DESIGN OF AN ARITHMETIC ELEMENT FOR SERIAL PROCESSING IN AN ITERATIVE STRUCTURE

Lakshmi N. Goyal
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Abstract

This paper describes the arithmetic and logic design of the digit processing logic of an arithmetic element. The arithmetic element is used in an iterative structure and arithmetic processing takes place serially on a digit by digit basis with the most significant digit first. Starting from the arithmetic specification of the digit processing logic, the arithmetic design (namely, the choice of number system, number representation and the digit algorithm) is developed. Algebraic and logic designs of the logic necessary to execute the digit algorithm and its implication for LSI implementation are discussed.

1. Introduction

The advent of large-scale integration (LSI) technology for the manufacturing of logic circuits has posed a new challenge to the designers of digital systems. Efficient use of LSI capabilities require that the digital systems be modular, that these modules consist of as few different types as possible and that the interconnection structure between the modules be uniform. On the module level, the module itself should have a high gate to pin ratio and a regular and repetitive internal logical structure. Further, the module should be as locally autonomous as possible, communicating with only a few of its neighboring modules for information. This avoids the high pin count and the necessity of large on-chip drive capability and consequent high power dissipation.

In this paper, we consider some aspects of the design of a basic arithmetic processor and its implications for LSI technology implementation. A basic arithmetic processor executes four arithmetic instructions of addition/subtraction, multiplication and division of two operands. The basic arithmetic micro-instruction of an arithmetic unit which performs multiplication and division by alternately doing one addition/subtraction and shifting can be characterized by the transfer function

$$A' = A \pm (m * \Phi) \qquad (1)$$

where A', A and Φ are consistently represented numbers and m is the multiplier or quotient digit.

Atrubin [1], Goyal [2], and Pisterzi [3] describe algorithms to process a sequence of such micro-instructions in a modular, iteratively structured network of identical finite state machines. This structure very eminently meets the constraints of LSI technology. In such a structure, the results are obtained on a digit by digit basis. In Atrubin's method, the processing begins with the least significant digits of the operands first and correspondingly the least significant digits of the result are available first. On the other hand, in Pisterzi's model, the processing takes place beginning with the most significant digits of the operands first and most significant digits of the result are obtained first. The most-significant-digit-first (MSDF) approach has the advantages of early overflow detection, normalization concurrent with processing and early termination of processing as soon as enough significant digits in the result have been obtained. Moreover, the quotient generation and operand normalization processes inherently require the examination of most significant digits first. Finally, the MSDF algorithm in an iterative structure has the potential, under suitable environment, of meshing (pipelining) of successive instructions.

This paper presents the results of a study into the arithmetic and logic design of the digit processing logic of the finite state machine. The radix chosen for use in the digit processing logic is a design parameter. The main considerations of the design were its suitability for LSI implementation.

To put things in perspective, a brief description of the organization and operation of the iterative structure is given in section 2. Section 3 presents the major content of this paper. It describes the arithmetic design of the digit processing logic of the finite state machine in terms of number system, number representation and the digit algorithm. The algebraic design of the logic necessary to implement the digit algorithm is explained next. We further give a brief description of the logic design of the digit processing logic and its implications on the pin requirements for various values of radix. Finally alternatives are suggested for reducing the number of pins.

2. Arithmetic Unit Organization

The arithmetic unit organization considered in this paper was first proposed by Pisterzi [3]. Structurally, it consists of a linear cascade of identical arithmetic elements called Digit Processing Modules (DPM) and a Global Control Unit (GCU). The GCU receives an arithmetic instruction (e.g., ADD, MULTIPLY, etc.) from an external source and converts them into a sequence of elementary microinstructions to be executed by the linear cascade of DPMs. However, the GCU communicates only with the most significant DPM and the microinstructions flow serially (in a pipelined manner) from the most significant DPM (closest to the GCU) to the least significant DPM, instead of being broadcast to all modules simultaneously. Figure 1 shows the schematic organization of such an arithmetic unit. Each DPM retains the values of one digit of each of the active operands in its registers and collectively, the DPMs contain the mantissas of all active operands and do the processing on them. The DPMs have the capability of performing microinstructions which will (when performed by all DPMs) form sums, perform shifts, and do inter-register transfers, etc. Because the quotient generation and operand normalization processes require the examination of the most significant digits, the operands are placed in the DPMs so that the digits of each of the operands are available to the microinstructions in order of decreasing significance. Thus the most significant digits of the operands are placed in the DPM which communicates with the GCU.

Each DPM performs the same sequence of micro-instructions. A given microinstruction is not executed by all DPMs in synchronization, but rather must be executed by them in sequence (i.e., first by DPM_1, then DPM_2, ...). As soon as all the DPMs, which contain information required by DPM_i to perform microinstruction

j+1 (referred to as μ_{j+1}), have executed μ_j and have sent the required information to DPM_i, μ_{j+1} may be performed by DPM_i. The microinstructions must have regular data requirements so that as each additional DPM executes μ_j, one more DPM may execute μ_{j+1}. Clearly, the DPM registers do not contain entire operands as long as any of the DPMs are actively executing microinstructions. Each DPM contains the digits from the results of the last microinstruction executed. A more formal and detailed explanation of the operation of a DPM is given in the appendix.

