ENGINEERING RESEARCH INSTITUTE
UNIVERSITY OF MICHIGAN
ANN ARBOR

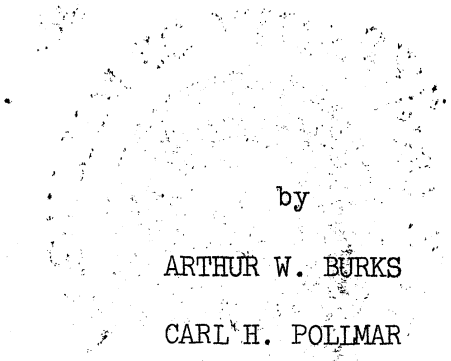
LANGUAGE CONVERSION FOR DIGITAL COMPUTERS

General Introduction

and

Volume I

The Logical Realization of Transliterative Functions



by

ARTHUR W. BURKS

CARL H. POLIMAR

DON W. WARREN

JESSE B. WRIGHT

Project M828

BURROUGHS ADDING MACHINE COMPANY
DETROIT, MICHIGAN

1 June 1952

engn

UMR0940

ABSTRACT

The aim of this report in general is to present some theories concerning code conversion and the design of code conversion equipment for digital computers.

Volume I deals with a restricted type of character-to-character conversion represented by the Transliterative Function. Such conversions do not require remembering previous states nor any time-space transformations. Some theory and a number of effective techniques are given for designing the required equipment—such design being presented at a level of considerable logical abstraction.

The remainder of the report, to be published subsequently, will cover the following topics:

Volume II will discuss the use of various types of equipment—tubes, relays, etc.—as means (1) for carrying out the logical designs of Volume I, and (2) for effecting more general code conversions and format conversions—e.g., those involving serial-parallel transformations, shift codes, etc.

Volume III will present detailed proofs and some relevant discussion of two topics presented in Volume I, concerning (1) minimal networks—in particular, the Balanced MS Net, and (2) the theory behind and validity of the technique for folding trees.

TABLE OF CONTENTS

LANGUAGE CONVERSION FOR DIGITAL COMPUTERS

	Page
ABSTRACT	ii
GENERAL INTRODUCTION	iv
VOLUME I. THE LOGICAL REALIZATION OF TRANSLITERATIVE FUNCTIONS	1
Section 1. Introduction	1
Section 2. Definition of Logical Elements and Networks	3
Section 3. Logical Design of Networks	9
3.1. Conjunctive Switches: MS (Multiplicative Switches) and MS Nets	9
3.2. Alternative Switches Using Other Logical Elements	22
Section 4. Techniques for Realizing Transliterative Functions	25
4.1. Decoding Functions	27
4.2. Encoding Functions	30
4.3. Arbitrary Transliterative Functions	31
4.4. Modifications to Provide Pulse Output	36
Section 5. Characteristics of Function Switches	39
5.1. Element Input Count and General Loading Properties	39
5.2. The Folded Tree Technique	43
Bibliography	53
Index of Terms	54

(To be published subsequently)

VOLUME II. THE PHYSICAL REALIZATION OF CODE AND FORMAT CONVERSIONS

VOLUME III. MINIMAL SWITCH THEORY AND THE FOLDED TREE

LANGUAGE CONVERSION FOR DIGITAL COMPUTERS

General Introduction

It is the purpose of this report to make a general survey of code and format conversion. This will not be a complete survey of the problem, but only a preliminary investigation which is designed to make clear the general nature of the problem, to formulate some of the chief available techniques, and perhaps to serve as a foundation for more detailed studies of such conversions and the design of equipment therefor.

Code conversion is a special case of language translation, and it is useful to consider it in this light. While no sharp lines can be drawn between what is normally called a code conversion and an ordinary linguistic translation (e.g., German to English), the difference can be indicated roughly. Basically, a code conversion is a translation at the character level, while an ordinary linguistic translation takes place at the sentence level. Later in this section the concept of message transliteration will be defined; put in terms of this concept, the difference is that code conversions are primarily message transliterations, while ordinary language translations are not.

In accounting work the initial data and end result are expressed in a natural language whose alphabet (in this part of the world) consists of Latin letters, Arabic numerals, and a few auxiliary characters such as the dollar sign. Since electronic and electro-mechanical circuits generally work most easily in a binary system, most machine languages have two basic characters: represented in this report by '0', '1'. Thus, an adequate machine language is an artificial language with two basic characters (bits), which characteristically represents alphanumeric characters by relatively short sequences of bits so chosen that the translation between the natural language and code (and hence between codes) is essentially a message transliteration.

It should be noted that many machine languages are not quite binary but are almost so. Consider for example a representation of data on a magnetic tape which is six channels wide. On such a tape there is a natural grouping of bits into sequences of six; the edges of the tape constitute, so to speak, parentheses. This grouping provides information in the technical sense of that term, since a machine when turned on in the middle of a message finds, not a continuous sequence of 0's and 1's , but

rather a sequence of clearly defined sequences each of which consists of six 0's and/or 1's . The same point holds with regard to the usual organization of characters into "words" inside a machine.

Thus a typical machine language has two primary basic characters (0 and 1 or their equivalents) and sometimes secondary basic characters—e.g.,) or its equivalent . As a consequence it is convenient in discussing code conversion to apply a notion of character broader than that of basic character. For example, in discussing the tape language mentioned above it is often convenient to regard the characters not as 1's and 0's but as ordered sextuples of 1's and 0's . For the purposes of this report and from a syntactical point of view, a language L consists of character-type* class, C , and formation-rule class, F , determining a set of message types {M} , each message being a finite sequence of character-tokens of class C : $m_1, m_2, \ldots, m_n$.

Though code conversion is largely a matter of syntax rather than of semantics, it should not be overlooked that a language has a semantic (meaning) dimension, as well as a syntactic (formal) one. The concluding statement of the previous paragraph does not constitute an entirely adequate definition of 'language' because, among other things, it ignores this dimension. Thus the statement in question does not establish the proper identity (or diversity) criteria for codes. For example, there are two teletype codes, each comprised of the same character-type class (thirty-two five-bit sequences) and the same formation-rule class (i.e., the same sequences of characters are allowable messages in both), which nevertheless are different because different five-bit sequences (01000 and 00101) are employed to represent a '#' .

Not all the syntax of a machine language is represented by the use of basic characters other than 0 and 1 . Often, perhaps usually,

---

* As used herein, the terms 'character-type' and 'character-token' are synonymous with the terms 'character-design' and 'character-event', respectively. Since there is no standard terminology on this point, either set of terms seems permissible although 'type' and 'token' would appear to be preferable to 'design' and 'event' for the following reasons. First, the former terminology is older (cf. The Collected Papers of Charles Sanders Peirce, vol. IV, paragraph 537) and an existing terminology should not be changed without due cause. Second, 'type' and 'token' may be used alone as abbreviations for 'expression-type' and 'token-type', whereas 'event' and 'design' are somewhat misleading. Third, it is often desirable in semantic work to take as a token, not a single event (e.g., a single reading of 'red'), but a substantive or sequence of events (e.g., one occurrence on a given page of a specific token of a book throughout the life of that book).

syntax is represented by sequences of bits in the same way that alphanumeric characters are represented. Thus, spaces, paragraphs, indentations, and various kinds of brackets that occur in a message written in a natural language may be coded in the same way that alphanumeric characters are coded. There is an interesting consequence of this: namely, a coded "message" can contain two or more syntactical structures. Consider as an example a message written in a natural language which is organized into unequal-length words, sentences, paragraphs, and pages. This message can be translated into a machine language which is organized syntactically into equal-length words and blocks in such a way as to preserve the syntactical structure of the original message, in the sense that the original message can be recovered by translating back into the natural language. Thus the coded machine representation of the original message contains both the original syntactical structure and the syntactical structure of the machine language.*

The concept of translation can now be applied to languages whose characters are finite sequences of two basic characters. Consider two languages, L and L' , consisting of character-type classes, C and C' , and formation-rule classes, F and F' , determining sets of message-types {M} and {M'} . Any function, ϕ , which maps {M} into {M'} is called a translation function. Such a function is called reversible if and only if

$$M_1 \neq M_2 \;.\supset.\; \phi(M_1) \neq \phi(M_2) \;.$$

If ϕ is a reversible translation function from L to L' , no information is lost in translating by means of it; that is, any message M_1 may be completely recovered from its translation $\phi(M_1)$.

In this report we are interested only in translation functions which are reversible or almost reversible.** Furthermore, we are not interested in all such functions. Thus translation functions such that for long messages M any character of M' requires for its determination simultaneous storage of all characters of M , or even such that the first character m'_1 cannot be produced until all of M is scanned, are not very useful from the point of view of mechanized accounting. However, we know of no sharp lines dividing the translation functions which are useful in mechanized accounting from those that are not. The most that can be said is that the interesting translation functions belong either to the special

---

* See for example the internal language of the Michigan report, The Languages for an Electronic Accounting System, Sec. 2.3.1 and 2.3.2.

** For example, the translation discussed in the report previously cited is not completely reversible. E.g., 37.00 and 37 would both translate into the external language as ... 037.000 .

class of message transliteration functions (to be defined below) or strongly resemble members of this class.

It is now possible to define the concept of a transliterative function: a single-valued function whose domain and range are sets of finite sequences of 0's and 1's —each set (domain and range independently) consisting of sequences of equal length. (Later this definition will be restricted to exclude non-information-carrying bits—see section 4.)

Consider now a translation function ϕ from L to L' which is explicitly defined as follows:

$$\phi(m_1,m_2,\ldots) = \Psi(m_1), \Psi(m_2), \ldots ,$$

where Ψ is a transliterative function and $m_1,m_2,\ldots$ is an arbitrary message of L (the individual m_i being character-tokens of class C of L). Any function, ϕ , so defined is a message transliteration function.

So far, the format arrangement of a message has not been discussed. Indeed it will be noted that our concept of translation function has no place for changes in format, since any two messages which differ only in format arrangement will be related to one another by the identity translation function. However, it may be assumed for the present that the messages are transmitted a character at a time, bits of a sequence representing a character—an m_i —all being available at one time (compare the definition of a language given earlier). Under these circumstances a message transliteration could be most simply executed as a character-by-character translation. For this purpose the following kinds of equipment would be required: (1) a function switch to realize Ψ , (2) memory for one character, (3) whatever memory is needed for buffering purposes (i.e., velocity matching), and (4) a few fairly simple control circuits. Since we are interested in code conversions that are message transliterations, or nearly so, it is clear that a function switch is an essential part of all code converters discussed in this report. But function switches have other uses than in code conversion, so it seems desirable to study function switches per se. This will be done in Volume I.

Function switches may be realized (constructed) with various circuit elements: relays, vacuum tubes, mechanical lever arrangements, etc. Not all such elements are equally flexible, however. For example, crystal rectifier "and's" and "or's" cannot be combined as freely as can, say, d-c vacuum-tube "and's" and "or's". On this account it is useful to think of the designing of a function switch as progressing through three stages. The first is the construction of an abstract logical design based on logical operation. The second is the translation of this design into one in terms of elements which are idealized representatives of equipment. The

theory developed by Shannon (6, 3, 4, 12), Aiken (11), and Brown and Rochester (8) is essentially at this second level, for to a large extent they base their work on relays, tubes, and diodes, respectively. The third stage is the construction of the complete circuit design with all details indicated that are needed for actually assembling the physical switch. The last of these three stages is beyond the scope of the present report. The first two stages will be treated in Volumes I and II, respectively.

In code conversions which are not message transliterations, memory circuits often play a more dominant role. For example, if the incoming message is expressed in a shift code, the last shift character to have occurred must be remembered. Again, in converting from unequal-length to equal-length numbers an incoming number must be scanned (and hence remembered) before it can be positioned properly and before dummy characters can be added. It should be noted that while such a converter realizes a non-trivial translation function (one in which it is not the case that $\phi(M) = M$) it requires only a rudimentary function switch, so that the circuit emphasis is upon memory. For this reason it is doubtful that it should be called a "code converter" at all. It is perhaps more properly called a "format converter".

There are other conversions which doubtless should be called format conversions and not code conversions. All merely space-time conversions fall in this class: e.g., a "serial-serial to parallel-parallel conversion". The translation function governing such a conversion is completely trivial, since for all messages, M , $\phi(M) = M$ (at least if the ordinary concept of character is maintained), and it is for this reason that such conversions are not properly called code conversions. Nevertheless, any study of code conversion should be extended to include these format conversions for two reasons. First, such conversions involve the same kinds of equipment as are required for code conversion. Thus in a "serial-serial, parallel-parallel" conversion, where the instantaneous rate of flow of information out of a converter is not always equal to the input rate of flow, memory for buffering is required as well as distributing circuits for rearrangement of the data. Second, in actual practice such conversions are often combined with bona fide code conversions.*

For these reasons it is convenient to make a rough-and-ready classification of conversion into three kinds, depending on the characteristics of the equipment required:

> Code Conversions—those requiring primarily function switches; e.g., conversions between six-bit codes, four-eight codes, and five-bit shift codes.

---

* See for example The Design of the Languages for an Electronic Accounting System, Sec. 2.3.2 and 2.3.3; and The Program Translations: Tape Language to Internal Language.

Format Conversions—those requiring primarily memory units and distributing and arranging equipment; e.g., space-time conversions and simple equal-length—unequal-length conversions.

Code-and-format Conversions—those requiring a substantial amount of both function switch equipment and memory and distributing equipment; e.g., a conversion from a serial-parallel, unequal-length-word, five-bit shift code to a parallel-parallel, equal-length-word, four-eight code.

It is desirable to have a single name for all of these conversions. 'Language translation' seems too restrictive, since a mere format conversion hardly involves a translation between two different languages. On the other hand, 'language conversion' seems satisfactory, since 'conversion' is flexible enough to cover all the cases, and we are certainly dealing with languages.*

Another example of a conversion that is both a code and a format conversion is that between the coded language and the internal language designed by the Michigan Project.** In fact, an appreciable proportion of the translation which will occur in a highly developed automatic accounting system is of a comparable degree of complexity. Under these circumstances one would expect in a report of this character a discussion of the logical design of converters for all three types of conversion. But it must be remembered that general design techniques for such converters are being discussed rather than the design of specific converters. Now the logical components of code-and-format converters are the components of code converters and format converters as well. Hence it seems fairly sufficient at this level of generality to discuss the design of code converters and format converters, omitting any detailed discussion of code-and-format converters as such. An additional reason for this omission is that any example of a code-and-format converter is likely to constitute enough material by itself for a fairly substantial report. In reading this report, however, it should be kept in mind that the individual code converters (Volume I) and format converters (Volume II) are emphasized primarily because this seems a suitable way of discussing the logical components (and their interconnections) for all three kinds of converters.

---

* The term 'language conversion' was suggested by Don Stevens.

** See the references in the penultimate footnote preceding.

VOLUME I

THE LOGICAL REALIZATION OF TRANSLITERATIVE FUNCTIONS

1. Introduction

Volume I is a study of the design of equipment for realizing* Transliterative Functions (essentially, single-valued functions of sequences of 0's and 1's of uniform length to sequences of 0's and 1's of uniform length—see Def. 4, section 4). Such devices will be called Function Switches. They may also be defined in more customary terms as IRE (1)** function switches*** modified by requiring (1) that a simultaneous application of signals to the input will cause the simultaneous appearance of the corresponding output signals and (2) that none of the auxiliary circuits for timing, pulse generating, etc., be regarded as part of the Function Switch. These requirements do not prevent the bits of a sequence from coming serially into the mechanism (of which the Function Switch is a part). The circuits for handling such an input, however, would not be part of the Function Switch proper as the term is used here.

---

* The term Realization is formally defined in section 4. This formal meaning, however, is simply an attempt to make precise, in a particular context, the generally understood meaning of the term which is adequate for all but the most 'formal' occurrences of the term throughout the report. In general, technical terms when used in special sense peculiar to this report will be capitalized and will be underlined at the first occurrence. The index will show where the term is introduced or defined.

** Such numbers appearing in parentheses after an author or term refer to the bibliography at the end of the volume.

*** The IRE definition states that a function switch is "a network or system having a number of inputs and outputs and so connected that signals representing information expressed in a certain code, when applied to the inputs, cause output signals to appear which are a representation of the input information in a different code". In the terminology of this report the IRE function switch is a "language converter".

As was pointed out in the General Introduction, the design of Function Switches is broken down into three stages: logical, idealized equipment, and complete circuit design. The present volume is devoted exclusively to the first of these stages. The principles of the propositional calculus are used freely. "Logical elements" (assumed realizable in equipment) are defined having the properties of conjunction, disjunction, and nagation, and symbolic networks are constructed by "wiring" these elements together to produce switches.

The aim of the volume is achieved by first giving constructive procedures for a few basic component—or "building-block"—switches which may not in themselves constitute a useful realization technique; next, in terms of these components, defining switches for certain special kinds of Transliterative Functions; and finally, giving techniques for combining and modifying these switches to give switches realizing an arbitrary Transliterative Function. In section 2, the basic concepts used in the construction and study of the abstract logical circuits are defined. All the methods for constructing Function Switches are given in sections 3 and 4 with the exception of that for the folded tree, which is discussed in section 5 in connection with minimality and loading properties of switches.

