Carry-Select Adder*

O. J. BEDRIJ†, MEMBER, IRE

Summary—A large, extremely fast digital adder with sum selection and multiple-radix carry is described. Boolean expressions for the operation are included. The amount of hardware and the logical delay for a 100-bit ripple-carry adder and a carry-select adder are compared.

The adder system described increases the speed of the addition process by reducing the carry-propagation time to the minimum commensurate with economical circuit design. The problem of carry-propagation delay is overcome by independently generating multiple-radix carries and using these carries to select between simultaneously generated sums.

In this adder system, the addend and augend are divided into subaddend and subaugend sections that are added twice to produce two subsums. One addition is done with a carry digit forced into each section, and the other addition combines the operands without the forced carry digit. The selection of the correct, or true, subsum from each of the adder sections depends upon whether or not there actually is a carry into that adder section.

Introduction

TO INCREASE the speed of a digital computer, a designer may either develop faster electronic components and circuitry or organize slower components into faster, more efficient over-all systems. In the latter method, increased speed is obtained by combinating available electronic components into more complex logical structures. These structures, unless used with more subtle algorithms for executing basic operations, often require large amounts of additional equipment.

In digital adders, the speed of the addition is limited by the time required to propagate a carry through the adder. The sum for each bit position in an elementary adder is generated sequentially (starting at the lowest order bit position) only after the previous bit position has been summed and a carry (if generated) propagated into the next position.

Improved adders generate carries simultaneously [1, 2]. These adders employ the principle that the carry from each bit position may be generated independently as an explicit function of all the less significant addend and augend bits. However, because of the inherent limitations in available components, the construction of simultaneous carry-generation adders is not always practical. This paper describes a fast digital adder that derives its speed from a complex logical structure without requiring an excessive amount of additional hardware.

* Received September 21, 1961; revised manuscript received March 26, 1962.

† Development Laboratories, Data Systems Division, IBM Corporation, Poughkeepsie, N. Y.

Notation

To facilitate an understanding of the system, Boolean notation of the sort described by R. K. Richards [3] will be used. Note, however, the functional symbols:

$\forall$ = EXCLUSIVE OR

$\oplus$ = Either EXCLUSIVE or INCLUSIVE OR.

Addend, augend and true sum digits are designated by A, B, and S, respectively, followed by a subscript to indicate a digital position. Carries are indicated by C and a subscript to indicate the digital position from which the carry is generated. An S or C followed by a y or n subscript indicates that the sum or carry is provisional and is generated under one of the following assumptions:

y—There was a carry into the lowest-order bit position of the section.

n—There was no carry into the lowest-order bit position of the section.

The absence of an n or y subscript indicates a true sum or a true carry.

Basic Theory

To illustrate the principles involved in reducing the delay caused by carry propagation, assume that a 25-bit addend and a 25-bit augend are to be combined to form a 25-bit sum. The adder is first divided into five 5-bit adder sections. Each adder section is in duplicate so that simultaneous additions of the addend and augend may be made, one with a carry and one without a carry (Fig. 1). Two sums are produced by adding the addend and augend digits in each section and by propagating the carries for sum generation sequentially from the lowest to the highest-order bit position of the section. Note, however, that:

1) Five of the adder sections have a carry forced into their lowest-order bit position and five do not.
2) All adder sections operate simultaneously to produce their respective sum and carry bits.
3) Complete circuit duplication is not required (Fig. 2) since the primary functions, $A \forall B$, and AB are used in producing both sums.

