A typical operation is to request the desired selector setting and the gating operation into a register from the same CP. Then during the first substep of that particular control operation, the newly-requested selector setting is set by the time a "0" reply is received from the selector control and a "0" reply from the gate logic signifies that the gate is on. The control has therefore verified that the information path into the register is stable and the gate can be turned off and the selector input returned to its quiescent state.

BLD 7b shows the actual logic used to control the selector into the E register in the Exponent Arithmetic Unit. This shows how the simplified control of BLD 7a is developed to perform a needed operation. The two four-input nonrestoring AND circuits shown in the second column from the left are used merely to reduce the number of inputs into the AND's in the third column. Since every request and reply should be cross-coupled, these third column AND's should have six inputs, whereas through the use of these combining AND's the number of inputs is cut to three. The inputs to these combining AND's must be in pairs of request and reply as shown, and there is no restriction as to the number of options which can be combined in one of these circuits provided one is careful to use the output only where all of its inputs are required. For example, neither of the combining AND's shown could be used in any more of the tnird-column AND's without introducing signals not required.

As has been mentioned earlier, the Arithmetic Units to be controlled were not designed as speed-independent logic. This is one more place where a division must be made between the logic of control and the logic of the Arithmetic Unit. The selector reply signal is merely a sample of the selector-driver output, and since it is a sample one must decide from where the sample is to be taken. The closer one approaches to the register with the sampling point the more accurate the sample will be, but also more time will be taken before a reply is received and thus the control will operate slower. The opposite extreme is shown on BLD 7b where the sample is taken from the output of the third-column AND's and a standard NOT circuit is used to obtain the proper signal parity for the reply. This system will operate quite fast but it suffers from the disadvantage that very little information about the condition of the selector signals which are sent to the Arithmetic Unit is contained in such reply signals.

BLD 8a. Scale of Two Loop with Entry and Exit Conditions

BLD 8b. Slow-Fast Shifting in a Loop

Almost all of the repeated shifting operations necessary in executing an instruction such as multiply, add, or normalize are done in a scale of two loop of the type shown in BLD 8a. This control repeatedly does two sets of operations until an exit condition is met and the control then passes to the next subsequence. Each of these sets will probably contain some conditional operations; for example, the Add loop has many conditional operations.

CP 81 is the entry point and CP 83 is the exit point. Note that the NAND's of CP 81 and 83 have signals from two memory elements. A signal cannot enter or leave unless CP 81 is active and unless the memory element in CP 82 is in the "0" state. This is, incidentally, the quiescent state for this memory element and an exit leaves this element ready for the next entry. Note that in this type of loop the OR's must be ahead of the memory element since during the loop operation there is only one memory element changing state.

Although the logic for the Exit memory element (EX) might appear to be complicated, the method of handling the requests and replies is identical to the method explained in BLD 6. The unique part is the fact that the outputs are used to determine whether the NAND at CP 81 is active (which means stay in the loop one more time), or the NAND at CP 83 is active (which means leave the loop). This logic illustrates how the entry can also be an exit. Note that the input signal to CP 81 also gates the Exit Decoder and thus it might happen that although the control enters the loop, it also leave it immediately. By this means one can unconditionally send a control signal to the normalize subsequence and if the number is indeed already normalized, the Exit Decoder will detect this fact and control will then immediately leave the Normalize Loop without affecting the number.

Several variations of these entry and exit operations exist, but all use the same basic loop and sequencing methods. For example, in ADD the control always passes through the loop at least once and thus the entry signal does not gate the exit decoder but instead sets $\overline{\overline{EX}}$. For multiply the exit signal comes from a memory element without reply (signals direct from the Exponent Arithmetic Unit). There is no exit memory element and the input signal is taken also to CP 83 (shown dotted) to hold CP 83 off until the previous CP's operation has been completed. Also note that after control has left the loop, the signal coming back to turn the loop off returns all the way to the memory element at CP 81. In several of the exit decoders the signals from the Arithmetic Units that are used may be changing due to the gating operations being performed. To allow sufficient time for these signals to become stable the signal used to gate the exit decoder is not taken from the NAND as shown in BLD 8a but from an OR of the appropriate reply signals.

The use of the BLD 8a logic was first suggested by John O. Penhollow of the University of Illinois' Digital Computer Laboratory.

Whenever a selector control or a memory element receives a request to set the device to its present state, a reply signal is quickly sent back to sequencing control. A check of BLD 6 or 7b will show that a reply to a bypass request is obtained in even less time. The logic of BLD 8b is used to capitalize on this fact and cause long shifting operations within the floating addition subsequence to be speeded up. For example, it is possible that an add instruction might require fifteen or twenty left shifts prior to the addition and then the same number of right shifts. If BLD 8b were a part of DC, the condition (α), and thus the selector setting, would be set at the start of the shifts and would remain stationary during the shifts. The condition (s) would be so defined as to cause a bypass request to be generated whenever the

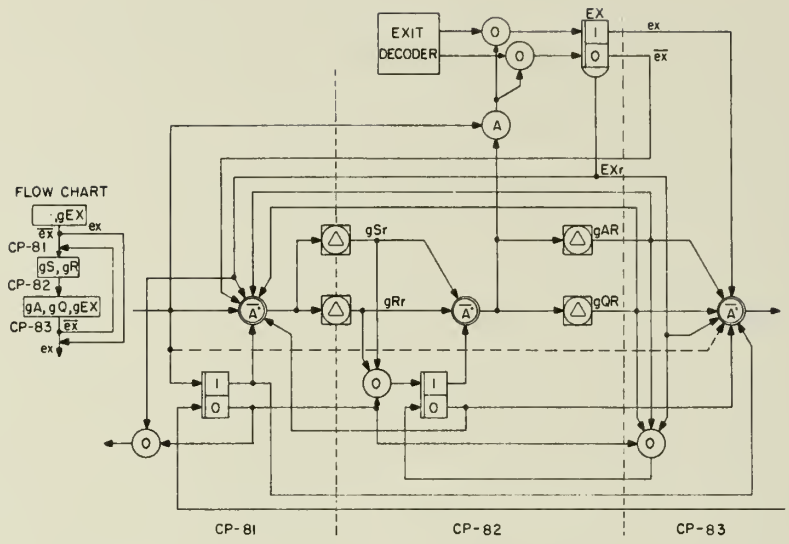


Figure 11-BLD 8a-Scole Of Two Loop With Entry And Exit Conditions

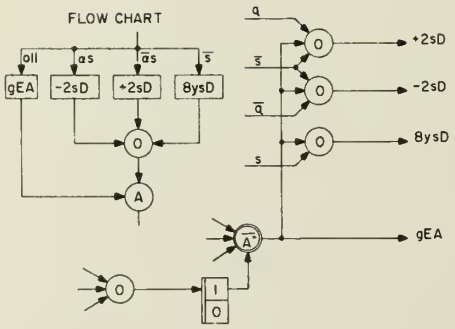


Figure 12-BLD 8b-Slow-Fost Shifting In A Loop

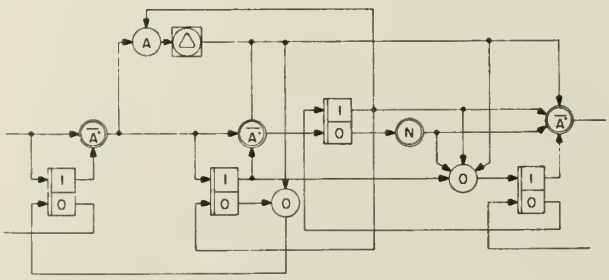


Figure 13-BLD 9-Using A Devic Twice In A Row

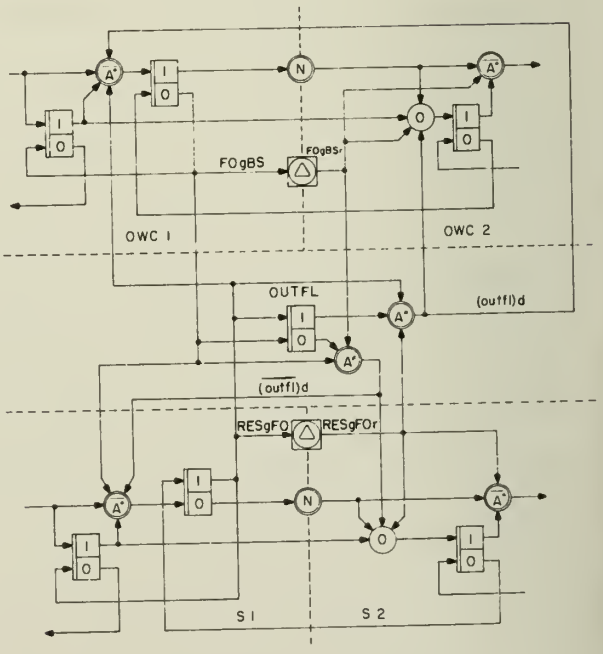


Figure 14-BLD 10-Interconnection Of Two Otherwise Autonomous Controls

conditional logic would request the same selector setting. The subject of generating the conditional signals such as (α) and (s) is discussed by Gillies [GI-2] in a report concerning the DC flow chart.

BLD 9. Using a Device Twice in a Row

In the normal sequencing method shown in BLD 1 it is not possible to perform the same operation in two successive CP's and do it in a speed-independent manner. The first request would be executed properly; however, the "0" reply caused by the second request would cause the second NAND to turn off prematurely. The particular device would operate properly but the sequencing logic and any parallel operations might malfunction. The two operations can be done properly if they are separated by a null step or other step.

The logic of BLD 9 is sometimes referred to as a fast null step because it isolates the second request from the first and thus allows the same operation to be performed in two successive steps. Note that if the two CP's request different settings of a selector control, this must be considered as using a device twice because the same reply signal is involved. This form of fast null step was suggested by John O. Penhollow and is used, with a few modifications, in BLD 10.

BLD 10. Interconnection of Two Otherwise Autonomous Controls

The new Illinois computer will contain several controls. DC is concerned with sequencing of operations in the Arithmetic Units, AC is concerned with "look ahead" and address arithmetic and the Out Write Control (abbreviated OWC) is concerned with storing of arithmetic results in the memory. When a word, or quarter-word, is to be passed from one of these controls to the other, it is necessary to verify that operations are done in the

proper order. The example given in BLD 10 is for the case where an arithmetic result is to be stored in memory. DC will place the word in a buffer register called the OUT register and OWC will subsequently place the word in its proper storage location. The need for an interlock is obvious since DC must be inhibited from loading OUT until OWC has stored the previous word placed in OUT and similarly, OWC must be prevented from storing the contents of OUT until DC has indeed placed the new word into the register.

In BLD 10 the two CP's S 1 and S 2 are a part of DC and CP's OWC 1 and OWC 2 are parts of OWC. DC turns OUTFL (OUT is full) memory element to the "1" state when it loads the OUT register, but it cannot begin the step until $(\overline{\text{outfl}})_d$ is true. It can be seen from the drawing that the symbol $(\overline{\text{outfl}})_d$ means that OUT is not full and that the CP in OWC which empties OUT is through with its job. Similarly when S 1 has loaded OUT and is truly finished then $(\text{outfl})_d$ is true and OWC 1 can energize the gate FOgBS to store the contents of OUT. The two controls do not energize the same gate but rather a gate into and a gate out of the same register.

3.2 A Brief Evaluation of the Design of DC

Earlier in this dissertation some parts of the logic of DC were described as requiring special consideration if they were to meet the requirements of a speed-independent circuit. Although this is a true statement, it is hardly precise. One is tempted to contrive a special terminology such as pseudo-speed-independent or neo-speed-independent; however, these terms are also not precise and will not be used for fear of being misunderstood. The portions of circuitry which require special consideration are contained within the signal distribution systems. Since only 12 per cent of the DC transistors are used in the entire distribution system, and since portions of the system

are speed-independent without special assumptions, the desire for an "almost" speed-independent designation is understandable. Nevertheless, DC contains logic which only operates in a speed-independent manner if an unrealistic non-physical assumption is made; therefore, DC must be considered as non-speed-independent.

3.2.1 Signal Distribution Systems

The primary cause for the design of DC to be non-speed-independent was the method of signal distribution used throughout the control. This has often been referred to as the reply problem since the majority of occasions for high fan-out signal distribution occurs in the reply system. It must be emphasized though, that any signal distribution system in which the required fan-out was obtained through the use of parallel current and/or voltage amplifiers would probably not be a speed-independent circuit.

3.2.1.1 Reply Signals

If the reply signal from a device is used at only one CP as shown on BLD 6, then that particular reply logic can be considered to be a speed-independent circuit. When the reply signal is used at many CP's and the fan-out is obtained through the use of an expanding tree of logic as shown on BLD 3b, then the equivalent delay shown on BLD 3a is present and that signal is not handled in a speed-independent manner. Most of the gate and selector reply signals and many of the memory element replies are distributed in the latter manner. This is the reason for the comment that the reply system is a major cause for DC being non-speed-independent.