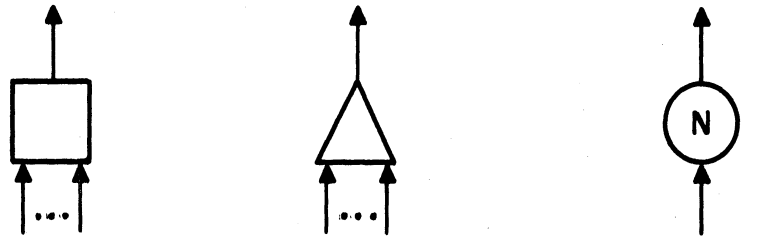
2. Definition of Logical Elements and Networks

This section introduces the basic logical elements which will be combined into symbolic circuits (representing physical Function Switches) by techniques to be considered later. It gives rules for the combination of these elements into networks and places some restrictions on the types of circuits to be considered. Finally, it defines and discusses some general properties of these logical networks, which may have some significance for their physical realization and which may be used as a basis for comparison.

The logical operators: conjunction, disjunction, and negation, are chosen as the basic elements. It is assumed (and will be shown in a later volume) that they can be realized by physical equipment (e.g., pulse control units, relays, tubes, etc.). Each such piece of equipment is represented by a geometric figure or enclosure along with a set, W , of directed line segments associated with it (see Figure 1). The line segments correspond to the Inputs and Outputs of the physical element and will be referred to as Wires. The figure together with its wires is called a Logical Element. Consider a Logical Element, E . Let W represent the set of wires associated with it. Then the inputs of E form the subset, W_I , composed of all the wires of W directed toward E (i.e., with the arrowheads pointing toward E). Only the inputs may be 'driven'—i.e., put into one or the other of two electrical states, '0' or '1' , by connection to an outside source. Any subset W_O of W may be denoted as the set of outputs of E provided that the union of inputs and outputs is the set W . The output will be the single arrow (wire) directed away from the Element unless otherwise specified.

The Logical Elements are defined as follows: The square in Figure 1(a) represents a piece of equipment with two or more inputs and a single output which is in the 1 state if and only if all of the inputs are in the 1 state. Thus it represents the logical operation, conjunction, and is referred to as a Conjunctive Element. The triangle of Figure 1(b) represents a Disjunctive Element: its single output is in the 1 state if and only if at least one of its inputs is in the 1 state. The circle enclosing an 'N' , Figure 1(c), is a Negation Element—a device with a single input and a single output having the property that the state of the output is opposite to the state of the input.

The above three Primitive Elements are sufficient for constructing all of the networks to be considered and all such networks will be assumed,

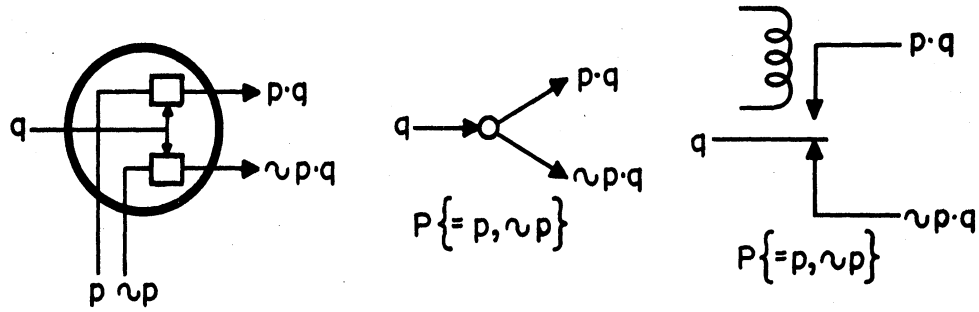


(n INPUTS) CONJUNCTION (a)

(n INPUTS) DISJUNCTION (b)

NEGATION (c)

PRIMITIVE LOGICAL ELEMENTS
FIGURE 1.



STANDARD NOTATION (a)

ABBREVIATED NOTATION (b)

RELAY REALIZATION (c)

AN AUXILIARY LOGICAL ELEMENT
THE "DOUBLE-AND"
FIGURE 2.

in the interests of simplicity, to be compounds of these Primitives even though, in certain applications, another choice of Primitives would seem to serve as well or better.

As an example of an auxiliary compound element, the Double-and is introduced in Figure 2. This element is a logical interpretation of the relay (Figure 2c) as it is frequently used in contact networks (cf. section 5). It is represented as a compound of Primitive Elements (Conjunctive) in 2(a) and in convenient abbreviated symbolism in 2(b). The labelling of the input pair (~p, p) represents the standard relay interpretation which is the only mode of operation to be considered.

If tubes are to be used, it is natural to define enclosures (elements) representing the stroke functions. With the proper interpretation of voltage levels, a double triode corresponds to the disjunctive and the pentode to the conjunctive stroke function; these elements are discussed in section 3.3.

Although such compound elements as the two above are of secondary importance to this report, the possibility of such alternatives deserves emphasis because of their usefulness in more specialized applications.

The Function Switches considered in Volume I will be compounded from the Logical Elements exclusively, though not all possible combinations will be considered. The allowable combinations are defined by the following rules whose primary purpose is to ensure that each output will be a function of the inputs. Such a combination is called a Network.

1. Any one of the Primitive Logical Elements, or—the "degenerate element"—a single wire with terminals labelled 'input' and 'output', constitutes a Network whose Inputs, Outputs, and Wires are those of the Element.

2. If F_1 and F_2 are distinct Networks with designated inputs and outputs, they can be combined to form a compound Network F as follows:

A.1 All inputs of F_1 are designated inputs of F .

.2 Each input of F_2 must be designated an input of F or it must be connected to a single output wire of F_1 . (Several inputs of F_2 may be connected to the same output wire of F_1 .)

.3 No other connections between F_1 and F_2 are permitted.

B.1 All outputs of F_2 are designated outputs of F .

.2 Every output of F_1 which is not connected to an input of F_2 must be designated an output of F .

.3 Any inputs of F_1 or F_2 may also be designated outputs of F .

C. The Wires of F are defined to be its inputs together with its outputs.

The force of 2C is that the internal connections of a network—unless they are Network outputs—are not called Wires of F but must be referred to as Wires of the components, F_1 , F_2 , etc.

The rules B.1 and B.2 (omitting B.3) define a set of outputs which is adequate for all specific Switches* discussed in this report and will be referred to as the Natural Outputs. Rule B.3 will, of course, be assumed in any general remarks or theorems concerning the class of Networks.

Some typical examples of Networks are shown in Figures 3a, 3b, and 5b. Figure 5c is an application of the Double-and notation to the Network of Figure 5b.

It is the philosophy of this volume to treat Networks as logical entities. Nevertheless, there are frequent allusions to concrete applications and it is to be expected that some readers will carry this translation into physical equipment even further. It should be pointed out, therefore, that the Networks are most easily interpreted in terms of static operation; i.e., the 1 and 0 signals or states of the Wires should be thought of as "high" and "low" d-c levels rather than as pulses. The ramifications of the pulse or dynamic interpretation are taken up in section 4.4.

Although the general question of which of a number of circuits or Networks is "best" for a particular job must remain unanswered (or, perhaps even unformulated), two criteria for comparison which appear to be of some use are presented below.

The first is the Element Input Count denoted by 'C'. It is the total number of inputs for all of the Logical Elements of the Network.

---

* The term 'Switch' will be used frequently in referring to a specific Network.

The network of Figure 3(a) has 12 three-input Conjunctive Elements and so C = 36, while the "tree" of Figure 5(b) has 28 two-input Elements and C = 56. The Element Input Count is considered a rough index of the complexity of the circuit relative to the type of Element used in its construction. Its significance depends on the particular physical elements used in its realization. Thus in trees (pyramids) designed by Aiken's (1) techniques the number of control grids is a specific interpretation for the Element Input Count of the corresponding abstract circuit.

The second property to be defined is a loading property. Two types of loading are distinguished for each Input of a Network, serial loading and parallel loading. To determine the serial loading of an input q, consider the set of all sequences of Logical Elements of the Network, $\{(E_1,E_2,\ldots,E_m)\}$, where (1) q is an input of E_1, (2) the output q_i of E_{i-1} is connected to an input of E_i (and possibly to other elements), and (3) the output of E_m is an output of the Network. Such sequences are called Chains. (A Conjunctive Chain is one composed entirely of Conjunctive Elements.) The length of a Chain is the number of Logical Elements it contains. Pick out any Chain L of this set which has a length equal to or greater than that of any other Chain in the set. Then the length of L is defined to be the Serial Loading Coefficient of q.

Although no use is made of it here, it is interesting to note that the idea of serial loading may be extended by defining the "serial loading coefficient of a logical element, E" in an analogous fashion, the output of E replacing q in the definition given above.

The Parallel Loading Coefficient of a Network input, q, is defined as the number of Logical Elements to which q is directly connected. As in the case of serial loading, this idea may be extended to any Logical Element.

The parallel loading of an input, q, does not describe the situation completely. This can be done only by considering the parallel loading coefficients of all the Logical Elements appearing in Chains having as a first term a Logical Element with q as an input. These coefficients might be displayed in the form of a matrix to be associated with q. For the special cases to which it will be applied in this report (with one exception in section 5), the concept as defined is adequate.

The loading characteristics of a Network will be presented as a set of Loading Couples: number pairs, (S,P), one for each Network input, where S is the Serial and P the Parallel Loading Coefficient. In Figure 3b, the Loading Couples for all Network Inputs are the same, (1,8). In Figure 5b the loading of the two wires of a pair is the same: for a

wire of P_1 or of P_2, the couple is (3,2); for P_3, (2,4); and for P_4, (1,8).

The significance of these numbers varies with the physical equipment considered for the realization. For example, in a relay tree the serial loading and the parallel loading for Logical Elements is of little significance, while the Parallel Loading Coefficient for the switch inputs is of great importance: it is the number of transfer contacts operated by an input relay. It is studied in section 5 and more extensively in Volume III. On the other hand, serial loading is of considerable interest if the circuit is to be realized by crystal rectifiers.

3. Logical Design of Networks

The purpose of sections 3 and 4 is to present techniques for "constructing"—at the abstract level of design—switches which realize Transliterative Functions. It has been found possible to realize all Transliterative Functions by simple combination of a very small number of components (Switches). Since these components may have wider application than it is the purpose of this report to consider, they will be presented separately in the present section, independently of their use in realizing Transliterative Functions. In section 4 this application will be considered in detail. Section 3 will, however, anticipate the subject matter of section 4 to the extent that the components are capable, without modification, of realizing functions.

The present section will be divided into two subsections:

3.1. "Conjunctive Switches: MS (Multiplicative Switches) and MS Nets" showing first the construction and operation of these components, and, second, a more detailed discussion of two important specific examples, the "Tree" and the "Balanced MS Net".

3.2. "Alternative Switches Using Other Logical Elements" which describes, primarily, the use of stroke function elements.

The definitions of Switches in this section (with one exception) have the form of constructive procedures: i.e., given the proper specification of input sets, use of the rules laid down by a definition will produce a unique Switch. This aspect will be pointed out specifically for each definition as it is presented.

3.1. Conjunctive Switches: MS (Multiplicative Switches) and MS Nets

The Multiplicative Switch—hereafter referred to as 'MS'—derives its name from the mathematical concept Cartesian Product.* The Cartesian Product idea, which is fundamental to this volume, is most frequently applied to sets of wires. To facilitate its application to Networks, a

* Given a sequence, $\mathcal{S}$, of sets, $S_1,\ldots,S_n$, the Cartesian Product of $\mathcal{S}$ is the set of all possible n-term sequences such that the ith terms of any sequence belong to S_i. Note for future reference that this operation is associative: $(S_1 \times S_2) \times S_3 = S_1 \times (S_2 \times S_3)$.

closely related Concept called the "Cartesian Conjunction" will be defined. Consider a Switch A, a collection, $\mathcal{I}$, of disjoint sets, $I_1,\ldots I_n$, each consisting of two or more input wires of A, and the set, O, of the output wires of A; then

> **Def. 1.** A subset, S, of O is called a Cartesian Conjunction of $\mathcal{I}$, denoted '$CC(\mathcal{I})$', if there is a 1-1 correspondence between the Cartesian Product of $\mathcal{I}$ and S such that if w_e is the wire of S corresponding to e, an element of the Cartesian Product, then all Conjunctive Chains with w_e as output originate in e and at least one Chain is connected to each wire of e (e.g., see Figure 3a or 5g).*

In all the Switches explicitly discussed in this volume $CC(\mathcal{I})$ will be unique and will be referred to as the Cartesian Conjunction of $\mathcal{I}$ although in general there may be several distinct sets which are Cartesian Conjunctions of a given $\mathcal{I}$.

The Cartesian Conjunction operation can be iterated: i.e., the I_i sets may themselves be Cartesian Conjunctions. This leads to:

> Corollary to Def. 1. Cartesian Conjunction is associative; i.e., $(CC(\mathcal{I}_1) \times CC(\mathcal{I}_2)) \times CC(\mathcal{I}_3) = CC(\mathcal{I}_1) \times (CC(\mathcal{I}_2) \times CC(\mathcal{I}_3))$.

This follows directly from the relation to Cartesian Products which is associative.

Throughout the report, the number of elements in a set or sequence, S, will be denoted by the expression $N(S)$. Applying this notation to the present case, clearly

$$N(CC(\mathcal{I})) = \prod_{j=1}^{n} N(I_j) \quad . \tag{1}$$

The following property is also worth noting. Given an input set, I, a grouping into a collection, $\mathcal{I}$, of disjoint subsets $I_1,\ldots,I_n$, such that $I = \bigcup_{j=1}^{n} I_j$, and the resultant $CC(\mathcal{I})$, it is not possible, on the same I, to form a different grouping, $\mathcal{I}'$ (of subsets I'_j, $I = \bigcup_{j=1}^{n} I'_j$) such that $CC(\mathcal{I}') = CC(\mathcal{I})$; and, conversely, two unequal output sets

---

* The operating condition being established here is that, to each e corresponds an output wire which is in the 1 state if and only if every wire of e is in the 1 state. Note that $CC(\mathcal{I})$, unlike the Cartesian Product, is independent of the order of the sets in $\mathcal{I}$.

cannot be the Cartesian Conjunction of the same $\mathcal{I}$. These statements follow easily from the definition of Cartesian Product.

An MS is a Switch for producing the Cartesian Conjunction—specifically:

> Def. 2.* A Multiplicative Switch (MS) is a network whose set of inputs, I , can be grouped into a collection, $\mathcal{I}$, of disjoint subsets such that $I = \bigcup_{j=1}^{n} I_j$, $(N(I_j) \geq 2)$ in such a way that the set of outputs is $CC(\mathcal{I})$, each output wire being the output of a single n-input Conjunctive Element whose inputs are the wires of the corresponding element, e , of the Cartesian Product of $\mathcal{I}$.

The constructive force of this definition is that, given an input set I and a partitioning, $\mathcal{I}$, then one and only one MS can be produced.**

Figure 3a shows an MS with the input set I of 7 wires grouped into subsets, I_1 of 3 wires and I_2 and I_3 of 2 wires each, and with the Cartesian Conjunction represented by the 12 output wires in accordance with formula (1).

For MS's in general the number of Logical Elements (Conjunctive) required is equal to the number of output wires, $\prod_{j=1}^{n} N(I_j)$, and the Element Input Count is given by the formula

$$C = n \prod_{j=1}^{n} N(I_j) \quad .$$

---

* One of the essential characteristics of a Multiplicative Switch is that every sequence of states appearing on its inputs has the same number, k , of 1's . This property might be taken as the basis for defining a broader class of switches which includes the class of MS's as a subclass. Such Switches would realized Transliterative Functions whose domain sequences each have a fixed number, k , of 1's , and whose range sequences, as for the MS (cf. Decoding Function, section 4), each have a single 1 . A switch of this class would consist of a set of k-input Conjunctive Elements, one for each sequence in the domain, the Element inputs being connected to the k switch inputs which register 1's corresponding to the associated domain sequence. This type of switch would, for example, decode a 2-out-of-5 code which cannot be handled by the MS.

** For this purpose, Switches with inputs or outputs permuted are not considered distinct—the problem is discussed in more detail in connection with MS Nets (last footnote, p. 19).