All the microinstructions executed by the DPMs are so designed that the processing begins with the most significant digit of the operands and proceed to those with decreasing significance. For example, consider the multiplication algorithm which is right directed. It is implemented as a repeated sequence of shift-left multiplier, multiply and add, and shift-left-accumulator microinstructions in that order. During the multiplication process, the cascade network of DPMs behave as what Dadda and Ferrari [4] call a "column-wise" operating multiplier: the product digits of a column of product-matrix are generated within different DPMs and summed along the cascade network with other digits of the same column and with carries (transfers) of the preceding columns.

For a detailed description of the implementation of various arithmetic algorithms and a complete description of the different elementary microinstructions executed by the DPMs, see Goyal [5].

3. Design of the Digit Processing Module

From the description of a typical DPM, it is evident that the operations must be completely independent of the significance of the digits retained by a DPM. All DPMs may then be identical. A DPM is a finite state, complex logic module with control logic and combinational digit processing logic.

To achieve a compromise between the serial processing and the desired arithmetic speed, an arithmetic step is carried out in higher radix, $r = 2^k$, such that k-bits of the result are obtained at any step. k is chosen as the design parameter. The major consideration in the arithmetic and logic design has been the desirability of its implementation in LSI technology.

3.1 Arithmetic Design

The basic digit level arithmetical function performed by a DPM can be specified as

$$a_i' = a_i \pm m_j \Phi_i \qquad (2)$$

where a_i', a_i and Φ_i are consistently represented digits in the active registers of the DPM and m_j is a multiplier or quotient digit such that

$$|m_j| \leq r-1 , \quad r \text{ being the radix.}$$

3.1.1 Choice of Number System

Let α_j denote the number of DPMs from which a given DPM requires information in order to execute the microinstruction μ_j. Because of the iterative structure, it is evident that for efficient operation of the unit, α_j should satisfy two constraints:

a) The microinstructions should have regular data requirements independent of the significance of the digits retained by the DPM. That is, α_j should be the same for each DPM.

b) The value of α_j should be as small as possible because the execution rate of a given microinstruction is inversely proportional to α_j.

In a conventional weighted number system, carry or borrow into any digital position is a function of all the digits to the right of it. The value of α_j, therefore, is clearly a function of significance of the digit itself. Hence conventional number system is unsuitable. The redundant number system which gives a bounded value of α_j is essential.

3.1.2 Choice of Number Representation and Redundancy

The major factors influencing the choice of redundant number representation and the amount of redundancy in the number system are the following:

a) the ease of conversion from conventional number representation to redundant number representation,

b) its compatibility with the widely employed conventional binary number system,

c) normalization of operands to radix-2 limits, and

d) LSI technology constraints, namely

i) minimization of the number of types of cells (in the arithmetic and logic sense) required for higher radix ($r=2^k$) implementation of the digit processing logic, and

ii) minimization of the number of input and output pins.

In this study, signed digit (SD) representation and maximal redundancy is chosen because it meets most of the requirements. That is, for radix 2^k, the digit set $\{\overline{(2^k-1)}, \ldots, \bar{1}, 0, 1, \ldots, (2^k-1)\}$ is used. The overbar is used to designate negative digit values. Implications of maximal redundancy, "pseudo normal" forms, and the use of "pack-add" algorithm are discussed in Avizienis [6] and Goyal [5].

Two modes of representation for the signed digit are used, depending on the area of application:

a) Redundant Binary Mode - Each digit, radix 2^k, is represented by k redundant binary digits. Each redundant binary digit is chosen from the digit set $\{\bar{1}, 0, 1\}$.

b) Sign-Magnitude (SM_r) Mode - Each digit, radix 2^k, is represented by a single sign bit and a magnitude represented by k bits.

3.1.3 Digit Algorithm

Figure 2 shows the functional representation of the digit processing logic of a DPM. Essentially, it consists of a digit multiplier and an adder. Efficient use of LSI technology in the implementation of radix-2^k digit processing logic dictates that it should be made up of identical logical cells. One approach that achieves this goal is to design the logic as a one

dimensional linear cascade of k-stages of radix-2 arithmetic processing structures.

Since the arithmetical design of the adder is influenced by the arithmetical and logical organization of the multiplier, the design of the digit multiplier is discussed next followed by that of the adder.

3.1.3.1 Design of the Digit Multiplier

The digit multiplier forms the product of the multiplicand and multiplier digit. In order to illustrate the algorithm, let the two digits to be multiplied be denoted by X and Y. In the most general way, they can be represented as

$$X = \sum_{i=0}^{K-1} x_i 2^i \qquad x_i, y_i \in \{\bar{1}, 0, 1\}$$

$$x_i, y_i \in \{\bar{1}, 0, 1\}$$

$$Y = \sum_{j=0}^{K-1} y_j 2^j$$

The product X*Y is achieved by a K × K square array of redundant binary product cells. Each cell forms the product of two redundant binary digits x_i and y_j and its output product digit p_{ij} is in the digit set $\{\bar{1}, 0, 1\}$. (In actual practice, the product-matrix generator forms the product of two radix-2^k signed digits encoded in SM_r mode.)

The product may be viewed in terms of the sums of the p_{ij} terms of the same weight in the product-matrix.

$$X*Y = \sum_{i=0}^{2K-2} \left(\sum_{\ell=0}^{i} x_\ell \, y_{i-\ell} \right) 2^i$$

$$= \sum_{i=0}^{2K-2} \left(\sum_{\ell=0}^{i} p_{\ell, i-\ell} \right) 2^i$$

where $x_i, y_i = 0$ for $i < 0$ and $i > K-1$.