In Fig. 1, the circuits labeled K_y and K_n represent 5-bit adder sections operating with and without the forced carry digit, respectively. The OG circuits (insert, Fig. 2) gate out the true sum digits of the adder section. The blocks labeled EQ (Fig. 1) contain the

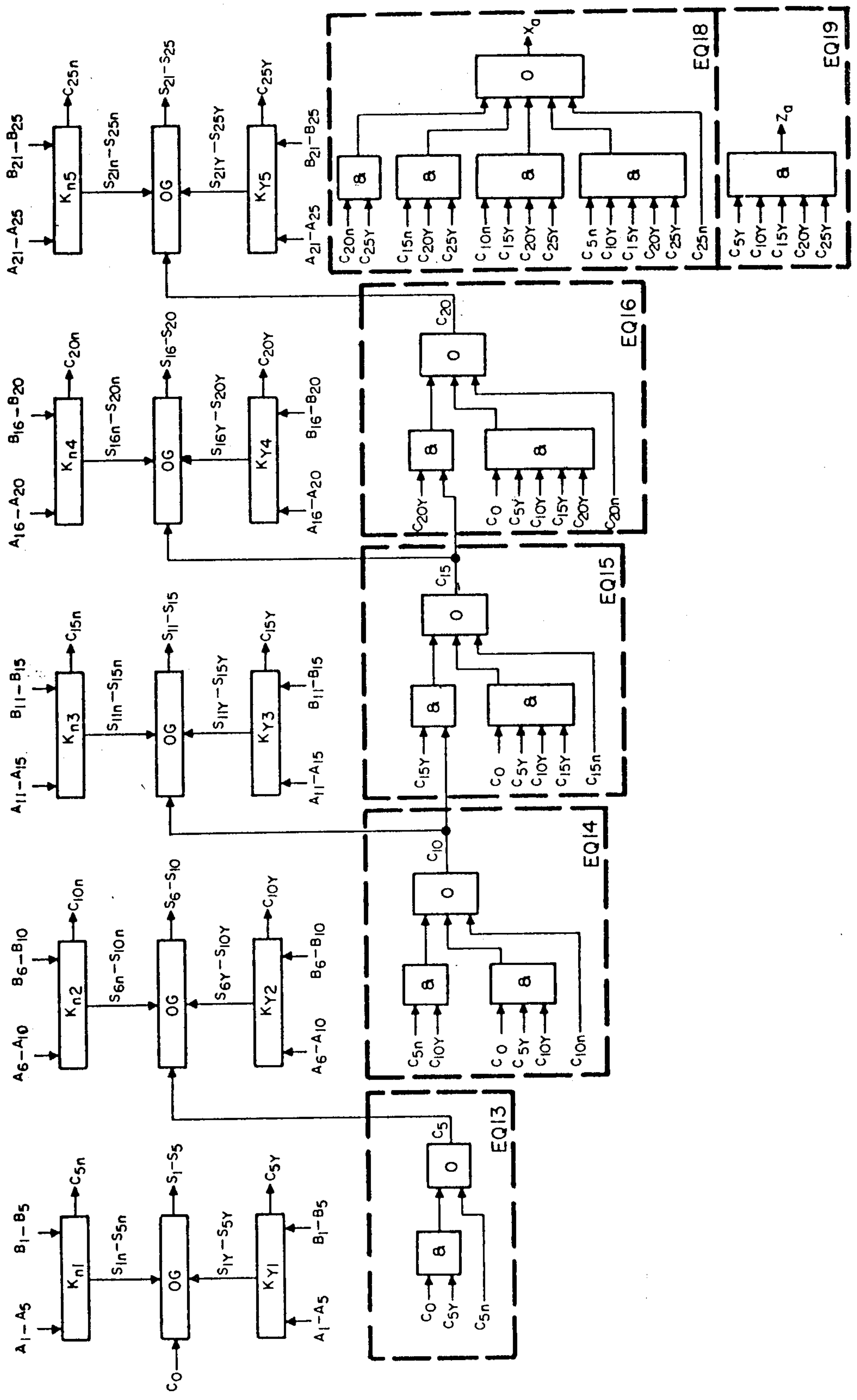

Fig. 1—25-bit adder group.