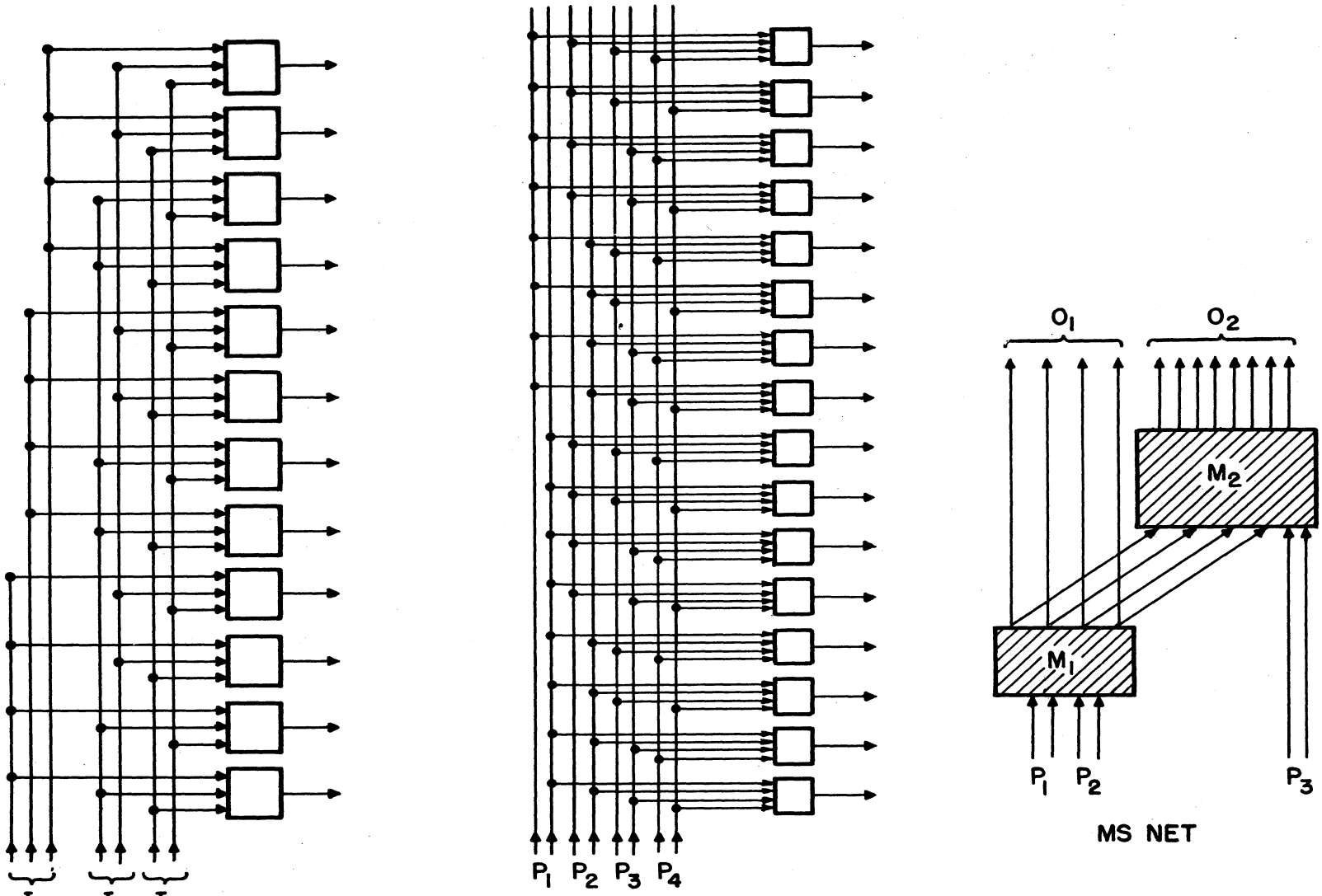


FIGURE 3. EXAMPLES OF NETWORKS

The Serial Loading Coefficient for any input wire is obviously 1 ; the parallel loading, however, is dependent on the subset, I_j , to which the wire belongs and the loading characteristic is expressed, for a wire $w_{\bar{j}}$ of $I_{\bar{j}}$, by the Couple

$$(1, \prod_{j=1}^{n} N(I_j)) \qquad (j \neq \bar{j}) \ .$$

E.g., for the MS of Figure 3b, the Loading Couple for any wire of I_1 is (1,4); for any wire of I_2 or I_3 , (1,6).

The MS shown in Figure 3b deserves special mention. It is a familiar piece of decoding apparatus and is characterized by the property of having its set of input wires grouped into pairs (i.e., for all j=1, ...,n , $N(I_j) = 2$). Henceforth, such input pairs will be referred to as Polar Pairs because of the common mode of operation which uses such a pair to represent a truth value-variable (or a binary digit-position): a signal on one wire meaning 'true' (or 1) and a signal on the other meaning 'false' (or 0), which requires that the two wires be in opposite states, i.e., polarized. The Switch itself will be called an Exponential Switch because of the analogy between its relation to the general MS and the mathematical relation of exponentiation to multiplication.*

The concept of Polar Pair provides motivation for the introduction of a component which, though not itself a conjunctive switch, is frequently used as an auxiliary to such switches: the Polarizer. The purpose of this Network is to convert a set of inputs representing one variable per wire into the Polar Pair representation. This is done by the obvious device, shown in Figure 4, of introducing a negation element in parallel with each original wire. Clearly, given a set of inputs, the Polarizer is unique. The primary importance of the Polarizer in the development of the present theory is that it allows the existence of Polar Pair inputs to be assumed, without loss of generality, whenever it is convenient to do so.

---

* The analogy is particularly interesting if the set-theoretic basis for exponentiation (as related to Cartesian Product) is considered where, letting m and n represent the cardinality of two sets, S_m and S_n , m^n is interpreted as the number of possible mappings of S_n into S_m , which is also the cardinality of the Cartesian Product, S_m X...n factors ...X S_m . The analogy can be followed through easily to produce a 1-1 correspondence between the set of mappings and the set of switch outputs. In the present case, n is, as before, the number of subsets of the set of inputs, and m is 2 . Note that the definition could apply equally well for any integer, m , other than 2 . ($N(I_1) = N(I_2) = \ldots = N(I_n) = m$); the restriction to $m = 2$ is merely to effect a slight simplification in the present report.

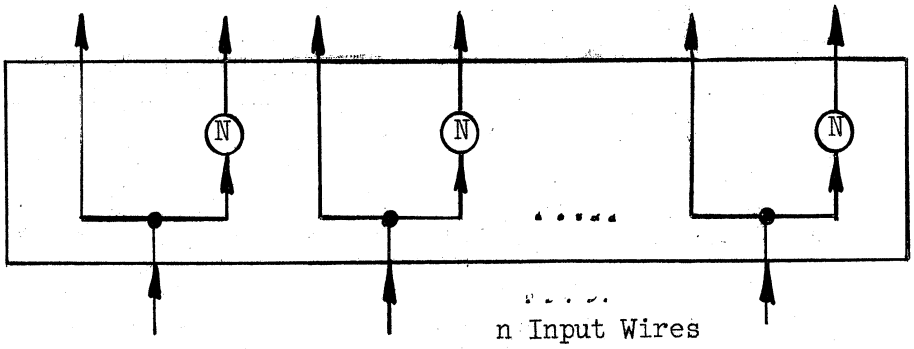


Figure 4. Polarizer

The final definition of this subsection provides a means for interconnecting MS's to give greater flexibility and at the same time provides limitations which keep the resultant class within reasonable bounds for purposes of analysis.

> Def. 3. An MS Net is a Network composed entirely of MS's and (1) having a set of input wires, I, which can be grouped into a collection, $\mathcal{L}$, of subsets, $I = \bigcup_{j=1}^{n} I_j$, such that $CC(\mathcal{L})$ is a subset of the set of Network outputs;* and with the MS's connected as follows: (2) If I_j^M is a subset of the input set of an MS (as in the definition of MS) and if S is the set of outputs of an MS or one of the I_j of the MS Net, then any connection between the two sets requires that $I_j^M = S$ (i.e., the two sets have the same number of wires and the wires of I_j^M are connected to those of S in 1-1 fashion). And similarly if a wire of S is designated as a Net output, then the entire set S must be so designated.

This definition is the exception to the constructive procedure policy. The rules of the definition do not lead to a unique switch, given specified input sets. The definition merely defines a broad class of networks to which some of the most useful Switches (e.g., Trees) belong.

---

* The set of outputs of an MS Net may consist of any outputs allowed by the definition of Network which satisfy (1); however, most of the useful examples explicitly referred to will have as their designated output set the set of Natural Outputs. Similarly with inputs: only the Polar Pair case will be considered in detail.

Some of the conditions of this definition will bear amplification. The explicit requirement that the MS Net be a Network is necessary to rule out feed-back loops which are unnecessary in code transliteration. Condition (2) is of some interest and will be discussed in more detail in Volume III of this report. For the present it may be said that its purpose is to prevent Splitting of MS output sets—some wires connected to one MS sub-input, some connected to another—a situation which has thus far resisted general analysis to a degree altogether out of proportion to any advantages apparent in the procedure. It should be pointed out, however, that condition (2) does permit a phenomenon which shall be referred to as Branching: in the terms of the definition, an S , if taken in its entirety, may be identified, in the prescribed manner, with more than one I_j and/or be designated a Net output.

Figure 3c is an MS Net (with Branching), the shaded enclosures representing the MS's, $O_1 = CC(\mathcal{L}')$, $\mathcal{L}' = \{P_1, P_2\}$ and $O_2 = CC(\mathcal{L})$, $\mathcal{L} = \{P_1, P_2, P_3\}$. The output set consists of the union, $O_1 \cup O_2$.

Figures 5 and 6 represent a type of MS Net which is of particular importance and is characterized by having no Branching; i.e., an MS output (or Net input) is directly connected to exactly one MS input subset or is a Net output (not both). Most MS Nets to be dealt with are of this type and will be referred to as Simple MS Nets. In such a Net it is easily seen that the outputs are the Natural Outputs and constitute precisely the Cartesian Conjunction of the input subsets.

The construction of an MS Net can be expressed notationally by a Construction Formula. This formula gives the output of the Net as a function of its inputs. The connective, 'X' , denotes Cartesian Conjunction, a parenthesized expression denotes and MS (if an expression occurs more than once it represents a Branching of the output of the MS), and the connective, u , indicates that the Switch output set is the union of the output sets of the MS's so joined—no additional connections being made. (The symbol P_j will be used rather than I_j whenever the input subsets are Polar Pairs.)*

The MS Net of Figure 6 for $n = 4$ is expressed by the formula

$$(P_1 \text{ X } P_2) \text{ X } (P_3 \text{ X } P_4) \qquad (2)$$

---

* Although no such cases arise in this report, it should be pointed out that the occurrence of identical MS's (i.e., described by identical Formulas) has not been provided for. The situation can be handled easily by attaching subscripts to the parentheses if distinct MS's are to be formed.

and that of Figure 3c by

$$(P_1 \times P_2) \cup ((P_1 \times P_2) \times P_3) \quad . \tag{3}$$

The 'u' in formula (3) indicates that the output is produced by two MS's. Note that (2) represents a Simple MS Net (no Branching), since no expression is repeated, but that (3) has Branching as indicated by the repeated occurrence of $'(P_1 \times P_2)'$.

The associative property of Cartesian Conjunction offers a method of constructing a variety of Nets to produce the same output function. For example:

$$(P_1 \times P_2 \times P_3 \times P_4) \tag{2'}$$

$$(((P_1 \times P_2) \times P_3) \times P_4) \tag{2"}$$

represent alternative methods (i.e., alternative MS Nets) for realizing the output given by formula (2). Formula (2') yields the Exponential Switch of Figure 3b, and (2") the Tree of Figure 5.

Having established all of the basic conjunctive components and discussed some of their general properties, two specific MS Nets of particular importance will now be considered.

The Standard Tree. The most succinct characterization of a Standard Tree is by means of its Construction Formula,

$$((\ldots(((P_1 \times P_2) \times P_3) \times P_4) \times \ldots) \times P_n) \quad .$$

From this formula it can be seen that the input set (I) consists of a collection ($\mathcal{J}$) of n (Polar*) Pairs, $P_1,\ldots,P_n$; that P_j is conjoined with the preceding P's by constituting one of two input subsets for an MS whose other input subset is (in effect) $P_1 \times \ldots \times P_{j-1}$; and that the Tree output is the output of the "final" MS—the one having P_n among its Inputs. It is clear, also, that the Standard Tree is a Simple MS Net, since no expression is repeated in its Construction Formula and the Formula expresses precisely $CC(\mathcal{J})$.

---

* The only mode of operation to be considered for the Tree (as well as for most other Switches) is with Polar Pair inputs; therefore it is convenient to use the P_j notation from this point on. It must be pointed out, however, that, for purposes of constructing the switch it is not intended to impose any restriction on the manner in which the inputs are energized—it is required only that the input set satisfying $I = \bigcup_{j=1}^{n} I_j$ has $N(I_j) = 2$ for all j .

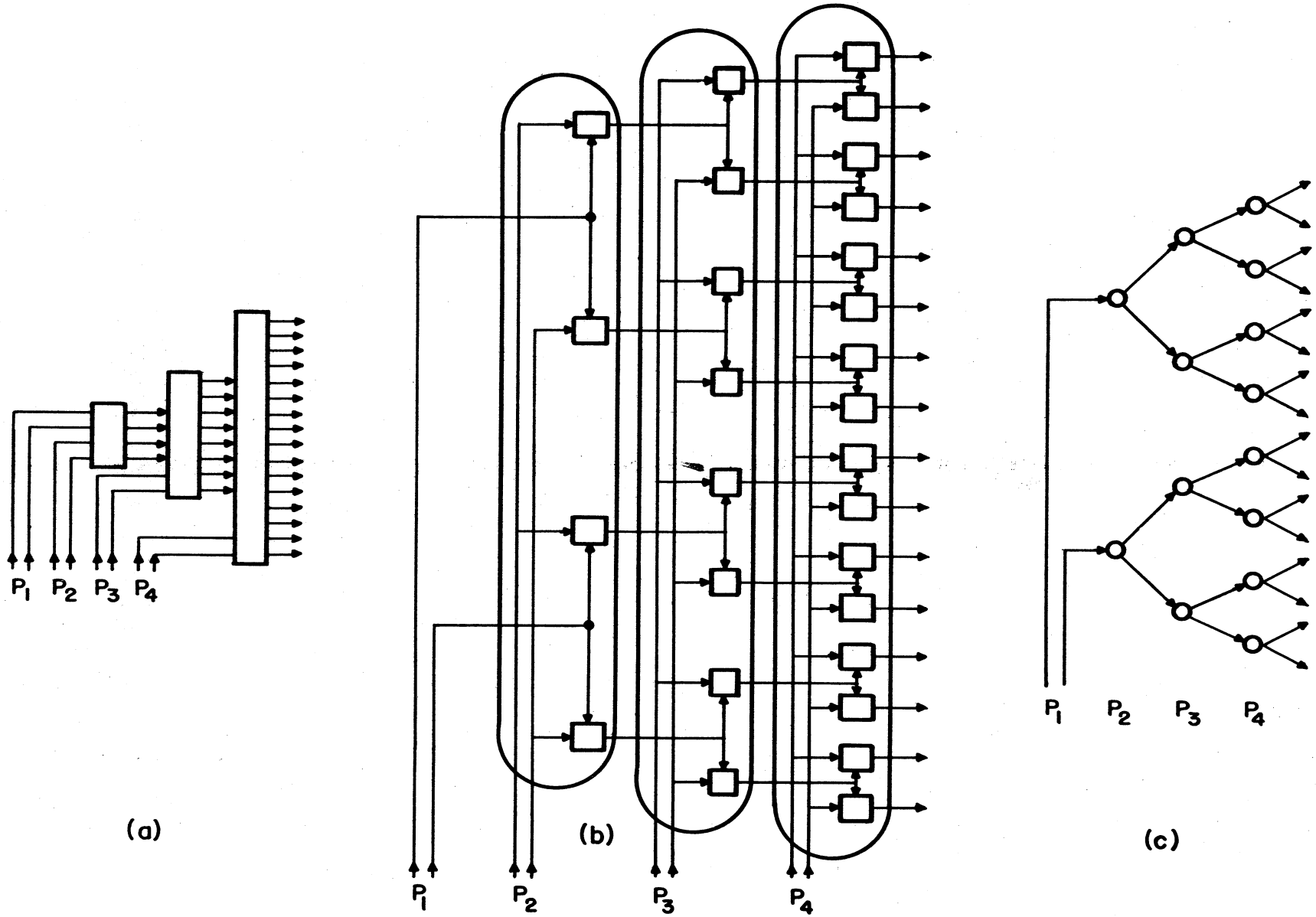


FIGURE 5. STANDARD TREE

As a constructive procedure, it is clear that, given the input set, I, and its collection of subsets, $\mathcal{A} = \{P_j\}$, a "unique" switch is determined—unique, that is, with the usual minor qualifications concerning permutations (see last footnote, p. 19).

Figure 5 represents a typical Standard Tree. In 5a the basic elements are MS's. In 5b the MS's are isolated by the heavy-line enclosures and their internal Elements shown in detail. Figure 5c shows the abbreviated Double-and notation which derives its simplicity from two important characteristics of the Standard Tree: (1) the Conjunctive Elements always occur in pairs; and (2) all of the Conjunctive Elements (or, equivalently, the Double-ands) of an MS (those constituting a vertical column in the diagrams) are connected directly to one and only one P_j —each such column is said to form a Bay. Characteristic (1) is common to the larger class of "Folded Trees" (which is discussed in section 5 and includes the Standard Tree), while (2) is peculiar to the Standard Tree. In Figure 5c all the Double-ands in a column are associated with the P_j appearing below it.