The only purpose served by the logic elements shown in the Reply Logic section of BLD 3b is to duplicate the reply signal many times so that

the required fan-out could be attained. As a result, these decision elements could be considered as identity elements which perform no logic function and might be disregarded in the formulation of the logic expression for the circuit. If these elements are disregarded, then all of the reply logic would operate in a speed-independent manner. However, the elimination of these elements from the expression is equivalent to making the non-physical assumption that every one of the reply output signals changes state at the same time. Which is to say, all of the identity elements in the reply logic are ideal elements with zero operation time. This is the non-physical assumption referred to earlier which is necessary if all of the DC logic is to be considered as speed-independent. The mathematical model of a speed-independent circuit is based on three non-physical assumptions which have been shown to be realistic. This non-physical assumption is not realistic and therefore the logic of DC must be referred to as non-speed-independent.

It is the writer's opinion that the reply logic evolved into this form as the result of two separate factors. The first factor was the two-valued or non-unique signal value problem at the input to the CP which is discussed in Sec. 1.3. Through the use of a shifted threshold it was possible to maintain speed-independent operation at this point. The reason this fact was important in the reply circuitry was that all reply signals were then required to have a very large signal voltage swing (-1.22 v to +2.50 v). This is almost impossible to obtain without using a restoring circuit such as the NOT at the point where the reply is used.

The second factor in the reply problem was the circuit designers' decision to design cable drivers using the same transistor as was used for all other circuits. This placed a tight restriction on the cable driver design because of the limited current capabilities and power rating of this transistor.

Several of the devices in DC have a reply fan-out requirement of over fifty. With cable drivers of moderate-current driving capabilities and with the need to have voltage swing and fan-out at the CP's, that is, fan-out at a distance from the reply logic, the only solution was to divide up the load into small pieces as is shown on BLD 3b. Speed-independent operation of the reply system was lost when the load was divided in this manner. It is possible that a speed-independent reply system with some form of divided load exists; however, none of the attempts to design such a system were successful.

It does seem ironical that the solution to the two-valued signal problem which made it possible for one portion of the logic to operate in a speed-independent manner was instrumental in causing another portion of the logic to be non-speed-independent.

3.2.1.2 Memory Element Outputs and Conditional Operations

In evaluating BLD 2, two assumptions were made: namely, that the conditional signals d1 and $\overline{d1}$ had attained stable states prior to the excitation of this particular CP and, secondly, that these signals would not change value until after the conditional operation was completed. It can be seen from BLD 6 that if the conditional signals were the direct outputs of a memory element, the two assumptions would be valid and both the memory element logic and the conditional logic would operate in a speed-independent manner. This is true because the control step which set the memory element prior to the conditional step could not have been completed until the outputs were stable complementary values. If, however, a logic element has been inserted between the memory element output and the conditional logic, then neither logic can be considered speed-independent.

The logic for several of the DC memory elements is not speed-independent because the fan-out requirements could be met only by adding logic elements between the memory element and the conditional logic. This could be avoided in future designs either by creating a memory element output circuit with greater fan-out capability or by duplicating memory elements.

One further comment should be made about the use of conditional signals. Many of the conditional signals used throughout DC originate in the non-speed-independent Arithmetic Units rather than from within control, e.g., signals C_1 and C_2 on BLD 6. Since these signals which originate in non-speed-independent logic could never be handled in a truly speed-independent manner, their use was justified whenever a "worst-case" timing analysis indicated that the signals would be stable before the conditional OR's were gated. This apparent departure from speed-independent operation is another situation where the designer is free to set the boundary between the speed-independent control and the non-speed-independent Arithmetic Units.

3.2.2 The Size of DC

DC seems large--large in physical size and large in the number of transistors used. The physical size of DC, per se, could not have been responsible for DC being non-speed-independent. It could, however, create difficult problems for the signal distribution system and thereby be partially responsible. The Main and Exponent Arithmetic Units together utilize about 6,000 transistors whereas DC uses 6,800. It hardly seems necessary to use one transistor to control one transistor.

3.2.2.1 Design Requirements

There were three design requirements that could have been partially responsible for increasing the size of DC: (1) the requirement that

speed-independent logic be used, (2) the decision to perform the design of DC using the building-block method, and (3) the complexity of the flow chart.

The primary way in which speed-independent logic differs from other forms of asynchronous logic is in the use of reply signals and in the cross-coupling of outputs to detect both signal transitions. These reply signals cannot be created nor can signals be cross-coupled without the addition of transistors. Therefore, speed-independent circuits require more logic and more transistors than other forms of asynchronous logic. The magnitude of the difference is difficult to predict because the elimination of reply signals would remove the indication of the status of the devices being controlled. Therefore, proper operation could be obtained by the addition of timing delays. The number of delays and complexity of delay circuitry required is the quantity that is difficult to estimate. Only when these items are established could a comparison of the size of a speed-independent control and another asynchronous control be made.

The second factor that influenced the size of DC was the decision to design DC using the building-block method. When this method is used, almost all of the flow chart CP's have a sequencing memory element associated with it. There are 83 sequencing memory elements in DC and only a very small fraction of the (2^{83}) states defined by these memory elements are used by control. Type D logic, which is discussed in Sec. 4, is a form of logic which greatly reduces this number of sequencing memory elements. One of the questions this investigation attempts to answer is whether a reduction of these memory elements produces a significant reduction in the total transistor requirement.

The two previous paragraphs have discussed factors which would affect the design of a control and the size of a control when different design methods were used. A consideration of the flow chart, on the other hand, involves

evaluating the complexity of the sequencing job which the control is required to perform. That is to say, DC would have been smaller, regardless of the design method used, if a less complex flow chart with fewer instructions had been used. In discussion the creation of the flow chart, Gillies [GI-2, GI-3] points out that one way to make allowance for the imbalance between the standard operating time of Arithmetic Units and Core Memory is to use an order code which will "...provide more powerful instructions for a given number of bits, making one instruction do the work of several." This principle was followed in the formation of the DC order code; nine out of 37 DC orders are multipurpose orders which could be eliminated without impairing the ability of the machine to compute. However, the purpose of these instructions is to correct an imbalance in operation times and elimination of these orders might significantly affect the overall performance of the computer. DC has been made larger by the addition of these multipurpose orders, and the increased size of DC has made it difficult to design a speed-independent control; however, in light of the overall time for a given computation this may have been a justifiable decision.

3.2.2.2 The Use of Selectors

Most of the shifting operations in the Arithmetic Units of the new Illinois computer are performed through the use of selectors. A selector is a special AND-OR circuit usually located ahead of a register gate. Thus two logic elements, a gate and an n-way selector, can perform the shifting operations usually accomplished by n gates. Due to the nature of the register flipflops used, a multiple-gate input required more transistors than a gate and selector input whenever more than three ways were desired. When the Arithmetic Units were designed, most of the registers were expected to have at least four input paths; therefore, they were designed with a gate and selector input.

At the time of this decision, no specific information was available as to the logic required in control.

The selector mechanisms of BLD 7 are used to control the Arithmetic Unit selectors just described. The total of all logic which is concerned with selectors and selector control mechanisms accounts for 26 per cent of the transistors in DC. Granted that the operation performed is necessary, there is reason to question such a large percentage of the total. One further reason to question the use of selectors is that during the design process the number of input paths into four of the Main Arithmetic Unit registers was reduced to three, thus eliminating the advantage mentioned above.

Except for one reservation, it appears that the design of DC would have been easier and DC would have been smaller in size if the selectors had been eliminated. The reservation is concerned with the number register where the selector is located on the register output and the selector output serves as one input to the adder. In this situation the memory of the selector control mechanism is essential so that the adder input will remain stable. If the selectors were completely eliminated, a special form of gate would be required at this point or the method of shifting the contents of the number register would need to be changed. It is possible that special gates and their control logic might be no less expensive than the present selectors.

3.2.2.3 Mechanical and Circuit Factors

An actual count of the number of transistors used for various functions within DC, coupled with a re-evaluation of the construction techniques used, has shown that the size of DC could have been reduced. The reduction could have been accomplished by increasing the transistor density and by relocating some nonessential transistors. The size could also have been reduced if some of the

transistors used were replaced by presently available high-current, high-power transistors. Both of these factors are ones which would be adjusted if DC were a production prototype and a final model were to be built. It does not appear that the size reduction would be any greater than it would for any other piece of electronics equipment subject to revision and reconstruction.

In DC a form of nonrestoring circuitry is used which requires one transistor and one diode for each input. This nonrestoring circuitry accounts for 31 per cent of the DC transistors. If all, or part, of this logic were converted into diode logic, a considerable savings in transistors would result. However, since the purpose of this evaluation is logic design rather than circuit design, and since such a change would be justified only after considerable circuit analysis, this interesting possibility will reluctantly be left as an unverified conjecture.

3.2.3 Speed of Operation

Several sets of timing measurements have been made in the Arithmetic Units and in DC. These measurements indicate that for some of the arithmetic operations, DC allows more time than needed between successive gating operations. That is, for these operations the speed of the Arithmetic Units is higher than the speed of control. However, for other operations the logic of the Arithmetic Units operates too slowly and reliable operation is obtained by adding timing delays in DC. These cases are very much a function of the method of doing arithmetic. The reason is as follows: the conditional signals used at one CP are very often the result of arithmetic operations in the previous step. For example, some of the inputs to the Exit Decoder, shown on BLD 8a, are the result of performing the gS and gR operations at CP 81. These results must then arrive at CP 82 in time for the conditional setting of the EX memory

element to be performed correctly. Since this same form of operation occurs throughout control, the speed of operation of DC cannot exceed a certain value.

DC sends gate and selector signals into the non-speed-independent Arithmetic Units. Somewhere along this path the speed-independent control ends and the non-speed-independent logic begins. In practice this point is where the gate or selector reply signal is obtained. As this break point moves toward control the speed of operation increases and the reliability of operation decreases because less information is contained in the reply. As the point moves toward the Arithmetic Units the control will operate more slowly but more reliably due to the increased information content of the reply signal. As long as faulty operation of the type described above does not occur, the speed of operation can be controlled by adjusting this reply take-off point.

3.2.4 Summary

The objective in designing DC was to create a speed-independent control. This objective was almost attained; however, the method used to distribute some signals was not speed-independent. Thus even though at least 90 per cent of the circuitry of DC does operate in a speed-independent manner, the total evaluation of DC must be that it is not speed-independent. The primary causes for the difficulty were that the signal distribution systems were to drive (1) a large fan-out, (2) fan-out at a distance from the source, (3) with a fairly large signal swing, and (4) with cable drivers of moderate current driving capabilities. The method of accomplishing it in DC was to divide the load into small sections and use parallel circuit elements in a non-speed-independent expanding tree of logic. The reduction or elimination of any of these factors would have made the problem much easier. Additional investigation will be required to determine whether the problem could have been eliminated.

DC is larger than desired and its large size has made the distribution of signals difficult. The following five important factors deserve consideration when evaluating the size of DC: (1) the speed-independent logic used in DC requires somewhat more circuitry than is used by other forms of asynchronous logic; (2) the logic of DC makes very inefficient use of its sequencing memory elements; (3) the flow chart used in the design of DC could be simplified without losing the ability to compute, but the advisability of simplifying the flow chart is questionable; (4) the advantages gained through the use of selectors to perform shifting have been nearly eliminated and thus DC could be designed without them; and (5) the experience gained in building this computer has brought to light several mechanical and circuit refinements which could reduce the size of all sections of logic.

Although the design of DC did not completely achieve the goal of speed-independent operation, a great deal of information about speed-independent circuits has been gained. DC has not been operating long enough to obtain very much quantitative reliability data, but the process of checkout of the newly-constructed sections of logic proved that the location of faults within speed-independent circuits is much easier than in non-speed-independent logic.

3.3 Spindac P

Spindac P was designed using the design methods for type P logic and the general rules for all of the Spindacs as set forth in Sec. 2. This control realizes the sequencing operations specified in the flow chart of Fig. 1. Although this flow chart is less complicated than the one used for the design of DC, Spindac P is a miniature equivalent of DC. This fact is illustrated by the following comparison. The sequencing operations of DC are controlled by 89 CP's and since DC contains 6,830 transistors this is an average of 76.7

transistors per CP. Spindac P contains 2,190 transistors and 29 CP's which is an average of 75.7 transistors per CP. These averages have no real meaning other than to serve as an indication that Spindac P is indeed a miniature equivalent of DC.

In the process of interconnecting the BLD's of type P logic to form DC, each of the previously described BLD's was used at least once. In the design of Spindac P all but two were used. Spindac P does not have need to conditionally bypass a step (BLD 4) or to use slow-fast shifting in a loop (BLD 8b). With these two exceptions, Spindac P makes use of all of the design techniques discussed in the previous subsections of Sec. 3. A detailed description of Spindac P will not be made since Sec. 7 contains a thorough discussion and comparison of the various Spindacs. Tables II, III, and IV in that section give the numbers of transistors used in the various sections of the logic for Spindac P.

