Arithmetic Microsystems for the Synthesis of Function Generators

ALGIRDAS AVIŽIENIS, MEMBER, IEEE

Abstract—**The continuing reduction of the size and cost of integrated circuit logic elements encourages the utilization of more complex logic nets in digital computers. In arithmetic processors, the application of integrated circuits will permit the replacement of sequential logic nets by their combinational equivalents, and the replacement of programmed subroutines by arithmetic function generators. At the present time, such function generators are used in the Variable Structure Computer at the University of California, Los Angeles. The potentially low cost of logic elements can be realized if large numbers of elements are contained in one package with a limited number of connections to the outside, and if very few types of packages are used in large quantities. Present-day arithmetic processors cannot be readily subdivided into uniform packages because of irregularities in their internal structure. The paper describes the advantages gained by the application of signed-digit number systems in the design of such packages, called arithmetic microsystems. Three types of combinational microsystems are described and their internal design is illustrated by examples for a radix 16 system. The microsystems can accept conventional binary or signed-digit operands. Combinational reconversion of results to the binary system is also discussed. The application of microsystems is illustrated by radix 16 combinational arrays for the operations of multiplication and division.**

I. The Application of Arithmetic Function Generators

THE CLASSIC MODEL of a digital computer contains an arithmetic processor which is organized around a two-operand adder. Recent large-scale computing systems contain much more elaborate arithmetic elements; in some cases, several different one-algorithm processors are contained in a single computing system. The rapid development of integrated circuit technology promises continued reductions in the cost, size, and operation time of logic circuits; it is to be expected that it will soon become economical to replace sequential nets in the arithmetic processor by their combinational equivalents, as well as to replace programmed arithmetic subroutines by arithmetic function generators. An existing system which utilizes such function generators is the Variable Structure Computer at the University of California, Los Angeles [1]. Other application areas for fast function generators are in very large high-speed systems and in special-purpose computers which require various arithmetic functions to be generated at high speed.

One significant obstacle exists in the application of batch-fabricated integrated circuits to the construction of arithmetic function generators. The potentially low cost of logic circuitry can be realized only if one circuit package contains a large number of logic elements with few external connections and if the circuit packages can be manufactured in large quantities [7]. This in turn implies that the various function generators should be assembled from one-package microsystems of very few types; in the ideal case, they would be of one type only. Today's high-speed processors cannot be readily subdivided into identical blocks of many logic elements. The conventional building blocks of an arithmetic processor are flip-flops, one-bit half-adders or full-adders, and combinational gates. Being relatively small with respect to the entire array, they do not impose a requirement for array standardization at a microsystem level.

Currently existing arithmetic processors reveal a great variety of designs and algorithms. In particular, many methods for accelerating carry-propagation are found in parallel adders. The fast parallel adder is the heart of every high-speed arithmetic processor. The processor is a complex logic array consisting of the adder and associated circuitry for shifting, complementing, multiple generation, and sequencing. The main reasons for the great variety of parallel adders are 1) the differing preferences for carry-propagation methods, 2) the existence of two basic subtraction methods ("one's" and "two's" complements), and 3) the wide variation of word length for single precision operands. Irregularity in the structure of the processors is also caused by multiple-precision operations and by diverse sequential algorithms for fast multiplication and division.

The factors mentioned above preclude a direct subdivision of existing processor designs into arrays of microsystems of only one or of very few types. The need for applicability in processors of diverse capabilities establishes the following properties as desirable in the integrated circuit arithmetic microsystems:

1) They are suitable for the construction of fast adders/subtractors for operands of any length.
2) They can be arranged into large arrays which perform multiplication, division, and other arithmetical algorithms (square root, trigonometric functions, logarithms, matrix arithmetic, etc.).
3) The execution times of algorithms can be decreased to the limit of a combinational logic net by the use of more identical microsystems in the array.
4) The number of data input and output lines and control signals into the microsystem package is low.
5) The complexity of the internal logic net can be systematically increased to accommodate increases in the

Manuscript received July 13, 1966; revised August 22, 1966. The research reported here was supported in part by the Office of Naval Research and the Atomic Energy Commission under Contracts Nonr 233(52) and AT(11-1) Gen 10, Project 14.

The author is with the Engineering Department, University of California, Los Angeles, Calif.

number of logic elements which can be placed in a standard package.

The limitations of conventional arithmetic with respect to these properties warrant the consideration of other number representations for digital arithmetic.

II. Signed-Digit Arithmetic

Considerable promise for the development of arithmetic microsystems which meet the preceding specifications is offered by the application of the class of *signed-digit* (abbreviated *s-d*) number systems [2]–[4]. In signed-digit numbers each digit value carries its own sign, instead of the one sign which applies to an entire conventional number; otherwise they possess most of the properties of conventional numbers with a constant, positive radix. Of practical importance are signed-digit number forms with a constant radix $r>2$, in which the allowed digit values are a sequence of $2a+1$ integers:

$$\{\bar{a}, \cdots, \bar{1}, 0, 1, \cdots, a\}; \text{ with } r/2 < a < r. \tag{1}$$

The overbar ($\bar{1}$, $\bar{a}$, etc.) will be employed throughout this paper to designate negative digit values. The number forms in this class are redundant; that is, one number may be represented by more than one form. An addition/subtraction algorithm exists for these forms in which each digit of the result is the function of only two adjacent digits of the operands, regardless of their length.

The two-digit addition algorithm consists of two steps. In the first step, an *interim sum* w_i and a *transfer digit* t_{i-1} are computed from the sum of the input digits x_i and y_i:

$$w_i = x_i + y_i - rt_{i-1} \tag{2}$$

where

$$\begin{aligned} t_{i-1} &= 0 \text{ if } |x_i + y_i| < a \\ t_{i-1} &= 1 \text{ if } x_i + y_i \geq a \\ t_{i-1} &= \bar{1} \text{ if } x_i + y_i \leq \bar{a}. \end{aligned}$$

In the second step, the *sum digit* s_i is obtained by the addition:

$$s_i = w_i + t_i. \tag{3}$$

Since $|w_i| \leq a-1$ and $|t_i| \leq 1$ held above, then $|s_i| \leq a$ and the sum is again a signed-digit number of the same class. It is noted that the addition time (in terms of logic level delays) is the same for operands of any length. Subtraction is performed as the change of all individual digit signs of the subtrahend, followed by an addition. Block diagrams of the *two-digit sum unit* (abbreviated SU), which performs the s-d addition/subtraction algorithm, are shown in Fig. 1. Left-to-right one-origin indexing of digits $(x_1, \cdots, x_n)$ is employed in the figures and discussion; the radix point is assumed to be to the left of x_1 unless otherwise specifically noted.