Defining

$$S_i = \sum_{\ell=0}^{i} p_{\ell, i-\ell}$$

$$X*Y = \sum_{i=0}^{2K-2} S_i \, 2^i \qquad (3)$$

where S_i is the sum of the enteries in the i^{th} column of the product-matrix. The number of product-matrix elements in the i^{th} column is given by

$$N_i = \begin{cases} i+1 & 0 \le i \le K-1 \\ -i+(2K-1) & K \le i \le 2K-2 \end{cases}$$

The number of product elements, p_{ij}, is maximum in column of weight 2^{K-1} and is equal to K. The product elements in the other columns decrease uniformly by one on either side of this column as shown in Figure 3. Equation (3) shows that a linear cascade of 2K-3 multi-input redundant binary adders (MIRBA) corresponding to each column of the product matrix are needed to generate the radix-2^K digit product. Note that the number of inputs to each stage of the linear cascade of adders is different and is given by N_i.

The columns of weight 2^i ($K \le i \le 2K-2$) of the product matrix can be considered as forming a carry or Collective Product Transfer (CPT) to the next more significant radix-2^K digital position (see Figure 4). These CPT columns have weights 2^{i-K} with respect to the higher significant digital position. When these CPT columns are added in the appropriate (of the same weight) MIRBA of the higher significant digital position, all the stages of the linear cascade of MIRBAs become identical, each stage having K inputs. This is shown in Figure 4.

Equation (2) shows that it is necessary to add one radix-2^K digit a_i to the output of digit multiplier. So, a MIRBA capable of adding (K+1) redundant binary $\{\bar{1}, 0, 1\}$ inputs is required.

3.1.3.2 Design of Multi-input Redundant Binary Adder (MIRBA)

A MIRBA is a limited carry-borrow propogation adder which accepts several redundant binary inputs (digit set $\{\bar{1}, 0, 1\}$) and produces one redundant binary output (with appropriate adder "Transfers" for more significant adjacent adder stages).

Let us define a new parameter α^b. The redundant binary output of any MIRBA is dependent on the "Transfers" (the composite term for carry/borrow) input to that MIRBA. In redundant number system, the "Transfers" are functions of "primary" inputs (other than Transfer inputs) to only limited number of adjacent less significant MIRBAs. α^b denotes the number of such adjacent MIRBAs whose "primary" inputs determine the output of a given MIRBA.

The radix-2^K digit processing logic in say DPM_i consists of a K-stage linear cascade of (K+1) input MIRBAs. Except for the most significant MIRBA in K-stage cascade, the inputs to the MIRBAs in DPM_i are functions of radix-2^K operand digits in DPM_i and DPM_{i+1} (accumulator digits a_i, multiplier digit m_j and multiplicand digits Φ_i, Φ_{i+1}). (See Figure 9.) Thus α_j is related to α^b by equation (4).

$$\alpha_j = \left\lceil \frac{\alpha^b - 1}{K} \right\rceil + 1 \qquad (4)$$

Three different approaches for the arithmetic (algebraic) design of a MIRBA were considered.

3.1.3.2.1 Rohatsch's Technique [7]

This technique is an explicit transformation technique which converts a given input digit set into the required output digit set by a series of Simple Transformations. Using this approach, we find that for a (K+1) input MIRBA, a four level (3 stages of Simple Transformations) redundant binary structure is necessary for K > 2. It can be shown [5] that a four level (in the arithmetic sense) redundant binary structure can be designed to accept as many as 51 redundant binary inputs and produce one redundant binary output by higher order Simple Transformations.

However, the logic design of the bottom (nearest to input digit set) two levels is highly complicated for K ≥ 5 that is radix 32, if they are to be implemented in two or three logic levels. In practice, the technique is to break down the bottom level structure into equivalent simpler structures frequently at the cost of increasing the number of levels. Moreover,

such a structure is not suitable for LSI implementation because no elementary logic cell, other than the primitive NANDS, NORS can be repetitively used in a uniformly interconnected pattern for the implementation of (K+1) input MIRBA.

Table 1 shows the values of α^b and α_j for various values of K for a (K+1) input MIRBA.

3.1.3.2.2 $\log_2$-sum Tree Technique

A conceptually simple approach is to realize the (K+1) input MIRBA by a $\log_2$-sum tree structure of two input redundant binary adders (RBA-2). The logic design of such RBA-2 was studied in detail by Borovec [8] and we shall interchangeably use the term Borovec Unit (BU) for RBA-2 in the following discussion. Figure 5 shows one such BU consisting of a cascade combination of a symmetric subtractor [9] (SS) and a symmetric adder [9] (SA) and a D-element. The D-element decomposes a redundant binary input into positive (0,1) and negative (0,$\bar{1}$) binary outputs.

For a (K+1) input MIRBA, the tree structure has t levels such that

$$t = \lceil \log_2 (K+1) \rceil$$

and the number of BUs required is K. Figure 6 shows the $\log_2$-sum tree structure for a six input MIRBA.

In this configuration,

$$\alpha^b = 2t = 2\lceil \log_2 (K+1) \rceil \text{ and}$$

$$\alpha_j = \left\lceil \frac{2\lceil \log_2 (K+1) \rceil - 1}{K} \right\rceil + 1 .$$

The value of α_j for various values of K is tabulated in Table 1. From the table we find that for K=2 and K=4, that is, radices 4 and 16, the value of α_j=3, and α_j=2 for all other values of K. Since minimum value of α_j is desirable, a different arrangement of BUs as described in third approach given next can be used to achieve α_j=2.