4. TYPE D LOGIC AND SPINDAC D

The design of a control using type P logic is accomplished by making speed-independent interconnections of the needed BLD's, that is, the control is designed using the building-block method and logic is "built up" one CP at a time. It seems reasonable to assume that a system design method exists whereby the entire sequencing problem can be solved first and then the details of the logic required at various CP's added later. With such a design method, the various branchings and recombinations required by the flow chart would be incorporated into the sequencing portion of the control initially rather than as the design proceeds from CP to CP. One means of visualizing such a process is to consider that each of the CP's on the flow chart is assigned to one node of an n-cube; then as a control signal is caused to move through the cube in a Gray Code fashion, the CP's will be energized one at a time.

The first new design method proposed by this dissertation is a system design method in which the flow chart CP's are indeed considered to be the nodes of an n-cube. The movement of a control signal from one CP to the next is determined by a sequencing register containing a small number of memory elements. The outputs of these flipflops are decoded to identify the control CP which is to be activated. As a result this logic is called type D logic (D for Decoding).

In addition to any advantage which might accrue due to the change from the building-block design method of type P logic to the system design method of type D logic, it is anticipated that type D logic will produce designs requiring fewer transitors since fewer sequencing memory elements are used. In almost every case within type P logic, a NAND circuit at a CP will have a sequencing memory element associated with it. Thus for the 29 CP's in Spindac P, 26 memory elements were used, yet only a very small fraction of the

67 million (2^{26}) states defined by these 26 flipflops were ever used. In performing this same sequencing job, Spindac D uses only six memory elements. One of the criticisms of type P logic was that it required too many transistors and this larger size was an important factor in causing DC to be non-speed-independent.

A third anticipated advantage of the type D logic is that it will require fewer reply transistors. Reference to the BLD's of type P logic will reveal that when a reply signal is used at a CP it is used as an input to a NAND and an OR circuit. It will be shown that type D logic will also require the input to the NAND circuit; however, the decoding system should greatly reduce the use of replies in OR circuits. It is anticipated that the previously mentioned reduction in memory elements and this reduction in reply transistors will produce designs requiring significantly fewer transistors.

4.1 Development of the Design Method

In general, the synthesis procedure followed when a given flow chart problem is to be realized using type D logic contains only three steps. The initial step is to assign the CP's from the flow chart to the nodes of an n-cube in an efficient and convenient manner. Then a register containing a sufficient number of sequencing memory elements is created and connected to NAND circuits to perform the decoding and to form a signal for each CP. The last step is to introduce additional logic at the CP's and at the sequencing register to perform the gating, selecting, and other needed operations and to include the required conditional operations. Even though the assignment of CP's to the nodes of the n-cube is the first part of the synthesis procedure, the design of the decoder logic and the register of sequencing memory elements

will be discussed first. If this could not be done efficiently and in a speed-independent manner, the description of the remaining steps in a type D logic realization would be useless.

4.1.1 Selection of a Decoding System

The objective of this portion of the study was to develop a speed-independent manner in which the outputs of a small number of memory elements could be decoded to activate one of a group of CP's. When a logic design problem such as this is to be solved, it is very easy to overlook the simple, direct solution while studying elegant or unusual configurations. This in fact did happen as a decoding design for type D logic was being sought. Many attempts were made before the logical realization presented here was tested and found to be usable and useful. This circuitry was investigated only after the "art" of logic design had given way to a Huffman synthesis procedure [HU-1].

At the center of a type D logic there is a register of (n) sequencing memory elements whose outputs can be decoded to define 2^n different signals. Each of the 2^n signals can be formed with a NAND circuit whose output is the activating signal for the CP assigned to that node of the n cube. Figure 15 shows a four-step loop of type D logic in which CP's D11 and D14 realize the parallel operation of two gates, D12 performs a Null step and D13 performs two gates in parallel and indicates the method used to obtain multiple operation. Note that the replies and/or other signals used to change the state of the sequencing memory elements cause only one memory element to change state at any given time. This causes the excited or activated point in the n-cube to move in the desired Gray Code fashion. Probably the greatest surprise was the fact that this simple direct approach to the problem could be done in a speed-independent manner.

The utter simplicity of this means of decoding makes it possible to design the system logic almost as fast as one can write. To explain this statement let each of the memory elements in the sequencing register be designated by a Boolean variable (x_i) where i = 1, 2, 3, ..., n. These (n) variables can then be combined to form the 2^n fundamental products or minterms of (n) variables and each minterm is associated with one node of the n-cube. In type D logic the output signal for a CP is obtained from a NAND circuit. Each of these CP's is assigned to a node of the n-cube and the decoding is accomplished by using the signals corresponding to the minterm for that node as inputs to the NAND circuit. The fact that the decoding requires no logic other than the NAND circuits which are required for other reasons, and that the inputs to the NAND are the well-known minterms of (n) variables is the reason that this has been referred to as a simple, direct decoding method.

The logic of Fig. 15 will operate in a speed-independent manner provided the meory elements used exhibit the characteristics of those described on BLD 6 in Sec. 3.1.2.6. These memory elements have two unique features which are important. The first characteristic is the sequential manner in which the outputs change state, that is, only one of the element's outputs is ever excited at one time. If both outputs became excited whenever an input condition changed, then the outputs could change state concurrently and the logic of Fig. 15 would not be speed-independent. The second important feature is the fact that this memory element does not possess a disallowed input state. Whenever one of the sequencing memory elements is presented with the input state (0-0), which would be the disallowed state, it indicates that some logic element is excited but has not reacted. With this logical configuration, the next step in the Gray code movement through the n-cube is not taken until this excited element reacts and the (0-0) input is changed to (0-1) or (1-0).

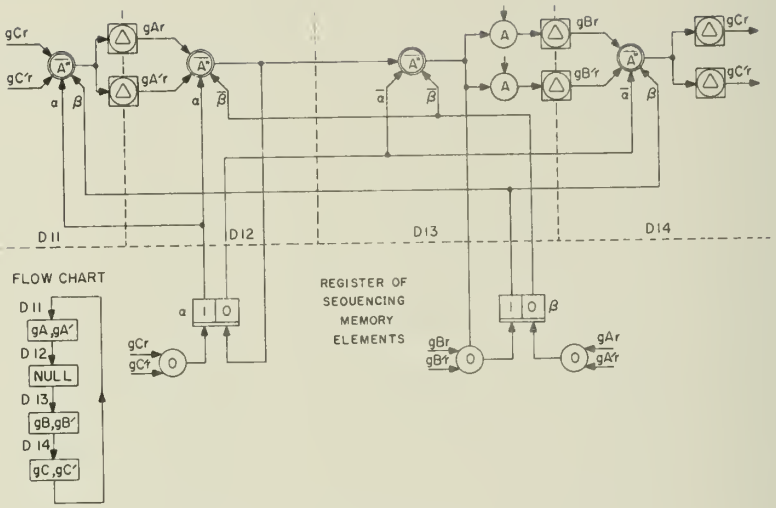


Figure 15 - Four Step Loop Using Type D Logic

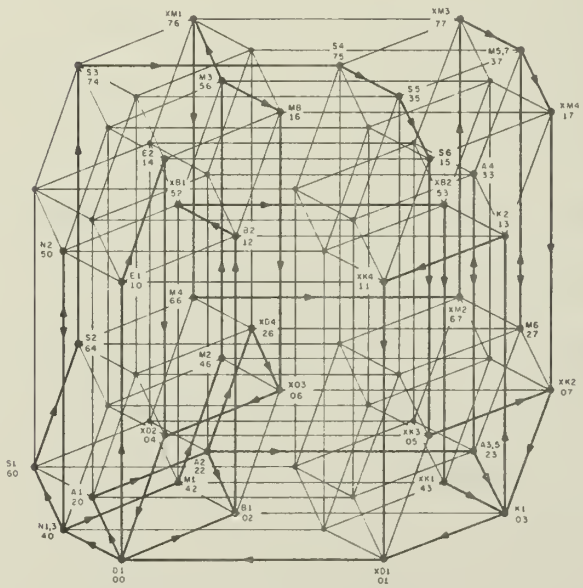


FIGURE 16 - 6-CUBE WITH NODAL ASSIGNMENTS

4.1.2 Assignment to the Nodes of an n Cube

Now that a speed-independent manner of decoding the outputs of the sequencing memory elements has been established, the assignment portion of the type D logic design process can be studied. The assignment part of the design process is the selection of an efficient mapping of the flow chart CP's onto the nodes of an n-cube. This process will be illustrated by discussing the assignment portion of the design of Spindac D.

All of the control operations required of the Spindacs are given by the flow chart of Fig. 1. Each CP of this flow chart must be assigned to one node of the n-cube. Then additional assignments not appearing on the flow chart can be made to facilitate sequencing and to simplify decoding. Although it might appear as if this portion of the design process was a problem of minimum assignment of states of the decoding register, this is not the case. The most desirable assignment for this problem is the one which utilizes the fewest transistors for the entire control, that is, this is a minimum transistor problem which is not necessarily a minimum decoding register state assignment problem.

One of the developments to be discussed later is the fact that the two CP's which form the entrance and exit points for an iterative loop, such as is used in addition and multiplication, can be assigned to the same node in the n-cube. As a result, the 29 CP's of the flow chart need have only 26 nodal assignments. Arithmetically, it should be possible to make these 26 nodal assignments within a 32-node 5-cube. But these 26 nodes are interconnected with many branchings and recombinations so that it is not at all obvious that they will fit onto a 5-cube. As a matter of fact an assignment was made in which 26 nodes were assigned onto a 5-cube, but the result was not a minimum transistor

UNIV... OF ILLI... LIBRARY

assignment even though it was a fairly efficient nodal assignment. Some of the reasons for this will be discussed in subsequent paragraphs.

A far more efficient assignment, transistor-wise, for these 26 nodes was made onto the 64-node 6-cube shown in Fig. 16. The heavy lines indicate the particular Gray code movements on the 6-cube which are used as flow chart transitions. Light lines are used for all the remaining movements which are not used as flow chart transitions. The two numbers adjacent to each node give the state of the six sequencing memory elements (as two octal digits) and the CP assigned to that node.

All of the possible transitions for Spindac D are shown as heavy lines on the 6-cube of Fig. 16. The arrow indicates the direction of movement of a control signal--those transitions which show arrows in both directions are the iterative loops and the signal does alternately proceed in each direction. Only the nodes assigned have been labeled to simplify the figure. This assignment includes 14 assignments over and above the required 26; each of these additional assignments has a label beginning with an X. For example, in the multiply sequence, M 3 is at node 56 and M 4 is at 66. To move in a Gray Code manner one must pass through XM 1 which is at node 76. Each of these additional assignments was made to facilitate Gray Code movement, or to simplify the reply logic. In almost all cases the movement to, or away from, one of these additional assignments was a Null step which requires only five transistors. This then formed the criterion used to evaluate the advisability of an additional assignment.

The actual process of node assignment was done in the following manner regardless of whether the attempt was onto a 5-cube or 6-cube. An arbitrary assignment was made for the K 1, K 2 and D 1 CP's (where the CP designations refer to the flow chart). This was done first because all

subsequences end in either K 1 or D 1 and all but K 1 and M 1 are fed from D 1. Next, a set of tentative assignments was made for the normalization (N's), addition (A's) and clear addition (B's) subsequences. It was done in this manner because 25 of the 45 transitions specified by the flow chart are associated with these five subsequences which account for only half of the CP's. From this point on the assignment process was one of cut-and-try until the design was completed or scrapped.

Earlier it was mentioned that the 26 essential nodes of the flow chart would fit onto a 5-cube but that this was not a minimum transistor assignment. It is possible that this situation could be greatly improved and the assignment onto a 5-cube could be optimally done, provided the flow chart was modified to have fewer essential nodes. Note on the flow chart that A 1, B 1 and E 1 all perform the gate (gEA), set the memory element (C), and set the selectors (sEA) and (sD). By using the signals β and θ in conditional logic, a combined CP ABE 1 could be formed. Similarly the operations of M 1 and S 1 could be performed by the normalization exit N 3; also M 2 and S 2 could be combined. Through the use of these combining techniques the number of nodes required for the flow chart could be reduced to 19 or less. It is quite possible that these 19 nodes could be satisfactorily assigned onto a 5-cube. If such a flow chart were formed the amount of conditional logic would be increased but the sequencing logic would certainly be reduced regardless of the design method used. The idea that a simplified flow chart would reduce the size of the control was also mentioned in Sec. 3.2.2.1 concerning type P logic. However, the simplification desired by each design method would probably not be considered an advantage by the other design methods. Therefore, the advantages gained by simplification of the flow chart have been enumerated but no actual designs will be made.