The element input count for the Standard Tree (and, in fact, for Trees in general) is given by the formula

$$C = 8(2^{n-1} - 1) \ ^{*} ,$$

where n is the number of pairs in the input. The number of conjunctive elements is, of course, $C/2$.

The loading figures will be given for a P_j, since they are identical for the two wires. Both the serial and parallel loadings vary with the input pair and are given in the usual Couple notation by $(n-j+1, 2^{j-1})$ for $j \geq 2$. For P_1 the value of the Couple for $j=2$ applies. The second (parallel) Coefficient here is of special interest (particularly in relay networks), since it may become excessively large as j increases. The folding process, described in section 5, is an effective method for distributing this load over the P_j.

The Balanced MS Net.** The second of the specific Switches to be described is, again, a Simple MS Net. The designation 'Balanced' is

---

* In line with the general policy of this section, this formula is for a "static" tree. The "dynamic" tree, which is more common in the literature and which is discussed extensively in section 5 and in Volume III, has the formula $C = 4(2^n - 1)$.

** The concept of this Switch was derived from Brown and Rochester (8), where a crystal diode version was presented and discussed in somewhat unprecise language.

derived from the fact that each of its MS's has exactly two input subsets*— $I = I_1 \cup I_2$ —and that $N(I_1)$ is as nearly equal to $N(I_2)$ as possible. Here, as in the case of the Tree, the definition is in terms of Polar Pair inputs. The switch will be defined recursively by giving rules for its construction starting with the output and proceeding systematically to the input.

The Balanced MS Net with input set $I = \bigcup_{j=1}^{n} P_j$ is constructed as follows:

1) Designate a set of 2^n wires as Network outputs. They will constitute the output of a single MS with $I = I_1 \cup I_2$ and $N(I_1) = 2^{[(n+1)/2]}$** , $N(I_2) = 2^{[n/2]}$.

2) Let $\bar{I}$ be an input set of an MS of a partially completed Net such that $\bar{I}$ has not as yet been connected to any other MS, and $N(\bar{I}) = 2^x$ for some integer x .***

a) If $x \geq 2$ then connect in 1-1 fashion to $\bar{I}$ the output, 0 , $(N(0) = N(\bar{I}) = 2^x)$ of an MS with input set $I = I_1 \cup I_2$ having $N(I_1) = 2^{[(x+1)/2]}$ and $N(I_2) = 2^{[x/2]}$.

b) If $x = 1$, then designate the wires of $\bar{I}$ as an input pair of the Net.

Figure 6 shows a number of Balanced MS Nets. Consider the case of $n = 5$, $I = P_1 \cup P_2 \cup \ldots \cup P_5$. There are $2^5 = 32$ output wires for M_4 . The inputs of M_4 are split as evenly as possible into I_1 , $N(I_1) = 2^{[(5+1)/2]} = 2^3 = 8$ and I_2 , $N(I_2) = 2^{[5/2]} = 2^2 = 4$. In turn, the inputs of M_2 are split into I_{11} , with $N(I_{11}) = 2^{[(3+1)/2]} = 4$ and I_{12} , with $N(I_{12}) = 2^{[3/2]} = 2$. The Construction Formula for this switch is obviously

$$(((P_1 \times P_2) \times P_3) \times (P_4 \times P_5)) \quad ****$$

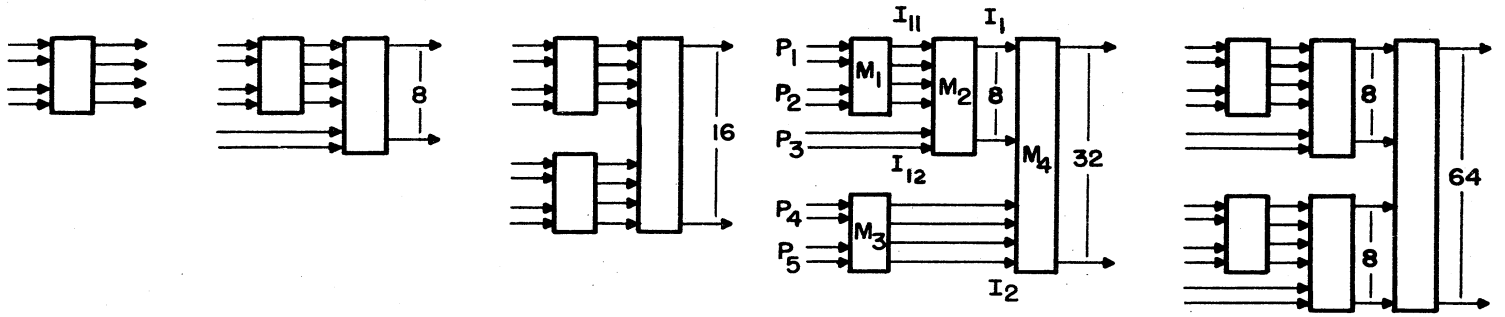
---

* Such MS's may be referred to as 'two-dimensional'.

** '[x]' means 'the integer part of x'; this notation will be used frequently throughout the report.

*** It can be established inductively that every $\bar{I}$ will satisfy this condition.

**** Note that in such a Net as this, the Construction Formula might have been written with any permutations of the subscripts on the P's without in any way changing the form of the Net. For present purposes (and there seems to be no reason in practice for doing otherwise) all such permutations of inputs which have no effect on the outputs (other than permutation) nor on the construction of the switch, will be identified—i.e., a switch for given input conditions specified by the definition will be considered unique.



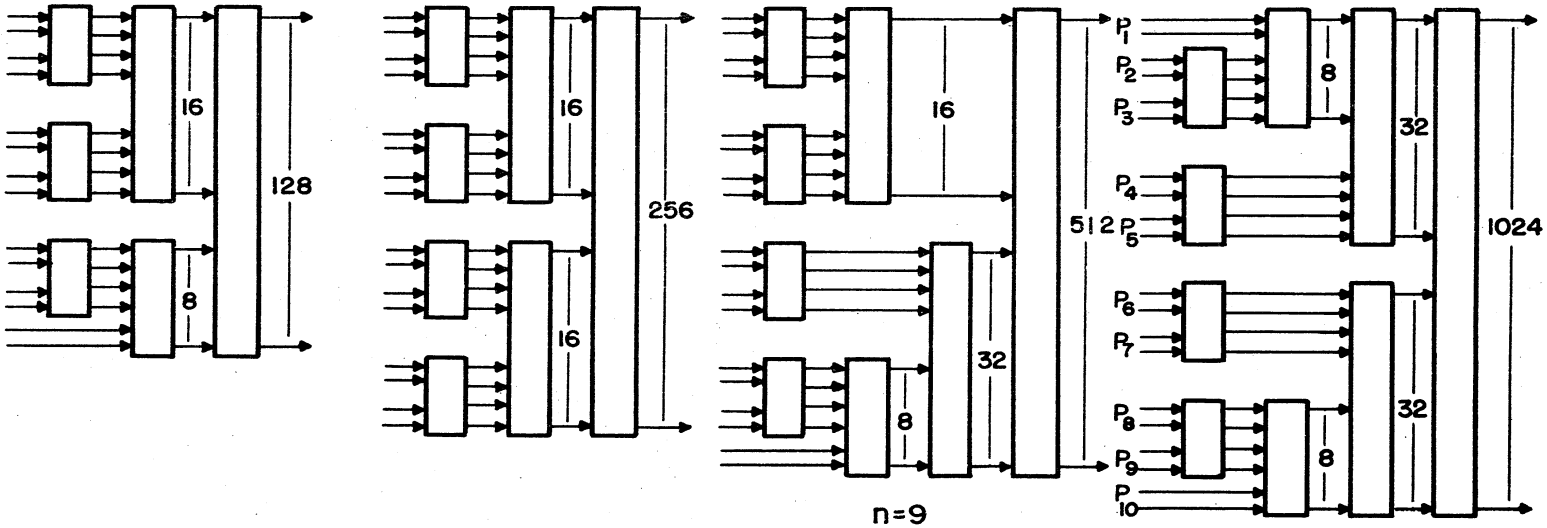
n=2
C=8

n=3
C=2(4+8)=24

n=4
C=2(4+4+16)=48

n=5
C=2(4+4+8+32)=96

n=6
C=2(4+4+8+8+64)=176



n=7
C=2(4+4+4+16+8+128)=328

n=8
C=2(4+4+4+4+16+16+256)=608

n=9
C=2(4+4+4+4+8+16+32+512)=1168

n=10
C=2(4+4+4+4+8+8+32+32+1024)=2240

LOADING n=10 { P_1, P_{10} - (3,4); P_4 THRU P_7 - (3,2); P_2, P_3, P_8, P_9 - (4,2).

FIGURE 6. BALANCED MS NETS WITH n-INPUT PAIRS (n=2,...,10)

It is not entirely obvious from the rules of construction that the resultant Net will have the required n input pairs. That this is actually the case can be demonstrated conveniently by means of the Construction Formula. Start with the output set expressed as the 5-term Cartesian Conjunction, $(P_1 \times P_2 \times P_3 \times P_4 \times P_5)$, (to use the example of Figure 6). Then the rule for inserting additional parentheses corresponding to the construction rule for adjoining MS's is to 'bifurcate' the smallest parenthetical expression into two such which differ at to number of P's by at most one. The resultant formula obviously denotes the proper number of inputs (P's), and the correspondence to the actual rules of construction can be verified readily.

The above definition defines the Balanced MS Net by telling how to built it. An alternative definition which is primarily descriptive rather than constructive and will be used in the discussion of minimality properties in Volume III may be stated as follows:

> The Balanced MS Net with input set $I = P_1 \cup \ldots \cup P_n$ is an MS Net
>
> (a) having 2^n final outputs from a single MS,
>
> (b) using 2-dimensional MS's throughout, and
>
> (c) such that each MS has its input sets 'balanced'; i.e., for 2^x outputs the two input sets have $N(I_1) = 2^{\lceil (x+1)/2 \rceil}$ and $N(I_2) = 2^{\lfloor x/2 \rfloor}$.

The two definitions have been proved equivalent but, in the interests of brevity, the proof is omitted.

The Element Input Count and Loading Couples for the Balanced MS Net cannot be conveniently stated for the general case. For the case $n = 2^k$, however, the Element Input Count is $C = 2 \sum_{i=0}^{k-1} 2^{(2^{k-i}+1)}$. Figure 6 presents schematic diagrams of the Nets with n input pairs for values of n ranging from 2 to 10 with the Element Input Count, C, computed for each and the Loading Couples given for the case of $n = 10$.

The most important characteristic of this switch is stated formally as follows:

> Theorem. The Element Input Count of a Balanced MS Net with n input pairs is less than or equal to the Element Input Count of any n-input-pair MS Net.

The proof of this theorem is rather lengthy and will be reserved for Volume III.

3.2. Alternative Switches Using Other Logical Elements

The construction of Networks consisting exclusively of Disjunctive Elements is quite straightforward. Such Nets are important in encoding, but because of their simplicity and because no alternative modes of construction are considered, the treatment of section 4.2 in connection with realization suffices without additional treatment here.

In section 2 the use of logical elements other than conjunction, disjunction, and negation was suggested, although these three were accepted as the "vocabulary" for the report. One slight digression from this policy was made with the introduction of the Double-and in the discussion of Standard Trees. The present section will consider some additional alternatives. All of the elements introduced are definable in terms of the standard vocabulary (e.g., see Figures 7a' and 7b'). While the alternatives could have been tied into the general theory of the report by this means, they have not been so treated and the present section can be considered as an appendix upon which one of the subsequent theory will depend.

The use of vacuum tubes leads naturally to the logical stroke functions; two examples of these functions will be discussed below.*

The so-called "Sheffer" or Conjunctive Stroke (non-conjunction), p/q = def. $\sim(p \cdot q)$, is realized by a plate-loaded pentode without a phase inverter. It is represented diagrammatically as in Figure 7a, whose definition is the same as the definition of the standard network of Figure 7a'.

The Disjunctive Stroke (non-disjunction), $p \downarrow q$ = def. $\sim(p \vee q)$, is realized by a pair of triodes in parallel. This function is diagrammed in Figure 7b, with the corresponding standard network as in Figure 7b' or as in 7b" by the duality, $\sim(p \vee q) = \sim p \cdot \sim q$.

These elements can be combined into Networks, following the rules of section 2. In particular, switches similar to the MS may be constructed: the Conjunctive Stroke MS (CSMS) as in Figure 7c and the Disjunctive Stroke MS (DSMS) as in Figure 7d. These switches give the same result as the standard MS with the qualification that the CSMS produces the negations of the outputs of an MS and the DSMS produces the output which would be produced by an MS with each of its inputs negated—cf. Figure 7b". (In other words, the CSMS with a negation on each of its outputs and the DSMS with a negation

---

* In the two realizations below it is assumed that a high voltage level represents 'true' and a low level 'false'. These conventions could, of course, be reversed with a resultant interchange in the functions realized.

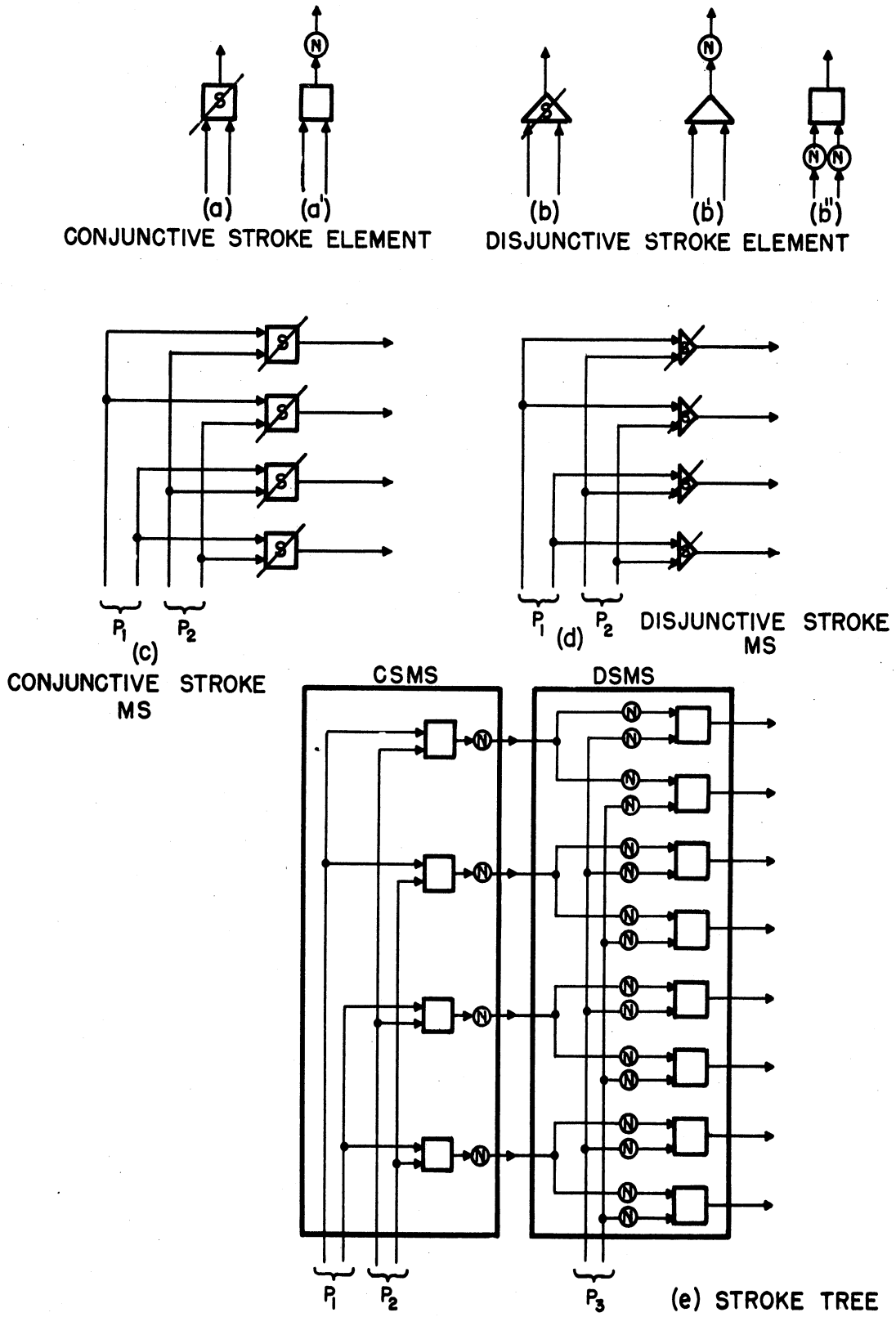

(a)
(a')
CONJUNCTIVE STROKE ELEMENT
(b)
(b')
(b'')
DISJUNCTIVE STROKE ELEMENT
P1
P2
(c)
CONJUNCTIVE STROKE MS
P1
P2
(d)
DISJUNCTIVE STROKE MS
CSMS
DSMS
P1
P2
P3
(e) STROKE TREE

FIGURE 7.

on each of its inputs would produce exactly the result of a standard MS—this is obvious from a consideration of the stroke elements themselves.)