The conversion of a conventional number to the s-d form can be performed by the standard s-d sum units of Fig. 1. The incoming conventional digit m_i replaces the sum x_i+y_i in (2); the digits w_i and t_{i-1} are generated for every m_i and then the addition (3) yields the s-d form. An example

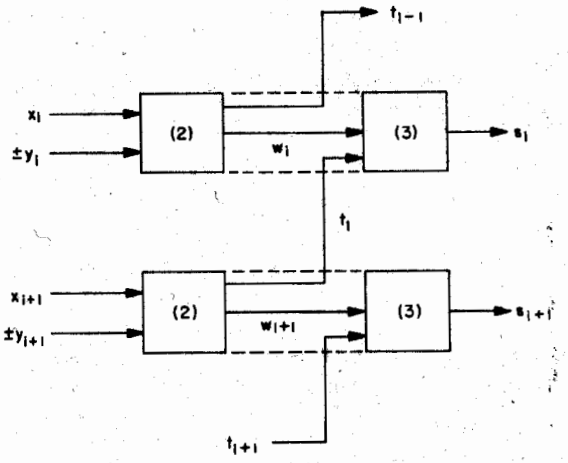

Fig. 1. Two-digit sum units (SU).

for the system with radix $r=10$ and maximal digit value $a=6$ (minimally redundant case) is shown below:

position i	0	1	2	3	4	5
conv. digits m_i	0	2	6	9	5	7
interim sums w_i		2	$\bar{4}$	$\bar{1}$	5	$\bar{3}$
transfers t_i	0	1	1	0	1	
s-d digits s_i	0	3	$\bar{3}$	$\bar{1}$	6	$\bar{3}$.

For reconversion to conventional decimal form, the s-d number is considered to be the sum of a positive and a negative conventional number, and a conventional subtraction is performed:

$$0.3\bar{1}6\bar{3} = 0.30060 + (-0.03103) = 0.26957.$$

Other possible choices of a for $r=10$ as defined by (1) are $a=7, 8, 9$. In the maximally redundant case $a=9$, no conversion is needed. A fast method of reconversion to conventional forms is discussed in Section VI.

The *algebraic value* of the s-d number x is given by the conventional weighted-sum expression; the value *zero* is uniquely represented by the form with all zero digits. The *sign* of x is determined by the sign of the leftmost (most significant) nonzero digit. The *additive inverse* $-x$ of x is formed by changing the individual signs of all nonzero digits. Significant digit-arithmetic [5] can be conveniently implemented by the use of a special *space-zero* digit which designates nonsignificant positions [3], [6]. Arithmetic control is partially localized, since the space-zero digit ϕ can be applied to remove step-counting and the explicit specification of operand length from arithmetic control. In multiple-precision addition and in iterative multiplication and division, the most significant digits of the results are always generated first; consequently, interconnected arrays of arithmetic processors can operate without temporary storage. Implementation of floating-point arithmetic has been studied [3], [4]. No difficulties were discovered which would be due to the use of s-d forms in floating-point algorithms.

The following sections describe several signed-digit arithmetic microsystems, to be called *units*, which have been

developed for application in the UCLA Variable Structure Computer. The desirable properties listed in the preceding section have served as guidelines in the choice among many possible alternatives.

III. The Augmented Two-Digit Sum Unit

The standard two-digit sum algorithm in (2) and (3) accepts two operand digits and produces one sum digit, all in the system of the same redundancy, i.e., the same value of a in (1). It was previously observed that when the maximally redundant form ($a=r-1$) is employed, numbers of the conventional representation can be accepted as the input operands [4]. Further design studies showed, however, that maximally redundant s-d systems contain "pseudonormal" forms, which demand special handling and increase the complexity of floating-point algorithms [6]. A further disadvantage of maximally redundant forms is the wide range ($0 \le |x_i| \le r-1$) of multiplier digit values, which requires a recoding of the multiplier for efficient multiplication.

We observe that minimally redundant forms [$a=(r/2)+1$ for even r; $a=(r+1)/2$ for odd r] can employ a variation of the two-digit adder in which one operand digit (x_i) is minimally redundant, and the other is allowed to be in the maximally redundant or conventional form ($-r+1 \le y_i \le r-1$), but the sum digit is again in minimally redundant (abbreviated min-r.) form. The range of the sum in this case is

$$0 \le |x_i + y_i| \le r + \frac{r}{2}. \tag{4}$$

The conditions $|t_i| \le 1$, $|w_i| \le r/2$, needed for min-r. sums, are satisfied by the range (4). The *augmented two-digit sum unit* (abbreviated ASU), which accepts two min-r. digits (x_i, y_i) as well as a third digit z_i, and produces a min-r. sum digit s_i, is shown in Fig. 2. The algorithm is

$$w_i = x_i + y_i + z_i - rt_{i-1} \tag{5}$$

$$s_i = w_i + t_i. \tag{6}$$

The value of t_{i-1} in (5) is the same as defined in (2), with the value of a chosen for the min-r. system. The digits y_i and z_i taken together may represent a maximally redundant (or conventional) digit value. Since s_i is required to be a min-r. digit, the magnitude limits for the input z_i are as follows:

$$0 \le |z_i| \le \frac{r}{2} - 2; \quad \text{for even } r > 4, \tag{7}$$

$$0 \le |z_i| \le \frac{r-3}{2}; \quad \text{for odd } r > 3. \tag{8}$$

For example, for $r=8$, $|z_i| \le 2$; for $r=10$, $|z_i| \le 3$; and for $r=16$, $|z_i| \le 6$. The three-input adder was selected instead of the simpler two-input configuration because of its compatibility with the two-digit product unit, which is discussed later.

The internal logic design of the ASU and other units will be illustrated using a radix 16 minimally redundant s-d number system, which is compatible with conventional radix 2 or radix 16 input operands. It is evident that the logic design will depend on the encoding chosen for individual digit values. In the following illustrations, the min-r. digit $x_i(\bar{9} \le x_i \le 9)$ is represented by five bits $(\alpha_{16}, \alpha_8, \alpha_4, \alpha_2, \alpha_1)$, while the min-$r$. digit y_i is represented by $(\beta_{16}, \beta_8, \beta_4, \beta_2, \beta_1)$. The positive values are encoded as binary integers, and the negative values are encoded as 32's complements of the positive values. The encodings are shown in Table I. The value ϕ is employed to designate nonsignificant (space) zero values; it is converted to the value zero (00000) before entering the adder. The third digit $z_i(\bar{6} \le z_i \le 6)$ is represented by four bits $(\gamma_8, \gamma_4, \gamma_2, \gamma_1)$, and the negative values are 16's complements. The encodings of z_i are obtained from Table I by removal of the leftmost bit for values $\bar{6}$ to 6 inclusive.

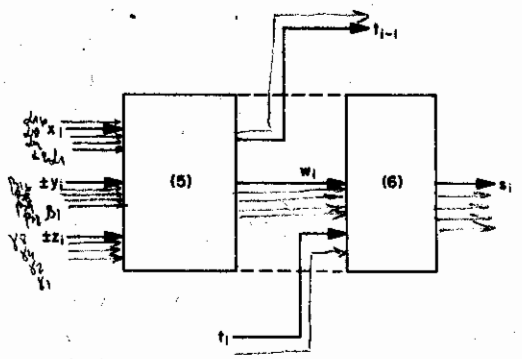

Fig. 2. Augmented two-digit sum unit (ASU).