The discussion of the assignment of additional nodes made mention of the fact that some of these additional assignments were made to simplify the reply logic. The simplification results when a device reply is used to change the state of a minimum number of sequencing memory elements. On the n-cube used to represent a type D logic control, each of the (n) degrees of freedom is associated with a particular sequencing memory element. To move the control signal from one CP to a neighbor, one must change the state of the particular flipflop associated with that direction of movement. If then, it is desired to move the signal back to the original CP, the same memory element must be changed back to its original state. Thus the n cube has (2n) signals which are used to change the state of the sequencing memory elements. The logic required to form these (2n) signals will be simplified if the reply from a device is used in only one or two places. In Spindac D for example, each of the 12 possible directions of movement on the 6 cube is used at least once; however, most of the transitions which require the use of replies are made in one of five directions. Of the 17 reply signals coming from memory elements or from Main Arithmetic Unit devices, 15 have a fan-out of either one or two. The efficiency with which the Exponent Arithmetic Unit replies are used is not quite as high.

4.1.3 Design Details at the CP's

The fundamental difference between type P logic and type D logic is the manner in which the sequencing memory elements are used to control the NAND circuits at the CP's. As a result, many of the techniques shown on the BLD's of type P logic are used in type D logic as well. This is quite evident when one compares Fig. 15 with BLD's 1 and 3. These discussions of the design details will make frequent references to the BLD's of Sec. 3. Many references

will also be made to Fig. 15 which shows all of the essential methods of causing control to move from one CP to the next.

In the process of designing using the type D logic, the designer will have need to realize various sequencing situations as presented by the flow chart. The following fourteen possible flow chart situations were discussed for type P logic (the set of BLD's) and thus this list will provide a comprehensive comparison of the logic design techniques.

a. Null Step. When a Null step is performed a control signal passes from one CP to the next but no machine or device operations other than this take place. The transition from D 12 to D 13 of Fig. 15 is a Null step. The output of the D 12 NAND circuit is used to change the state of the α sequencing memory element. If D 12 had contained conditional logic, then the output of this conditional logic would have been used at the α memory element.

b. Parallel Operation. The logic at D 11 of Fig. 15 is used when parallel operation of devices is desired. To move from D 11 to D 12 it is necessary that the β sequencing memory element be changed from the "1" to the "0" state. The flow chart also specifies that this transition is to operate the two gates gA and gA'. Except in the case of multiple operation, which is discussed in the next paragraph, the replies from the devices are used to cause the memory element to change state. Device replies can be used as shown at the sequencing memory elements when conditional operations

are involved as well as the unconditional operations shown on Fig. 15. The logic at D 14 also performs the parallel operation of gates.

c. Multiple Use of a Device. The logic of D 13 and at the input to the "1" side of the β memory element of Fig. 15 is used when multiple operation of a device is desired. The AND's ahead of the gate logic symbol are used to collect the various requests for the device. The output of each NAND involved in the multiple operation is used as an added input to the proper OR so that the activations of the gates gB and gB' will excite the proper memory element in the sequencing register. If α and β were only part of a register of sequencing memory elements, then for efficient use of reply transistors the designer would make the nodal assignments so that all uses of gB and gB' were associated with transitions that changed β from a "0" to a "1". In this case all the NAND signals would be collected in the AND portion of an AND-OR with the gate replies as inputs to the OR. The logic for this is shown in Fig. 17, at the input to the Y_1 memory element. In this case the two transitions are from node 24 to 25 and from 60 to 61. Fig. 17 also shows the method used to connect OR's in cascade so that the logic will operate in a speed-independent manner. The dotted signal line from CP 24 is necessary if gA and gA' also have multiple requests; however, if the gates are

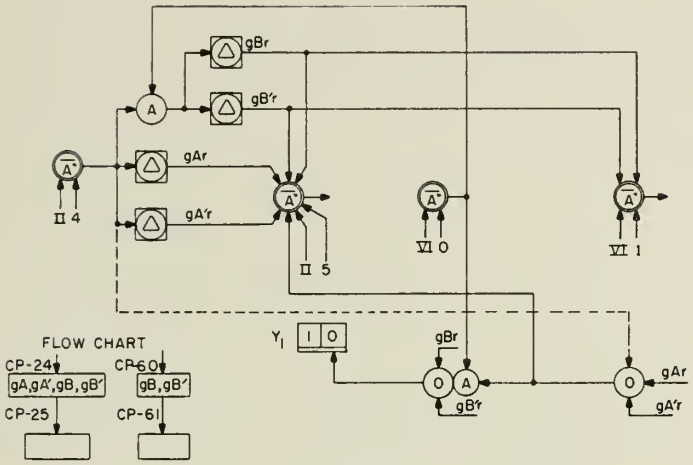


Figure 17-Using Cascaded OR's To Decode Replies

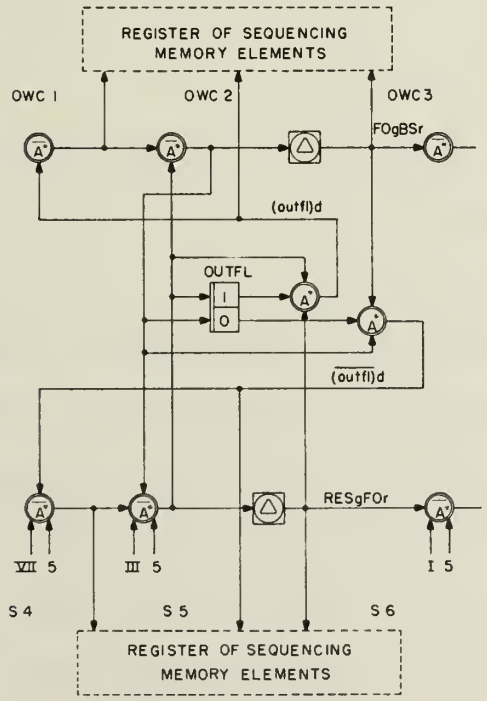


Figure 18-Interconnection Of Two Controls Using Type D Logic

used exclusively by CP 24, then this signal is not needed.

d. Conditional Operation and Bypassing. The logic necessary to form conditional requests and bypass requests as shown on BLD 2 is used in all four types of logic. Since the device gives a reply regardless of the condition, the reply signal is handled as in (b) or (c) above.

e. Conditional Bypassing of a Step. This operation is not used in Spindac D. If, however, this operation were to be realized in type D logic it would be accomplished in the following manner. Conditional branching logic would be used at the output of the NAND circuit and the appropriate sequencing memory element would be excited either by a Null-step type of operation or one in which devices were activated as discussed in (a) and (b) above.

f. Conditional Branching and Recombination. Conditional branching signals are formed with OR's at the output of the NAND as shown on BLD 5. The execution of the desired operations or of a Null step after the branch is discussed in (a) and (b) above. It is interesting to note that type D logic does not need any special logic for recombination. A recombination is accomplished when two or more transitions, each moving in a different direction, terminate at the same node of the n-cube, as for example, the K 1 node at 03 on the

6-cube of Fig. 16. Signals from nodes at 07, 23 and 43 are combined. Each signal causes a different sequencing memory element to change state and thus the only circuit elements that could be interpreted as recombination logic are the reply transistors at the NAND circuit of the CP.

g. Setting of Memory Elements. The creation of a request to set a memory element (or a selector) is accomplished in exactly the same manner for all types of logic. The reply from the device is handled as discussed in (b) or (c) or (d).

h. Setting of a Selector. See comment in (g).

i. Scale of Two Loop. The logic necessary to execute the iterative operation within a scale of two loop is simple with type D logic. The outputs of the exit memory element are used in conditional branching OR's at a CP which serves as both entry and exit point for the loop. The loop transitions repeatedly change the state of only one of the sequencing memory elements. On the n-cube, the control moves back and forth between two nodes until the exit condition is met and then a different sequencing memory element is changed and control moves off in a new direction. The 6-cube of Fig. 16 contains three of these scale-of-two loops. The CP's which make the addition loop are A 3,5 (23) and A 4 (33), the multiplication loop includes the

M 5,7 (37) and the M 6 (27) nodes, and the normalization loop uses N 1,3 (40) and N 2 (50).

j. Slow-Fast Shifting in a Loop. Slow-fast shifting in a loop is nothing more than a special case of conditional operation and bypassing. It is not performed by any of the Spindacs; however, if it was to be done, it would be accomplished as discussed in (d).

k. Using a Device Twice in a Row. There are several places in Spindac P where it has been necessary to operate a device in two succeeding CP's. If a Null step, or any other step had occurred between these two operations of the device, then standard sequencing techniques would have been used. There is no place in Spindac D that a device must be operated twice in a row simply because the introduction of the additional nodal assignments has separated the two device operations by a Null step.

l. Interconnection of Two Controls. The logic used to interconnect Spindac D and its Out Write Control is shown in Fig. 18. This logic is not a type D logical equivalent of BLD 10 even though the two logics perform the same function. The reason the two are not equivalent is that BLD 10 used a device, the OUTFL memory element, twice in a row and the logic of Fig. 18 does not. In every case where Spindac P was required to operate a device twice in a row, the introduction of an additional nodal assignment for Spindac D has removed the need for this type of logic as discussed in (k) above.

4.2 Spindac D

The design of Spindac D began with the assignment of the essential CP's of the flow chart to the nodes of an n-cube. The initial assignment to a 5-cube was never carried to a complete design; rather it soon became evident that the assignment was not desirable and the 6-cube was used instead. One of the most important areas in the logic for a decoder system such as this is the register of sequencing memory elements. Therefore, in both cases (the 5-cube and the 6-cube assignments) the logic design portion of the process began at this register. Once this was accomplished the difficult part of the logic design was complete and there remained only the task of establishing the CP's and performing the conditional operations desired.

The simplicity of the decoding system used for type D logic was mentioned in Sec. 4.1.1. In this section it was explained that the decoding is actually accomplished in the NAND circuits at the various CP's. The signal inputs to each NAND circuit which come from the register of sequencing memory elements are those which correspond to the minterm expression for that node of the n-cube. For Spindac D this would imply that every NAND had an input from each of the six sequencing memory elements as well as from the appropriate replies. Since Spindac D uses 40 nodes of the 6-cube, 240 transistors would be required for just this purpose. Fortunately this number of transistors can be reduced without losing speed-independent operation. The outputs of the sequencing memory elements are combined using AND circuits into 16 special signals--one signal for each of the eight most significant octal digits and one for each of the least significant octal digits of the node. These signals are shown on Fig. 17 with the most significant digit being specified by a Roman numeral and the least significant one by an Arabic numeral. This system is generally applicable provided the output signals from the sequencing memory elements are used only in one set of combining AND circuits.

Spindac D was designed in accordance with the general rules given in Sec. 2 and with the rules for designing type D logic presented in the preceding subsections of this section. The logic for Spindac D required the use of 1,986 transistors which is 205 or 9.4 per cent fewer transistors than used by Spindac P. Although this reduction is gratifying, it was anticipated that the savings would be more significant than this. Several possible reasons for this limited improvement and a general discussion of the Spindacs will be found in Sec. 7. Even though a great deal of care has been taken to investigate various approaches to each problem, the conclusions reached and the number of transistors required are a function of the particular assignment made onto the 6-cube and the decoding system chosen.

4.3 Synthesis Procedure for Type D Logic

These steps were used in the design of Spindac D and are given as a guide to be used if a similar type of control design problem is to be solved. It is assumed that the design problem is described by a flow chart and that in addition to desiring a speed-independent control, it is desirable to utilize a minimum number of transistors. The synthesis can be performed in the following manner:

1. Choose an n-cube that has more nodes than the number of CP's in the flow chart. The experience with Spindac D would indicate that the number of nodes needed is nearly twice the number of CP's.
2. Attempt to form groups of devices which are requested together. This is done by studying the flow chart. This process is described in Sec. 5.1.1 in connection with type R logic; however, a less extensive form of

this grouping was done for Spindac D. After the devices have been grouped, assign a transition direction on the n-cube to each group. This direction of movement on the n-cube will be used whenever the devices in the group are requested by the flow chart.

3. Make an assignment of the CP's of the flow chart onto the nodes of the n-cube chosen. This step will also include the introduction of additional assignments whenever needed to facilitate Gray code movement or to cause the assignment to be compatible with the directions assigned in (2).
4. Design the sequencing memory element register and the reply decoding logic required to form the excitation signals for the memory elements. The minterm expression for the pair of nodes involved in a transition will identify the sequencing memory element to be excited.
5. Repeat steps 1 through 4 inclusive as many times as needed to obtain a minimum transistor compromise between the additional assignments and the size of the reply decoder logic.
6. Complete the details of the logic at the various CP's. Much of this logic will be independent of steps 1 through 5; however, the additional CP's and the signal inputs to the NAND circuits will not be known until (5) is completed.