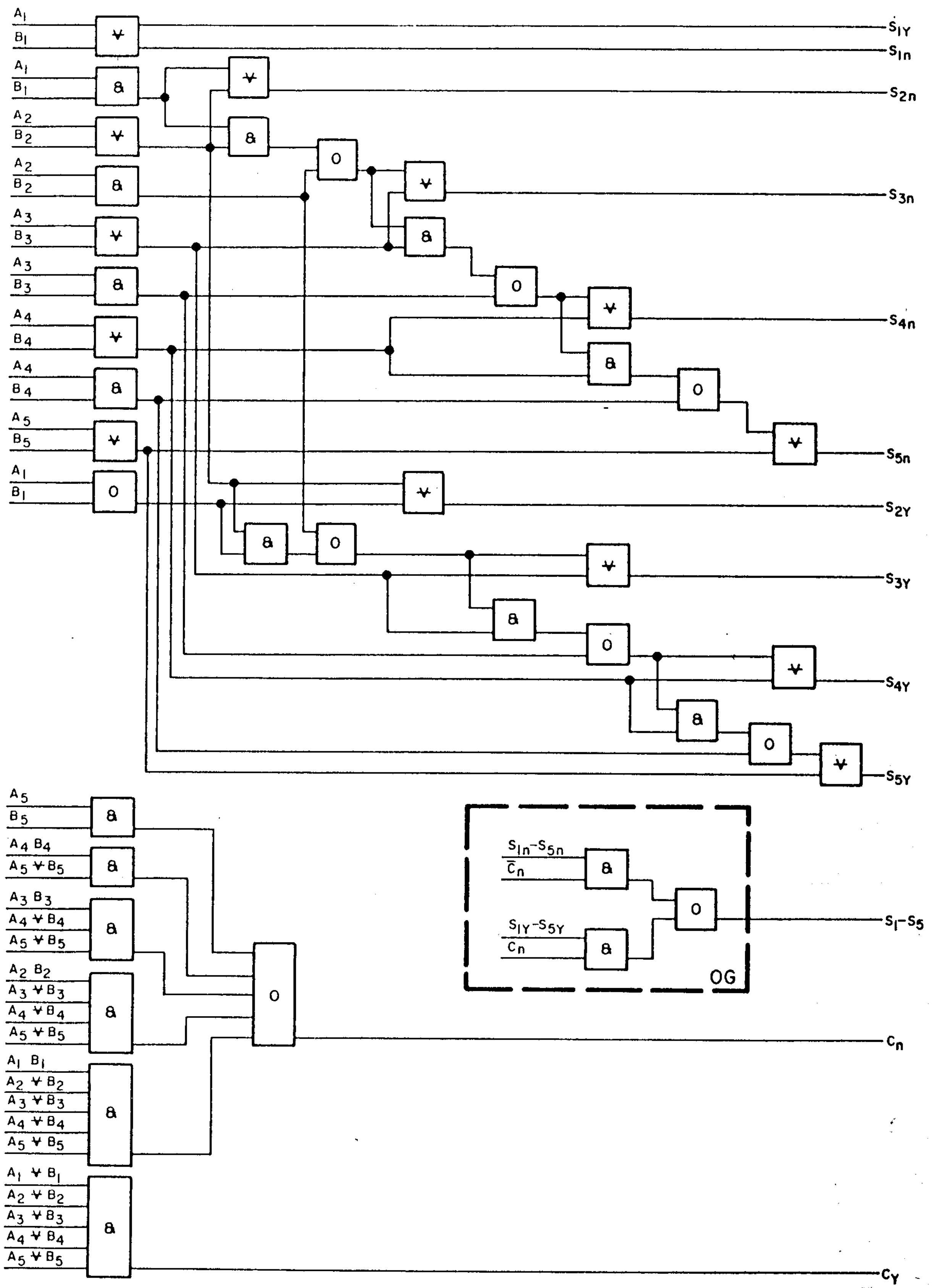

Fig. 2—5-bit adder section.

circuitry necessary to implement the equation identified by the included number.

A detailed drawing of a typical duplicated adder section is shown in Fig. 2. This adder section, which generates five S_y and five S_n sum digits, is a conventional adder with sequential carry propagation. Each adder section produces sums and carries according to the following Boolean equations.