3.1.3.2.3 Tree-structure using RBA-3s and RBA-2s

In this configuration, 3-input redundant binary adders (RBA-3) and RBA-2s are connected in a tree structure.

An RBA-3 consists of two BUs, a D-element and a C-element arranged as shown in Figure 7. The C-element composes two binary inputs {0,1; 0,$\bar{1}$] into one redundant binary {$\bar{1}$,0,1} output. The lower BU in combination with C-element and D-element acts as a redundant binary (3,2) counter. The upper BU forms the sum of the sum-output of the lower BU and the "Transfer" output of the lower BU of adjacent less significant RBA-3.

For a design of (K+1) input MIRBA, RBA-3s are used whenever it can be fully utilized, that is, three inputs are available for addition; and RBA-2s are used when only 2-inputs are to be added at any level of the tree structure. (An exception occurs for K=3 where $\log_2$-sum tree technique is necessary.) Figure 8 shows a 6 input MIRBA using this configuration.

The number of BUs required in this technique is K for a (K+1)-input MIRBA. The number of BU levels is also $2\lceil \log_2 (K+1) \rceil$. Table 1 shows the values of α^b and α_j for various of K. It shows that α_j=2 for all values of K except K=3.

The tree structure configurations described in 3.1.3.2.2 and 3.1.3.2.3 have the following advantages compared to Rohatsch's technique:

a) It is more general and has the same configuration for any value of K.

b) It makes use of only one kind of cell, that is, Borovec Unit for the implementation of MIRBA.

c) The various BUs are uniformly and regularly interconnected.

Because of b) and c) above, this implementation meets the LSI constraints of structure regularity and minimum part number type.

3.2 Logic Design

The major consideration in the logic design is the choice of an encoding for the radix-2^K Signed Digit. The encoding of the digit has implications on the logic complexity of the digit processing logic and also on the pin complexity (total number of pins) of the DPM.

As suggested in section 3.1.2, two modes of representation for a radix-2^K Signed Digit are used:

a) Sign-Magnitude (SM_r) mode

b) Redundant Binary mode

In SM_r mode, a radix-2^K Signed Digit is encoded as a (K+1)-tuple binary logic vector. This requires (K+1) binary storage elements. Conversion of an input operand in conventional number system (binary and sign-magnitude) to the Signed Digit form necessary for the DPMs is trivial. It is carried out by just attaching the sign of the conventional number to each group of K bits. For these reasons, SM_r mode encoding for the radix-2^K Signed Digit is used for internal storage of the operand and result digits and for transfer of these digits between DPMs.

Redundant binary mode is used for the arithmetic processing specially for the multi-input redundant binary adders in the DPM. Each redundant binary digit requires 2 bits for representation and thus a radix-2^K Signed Digit is represented by a logic vector of length 2K. Borovec [8] studied in detail the logic design of two input redundant binary adders (RBA-2) for all the nine distinct ways, under permutation and negation, of assigning three values ($\bar{1}$,0,1) to four states of two binary variables.

Out of these nine distinct formats, the sign-magnitude format (referred as SM_b) is found to be the most suitable. In this format, if a redundant binary digit ℓ_i^* {$\bar{1}$,0,1} is represented by two binary variables λ_i and ℓ_i {0,1}, the SM_b encoding is given by $\ell_i^* = (-1)^{\lambda_i} \ell_i$. Although Borovec's study shows that this format does not give minimal logic complexity for a BU, this still seems to be the best compromise for the following reasons:

i) Conversion from SM_r mode to SM_b format is

trivial and involves appending the single sign bit to each of the K-magnitude bits.

ii) Negation of a radix-2^K digit for subtraction requires only complementation of each of the sign bits.

iii) Design of a one-dimensional iterative encoder to convert the redundant binary output of the MIRBAs to SM_r mode is the simplest for the SM_b format encoding.

iv) In the product-matrix generator, the logic for the generation of each product digit of the matrix consists of a single AND-gate and an Exclusive-OR gate. (In practice, however, both the multiplier and multiplicand radix-2^K digits are in SM_r format and the product-matrix logic consists of K^2 AND-gates and only one Exclusive-OR gate for the sign.

Let $\ell_i^*(\lambda_i, \ell_i)$ and $m_i^*(\mu_i, m_i)$ denote the redundant binary inputs and $d_i^*(\delta_i, d_i)$ denote the output of a BU. Let t_i^+, t_i^- and t_{i-1}^+, t_{i-1}^- be the input and output "Transfers." The internal logic of a BU is given by the set of Boolean equations (5).

$$\left.\begin{aligned} &\ell_i^* = (-1)^{\lambda_i}\ell_i,\ m_i^* = (-1)^{\mu_i}m_i \text{ and } d_i^* = (-1)^{\delta_i}d_i \\ &d_i = \ell_i \oplus m_i \oplus t_i^+ \oplus t_i^- \\ &\delta_i = \overline{t_i^+} \\ &t_{i-1}^- = \lambda_i\,\ell_i \vee \mu_i\,m_i\,\overline{\ell}_i \\ &t_{i-1}^+ = (\ell_i \oplus m_i)\,\overline{t_i^-} \vee \ell_i\,\overline{\mu}_i(m_i \vee \overline{t_i^-}) \end{aligned}\right\} \quad (5)$$

If, however, the inputs and outputs are encoded in Transfer Format ($\ell_i^* = \lambda_i - \ell_i$, etc.), then a BU can be implemented by a cascade of two ordinary binary full adders--each acting as a (3,2) counter [5]. This leads to a simpler logic cell for the LSI implementation of MIRBA at the cost of larger logic delay in the generation of MIRBA output.