It has been the writer's experience with all four of the design techniques investigated, that the process of achieving a complete logic design is an iterative process. Seldom is a design satisfactory without at least two repetitions of the entire design process and many repetitions of various parts of the procedure. This is particularly true of the nodal assignments. Several attempts must be made before the designer can appreciate the advantages and disadvantages of additional assignments, grouping of devices, etc.

5. TYPE R LOGIC AND SPINDAC R

One of the most difficult problems encountered in the formulation of type P logic was the creation of a signal distribution system. In the evaluation of the design of DC, Sec. 3.2, it is stated that the signal distribution systems are the cause for DC being non-speed-independent. The primary objective of this investigation into a second new design method is to develop a speed-independent signal distribution system. Most of the signal distribution problems occur in connection with device replies; therefore, this logic will be referred to as type R logic (R for Reply).

A secondary objective of the study of type R logic is to reduce the size of the control through a reduction in the number of transistors required by the reply systems. Such a reduction would reduce the overall size of the control and thereby simplify the signal distribution problem. It is rather obvious that these two objectives are not independent and quite possibly the solution of one will facilitate the attainment of the other.

Consider for a moment the logic of BLD 1 which realizes, for type P logic, the parallel operation of devices. In CP 11 the two gates gS and gR are operated in parallel and as a result their replies, gSr and gRr, are used as inputs to the NAND and OR circuits of the following CP. A study of the flow chart will reveal that in an actual control logic each of the CP's will actuate many devices in parallel. If the replies from two, three or more of these devices could be combined into one collective reply, then the number of transistors used for reply inputs to these NAND and OR circuits would be greatly reduced. This study into type R logic will investigate the possibilities of combining replies as a means to reduce the number of reply transistors used and as a means to reduce the number of reply-driver circuits needed. If the number of reply-driver circuits is significantly reduced, then the designer can

justify using more powerful, and more expensive, transistors to obtain the needed current output and thereby the reply distribution system might be made speed-independent.

5.1 Development of the Design Method

The primary concern of the investigation into type R logic will be the development of speed-independent reply systems and signal distribution systems. In order for the type R design method to represent a complete method for designing control logic, some sequencing means will have to be added to the new reply systems. The building-block method of type P logic was chosen since the substitution of a speed-independent signal distribution system for the present non-speed-independent type P system should result in a complete design method which is speed-independent. Type R logic will make use of most of the design techniques and the sequencing methods of type P logic and will make only those modifications necessary to accommodate the new signal distribution systems.

The development of the reply system used in type R logic was accomplished in three steps. The initial step was to make two interrelated, but apparently unjustified, assumptions. The first assumption was that the fanout required by any of the signal distribution systems could be driven in a speed-independent manner. In other words, it was assumed that the divided load technique of type P logic was not needed and that some form of acceptable driver circuit could be designed. Secondly, it was assumed that a speed-independent method for combining the replies from several devices into a single reply existed. The effect of making these two assumptions was to assume that type R logic could be designed.

Since the initial step in the design process was the assumption that type R logic could be designed, the second step was to determine whether a speed-independent combined reply system would be useful if it existed. This amounted to finding sets of requests and replies which could profitably be combined into a single reply. When this study showed that combined reply systems would be advantageous, the design of the combining logic and of the reply-driver systems was completed as the last step in the formulation process. This last step was essentially the task of justifying the two assumptions made in the initial step.

5.1.1 Request and Reply Subsets

Assuming that combined replies can be formed, the next portion of the design process is to determine the extent of advantage or disadvantage obtained from combining replies into groups. In any effort to determine what combinations of devices would be usable, a careful study of the Spindac flow chart was made. The device requests (which imply device replies) listed for each of the CP's of the flow chart were grouped in many ways until an advantageous set of combinations was achieved. It was found that nineteen of the twenty-five major devices controlled by Spindac could be grouped into five subsets. These subsets included all of the devices with large fan-out requirements. The flow chart specifies a total of 189 requests for all of the twenty-five devices. Ninety-one per cent of this number are included in the five subsets which means that the vast majority of the requests will involve one of these subsets. The remaining nine per cent of the requests will be handled as individual device replies. It was, therefore, concluded that a system of combining replies could very profitably be used.

The five subsets chosen, the designations given and the devices included in each are as follows:

1. The AQ subset includes the gA and gQ gates, the MsA and the sAQ selector mechanisms, and no memory elements.
2. The SR subset includes the gS, the gR, and the gES gates, and MsS and the sSR selector mechanisms, and no memory elements.
3. The ED subset includes the gate gEA, the two selector mechanisms, sD and sEA, and the C memory element.
4. The OZ subset includes the DgE gate, the OV and Z memory elements and no selector mechanisms.
5. The ΩL subset includes the EMgE gate, and the two memory elements, Ω and CL and no selector mechanisms.

A study of the flow chart will show that the devices in these subsets are indeed grouped together at the various CP's of the flow chart. For example, the operations at E 2 involve only the OZ subset and at M 4 the SR subset is needed. Several of the CP's require the use of more than one subset. The A 3 operations involve the SR and ΩL subsets and A 4 requires the use of the AQ, and the ED subsets as well as separate replies from the ESgEM gate and from the XA and CR memory elements. The operations at A 4 illustrate the one disadvantage of grouping replies into sets. Note that the flow chart requests the operation of only three of the devices in the ED subset; there is no request to change the setting of the C memory element. In these cases the logic at the CP will need to form a bypass request for the C memory element in order that all devices within the subset are activated. This fact will be explained completely in the next section. The gate ESgEM and the memory

elements XA and CR are examples of devices which could not profitably be included in the five subsets. A reply will exist for each of these devices; however, the reply driver will be capable of handling the fan-out requirement in a speed-independent manner.

5.1.2 Combined Reply Logic

In discussing the selection of the decoding system for the type D logic, mention was made of the fact that there are times when the simplest solutions to problems are overlooked. In designing the subset reply circuitry, the simple direct solution was the first one tried and all subsequent attempts to find a better solution were unsuccessful. It was imperative that the logic for combining replies be made simple so that the transistors added to the logic to form the combined reply would not exceed the number of transistors saved at the CP's so that the net result would be a smaller control. The logic used to form the combined replies for the OZ subset is shown in Fig. 19. Although some of the subsets involve more than three devices, the logic shown in Fig. 19 is typical of the circuitry used for all subsets.

An understanding of this logic can best be obtained from a short review of the two substeps involved in each sequencing control step when using type P sequencing methods. This review might be aided by reference to BLD's 1 and 2 as the operations are explained. A control step begins when a CP is activated and requests ("0" signals) are sent out from the NAND circuit at the CP to each of the devices to be operated. The several devices perform their intended operations and when part or all of the job is done, the device sends a "0" reply to the CP. When all of the device replies involved have changed to "0", the output of an OR changes to "0" and this causes some operations in sequencing control which result in the NAND output changing back to a "1". This changes all the requests to "1" and subsequently all of the

device replies also return to "1" to signify the end of that particular control step. The fact that all of the appropriate replies have changed back to the "1" state is determined by the NAND circuit in the CP following the one which initiated the requests.

It will be noted from Fig. 19 that the reply subset logic does not change this basic method of sequencing since when the signal OZO changes to "0" it signifies that all the device replies are "0" and when the signal OZA is in a "1" state it implies that all of the devices' replies are also "1". If the signals OZO and OZA are cross-coupled as shown in Fig. 19, then the subset replies can be connected to the sequencing control as shown in Fig. 20. Even though the combining logic for each subset has two outputs rather than the desired single output, the fan-out requirement for each subset output is only one-half of the fan-out requirement for the devices in type P logic. This is true since OZO is used only in the OR circuits at the CP's and OZA is used only in the NAND circuits.

The logic of Fig. 20 is a simplified presentation of the control logic needed to execute the operations specified at B 2 on the flow chart. The flow chart specifies that each device in the AQ subset be activated; however, the OV memory element of the OZ subset is not mentioned in the flow chart. Therefore, a bypass request is created in order that all devices in the subset give a reply signal. If this were not done, the signal OZA would change to "0" when the Zr or the DgEr changed to "0"; however, the OZO signal would remain in the "1" state. This illustrates the one undesirable feature of type R logic, namely, that each device within a subset must be active whenever any device in the subset is active. In the design of Spindac R, 36 bypass requests were added to the control to fulfill this requirement.

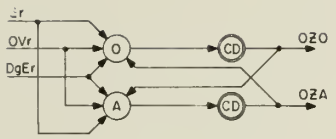


Figure 19-The OZ Subset Reply Logic

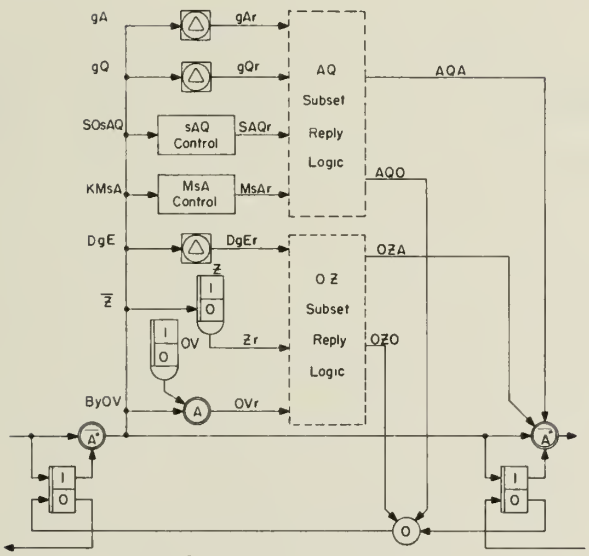


Figure 20-Control Point B2 Of Spindoc R

When it was determined that the subset reply logic of Fig. 19 was speed-independent and that its incorporation into sequencing logic as shown in Fig. 20 was also speed-independent, the second assumption in the design of type R logic was proven to be valid. A check of Fig. 20 will also show that the objective of reducing the number of reply transistors has been accomplished as well. The type R logic shown in Fig. 20 uses four transistors for reply signals whereas an equivalent control using type P logic would use 12 reply transistors. This is a clear savings of 67 per cent in the reply transistors for this particular CP.

5.1.3 Reply Drivers

At the time the cable drivers for DC were being designed, the presently available generation of silicon, high-current, high-speed epitaxial transistors were just beginning to emerge from the transistor manufacturer's research laboratories. Some of these units and some of the specifications for devices of this type were available for the circuit design engineer, but complete spread curves and the reliability data necessary for worst-case design methods (which is the method used to design the circuits for DC) were either nonexistent or of questionable validity. As a result, the circuit design group decided not to use these new transistors in the design of the DC cable drivers (reply drivers). Instead the drivers were designed with the transistor used throughout the new Illinois computer since it had been thoroughly tested and complete specifications and spread curves were available for use in the design. This was a low-power, high-speed transistor which produced drivers of only moderate current capabilities.

The situation has now changed and there is sufficient design data available to permit the design of high-current driver circuitry. As a matter

of fact, Wallace of the University of Illinois' Digital Computer Laboratory [WA-1] has designed an entire group of drivers with current capabilities ranging up to 500 milliamperes and with short enough operating times to be usable for these applications. Using these circuits as evidence of the fact that high-current devices can be designed and built, it is claimed that the reply-driver circuitry needed for Spindac R can now be designed.

5.2 Spindac R

After the design techniques and special logic diagrams needed for type R logic had been developed, the Spindac R control was designed. The design process was regulated by the general rules for all Spindacs which are given in Sec. 2. The control realizes, in type R logic, the sequencing operations specified by the flow chart of Fig. 1. The general design procedure which is described in the next section was used in the design of Spindac R.

Except for the modifications required by the type R reply systems, the sequencing methods of Spindac R are identical to those of type P logic. As a result, all but two of the BLD of type P logic were used in the design of the control. Spindac R does not have need to bypass conditionally a step (BLD 4), or to use slow-fast shifting in a loop (BLD 8b).

Spindac R was designed and found to require the use of 1,949 transistors. This number is over 11 per cent or 242 transistors less than Spindac P required. As with Spindac D, the reduction in the number of transistors required is gratifying; however, it is a little disappointing that the reduction is not more significant. The detailed description of the transistor distribution and a discussion of the various Spindacs will be found in Sec. 7.

5.3 Synthesis Procedure for Type R Logic

Each logic designer has his own special way of approaching the creation of a logic realization; however, the following steps were used in designing Spindac R and might serve as a guide to be used in a similar design problem. The synthesis could be performed in the following steps:

1. Study the flow chart and group requests into subsets with three or more devices per subset. In many situations the logic designer can obtain a more optimum flow chart from the logic design standpoint. Such feedback can influence the designation of "don't-care" situations and also, some time-ordering relationships might be altered if the logic can thereby be simplified.
2. Once the reply subsets have been established, the flow chart must be modified to include the bypass requests needed to activate all members of a subset whenever any member is active.
3. Design the combining logic for each subset. This will follow the general idea of Fig. 19 unless some special problem has to be solved. The reply drivers for the combined replies and for the individual devices can also be designed. It is anticipated that the only devices which will need reply drivers are those devices which were not included in any of the subsets.
4. Design the sequencing logic using the building-block method of type P logic and the BLD's of Sec. 3. The most successful method has been to create the logic required by each of the subsequences of the flow

chart as a separate logic first, and then when they all exist, make the necessary interconnections. The subsequences which contain an iterative loop, such as addition and normalization, usually take special care as do the store and decode subsequences which require the interconnection with another control.