Sums and Carries Produced in Adder Sections with Forced Carry Input

$$S_{1y} = \overline{A_1 \forall B_1} \tag{1}$$

$$S_{2y} = A_2 \forall B_2 \forall C_{1y} = A_2 \forall B_2 \forall (A_1 + B_1), \tag{2}$$

where $C_{1y} = A_1 + B_1$.

$$\begin{aligned} S_{3y} &= A_3 \forall B_3 \forall C_{2y} \\ &= A_3 \forall B_3 \forall [(A_1 + B_1)(A_2 \forall B_2) + A_2B_2], \end{aligned} \tag{3}$$

where $C_{2y} = C_{1y}(A_2 \oplus B_2) + A_2B_2$.

$$\begin{aligned} S_{4y} &= A_4 \forall B_4 \forall C_{3y} \\ &= A_4 \forall B_4 \forall [(A_1 + B_1)(A_2 \forall B_2)(A_3 \forall B_3) \\ &\qquad + A_2B_2(A_3 \forall B_3) + A_3B_3], \end{aligned} \tag{4}$$

where $C_{3y} = C_{2y}(A_3 \oplus B_3) + A_3B_3$.

$$\begin{aligned} S_{5y} &= A_5 \forall B_5 \forall C_{4y} \\ &= A_5 \forall B_5 \forall [(A_1 + B_1)(A_2 \forall B_2)(A_3 \forall B_3) \\ &\quad \cdot (A_4 \forall B_4) + A_2B_2(A_3 \forall B_3)(A_4 \forall B_4) \\ &\quad + A_3B_3(A_4 \forall B_4) + A_4B_4], \end{aligned} \tag{5}$$

where $C_{4y} = C_{3y}(A_4 \oplus B_4) + A_4B_4$.

$$\begin{aligned} C_{5y} &= (A_1 + B_1)(A_2 \oplus B_2)(A_3 \oplus B_3)(A_4 \oplus B_4) \\ &\quad \cdot (A_5 \oplus B_5). \end{aligned} \tag{6}$$

Sums and Carries Produced in Adder Sections Without Forced Carry Input

$$S_{1n} = A_1 \forall B_1 \tag{7}$$

$$S_{2n} = A_2 \forall B_2 \forall C_{1n} = A_2 \forall B_2 \forall (A_1B_1), \tag{8}$$

where $C_{1n} = A_1B_1$.

$$\begin{aligned} S_{3n} &= A_3 \forall B_3 \forall C_{2n} \\ &= A_3 \forall B_3 \forall [A_1B_1(A_2 \forall B_2) + A_2B_2], \end{aligned} \tag{9}$$

where $C_{2n} = C_{1n}(A_2 \oplus B_2) + A_2B_2$.

$$\begin{aligned} S_{4n} &= A_4 \nless B_4 \nless C_{3n} \\ &= A_4 \forall B_4 \forall [A_1B_1(A_2 \forall B_2)(A_3 \forall B_3) \\ &\qquad + A_2B_2(A_3 \forall B_3) + A_3B_3], \end{aligned} \tag{10}$$

where $C_{3n} = C_{2n}(A_3 \oplus B_3) + A_3B_3$.

$$\begin{aligned} S_{5n} &= A_5 \forall B_5 \forall C_{4n} \\ &= A_5 \forall B_5 \forall [(A_1B_1)(A_2 \forall B_2)(A_3 \forall B_3)(A_4 \forall B_4) \\ &\quad + (A_2B_2)(A_3 \forall B_3)(A_4 \forall B_4) \\ &\quad + (A_3B_3)(A_4 \forall B_4) + A_4B_4], \end{aligned} \tag{11}$$

where $C_{4n} = C_{3n}(A_4 \oplus B_4) + A_4B_4$.