3.2.1 Pin Complexity

Figure 9 shows the schematic diagram of the radix-2^K digit processing logic in a DPM. It consists of one product-matrix generator, K of (K+1)-input redundant binary adders (MIRBA) and an encoder which converts the redundant binary result output of the MIRBAs to SM_r format for storage or inter DPM communication. $AT_i(AT_{i-1})$ represent the adder "Transfers" into (out of) the least significant (most significant) MIRBA of DPM_i from (into) the most significant (least significant) MIRBA of adjacent $DPM_{i+1}(DPM_{i-1})$. $CPT_i(CPT_{i-1})$ is the "collective product transfer" from (into) the product matrix generator of the adjacent $DPM_{i+1}(DPM_{i-1})$ into (from) DPM_i.

The number of pins required for the digit processing logic is the sum of the number of pins required for AT_i, AT_{i-1}, CPT_i, CPT_{i-1}, for one multiplier digit and for the SM_r encoded output of the MIRBAs. The tree structure configuration of (K+1)-input MIRBA uses K BUs and each BU requires two pins for the Transfers $\{0,1;\ 0,\overline{1}\}$. Thus 2K pins are needed for AT_i or AT_{i-1}. $CPT_i(CPT_{i-1})$ require one pin for the sign information (assuming that both the multiplier and multiplicand digits are SM_r encoded) and $(K-1) + (K-2) + \ldots + 1 = \frac{K(K-1)}{2}$ pins for magnitude information. Therefore, the total number of pins required by the digit processing logic, excluding control is given by

$$\begin{aligned} \text{Total \# of pins required} = P &= P_{AT_i} + P_{AT_{i-1}} + P_{CPT_i} + P_{CPT_{i-1}} + P_{ier} + P_{EAO} \\ &= 2K + 2K + \frac{K(K-1)}{2} + 1 + \frac{K(K-1)}{2} + 1 \\ &\quad + (K+1) + (K+1) \\ &= K^2 + 5K + 4 \end{aligned}$$

where

P_{AT_i} = # of pins required for AT_i, etc.

P_{ier} = # of pins required for multiplier digit

P_{EAO} = # of pins required for SM_r encoded output of MIRBAs.

Table 2 shows the values of P for various values of K. The value of P for $K \geq 5$ are impractical.

However, the number of pins required for $AT_i(AT_{i-1})$ can be reduced if the K "positive transfers" (0,1) and the K "negative transfers" $(0,\overline{1})$ of the (K+1)-input MIRBA are encoded. They can be each encoded as $\lceil \log_2(K+1) \rceil$ outputs. The "Transfer encoder" can be implemented using $\leq$ K conventional binary full adders in $\lceil \log_3 K \rceil + \lceil \log_2 K \rceil - 2$ levels [10]. The corresponding "Transfer decoder" in the adjacent DPM is simply a fan-out network. In this network, the encoded Transfer of weight W is fanned-out to W "Transfer" inputs (of the same sign as the encoded Transfer) of the least significant MIRBA of the adjacent DPM. Similarly, the number of pins required for $CPT_i(CPT_{i-1})$ can also be reduced from $(\frac{K(K-1)}{2} + 1)$ to only (K+1) if CPT_i is generated in DPM_i instead of in DPM_{i+1}. Note that CPT_i is a function of the multiplicand digit in DPM_{i+1} and the multiplier digit, the latter being the same in both DPM_i and DPM_{i+1}. Thus DPM_i needs to know only the multiplicand digit in DPM_{i+1} to generate CPT_i. This requires only (K+1) pins. Let us call this method of transmitting CPT_i to DPM_i as "Indirect Generation" (IG) for lack of a better term. Let P_{TEIG} denote the total number of pins required when the Transfer Encoder (TE) and Indirect Generation (IG) of CPT_i methods are used. Then

$$\begin{aligned} P_{TEIG} &= 2\lceil \log_2(K+1) \rceil + 2\lceil \log_2(K+1) \rceil + (K+1) + (K+1) + (K+1) + (K+1) \\ &= 4\lceil \log_2(K+1) \rceil + 4(K+1) \end{aligned}$$

The value of P_{TEIG} is tabulated in Table 2 for various values of K. The table shows quite a reduction in the pin complexity although it is achieved at the expense of introducing an additional type of cell (full adders) in the LSI implementation of the DPM.

The number of pins can be still further reduced if the multiplier digit has a redundancy ratio ρ such that $\rho \leq 2/3$. Such a multiplier digit can be recoded in Nonadjacent Format (that is, the recoded digit has only nonadjacent nonzero redundant binary digits). This reduces the maximum number of nonzero redundant binary inputs to a MIRBA from (K+1) to $\lceil \frac{K}{2} \rceil + 1$. The number of Borovec Units required in the realization of MIRBA are correspondingly reduced to $\lceil \frac{K}{2} \rceil + 1$. Hence the number of pins when multiplier digit redundancy is reduced to $\rho \leq 2/3$ is given by

$$P_{\substack{TEIG \\ \rho \leq 2/3}} = 4\lceil \log_2 (\lceil \tfrac{K}{2} \rceil + 1) \rceil + 4(K+1) .$$

This is tabulated in Table 2 and shows that only a minor reduction occurs.