One practical example of the use of these Stroke MS's is in the tree type of Network. Figure 7e shows a Stroke Function of 2 Bays (3 input pairs). The CSMS and DSMS are each enclosed in heavy-line boxes with their internal elements represented as in 7a' and 7b" in order to show how the interconnection effects a cancellation of the negations. In the case of a switch input connected directly to a DSMS the effect is to negate the signal on each wire of the pair, but since these are Polar Pairs, the situation is perfectly symmetric and amounts merely to interchanging the labels of the two wires. A Stroke Tree of any number of Bays may be similarly constructed by using a DSMS for the final (output) Bay and then alternating CSMS and DSMS for the remaining Bays.

4. Techniques for Realizing Transliterative Functions

The purpose of the present section is to present effective techniques for constructing, in terms of the components of the preceding sections, switches for realizing all of the transliterative functions. These functions are considered in three categories with one or more realization techniques for each category: "Decoding Functions" in section 4.1, "Encoding Functions" in section 4.2, and the general case, "Arbitrary Transliterative Functions" in section 4.3.* Up to this point Volume I has been concerned exclusively with the static mode of operation. Pulse operation is discussed separately in section 4.4, "Modifications to Provide Pulse Outputs".

First some terminology will be established. Both "realizations" and "transliterative functions" have been referred to previously, with varying degrees of formality. The formal definitions will now be given.

---

* One might naturally ask at this point why the title of this section is not expressed in a more direct manner in terms of switches (e.g., Design of Transliterative Switches) with the internal organization of the section based on types of switches rather than on functions. This approach was attempted in the initial study of the subject and was discarded for reasons which, if is felt, warrant the present digressive explanation. The difficulty is clearly exemplified by the IRE definition: "Function Switch, One-Many. A function switch in which only one input is excited at a time and each input produces a combination of outputs." (sometimes referred to as an "Encoding Switch"). Note that, in general, this definition describes a "mode of operation" rather than a type of switch—i.e., it merely defines the type of Transliterative Function the switch is to realize. Specifically, it says of the switch itself simply that it has "outputs" whose state is in some way dependent on the state of its "inputs". Under such a definition, any Function Switch (e.g., a "many-one" or "decoding" switch) would be a "one-many" switch if its inputs were properly activated. Since this type of definition offers no means for distinguishing various kinds of switches per se, there appear to be two alternatives: (1) the method of this report, which relies on functions since functions can be distinguished, and does not attempt to classify the switches; and (2) the formulation of definitions which do permit the classification of switches (perhaps from a consideration of their output under all possible input states)—an undertaking which, in the light of a cursory investigation, appears to be somewhat formidable.

Def. 4. A Transliterative Function, T , is a single-valued function (mapping) from a set of sequences of 0's and 1's all of the same length (the domain of definition) onto another set of sequences of 0's and 1's also all of the same length (the range) with the condition that there are no "dummy" digits in either the domain or the range; i.e., for any digit-position, x , of a sequence, the set (range or domain) must contain two sequences, s and s' , such that s has a 1 in position x and s' has a 0 in position x .*

The domain of T will be denoted by 'D(T)' , its range by 'R(T)' , and an element (sequence) of the domain or range, respectively, by $'d_i(T)'$ or simply $'d_i'$ and $'r_i(T)'$ or simply $'r_i'$ —i.e., $D(T) = \{d_i(T)\}$, $R(T) = \{r_i(T)\}$.

Def. 5. A network is said to Realize or to be a Realization of a Transliterative Function, T , if there exists an ordering $w_1, w_2, \ldots, w_n$ of the set of input wires and a corresponding ordering $\bar{w}_1, \ldots, \bar{w}_k$ of a subset (perhaps proper) of the output wires having the property that whenever the sequence of states of $w_1, \ldots, w_n$ is equal to a d_i, $d_i \in D(T)$, then the sequence of states of $\bar{w}_1, \ldots, \bar{w}_k$ is equal to $T(d_i)$.

This definition postulates the existence of a 1-1 correspondence between the bit positions of the domain sequences and the set of input wires, and a similar one between range sequences and a subset of the output wires. There is another important aspect of domain sequences: they may be thought of as representing information impressed on the inputs from outside. If this information is presented on n Polar Pairs of wires, it is not to be considered an n bit sequence but instead a 2n bit sequence.

The concept of Realization defined here is, admittedly, a narrower concept than the ordinary intuitive one.** It is, however, adequate for the purposes of this report and has the advantage of stating precisely the kind of realization which the given techniques produce.

---

* This condition could be omitted, enlarging the class of Transliterative Functions, but the functions thus admitted would hardly be of sufficient interest to warrant even the slight modifications that would be necessary in the subsequent theory.

** A definition closer to the intuitive idea of realization would, for example, admit, under certain circumstances, realizations in which a proper subset of the inputs is activated in accordance with $d_i(T)$.

The term 'technique' is used here in a special sense. Techniques are constructive procedures, each associated with a specified class of Transliterative Functions, describing step by step a fully effective method for building a switch to Realize a given Function of the class. Furthermore, it is desirable, and will be a characteristic of all techniques presented, that the relation between inputs and outputs required by the given function be made explicit by means of a labeling of the wires as in the definition of Realization.

The application of these Techniques together with simplifications of the Switch produced may result in "Disjunctive Elements" with but a single input. When this occurs the "Disjunctive Element" is to be eliminated and its input designated a Switch output. It is also possible that a Technique will produce a switch with a "dead-end" input—one which is neither a Switch output nor an input to a Logical Element (e.g., if an Encoding Function—see section 4.2—has the all-zero sequence in its range). Such a switch is not strictly a Network by the formation rules of section 2. However, the status of such a wire is so obvious, and the treatment so simple—merely ignore or eliminate it—that it seemed advisable to admit the discrepancy rather than to complicate the theory further in order to treat this special case with full rigor.

4.1. Decoding Functions

Def. 6. A Decoding Function is a Transliterative Function, T_d, such that:

(a) every $r_i(T_d)$ contains a single 1, and
(b) $N(D(T_d)) = N(R(T_d))$.

The effect of condition (b) is to make Decoding Functions 1-1. Functions satisfying condition (a) alone are handled by the techniques for arbitrary functions.

The inputs of the Switches constructed in section 3 were in Polar Pair form. In order to consider such input sets it is useful to define a Polar Sequence as a sequence $a_1, a_2, \ldots, a_{2k}$ having a_{2j-1}, a_{2j} either 0,1 or 1,0 for $j = 1,2,\ldots,k$.

Def. 7. A Complete Decoding Function, T_c, is a Decoding Function satisfying the following:

(a) every element of $D(T_c)$ is a Polar Sequence, and
(b) every Polar Sequence of length $N(d_i(T_c))$ is an element of $D(T_c)$.

It follows clearly from this definition that

$$N(D(T_c)) = 2^{N(d_i(T_c))/2} = N(R(T_c)) .$$

Theorem 1. A Complete Decoding Function, T_c, with $N(d_i(T_c)) = 2n$ can be realized by any MS Net with $I = \bigcup_{j=1}^{n} P_j$ (i.e., with input set consisting of n Polar Pairs).

Corollary. If T_c is realized (in accordance with Theorem 1) by a Simple MS Net, then there are no extraneous output wires, i.e., the set of output wires, $0 \overset{(1\text{-}1)}{\longleftrightarrow} R(T_c)$.

Proof. $D(T_c)$ is the Cartesian Product of n sets, each consisting of the two pairs, (0,1), (1,0), i.e., $D(T_c) = ((0,1), (1,0))^n$, and is represented by the states of the pairs of wires of $\mathcal{A}$, the set of n P_j's of the Net. The Network output, 0, by definition contains the Cartesian Conjunction, $CC(\mathcal{A})$, i.e., a distinct output wire of $CC(\mathcal{A})$ is activated for each sequence of the Cartesian Product, $D(T_c)$, represented on the inputs. Each wire of the $CC(\mathcal{A})$ may be assigned an integer, x, as follows: Consider the image sequence, $T_c(d_i)$, for $d_i \in D(T_c)$. It contains a single 1. Denote the position in which the 1 appears by x, and assign x to the wire of $CC(\mathcal{A})$ which is in the 1 state for d_i. If this is done for all $d_i \in D(T_c)$, all wires of $CC(\mathcal{A})$ will be labeled by an x, $1 \leq x \leq 2^n$.

The wires of $CC(\mathcal{A})$ then, considered in the sequence of the x's, clearly effect a Realization of T_c. The corollary follows immediately from the fact that for a Simple MS Net $0 = CC(\mathcal{A})$.

With the result of this theorem, the following specific Realizations are immediately available:

Technique 0. A Complete Decoding Function, T_c, with domain sequence of length 2n ($N(d_i(T_c)) = 2n$) can be Realized by

(a) an Exponential Switch,
(b) a Tree, or
(c) a Balanced MS Net, with input set $I = \bigcup_{j=1}^{n} P_j$.

Since explicit Construction Procedures were given in section 3 for each of the switches, (a), (b), and (c), and since the labeling of the wires described in the proof of Theorem 1 gives an ordering of input

and output sequences which establishes the functional correspondence, Technique 0 is fully effective as required. Note that since the three Switches are simple MS Nets, the Corollary to Theorem 1 indicates that there will be no extraneous output wires.

A Decoding Function, T , may fail to be Complete by violating either of the two conditions characterizing a Complete Decoding Function—i.e.,

(1) The elements of $D(T)$ are not all Polar Sequences.

(2) Not every Polar Sequence of length $N(d_i(T))$ is an element of $D(T)$.

In the technique given below, step A provides for failures of type (1) and step C for those of type (2).

Technique 1.

A. Examine the sequences of $D(T)$.

1. If not all the sequences of $D(T)$ are Polar Sequences, provide a Polarizer with $N(d_i(T))$ inputs.* Its outputs are to be connected, in the obvious manner, to the Switch constructed below. Define $n = N(d_i(T))$.

2. If every $d_i(T)$ is a Polar Sequence, define $n = N(d_i(T))/2$ and proceed to B.

B. By Technique 0(a), (b), or (c) construct a Switch for the Complete Decoding Function with n Polar Pair inputs.

C. From the Switch produced by A and B remove any Logical Element which is not in a Chain terminating in any one of the $N(D(T))$ required outputs.

Technique 1 is illustrated in Figure 8a (disregarding Switch M_2) by applying it to the function T_d defined in Table II of section 4.3.

* It may be the case that some part of every domain sequence—the same part for each—is a Polar Sequence and the remainder not. In practice it would be possible in this case to provide a Polarizer of less than $N(d_i(T))$ inputs for use with only the "unpolarized" positions, although formally this simplification has been ignored in the interests of presenting a unified general theory.

Clearly the sequences of $D(T_d)$ are not all Polar Sequences and so according to step A.1 a 4-input Polarizer, P , must be provided. Step B requires the construction of a Switch for the Complete Decoding Function with $N(d_i(T)) = 4$ Polar Pair inputs by Technique 0(a), (b), or (c)—i.e., by an Exponential Switch, a Tree, or a Balanced MS Net. In the example a Tree is used— M_1 in the figure. The inputs of P may be labeled in any order and the outputs of P are connected to the inputs of M_1 in the obvious manner. After a particular labeling of the inputs of P has been chosen, the outputs of M_1 are labeled by considering the $d_i(T_d)$. Assign the label '$\bar{w}_j$' to the output wire which is activated whenever the sequence of states of the input wires, w_1, w_2, w_3, w_4 , is d_i (the state of w_k corresponding to the kth bit of d_i), where j is the position number of the 1 in $T_d(d_i)$. Finally, proceeding according to step C, the unnecessary Logical Elements are eliminated—in the figure they are drawn with broken lines.

Simplifications may be possible in Switches constructed by this technique. No comprehensive study of simplification procedures has been made, but examples illustrating some of the possibilities are given in section 4.3.

4.2. Encoding Functions

Def. 8. An Encoding Function is a Transliterative Function, T_e , such that every $d_i(T_e)$ contains a single 1 .

The following technique provides a Realization for any given Encoding Function, T_e , where n is the number of domain sequences, $\bar{n}$ is the number of range sequences, and d_i is defined to be the domain sequence whose ith bit is a 1 .

Technique 2.

(1) Provide n input wires and $\bar{n}$ Disjunctive Elements. Designate the Element outputs as switch outputs; assign the integers from 1 to n to the input wires, and those from 1 to $\bar{n}$ to the Disjunctive Elements.

(2) Connect the ith input wire to a distinct input of the jth Disjunctive Element if the jth bit of $T_e(d_i)$ is a 1 .

Nothing was stated about the number of inputs of the Disjunctive Elements, since this would depend on the nature of T_e . In general, it is clear (by virtue of the exclusion of "dummy" digits in Def. 4) that no Element will require more than $N(D(T_e))-1$ inputs. In the "most efficient"

case—a 1-1 function with $N(D(T_e)) = 2^x$, $N(r_i(T_e)) = x$ —every Element will have 2^{x-1} inputs, since each output must be in the 1 state for exactly half of the 2^x input states.

Technique 2 is illustrated in Figure 8a for the function, T_e, defined in Table II of section 4.3. The Switch produced is M_2. The input wires may be numbered $w_1,\ldots,w_n$ in any order—the order here being chosen merely for convenience in the example of section 4.3. Note that in Figure 8b the single-input Disjunctive Element has been eliminated.

A Switch produced by Technique 2 can often be simplified. Such a Switch will consist of Disjunctive Elements whose inputs are wires of I, the set of input wires for the Switch. Suppose some subset, I_j, of I is contained in the input set of several of these Disjunctive Elements, say $E_1,\ldots,E_k$. Then let I_j be the input set for a new Disjunctive Element, E, and replace each E_i $(i = 1,\ldots,k)$ by a Disjunctive Element E_i' having the same input set as E_i except that the output of E replaces the subset I_j of inputs of E_i. This procedure will reduce the Element Input Count, C, by an amount

$$\Delta C = (k-1)\,(C(E)-1) - 1 ,$$

from which it is apparent that $\Delta C > 0$ if $k \geq 2 \leq C(E)$ but not $k = C(E) = 2$ —for which $\Delta C = 0$.

4.3. Arbitrary Transliterative Functions

While the Technique of this subsection is intended, as the title implies, to Realize any Transliterative Function, the functions of primary interest are those which are neither Decoding nor Encoding Functions. For the special functions previously dealt with, the present Technique degenerates to either Technique 1 or Technique 2.

The procedure for an arbitrary Transliterative Function requires two stages: first "decoding", then "encoding". That is, for a given arbitrary function, T, define a Decoding Function, T_d, and an Encoding Function, T_e, such that for any $x \in D(T)$, $T_e(T_d(x)) = T(x)$. This requires that $R(T_d) = D(T_e)$. These conditions fully specify the two functions.*

* Note that the definition of Decoding Function, plus the exclusion of "dummy digits" in the definition of Transliterative Function requires that $R(T_d)$ be constructed of exactly $N(D(T))$ sequences, each of $N(D(T))$ digits, and each having exactly one 1 in a unique position. The specific correspondence between range and domain for each of the two functions, so long as it satisfies the conditions that T_d is 1-1 and that $T_e(T_d(x)) = T(x)$, is irrelevant since such distinctions can all be resolved by the proper labeling of the wires of the same Switch.

Technique 3. Given an arbitrary Transliterative Function, T, form the related Decoding Function, T_d, and Encoding Function, T_e. Realize these functions, respectively by Technique 1 and Technique 2. Then connect the outputs of the switch for T_d to the inputs of the switch for T_e, joining wires with corresponding labels.

The Switches constructed by the application of this and the preceding Techniques may often be simplified. Some of the ways in which this may be done are given by the following example.

Consider the function T defined below:

TABLE I

d	T	T(d)
1111, 1100	→	110 000
1110, 1010	→	001 000
1101	→	010 000
1001, 0110, 0100	→	000 110
0000	→	000 001
1000	→	001 110

Applying Technique 3 will yield the Switch of Figure 8a (if a Tree is used to realize T_d) as is shown below.

The related Decoding Function, T_d, and Encoding Function, T_e, are given in Table II on the following page.

The labeling of the input wires of P and the output wires of M_1, and the construction of the Switches P and M_1, are as discussed in the example following Technique 1. The Switch, M_2, for T_e is constructed by Technique 2. Ten input wires and six Disjunctive Elements are provided. Input and output wires may be labeled in any order (the ordering of the input labels in Figure 8 was chosen merely to facilitate the drawing of the diagram). After the wires have been labeled, M_2 is constructed by connecting the Switch inputs to the Disjunctive Elements as in step (2) of Technique 2. The Switch for T is completed by connecting the input wires of M_2 to the output wires of M_1 : $\bar{w}_i$ to w_i.