$$\begin{aligned} C_{5n} &= A_1B_1(A_2 \oplus B_2)(A_3 \oplus B_3)(A_4 \oplus B_4)(A_5 \oplus B_5) \\ &\quad + A_2B_2(A_3 \oplus B_3)(A_4 \oplus B_4)(A_5 \oplus B_5) \\ &\quad + A_3B_3(A_4 \oplus B_4)(A_5 \oplus B_5) + A_4B_4(A_5 \oplus B_5) \\ &\quad + A_5B_5. \end{aligned} \tag{12}$$

Eqs. (13)–(16) represent the logical expressions for the circuits that select the true subsums for the adder sections. Eq. (17) represents the end-around carry.

$$C_5 = C_0C_{5y} + C_{5n} \tag{13}$$

$$C_{10} = C_0C_{5y}C_{10y} + C_{5n}C_{10y} + C_{10n} \tag{14}$$

$$C_{15} = C_0C_{5y}C_{10y}C_{15y} + C_{5n}C_{10y}C_{15y} + C_{10n}C_{15y} + C_{15n} \tag{15}$$

$$\begin{aligned} C_{20} &= C_0C_{5y}C_{10y}C_{15y}C_{20y} + C_{5n}C_{10y}C_{15y}C_{20y} \\ &\quad + C_{10n}C_{15y}C_{20y} + C_{15n}C_{20y} + C_{20n} \end{aligned} \tag{16}$$

$$\begin{aligned} C_{25} &= C_0C_{5y}C_{10y}C_{15y}C_{20y}C_{25y} + C_{5n}C_{10y}C_{15y}C_{20y}C_{25y} \\ &\quad + C_{10n}C_{15y}C_{20y}C_{25y} + C_{15n}C_{20y}C_{25y} \\ &\quad + C_{20n}C_{25y} + C_{25n}. \end{aligned} \tag{17}$$

The lowest-order adder section has as its true subsum A_{1-5} plus B_{1-5} (without a forced carry) unless an end-around carry exists or a carry from a previous addition has been stored.

Within a carry select adder, the subsum generation path and the carry select path should be approximately equal. No speed advantage is realized by producing subsums before carries are available to select true sums. Therefore, to save components short ripple-carry paths can be used within sections. In the adder discussed above, these paths are not equal because the 25-bit adder was designed as a portion of 100-bit adder.

100-Bit Adder

Theoretically the basic adder shown in Fig. 1 could be extended to include any number of adder sections. However, as the number of adder sections increases, the carry-selection circuits become more costly and complex. In large adders, to retain the advantage of a simple system and to permit the use of carry-select technique, the multiple-radix carry is extended to include higher order radices.

A multilevel adder operates on the same principle as a single-level adder. For example, the first level of a 100-bit multilevel adder consists of a series of 5-bit adders, such as the one shown in Fig. 2. The second level consists of groups of 5-bit adders, such as the 25-bit adder group shown in Fig. 1. Successive levels consist of combinations of two or more groups (Fig. 3).