TABLE I
Digit Encodings for Radix 16

Digit Value	Binary Encoding	Digit Value	Binary Encoding
0	00000	ϕ	10000
1	00001	$\bar{1}$	11111
2	00010	$\bar{2}$	11110
3	00011	$\bar{3}$	11101
4	00100	$\bar{4}$	11100
5	00101	$\bar{5}$	11011
6	00110	$\bar{6}$	11010
7	00111	$\bar{7}$	11001
8	01000	$\bar{8}$	11000
9	01001	$\bar{9}$	10111

The logic design of various units is entirely combinational and will be shown in terms of binary full-adders (denoted as FA) and half-adders (denoted as HA), as well as by special blocks of combinational logic. The FA has three inputs (α, β, γ) and produces two outputs: the sum $s=\alpha \oplus \beta \oplus \gamma$, and the carry $c=\alpha\beta+\alpha\gamma+\beta\gamma$. (The symbol $\oplus$ denotes the EXCLUSIVE-OR operation, $+$ denotes OR, and adjacency denotes AND.) The HA has two inputs (α, β) and produces two outputs: the sum $s=\alpha \oplus \beta$, and the carry $c=\alpha\beta$. The examples emphasize a clear exposition of net structure rather than a minimization of signal delay or component count of the unit. The combinational logic of each unit is intended to be contained in a single integrated circuit package. In the

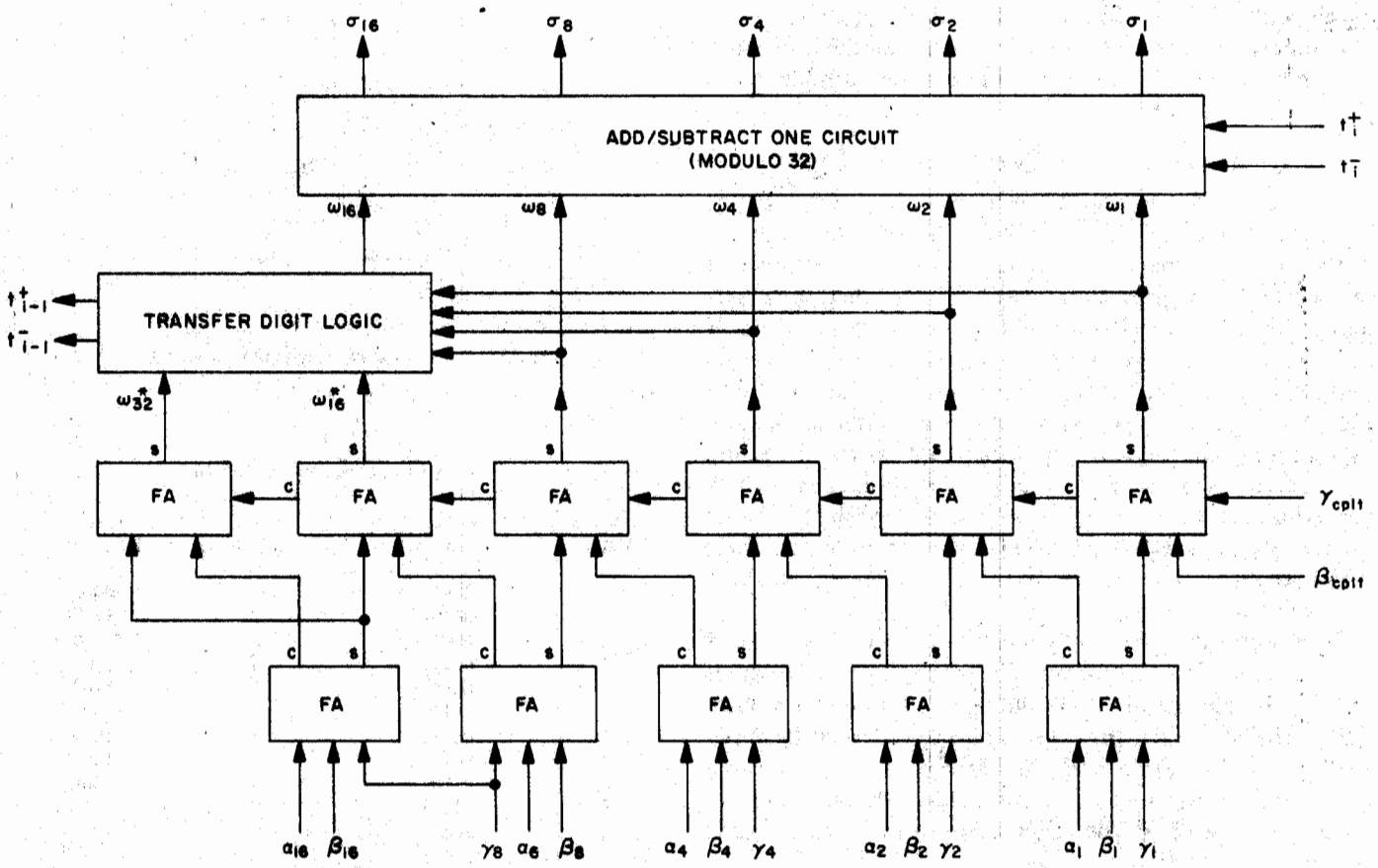

Fig. 3. Organization of the radix 16 ASU.

final design, the signal delay of a package will be minimized wherever possible by employing more complex logic functions (for instance, carry-lookahead rather than ripple-carry addition nets).

Figure 3 shows the logic design of a typical radix 16 ASU, implementing the algorithm (5) and (6). To subtract the digits y_i and/or z_i, their bits are individually complemented and the signals $\beta_{cplt} = 1$ and/or $\gamma_{cplt} = 1$ are applied as inputs. The inputs y_i and z_i may be employed together to enter $m_i (0 \leq m_i \leq 15)$, represented by four bits $(\eta_8, \eta_4, \eta_2, \eta_1)$ for addition to a min-r digit x_i, producing a min-r sum digit $s_i(\sigma_{16}, \sigma_8, \sigma_4, \sigma_2, \sigma_1)$. The bits of m_i are reassigned as bits of y_i and z_i as follows:

$$\beta_8 = \eta_8, \qquad \beta_1 = \eta_1; \tag{9}$$

$$\gamma_4 = \eta_4, \qquad \gamma_2 = \eta_2; \tag{10}$$

$$\beta_{16} = \gamma_8 = \beta_4 = \beta_2 = \gamma_1 = 0. \tag{11}$$

If the conventional number $(m_1, \cdots, m_n)$ is negative, both digits y_i and z_i are subtracted from x_i. The conversion to the s-d form is performed concurrently with an addition; special conversion algorithms or circuitry are not needed. The array of eleven FA's in Fig. 3 forms the sum of three binary numbers which is represented by six bits $(\omega^*_{32}, \omega^*_{16}, \omega_8, \omega_4, \omega_2, \omega_1)$. Negative sums appear as 64's complements. In the final design carry-lookahead will be employed to reduce the delay of addition. The net "transfer digit logic" implements the following functions, defined by (5):

$$t^+_{i-1} = \bar{\omega}^*_{32} [\omega^*_{16} + \omega_8(\omega_4 + \omega_2 + \omega_1)] \tag{12}$$

$$t^-_{i-1} = \omega^*_{32}(\bar{\omega}^*_{16} + \bar{\omega}_8) \tag{13}$$

$$\omega_{16} = \bar{\omega}^*_{32}\omega_8(\omega_4 + \omega_2 + \omega_1) + \omega^*_{32}(\bar{\omega}^*_{16} + \omega^*_{16}\omega_8). \tag{14}$$

The functions t^+_{i-1} and t^-_{i-1} are the outgoing transfer digits, while ω_{16} incorporates the correction required by a nonzero transfer digit output. The five bits $(\omega_{16}, \omega_8, \omega_4, \omega_2, \omega_1)$ represent the correct interim sum w_i of (5). The "add/subtract one" circuit increments w_i by one for $t^+_i = 1$, and decrements w_i by one for $t^-_i = 1$, producing the final sum digit $s_i(\sigma_{16}, \sigma_8, \sigma_4, \sigma_2, \sigma_1)$ encoded according to Table I.