It is quite possible that steps 1 and 4 might have to be done over several times before a satisfactory logic design is achieved. The desire to include more devices into the subsets to reduce the reply transistors at the CP's must be tempered with the knowledge of how many additional bypass requests are demanded by so doing. The need to redo step 4 is occasioned by the fact that there is a great deal of possibility for error in the interconnection of two logics.

6. TYPE C LOGIC AND SPINDAC C

Previous sections of this dissertation have shown that a control designed with either type D or type R logic will utilize fewer transistors than a type P logic design. The purpose of this section is to investigate the possibilities in combining these two design techniques into a third new technique. This last new design method will be referred to as type C logic (C for Combination).

6.1 The Design Method

In a very general way, the difference between type D logic and type P logic is in the method used to control the NAND circuits at the CP's. Both types of logic use a NAND circuit at the CP to create the signal which activates the devices requested by the flow chart; however, they use quite different means to select which NAND circuit will be activated. These two logic design methods use the same signal distribution systems and the same method of handling reply signals. Using this same mode of general analysis, the difference between type P logic and type R logic is in the signal distribution systems and reply systems. In fact, the design method for type R logic was formed by replacing the signal distribution system used for type P logic with a new one. It is proposed that type C logic be formed in a similar way by adding the new signal distribution system of type R logic to the new sequencing method of type D logic and thereby create a completely new logic design method. All of the techniques of type C logic are therefore already in existence--it is merely necessary to collect the ones needed and form them into a design method.

6.1.1 Request and Reply Subsets

The selection of sets of devices whose replies could be combined was for use in type R logic and was discussed in Sec. 5.1.1. A similar process was done in the design of Spindac D and was discussed in Sec. 4.3 as step 2 of the synthesis procedure. These two design methods had slightly different reasons for grouping replies and this should be kept in mind when the grouping is formed for type C logic. The grouping for type R logic was solely for the purpose of reducing the number of transistors used at the CP's for reply signal inputs. The grouping of replies for type D logic was also done to reduce the number of reply transistors required; however, there was a second important effect of grouping, namely, the group of devices would then always be activated when the n-cube transition was in a particular direction. In other words, the grouping of device replies must now be done in light of transitions on the n-cube as well as in the interest of reducing the number of transistors at the CP. The logic of Spindac C was designed after all the other Spindac designs had been completed and therefore this concept was not difficult to keep in mind. If, however, a type C design was being done without the benefit of a prior type D design, this factor could be overlooked. As it turns out, the reply subsets selected for type R logic are directly usable for type C logic.

6.1.2 Design of the Sequencing Logic

Basically the sequencing logic for a type C control design would be formed using the design techniques and ideas of type D logic which are described in Sec. 4. This is a type of design method in which the branching and recombinations required by the flow chart are incorporated into the sequencing portion of the control initially. This is visualized as the mapping

of the flow chart CP's onto the nodes of an n-cube. Once this assignment has been made, the detailed design of the logic required at the CP's can be completed. The design of the logic for a type C control will proceed in exactly the same manner as is described in Sec. 4 except that the reply signals will be, in general, subset reply signals. With this exception, the descriptions of the assignment to the nodes of an n-cube and the design details at the CP's in Sec. 4.1.2 and 4.1.3 can be used directly.

6.2 Spindac C

The design of Spindac C was the easiest of the four controls designed since it amounted to copying the sequencing logic of Spindac D and then making the revisions necessary to use combined replies. Spindac C was designed using only 1,820 transistors which is almost 17 per cent less than used by Spindac P. This is more of a reduction in transistor count than was obtained with either of the other new design methods. This, in a way, is a verification of the statement that the techniques of type D and type R logic could be combined efficiently and effectively. A complete analysis of the designs will be found in Sec. 7.

6.3 Synthesis Procedure for Type C Logic

The design of control logic using type C logic methods could be accomplished in the following manner. The use of this procedure assumes that a flow chart describing the time ordering of operations for the control exists prior to the synthesis attempt.

1. Choose an n-cube that has more nodes than the number of CP's in the flow chart. The experience with Spindac has shown that the number of nodes required is of the

same order as the total number of transitions specified by the flow chart. The Spindac flow chart specifies 45 different allowable transitions and the 6-cube assignment used 40 nodes.

2. Study the flow chart and group requests and replies into subsets with three or more devices per subset. Assign a transition direction on the n-cube to each of the subsets. This direction of movement will be used whenever the devices in that particular subset are requested by the flow chart. In many situations the logic designer can influence the persons creating the flow chart and thereby obtain a more optimum flow chart from the logic design standpoint.
3. Once the reply subsets have been established, the flow chart must be modified to include any bypass requests necessary to activate all members of a subset whenever any member is active. The logic required to form the combined replies can also be designed using Fig. 19 as a guide.
4. Make an assignment of the CP's of the flow chart onto the nodes of the n-cube selected. This step will also include the introduction of additional assignments whenever needed to facilitate Gray code movement or to cause the assignments to be compatible with the transition directions designated in step 2.
5. Design the sequencing memory element register and the reply decoding logic required to form the excitation

signals for the memory elements. The minterm expressions for the pair of nodes involved in a transition will identify the sequencing memory element to be excited.

6. Repeat steps 1 through 5 as many times as needed to obtain a minimum transistor compromise between the additional assignments of step 5, the reply decoder logic, and the number of bypass requests added in step 3.

7. Complete the details of the logic required at the various CP's. This will include the CP's specified by the flow chart and those added as additional assignments in step 5. It might also include the creation of combined sequencing memory element register signals of the type described in Sec. 4.2.

7. CONCLUSIONS AND SUGGESTIONS

The general objective of these studies has been to investigate some of the problems encountered when an attempt is made to design speed-independent logic for a control. The information contained in this dissertation is the result of a carefully controlled experiment into the design of logic for an Arithmetic Control. The experiment consisted of designing four controls, called Spindacs, one for each of four design methods. Each of these controls was to perform identical sequencing operations as set forth in the Flow Chart of Fig. 1. The purpose in designing a separate control using each of the four different methods was to provide a realistic basis for comparing the design methods. The development of the design methods and the design of the Spindacs are discussed in Secs. 3 through 6 inclusive. The objective of this section of the dissertation is to compare and evaluate the Spindacs and their respective design methods. Section 7.1 presents the comparisons of the four Spindacs as well as the general evaluations and conclusions reached as a result of these investigations. The discussions in Sec. 7.2 attempt to extend or extrapolate the results of these studies into future work and to suggest possible future design ideas.

7.1 Conclusions

The motivation for these investigations into speed-independent logic for a control was the fact that the first attempt at speed-independent control design was not completely successful. This initial design was for the Arithmetic Control for the new Illinois computer. This control is called DC. The final evaluation of the logic of DC was that it could be considered speed-independent only when it was examined in the light of the unrealistic,

non-physical assumption that all identity elements used in the reply logic could operate in zero time. The reasons for making this assumption are given in Sec. 3.2.1.1.

The primary purpose in making these studies was to determine if the use of a design method which was different from the one used for DC would produce logic which could be considered speed-independent without making special assumptions. The initial conclusion drawn from these investigations is that at least two methods exist which can be used to design speed-independent logic for a control like Spindac. Both the type R and the type C logic design methods can produce speed-independent logic. However, any logic designed using either of the other two methods considered, the type P and type D design methods, could not be considered speed-independent without making the same assumption needed for the logic of DC.

The design of the Spindacs was performed within the limitations of a very carefully controlled experiment. As a result, the conclusions reached must be considered in the light of these constraints. The most important assumptions made in establishing the requirements for the Spindacs were the following:

a. The sequencing operations to be performed by each Spindac were the ones specified by the Flow Chart of Fig. 1. This included the definitions of the various symbols and the two substep operations discussed in Sec. 3.1.2.

b. These sequencing operations are associated with one particular Arithmetic Unit with fixed data paths and fixed numbers of registers, etc.

c. Each Spindac was to operate in a speed-independent manner; however, the Arithmetic Unit being controlled was not to operate in a speed-independent manner.

d. The circuit elements used were to be similar to those used in the new Illinois computer.

All of the data necessary in making a comparison of the sizes of the four Spindacs is given in Tables II and III. The first of these tables presents a detailed breakdown of where the transistors are used in the various controls. This data makes it possible to analyze each of the Spindacs as well as compare one with another. The data of Table III gives an analysis of the distribution of transistors within each Spindac. This is the same data as is presented in Table II, but it is grouped in more general classifications.

The main cause for the logic of DC to be non-speed-independent was the use of the expanding tree of logic to achieve the reply fan-out. In attempting to circumvent this difficulty, the three new design methods described in this dissertation relied upon two fundamental ideas. Basically, the first idea was this: if the reply signal distribution problem was caused by the large fan-out requirement for the reply signals, then a design method or technique which requires significantly smaller fan-out for reply signals might well permit the distribution systems to be designed in a speed-independent manner. The desired reduction in reply fan-out was accomplished in different ways by the different design methods. In the decoding system of types D and C logic, the assignment of CP's to the nodes of the n-cube is very important in determining the number of reply transistors used at the register of sequencing memory elements. The data of Table II would indicate that the assignment used for Spindacs D and C was rather efficient in that a large

TABLE II Comparison of the Spindac Designs

Spindac	P	D	R	C
Basic Sequencing Logic	454	426	432	416
Branching and Recombination Logic	268	208	252	193
Request Transistors	192	196	228	226
Conditional Request Logic	198	183	189	173
Additional Request Logic (Cable Drivers, AND, etc.)	126	136	158	158
Reply Transistors at the NAND and OR	296	217	117	80
Additional Reply Logic (Cable Drivers, NOT, etc.)	166	129	82	83
Device Logic (Gates, Selectors, etc.)	491	491	491	491
Total Transistors	2,191	1,986	1,949	1,820
Reduction--Transistors	0	205	242	371
Reduction--Per cent	0	9.4	11.1	16.9

TABLE III Distribution of Transistors in Major Sections

Spindac	P	D	R	C
Sequencing Logic	722	634	684	609
Per cent of Total	33%	32%	35%	33.5%
Request Logic	516	515	575	557
Per cent of Total	23.5%	26%	29.5%	30.5%
Reply Logic	462	346	199	163
Per cent of Total	21%	17.5%	10%	9%
Function Logic	491	491	491	491
Per cent of Total	22.5%	24.5%	25.5%	27%
Total	2,191	1,986	1,949	1,820
	100%	100%	100%	100%

percentage of the reduction in transistors was a result of this simplified reply logic. The second manner in which the reply requirements were reduced was through the use of the combined reply logic of the type R and type C methods. The use of a combined reply system divides the fan-out requirement for each reply signal in half. It also reduces the overall complexity of the control since it reduces the number of reply signals which are distributed throughout the control. Probably the main reason that the Spindac C logic required so few transistors is this reduction in reply logic. It will be noted from Table III that Spindac P used 462 reply transistors whereas Spindac C required only 163--a reduction of 299. Since the type C logic used both a decoding system and combined replies the Spindac C logic was simplified in both of these ways. Using these comments as a basis, the second conclusion drawn from these studies is that the most effective means investigated for reducing the complexity and size of the controls was the direct reduction of the reply logic.

Earlier it was mentioned that the attempts to reduce the complexity and size of the Spindacs were based upon two different ideas. The first idea was the simplification of the reply logic which was quite successful and was discussed in the previous paragraph. The second idea was basically this: if the size of the sequencing portion of control can be significantly reduced, then the total control will be smaller and less complex. Thereby the reply signal distribution problem will be simplified to the point where the reply signals can be handled in a speed-independent manner. The distribution of reply signals is still the primary concern; however, this approach was to reduce the overall size of the control without specifically attacking the reply logic. The method used in these studies to reduce the size of the sequencing control was to use a decoding system as described in Sec. 4 for the

type D logic. One of the things that is apparent in Table II is that the objective of reducing the transistor count in the sequencing portion of Spindac D was not very successful. Spindac P uses 26 sequencing memory elements whereas Spindac D uses only six. This should make a difference of 120 transistors but the table shows a difference of only 28; therefore, too many additional transistors were required in the decoding at the sequencing memory element register or for additional nodal assignments. The third conclusion derived from these investigations is that the use of a decoding method of controlling the sequencing portion of control was ineffective in reducing the size of either Spindac D or C.