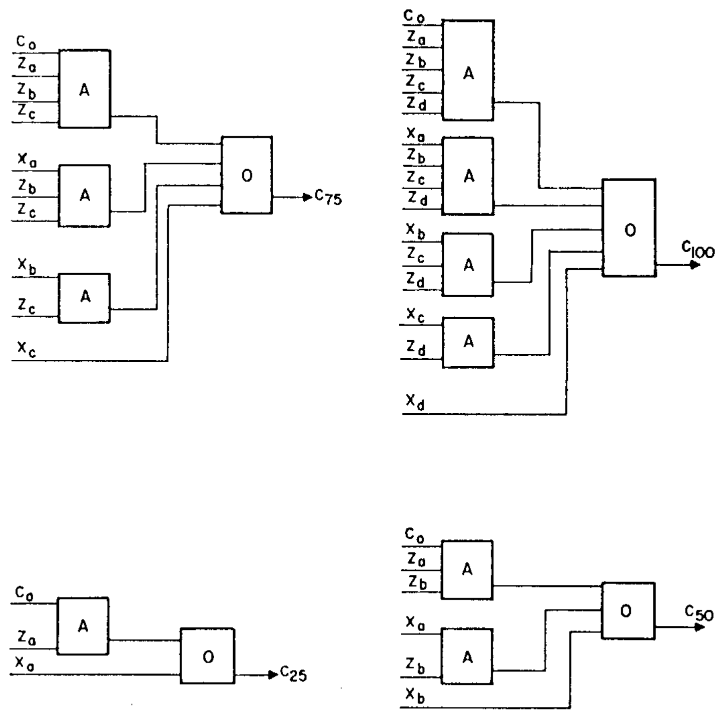

Fig. 3—Combining of group carries.

The multiple-radix carry technique determines:

1) Whether a carry-out will be generated internally within a section or group.
2) Whether a carry-out will be produced by a section or group as a result of a carry being sent into the section or group.

In effect, this technique permits a carry to bypass a section or group.

The several radices are developed by the carries from the sections and groups of an adder. The radix of a carry section is defined by

$$R_{cs} = 2^b$$

where b = number of bits in a section.

Therefore, if an adder section performs an addition that produces a result of 2^5, a carry is produced. The carry radix is 32.

The carry radix from a group (five 5-bit sections) is defined by

$$R_{cg} = 2^{bs}$$

where b = number of bits in a section, s = number of sections in a group.

Group carries X and Z are defined by the following equations (subscript numbers refer to bit position within an adder group):

$$X = C_{5n}C_{10y}C_{15y}C_{20y}C_{25y} + C_{10n}C_{15y}C_{20y}C_{25y} + C_{15n}C_{20y}C_{25y} + C_{20n}C_{25y} + C_{25n} \qquad (18)$$

$$Z = C_{5y}C_{10y}C_{15y}C_{20y}C_{25y}. \qquad (19)$$

The X and Z carries from each 25-bit group are combined with C_0 to produce C_{25}, C_{50}, C_{75} and C_{100} according to the following equations. (The subscripts a, b, c and d identify the 25-bit groups.)

$$C_{25} = C_0Z_a + X_a \qquad (20)$$

$$C_{50} = C_0Z_aZ_b + X_aZ_b + X_b \qquad (21)$$

$$C_{75} = C_0Z_aZ_bZ_c + X_aZ_bZ_c + X_bZ_c + X_c \qquad (22)$$

$$C_{100} = C_0Z_aZ_bZ_cZ_d + X_aZ_bZ_cZ_d + X_bZ_cZ_d + X_cZ_d + X_d. \qquad (23)$$

The carries C_{25}, C_{50}, C_{75} and C_{100} (represented as C_0 in Fig. 1) are returned to the appropriate group where they are used to gate the true subsum from each 5-bit adder section. The formation of carries for a 100-bit adder is shown in Fig. 4. To save hardware during the production of carries C_{25}, C_{50}, C_{75} and C_{100}, the internally generated carry of a group may be ripplied serially (Fig. 1, Eqs. 14–16).

A 100-bit carry-select adder operates in a 6-step sequence:

1) Carries C_y and C_n are generated in each 5-bit adder section.
2) Sums S_y and S_n are produced in each adder section during carry propagation time.
3) Carries X and Z are formed for each 25-bit group by combining carries C_y and C_n within the group.
4) Carries C_{25}, C_{50}, C_{75} and C_{100} are formed by combining X and Z carries.
5) The appropriate carry is returned to each 25-bit group.
6) The true subsum is selected for each 5-bit adder section.