Table 2 shows the pins required for only the combinational part of the digit processing logic of the arithmetic element or DPM. In order to perform the arithmetic algorithm serially on a digit by digit basis, some amount of local sequential control is necessary, although such control is very simple. The local control complexity is independent of the radix. Choice of the radix for implementation of a DPM on a single LSI chip would depend on maximum allowable pads (pins) on the chip. A trade off must also be made between the cost of combinational part (arithmetic cost) and cost of sequential control both in terms of their logic complexity and pin complexity when choosing a radix or value of K. Details about the local control required for a DPM can be found in Goyal [5].

Although the design presented in the present paper was specifically developed for serial processing in an iterative structure, the same arithmetic element can also be used in a purely combinational, parallel two-dimensional array arrangement for arithmetic processing. It can also be used in a bus structured and associative processor configuration with proper sequential local control.

4. Summary

The design of an arithmetic element for use in a linear, iteratively structured arithmetic unit, in which arithmetic processing takes place serially, on a digit by digit basis and with most-significant digit first, is presented. The main considerations in the design were the desirability of the arithmetic element's implementation (as a single module) in LSI technology. The design of the digit processing logic is described at two levels. Starting from the arithmetic specification of the digit processing logic, the arithmetic design (namely, the choice of number system, number representation and the digit algorithm) is developed, with radix as the design parameter. Then a brief description of the logic design of the digit processing logic which implements the digit algorithm is given. The implications of the logic design on the LSI implementation, namely on the logical complexity, part number type and pin requirements are discussed.

5. Acknowledgments

The author is grateful to his advisor, Professor James E. Robertson, for the invaluable guidance, useful suggestions and encouragement throughout this work. The author would also like to express his appreciation to his colleague Ms. Mary J. Irwin for many suggestions that improved the readability of this paper, to Mrs. June Wingler for typing the paper and to Mr. Mark Goebel for the figures.

References

[1] A J. Atrubin, "A One-Dimensional Real-Time Iterative Multiplier," IEEE Trans., Elect. Comput., Vol. EC-14, pp. 394-399, June, 1965.

[2] L. N. Goyal, "A Note on Atrubin's Real-Time Iterative Multiplier," accepted for publication as correspondence in IEEE Trans. on Computers.

[3] M. J. Pisterzi, "A Limited Connection Arithmetic Unit," Ph.D. Thesis, Department of Electrical Engineering, University of Illinois, Urbana, Illinois, June, 1970. Also report no. 398, Department of Computer Science, University of Illinois, Urbana.

[4] L. Dadda and D. Ferrari, "Digital Multipliers: A Unified Approach," Alta Frequenza, Vol. XXXVII, No. 11, pp. 1079-1086, November, 1968.

[5] L. N. Goyal, "A Study in the Design of an Arithmetic Element for Serial Processing in an Iterative Structure," Forthcoming Ph.D. Thesis, Department of Electrical Engineering, University of Illinois, Urbana.

[6] A. Avizienis, "Binary-Compatible Signed-Digit Arithmetic," AFIPS Conference Proceedings, Vol. 26, 1964.

[7] F. A. Rohatsch, "A Study of Transformations Applicable to the Development of Limited Carry-Borrow Propagation Adders," Ph.D. Thesis, Department of Electrical Engineering, University of Illinois, Urbana, Illinois, June, 1967.

[8] R. T. Borovec, "The Logical Design of a Class of Limited Carry-Borrow Propagation Adders," M.S. Thesis, Department of Electrical Engineering, University of Illinois, Urbana, Illinois, August, 1968.

[9] J. E. Robertson, Private communication, October, 1974.

[10] C. C. Foster and F. D. Stockton, "Counting Responders in an Associative Memory," IEEE Trans. Computers, Vol. C-20, pp. 1580-1583, December, 1971.

Appendix

Operation of a DPM

The following description has been adopted from Pisterzi [3] with slight modifications.

The processing performed by the DPMs can be described by the following:

$$ {}_j\vec{X}_i = \Psi_j({}_{j-1}\vec{X}_i, {}_jF_{i-1}, {}_jG_{i+1}) \qquad (A.1) $$

$$ {}_jF_i = \Phi_j({}_{j-1}\vec{X}_i, {}_jF_{i-1}) \quad \text{and} \qquad (A.2) $$

$$ {}_jG_k = \Gamma_j({}_jF_{k-1}, {}_{j-1}\vec{X}_k, {}_{j-1}\vec{X}_{k+1}, \cdots, {}_{j-1}\vec{X}_{k+\alpha_j}) \qquad (A.3) $$

where

${}_j\vec{X}_i$ is the operand information contained in the i^{th} DPM immediately following the execution of μ_j. It consists of the i^{th} digit of

each of the active operands.

ψ_j is the function employed to obtain the new operand set and is dependent on the microinstruction to be performed.

${}_jF_i$ is a "modifier" value which DPM_i transmits to DPM_{i+1} with the microinstruction to be performed next.

Φ_j is the function which each DPM performs to determine ${}_jF_k$.

α_j is the number of DPMs which must cooperate with the right neighbor of DPM performing μ_j in order to generate the necessary ${}_jG_{i+1}$.

${}_jG_k$ is the value which DPM_k transmits to the DPM executing μ_j.

Γ_j is the function DPM_k employs to determine ${}_jG_k$.