The two design methods referred to in the first conclusion are the type R design method and the type C design method. The type R design method, described in Sec. 5, is very similar to the design method used in the design of the Arithmetic Control for the new Illinois computer whereas an entirely different design technique is used in the type C design method which is discussed in Sec. 6. The results of this study indicate that a design using either of these methods would produce a smaller control than one designed using either of the other two design methods studied. The type R method is based on the building-block technique where a control design is formed by properly interconnecting sections of standard logic. The type C method is based on a design technique where the total sequencing problem is represented by Gray Code movements on an n-cube and these sequencing movements are determined by the mapping of flow chart requirements onto the n-cube. Surprisingly, the controls designed using these very different methods were only slightly different in size--the type C design was six per cent smaller than the type R control. The fourth conclusion is, therefore, that the design method used had only a small effect on the control size.

The basic form of the decoding method used in types D and C logic was the essence of simplicity and straightforwardness, whereas the basic forms of the standard logics used in the building-block method of types P and R logic appear to use an excessive number of sequencing memory elements. The results of these studies indicate that the utter simplicity of this decoding method is apparently lost when a complex set of interconnected branchings is imposed upon the design. On the other hand it would appear that the ease with which the standard logics of the building-block method are interconnected justifies the apparent complexity of the basic blocks.

The final conclusion reached concerning the Spindac designs was that any future attempts at speed-independent control design should carefully consider using arithmetic methods which are different from those used in these studies. The following two techniques are the most important reasons for making this statement: first, quite often the arithmetic results of one control step are used to control some aspect of the very next step. For example, in one step the exponent of the operand is gated into a register and in the very next step the numerical value of the exponent in that register is used to determine the type of arithmetic operation to be performed. The finite time required to do all of the operations necessary to obtain the numerical value of the exponent in control for use as a conditional signal places a fixed, maximum speed on arithmetic operations. The second point was revealed by the foregoing discussion of arithmetic. The discussion assumed that a speed-independent control logic is controlling a non-speed-independent arithmetic unit. But a signal from this non-speed-independent logic is being used as a conditional signal in the control. It is the opinion of the writer that the method of handling these signals is not satisfactory and certainly an alternative method of accomplishing the conditional arithmetic is possible.

7.2 Suggestions for Future Work

The design of the Spindacs was a controlled experiment in logic design. The Spindac designs were based upon several rather rigid, previously discussed assumptions. The comments in this section will attempt to extend the results of the Spindac design experiment into situations where one or more of these assumptions is relaxed. Since it is foolish to expect that future attempts at speed-independent control design will be made within the confines of the assumptions used for the Spindacs, these extensions are made with the idea of assisting future studies into speed-independent logic for a control.

Time and again in the process of performing these designs it has become apparent that the smaller the control logic, the easier it would be to obtain a speed-independent design. Therefore, it seems reasonable to recommend that any future control design seriously investigate the possibility of using a control with only two, three or four CP's. This would be a major departure from the concepts used in the Spindac designs. In each Spindac design a given set of operations on the flow chart is associated with specific elements of sequencing logic. Both the set of operations on the flow chart and the logic elements in the control are referred to by the same CP designation number. In this proposed system, each CP would perform many different sets of flow chart operations. This would be accomplished by having each CP contain many conditional operations and much conditional bypass logic. The decision as to which operation would be performed could be made by a large speed-independent decoder. In a system such as this, the reply problem would be reduced to the absolute minimum and the formation and use of conditional signals would become of most importance. It is not felt that this would be a gamble since in the Spindac designs the collapsing trees of logic associated with the formation of requests were found to be speed-independent. The expanding trees of reply logic which

were found to be non-speed-independent would be eliminated. Such a system would require the use of one or more counters and it might also require the use of sequencing operations not used in any of the Spindacs. It is conceivable that logic might be required which would permit the operations of two CP's to be performed in parallel for one instruction and then sequentially for another instruction.

The second suggestion is prompted by the expectation that future computers will operate faster and will be larger in size. It is possible that in such an environment, even the newly developed type R and type C design methods might not be adequate. One of the basic requirements of a speed-independent circuit is that every change in state of every node is important to subsequent operations. In most cases this implies that a decision elements' output is used as an input to several other decision elements. Since the mathematical model of a speed-independent circuit does not permit delays along signal wires (see Sec. 1.3), a practical circuit design can meet this mathematical requirement in either of two ways. When the several uses of the signal are physically close together and close to the signal output, the circuit meets the requirements directly. When this condition is not met, an equivalent delay element must be placed in the logical expression for the circuit. It is felt that the advent of subnanosecond circuitry will require that these equivalent delays be introduced into many signal lines. In many places within the controls designed for this study, such as request signals, the inclusion of a delay in the line would not have affected the speed-independent evaluation of the circuits. However, there are also some situations, such as signal distribution systems, where these delays might cause the loss of speed-independence.

In every case where a signal is transmitted along a wire, there is a finite time difference between the transmitted and received signal no matter

how short the wire. It is the responsibility of the designer to determine when the time delay along wires is significant from the standpoint of a speed-independent evaluation of the circuit. In the case of signal distribution systems, the time delay from source to user is not as important, in the speed-independent sense, as the difference in time delay along two wires transmitting the same signal. It is suggested that one possible method of dealing with this situation is to use many small subcontrols distributed throughout the Arithmetic Units being controlled. These subcontrols will be so designed that they control only the portion of the Arithmetic Unit in which they are found. In all cases where a large signal fan-out is required, all of the users of the signal will be physically located in one place. Thus there will be an insignificant difference in time delay between the various users of the signal, and the need to include an equivalent time delay for the output line will not affect the speed-independent nature of the circuit.

This form of control would be a major departure from the concepts used in these investigations. In all four of these studies it was common practice for all parts of the control to be in communication with all other parts of control and all parts of the Arithmetic Units being controlled. If every circuit element in Spindac R, for example, had had a subnanosecond operation time, then many of these equivalent delays would have been required in the logic expressions for the circuits and in all probability the result would have been a non-speed-independent control. It is the intent of this suggested distributed control to curtail greatly or eliminate this policy of "everything communicating with everything else" so that speed-independent operation might be retained.

It seems possible that this idea of a distributed control might also be consistent with the idea of attaining higher computing speeds. By creating many specialized Arithmetic Units or parts of Arithmetic Units, it should be

possible to maximize parallel operation of computer subsections. The more of these subsections which can operate concurrently, the shorter will be the overall computation time. The suggestion that a distributed control be considered might also be consistent with the trend toward microminiaturization. If the subcontrol were built into the same module with the Arithmetic Sub-unit being controlled, the signal distribution problem within that module would be simplified.

During the design of the Spindacs and during the development of the decoding system for the type D logic, it became apparent that the logic for a flow chart with a small number of CP's is more easily designed than the logic for a large one. Also an efficient assignment of the CP's of the flow chart is easier with a small number of CP's. A consideration of these facts leads to the third suggestion which is that the flow chart should be carefully studied before a new design is attempted. This should include a study of the two aspects of the problem mentioned here as well as others.

The first aspect of the flow chart that should be studied is the possibility of combining CP's as is discussed in Sec. 4.1.2. When two CP's for different instructions request similar operations, it is possible that these operations could be performed by one CP with properly chosen conditional operations. A careful investigation should be made of the interrelationship between the sequencing logic eliminated and the conditional logic added when such a combination of CP's is attempted. This type of study has meaning only when the control design will be accomplished by realizing each flow chart CP as control logic as was done for all four Spindacs. If either of the previous suggestions were being followed in the control design, this suggestion might not have meaning.

The second aspect of the flow chart that should be carefully studied is the meanings of the various symbols used. On the flow chart used for these studies, a symbol such as gA was interpreted to mean that the gate into the A register was to be opened and then subsequently closed. If the two symbols gA, gQ were shown together, this meant that both the A gate and the Q gate were to be opened, the control would wait until both were open at the same time, and then they both would be closed. That is to say, these two gating operations are cross-coupled in the change chart sense. It is not known whether a deviation from this type of operation will produce any savings in control logic, but it is known that there are many places on both the Spindac and the DC flow charts where operations are cross-coupled when there is no arithmetic reason for so doing.

Within the class of speed-independent circuits there are four subclasses. Throughout these discussions the term speed-independent circuits has been used to mean this entire class of circuits. This includes all circuits which are semi-modular, distributive, or totally sequential, as well as those which are just speed-independent. As was mentioned in Sec. 1.4, the verification that a given logic diagram was a speed-independent circuit was taken from the Illiac computer using the Q 5 Circuit Analyzer Routine. This program can identify semi-modular, distributive, and totally sequential circuits; however, it cannot identify a circuit which is speed-independent but not semi-modular. Circuits which are speed-independent but not semi-modular certainly do exist and it is possible that realizations using such circuits might be less complex or more flexible than the ones studied. A recommendation that a new computer program be written that will be capable of identifying all four of the subclasses within speed-independent circuits is then made as the fourth suggestion.

As has been mentioned earlier, the smaller the control the easier it is to make the reply systems speed-independent. The fifth suggestion is that careful consideration be given to a computer design in which the control is made very small and the Arithmetic Unit is allowed to grow larger to retain the ability to perform arithmetic. One method which could be used to accomplish this would be to use a large matrix for multiplication so that a multiply instruction could be performed in one or two steps. This same concept could be used for addition and for normalization. In such a system the control would be so small that there would be no possibility of the control time being significant in comparison to the time required for the arithmetic operation being performed. As a result, the time for a given computation would be set by the Arithmetic Units entirely and the computer speed would be increased since each of the important operations would be accomplished in a specially designed matrix.

In the evaluation of the design of DC and in the conclusions drawn from the Spindac designs, it has been pointed out that a few different methods of doing arithmetic are required. One of the problems that is not satisfactorily solved is the method of handling conditional signals. The sixth suggestion is that many of these problems might be solved by designing the Arithmetic Unit as speed-independent logic as well as the control. There are several places within DC where a signal delay has been placed in the logic in order that the arithmetic operations will be completed before a subsequent control step is taken. If the Arithmetic Unit were designed with speed-independent logic, this would not have to be done. In addition to this advantage, the present practice of using worst-case timing studies to justify the use of conditional signals which originate in the non-speed-independent Arithmetic Unit would not be needed.

The last suggestion to be made is that the difference between speed-independent operation of logic and the reliability of logic be carefully studied. As was pointed out in Sec. 1.1, the use of a logical configuration that is proven to operate as free of malfunction as possible is only one part of reliability. If a malfunction occurs in a logical AND circuit such that the input is held on a "1", the output of the AND circuit will change from "1" to "0" as the other inputs to the AND vary. Such a malfunction renders that specific input invalid and if this circuit were in speed-independent logic, the circuit would operate as though that particular input to the AND had not been connected in the original design. A similar comment can be made about an input to an OR which fails to a "0". Many component failures within the decision elements of a speed-independent circuit will be quickly found due to the speed-independent operation of the logic; however, there are some component failures such as these which will not be detected.

Finally, there are several other suggestions that appear interesting, but which have not been developed to the point where their advantages can be specified. It seems possible that the preoccupation of computer designers with high-speed operation might have caused some worthwhile ideas to be undeveloped. Space-probe computers or other applications where speed is of less importance could very possibly use techniques which have been rejected in these investigations. It also seems possible that some adaptation or restatement of speed-independence theory might be made applicable to synchronous circuits. The author hopes that these studies, the conclusions, and suggestions have been worthwhile in contributing new and useful information about the methods of designing speed-independent logic for a control. In a computer where some very different technique is used, such as one of these suggestions, many different conclusions might have been reached. Despite this possibility, it is hoped that these studies will aid in any future investigations.

BIBLIOGRAPHY

BA-1 Bartky, W. Scott, "A Theory of Asynchronous Circuit III," University of Illinois, Digital Computer Laboratory Report No. 96 (January 1960).

CA-1 Caldwell, Samuel H., Switching Circuits and Logical Design, New York, John Wiley and Sons, Inc., 1958.

DC-1 "On the Design of a Very High-Speed Computer," University of Illinois, Digital Computer Laboratory Report No. 80 (October 1957).

DC-2 Technical Progress Report for July 1960, University of Illinois, Digital Computer Laboratory.

DC-3 "Complete Circuit Analyzer," Illiac Auxiliary Library Routine Q5-237, University of Illinois, Digital Computer Laboratory (July 1957).

GI-1 Gillies, Donald B., "A Flow Chart Notation for the Description of a Speed-Independent Control," Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design, Detroit, Michigan (October 1961) AIEE Publication S134.

GI-2 Gillies, Donald B., "A Description of DC in Terms of Its Flow Chart," University of Illinois, Digital Computer Laboratory File No. 397 (August 1961).

GI-3 Gillies, Donald B., "The Design of a Very High-Speed Scientific Computer," University of Illinois, Digital Computer Laboratory File No. 376, Presented at sessions on the Theory of Computing Machine Design at the University of Michigan (June 1961).