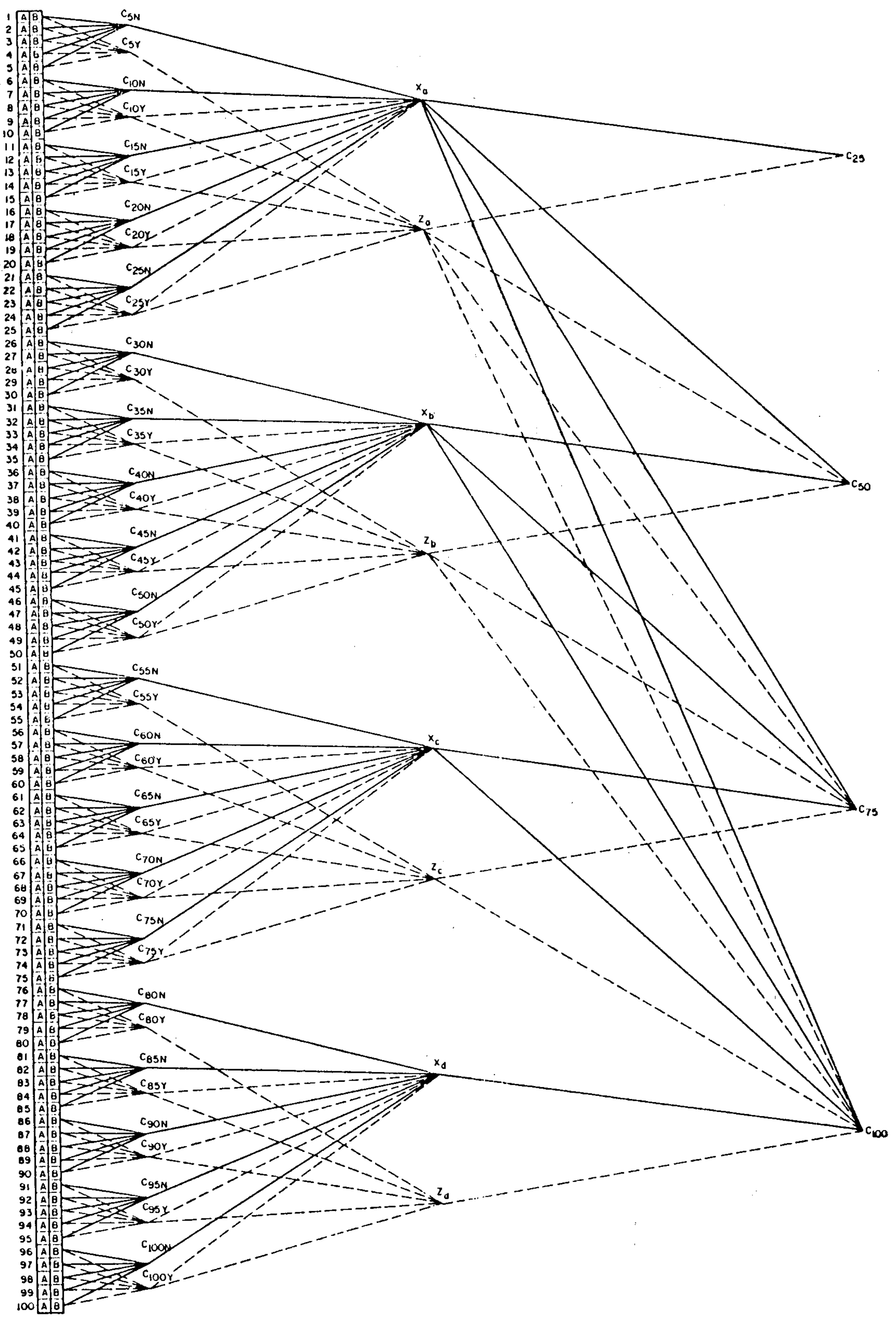

Fig. 4—Carry formation path.

Table I gives a comparison of the speed and the number of components required for a carry-select adder and a simple ripple-carry adder. Hence a speed improvement by a factor of 20 may be achieved with approximately twice the hardware. The breakdown of an adder into the specific section-group arrangement

TABLE I

Adder Type	Bits	Logical Elements	Time in Logical levels
Ripple-carry	100	500	202
Carry-select	100	1122	11

cussed was intended merely to help describe the general carry-select principle. Modifications of this arrangement may readily be realized.