TABLE II

$d(T_d)$	T_d	$T_d(d) = d(T_e)$	T_e	$T_e(T_d(d)) = T(d)$
1111	→	1 000 000 000	→	110 000
1100	→	0 100 000 000		
1110	→	0 010 000 000	→	001 000
1010	→	0 001 000 000		
1101	→	0 000 100 000	→	010 000
1001	→	0 000 010 000	→	000 110
0110	→	0 000 001 000		
0100	→	0 000 000 100		
0000	→	0 000 000 010	→	000 001
1000	→	0 000 000 001	→	001 110

The Simplification of this Switch is carried out by a sequence of steps, each illustrated by a figure. Figure 8a represents the switch as constructed by Technique 3 and Figures 8b and 8c present further simplifications.

In the Switches of Figure 8 some of the Elements and wires are drawn in broken lines and some in "x-ed" lines. The former represent the omissions and the latter the additions effected at the stages of simplification represented.

In Figure 8b the simplifications carried out are the elimination of the "Disjunctive Element" with one input and the omission of five Conjunctive Elements, each of which has an input that is not an input of any other element (this is not always possible, as is pointed out in the second paragraph following).

Another type of simplification is illustrated in Figure 8c. Here two Disjunctive Elements contain, as a subset of their inputs, the output of all the Conjunctive Elements with a given input, ⓘ . In this example these Conjunctive Elements may be eliminated and i connected directly into the Disjunctive Elements (since if i is in the "1" state one of the possible combinations of i must be in the "1" state).

Caution must be exercised in the process of simplification. For example, the function T may have in its range the zero sequence. No connections are necessary for the switch to realize this output. Thus an input

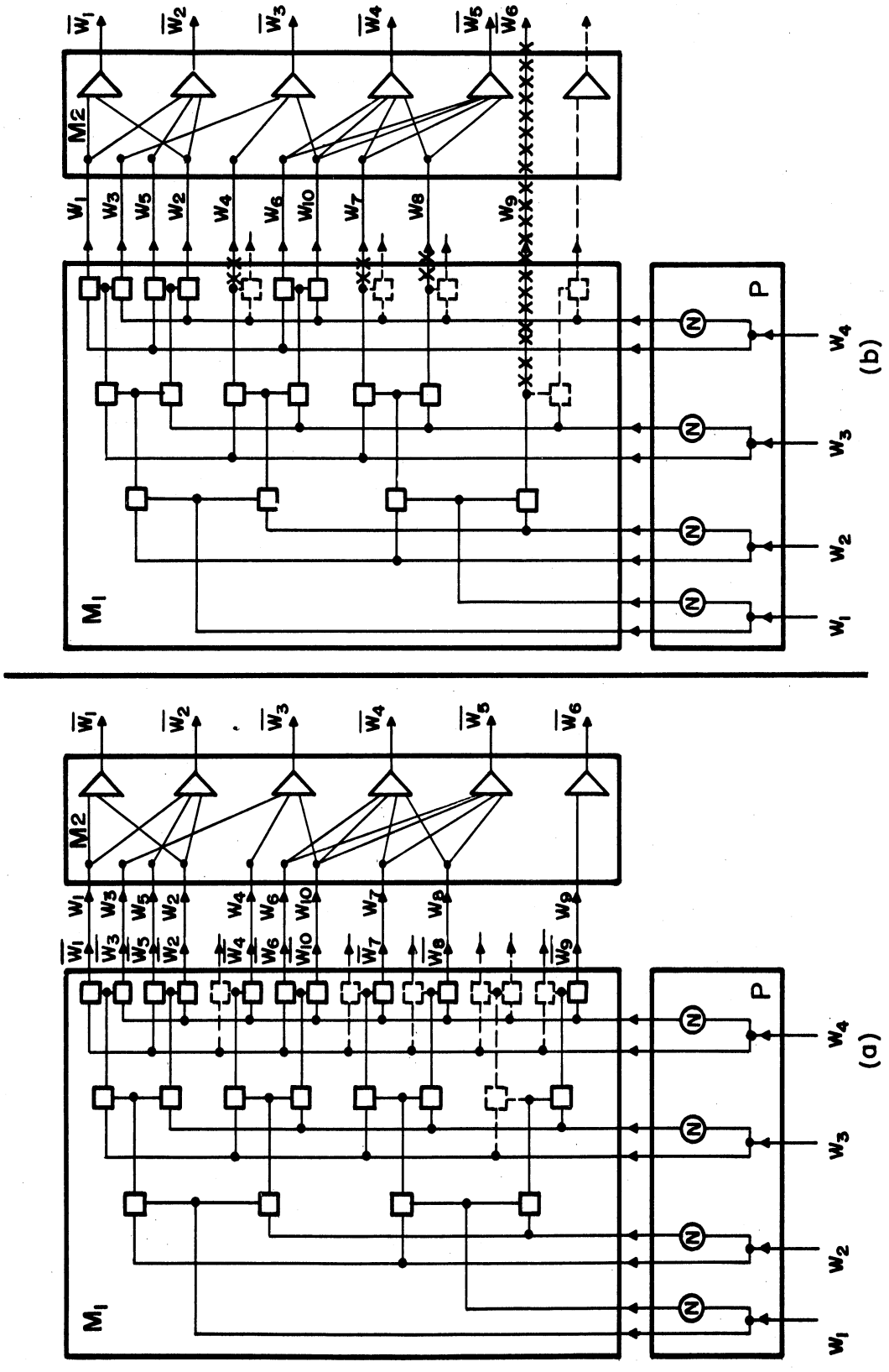


FIGURE 8.

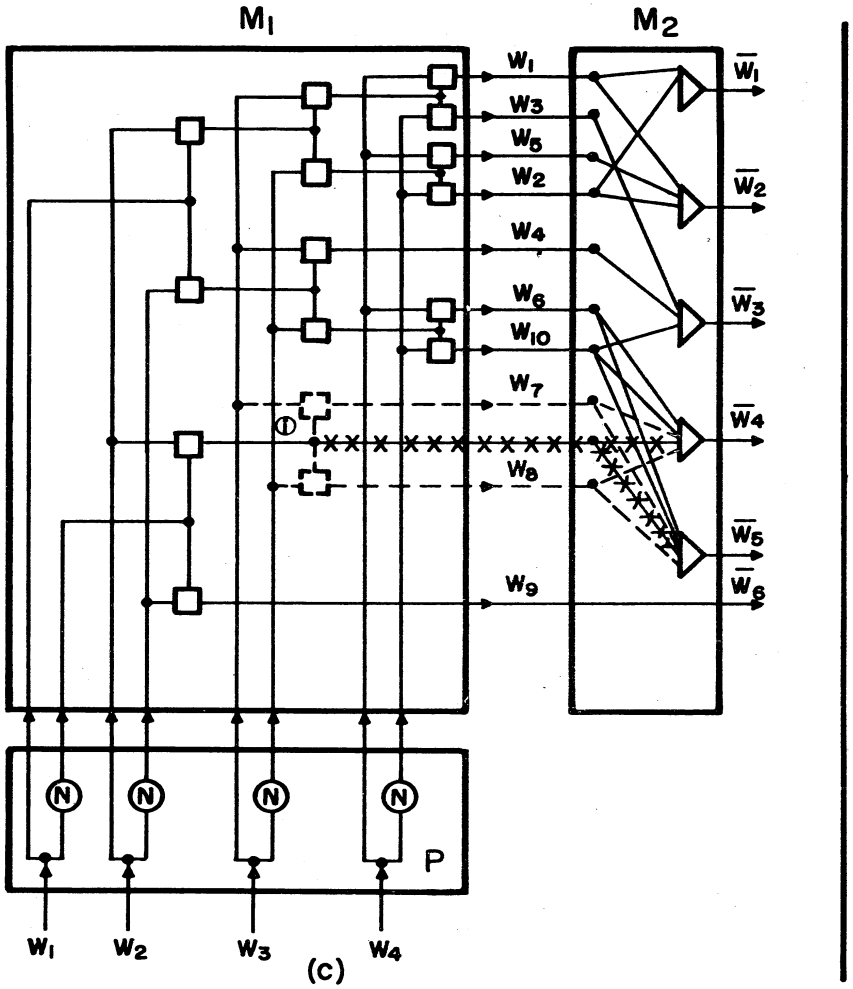


FIGURE 8

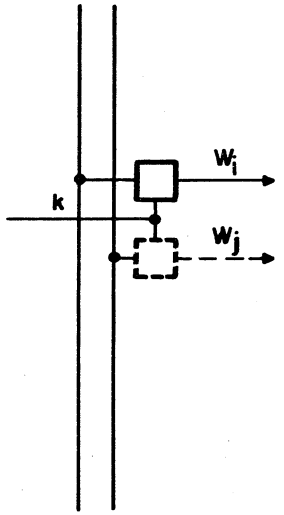


FIGURE 9

to the switch for T_e , the corresponding output of the switch for T_d , and its Conjunctive Element may be eliminated (possibly others, depending on the function). Consider Figure 9. It shows a Double-and in the output MS of the Tree realizing T_d . w_i is in the one state for d_i on the inputs and w_j is in the one state for d_j . Suppose $T(d_j)$ is the zero sequence; then this output and Conjunctive Element may be eliminated. However, although the Element for w_i has an input, k , connected to no other Conjunctive Element, it cannot be eliminated since both d_i and d_j may occur on the inputs. Thus the simplification effected for the switch of Figure 8b is impossible here.

Unfortunately no technique is known for producing the simplest switch for an arbitrary Transliterative Function or even, in terms of this report, a switch with minimal element input count. The application of Boolean Algebra and the methods of Aiken (11) and Veitch (13) are aimed at this problem but are heuristic in nature, not effective as the term is used here. For any given case, further methods of simplification may be obvious, and trial and error procedure is in order.

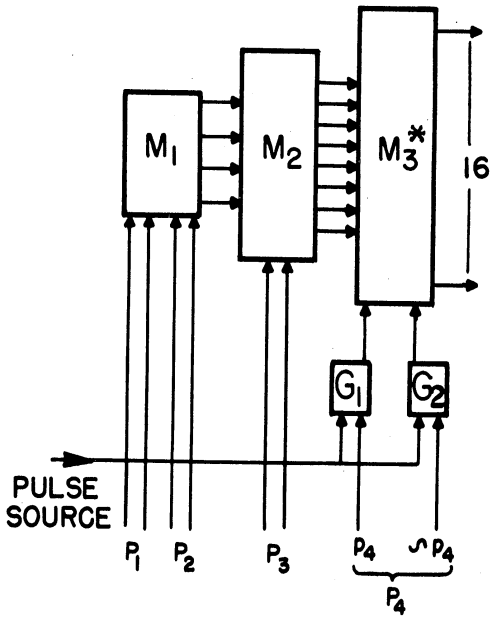
4.4. Modifications to Provide Pulse Output

It has been mentioned before that, whenever such an assumption is required, the switches of this volume are to be considered as static. Primarily this qualification is a precaution against raising questions of pulse-matching whose answers might lie far beyond the scope of this report. It is clear, however, that if equipment is available which is sufficiently flexible to gate pulses with pulses or with static signals without requiring prohibitive conditions of simultaneity, then the preceding theory can be considered applicable to pulse as well as static operation or a combination of the two.

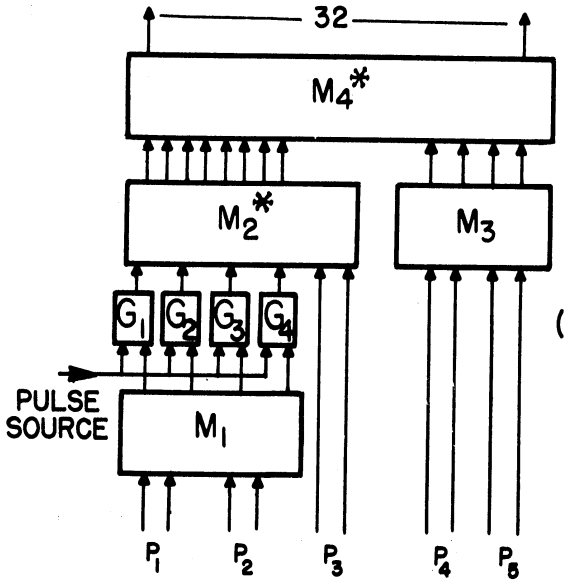
Since the general problem is closely dependent on detailed engineering considerations, the present discussion will be limited to a few simple methods for producing pulse outputs from a system designed for static operation.

An obvious method which can be applied to any static switch whatsoever is to connect a "dynamic gate" (a 2-input conjunction with one static and one pulse input) via its static input to each output of the switch and then connect all of the pulse inputs in parallel to an external source which will produce a pulse whenever the switch is to be "read".

For Simple MS Nets it is possible to introduce the pulse at the input rather than at the output—or, for that matter, at the input (of an entire I_j) of any MS of the Net. This procedure is shown in Figure 10a



(a) TREE
(M_i* DENOTES AN MS EACH OF WHOSE CONJUNCTIVE ELEMENTS HAS 1 PULSE AND 1 STATIC INPUT)



(b) BALANCED MS NET

MODIFICATIONS FOR PULSE OUTPUT
FIGURE 10

for a Tree and in Figure 10b for a Balanced MS Net. Under this method every Chain of Logical Elements in which a pulse is introduced must be composed entirely of dynamic gates from the point of occurrence of the pulse to the output. The method is restricted to Simple MS Nets for two reasons: (1) an element must not have more than one pulse input in order to avoid the difficulty of matching pulses against pulses—this is assured by the restriction against Branching; and (2) it must be possible for the pulse to reach any output—this is assured since every output is an element of the Cartesian Conjunction and therefore is fed by each input.

An effective comparison of the relative desirability of these methods depends ultimately on such considerations as relative cost of dynamic and static gates. The following example, however, may help to clarify the problem. Consider Figure 10b. The method depicted requires four additional dynamic gates and the replacing of forty static gates by dynamic gates —eight in M_2 , thirty-two in M_4 . Represent the "cost" of conversion by $4\overline{G} + 40\Delta\overline{G} - \overline{G}$ representing the "cost" of dynamic gates and $\Delta\overline{G}$ the "cost" of dynamic gates less the cost of static gates. The method is clearly a poor one for this particular switch—assuming that dynamic gates cost more than static gates (i.e., $\Delta\overline{G} > 0$)—since the same effect could be produced, for example, by inserting the pulse at the input pair P_5 at a cost of only $2\overline{G} + 36\Delta\overline{G}$. But if the pulse were introduced at the output of M_2 at cost $8\overline{G} + 32\Delta\overline{G}$ the comparison between this and the other two methods could not be made without knowledge of the ratio of $\overline{G}$ to $\Delta\overline{G}$.

5. Characteristics of Function Switches

The object of this section is to compare in terms of the Element Input Count and loading properties some of the different Switches introduced in sections 3 and 4. The Techniques of section 4 combine an MS Net with, perhaps, a Switch for an Encoding Function and possibly a Polarizer to realize an arbitrary Transliterative Function. The only choice is in the selection of an MS Net which may be an Exponential Switch, a Tree, or a Balanced MS Net. It is primarily these switches which are to be studied.

In the case of a Tree, loading is an obvious problem and a Technique is presented here for designing a Switch—the "folded tree"—closely related to the Standard Tree but having more desirable loading properties. Although its Element Input Count is the same as that of a Standard Tree and its set of outputs is the Cartesian Conjunction of its inputs, it is not necessarily an MS Net and the theory behind it is somewhat different. For these reasons it is discussed in this section rather than as a part of the theory presented in section 3.

5.1. Element Input Count and General Loading Properties

The problem of selecting the "best" switch for a given purpose cannot be dealt with. A switch is "best" only with respect to some well-defined set of properties (probably requiring specifications of actual physical equipment) and usually with respect to a certain class of switches. Two properties, Element Input Count and Loading, have been defined for symbolic Networks which under proper interpretation may have a certain general significance for their physical counterparts. The Element Input Count is considered first.

The Element Input Count of an Exponential Switch is given by the formula $C = m \prod_{i=1}^{m} N(P_i) = m\, 2^m$, where m is the number of input pairs.

For a Standard Tree (with static output) the formula is $C = 8(2^{m-1} - 1)$.

For the Balanced MS Net the only formula available is $C = 2 \sum_{i=0}^{k-1} 2(2^{k-i} + i)$, where the number of input pairs is restricted to $n = 2^k$. However, the three may be compared for various values of m .

Element Input Count Comparison

Switch \ m	2	4	5	8
Exponential Switch	8	64	160	2048
Standard Tree	8	56	120	1016
Balanced MS Net	8	48	96	608

Switches may be judged on the basis of this property alone, the "best" switch being the one with the smallest C . In this sense, the example suggests that the Balanced MS Net is "best", the Exponential Switch worst. Switches "best" in this sense are called "minimal": a switch for a given Transliterative Function is Minimal with respect to a set of switches, S , which realize this function if its Element Input Count is at least as small as that of any other switch in S .