The ASU package of Fig. 3 requires 25 external data connections: 18 inputs and 7 outputs. The package is entirely combinational, and the final design of the logic net will aim for least signal delay, rather than component count minimization in the ASU.

IV. The Two-Digit Product Unit

In view of the decreasing cost of logic arrays, two-operand multiplication using combinational circuits is a practical objective for high-speed computers. Minimally redundant s-d forms are convenient for two-digit [redacted] units (PU) because of the relatively limited range of prod-

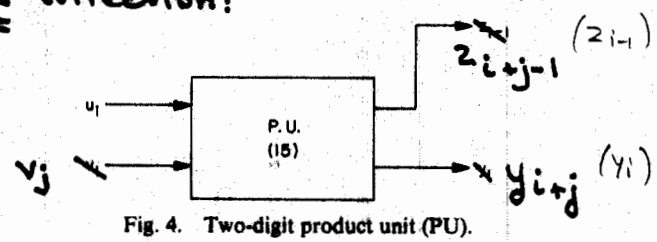

Fig. 4. Two-digit product unit (PU).

ucts. We consider the product unit shown in Fig. 4. The algorithm is

$$u_i v_i = r z_{i-1} + y_i. \tag{15}$$

For even radices $r > 2$ and minimally redundant digits u_i and v_i, the greatest product of two digits will be

$$|u_i v_i|_{\max} = \left(\frac{r}{2}+1\right)^2 = r\left(\frac{r}{4}+1\right)+1$$
$$= r\left(\frac{r+2}{4}\right) + \underbrace{\left(\frac{r}{2}+1\right)}_{y_i}. \tag{16}$$

Given the requirement that y_i should be minimally redundant, the maximum values of $|z_{i-1}|$ are determined from (16) above (with $k = 1, 2, \cdots$) to be

$$|z_{i-1}| \le \frac{r}{4}+1, \quad \text{for } r = 4k; \tag{17}$$

$$|z_{i-1}| \le \frac{r+2}{4}, \quad \text{for } r = 4k+2. \tag{18}$$

For odd radices $r \ge 3$ the same conditions yield

$$|z_{i-1}| \le \frac{r+3}{4}, \quad \text{for } r = 4k+1; \tag{19}$$

$$|z_{i-1}| \le \frac{r+1}{4}, \quad \text{for } r = 4k-1. \tag{20}$$

The allowed ranges (7) and (8) for the input z_i of the ASU (Fig. 2) are compared with (17)–(20) above. We observe that the ASU input z_i has the range to accept the z_{i-1} output of the product unit for all even radices $r \ge 10$ and all odd radices $r \ge 7$. In these cases a row of ASU's can form the sum $x + u_i v$, with x as the previous partial product, v the multiplicand, and u_i the present multiplier digit. The product $u_i v$ is formed by a row of PU's which are connected to the y_i and z_i inputs of the ASU's.

The organization of a radix 16 PU is shown in Fig. 5. The PU is compatible with the ASU of the preceding section; the same notation and digit encodings are employed as in Fig. 3 and Table I. The internal logic net is entirely combinational. There are 19 external data connections: 10 inputs and 9 outputs.

Three distinct functions are performed by the logic net of the PU. The "recoding logic" net recodes the five-bit representation of the multiplier digit u_i (in true or 32's complement form) into the four-bit minimal form in which bit values 0, 1, and -1 are permitted [8]. The minimal form has no adjacent nonzero bits; therefore, at most, two nonzero bits can occur and a two-operand adder is sufficient in the "summation logic" net. The logic equations for recoding are implicitly given by the recoding rules of Table II.

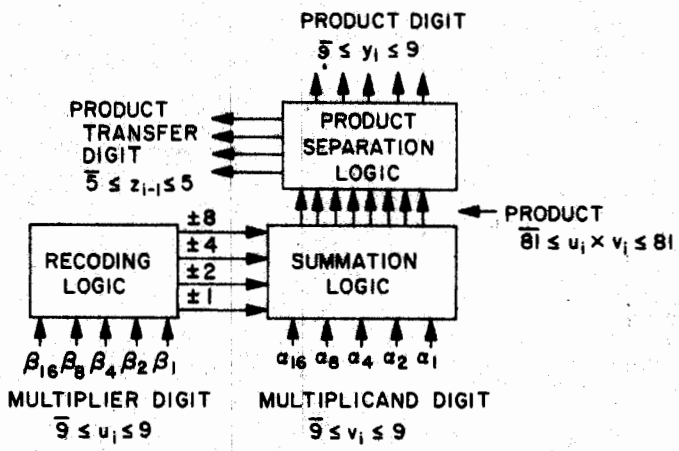

Fig. 5. Organization of the radix 16 PU.

TABLE II
MULTIPLIER DIGIT RECODING

Digit Value u_i	Recoding	Digit Value u_i	Recoding
0	0 + 0	$\bar{4}$	0 + $\bar{4}$
1	0 + 1	$\bar{3}$	0 + $\bar{3}$
2	0 + 2	$\bar{2}$	0 + $\bar{2}$
3	4 + $\bar{1}$	$\bar{1}$	$\bar{4}$ + 1
4	4 + 0	0	$\bar{4}$ + 0
5	4 + 1	$\bar{1}$	$\bar{4}$ + $\bar{1}$
6	8 + $\bar{2}$	$\bar{2}$	$\bar{8}$ + 2
7	8 + $\bar{1}$	$\bar{3}$	$\bar{8}$ + 1
8	8 + 0	$\bar{4}$	$\bar{8}$ + 0
9	8 + 1	$\bar{5}$	$\bar{8}$ + $\bar{1}$

The details of the summation logic net are presented separately in Fig. 6. The outputs of the recoding logic net select the inputs to the array of full-adders and half-adders from the five bits of the multiplicand digit $v_i(\alpha_{16}, \alpha_8, \alpha_4, \alpha_2, \alpha_1)$. If a negative multiple $-k$ is required, then the multiple k of the 32's complement of v_i is entered into the summation logic. The 32's complement is shifted left for the multiples 2, 4, and 8; the bits of the specified multiple are individually complemented and a unit input is applied on the appropriate line marked with the subscript (cplt). For the multiple -2, two unit inputs in the rightmost position are applied, simplifying the design. The multiple pairs $(\pm 1, \pm 2)$ as well as $(\pm 4, \pm 8)$ can share the same input lines, since members of these pairs cannot be required simultaneously. The product $u_i v_i$ is represented by eight bits (σ_{128} to σ_1 in Fig. 6); negative products appear as 256's complements.

The remaining subnet of the PU separates the product $u_i v_i$ into digits y_i and z_{i-1} according to (15). This function is performed by the "product separation logic" net of Fig. 5.