The operation of a typical DPM, DPM_i say, is as follows. It begins in a state in which it is receptive to information defining the next microinstruction to be performed. DPM_i receives this information and the value of ${}_jF_{i-1}$ from its left neighbor DPM_{i-1}. Then DPM_i determines ${}_jG_{i-1}$. The function Γ_j can be implemented either totally in parallel in DPM_i from inputs of active operand digits in DPM_{i+1} through $DPM_{i+\alpha_j}$ or it can be realized in a time sequential fashion. The former approach takes many more pins. A detailed description of the two approaches can be found in [5]. DPM_i also determines ${}_jF_i$ by performing equation (A.3) and transmits this value and the identity of μ_j to DPM_{i+1}. Sometime later, DPM_i receives a signal from DPM_{i-1} indicating that DPM_{i-1} has executed μ_j. DPM_i then executes μ_j (the necessary ${}_jG_{i+1},\ldots,{}_jG_{i-\alpha_j}$ being ready by now). DPM_i transmits a signal at this time to DPM_{i+1} which indicates that DPM_{i+1} may execute μ_j. When DPM_i receives an acknowledgment from DPM_{i+1}, it goes into a state where it is receptive to information concerning μ_{j+1}. The sequence above then repeats.

Table 1

Values of α^b and α_j for Various (K+1) Input MIRBA Configurations

r	K	Rohatsch's Technique		4q-sum Tree		RBA-3, RBA-5 Tree Structure	
		α^b	α_j	α^b	α_j	α^b	α_j
4	2	3	2	4	3	3	2
8	3	4	2	4	2	5	3
16	4	4	2	5	3	5	2
32	5	4	2	6	2	5	2
64	6	5	2	6	2	6	2
128	7	5	2	6	2	6	2
256	8	5	2	8	2	6	2

Table 2

Pin Complexity V_s Radix

radix	K	P	$P_{TEIG},\rho^*=1$	$P_{TEIG},\rho^* \leq 2/3$
4	2	18	20	16
8	3	28	24	24
16	4	40	32	28
32	5	54	36	32
64	6	70	40	36
128	7	88	44	44
256	8	108	52	48

*ρ is the redundancy ratio for multiplier digit

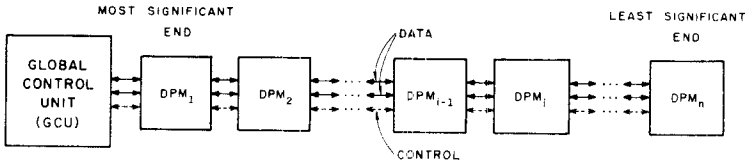


Figure 1. The Organization of Linear Iteratively Structured Arithmetic Unit

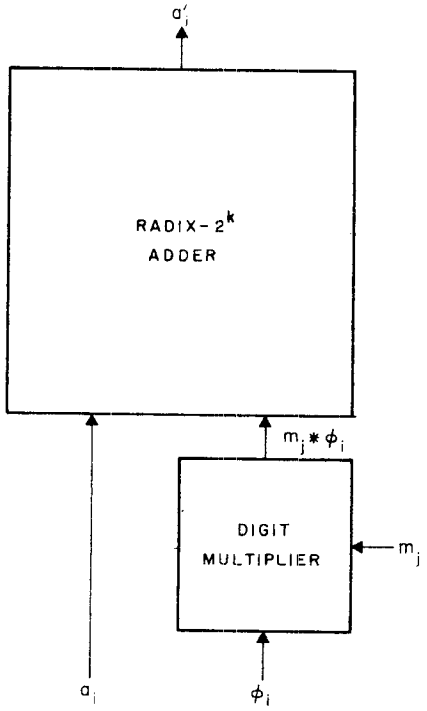


Figure 2. Functional Representation of Digit Processing Logic

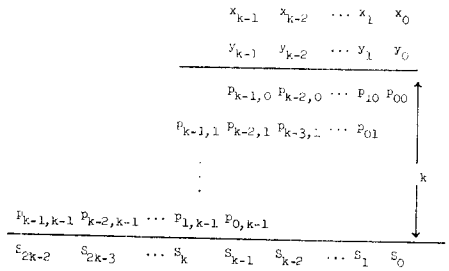


Figure 3. Product Matrix

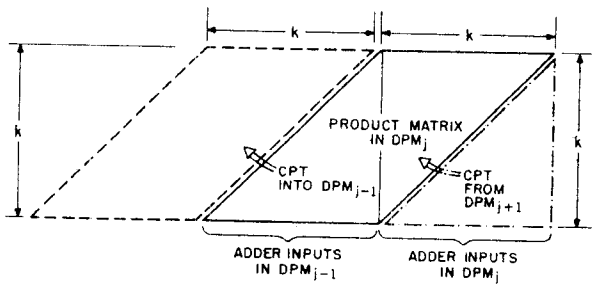
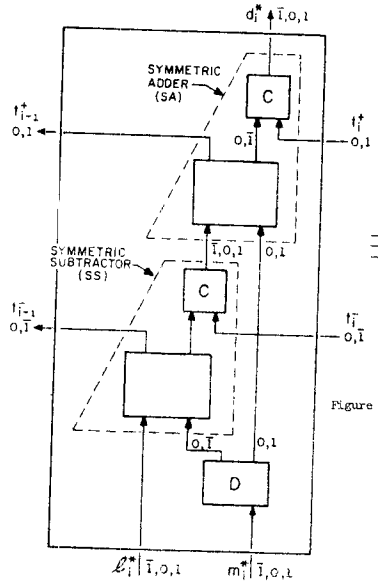


Figure 4. Illustration of Overlapping Product Matrices and "Collective Product Transfer"



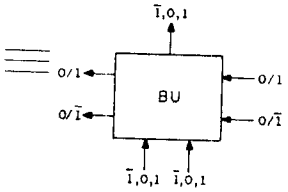


Figure 5. Borovec Unit/Two Input Redundant Binary Adder (RBA-2)