References

[1] A. Weinberger and J. L. Smith, "A one microsecond adder using megacycle circuitry," IRE Trans. on Electronic Computers, vol. EC-5, pp. 67–73; June, 1956.
[2] A. Weinberger and J. L. Smith, "A Logic for High Speed Addition," National Bureau of Standards, Washington, D. C., Circular 591, sec. 1; February, 1958.
[3] R. K. Richards, "Arithmetic Operations in Digital Computers," D. Van Nostrand Co., Inc., New York, N. Y.; 1955.
[4] O. L. Mac Sorley, "High-speed arithmetic in binary computers," Proc. IRE, vol. 49, pp. 67–91; January, 1961.
[5] J. Sklansky, "Conditional sum addition logic," IRE Trans. on Electronic Computers, vol. EC-9, pp. 226–231; June, 1960.

Load-Sharing Core Switches Based on Block Designs*

RICHARD C. SINGLETON†, SENIOR MEMBER, IRE

Summary—Designs for load-sharing zero-noise core switches have been proposed by Constantine, Marcus, and Chien. Blachman has proposed a core memory wiring plan which with modification can be converted to a load-sharing zero-noise switch. An examination of these switch plans shows that they have a common relationship to a class of mathematical structures known to mathematicians and statisticians as balanced incomplete block designs. This relationship is formulated, and it is then shown that all balanced incomplete block designs lead to load-sharing zero-noise switches. Three methods of forming the winding matrix for a switch are given, and expressions for the load-sharing factor, set bias, and reset bias in terms of the balanced incomplete block design parameters are derived for each switch type. Similarly, partially balanced incomplete block designs are shown to lead to low-noise load-sharing switches. Switch operation under fault conditions is briefly discussed.

Most of the known load-sharing core switch types can be viewed as based on either balanced or partially balanced incomplete block designs. A review of the available block designs indicates that a number of new switches can be based on these designs.

A modification of a distributed memory model proposed by C. Rosen is discussed. With wiring plans based on block designs, it appears possible to construct very-large-capacity memory units which are relatively insensitive to wiring errors.

Introduction

BALANCED and partially balanced incomplete block designs have long been a subject for study by mathematicians and statisticians, and this research continues at present. The purpose of this paper is to show the relationship between these designs and load-sharing core switches, so as to make the results on block designs more readily available to designers of core switches and memories.

Balanced incomplete block designs lead to a large class of zero-noise switches; most of the known zero-noise load-sharing switches fall into this class. Similarly, partially balanced incomplete block designs lead to low-noise load-sharing switches. Several examples are given to illustrate these relationships. The application of block designs to distributed memory design is briefly discussed.

It is assumed that the reader is generally familiar with magnetic core access switches and memories; however, the reader may wish to refer to the paper by Minnick and Haynes[1] for additional background.

Core Switch Notation

A magnetic core switch will be regarded here as specified by its core winding plan and a listing of the input patterns to be used for set and reset. From this information, the set and reset excitations received by a selected core can be calculated.[2] The core winding plan of a switch with v cores, (*i.e.*, outputs) and b regular inputs is given by a v-by-$(b+1)$ "winding matrix" $W=(w_{ij})$, where w_{ij} gives the number and direction (positive or negative) of turns on the ith core from the jth input; the $(b+1)$st input is assumed to be used for both set and reset bias. If the input patterns (of *ones* and *zeros*) used to set the switch are arranged as the columns of a $(b+1)$-by-v "selection matrix" C_s, with the entries in the final row equal to the set bias level, then the v-by-v set excitation matrix $X_s=(x_{ij})$ is given

* Received November 20, 1961. This work was supported in part by the Rome Air Force Development Center, United States Air Force.

† Engineering Division, Mathematical Sciences Department, Stanford Research Institute, Menlo Park, Calif.

[1] R. C. Minnick and J. L. Haynes, "Magnetic core access switches," this issue, p. 352.
[2] This representation of a magnetic core switch is due to Minnick, *ibid.*