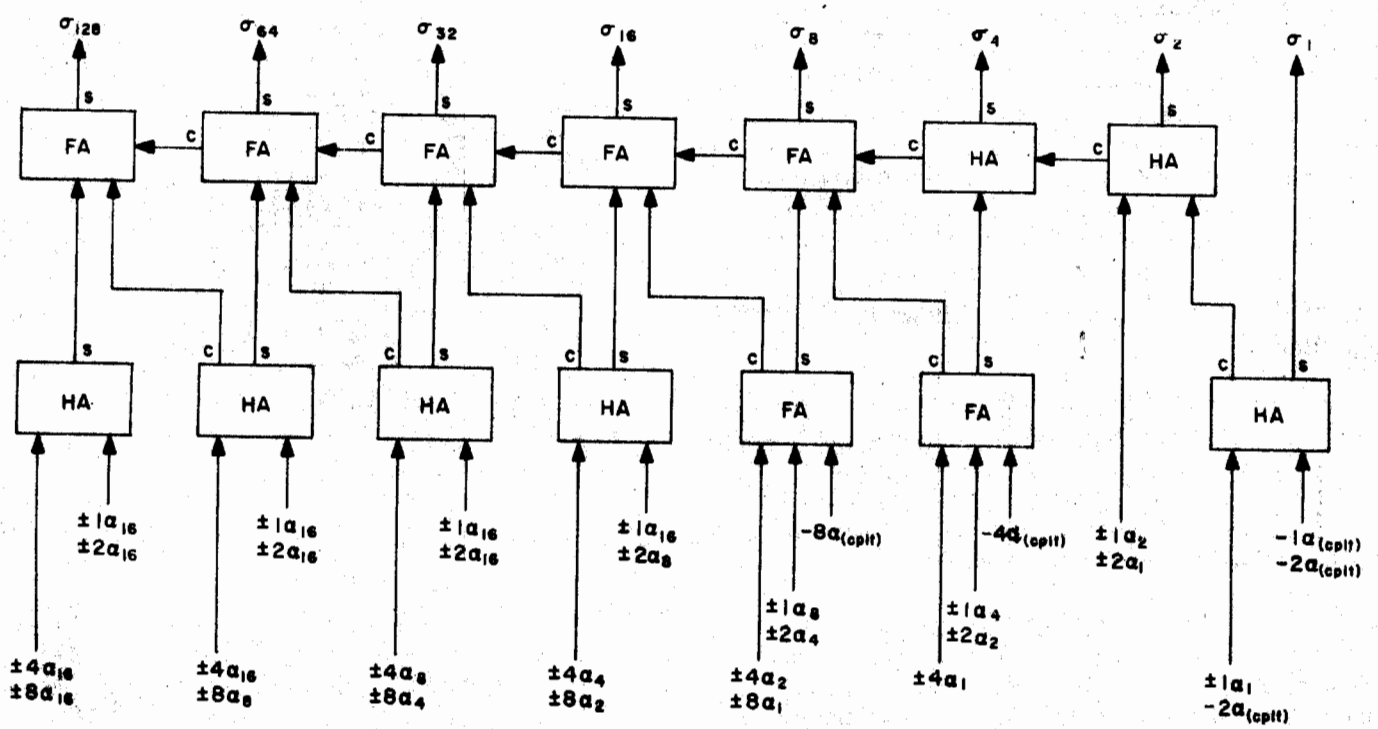

Fig. 6. Summation logic of the radix 16 PU.

A tentative separation assigns the four leftmost bits of $u_i v_i$ to z_{i-1}, and the remaining four bits to y_i. However, y_i must be in the range $\bar{9} \leq y_i \leq 9$; if the value represented by the bits $(\sigma_8, \sigma_4, \sigma_2, \sigma_1)$ exceeds 9, a correction is required which adds -16 to y_i and adds 1 to the four-bit number $(\sigma_{128}, \sigma_{64}, \sigma_{32}, \sigma_{16})$ to give the correct z_{i-1}. The need for correction is determined by the sign bit of y_i, computed as

$$\sigma^*_{16} = \bar{\sigma}_8(\sigma_4 + \sigma_2). \tag{21}$$

The digit y_i is represented by the five bits $(\sigma^*_{16}, \sigma_8, \sigma_4, \sigma_2, \sigma_1)$. The entire product separation net consists of the function (21) and a four-bit "add one" circuit which generates z_{i-1} by adding a unit to the tentative z_{i-1} when $\sigma^*_{16} = 1$.

The internal complexity of the PU is relatively low; carry-lookahead in the adder chain of the summation logic box will be included in order to reduce the maximum delay of the PU. Product units for higher radices $r = 2^k$ $(k > 4)$ are expected to fit within a unit package. For example, only a four-operand summation logic is needed for the radix $r = 256$ when recoding of the multiplier digit is used; there are 18 inputs and 17 outputs. The recoding logic becomes more complex, while the separation logic remains simple for higher radices.

V. The Multidigit Sum Unit

The *multidigit sum unit* (MSU) which adds m digits of s-d numbers is an even more general microsystem. The MSU can perform the functions of the PU, the ASU, and the SU with relatively small augmentation of its internal logic structure and with the addition of command input lines identifying the operation to be performed. A general diagram of the MSU is shown in Fig. 7. The main practical limitation of the MSU is the relatively large number of inputs into the package. Overcoming this constraint will make the augmented MSU a truly general arithmetic microsystem, replacing the more specialized units discussed earlier.

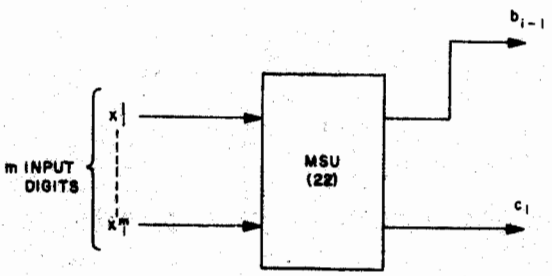

Fig. 7. The multidigit sum unit (MSU).

The most general variant of the MSU accepts m operand digits x_i^j $(j = 1, 2, \cdots, m)$ and produces two output digits (c_i, b_{i-1}) of the same redundancy:

$$\sum_{j=1}^{m} x_i^j = rb_{i-1} + c_i. \tag{22}$$

Since all digits in (22) have the same range $\bar{a}$ to a, as defined in (1), we have

$$|rb_{i-1} + c_i| \leq (r + 1)a \tag{23}$$

as the magnitude limit for the sum in (22). The number m of input digits in the MSU is therefore limited to

$$m \leq r + 1. \tag{24}$$

The all-combinational logic net of the radix 16 MSU contains a "carry-save" cascade of binary full- and half-adders for the summation of m input digits or their 32's complements. It is a direct expansion of the three-input arrangement of the ASU shown in Fig. 3. The separation of the sum into the digits b_{i-1} and c_i follows the procedure described for the PU in the preceding section. The representation of any possible sum of all 17 allowed input digits requires 9 binary digits; negative sums are represented by 512's complement. Otherwise, (21) and its associated conditional correction of +1 to the five leftmost bits of the sum apply directly.

It is evident that the radix r MSU with provision to add or subtract at least a [as defined in (1)] input digits is readily converted to the product unit for radix r. When the "product" command is present, the multiplier digit appearing on the designated input lines determines the number of inputs to the summation net which receive the multiplicand digit or its complement. To perform the functions of the SU (Fig. 1) or the ASU (Fig. 2), the required internal reorganization is imposed by appropriate control commands, and certain inputs are designated for operand and transfer digits.