It was stated in section 3 as a theorem and will be proved in Volume III that in the set of all MS Nets whose input consists of m input pairs the Balanced MS Net is Minimal.

The generalization of the idea of minimization to switches for arbitrary Transliterative Functions is very difficult. It is partly at this problem that the methods of Aiken and Veitch are directed. These methods are heuristic and ineffective in the sense that while they are methods for simplifying they do not lead directly to the desired "minimal" switch. This general problem of Minimization is formulated and a possible approach indicated in Volume III.

The second characteristic to be considered is Loading. This property (as defined in section 2) is described by a set of Loading Couples of the form (S,P) , one for each input, where S is the Serial Loading Coefficient and P the Parallel Loading Coefficient. All the Switches considered in this section have the property that both inputs of a pair, P_i , have the same Loading Couple, which will be referred to as the Loading Couple of P_i .

For the Exponential Switch the Loading Couple is $(1,2^{m-1})$ for every input, where m is the number of input pairs. If the input pairs of a Standard Tree are numbered in the "natural" way (i.e., starting with those farthest from the Switch Output), then for the ith input pair $(i>1)$,

it is $(m-i+1, 2^{i-1})$. The couple for $i=1$ is the same as that for $i=2$. Again, no general formula has been developed for the Balanced MS Net because the mode of construction makes it impractical. The Loading Couples for an example are compared below.

Comparison of Loading Couples*

	P_1	P_2	P_3	P_4	P_5
1. Exponential Switch	$(1,2^4)$	$(1,2^4)$	$(1,2^4)$	$(1,2^4)$	$(1,2^4)$
2. Tree	$(4,2)$	$(4,2)$	$(3,2^2)$	$(2,2^3)$	$(1,2^4)$
3. Balanced MS Net	$(3,2)$	$(3,2)$	$(2,4)$	$(2,2)$	$(2,2)$

The Parallel Loading Coefficient of every input of the Exponential Switch is inordinately large for even a moderate number of input pairs though its serial loading is ideal. The Tree also has poor loading characteristics but they can be improved by the "folding Technique" of section 5.2.

While these Couples give some insight into the loading properties of a switch, a more complete picture is given by considering the parallel loading properties of the individual Logical Elements. This can be done easily for any MS Net because of the following:

> Remark: If $I = I_1 \cup \ldots \cup I_m$ is the input set of an MS Net, then the input wires constituting I_j all have Parallel Loading Coefficient L_j $(j=1,\ldots,m)$. Furthermore, all outputs (i.e., Element Outputs) of a given MS of the MS Net have the same Parallel Loading Coefficient.

The truth of this statement derives from the "non-splitting" condition in the definition of MS Net.

Let M be an MS Net. Label the MS's of M : $M_1,\ldots,M_k$ and let the corresponding Parallel Loading Coefficients be $L_1,\ldots,L_k$. Then define L_iM_i to mean that every Element Output of M_i has Loading Coefficient L_i (e.g., $3M_1$ would mean that every element in M_i has Coefficient 3). For the input wires the corresponding notation is K_jI_j ,

* Cf. Figures 3b, 5, and 6 for examples of Switches 1, 2, and 3, respectively. Note, however, that the first two figures are Switches of 4 rather than 5 input pairs.

where K_j is the Parallel Loading Coefficient of each wire in the set I_j. The parallel loading properties of the entire MS Net, M., can then be described by a sum

$$S(M) = \sum_{i=1}^{n} L_i M_i + \sum_{j=1}^{m} K_j I_j \quad . \tag{1}$$

In the cases to be considered the I_j will be Polar Pairs and so will be replaced by P_j. Formula (1) may be thought of as a simplified form of the following sum, which may be defined for an arbitrary Network, A :

$$\bar{S}(A) = \sum_{i=1}^{n'} L'_i E_i + \sum_{j=1}^{m'} K'_j W_j \quad , \tag{2}$$

where E_i ranges over all of the Logical Elements of A, W_j over all its inputs, and L'_i and K'_j are the corresponding Parallel Loading Coefficients. If $A = M$, formula (1) may be derived from (2). Group together the Logical Elements of each M_i of the Network: for example, let $L'_1 E_1 + \ldots + L'_r E_r$ be the expression for M_i. Since this is an MS, $L'_1 = L'_2 = \ldots = L'_r$, represent this common value by L_i and substitute it for the L'_j. Replace the resulting expression, $L_i E_1 + \ldots + L_i E_r$, by $L_i M_i$. The derivation is completed by doing this for each MS of the Network and proceeding similarly for the input wires.

Many variations of formula (2) exist. In this report we are primarily interested in switches whose inputs are Polar Pairs, the two wires of a pair having the same loading coefficient. Consequently, we are interested in formulas of the form

$$\bar{S}(A) = \sum_{i=1}^{n'} L'_i E_i + \sum_{j=1}^{m} K_j P_j \quad . \tag{3}$$

There will be many occasions for referring to the second of these sums, $\sum_{j=1}^{m} K_j P_j$, which is consequently defined to be the Load Distribution of the Switch A. The term, Load Distribution, may be used without reference to a particular switch to refer to any expression of this form (since there will always exist some switch, D, with this as the Load Distribution of D). If S is a given Load Distribution and D is a switch having S as its Load Distribution (there may be many such D) then S is said to be associated with D.

It is sometimes desirable in studying a Network, A, to "break" it into two or more "subnetworks", to consider the Load Distributions of these parts, to add the Load Distributions together when the Networks are combined, and so on. In general Load Distributions may be added together

(when the corresponding switches are combined) as linear expressions. For example, $(3P_1 + 5P_2 + 8P_4 + 1P_3) + (1P_4 + 2P_1 + 5P_3 + 7P_5) = 5P_1 + 5P_2 + 6P_3 + 9P_4 + 7P_5$.

Consider S for the Standard Tree, T , and the Balanced MS Net, B , constructed on 5 input pairs (see Figures 6 and 11, respectively). For the Tree:

$$S(T) = (2M_1 + 2M_2 + 2M_3 + 0M_4) + (2P_1 + 2P_2 + 4P_3 + 8P_4 + 16P_5) .$$

For the Balanced MS Net:

$$S(B) = (2M_1 + 4M_2 + 8M_3 + 0M_4) + (2P_1 + 2P_2 + 4P_3 + 2P_4 + 2P_5) .$$

The Load Distribution of the Tree is:

$$2P_1 + 2P_2 + 4P_3 + 8P_4 + 16P_5 .$$

For the Balanced MS Net it is:

$$2P_1 + 2P_2 + 4P_3 + 2P_4 + 2P_5 .$$

In the case of the Tree the parallel loading is good except for the input pairs with higher subscript (e.g., P_4 and P_5). The loading for the Balanced MS Net is good except for the elements in the MS's near to the output MS (e.g., M_3).

The problem is more acute for the Tree and has serious physical implications. For example, in a relay tree the Parallel Loading Coefficients of the inputs may be interpreted as the spring load on the corresponding relay coils. Because the Load Distribution, $\sum_{j=1}^{m} K_j P_j$, for the Standard Tree has these undesirable properties, the next section considers a method for modifying the Tree Network to give a better Load Distribution.

5.2. The Folded Tree Technique

The folding process for improving Load Distribution and the resulting "folded tree" Network are not new. A folded tree is generally considered to be a modification of a Standard Tree, but in this report a slightly different point of view is taken. The Folded Tree is considered as a distinct and independent Network and is defined as one having as its outputs the Cartesian Conjunction of its inputs and differing from a Standard Tree, if at all, only in the assignment of its Double-ands to input pairs. (A Double-and, A , is assigned to input pair P_k if P_k is connected to the inputs of A .)

By making the assignments in different ways, different Load Distributions may be obtained. The distribution desired in a given case involves consideration of the load limitations of the driving components, the operating characteristics of the Conjunctive Elements of the Network, the available sizes of the conjunctive components if they are in package units, etc.

The remainder of this section is devoted to (1) describing a simple method for determining if a given Load Distribution can be realized in a Folded Tree, and (2) describing a practical method for constructing a Folded Tree with any desired realizable Load Distribution.

Consider a Standard Tree, T , and label its Double-ands iAj , where i specifies the Bay or MS in which the Double-and appears counting from the left, and j the position in the Bay counting from the top. The "subnetwork" whose Double-ands all belong to Chains passing through iAj and have "prescript" $k \geq i$ ("prescript" refers to the k in kAj) is a Network which is itself a Tree. It is called a Minor Tree of T and is denoted T(i,j) .* The Double-and, iAj , is called its Key (see Figure 11). Standard Trees do not themselves have Keys. However, if they are modified for pulse output by inserting dynamic gates at the P_1 input, they will have a Key (see section 4.4).

In the remainder of this section, the words Tree, Standard Tree, and Folded Tree will always refer to Trees with Keys unless otherwise specified. Furthermore, the method of construction given is designed to yield a Folded Tree with a Key. The modifications needed to apply the method to Trees without Keys are indicated at the end of this section.

A Folded Tree may be drawn exactly as a Standard Tree except for the connections of input pairs to Double-ands. If these connections are omitted and the Double-ands labeled as for the Standard Tree, then the definitions and notation for Minor Trees, Keys, and so on may be used in the same manner as for a Standard Tree. This is true because these concepts depend on the "internal" connections of the Network and not on the connections of Double-ands to input pairs. Consequently, in the remainder of this volume the terms Tree, Minor Tree, etc. will refer to Folded Trees (of which the Standard Tree is a special case).

With every Minor Tree, T(i,j) , of T is associated a Load Distribution denoted 'S(i,j)' . Since the Key of T is 1A1 , the Load Distribution of T is denoted 'S(1,1)' .

---

* Included in this definition of a Minor Tree are the Networks containing a single Double-and with outputs which are outputs of T .

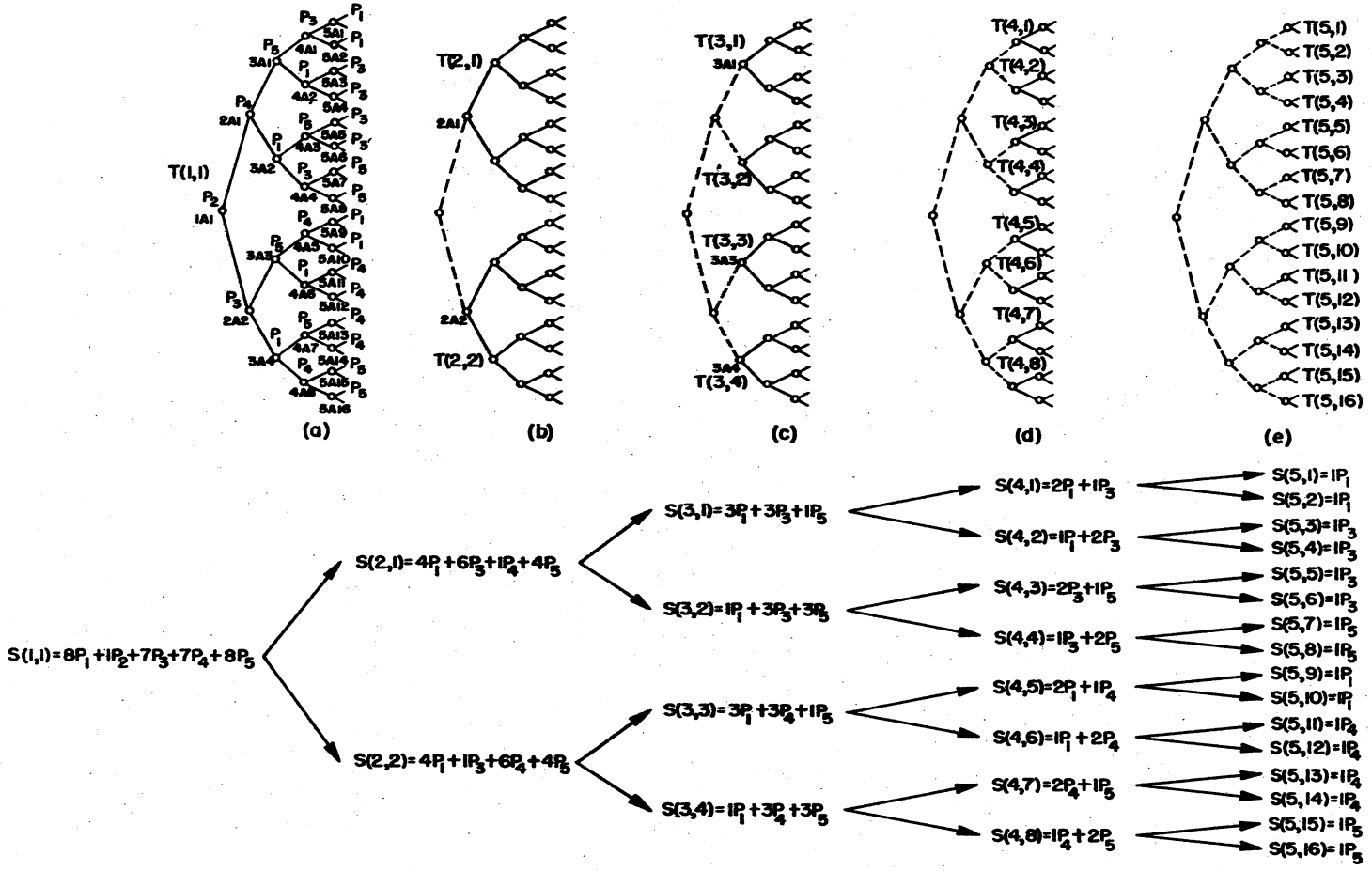

T(1,1)
(a)
T(2,1)
T(2,2)
(b)
T(3,1)
T(3,2)
T(3,3)
T(3,4)
(c)
T(4,1)
T(4,2)
T(4,3)
T(4,4)
T(4,5)
T(4,6)
T(4,7)
T(4,8)
(d)
T(5,1)
T(5,2)
T(5,3)
T(5,4)
T(5,5)
T(5,6)
T(5,7)
T(5,8)
T(5,9)
T(5,10)
T(5,11)
T(5,12)
T(5,13)
T(5,14)
T(5,15)
T(5,16)
(e)
S(1,1)=8P1+1P2+7P3+7P4+8P5
S(2,1)=4P1+6P3+1P4+4P5
S(2,2)=4P1+1P3+6P4+4P5
S(3,1)=3P1+3P3+1P5
S(3,2)=1P1+3P3+3P5
S(3,3)=3P1+3P4+1P5
S(3,4)=1P1+3P4+3P5
S(4,1)=2P1+1P3
S(4,2)=1P1+2P3
S(4,3)=2P3+1P5
S(4,4)=1P3+2P5
S(4,5)=2P1+1P4
S(4,6)=1P1+2P4
S(4,7)=2P4+1P5
S(4,8)=1P4+2P5
S(5,1)=1P1
S(5,2)=1P1
S(5,3)=1P3
S(5,4)=1P3
S(5,5)=1P3
S(5,6)=1P3
S(5,7)=1P5
S(5,8)=1P5
S(5,9)=1P1
S(5,10)=1P1
S(5,11)=1P4
S(5,12)=1P4
S(5,13)=1P4
S(5,14)=1P4
S(5,15)=1P5
S(5,16)=1P5

FIGURE 11.

The Load Distribution of a Tree contains a single term with coefficient 1 . It will be necessary to refer to this term often, and so it is denoted 'U(S(i,j))' which is read 'the term with unit coefficient in S(i,j) ' or 'the unit term of S(i,j) '. These concepts are illustrated in Figure 11, where the P_k above each iAj is the input pair to which iAj is assigned.

Figure 11a is a Folded 5-bay tree with Load Distribution $S(1,1) = 8P_1 + 1P_2 + 7P_3 + 7P_4 + 8P_5$. (A 5-bay Standard Tree has $S(1,1) = 1P_1 + 2P_2 + 4P_3 + 8P_4 + 16P_5$.) Figure 11b shows the two 4-bay Minor Trees of T, T(2,1) and T(2,2) , with Load Distributions $S(2,1) = 4P_1 + 6P_3 + 1P_4 + 4P_5$ and $S(2,2) = 4P_1 + 1P_3 + 6P_4 + 4P_5$, respectively. Clearly, $U(S(2,1)) = 1P_4$ and $U(S(2,2)) = 1P_3$. The assignments of input pairs in Figures 11b, c, d, and e are the same as in 11a.

Method for determining if a desired Load Distribution can be realized in a Folded Tree. A Load Distribution $S = d_1P_1 + d_2P_2 + \ldots + d_nP_n$ is called Admissible if and only if

A.1 $$\sum_{i=1}^{n} d_i = \sum_{i=1}^{n} 2^{i-1} \quad ;$$

A.2 $$\sum_{i=1}^{k} d_i' \geq \sum_{i=1}^{k} 2^{i-1} \qquad k = 1,2,\ldots,n \quad ,$$

where the d_i' refers to the d_i in monotonic non-decreasing order; and

A.3 there is but a single 1 among the d_i .