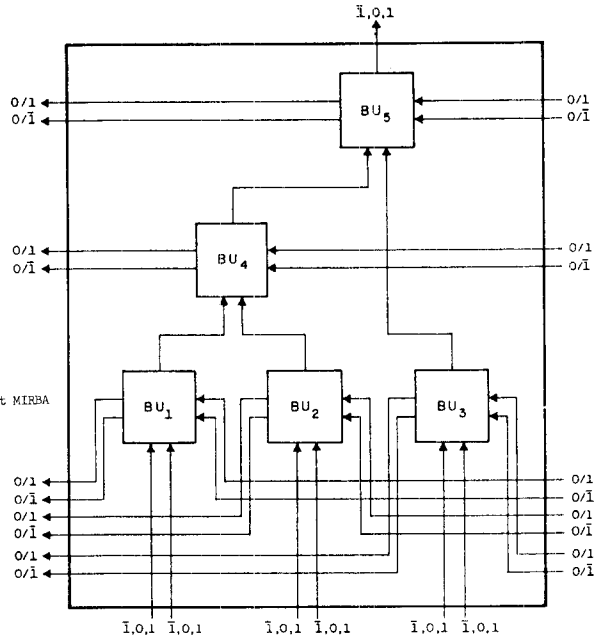


Figure 6.
Φ_3-sum Tree Structure of Six Input MIRBA

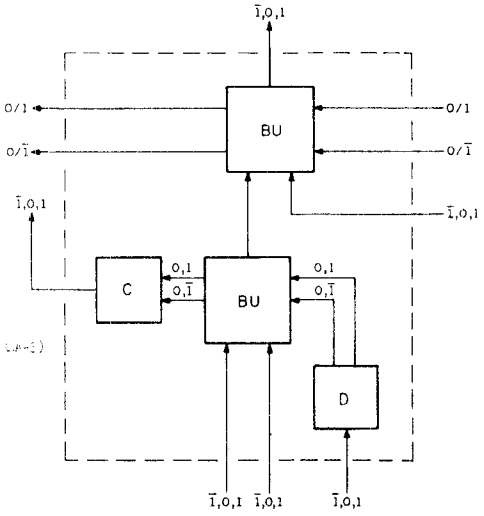


Figure 7. Three Input Redundant Binary Adder (RBA-3)

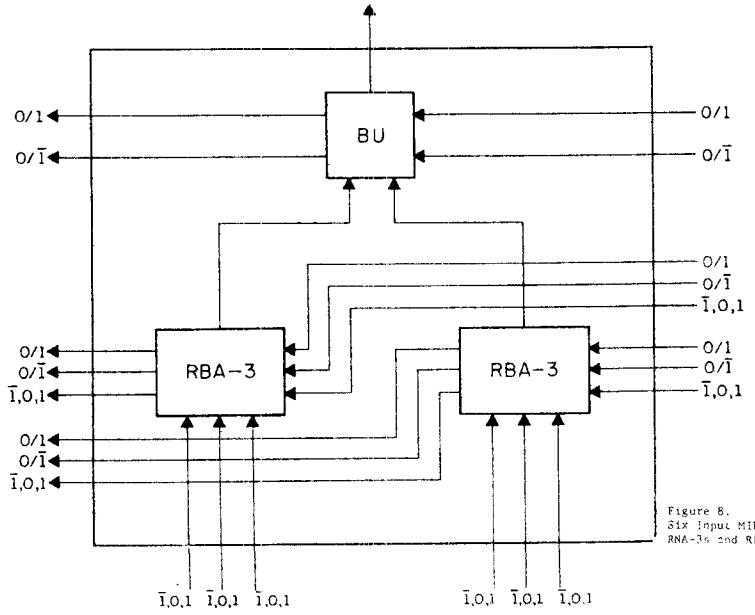


Figure 8.
Six Input MIRBA using RNA-3s and RBA-2s.

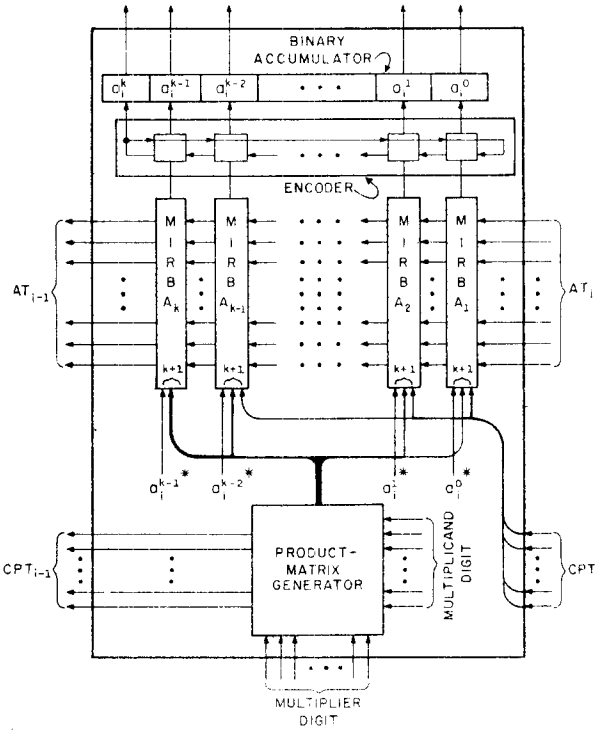


Figure 9. Schematic Representation of Digit Processing Logic of a Arithmetic Element