In the case of the minimally redundant radix 16 system, a convertible MSU must be able to accept and complement at least 9 input digits; the maximum allowable number of input digits is 17. A 10-digit MSU has 50 data input lines, up to 9 control input lines to specify complementation of individual input digits, and 3 control input lines to specify the other functions (ASU, PU, or SU). The output requires 10 lines for two complete digits; only one digit will be produced when the SU or ASU function is being performed. About 50 binary full-adders in a five-level carry-save cascade and an eight-bit carry-propagating adder with lookahead are contained inside, as well as the logic nets for input complementation, output separation, and conversion to other functions. The internal complexity and the count of input/output connections appear to present a challenge to continue the development of integrated circuit arrays.

Other variants of the MSU of Fig. 7 which are of some interest may have the restricted ranges of (7) or (8) for the output b_{i-1} or the restricted ranges of (17)–(20) for one half the total number of input digits. The former is intended to feed an ASU, the latter to receive inputs from several PU's. A combination of both may also be of interest. Algorithm (22) is performed by all of these units, and the allowable number of inputs is established by the method used for (24).

VI. Reconversion to Conventional Forms

The algorithms for reconversion of s-d numbers to conventional forms are strongly affected by the choice of encodings for s-d digit values. Any s-d number may be considered to be represented by two conventional numbers—one positive, containing all positive digits; the other negative, containing all negative digits of the s-d form. A conventional addition will combine both into one conventional number.

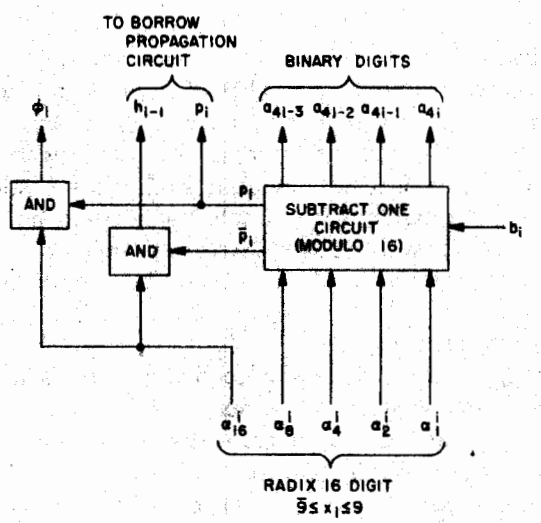

Fig. 8. One stage of the binary reconverter.

A faster method which employs borrow-propagation has been devised for the radix 16 encodings of Table I. The reconversion network requires a borrow-propagation network and one reconverter stage, shown in Fig. 8, for every digit of the s-d form. The ith reconverter stage produces the *borrow-propagate* output p_i which detects a zero digit value of x_i

$$p_i = \overline{\alpha_8^i + \alpha_4^i + \alpha_2^i + \alpha_1^i} \tag{25}$$

and the *borrow* output h_{i-1} if the digit is negative, but not the space-zero marker (encoded as 10000):

$$h_{i-1} = \alpha_{16}^i \bar{p}_i. \tag{26}$$

If the digit is the space-zero, a special output line ϕ_i is energized to identify the radix 16 position i as being nonsignificant:

$$\phi_i = \alpha_{16}^i p_i. \tag{27}$$

Zeros will fill the nonsignificant positions in the binary result of the reconversion algorithm.

An incoming borrow b_i into the position i of the n-digit radix 16 number $x\,(x_1, x_2, \cdots, x_n)$ is computed as

$$b_i = h_i + h_{i+1}p_{i+1} + h_{i+2}p_{i+1}p_{i+2} + \cdots + h_{n-1}p_{i+1}p_{i+2}\cdots p_{n-1}. \tag{28}$$

The incoming borrow b_i is subtracted modulo 16 from the four right-end bits of x_i and four binary digits of the $(4n+1)$ bits long binary number $a\,(a_0, a_1, \cdots, a_{4n})$ are generated. The borrow b_0 sets the "sign bit" a_0

$$a_0 = b_0 \tag{29}$$

and negative binary results are produced in the "two's complement" form with $a_0 = 1$. An end-around borrow arrangement

$$b_n = b_0 \tag{30}$$

will yield a "one's complement" form for negative results. This form is readily converted by digitwise complementation to a sign-and-magnitude binary number. Direct application of (28) to generate b_i requires logic gates with $n-i$ inputs. Explicit borrow computation is far less costly than in binary arithmetic; for example, for a 60-bit binary result, we have only $n=15$, and direct implementation of (28) is still practical. Evidently, less costly but slower borrow-lookahead nets may be employed if lower speed of conversion can be accepted.

The logic complexity of one reconverter stage is relatively low, and several stages with their associated borrow nets would be needed for an integrated circuit reconverter unit (RU) package. A considerable number of reconversion points are needed to make the RU desirable as a microsystem; otherwise, assembly from smaller building blocks would be preferable.

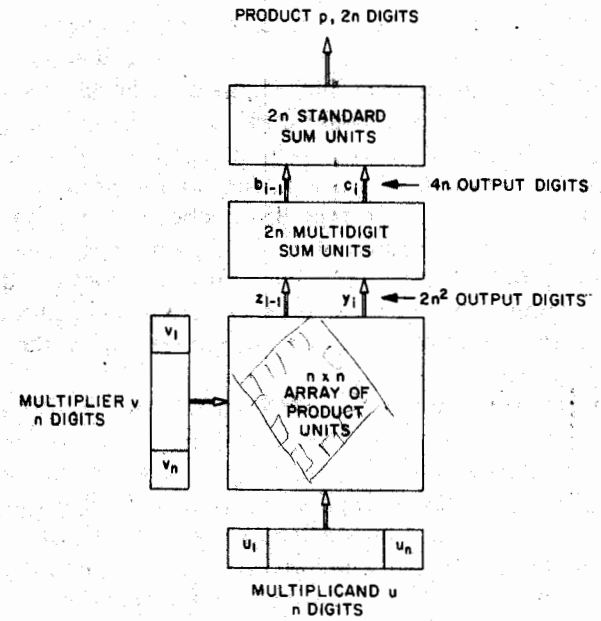

Fig. 9 Combinational multiplication array.

VII. Design of Function Generators

The microsystem units may be used individually in byte-organized processors, or they may be arranged into arrays which generate arithmetic functions by means of combinational logic nets. Two examples will illustrate the application of radix 16 units to combinational generation of products and quotients.

Figure 9 shows the block diagram of a product generator for n-digit operands. All digit products are generated at once in the n by n array of product units, each generating two output digits (y_i and z_{i-1}) for a total of $2n^2$ digits. Digits associated with the same position of the product are summed in a row of $2n$ multidigit sum units. The number $2n$ here is an approximation; for higher values of n, more than one MSU may be required for some center positions, especially for positions n and $n+1$ which must accept $2n-1$ input digits each and for some neighboring positions. The number of input digits decreases by two per position toward the ends; only one input (and no MSU) is required for the end positions $i=1, 2n$. The product in s-d form is obtained by summing the MSU outputs in $2n$ standard sum units. The delay in the array is the same for all input signals and consists of the delays in one PU, one MSU, and one standard SU or ASU. When the input digit capacity of the MSU's is exceeded in the case of higher values of n, some signals will need to pass through one more MSU before entering the last row of sum units.