Using this concept, the following theorem* holds.

---

* The theory behind the methods given here was suggested by Shannon (10), Part II. It is fully developed in Volume III, where the problem of folding and the procedure given here are fully discussed.

Conditions A.1 to A.3 for Admissibility are given by Shannon, and he also proves that any Admissible sequence can be derived by folding (i.e., the "if-" or "sufficing-statement" in the theorem), although he does not give an effective method for doing it. The proof in Volume III that the effective procedure described in this section is a valid one affords a new and constructive proof of the "if statement" of the theorem. The necessity or "only if" statement in the theorem (i.e., any sequence derived by folding is Admissible) does not appear in Shannon's article but is proved in Volume III.

Theorem: A Load Distribution, S , of n terms is the Load Distribution of a Folded n-Bay Tree if and only if S is Admissible.

Thus to decide if a desired Load Distribution can be realized in a Folded Tree it is necessary only to determine its Admissiblity. Admissible Load Distributions are those which can be associated with Folded Trees.

Consider $S(1,1) = 1P_1 + 12P_2 + 12P_3 + 12P_4 + 13P_5 + 13P_6$ as such a desired distribution. Conditions A.1 to A.3 are satisfied for

(1) $1 + 12 + 12 + 12 + 13 + 13 = 63 = \sum_{i=1}^{6} 2^{i-1}$ satisfying A.1.

(2) The partial sums of S computed in monotonic order are 1, 13, 25, 37, 50, 63 and those of the powers of 2 are 1, 3, 7, 15, 31, 63 . Comparison shows that condition A.2 is satisfied.

(3) S(1,1) has a single 1 satisfying A.3.

Hence S(1,1) is Admissible and so can be realized in a Folded Tree.

These conditions can be applied to classes of Load Distributions to determine if the members of the class can all be realized in Folded Trees. For example, consider the set of Load Distributions, $\{\underline{S}(1,1)_n\}$. The 'n' indicates the number of input pairs and the lower bar denotes the Load Distribution which, for the given n , has its maximum coefficient as small as possible. There is just one such distribution for each n and it is obviously an important one. The Load Distributions of this class are given by the following rules:

(1) Let $1,d,d,\ldots,d,\bar{d},\ldots,\bar{d}$ be the n coefficients of $\underline{S}(1,1)_n$ (we are not interested in the correspondence between Loading Coefficients and input pairs here and so consider only the coefficients).

(2) Determine d from $d + F = (2^n-2)/(n-1)$, where d is the integral and F the fractional part of the quotient. If the denominator of F is (n-1) then the numerator of F gives the number of $\bar{d}$'s in the sequence, and $\bar{d} = d+1$.

The distribution, S(1,1) , considered in the preceding example is the element of $\{\underline{S}(1,1)_n\}$ for n=6 , for $d + F = (64-2)/(6-1) = 12-2/5$; d=12, $\bar{d}$=13 and the number of 13's is 2 .

Although the Load Distribution for this example is Admissible, it is not obvious that every Load Distribution in the set, $\{\underline{S}(1,1)_n\}$,

is Admissible. However, there is the following:

Theorem. The Load Distributions of $\{\underline{S}(1,1)_n\}$ are Admissible.

Proof. Conditions A.1 and A.3 hold obviously for all $\underline{S}(1,1)_n$. A.2 can be shown to hold by assuming that there exists a $k<n$ at which it fails. This can be shown to imply that it fails at $k=n$ (i.e., that A.1 fails), but this is impossible. Hence A.2 is satisfied. Q.E.D.

Constructing a Folded Tree with a given Admissible Load Distribution. Since a Folded Tree differs from a Standard Tree with the same number of inputs only in the way in which its inputs are connected, a Folded Tree with Admissible Load Distribution $S(1,1)$ may be constructed by first building a Standard Tree with the appropriate number of input pairs but without giving the connections of the Double-ands to these input pairs; second, determining from $S(1,1)$ the assignment of the iAj's to the inputs; and, third, making the connections indicated.

Clearly the major problem is the determination of the assignment of the iAj. For a given Load Distribution, alternative assignments will in general be possible. The procedure here presented leads, somewhat arbitrarily, to one possible assignment. The bare method is presented here, the theory behind it being reserved for Volume III. The method may be considered as falling into two parts. The first is the determination of the Load Distributions, $S(i,j)$, for every Minor Tree of the Folded Tree $T(1,1)$ being constructed. The second is the determination of the assignment of iAj from $S(i,j)$.

Consider part I. We are given an Admissible Load Distribution and are to determine from it the Load Distributions of the Minor Trees of $T(1,1)$. This is accomplished by repeated application of a process called "splitting".* If $S(i,j)$ is the Load Distribution of $T(i,j)$, a Minor Tree of $T(1,1)$, splitting will yield $S(i+1,2j-1)$ and $S(i+1,2j)$, the Load Distributions of $T(i+1,2j-1)$ and $T(i+1,2j)$, the Minor Trees into which $T(i,j)$ divides. Table III is the work sheet for a specific example showing how successive splitting of Load Distributions beginning with $S(1,1)$ will yield all the $S(i,j)$. Load Distributions with labels in the first column and with the unit term deleted (the reason for this will be apparent later) are those to be split; the two into which it splits appear immediately below. Each $S(i,j)$ appears twice, first with label in

* Not to be confused with the Splitting of MS outputs discussed in section 3.

the second column as the product of a splitting and then with label in the first as a Load Distribution to be split. Figure 11 shows the same example diagrammatically. Figure 11a is the Folded Tree being constructed with the assignments indicated (by the P_k above iAj —these assignments would not be known, of course, until the entire process has been completed). Figures 11b-e show its decomposition into Minor Trees. Below are the corresponding S(i,j) —the arrows indicating how the splitting took place. The S(i,j)'s shown in Figure 11 were calculated on the work sheet and used to determine the assignments given in Figure 11a. They are given there so that the relation of T(i,j) , S(i,j) , and iAj may be seen.

Consider now the following procedure for splitting. It is convenient to think of it as consisting of three steps:

Step One. Delete U(S(i,j)) from S(i,j) . (In the Table, these reduced Distributions, which are the ones to be split, are labeled S(i,j)* in column 1, the two determined by the split are labeled in column 2, and appear in the immediately following lines.)

Step Two. Consider the coefficients of the terms of S(i,j)* in monotonic increasing order. Suppose there are r terms: denote the coefficients in their monotonic order by '$a_1,a_2,\ldots,a_r$' and denote the corresponding coefficients of S(i+1, 2j-1) by '$b_1,\ldots,b_r$' and those of S(i+1,2j) by '$c_1,\ldots,c_r$' .

Step Three. Determine the b_i and c_i by the following rules:

(A) If $a_1 = 2$ then $b_i = \left[\frac{a_i + \Delta_i}{2}\right]$‡ and $c_i = \left[\frac{a_i + 1 - \Delta_i}{2}\right]$ for all i , where $\Delta_i = \sum_{j=1}^{i-1} (c_j - b_j)$.

(B) If $a_1 > 2$ then $b_1 = a_1 - 1$, $b_2 = 1$; $c_1 = 1$, $c_2 = a_2 - 1$; and b_i and c_i are given by the formulas of (A) for $i > 2$.

The idea behind these formulas is quite simple and they are easy to apply after a very little practice. It is: (1) to provide for the '1' in each set of coefficients, and (2) to split the a_i , $i > 2$, in such a way that the sum of the "b" coefficients determined through i is as

---

* The asterisks (stars) appearing in Steps One and Two are part of the symbolism.

‡ The brackets represent the greatest integer in the contained quantity as before.

nearly equal as possible to the sum of the corresponding "c" coefficients with the somewhat arbitrary proviso that at every step $\Delta_i \geq 0$. It is worth pointing out that in actual calculation the work is even simpler than it appears from this description, because all terms (except possibly the first three) are split as evenly as possible. If a_j is even, then $b_j = c_j = a_j/2$. If a_j is odd, then consider Δ_j : if it equals 0 let the larger part of a_j be c_j ; if it equals 1 let it be b_j . This does not mean that it is necessary to compute the sums each time; it is only necessary to keep track of the difference (which can be only 0 or 1 for $j \geq 3$).

The method is now applied to splitting $S(1,1)$ of the example into $S(2,1)$ and $S(2,2)$.

Step One. Delete $U(S(1,1))$: $S(1,1)^* = S(1,1) - U(S(1,1)) = 8P_1 + 7P_3 + 7P_4 + 8P_5$.

Step Two. Assign the labels 'a_i' so that the i's represent the monotonic order of the coefficients of $S(1,1)^*$:

$$\begin{array}{cccc} a_4 & a_1 & a_2 & a_3 \\ 8P_1 & +7P_3 & +7P_4 & +8P_5 \\ b_4P_1 & +b_1P_3 & +b_2P_4 & +b_3P_5 \\ c_4P_1 & +c_1P_3 & +c_2P_4 & +c_3P_5 \end{array}$$

Step Three. Calculate b_i and c_i : Since $7 = a_1 > 2$, we apply rule (B)

$$b_1 = a_1 - 1 = 6; \quad b_2 = 1; \quad c_1 = 1; \quad c_2 = a_2 - 1 = 6;$$

$$b_3 = \left[\frac{8 + (7-7)}{2}\right] = 4; \qquad c_3 = \left[\frac{8+1-(7-7)}{2}\right] = 4;$$

$$b_4 = \left[\frac{8+(11-11)}{2}\right] = 4; \qquad c_4 = \left[\frac{8+1-(11-11)}{2}\right] = 4.$$

Repetition of this 3-step procedure will yield all $S(i,j)$. Table III illustrates a systematic work sheet for recording the calculations. The two sequences determined above appear on lines 2 and 3 respectively. After practice all the necessary calculations can be performed mentally and the results written directly onto the sheet.

Now consider part II, the assignment of the Double-ands of $T(1,1)$ to input pairs. The rule is simple: Assign iAj to U(S(i,j)) .

Thus if the assignment of 2A2 is desired, consider $S(2,2) = 4P_1 + 1P_3 + 6P_4 + 4P_5$. Clearly $U(S(2,2)) = 1P_3$ and so by this rule 2A2 is assigned to P_3 . The assignments of all the iAj of the example are given in Diagram 11a.

The Folded Tree, $T(1,1)$, constructed by this method will have a Key. The following Technique summarizes the procedure and indicates, parenthetically, the modifications needed if a Key is lacking.

Technique 4.

1. Test the desired Load Distribution for Admissibility. If it is Admissible, call it $S(1,1)$ and proceed to step 2. (If the Load Distribution has no unit term, it may be an Admissible Load Distribution for a Tree without a Key. There must, however, be at least one term with coefficient 2 . Choose one of these, say $2P_k$, replace the 2 by a 1 , and apply the criteria to the resulting Load Distribution.)

2. Construct a Standard Tree with Key adjoined (unless the Key is lacking as indicated above) with the appropriate number of input pairs but without connecting them to Double-ands.

3. Determine the assignments of the iAj by:

   a. Generating the set of $S(i,j)$ from $S(1,1)$ and

   b. Assigning iAj to $U(S(i,j))$. (If $T(1,1)$ does not contain a Key, then $S(1,1)$, reduced by eliminating $2P_k$, is defined as $S(1,1)^*$ and handled accordingly. In all other respects the method is unchanged.)

4. In the Standard Tree constructed in step 2 make the connections indicated in step 3.

TABLE III

Column 1	Column 2	P_1	P_2	P_3	P_4	P_5	Column 1	Column 2	P_1	P_2	P_3	P_4	P_5
S(1,1)	S(1,1)	8	1	7	7	8	S(4,2)*				2		
	S(2,1)	4		6	1	4		S(5,3)			1		
	S(2,2)	4		1	6	4		S(5,4)			1		
S(2,1)*		4		6		4	S(4,3)*				2		
	S(3,1)	3		3		1		S(5,5)			1		
	S(3,2)	1		3		3		S(5,6)			1		
S(2,2)*		4			6	4	S(4,4)*						2
	S(3,3)	3			3	1		S(5,7)					1
	S(3,4)	1			3	3		S(5,8)					1
S(3,1)*		3		3			S(4,5)*		2				
	S(4,1)	2		1				S(5,9)	1				
	S(4,2)	1		2				S(5,10)	1				
S(3,2)*				3		3	S(4,6)*					2	
	S(4,3)			2		1		S(5,11)				1	
	S(4,4)			1		2		S(5,12)				1	
S(3,3)*		3			3		S(4,7)*					2	
	S(4,5)	2			1			S(5,13)				1	
	S(4,6)	1			2			S(5,14)				1	
S(3,4)*					3	3	S(4,8)*						2
	S(4,7)				2	1		S(5,15)					1
	S(4,8)				1	2		S(5,16)					1
S(4,1)*		2											
	S(5,1)	1					Total number of '1's'		8	1	7	7	8
	S(5,2)	1											

* These are S(i,j) without the term with unit coefficient.

BIBLIOGRAPHY

1. "Standards of Electronic Computers—Definition of Terms, 1950", The Institute of Radio Engineers, 1 East 79th Street, New York 21, N.Y.

2. Keister, Wm., "Logic of Relay Circuits", Trans. AIEE (Part I), 68, 571-576 (May, 1949); Ritchie, A.E., "Sequential Aspects of Relay Circuits", Trans. AIEE (Part I), 68, 577-581 (May, 1949); and Washburn, S.H., "Relay 'Trees' and Symmetric Circuits", Trans. AIEE (Part I), 68, 582-586 (May 1949).

3. Buffery, G.H., "A Contribution to the Algebra of Relay and Switch Contacts", Proc. IEE (Part I), 97, 357-363 (November, 1950).

4. Montgomerie, G.A., "Sketch for an Algebra of Relay and Contactor Circuits", Proc. IEE (Part III), 95, 303-312 (July, 1948).

5. Lewis, I.A.D., "A Symbolic Method for the Solution of Some Switching and Relay-Circuit Problems", Proc. IEE (Part I), 98, 181-191 (May, 1951).

6. Shannon, Claude E., "A Symbolic Analysis of Relay and Switching Circuits", Trans. AIEE, 57, 713-723 (1938).

7. Rjachman, J., "The Selective Electrostatic Storage Tube", RCA Rev. (No. 1), 12 (March, 1951).

8. Brown, D.R., and Rochester, N., "Rectifier Networks for Multiposition Switching", Proc. IRE, 37, 139 (1949).

9. Synge, J.L., "The Fundamental Theorem of Electrical Networks", Quart. App. Math. (No. 2), 9, 113-127 (July, 1951).

10. Shannon, C.E., "The Synthesis of Two-Terminal Switching Circuits", Bell System Tech. Jour. (No. 1), 28, 59-98 (January, 1949).

11. Staff of Computation Laboratory, "Synthesis of Electronic Computing and Control Circuits", Annals of Computation Laboratory of Harvard Univ., 27, Harvard Univ. Press, Cambridge Mass., 1951.

12. Keister, Wm., Ritchie, A.E., and Washburn, S.H., Synthesis of Switching Circuits, D. Van Nostrand Co., Inc., New York, 1951.

13. Veitch, Edward W., "A New Method of Logical Equation Minimization", Burroughs Adding Machine Co., Research Division (unpublished).

INDEX OF TERMS

The page numbers refer to occurrences of the terms where they are defined or explained.

Term	Page
Admissible	46
Bay	18
Bits	iv
Branching	15
Cartesian Conjunction	10
Cartesian Product	9
Chain	7
Chain, Conjunctive	7
Character	v
Character, basic	iv
Character-token	v
Character-type	v
Coefficient, Parallel Loading	7
Coefficient, Serial Loading	7
Construction Formula	15
Conversion, code	viii
Conversion, code-and-format	ix
Conversion, format	ix
Conversion, language	ix
Double-and	5
Element, Conjunctive	3
Element, Disjunctive	3
Element Input Count	6
Element, Logical	3
Element, Negation	3
Element, primitive	3
Function, Decoding	27
Function, Decoding, Complete	27
Function, Encoding	30
Function, message transliteration	vii
Function, Stroke	22
Function, Stroke, Conjunctive	22
Function, Stroke, Disjunctive	22
Function, translation	vi
Function, Transliterative	26
Input	3
Language	v
Load Distribution	42
Loading Couple	40
Minimal	40
MS (Multiplicative Switch)	11
MS Net	14
MS Net, Balanced	19
MS Net, Simple	15
Natural Outputs	6
Network	5
Output	3
Polar Pair	13
Polarized	13
Polarizer	13
Polar Sequence	26
Pulse output	36
Realization	26
Splitting (Folded Tree)	48
Splitting (of MS output)	15
Switch	6
Switch, Conjunctive	9
Switch, Exponential	13
Switch, Function	1
Switch, Multiplicative (MS)	11
Switch, unique	19
Technique	27
Tree, Folded	43
Tree, Minor	44
Tree, Standard	16
U, u Union (Set Theoretic)	
Wire	3
X Cartesian Product or Cartesian Conjunction	