HU-1 Huffman, D. A., "The Synthesis of Sequential Switching Circuits," Journal of the Franklin Institute, Vol. 257, Nos. 3 and 4 (March and April 1954.

HU-2 Huffman, D. A., "The Design and Use of Hazard-Free Switching Networks," Journal of the Association for Computing Machinery, Vol. 4 (March 1957), Initially presented at a meeting of the ACM in September 1955.

IR-1 IRE Dictionary of Electronic Terms and Symbols, New York, The Institute of Radio Engineers, Inc., 1961.

MI-1 Miller, Raymond E., "An Introduction to Speed-Independent Circuit Theory," Proceedings of the Second Annual Symposium of Switching Circuit Theory and Logical Design, Detroit, Michigan (October 1961) AIEE Publication S134.

MU-1 Muller, David E., "Theory of Asynchronous Circuits," University of Illinois, Digital Computer Laboratory Report No. 66 (December 1955)

MU-2 Muller, David E. and Bartky, W. Scott, "A Theory of Asynchronous Circuits I," University of Illinois, Digital Computer Laboratory Report No. 75 (November 1956).

MU-3 Muller, David E. and Bartky, W. Scott, "A Theory of Asynchronous Circuits II," University of Illinois, Digital Computer Laboratory Report No. 78 (March 1957).

MU-4 Muller, David E. and Bartky, W. Scott, "A Theory of Asynchronous Circuits," Proceedings of an International Symposium on the Theory of Switching, Harvard University, April 1957, Annals No. 29 of the Computation Laboratory of Harvard University, Cambridge, Massachusetts, Harvard University Press, 1959.

MU-5 Muller, David E. and Bartky, W. Scott, "Sequencing of Operations in a Computer," University of Illinois, Digital Computer Laboratory File No. 225 (July 1957).

MU-6 Muller, David E., "Asynchronous Switching Theory," University of Illinois, Digital Computer Laboratory File No. 243, Presented at Sessions on the Advanced Theory of Logical Design of Digital Computers at the University of Michigan (June 1958).

NE-1 Nelson, James C., "Speed-Independent Counting Circuits," University of Illinois, Digital Computer Laboratory Report No. 71 (July 1956).

RO-1 Robertson, James E., "Theory of Computer Arithmetic Employed in the Design of the New Computer at the University of Illinois," University of Illinois, Digital Computer Laboratory File No. 319, Presented at Sessions on the Theory of Computing Machine Design at the University of Michigan (June 1960).

RO-2 Robertson, James E., "Problems in the Physical Realization of Speed-Independent Circuits," Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design, Detroit, Michigan (October 1961) AIEE Publication S134.

SH-1 Shelly, James H., "Design of Speed-Independent Circuits," University of Illinois, Digital Computer Laboratory File No. 226 (July 1957).

SH-2 Shelly, James H., "The Application of Change Chart Theory to Control Design Problems," University of Illinois, Digital Computer Laboratory File No. 238 (April 1958).

SH-3 Shelly, James H., "The Decision and Synthesis Problems in Semi-Modular Switching Theory," University of Illinois, Digital Computer Laboratory Report No. 88 (May 1959).

SW-1 Swartwout, Robert E., "One Method for Designing Speed-Independent Logic for a Control," Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design, Detroit, Michigan (October 1961) AIEE Publication S134.

WA-1 Wallace, C. S., "Interplay," University of Illinois, Digital Computer Laboratory File No. 446 (March 1962).

APPENDIX A

A PRELIMINARY COMPARISON

In general, the information presented in the previous sections of this dissertation has been discussed in the chronological order in which the investigations took place. There was one important deviation from this practice. Time was spend in developing a previously unmentioned preliminary comparison of the four design methods. After the design method for type D logic was developed, and before the logic of Spindac D was designed, the type R and type C design methods were also developed. Once the four design methods had been developed, a preliminary comparison could be made of the logic required by each in realizing a straightforward design problem. The purpose of this comparison was to determine if these design methods appeared promising enough to warrant going on to develop the thorough comparison afforded by designing the Spindacs. The development of the design methods had shown that these new design ideas could be developed into methods for designing speed-independent control logic. However, the contention that each new design method would produce a design using fewer transistors than a type P design had not been verified. Therefore, the following problem was presented to each of the design methods: design three logics which realize the sequencing of a set of operations in a loop, one logic to have four CP's in the loop, one with eight CP's and the last with 16 CP's in the loop. Each CP of each logic is to activate two gates in parallel, but all gates will be energized only once per logic (no multiple operation). As far as the type R and type C logics were concerned, each pair of gates was considered to form a subset of devices whose replies could be combined. The type P logical realizations used for the

comparison were the same as the logic shown at CP 12 and CP 13 of BLD 1. The type D realizations used logic similar to that which is shown at CP 11 of Fig. 15. The type R logic was similar to the type P logic except for the use of subset replies, and the type C was similar to the type D except for subset replies. The results of this preliminary comparison are given in Table IV.

The obvious evaluation of Table IV was that each of the three new design methods was able to realize the designated operations with fewer transistors than required by type P logic. Therefore, even though Table IV showed a minimum reduction of only ten per cent, it was decided that the results of the preliminary comparison justified making a complete comparison of these methods through the design of the Spindacs. Further justification for this decision came from the fact that the preliminary comparison included only the logic required by the CP's and by the sequencing portions of control. Each of the three new design methods could theoretically reduce the transistor requirements for some portions of device logic, or of branching logic or in other ways not indicated by this preliminary comparison.

Following the completion of the Spindac comparisons in Tables II and III, the transistor counts were regrouped as shown in Table V to serve as an evaluation of the preliminary comparison. The data of Table V shows that the preliminary comparison was based on less than half of the transistors required by a complete control design. However, the transistor reductions for the Spindacs were very near to those predicted for a 16-step loop. The obvious conclusion to be drawn from this fact is that some of the logic not apparent in Table IV was also reduced by the new design methods. In fact, since the device or function logic categories shown on Tables II and III were not affected by the change in design methods, it is surprising that the preliminary and final comparisons were so close.

As indicated earlier, when the preliminary comparison was made, it was known that a complete control design would include many transistors which were not apparent in Table IV. It was also known that the logic for these uncounted transistors could be grouped into two general classifications: that logic which would be affected by the design method and that logic which would be unaffected by the design method. The comparison of the Spindac designs has shown that for each of the three new design methods discussed in this dissertation, less than 27 per cent of the logic was in this second classification which is not affected by the design method. Despite the fact that the percentage reduction figures were somewhat over-optimistic, it is felt that the preliminary comparison gave a reasonable prediction of the capabilities of the design methods. It is interesting to note that these studies could have been terminated after the preliminary comparison if this "hidden" logic had not existed.

TABLE IV Preliminary Comparison of the Design Methods

Type of Logic	P	D	R	C
4-Step Loop				
Transistors Required	164	48	56	40
Reduction in Transistors	0	16	8	24
Reduction in Per cent	0	25	13	38
8-Step Loop				
Transistors Required	128	110	112	94
Reduction in Transistors	0	18	16	34
Reduction in Per cent	0	14	13	27
16-Step Loop				
Transistors Required	256	230	224	198
Reduction in Transistors	0	26	32	58
Reduction in Per cent	0	10	13	23

TABLE V An Analysis of the Validity of Table IV

Spindac	P	D	R	C
Transistors counted in Table II which were apparent in Table IV	942	839	777	722
Per cent of Total	43%	42%	40%	40%
Transistors counted in Table II which were not apparent in Table IV	1,249	1,147	1,272	1,098
Per cent of Total	57%	58%	60%	60%

APPENDIX B

AN INDEX OF ABBREVIATIONS

The abbreviations which were used:	The meaning of the abbreviations:	A description is found in Section:
A 1, A 2, A 3, A 4, A 5	Control points for the floating-point addition subsequence of Spindac	2.1 and Fig. 1
AC	The Advanced Control for the new Illinois computer	1.2
AND	A decision element whose output function is: f = (abc...n)	3.1.2b
AQ Subset	A set of combined replies	5.1.1
AQA	A reply signal from the AQ subset	5.1.2 and Fig. 20
AQO	A reply signal from the AQ subset	5.1.2 and Fig. 20
Asynchronous circuit	An interconnection of decision elements	1.3
B 1, B 2	Control points for the clear addition subsequence of Spindac	2.1 and Fig. 1
BLD	Basic Logic Diagrams	3.1.1 and 3.1.2
C_1, C_2	Conditional signals	3.1.2, BLD 6
"C" Element	A memory element used in control design	3.1.1
CB	A memory element	3.1.2, BLD 6
CD, $\overline{CD}$	Noninverting and inverting cable driving amplifiers	3.1.2b
CP	Control point	3.1.2a
D 1	The instruction decode control point for Spindac	2.1 and Fig. 1
DC	The Arithmetic Control for the new Illinois computer	1.2 and 3
Decision element	Any logic element	1.3

The abbreviations which were used:	The meaning of the abbreviations:	A description is found in Section:
dl, $\overline{\text{dl}}$	Conditional signals	3.1.2, BLD 2
E 1, E 2	Control points for the exponent arithmetic subsequence of Spindac	2.1 and Fig. 1
ED Subset	A set of combined replies	5.1.1
EDA	A reply signal from the ED subset	5.1.2
EDO	A reply signal from the ED subset	5.1.2
EX	The exit memory element for a scale of two loop	3.1.2, BLD 8a
FOgBS	A register gate	3.1.2, BLD 10
gA, gA', gB, gB', gE, gES, etc., XXgXX	These symbols are used to designate register gates	2.1 and Fig. 1 3.1.2
gAr, gESr, etc., XXgXXr	A reply signal from the register gate logic	2.1 and 3.1.2
K 1, K 2	Control points for the correct overflow and detect zero subsequence of Spindac	2.1 and Fig. 1
M 1, M 2, M 3, M 4, M 5, M 6, M 7, M 8	Control points for the multiplication subsequence of Spindac	2.1 and Fig. 1
N 1, N 2, N 3	Control points for the normalization subsequence of Spindac	2.1 and Fig. 1
NAND	A decision element whose output function is: $f = \overline{(abc...n)}$	3.1.2b
n-cube	An n-dimensional cube	4 and Fig. 16
OR	A decision element whose output function is: $f = (a \vee b \vee c \vee ... \vee n)$	3.1.2b
OUT	A buffer register between the Arithmetic Unit and Core Storage of Spindac	3.1.2, BLD 10
OUTFL	A memory element used in the interconnection of two controls	3.1.2, BLD 10

The abbreviations which were used:	The meaning of the abbreviations:	A description is found in Section:
OWC	OUT Write Control of Spindac	3.1.2, BLD 10
OZ Subset	A set of combined replies	5.1.1
OZA	A reply signal from the OZ subset	5.1.2 and Fig. 19
OZO	A reply signal from the OZ subset	5.1.2 and Fig. 19
RO	A memory element	3.1.2, BLD 6
S 1, S 2, S 3, S 4, S 5	Control points for the store subsequence of Spindac	2.1 and Fig. 1
s, $\bar{s}$	Conditional signals	3.1.2, BLD 8b
sE, sD, etc., XXsXX	These symbols are used to designate selector control mechanisms	2.1 and Fig. 1 3.1.2, BLD 7
sEr, sDr, etc., XXsXXr	The reply signal from the appropriate selector control logic	3.1.2, BLD 7
Spindac	A fictitious speed-independent arithmetic control	2 and Fig. 1
Spindac C	The Spindac designed using type C logic	6.2
Spindac D	The Spindac designed using type D logic	4.2
Spindac P	The Spindac designed using type P logic	3.3
Spindac R	The Spindac designed using type R logic	5.2
SR Subset	A set of combined replies	5.1.1
SRA	A reply signal from the SR subset	5.1.2
SRO	A reply signal from the SR subset	5.1.2
Type C logic	Logic designed using a combination of the type D and type R design methods	6
Type D logic	Logic designed using a system design method and the assignment of flow chart CP's to the nodes of an n-cube	4

The abbreviations which were used:	The meaning of the abbreviations:	A description is found in Section:
Type P logic	Logic designed using a building-block design method. This is the type of logic used in DC	3
Type R logic	Logic designed using a building-block design method of type P logic but modified to permit use of combined replies	5
α, $\bar{\alpha}$	Conditional signals	3.1.2, BLD 4 and 5
β, θ, Ω	Memory elements whose outputs are used as conditional signals	3.1.2, BLD 4 and 5 2.1 and Fig. 1
ΩL Subset	A set of combined replies	5.1.1
ΩLA	A reply signal from the ΩL subset	5.1.2
ΩLO	A reply signal from the ΩL subset	5.1.2
	This symbol is the logic equivalent of the selector driver mechanism	3.1.2b
	This symbol is the logic equivalent of the register gate mechanism	3.1.2b

BOUND BY
HNM

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R v.1 C002 v.111-130(1961)
Some memory elements used in ILLIAC II /
3 0112 088404147