The division algorithm devised by Robertson has been successfully applied in signed-digit arithmetic [3]. Figure 10 illustrates the iterative method of Robertson's division for operands in fractional range. The quotient digits q_i are in the range $-h \le q_i \le h$, with $h=r/2$ giving the least possible amount of redundancy for the quotient. As long as the quotient is in redundant form, the values of the quotient digits q_i can be decided after an approximate comparison of the partial remainder (or dividend) d and several multiples of the normalized divisor x. The comparison is performed on k most significant digits of the operands; a total of h separate tests is needed to determine q_i.

In the radix 16 system of Fig. 10, the allowed quotient digits are $-8 \le q_i \le 8$; in this case $h=8$ comparisons are needed. They are:

$$|d'| < \left|\left(\frac{x}{2}\right)' + (cx)'\right|, \quad \text{with } c = 0, 1, 2, \cdots, 7. \tag{31}$$

The least value of c satisfying (31) gives the magnitude of q_i, the sign is plus if signs of d and x agree, minus otherwise. The choice $|q_i| = 8$ is made if all tests of (31) are not satisfied. With $h=8$, the required accuracy of comparison is $k=3$ digits (positions $i=1, 2, 3$), plus the temporary overflow digit (position $i=0$) of d and of the divisor multiples [2].

The multiple $(x/2)'$ is obtained from an array of three PU's which generate $8x'$; their two sets of output digits (z_{i-1} and y_i) are shifted one position to the right. Six arrays of four PU's each generate the multiples cx' for $c=2, 3, 4, 5, 6, 7$; again each multiple is represented by two sets of digits z_{i-1} and y_i. The digits y_i and z_{i-1} of $(x/2)'$ are added to x' or to the digits y_i of the multiples cx' in seven arrays of three ASU's each. The seven sets of four output digits of the ASU's are added to the digits z_{i-1} of cx', and the comparison (31) is performed by adding or subtracting the truncated dividend or partial remainder d' (d_0, d_1, d_2, d_3) in the comparator, which consists of eight arrays of four ASU's each. Note that one array receives $(x/2)'$ directly and compares it to d'.

The "quotient digit selection" (QDS) logic net compares the signs of x' and d'; if the signs agree, d' is subtracted in the comparators and q_i is positive; otherwise, d' is added and q_i is negative. The QDS net also determines the magnitude of q_i from the outputs of the eight comparator arrays

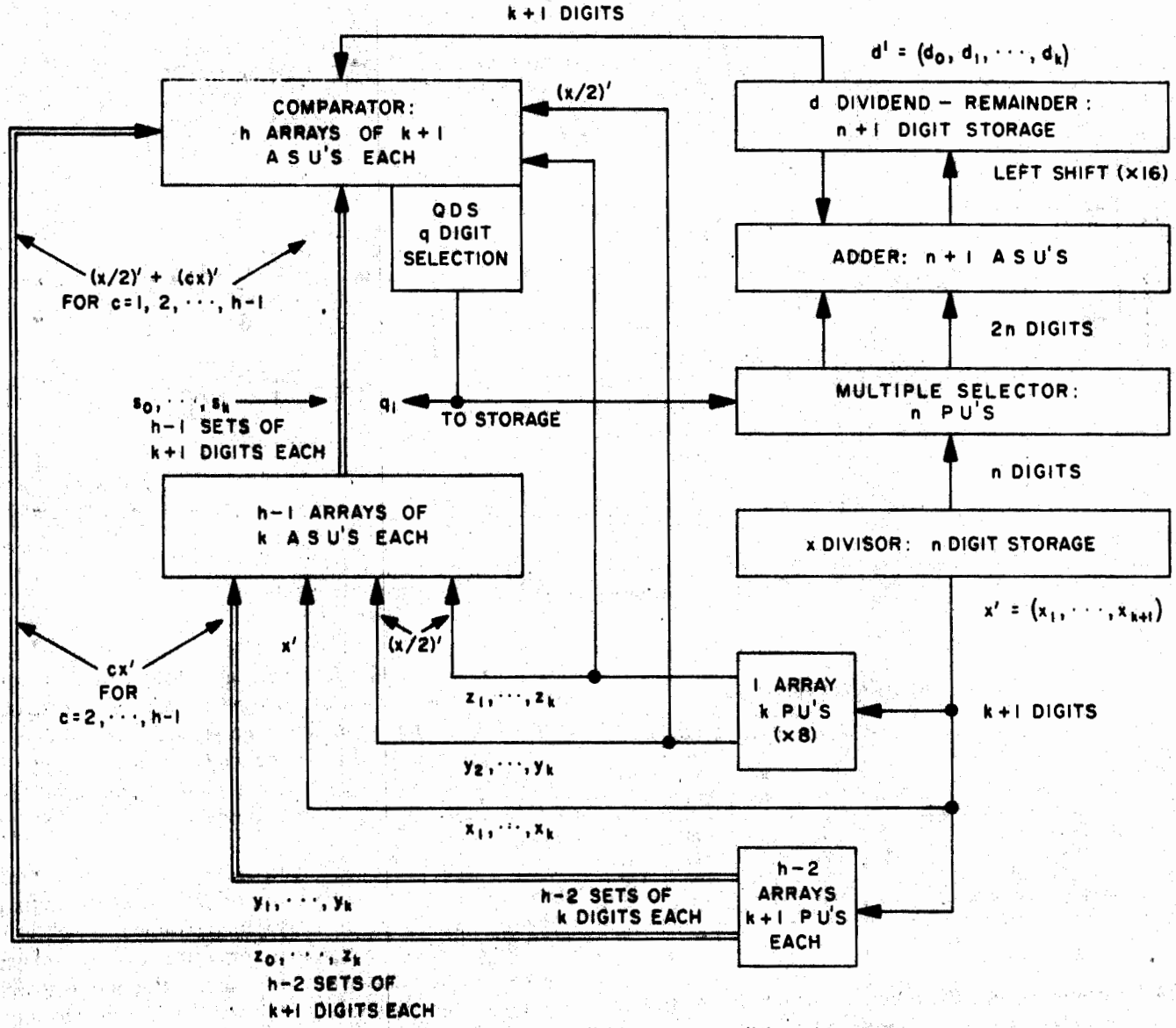

Fig. 10. Radix 16 sequential division array; one digit per step.

according to the rule given immediately following (31). The value of q_i is stored and also sent to the multiple selector which determines the multiple of divisor x to be subtracted from d. The result is shifted one digit left and serves as the next partial remainder. Division is completed when the required number of quotient digits has been generated.

A cascade of copies of the array in Fig. 10 can be employed to generate the entire quotient and remainder by combinational logic. Such an array is shown in Fig. 11. The divisor is initially in normalized form; a value of $q_0 = 8$ indicates a possible two-digit overflow of the quotient and requires a right shift of the dividend. Alternatively, one more digit (q_{-1}) at the left end of the quotient may be generated by adding one more copy of Fig. 10 below Adder #0 and entering the right-shifted dividend ($d \times 16^{-1}$) into its adder.

The preceding examples illustrate the application of arithmetic microsystems in the generation of arithmetic functions. The number of units used, although large, is not unreasonable if a unit can be contained within a single integrated circuit package. An estimate of the number of units needed for the radix 16 arrays of Figs. 9, 10, and 11 is given in Table III.

The length n of the operands is given in radix 16 digits; the equivalent length of binary operands (in bits) is also indicated. Two types of MSU are considered for the multiplication: type I has $m = 10$ input digits, and type II has the maximum $m = 17$ input digits. For all lengths except $n = 5$, cascades of two MSU's are needed to sum the PU outputs in central positions of the product. The maximum delay in these cases is four units. Further reduction of the number of input digits to the MSU will require deeper cascading of

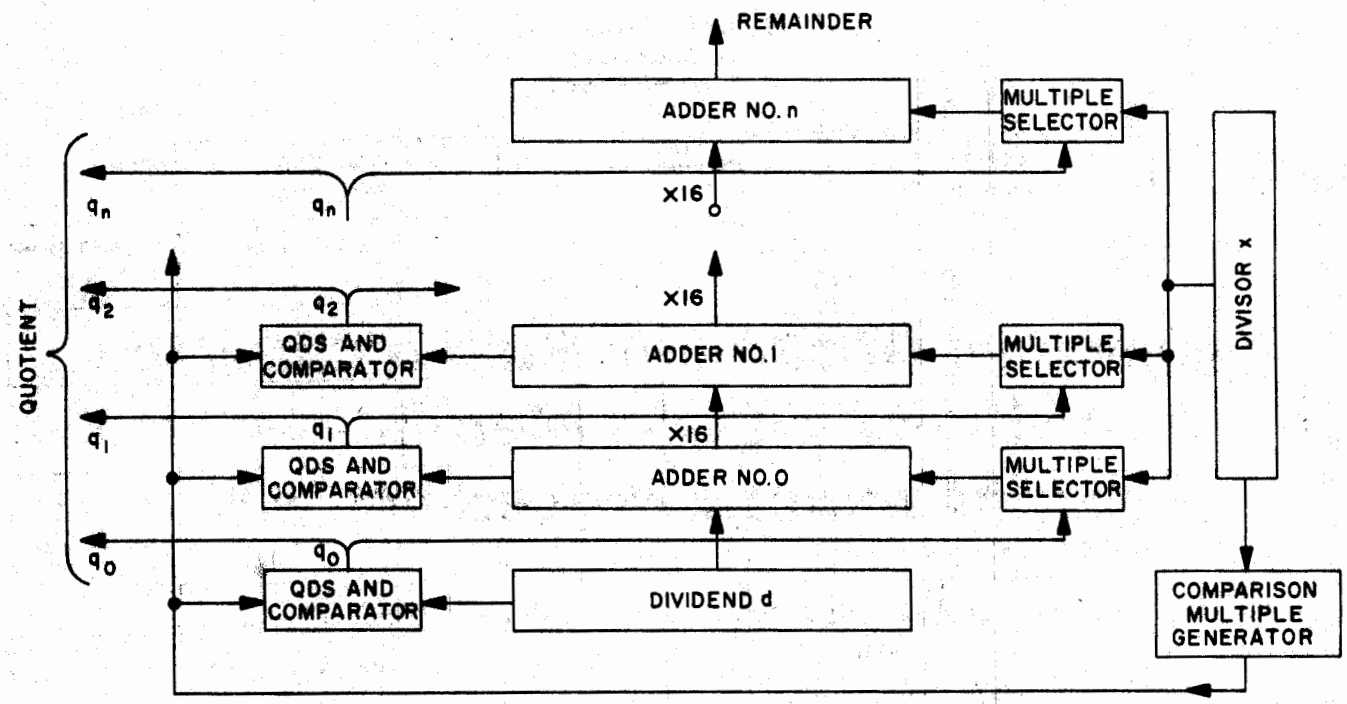

Fig. 11. Radix 16 combinational division array.

TABLE III
UNIT COUNT FOR FUNCTION GENERATORS

Figure No.	Function \ Operand Length	$n=5$ 20 bits	$n=10$ 40 bits	$n=15$ 60 bits	$n=20$ 80 bits	$n=25$ 100 bits
9	Multiply I ($m=10$)	45	160	335	570	865
9	Multiply II ($m=17$)	45	140	310	530	805
10	Divide (sequential)	90	100	110	120	130
11	Divide (combinational)	265	580	995	1510	2125

MSU's in summing the PU outputs. The unit count and the maximum delay both will be increased.

Neither division array requires an MSU. The QDS nets are considered to be part of the control logic and are not included in the unit count. The formation of divisor multiples requires two-unit delays; from then on every quotient digit requires three-unit delays plus the delay of a QDS net. The cost and delay of operand storage elements and sequence generation logic are to be added in the unit and delay count for Fig. 10. The combinational nets of Figs. 9 and 11 do not require intermediate storage or sequencing. The delay of the division net in Fig. 11 is proportional to the length of the quotient, plus the initial two-unit delay.

Function generators for other arithmetic functions can be arranged in a similar manner, either as combinational nets or as sequential nets with associated storage and sequence generating logic. It is interesting to note that a combinational reconversion net may be attached at the output of a function generator. In this case conventional numbers are accepted as operands and a conventional number is delivered as the result, although signed-digit arithmetic is employed inside the function generator. In comparison with conventional arithmetic, the principal advantages of signed-digit arithmetic are the ready separability of the arithmetic net into microsystems of considerable complexity and the independence of addition delay from the length of operands.

Research is being continued on several aspects of arithmetic microsystems. Internal optimization of the units in order to attain minimum delay is being studied, as well as the feasibility of speed-independent operation of function generators. Programmed synthesis of specified function generators under the constraints of speed (maximum allowable delay) and cost (count of microsystem units) is also being investigated.

REFERENCES

[1] G. Estrin, B. Bussell, R. Turn, and J. Bibb, "Parallel processing in a restructurable computer system," *IRE Trans. on Electronic Computers*, vol. EC-12, pp. 747–755, December 1963.
[2] A. Avižienis, "Signed-digit number representations for fast parallel arithmetic," *IRE Trans. on Electronic Computers*, vol. EC-10, pp. 389–400, September 1961.
[3] ——, "On a flexible implementation of digital computer arithmetic," in *Information Processing 1962*, C. M. Popplewell, Ed. Amsterdam, Netherlands: North-Holland Publishing Co., 1963, pp. 664–670.
[4] ——, "Binary-compatible signed-digit arithmetic," *AFIPS Conf. Proc.*, vol. 26, pp. 663–672, 1964.
[5] N. Metropolis and R. L. Ashenhurst, "Significant digit computer arithmetic," *IRE Trans. on Electronic Computers*, vol. EC-7, pp. 265–267, December 1958.
[6] A. Avižienis and D. M. Kimble, "A general building block for digital arithmetic," *Proc. of National Symp. on the Impact of Batch Fabrication on Future Computers*, pp. 173–180, 1965.
[7] M. J. Flynn, "Complex integrated circuit arrays: the promise and the problems," *Electronics*, vol. 39, pp. 111–116, July 11, 1966.
[8] G. W. Reitwiesner, "Binary arithmetic," in *Advances in Computers*, vol. 1, F. L. Alt, Ed. New York: Academic Press, 1960, pp. 244–260.